MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-15146-TDI/1278

UMA ARQUITETURA PARA GERAÇÃO DE INTERFACES ADAPTATIVAS PARA DISPOSITIVOS MÓVEIS

Giani Carla Ito

Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Drs. Nilson Sant'Anna, Maurício Gonçalves Vieira Ferreira e Solon Venâncio de Carvalho, aprovada em 3 de agosto de 2007.

INPE
São José dos Campos
2008

Publicado por: **esta página é responsabilidade do SID**

Instituto Nacional de Pesquisas Espaciais (INPE)
Gabinete do Diretor – (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 – CEP 12.245-970
São José dos Campos – SP – Brasil
Tel.: (012) 3945-6911
Fax: (012) 3945-6919
E-mail: pubtc@sid.inpe.br

Solicita-se intercâmbio
We ask for exchange

MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-15146-TDI/1278

UMA ARQUITETURA PARA GERAÇÃO DE INTERFACES ADAPTATIVAS PARA DISPOSITIVOS MÓVEIS

Giani Carla Ito

Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Drs. Nilson Sant'Anna, Maurício Gonçalves Vieira Ferreira e Solon Venâncio de Carvalho, aprovada em 3 de agosto de 2007.

INPE
São José dos Campos
2008

681.3.06

Ito, G. C.
Uma arquitetura para geração de interfaces adaptativas para dispositivos móveis / Giani Carla Ito. - São José dos Campos: INPE, 2007.
216 p. ; (INPE-15146-TDI/1278)

1. Adaptação. 2. Computação móvel. 3. Interface adaptativa. 4. Dispositivos móveis. 5. Web.
I. Título.

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de Doutor(a) em

Computação Aplicada

Dr. José Demisio Simões da Silva

Presidente / INPE / SJCampos - SP

Dr. Nilson Sant'Anna

Orientador(a) / INPE / SJCampos - SP

Dr. Mauricio Gonçalves Vieira Ferreira

Orientador(a) / INPE / SJCampos - SP

Dr. Solon Venâncio de Carvalho

Orientador(a) / INPE / SJCampos - SP

Dr. Nandamudi Lankalapalli Vijaykumar

Membro da Banca / INPE / SJCampos - SP

Dr. Ricardo Rodrigues Ciferri

Convidado(a) / UFSCAR / São Carlos - SP

Dr. Henrique Jesus Quintino de Oliveira

Convidado(a) / UMC / Mogi das Cruzes - SP

Aluno (a): Giani Carla Ito

São José dos Campos, 03 de Agosto de 2007

Não importa a distância: infinita, complicada, sombria
Não importa o tempo: infindável, incontável, interminável
Não importam as dúvidas: indivisíveis, abstratas, conceituais
Não importa o cansaço: físico, mental, emocional
Não importa a incompreensão: áspera, dolorosa, desprezível
Não importa a ansiedade: matinal, vespertina, noturna, boêmia
Não importa o trabalho: incomensurável, exaustivo, complicado

O que importa é...

Usar a distância para ir além do que se imagina
Usar o tempo para cultivar a experiência
Usar as dúvidas para abrir as janelas da inteligência
Usar o cansaço para valorizar a vitória
Usar a incompreensão para refinar a paciência
Usar a ansiedade para mover os sonhos
Usar o trabalho para comprovar que tudo vale a pena!

Giani Carla Ito

A minha querida avó Angélica

que não pôde ficar até o fim para celebrar comigo esta conquista, mas que está sempre presente por meio de seus ensinamentos.

À Luiza

minha maior professora, com quem aprendi princípios que me servirão por toda a vida.

Ao Norio

pela cumplicidade, compreensão e por todo o bem que me fazes.

AGRADECIMENTOS

Durante estes anos de doutorado, os desafios foram muitos. Superá-los só foi possível graças a pessoas muito especiais que direta ou indiretamente fizeram parte desse processo. A todas elas registro minha imensa gratidão.

Primeiramente agradeço a Deus e a Nossa Senhora por iluminar minha mente e conceder-me saúde, persistência, discernimento e sabedoria para a realização deste trabalho.

Meus eternos agradecimentos a meus pais Oscar e Luiza, pela sólida formação dada até minha juventude a qual me proporcionou a continuidade nos estudos até à chegada a este doutorado. Suas sábias lições de vida inspiraram-me a ser persistente diante dos obstáculos. À minha irmã Vivian, agradeço pelo carinho e apoio incondicional. Vocês sempre foram o meu alicerce.

A meu marido Norio, por suportar minha ausência em tantos momentos importantes de nossa vida. Somente um grande homem poderia desprender tanta paciência, amizade e compreensão durante todos esses anos. Sou-lhe imensamente grata. Todas as palavras seriam pequenas diante da magnitude de suas ações. Você é minha principal referência nos quesitos ética, dignidade, amor e companheirismo.

Ao Professor e orientador Maurício, agradeço profundamente por ter assumido a orientação desta tese, tendo-me brindado com importante colaboração dosando as críticas com comentários de incentivo. Sou-lhe grata por sua disponibilidade irrestrita. Agradeço, sobretudo, pelas muitas vezes que ultrapassando sua função de orientador, foi amigo, auxiliando-me em tarefas burocráticas que a distância não me permitia cumprir. Sua colaboração foi fundamental e indispensável em muitos momentos ao longo deste período.

Aos Professores e orientadores Nilson e Solon, agradeço pela oportunidade de me permitirem ingressar no programa de doutorado da CAP e por todo apoio recebido.

A todos os professores e funcionários do programa de pós-graduação da CAP por contribuírem para o meu aprimoramento científico e pelos serviços prestados.

À minha amiga Daniela, agradeço pela amizade, carinho, companhia e incentivo durante esses anos. Vivemos muitos momentos juntas. Dividimos sentimentos de tensão e cansaço e os multiplicamos por risos e alegria. Sua presença foi muito importante para minha saúde afetiva.

Aos meus amigos Íris, Adriana, Andréia, Warley, Junior, Dawilmar, Fabrício, Rodrigo e Bruno, companheiros de tantos estudos, viagens e de momentos de descontração. A presença e a amizade de vocês foram fundamentais durante esses anos. Jamais esquecerei o eco de nossos risos pelo INPE.

Agradeço ao amigo Luciano Sother pelas excelentes sugestões e por compartilhar comigo sua sábia experiência.

Um agradecimento carinhoso e especial as amigas Heloise e Viviane pela parceria na primeira parte dessa aventura, iniciada em Floripa.

Aos estudantes universitários que colaboraram para a realização deste trabalho, principalmente aos acadêmicos André, Douglas, Marilson, Antonio e Ney, expresso o meu reconhecimento e faço votos que consigam realizar todos os seus sonhos.

Ao Instituto Nacional de Pesquisas Espaciais pela oportunidade de estudos e utilização de suas instalações.

Já houve quem dissesse que uma pessoa é feliz pelos amigos que tem. Nesse sentido, minha fortuna não poderia ser maior: amigos ocasionais, amigos antigos, amigos recentes, amigos anjos, amigos que o tempo revelou... A todos aqueles que, embora não nomeados, presentearam-me com seus inestimáveis apoios em distintos momentos e por suas presenças afetivas. A todos o meu reconhecido e carinhoso muito obrigada!

RESUMO

O desenvolvimento de sistemas para o ambiente móvel é considerado complexo devido ao grande número de aspectos a serem analisados, como constantes mudanças de localização e a grande diversidade de dispositivos. Nesse contexto surge a necessidade de uma maior flexibilidade para a modelagem e a construção de sistemas. Este trabalho propõe uma arquitetura para a geração de interfaces adaptativas para dispositivos móveis (GIA) cujo objetivo é permitir uma implementação voltada a um ambiente multi plataforma, partindo de uma descrição genérica da interface. Além disso, deseja-se que os especialistas como desenvolvedores de software e designers possam implementar interfaces tanto para desktops e computadores de mão, sem a necessidade de programação adicional. Para tal, este trabalho apresenta um ambiente de desenvolvimento com o objetivo de integrar o reconhecimento de múltiplos dispositivos em tempo de execução a uma metodologia de fragmentação de código. Esta metodologia consiste em particionar uma interface Web dentro de um conjunto de subpáginas gerando um menu de conteúdos de forma hierárquica para facilitar a navegação em diversos tipos de dispositivos. Para a validação da arquitetura GIA, apresenta-se um estudo de caso com testes realizados em simuladores de telefones celulares e PDAs.

GAI - AN ARCHITECTURE FOR THE GENERATION OF ADAPTIVE INTERFACES FOR MOBILE DEVICES

ABSTRACT

The development of systems for the mobile environment is considered complex due to the great number of aspects to be analyzed, such as constant changes of localization and the great diversity of devices. In this context the necessity of more flexibility for the modeling and implementation of systems appears. This thesis presents an architecture for the Generation of Adaptive Interfaces (GAI) and its objective is to allow an implementation directed to a multi-platform environment, which starts from a generic description of the interface. Moreover, the desire is that the specialists, such as the software programmers and the interface designers can implement interfaces for desktops and mobile devices, without the necessity of additional programming. Thus, this study aims at to consider a development environment with the purpose to integrate the recognition of multiple mobile devices in time of execution to a methodology of division of a page in blocks of diverse sizes. This methodology consists of partitioning a Web interface inside a set of sub-pages generating a menu of contents in a hierarchic form to facilitate the navigation in several types of devices. For the validation of GIA architecture, this thesis presents a case study with tests applied to cellular phones simulators and PDAs.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

API	-	Application Programming Interface
CC/PP	-	Composite Capability Preference Profile
CDC	-	Connected Device Configuration
CF	-	Compact Flash
CLDC	-	Connected Limited Device Configuration
CLR	-	Common Language Runtime
CLS	-	Common Languagem Specification
CSS	-	Cascading Style Sheet
CTT	-	ConcurTaskTree
DELI	-	Delivery Context Library
DevInf	-	SyncML Device Information
DHTML	-	Dynamic Markup Language
DOM	-	Document Object Model
DYGIMES	-	Dynamically Generating Interfaces for Mobile and Embedded Systems
EG-XML	-	Event Graph - XML
EJBs	-	Enterprise JavaBeans
GIA	-	Geração de Interfaces Adaptativas
HP	-	Hewlett Packward
HTML	-	HyperText Markup Language
HTTP	-	HyperText Transfer Protocol
IHC	-	Interação Humano-Computador
J2EE	-	Java 2 Enterprise Edition
J2ME	-	Java 2 Micro Edition
J2SE	-	Java 2 Standard Edition
JCP	-	Java Community Proccess
JDBC	-	Java Database Connectivity
JSP	-	Java Server Pages
JVM	-	Java Virtual Machine

MAS	-	Mobile Application Servers
MAUI	-	Minimal Attention User Interfaces
MS	-	Memory Stick
MSIL	-	Microsoft Intermediate Language
MUSA	-	Multi User Interfaces Single Application
MVC	-	Model View Controller
NAC	-	Negotiation and Adaptation Core
PAC	-	Presentation - Abstraction - Controller
PDA	-	Personal Digital Assistants
RDF		Resource Description Framework
SD	-	Secure Digital
TCP/IP	-	Transmission Control Protocol -Internet Protocol
TERESA	-	Transformation Environment for inteRactive Systems representAtions
UAPROF	-	User Agent Profile
UIMS	-	User Interface Management Systems
UML	-	Unified Modeling Language
UPnP	-	Universal Plug and Play Standard
UPS	-	Universal Profiling Schema
URI	-	Universal Resource Identifier
URL	-	Uniform Resource Locator
USB	-	Universal Serial Bus
W3C	-	World Wide Web Consortium
WAP	-	Wireless Application Protocol
Wi-Fi	-	Wireless fidelity
WML	-	Wireless Markup Language
XHMTL	-	Extensible HyperText Markup Language
XML	-	Extensible Markup Language
XSL	-	eXtensible Stylesheet Language
XSLT	-	eXtensible Stylesheet Language Transformation

CAPÍTULO 1

INTRODUÇÃO

Com o crescimento da necessidade de mobilidade, surgem novas maneiras de utilização do computador a qualquer hora e em qualquer lugar. A diminuição de tamanho, peso e energia consumida tem tornado os meios de ligação entre computadores cada vez mais flexíveis, sendo que recursos não necessitam de uma localização fixa e nem precisam estar fisicamente conectados para se comunicar.

Nesse contexto, o surgimento de novas aplicações advém, sobretudo, da sinergia criada pela combinação da mobilidade com novas tecnologias de armazenamento digital, comunicação e processamento de dados. As redes sem fio e os dispositivos móveis introduzem novas exigências para a engenharia de *software* devido às limitações físicas. A demanda por acesso à informação sem restrições de local e de forma rápida e fácil tem motivado a pesquisa e o desenvolvimento para aplicações e novos tipos de sistemas.

O desenvolvimento de sistemas para o ambiente móvel é considerado complexo, pois são executados em ambientes dinâmicos e heterogêneos com freqüentes e rápidas flutuações na qualidade de serviço das redes e variabilidade de fontes de acesso e disponibilidade. Outros aspectos que aumentam a complexidade são fatores como tempo, espaço, conectividade, portabilidade e mobilidade. Para suavizar o impacto dessas trocas, a aplicação pode ter um comportamento adaptativo. O sistema pode modificar-se, para melhorar tempo de resposta, recuperar-se de uma falha, otimizar o uso de recursos ou incorporar novas funcionalidades.

Embora os projetos relacionados a esse contexto já venham sendo explorados há algum tempo, a usabilidade das aplicações móveis está longe de oferecer qualidade de navegação aos usuários. O conteúdo desenvolvido originalmente para a Web não deve ser transferido para o ambiente móvel, sem algum tipo de otimização.

De acordo com Nielsen (2006), a usabilidade é considerada um fator competitivo na Web. Relacionada às aplicações móveis está muito distante de oferecer uma boa experiência a seus usuários, por vários motivos, um deles devido ao *design* dos dispositivos. O conteúdo desenvolvido originalmente para a Web não deve ser transferido para o ambiente móvel, sem algum tipo de otimização. Para o autor, em uma tela pequena, seria preferível obter serviços mais específicos com menos opções e escrita abreviada.

Um aspecto importante com relação à computação móvel é a utilização de uma arquitetura adaptável, utilizando aplicações cientes de contexto, que devem ser capazes de adquirir informações de contexto do usuário de modo automatizado, disponibilizando-as em um ambiente computacional em tempo de execução.

O processo de adaptação pode ocorrer de formas diferentes, como exemplo pode-se citar a adaptação do conteúdo, de rede, dos dados e da interface do usuário. Para ser considerada adaptativa, a interface é desenvolvida apenas uma vez, no entanto ela pode ser visualizada em múltiplos contextos desde que se adapte a eles.

Para que sistemas dessa natureza sejam bem aceitos é fundamental que se conheça o perfil de usuário, pois estão em constante movimento, realizando diversas atividades enquanto se deslocam. Torna-se imperativo que a usabilidade de aplicações e aparelhos móveis seja um atributo de qualidade destacando a facilidade de uso e de aprendizado. Nesse ambiente, a mobilidade pode exigir interfaces dinâmicas que mudem de acordo com as necessidades de locomoção do usuário, estado e ambiente.

A reutilização de código é um aspecto fundamental nesse contexto. O reuso de interfaces do usuário é um aspecto problemático, pois a cada dia surgem novos modelos de dispositivos móveis com características diversas. Para flexibilizar a reutilização de código, é necessário criar e utilizar formas de desenvolvimento que sejam menos dependentes das propriedades de um dispositivo específico.

A interface deve ser capaz de migrar de um computador para outro, independente do conteúdo da aplicação, embora deva manter a consistência dos dados durante esse processo. Nesse contexto, implementar uma interface para cada tipo de dispositivo exige um grande esforço por parte do desenvolvedor devido à diversidade de tamanhos de telas existentes. Uma solução consiste em modelar a aplicação uma única vez de forma que ela possa ser exibida em diversos dispositivos.

A solução adotada pela arquitetura para Geração de Interfaces Adaptativas (GIA) utiliza a separação da aplicação em camadas a qual permite que se altere a exibição ou entrada de dados de qualquer sistema com um mínimo de impacto. Esta é uma separação pertinente, pois não há como se prever quais dispositivos, protocolos ou programas para visualização prevalecerão ou surgirão em breve. Atualmente, uma grande diversidade de fabricantes disputa o estabelecimento de um padrão no ambiente móvel, dentre eles diversos sistemas operacionais e soluções como Windows-CE, Palm-OS, Celulares com Wap, i-Mode, Java, entre outras.

1.1 Objetivos do Trabalho de Pesquisa

Esse trabalho tem como objetivo apresentar a proposta de uma arquitetura para Geração de Interfaces Adaptativas (GIA) que se ajuste às características de múltiplos dispositivos, permitindo que uma interface possa migrar de um computador para outro, preservando a consistência e usabilidade. Dessa forma, pretende-se reduzir o tempo de desenvolvimento, utilizando uma metodologia menos dependente de propriedades de um dispositivo único, utilizando para tal o conceito de adaptação.

A finalidade de se propor esta arquitetura é permitir que uma implementação esteja voltada a um ambiente multi-plataforma, partindo de uma descrição da interface genérica que seja independente de dispositivo e que atenda as solicitações de diversos usuários a uma mesma aplicação.

Além disso, deseja-se que os especialistas como desenvolvedores de *software* e *designers* de interface possam implementá-las tanto para *desktops* como para

dispositivos móveis, sem a necessidade de programação adicional. Para tal, esse trabalho visa propor um ambiente de desenvolvimento com o objetivo de integrar o reconhecimento de dispositivos móveis em tempo de execução a uma metodologia de fragmentação de código. Esta metodologia consiste em particionar uma interface Web dentro de um conjunto de subpáginas gerando um menu de conteúdos de forma hierárquica para facilitar a navegação em diversos tipos de dispositivos.

1.2 Motivação do Trabalho de Pesquisa

Ao se projetar uma aplicação para o ambiente móvel um grande número de aspectos devem ser analisados, principalmente fatores relacionados ao ambiente heterogêneo, a mobilidade do usuário e a diversidade de dispositivos existentes.

Uma das abordagens propostas para tratar questões de evolução dos sistemas computacionais é construí-lo com base em uma arquitetura e projeto concebidos de forma a serem adaptativos. Um sistema adaptável pode ser definido “como um sistema que pode ser facilmente modificado”, de acordo com Lieberherr e Lopes (1995).

De acordo com Sendin e Lores (2004), as arquiteturas para a antecipação de mudanças contextuais é umas das principais lacunas da literatura. Alguns trabalhos de pesquisa têm sido desenvolvidos com o objetivo de adaptação da interface no contexto móvel, como, a arquitetura MUSA, proposta por Menkhaus (2002) e o Dygimes por Coninx et. al. (2003). Os autores propõem uma forma de desenvolvimento que seja menos dependente de propriedades de um dispositivo único. Utilizam o conceito de separação por camadas, fazendo uma clara distinção entre o modelo de implementação de cada nível: interface do usuário e modelo lógico.

Um indicativo de que o desenvolvimento para esse contexto tem potencial para ser explorado foi dado em Nielsen (2006). Nesse artigo comenta-se que as aplicações móveis estão sendo planejadas sem qualquer estudo sobre usabilidade, dando mais ênfase a funcionalidade do sistema.

Apesar dos autores mencionados destacarem os benefícios que as técnicas podem trazer, torna-se necessária a elaboração de estudos para comprovação da aplicabilidade delas no desenvolvimento de sistemas reais de acordo com Murphy e Walker (1999). Estudos têm sido conduzidos com o intuito de avaliar a usabilidade e utilidade de algumas dessas técnicas por Kaikkonen e Virpi (2003) e Hassanein e Head (2003).

Diante dessas observações surge uma motivação importante para o desenvolvimento desse trabalho: a necessidade de uma maior flexibilidade do *software* sem a necessidade de programação adicional e que façam mudanças em seus modelos de domínio em tempo de execução, garantindo, sobretudo um grau de usabilidade para o cliente móvel.

A arquitetura GIA propõe uma solução na tentativa de superar os problemas atuais de complexidade de desenvolvimento de interfaces para dispositivos móveis. Para tal, utiliza conceitos de adaptação da interface a diversas classes de aparelhos, identificação do contexto do usuário em tempo de execução, preservação da consistência e usabilidade dos dados.

1.3 Metodologia de Desenvolvimento do Trabalho de Pesquisa

Na primeira etapa desse trabalho, realizou-se um levantamento da bibliografia existente e um estudo das diversas metodologias, tecnologias, filosofias e arquiteturas que abrangiam os aspectos que o motivaram. Assim, esse estudo abordou conceitos sobre o ambiente móvel, suas aplicabilidades, vantagens e restrições, bem como as aplicações existentes e, finalmente, conceitos e aplicações sobre interfaces adaptativas e tecnologias existentes, abordados nos capítulos 2 e 3.

Ainda nesta etapa, foram definidas e apresentadas as motivações do trabalho desenvolvido, delimitados seus objetivos e as contribuições esperadas e, também, definidas as tecnologias e metodologias que seriam utilizadas durante sua elaboração. Esta fase foi compreendida pela proposta de Tese de Doutorado.

Uma vez definidos os objetivos da arquitetura proposta, foram buscados complementos na literatura relacionada e procuradas as tecnologias e ferramentas pertinentes. Esse trabalho, tal qual proposto, incorpora e utiliza a *Unified Modeling Language* (UML) como notação para representar os diagramas de casos de uso, de classe, de pacotes, de atividades, bem como a tecnologia de comunicação *Apache Tomcat* e a tecnologia de banco de dados relacional para o armazenamento e recuperação dos dados.

Dentro desse universo, foram pesquisados e procurados sistemas e ferramentas que contemplassem estas tecnologias, metodologias e arquiteturas, e as escolhas recaíram sobre os sistemas e ferramentas que se mostraram suficientemente abrangentes, e que estavam disponíveis para estudo e pesquisa. Nesta fase, as ferramentas foram escolhidas, instaladas e analisadas. Para o reconhecimento de dispositivos móveis foi definido o uso do CC/PP descrito no capítulo 4.

Uma vez definidas as ferramentas, passou-se para a fase de delimitação e detalhamento da arquitetura GIA e seus componentes, ou seja, da estrutura e funcionamento da arquitetura como um todo, e em relação a cada um de seus componentes. Dessa forma, foram definidos quanto às suas estruturas e funcionamento, os seguintes componentes: serviço de conexão, serviço de adaptação e serviço de configuração e editoração, abordados nos capítulos 5 e 6. Em seguida, passou-se para as fases de análise e projeto do *software*, utilizando-se a ferramenta UML escolhida e instalada para a geração dos devidos diagramas de *software*, descrito no capítulo 7.

Terminada essa etapa, foi desenvolvido um protótipo baseado no projeto realizado, visando avaliar o quão factível mostrava-se a arquitetura GIA, e como seus componentes poderiam, de fato, interagir e colaborar para a elaboração de interfaces adaptativas. Primeiramente foram inseridos dados por meio da interface do Gerenciador de Recursos e posteriormente o sistema proposto foi testado em simuladores de alguns aparelhos utilizando-se para tal, a interface disponibilizada pelo ambiente de desenvolvimento. Esses testes foram demonstrados por meio de um estudo de caso, detalhado no capítulo 8.

Finalmente, o trabalho de pesquisa realizado foi expresso nesta tese, que tem a sua estrutura detalhada a seguir.

1.4 Organização da Tese

Para que a arquitetura GIA pudesse ser definida e projetada, algumas tecnologias já mencionadas são apresentadas, de forma sucinta, nos primeiros capítulos desse trabalho. Assim, tem-se:

Capítulo 2 – *O Ambiente Móvel e suas Aplicações*: nesse capítulo são abordados os conceitos sobre computação móvel, aplicabilidades, restrições e algumas arquiteturas utilizadas nesse ambiente. Quanto às aplicações aborda-se o desenvolvimento de sistemas para o ambiente móvel e em linhas gerais, as arquiteturas e tecnologias como as plataformas .NET e Java que surgiram como soluções nesse contexto e que estão incorporadas ao projeto da arquitetura GIA.

Capítulo 3 – *Adaptabilidade e Interface do Usuário*: nesse capítulo é apresentada uma visão geral sobre *software* adaptativo e tipos de interfaces para o ambiente móvel, bem como a complexidade de desenvolvimento delas para computadores de mão. São abordados, ainda, modelos de interface adaptável e adaptação dinâmica da interface do usuário. Apresentam-se formas de identificação do cliente móvel, como o *User-Agent* e o repositório de perfis proposto pela W3C, denominado *Composite Capability Preference Profile* (CC/PP). A biblioteca *Delivery Context Library* (DELI) utilizada pela arquitetura GIA também é detalhada nesse capítulo.

Capítulo 4 – *Trabalhos Relacionados*: nesse capítulo são descritas as principais arquiteturas relacionadas ao ambiente móvel, bem como suas características, vantagens e restrições. Os principais trabalhos estudados, MUSA descrito por Menkhaus (2002), Dygimes por Coninx et al. (2003), NAC (*Negotiation and Adaptation Core*) proposto por Lemlouma e Layaida (2004), Arquitetura Reflexiva descrita por Sendin e Lores (2004), *Mobile Application Servers* (MAS) sugerido por Viana et al. (2005) e o modelo MVC são detalhados nesse capítulo.

Uma vez que foram delineadas as tecnologias, metodologias e arquiteturas que embasaram esta pesquisa são apresentados os capítulos referentes à própria arquitetura GIA:

Capítulo 5 – *Arquitetura GIA*: nesse capítulo é apresentada a arquitetura GIA por meio da descrição e da definição do escopo e do funcionamento de seus serviços: serviço de comunicação, serviço de adaptação e serviço de configuração e editoração. É definida também a sua relação com as tecnologias de armazenamento, comunicação e distribuição que foram abordadas. São apresentados os atores e os diagramas de caso de uso e de atividades referentes à arquitetura GIA.

Capítulo 6 – *Metodologia de Desenvolvimento para Geração de Interfaces Adaptativas*: nesse capítulo são descritas as estratégias utilizadas para a utilização da arquitetura GIA, ou seja, a metodologia proposta para a implementação das interfaces adaptativas. São apresentadas diversas formas de exibição de uma interface, dentre elas a do *layout* original e a do *layout* reduzido. Nesse capítulo apresenta-se o *layout* fragmentado proposto nesta tese e os detalhes sobre as *tags* de regiões utilizadas para o processo de adaptação na arquitetura GIA.

Capítulo 7 – *Ambiente de Desenvolvimento GIA*: nesse capítulo apresenta-se o ambiente de desenvolvimento GIA, com o objetivo de integrar o reconhecimento de dispositivos móveis em tempo de execução e o desenvolvimento de interfaces utilizando a metodologia proposta no capítulo anterior, bem como o gerenciador de recursos do sistema. Ainda no capítulo 7 são apresentados os diagramas de classe referentes à arquitetura GIA, bem como detalhes referentes à implementação da ferramenta.

Capítulo 8 – *Estudo de Caso*: nesse capítulo são descritos testes obtidos para validação da arquitetura GIA, bem como da metodologia proposta nesse trabalho por meio de um estudo de caso com simuladores de dispositivos móveis, como telefone celular e *Personal Digitant Assistant* (PDA) . Para tal, são descritos dois cenários da utilização da

arquitetura, o cenário 1: do projeto á implementação e o cenário 2: solução proposta para uma interface existente.

Capítulo 9 – *Conclusão*: nesse capítulo são apresentadas as conclusões e os resultados obtidos com o trabalho de pesquisa, bem como algumas sugestões para a realização de trabalhos complementares.

Para finalizar esse trabalho são mostradas as referências bibliográficas e o apêndice. Assim tem-se:

- Referências Bibliográficas: nesse item são apresentadas todas as referências bibliográficas citadas no texto.

- Apêndice: são detalhados códigos relativos à implementação da arquitetura GIA, bem como o manual de instalação referente ao ambiente de desenvolvimento proposto.

CAPÍTULO 2

O AMBIENTE MÓVEL E SUAS APLICAÇÕES

A computação móvel é caracterizada pela utilização de computadores portáteis com capacidade de comunicação, fortemente associada com a mobilidade de *hardware*, *software* e informação. É caracterizada por um tipo de computação distribuída de acordo com Tewari e Grillo (1995), em que a visão de diversos computadores autônomos é transparente para os usuários e a distribuição das atividades é realizada automaticamente pelo sistema. Na computação distribuída existe a possibilidade de divisão do processamento entre computadores diferentes. Mais do que o simples compartilhamento de tarefas, esse paradigma permite a repartição e a especialização das tarefas computacionais, conforme a natureza da função de cada computador.

O modelo de computação móvel é baseado em computadores fixos e computadores móveis conectados por uma rede, permitindo mobilidade. Usuários podem se locomover levando consigo seu trabalho, realizando todas as operações necessárias em qualquer lugar e a qualquer momento, graças à possibilidade de mudanças em sua localização.

Nesse contexto, o termo computação onipresente, *Ubiquitous Computing*, cunhado por Wiser (1991), refere-se a ambientes com objetos operados por computador e conectados em redes sem fio, ou seja, mobilidade, comunicação e poder de processamento integrado em vários objetos com finalidades diferentes. Assim sendo, um dos principais desafios da computação ubíqua são as aplicações contextuais que implicam a capacidade de informar sobre o seu ambiente corrente e como reagir quando o seu usuário muda de um ambiente para outro.

A computação pervasiva é considerada a próxima etapa de desenvolvimento tecnológico. Sua intenção é propiciar uma junção de dispositivos em um ambiente de

valor computacional unificado. Uma tecnologia que permite que sistemas computacionais alterem drasticamente sua forma de utilização.

A idéia básica da computação pervasiva é disponibilizar acesso computacional de modo invisível, no sentido de que o usuário não precisa conhecer a tecnologia, mas utilizá-la em todo lugar o tempo todo, oferecendo uma comunicação tolerante a falhas, alta disponibilidade e segurança, além de proporcionar uma interface amigável ao usuário.

2.1 Aplicabilidade da Computação Móvel

A computação móvel tem várias aplicabilidades distintas que estão sendo cada vez mais discutidas, implementadas e aprimoradas. Usuários têm a possibilidade de participar de congressos, convenções e reuniões de negócios e realizar visitas a clientes e fornecedores, tendo acesso a seus arquivos pessoais armazenados em uma estação de trabalho distante, participando de teleconferências e efetuando, normalmente, suas tarefas computacionais diárias, mesmo quando distantes de sua residência ou local de trabalho.

Vendedores de campo são exemplos de trabalhadores com alta mobilidade que necessitam acessar bases de dados remotas e executar operações diversas. No seu computador portátil, carregam um banco de dados, contendo os produtos que são vendidos, podendo entrar em contato com a estação base de sua empresa repassando as vendas realizadas, atualizando o cadastro com novos clientes, realizando consultas de preços, de quantidades em estoque, do valor da comissão a ser recebida, da requisição de mercadorias, entre outras informações.

2.2 Restrições da Computação Móvel

Segundo Al-Bar e Wakeman (2001), a computação móvel apresenta três principais desafios que afetam seu desempenho, descritos a seguir:

- *Restrições dos dispositivos móveis:* devido à limitação de tamanho, dispositivos móveis continuarão tendo algumas limitações, como exemplo, velocidade do processador, capacidade de memória, tamanho da tela e sua resolução. A diversidade arquitetural proporciona grande variabilidade das configurações de *hardware* das unidades móveis;

- *Restrições de comunicação*: limitações como largura de banda, desconexão, alta latência e área de cobertura. O aspecto mais crítico é a largura de banda, que aumenta o consumo de bateria dos dispositivos;

- *Restrições de mobilidade*: o usuário de dispositivos móveis está em constante movimento e, portanto, realiza diversas mudanças de localização. Devido a essas movimentações, podem existir desconexões do dispositivo móvel, havendo falhas de comunicação com a rede. A conexão móvel é altamente variável em desempenho e confiabilidade. Além disso, existe a perda temporária de comunicação quando ocorre deslocamento entre áreas mantidas por diferentes estações base e renegociação de características de acesso.

2.3 Arquiteturas para Mobilidade

O ambiente móvel possui diversos tipos de arquiteturas que podem ser utilizadas de acordo com a necessidade de cada aplicação. A seguir são descritas as principais arquiteturas, bem como suas características, incluindo persistência de dados e tipos de interface.

2.3.1 Arquitetura para Computação Móvel

A computação móvel consiste em um sistema composto por dois tipos distintos de entidades, denominadas computadores móveis e computadores fixos, segundo Dunham e Helal (1995), relacionados a seguir:

- **Computadores Móveis:** são computadores portáteis, interligados em rede por meio de ligações sem fio, que permitem a mobilidade estabelecendo uma conexão virtual de qualquer localização dentro de uma área geográfica, como ilustra a Figura 2.1;

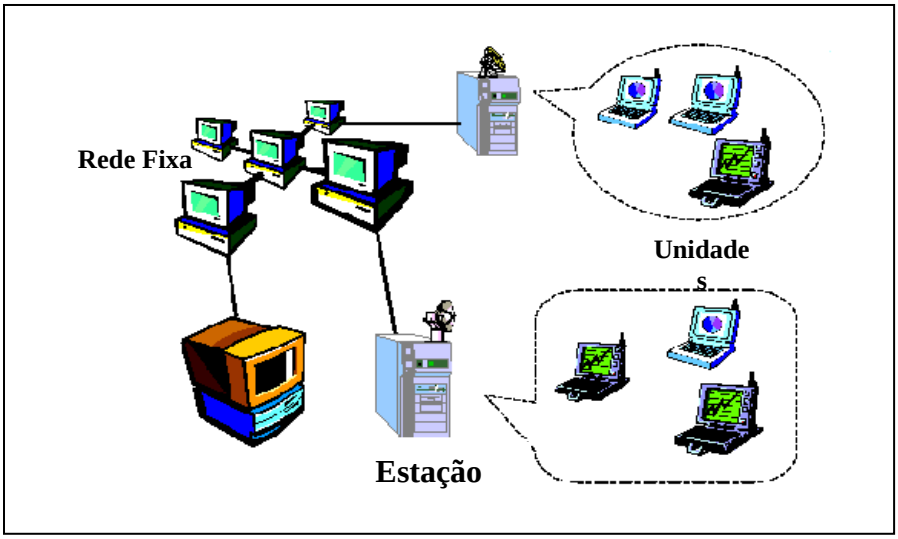


FIGURA 2.1 - Arquitetura para Computação Móvel
Fonte: Adaptada de SSU et al. (1998)

- **Computadores Fixos:** fazem parte de uma rede fixa cuja localização e conectividade não mudam. São identificados como servidores e estão localizados próximos ou em uma torre de antena, que transmitem ou recebem sinais eletromagnéticos dos dispositivos numa área específica;
- **Célula:** Para gerenciar a mobilidade das unidades móveis, o domínio geográfico é dividido em pequenos subdomínios que são cobertos por uma comunicação sem fio, denominados células. Cada célula é gerenciada por uma estação base, com transmissores e receptores para responder ao processamento de informações necessárias dos clientes. Caso os computadores móveis estejam próximos, a comunicação poderá ser realizada sem a intervenção de uma estação base.

2.3.2 Arquitetura Cliente Inteligente

A arquitetura cliente inteligente mantém um mecanismo de armazenamento de dados persistente, bem como, a estrutura lógica e a interface com o usuário. Os dados são

processados e armazenados no cliente como apresenta a Figura 2.2, fazendo com que uma aplicação possa ser executada sem a necessidade de comunicação com o servidor. A arquitetura cliente inteligente pode ser empregada em serviços que necessitam de alta mobilidade, como automação de vendas, profissionais da saúde, entre outros.

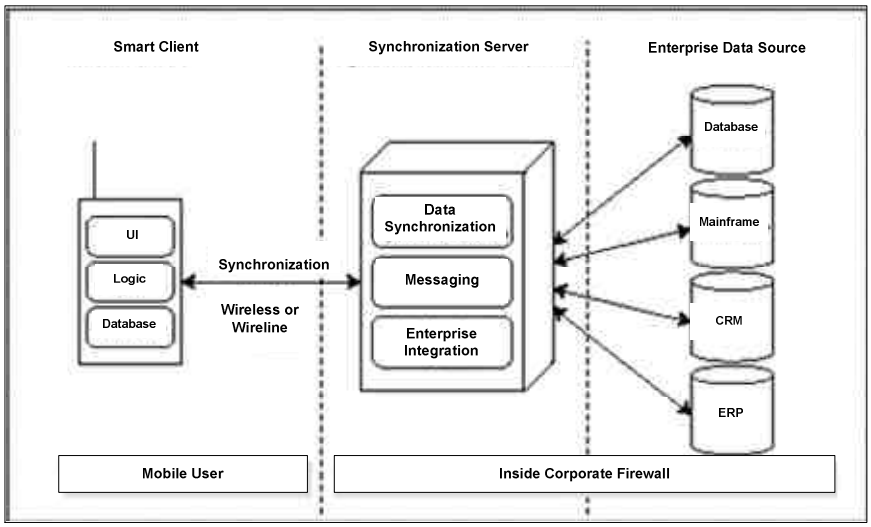


FIGURA 2.2 - Arquitetura Cliente Inteligente
Fonte: Mallick (2003)

2.3.3 Arquitetura Internet Sem Fio

Ao contrário do cliente inteligente, a arquitetura para Internet sem fio é considerada semelhante à de rede com fio, segundo Mallick (2003). Os componentes são os mesmos, a única diferença é a forma de transmissão da informação ao usuário. A estrutura lógica da aplicação é armazenada no servidor, bem como todos os dados. Essa arquitetura apresenta-se na Figura 2.3.

Para acessá-los o cliente necessita de uma conexão de rede sem fio. Esse modelo pode ser empregado em serviços que necessitam de atualização rápida, como serviços de informações, transações financeiras e comércio eletrônico.

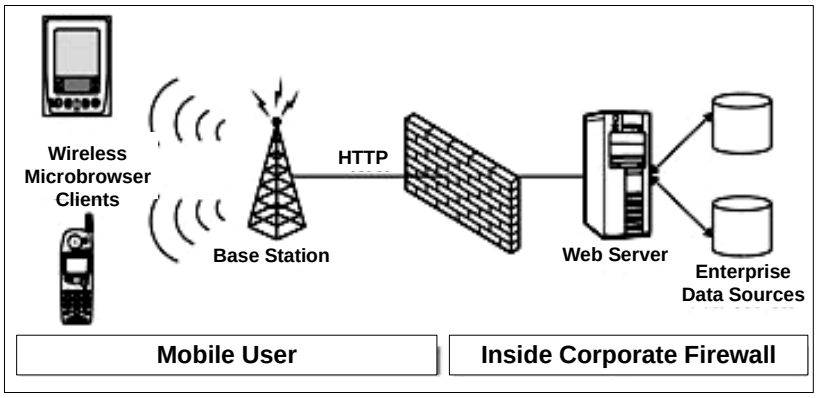


FIGURA 2.3 - Arquitetura para Internet sem fio

Fonte: Mallick (2003)

2.4 Tipos de Dispositivos Móveis

As tecnologias de computação móvel abrangem computadores móveis comumente designados por PDAs e *handhelds* e outros dispositivos que refletem a convergência entre os computadores móveis e celulares.

Os dois principais modelos de PDAs são o Palm e PocketPC. O *Palm* utiliza o sistema operacional Palm-OS restrito quanto ao número de fabricantes que o adotaram. Os PocketPCs utilizam o sistema operacional Windows *Mobile* e aplicativos para a plataforma *Windows*, sendo adotado por diversos fabricantes de PDAs, dentre elas a HP/Compaq e a Dell. Apresentam telas maiores e várias opções de ferramentas de desenvolvimento. A terceira alternativa para PDAs utiliza o sistema operacional Linux.

Nesse contexto, existem vários modelos de aparelhos que são utilizados com inúmeras funções. A arquitetura desses computadores de mão oferece tamanhos reduzidos, diversos aplicativos, interface com o usuário simples e prática e bateria suficiente para vários dias.

A escolha do equipamento adequado depende de fatores como tipo de atividade, modelo de captura para apresentação das informações, volume de dados esperados entre outras. Para equipes de profissionais móveis que consomem grande parte do seu tempo trabalhando remotamente, esses equipamentos são versáteis, dedicados, multifuncionais e de uso genérico. Do ponto de vista empresarial, eles são ótimos geradores de

informações, podendo ser utilizados desde a automação de processos até a coleta de informações estratégicas.

Segundo Ribeiro (2004), as principais características que distinguem os dispositivos móveis de outros tipos de computadores incluem:

- A utilização de telas sensíveis ao toque, combinados com a utilização de caneta óptica, como meio principal de introdução de informação que permite selecionar informação na tela e escrever símbolos que são interpretados como letras;

- A inclusão de botões de navegação rápida que dão acesso às aplicações de gestão de informação mais utilizadas;

- A disponibilidade de opções de comunicação com outros dispositivos computacionais, incluindo a utilização de cabos seriais e *Universal Serial Bus* (USB), infravermelhos e *bluetooth* para comunicações entre o dispositivo móvel e outro dispositivo, e *wireless fidelity* (WiFi) para integrar o dispositivo móvel numa rede local sem fios;

- A capacidade de sincronizar o conteúdo da memória do dispositivo móvel com um *desktop,* ou com um computador portátil. O processo de sincronização funciona de modo a assegurar que no final da operação, ambos possuam uma cópia exata da informação.

- A utilização de cartões de memória, que permitem expandir a capacidade de armazenamento limitada proporcionada pela memória RAM do dispositivo móvel, tais como os cartões *Secure Digital* (SD), *Compact Flash* (CF) e *Memory Stick* (MS). Esses cartões permitem ainda transferir informação de um dispositivo móvel para outro, ou diretamente para um computador de mesa, ou computador portátil.

- A utilização de baterias recarregáveis que, dependendo da tecnologia, permitem tempos de utilização efetivos que variam entre algumas horas a alguns dias.

- A utilização de sistemas operacionais especificamente desenvolvidos para dispositivos móveis, que gerenciam com mais eficiência o consumo de energia, bem como as características específicas do *hardware* dos computadores móveis, tais como o PalmOS, o Microsoft PocketPC e o Symbiam EPOC.

- A capacidade de executar aplicações móveis, isto é, qualquer aplicação que seja concebida para operar no ambiente computacional de um dispositivo de computação móvel.

2.5 Aplicações para o Ambiente Móvel

Os sistemas para ambientes móveis são mais complexos quando comparados aos ambientes tradicionais, pois as prioridades do usuário podem ser imprevisíveis e o contexto no qual a aplicação pode ser utilizada está em constante mudança. É importante considerar que os usuários geralmente buscam respostas rápidas, preferindo interfaces fáceis de usar, mesmo com funcionalidades mais restritas.

No entanto, a simplicidade é um elemento chave em uma solução de projeto, sendo traduzida em naturalidade e intuição no uso, proporcionando uma sintonia com as necessidades e experiências dos usuários. Dessa forma, o conteúdo deve levar em consideração tanto a impaciência e a atenção limitada dos usuários quanto sua alta mobilidade durante a utilização do dispositivo móvel, necessitando de alguma forma de adaptação.

Algumas soluções têm sido propostas no sentido de contribuir para o desenvolvimento de interfaces Web para dispositivos móveis. Para os pesquisadores Mennecke e Strader (2003), a largura de banda restringe a quantidade de material que pode ser enviado por meio da rede sem fio, do servidor ou de outro dispositivo.

Devido às restrições apresentadas pelos dispositivos móveis, os desafios em termos de *design* requerem habilidades dos projetistas, diferentemente da interface tradicional para computadores pessoais. Funções interativas devem se apresentar de maneira intuitiva para o usuário. Adicionalmente, a criação de modelos de PDAs tem sido influenciada pelo ambiente de negócios o qual é caracterizado pela constante pressão e necessidade de mercado por modelos inovadores, assim como significativo avanço tecnológico.

Nesse contexto, tais dispositivos pressupõem a participação efetiva de um projetista, pois, ao contrário, a não participação desse profissional pode incorrer em problemas como aplicações não intuitivas, falta de padronização, navegação confusa entre telas e uso desordenado de cores e gráfico.

Alguns pesquisadores têm formulado alternativas de métodos de interação com o usuário que viabilizam sua constante mobilidade. Kristoffersen e Ljungberg (1999) desenvolveram o método MOTILE o qual requer atenção reduzida do usuário utilizando o som como forma de resposta às suas solicitações. A entrada de dados é realizada por botões e comandos estruturados no computador de mão.

Pascoe et. al (2000) formularam dois princípios genéricos para interface e *design* móvel. O primeiro intitulado MAUI (*Minimal Attention User Interfaces*) procura minimizar a atenção do usuário. O segundo é ligada ao contexto, no qual o dispositivo móvel auxilia o usuário com base no conhecimento do ambiente ao qual se encontra.

Kaikkonen e Virpi (2003) especificaram três modelos de interfaces Web para a realização de testes sobre usabilidade em dispositivos móveis. Os *sites* abordam sobre o mesmo conteúdo, mas o que os diferencia é a forma como os dados são apresentados, pois a navegação e o uso dos elementos variam de acordo com cada estilo de interface proposta. Os autores definiram três *layouts* para a interface classificados por meio de nomes de frutas: banana, maçã e laranja. A Figura 2.4 ilustra os estilos de interfaces propostos.

O estilo banana é caracterizado por páginas muito longas, listas de seleção, tabelas, navegação por meio de *links* com ícones grandes. O estilo laranja, por páginas curtas, formulário com múltiplas páginas, escolha para entrada de texto ou seleção de valores, tabelas de dados e pequenos ícones. O estilo maçã apresenta uma página que possibilita a procura de algum conteúdo por meio de um serviço de busca, não utiliza imagens e a entrada de dados é a forma textual.

Com base nessas interfaces, Kaikkonen e Virpi (2003) realizaram testes com diversos dispositivos móveis, cujos resultados concluíram que, em termos de tempo de navegação, o estilo laranja foi o mais demorado. O estilo maçã obteve mais sucesso com usuários mais experientes, pois eles têm mais facilidade ao utilizar essa forma de navegação. O estilo banana obteve resultados mais rápidos, pois apresentam o conteúdo em páginas mais longas, ao contrário do estilo laranja, que proporciona uma navegação mais lenta, pois uma página é particionada em diversas menores.

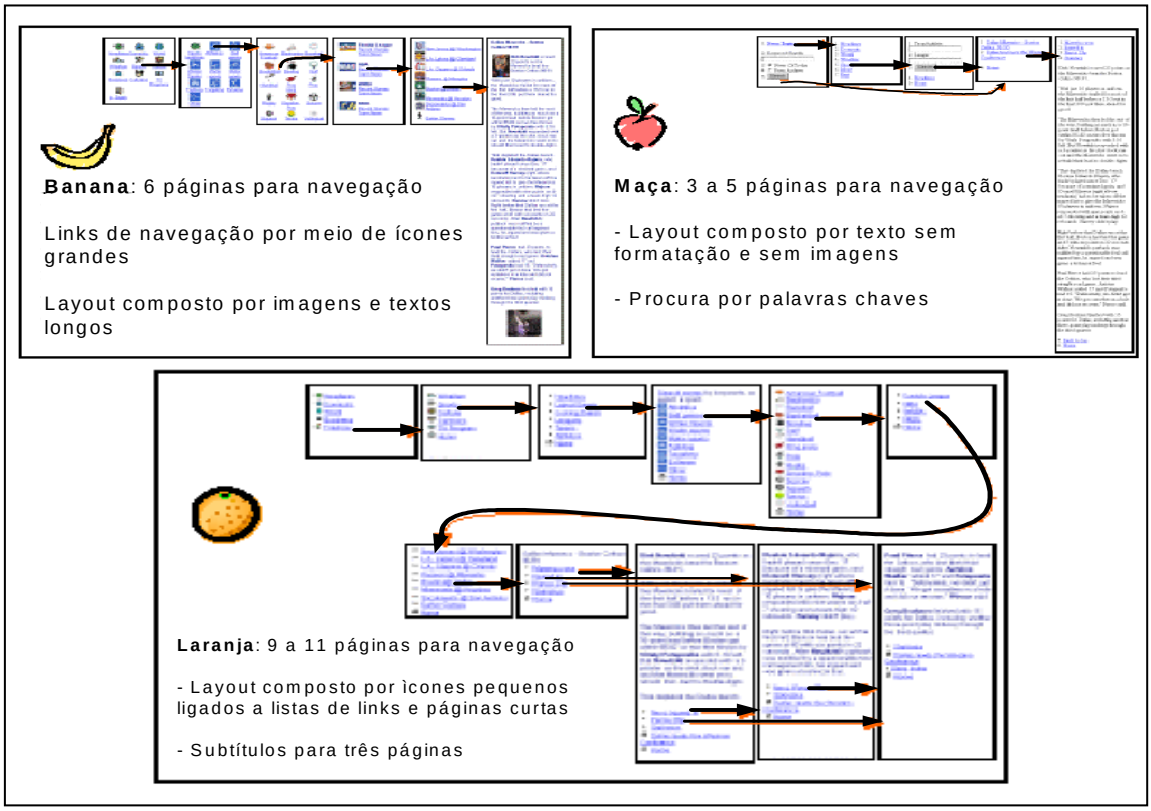


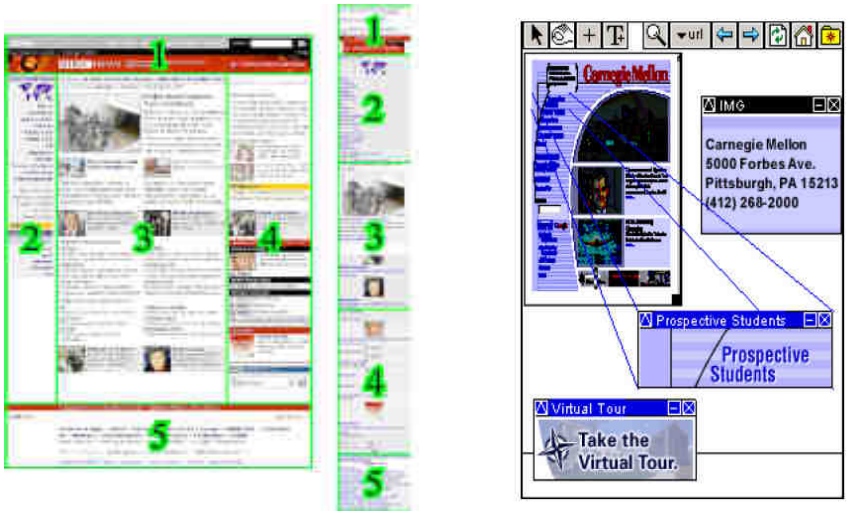
FIGURA 2.4 - Estilos de interfaces para dispositivos móveis
Fonte: Adaptada de Kaikkonen e Virpi (2003)

De acordo com Kaikkonen e Virpi (2003), o usuário prefere navegar em textos mais longos ao ter que escolher opções em diversas páginas. O tamanho ideal de uma página depende do tipo de conteúdo. Por exemplo, *sites* com conteúdo interativo devem ser mais curtos do que os que oferecem conteúdos textuais como textos informativos. Ambos devem conter *links* para o menu principal e com as opções anterior ou próxima. A questão é minimizar o número de passos para facilitar a navegação do usuário. Os autores afirmam que apresentar uma página de títulos inicialmente é a melhor forma de fazer com que o usuário localize a informações do *site* como um todo.

Segundo Xie et al. (2005), os trabalhos atuais focam em duas abordagens: a primeira é transformar páginas Web existentes, como apresentam os trabalhos de Chen et al. (2003), Gu et al. (2002) e Milic-Frayling e Sommerer (2002). A segunda introduz novos formatos e mecanismos para adaptar-se a diferentes tamanhos de tela, descritos nos artigos de Borning et al. (2000) e Han et al. (2000).

Na primeira abordagem, duas formas de transformação podem ocorrer, como exemplo, mudanças na forma de apresentação da interface sem qualquer modificação estrutural. Nesse caso, retira-se a barra de rolagem horizontal e o conteúdo é exibido em uma única coluna vertical, como mostra a Figura 2.5(a). Embora simples e rápida essa forma de navegação aumenta a altura da página forçando o usuário a usar a barra de rolagem vertical excessivamente.

Outra solução para essa abordagem utiliza o uso de palavras chave ou figuras como *zoom* em partes da página para facilitar a navegação, como ilustra a Figura 2.5(b). É mais indicada para usuários que tenham familiaridade com o conteúdo do *site*.



(a) Coluna Vertical (b) Visualizado com *Zoom*

FIGURA 2.5- Formas de Navegação em Sites Web

A segunda abordagem consiste em recuperar a estrutura semântica de um conteúdo original e reescrever a interface de acordo com o contexto do usuário. A técnica tem como objetivo particionar uma página Web dentro de um conjunto de subpáginas e gerar uma tabela de conteúdos com ou sem uma hierarquia gerando um índice da página. A arquitetura GIA pertence a essa categoria, pois considerando a metodologia de desenvolvimento proposta no capítulo 6, pode-se fragmentar uma interface em diversas partes para que se adapte ao tamanho da tela do dispositivo móvel que solicitar a aplicação.

2.6 Projeto de Interfaces para o Ambiente Móvel

Empresas líderes de telefonia móvel e Internet, como Nokia, Vodafone, Google, definiram um conjunto de melhores práticas para o desenvolvimento de W*eb sites* para telefones celulares. O *Worldwide Web Consortium* (W3C) também estabeleceu os principais fatores que devem ser considerados para o projeto de interfaces em ambientes móveis, descritos a seguir.

- **Seleção de Conteúdo:** as páginas devem conter apenas informações mais importantes, pois o tempo de *download* nos dispositivos móveis ainda é alto e a capacidade de memória de trabalho e armazenamento é pequena. O conteúdo deve ser adaptado para acesso em tela pequena, de modo que o usuário não use muito as barras de rolagem;

- **Janelas Independentes:** as imagens grandes ou anúncios em janelas independentes congestionam as telas dos telefones;

- **Visualização de Conteúdo:** os conteúdos mais importantes devem ficar no alto da página sem que o usuário precise fazer muitas rolagens de tela ou passar por diversos *links* para chegar ao que procura. A barra de navegação, por servir de referência para a navegação, deve também ficar localizada no alto;

- **Endereço do *Site*:** as URLs (*Universal Resource Locator)* devem ser curtas e fáceis de digitar e a entrada de dados pelos usuários em formulários reduzidos ao mínimo necessário, devido às limitações dos teclados. De modo geral, devem-se evitar entradas de dados que exijam digitação excessiva, pois o teclado nos telefones ou as canetas em assistentes pessoais não facilitam a escrita de textos longos;

- **Títulos:** os títulos dos textos devem explicar o conteúdo com o mínimo de palavras, e os textos devem ficar 50% menores do que os publicados em papel ou na Web;

- **Barra de rolagem**: deve-se limitar o uso da barra de rolagem em uma direção única, no caso dos PDAs indica-se a utilização da rolagem vertical, pois facilita a leitura do conteúdo do *site*. Deve-se evitar a barra horizontal devido à dificuldade de visualização do conteúdo. Sugere-se criar novas páginas ao se

utilizar à barra horizontal;

- **Cores:** deve-se evitar o uso de muitas cores na mesma página. As telas são pequenas e a mistura de diversas cores pode confundir o usuário;

- **Imagens:** recomenda-se utilizar imagens pequenas, simples e sem texto. Com o processo de redução do tamanho, os textos podem ficar totalmente ilegíveis;

- **Navegação:** para facilitar a navegação deve-se manter um *link* para o menu principal em todas as páginas, pois caso contrário, o usuário deve digitar sempre o endereço do *site* a cada navegação, causando um trabalho excessivo;

- **Tabelas:** recomenda-se não utilizar tabelas maiores que o tamanho da tela ou muito longas;

- **Textos:** em telas pequenas, a utilização de textos é mais indicada do que gráficos em cabeçalhos, pois são mais fáceis de adaptarem e serem visualizados. Os títulos das páginas devem ser curtos. É indicado o uso de no máximo três tipos de letras para não poluir o *layout* da interface;

- **Usabilidade:** nos dispositivos móveis, as informações essenciais devem ser mostradas na primeira tela, pois quanto mais rápido o usuário encontrar o que deseja mais usabilidade terá a interface.

2.7 Desenvolvimento de Aplicações para o Ambiente Móvel

Com a crescente popularidade dos dispositivos portáteis, incluindo os PDAs e celulares, há uma demanda também crescente pelos desenvolvedores, de criar um aplicativo único que possam ser executado numa ampla faixa de dispositivos.

Existem diversas alternativas para a implementação da interface do usuário. A idéia é a introdução de elementos da interface que sejam genéricos e independentes de plataforma, que podem ser implementados em linguagens como *Extensible HyperText*

Markup Language (XHMTL) e *Dynamic Markup Language* (DHTML) , *Wireless Markup Language* (WML) ou também em linguagens de programação apropriadas.

De acordo com Menkhaus (2002), existem basicamente duas formas de implementação da interface: encapsulamento de objetos e linguagem de marcação.

Encapsulamento de Objetos: é indicada quando um modelo lógico pode ter uma ou mais implementações. Para essa forma de implementação podem ser utilizadas a linguagem Java ou a plataforma .NET da Microsoft.

Linguagem de Marcação: podem ser utilizadas diversas linguagens de marcação como por exemplo, XML. Cada classe de dispositivo móvel possui o seu próprio *browser*, o que pode induzir o uso das linguagens *Hipertext Markup Language* (HTML) e suas combinações XHMTL e DHTML, WML entre outras.

Existem diversos conjuntos de ferramentas e bibliotecas para criar aplicativos no ambiente móvel, dentre eles o .NET *Compact Framework*, que atualmente se concentra no Microsoft PocketPC e dispositivos do Windows* CE .NET e o J2ME da plataforma Java. As linguagens XML e *eXtensible Stylesheet Language* (XSL) também são utilizadas para o desenvolvimento nesse contexto, como descrevem as próximas seções.

2.7.1 eXtensible Markup Language (XML)

A XML é um padrão desenvolvido pelo consórcio *World Wide Web Consortium* (W3C) para publicação, combinação e intercâmbio de informações. Assim como outras linguagens de marcação, algumas regras determinam onde a estrutura começa e termina por meio de *tags* que são colocadas antes e depois do conteúdo associado.

A linguagem XML provê um sistema para criar *tags* para dados estruturados. Uma das vantagens da utilização da XML, segundo Zucker et al. (2005) refere-se ao fato de que uma interface de usuário construída em uma linguagem markup, é facilmente desenvolvida e personalizada. Já Luyten e Coninx (2001) afirmam que devido à sintaxe

e gramática simples, a XML é uma linguagem intuitiva. É possível estruturar totalmente uma página, colocando os componentes que a interface irá conter e as propriedades referentes a ela.

Por ser um código bastante flexível, a manutenção da linguagem XML pode ser facilmente realizada. Além da descrição da interface, também pode ser usada para definir alguns padrões de estilo como cores de fonte, por exemplo, e essa referência será anexada à descrição da interface. Esses estilos podem ser definidos em um arquivo diferente da descrição para que se possa reutilizar o código, afim de que uma ou mais páginas possam utilizar os mesmos estilos aplicados em outra página.

A XML facilita a troca de informações entre aplicações, tornando-se útil como meio de integração de diversas fontes de informação e apresentação de interface uniforme para esses dados. Entretanto, a XML não é a solução geral para todos os problemas. Como toda a tecnologia apresenta vantagens e desvantagens. Como benefícios podem-se enumerar as seguintes características:

- **Extensível:** não são definidos elementos de marcação, mas informa como se podem criar seus próprios elementos ou regras;

- **Interoperabilidade:** A XML não tem dependência de sistema operacional ou plataforma;

- **Dados Autodescritivos:** uma das maiores vantagens da XML é que os dados são autodescritivos, assim sua estrutura é facilmente reutilizável em aplicações futuras.

Uma das limitações da linguagem XML refere-se ao código mais complexo, já que é preciso verificar a sintaxe com um analisador, a fim de garantir que os programas usados posteriormente funcionarão sem erros, tornando mais difícil sua codificação. A estrutura é bem definida, mas a semântica um tanto limitada.

Os documentos XML definem a estrutura dos dados, mas não descrevem como manipulá-los. Por exemplo, pode-se desenvolver um documento XML que tem um elemento <endereço>, mas não existe nenhuma facilidade em XML para especificar que tipo de endereço deve ser fornecido. O documento XML pode ter um elemento <endereço>, podendo ser um endereço residencial ou endereço de *email*.

A apresentação de um documento XML depende da folha de estilos a qual está associado. Os dois padrões mais utilizados para construir folhas de estilos são o *Cascading Style Sheet* (CSS) e o XSL.

O CSS é aceito por quase todos os *browsers*, mas é limitado em termos do que é capaz de construir. Não se pode com um CSS, por exemplo, mostrar os elementos em outra ordem que não à estruturada no documento. Já o XSL, apresenta mais recursos como transformar um documento XML em HTML, PDF, RTF e uma série de outros formatos, como descreve a seção a seguir.

2.7.2 eXtensible Stylesheet Language (XSL)

A XSL é um mecanismo que aplica uma formatação para a exibição dos dados descritos em XML. A linguagem de estilo XSL é composta de três linguagens descendentes de XML: XSL *Transformation* (XSLT) que especifica regras de transformações de documentos XML, Xpath que especifica como expressões são utilizadas para endereçar porções de um documento XML e XSL-*Format Objects* que corresponde a um vocabulário XML específico para a formatação de documentos.

A XSL utiliza *tags*, porém essas já são especificadas na própria linguagem. Dessa forma, uma mesma aplicação pode se apresentar de diferentes maneiras, de acordo com o dispositivo cliente.

É possível estruturar uma página inteira utilizando os dados descritos em XML aplicando-os às regras da XSL. No entanto vale ressaltar que o XSL ainda não é totalmente aceito nativamente por todos os navegadores Web, sendo que, na maioria dos

casos acaba sendo necessário fazer o processamento do lado servidor e enviar para o cliente o resultado em HTML.

2.7.3 eXtensible Stylesheet Language Transformations (XSLT)

A XSLT é uma linguagem para transformar arquivos XML em outros formatos como, exemplo, um documento XML em outro documento XML ou tipos diferentes de documento como TXT, PDF, RTF, HTML, entre outros.

XSLT é uma linguagem de consulta para documentos XML, uma vez que oferece comandos para filtrar o conteúdo que será apresentado. Pode ser usada para transformar um documento XML em um formato que é reconhecível para um *browser*. XSLT pode adicionar ou remover novos elementos no arquivo de saída, podendo rearranjar e ordenar e testá-los tomando decisões sobre quais elementos mostrar.

Durante o processo de transformação, a XSLT utiliza o XPath para navegar no arquivo XML em busca das partes do arquivo que possuam correspondência com um ou mais modelos definidos. Ao encontrá-los, a XSLT transforma essas correspondências e as coloca no arquivo resultante.

2.7.4 .NET Framework

O .NET é uma plataforma de desenvolvimento avançada produzida pela Microsoft, que apresenta como vantagem ser multi-plataforma e multi-linguagem, oferecendo suporte a mais de dez linguagens de programação, tanto as disponibilizadas pela própria Microsoft, como C#, VB.NET, C++.NET, J#, bem como outras linguagens oferecidas por outras empresas como Cobol, Pascal, Eiffel, Fortran, Perl e Phyton oferecidas pela Fujitsu.

É multi plataforma por apresentar um conceito similar à tecnologia JAVA. Todo código desenvolvido, ao ser compilado, é interpretado e depurado, já contendo as verificações de lógica e transformado em uma linguagem intermediária chamada MSIL (*Microsoft*

Intermediate Language). Essa linguagem intermediária somente é entendida pelo CLR (*Common Language Runtime*) que seria semelhante à JVM (*Java Virtual Machine)*. A interpretação é feita quando o código escrito é compilado, dessa forma, o papel da CLR é transformar o código MSIL na linguagem nativa da máquina em questão. Assim sendo, toda aplicação construída no .NET *Framework* pode ser executada em todas as plataformas que tem CLRs desenvolvidas.

É multi-linguagem, pois qualquer linguagem que seja compatível com a plataforma de desenvolvimento .NET pode ser utilizada, ou seja, se um determinado compilador de linguagem segue as especificações da CLS (*Common Languagem Specification*), ela é compatível com .NET, e gera códigos MSIL compatíveis com a CLR.

O .NET é orientado a objetos, pois permite a criação de classes com propriedades e métodos, incluindo método construtor e mecanismos de herança, polimorfismo, agregação, sobrescrita de métodos, sobrecarga de métodos, entre outras. Permite também que classes escritas em uma determinada linguagem, e compiladas, sejam lidas por qualquer uma das outras linguagens que seguem as especificações da CLS.

A plataforma .NET possui duas divisões principais de desenvolvimento, sendo um para computadores com maior capacidade de processamento e recursos de *hardware* denominada .NET *Framework* e outra destinada aos dispositivos que não possuem tantos recursos como os *smartphones* e *handhelds.* Para esses, a plataforma de desenvolvimento é a .NET *Compact Framework.*

2.7.5 Java para a Computação Móvel

A independência de plataforma que a Java oferece, a grande quantidade de bibliotecas disponíveis e a existência de máquinas virtuais embutidas em vários dispositivos tornou a plataforma Java uma tecnologia importante para o desenvolvimento de *software* na computação ubíqua.

A plataforma Java é atualmente dividida em quatro grupos, como ilustra a Figura 2.6.

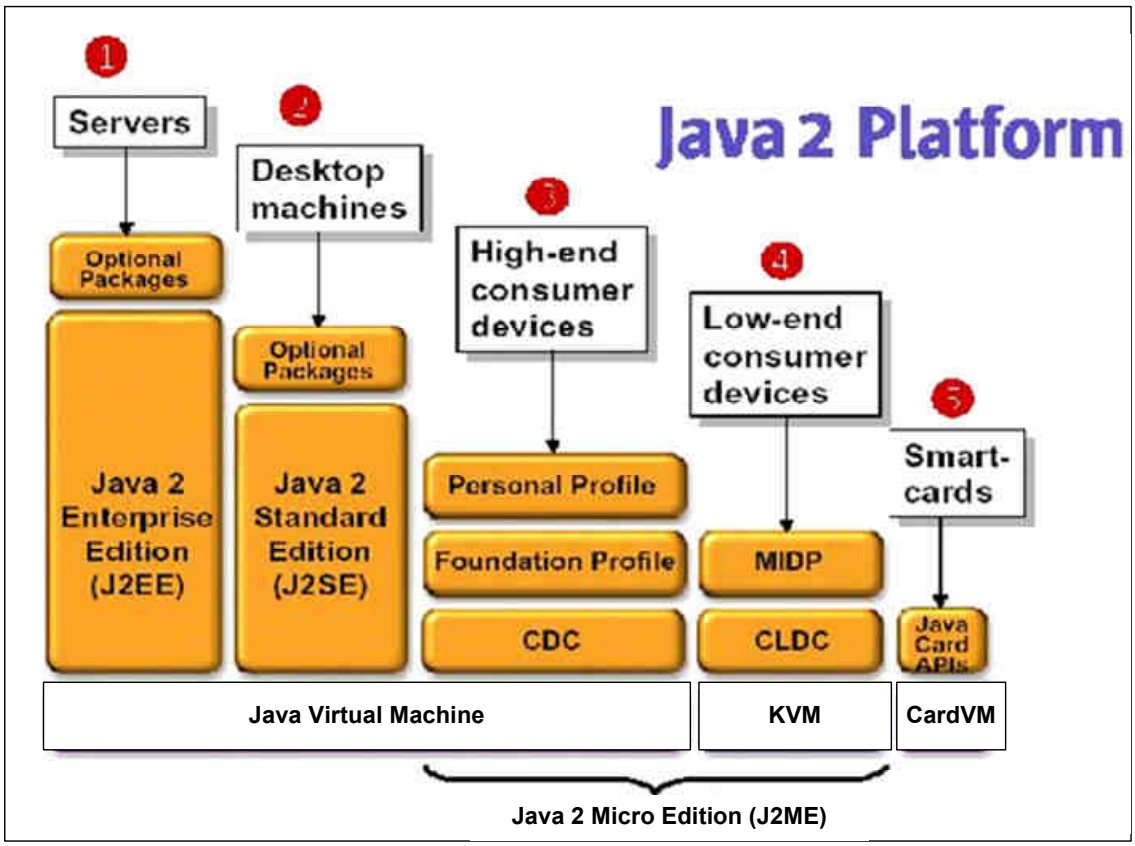


FIGURA 2.6- A Plataforma Java

O *Java 2 Standard Edition* (J2SE) é o grupo principal, destinado a computadores pessoais domésticos. O Java 2 *Enterprise Edition* (J2EE) é mais abrangente que a J2SE, sendo destinada a aplicações do lado do servidor. O *Java 2 Micro Edition (*J2ME) tem o objetivo de disponibilizar aplicativos Java em dispositivos portáteis, como PDAs , *pagers* e celulares. E a última, a *Java Card* é uma tecnologia utilizada em *smart cards* e outros dispositivos com configuração mais limitada.

De acordo com o contexto desse trabalho, apenas o J2ME e J2EE são detalhados na próxima seção.

2.7.6 Java 2 Micro Edition

A J2ME é uma coleção de APIs (*Application Programming Interface*) da plataforma Java definidas pela *Java Community Proccess* (JCP). É uma versão para dispositivos com limitações de memória e processamento, como celulares, Internet *screenphones*,

sistemas de navegação automotiva, comutadores e roteadores de rede, componentes para automação residencial, PDAs e sistemas embarcados em geral.

É caracterizada por um conjunto de especificações definidas pela Sun *Microsystems*, que define uma *Java Virtual Machine* (JVM) simplificada, um conjunto de APIs especializadas para pequenos dispositivos e ferramentas direcionadas. Consiste de documentos de configuração (CDC e CLDC) em conjunto com perfis (MIDP) para uma determinada aplicação.

A J2ME é dividida em configurações e perfis. As configurações são compostas de uma JVM e um conjunto de bibliotecas. Elas fornecem a funcionalidade básica para um número particular de dispositivos que possuem características similares, como conectividade e memória. São descritas duas especificações, CDC (*Connected Device Configuration*) e CLDC (*Connected Limited Device Configuration*).

A CDC pode ser utilizada com dispositivos com mais memória e poder de processamento como exemplo, televisão digital e sistemas automotivos com uma quantidade razoável de processamento ou memória. A CLDC é feita para dispositivos com menos de 512 KB de memória disponível para aplicações Java e uma conexão de rede limitada e não-permanente. Ela define uma máquina virtual Java reduzida, e um conjunto reduzido de APIs.

2.7.7 J2EE Edition

O J2EE *Edition* oferece suporte total para componentes *Enterprise JavaBeans* (EJB), *API Java Servlets*, *JavaServer Pages* (JSP), além de XML. O padrão J2EE inclui especificação completa e testes de conformidade para assegurar a portabilidade das aplicações no âmbito dos sistemas corporativos que aceitam o J2EE. A edição J2EE compreende as seguintes tecnologias:

- ***EJBs (Enterprise JavaBeans)*** – Arquitetura padrão de componentes para a construção de aplicações corporativas orientadas a objeto em Java. Constitui-se

de API que permite a criação, o uso e o gerenciamento de componentes da aplicação, reusáveis em diferentes plataformas;

- ***JSPs (Java Server Pages)*** – permite a criação de conteúdo dinâmico na web. Uma página JSP é um documento que descreve como processar uma solicitação para criar uma resposta, pela combinação de formatos padrão de marcação com ações dinâmicas (por meio de scripts);

- ***Java Servlets*** – API que estende a funcionalidade de um servidor Web, por módulos de aplicação implementados em Java *Conectores.* Permite que a plataforma J2EE seja conectada a sistemas de informação heterogêneos;

- ***CORBA*** – permite interoperabilidade de aplicações Java com outras aplicações empresariais, por meio de uma linguagem de definição de interface – IDL.

- ***JDBC (Java Database Connectivity)*** – API que promove a integração com bancos de dados relacionais, além de fornecer uma base comum sobre a qual ferramentas de alto nível e interfaces possam ser construídas.

Para a implementação dos serviços de comunicação, adaptação e integração da arquitetura GIA, bem como a metodologia de fragmentação de código proposta no capítulo 6, utiliza-se a linguagem de programação *Java Server Pages* (JSP) detalhada a seguir.

2.7.8 Java Server Pages

Java Server Pages (JSP) é uma especificação da *Sun Microsystems* que combina Java e HTML com o objetivo de fornecer conteúdo dinâmico para páginas Web. Pode ser comparada ao Microsoft *Active Server Pages* (ASP), porém tem a vantagem da portabilidade de plataforma podendo ser executada em diversos sistemas operacionais.

A JSP permite ao desenvolvedor Web produzir aplicações que acessem banco de dados e a arquivos-texto, a captação de informações a partir de formulários, a captação de informações sobre o visitante e o servidor, o uso de variáveis, entre outros.

As interfaces desenvolvidas em JSP são compiladas em *Servlets* Java, sendo carregadas em memória na primeira vez e são executadas por todas as chamadas seguintes, proporcionando velocidade e escalabilidade. Uma página JSP é basicamente um *site* desenvolvido em HTML tradicional com códigos Java.

A utilização da JSP proporciona ao desenvolvedor a possibilidade de separar a programação lógica da programação visual, facilitando o desenvolvimento de aplicações mais robustas, em que programador e *designer* podem trabalhar no mesmo projeto, mas de forma independente. Outra característica da JSP é produzir conteúdos dinâmicos que possam ser reutilizados.

De acordo com Temple et. al (2004) a utilização de s*ervlets* e de páginas JSP oferece diversas vantagens em relação ao uso de outras tecnologias como PHP, ASP e CGI. As principais vantagens:

- **Portabilidade:** a aplicação desenvolvida pode ser implantada em diversas plataformas, como Windows, Unix e Macintosh, sem que seja necessário modificar ou mesmo reconstruir a aplicação;

- **Facilidade de programação:** a programação é orientada a objetos, simplificando o desenvolvimento de sistemas complexos. A linguagem oferece algumas facilidades, como exemplo, o gerenciamento automático de memória (estruturas alocadas são automaticamente liberadas, sem que o desenvolvedor precise se preocupar em gerenciar esse processo);

- **Flexibilidade:** a linguagem Java encontra-se bastante difundida, com ampla documentação e diversas bibliotecas e códigos prontos, dos quais o desenvolvedor pode usufruir;

- **Escalabilidade:** na maior parte dos servidores de aplicações modernos, é possível distribuir a carga de processamento de aplicações desenvolvidas em diversos servidores, sendo que eles podem ser adicionados ou removidos de maneira a acompanhar o aumento ou decréscimo dessa carga de processamento;

2.7.9 *Tag* Library

Tag library é um elemento dinâmico que pode ser utilizado em uma página JSP, fazendo uma referência a uma biblioteca de *tags*, de acordo com Temple et. al (2004). O JSP permite desenvolver módulos reutilizáveis, chamados de ações personalizadas que são invocadas em uma página JSP. Alguns exemplos de tarefas incluem processamento de formulários, acesso a banco de dados e outros serviços.

Uma biblioteca de *tags* permite que a lógica de programação seja separada da aplicação do conteúdo da página JSP, assim como JavaBeans se propõe a fazer. No entanto, ao contrário de JavaBeans, uma biblioteca de *tags* oferece um acesso nativo aos objetos da página JSP, como, por exemplo, o objeto que encapsula a resposta do *Servlet* correspondente. Além disso, as *tags* tornam-se mais simples de se utilizar, principalmente para programadores HTML que não conhecem Java.

Outra característica das *tags* utilizadas com a linguagem JSP é o acesso a dados que fornecem acesso a um banco de dados a partir de uma visualização. Para aplicações simples essa metodologia pode ser empregada sem maiores problemas. Para uma grande aplicação, a melhor opção é utilizar um *Data Access Object*, que encapsula o do código de acesso a dados e fornece uma solução que pode ser mantida e reutilizada, segundo Alur et. al (2004).

A arquitetura GIA utiliza as *tags-libraries* com dois propósitos: para o redirecionamento de regiões e geração de componentes detalhados no capítulo 6.

O próximo capítulo apresenta uma visão geral sobre *software* adaptativo e tipos de interfaces para o ambiente móvel, bem como a complexidade de desenvolvimento para

computadores de mão. São abordados, ainda, modelos de interface adaptável e adaptação dinâmica da interface do usuário. Apresentam-se formas de identificação do cliente móvel, como o *User-Agent* e o repositório de perfis proposto pela W3C, denominado *Composite Capability Preference Profile* (CC/PP). A biblioteca *Delivery Context Library* (DELI) utilizada pela arquitetura GIA também é detalhada no próximo capítulo.

CAPÍTULO 3

ADAPTABILIDADE E INTERFACE DO USUÁRIO

Até recentemente, as interfaces eram criadas para aplicações estáticas. O projetista as construía e o usuário tinha que aprender como utilizá-las. Atualmente, diversos estudos estão sendo realizados em busca de interfaces mais flexíveis, que ajudem o usuário a realizar suas tarefas de maneira mais agradável e eficiente. A flexibilidade se refere geralmente a mudanças relativas à apresentação das informações, tais como mudança de cor, tamanho e posição de janela.

Existem porém, mudanças relativas ao conteúdo das informações, tais como definição de quais informações devem ser consideradas, quanto detalhadas elas devem ser, entre outras. Quando essas mudanças são diferenciadas por usuário ou por uma situação existente, elas assumem o nome de “adaptações”.

Dois fatores têm contribuído para o aumento do interesse pelo desenvolvimento de softwares adaptáveis, segundo McKinley et. al (2004). O primeiro é o paradigma da computação móvel que deve ter a capacidade de se adaptar a diversos ambientes e tipos de dispositivos móveis. O segundo é o crescimento da computação autônoma, a qual tem como objetivo o desenvolvimento de sistemas que tem a capacidade de se auto-gerenciar, configurar e proteger retirando das infra-estruturas tecnológicas todas as suas potencialidades, com a necessidade de menos administração humana do que a que é atualmente requerida.

O processo de adaptação pode ocorrer de formas diferentes e em níveis de uma aplicação e ambiente operacional. Por exemplo, adaptação do conteúdo, de rede, dos dados e da interface do usuário. De acordo com Menkhaus (2002), a estrutura, a navegação e a interatividade são critérios que devem ser acessados automaticamente,

sem que o usuário esteja envolvido. Outros critérios como conteúdo e *design* visual podem ser avaliados com o envolvimento do usuário.

Para ser considerada adaptativa, a interface é desenvolvida apenas uma vez, no entanto, ela pode ser visualizada em múltiplos contextos, desde que se adapte a eles.

3.1 *Software*s Adaptativos

Um programa é dito adaptativo se for capaz de alterar automaticamente seu comportamento de acordo com seu contexto, segundo Henricksen e Indulska (2001). A adaptação, nesse caso, é a capacidade de um algoritmo fornecer diferentes saídas válidas dependendo das características do ambiente onde o dispositivo móvel se encontre.

Os *software*s adaptativos devem possuir mecanismos que permitam coletar dados a respeito do estado de seu ambiente de execução, analisar esses dados visando identificar mudanças significativas e alterar dinamicamente seu comportamento para atingir seus objetivos. No campo das aplicações, muito se tem discutido a respeito daquelas associadas às unidades móveis e de mecanismos que permitam a elas se adaptarem às alterações do ambiente, dividindo a responsabilidade entre cliente e servidor.

Satyanarayanan (1996) define uma faixa de estratégias para a adaptação delimitada por dois extremos, *Laissez-Faire* e *Application-Transparent*, como mostra a Figura 3.1.

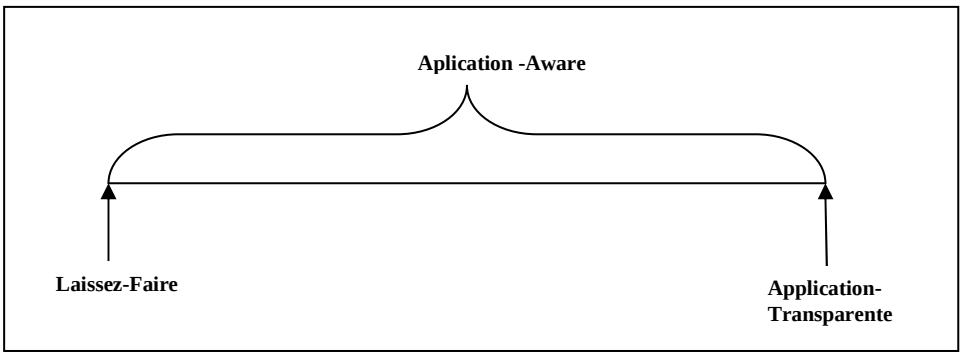


FIGURA 3.1- Modos de Adaptação
Fonte: Adaptada de Satyanarayanan (1996)

Na estratégia *Laissez-Faire*, a responsabilidade de adaptação é das aplicações individuais, não havendo nenhum suporte pelo sistema. Isso faz com que não haja um árbitro central para gerenciar incompatibilidades nos recursos provindos de diferentes operações. O sistema permanece intacto.

No outro extremo, chamado de *application-transparent*, ocorre o contrário: a infra-estrutura do sistema é responsável pela adaptação o qual contém um ponto central para gerenciamento e controle de dados. Para a aplicação, a adaptação é feita de forma transparente, dessa forma, unidades móveis continuam o processamento mesmo estando em movimento. Essa estratégia apresenta a desvantagem de que em muitas situações a adaptação realizada pelo sistema é inadequada com algumas aplicações e que existe a perda de controle sobre a adaptação realizada.

Entre esses extremos, existe uma estratégia colaborativa, chamada de *application-aware*, na qual a adaptação é feita tanto pelo sistema quanto pela aplicação. Nesse caso, o sistema operacional gerencia fontes (largura de banda, memória ou bateria), notifica aplicações de trocas ambientais relevantes, entre outras tarefas. A vantagem dessa aproximação é que a aplicação pode decidir qual a melhor forma de se adaptar às mudanças do ambiente, utilizando bibliotecas ou *toolkits*. A desvantagem é que desenvolvedor tem uma maior complexidade para a implementação da aplicação, de acordo com Menkhaus (2002).

Pesquisas recentes defendem a separação da funcionalidade do sistema para isolar a interface da parte lógica, pois facilitaria a integração de múltiplas interfaces. O conceito de abstração e separação é usado para garantir que mudanças de um componente não afetarão outros. Nesse caso, o objetivo é obter uma cobertura horizontal, ou seja, uma arquitetura que seja flexível a ponto de se adaptar aos diversos tipos de aparelhos que existem e aos que surgirão futuramente.

3.2 Interfaces para o Ambiente Móvel

A grande revolução provocada pela computação móvel e a necessidade de satisfação do usuário têm aumentado a dificuldade e a complexidade do desenvolvimento da interface homem-máquina, principalmente para computadores de mão. Um *design* nunca está completamente terminado, pois dispositivos móveis encontram-se em grande expansão, sendo lançados no mercado novos modelos constantemente.

Nesse contexto, aumenta a necessidade de estudo e implementação de interfaces inteligentes com o objetivo de adaptar-se às necessidades e preferências dos usuários. Um modelo é uma representação explícita das propriedades de usuários individuais ou de grupos de usuários, que permite ao sistema adaptar diversos aspectos de seu funcionamento às necessidades individuais. Alguns autores diferenciam interface adaptativa, interface adaptável e interface inteligente, como descrevem as seções a seguir.

3.2.1 Interface inteligente

Existem várias definições para interface inteligente, dentre elas: interface que entenda os objetivos e metas do usuário e saiba atingí-los ou que facilite uma interação mais natural, com uma maior tolerância a erros e com formatos mais agradáveis; e interface que se ajuste ao nível de conhecimento do usuário, conforme Harrington (1996). Outra definição, segundo Mctear (2000), é a que promove inferências de objetivos e planos do usuário, a fim de auto adaptar-se e fornecer aconselhamento, mantendo informações sobre o usuário num banco de dados de modelos de usuário.

Para Lieberman (1999), o que torna uma interface inteligente é essa poder se adaptar às necessidades de diferentes usuários; poder aprender novos conceitos e técnicas; poder antecipar as necessidades do usuário; poder tomar iniciativas e oferecer sugestões para o usuário e poder fornecer explicações de suas ações. Devem ser utilizadas quando existe uma grande distância semântica entre a linguagem dos usuários e a linguagem de máquina, o que poderá suficientemente complicar as tarefas do usuário.

Uma interface inteligente tenta reconhecer os planos do usuário e, então, usa esses planos para decidir como ajudar o usuário a atingir seus objetivos. O levantamento das características dos usuários deve ser o ponto de partida de todo projeto de interface. As determinações de quais informações sobre o usuário são necessárias à interface depende do objetivo do projetista e devem, conseqüentemente, ser agregadas ao modelo, permitindo assim que se pense em uma divisão da comunidade de usuários em grupos caracterizados por essas informações. Criam-se, assim, perfis de usuário. Para cada perfil, o projetista deve prever o comportamento adequado da interface.

Uma interface não pode atender ao mesmo tempo a todos os seus usuários em potencial. Para que ela não tenha efeitos negativos sobre o usuário, deve, conforme o contexto, adaptar-se a ele. Por outro lado, quanto mais variadas são as maneiras de realizar uma tarefa, maiores são as chances que o usuário possui de escolher e dominar uma delas no curso de seu aprendizado. Deve-se, portanto fornecer ao usuário procedimentos, opções, comandos diferentes, permitindo-lhe alcançar um mesmo objetivo.

Para uma interface ser considerada inteligente deve possuir um ou mais dos seguintes componentes, de acordo com Brusilovsky et. al (2001):

- **Modelo do Usuário:** é uma compilação de informações que descreve o usuário, e que é usada para determinar como apresentar os dados, que tipo de ajuda dar, e como o usuário irá interagir com a interface. É um dos componentes mais importantes das interfaces inteligentes;

- **Ajuda Inteligente:** apresenta ao usuário a ajuda que ele precisa para um tempo particular, ou numa situação particular. O sistema reconhece o erro e propicia uma solução para ele;

- **Adaptabilidade da Interface:** usuários podem configurar suas interfaces. Também o sistema pode se auto-adaptar para melhor interagir com o usuário, sem que ele tenha que definir a ação. Interfaces adaptáveis podem também

determinar que tipo de interface apresentar para o usuário, dependendo da análise do modelo do usuário;

- **Comunicação Multimodal:** o uso de vários meios de comunicação com uma interface é chamado comunicação multimodal;

- **Reconhecimento dos Planos:** é usado para deduzir o que o usuário planeja fazer. Esse reconhecimento torna o sistema inteligente. Nesse reconhecimento o modelo do usuário e as suas ações são considerados;

- **Apresentação Dinâmica:** a forma como o sistema decide mostrar os dados é determinado pelo exame do modelo do usuário.

Ao adaptarem-se às características do usuário, essas interfaces poderão minimizar o treinamento bem como melhorar a satisfação do usuário e a sua produtividade. Entretanto, não há consenso comum sobre modelos de interfaces adaptáveis. Nem há uma classificação dos tipos de adaptações que a interface deve empreender, nem estudo definitivo sobre o impacto dessas adaptações no desempenho dos usuários e no que eles aprendem, de acordo com Muller (2002).

3.2.2 Interface adaptativa

As interfaces adaptativas se apresentam promissoras na tentativa de superar os problemas atuais de complexidade na interação homem-computador. As aplicações tem se tornado cada vez mais complexas, levando o usuário a tratar uma grande quantidade de informações simultaneamente. Para melhorar essa interação, são necessárias interfaces que sejam capazes de se adaptar às necessidades do usuário.

Para que a interface seja considerada adaptativa, é necessário um modelo do usuário, no qual o sistema analisa as ações e perfis do usuário e adaptando-se automaticamente a ele.

Utilizando-se interfaces adaptativas, o sistema pode ser personalizado para estilos cognitivos individuais, necessidades de informações e tarefas personalizadas. As diferenças de cada usuário que podem ser controladas, pelo projeto da interface, são: a personalidade, o estilo cognitivo, o estilo de aprendizagem e a experiência. Quando um sistema muda automaticamente em resposta para sua experiência com usuários, isso é conhecido como interface auto-adaptativa.

3.2.3 Interface Adaptável

A interface é considerada adaptável quando realiza as adaptações unicamente no momento em que o usuário a requisita, ou seja, o usuário adapta o sistema a seu modo. Adaptar o usuário ao sistema significa oferecer-lhe treinamento, documentação, tutores, facilidades de ajuda, entre outras, enquanto o sistema permanece fixo. Esse enfoque apresenta a desvantagem de exigir do usuário dedicação de tempo para aprender a usar o sistema, tempo esse que não é utilizado em atividade produtiva. Sistemas adaptáveis permitem ao usuário adaptar seu próprio ambiente às suas preferências.

3.3 A Complexidade das Interfaces para Dispositivos Móveis

Os sistemas para ambientes móveis são mais complexos do que os sistemas para ambientes centralizados, pois as prioridades do usuário podem ser imprevisíveis e o contexto no qual a aplicação pode ser utilizada está em constante mudança. É importante considerar que os usuários geralmente buscam respostas rápidas.

Assim, o conteúdo deve levar em consideração tanto a impaciência e a atenção limitada dos usuários quanto sua alta mobilidade durante a utilização do dispositivo móvel. Aplicações dessa natureza apresentam o *design* de interfaces mais complexas, que devem atentar para os seguintes detalhes:

- **Tecnologia usada**: refere-se a capacidade de rede e detalhes da interface;

- **Perfil de usuários**: refere-se a idade, origem cultural, experiência de uso de dispositivos com características similares;

- **Contexto de uso**: refere-se a quantidade de usuários do dispositivo, serviços disponíveis, fatores sócio-econômicos;

Para os pesquisadores Mennecke e Strader (2003), a largura de banda restringe a quantidade de material que pode ser enviado pela rede sem fio, do servidor ou de outro dispositivo. A utilização de textos se torna eficiente em termos de transmissão de dados e pode ser utilizada em qualquer dispositivo móvel, porém a navegação pode se tornar mais difícil e lenta. Os gráficos oferecem informações mais concisas, mas aumentam os custos de transmissão, sendo mais limitados pelo tamanho das telas.

Pelas restrições apresentadas pelos dispositivos móveis, os desafios em termos de *design* requerem habilidades dos projetistas, diferentemente da interface tradicional para computadores pessoais. Funções interativas devem se apresentar de maneira intuitiva para o usuário. Adicionalmente, a criação de modelos de PDAs tem sido influenciada pelo ambiente de negócios, o qual é caracterizado pela constante pressão e necessidade de mercado por modelos inovadores, assim como significativo avanço tecnológico. Nesse contexto, tais dispositivos pressupõem a participação efetiva de um *designer*, pois, do contrário, a não participação desse profissional pode incorrer em problemas como aplicações não intuitivas, falta de padronização, navegação confusa entre telas e uso desordenado de cores e gráfico.

Alguns pesquisadores têm formulado alternativas de métodos de interação com o usuário que viabilizam sua constante mobilidade. Kristoffersen e Ljungberg (1999) desenvolveram o método MOTILE o qual requer atenção reduzida do usuário utilizando o som como forma de resposta às suas solicitações. A entrada de dados é realizada por botões e comandos estruturados no computador de mão.

Pascoe et. al (2000) formularam e discutiram dois princípios genéricos para interface e *design* móvel. O primeiro é intitulado como *Minimal Attention User Interfaces* (MAUI),

o qual procura minimizar a atenção do usuário. O segundo princípio está relacionado ao contexto, o qual o dispositivo móvel auxilia o usuário com base no conhecimento do ambiente em que se encontra. As próximas seções descrevem uma pesquisa sobre como direcionar o desenvolvimento de interfaces adaptativas.

3.4 Tipos de Interfaces para o Ambiente Móvel

Hassanein e Head (2003) afirmam que a interface para o ambiente móvel pode ser classificada em diversos tipos. Segundo os autores, existem cincos principais, quais são: Menu hierárquico, códigos curtos, baseada em árvore, baseada em tabelas e baseada em voz, como mostra a Figura 3.2.

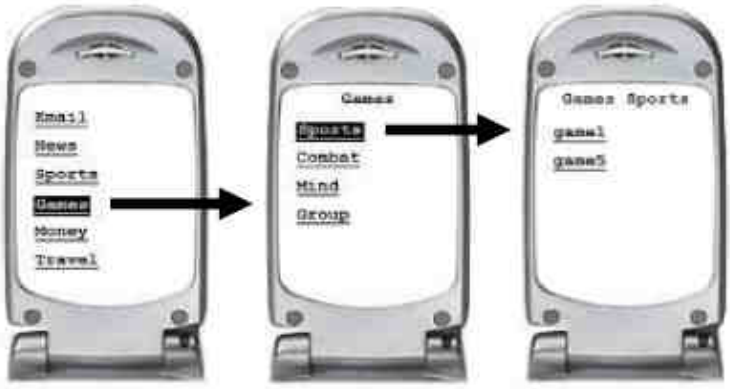
- **Menu Hierárquico:** as opções de menu são apresentadas por meio de uma série de itens, que após serem acionados, surgem como subitens. A navegação continua até que o usuário encontre a opção desejada;

- **Códigos Curtos:** o usuário escolhe as opções de menu por códigos numéricos, as quais são associadas a letras do alfabeto. A Figura 3.2, mostra um exemplo, caso o usuário queira acessar “Game Sport”, pressiona 4 para G, 2 para A e assim por diante;

- **Baseada em Árvore:** a árvore é composta por nós e folhas. Uma vez escolhida uma opção de um nó, as folhas são expandidas proporcionando a escolha de sub-opções;

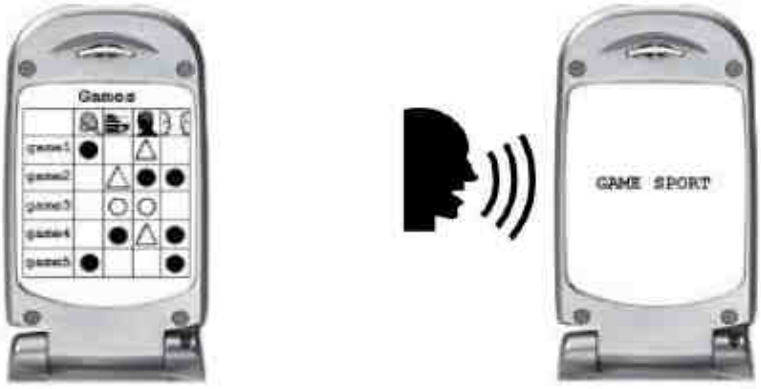
- **Baseada em Tabela:** as opções podem ser representadas por ícones ou símbolos dentro de tabelas, proporcionando assim a representação de uma quantidade maior de informações em telas pequenas;

- **Baseada em Discurso:** usuários de dispositivos móveis podem verbalmente selecionar itens de um menu ou um comando de entrada para ativar diretamente uma função ou recuperar a informação desejada;



Menu Hierárquico

Códigos Curtos Baseado em Árvore



Baseado em Tabela Baseado em Discurso

FIGURA 3.2 - Tipos de Interface para Dispositivos Móveis
Fonte: Adaptada de Hassanein e Head (2003)

3.5 Modelos de Interface Adaptável

A *User Interface Management Systems* (UIMS) é definida como o primeiro passo para o desenvolvimento de interfaces, permitindo ao desenvolvedor um alto nível de abstração, facilitando a rápida prototipação e desenvolvimento. São definidos três níveis para o processo de *design* da interface do usuário, segundo Foley (1990).

A Figura 3.3 apresenta os três níveis para o *design* da interface:

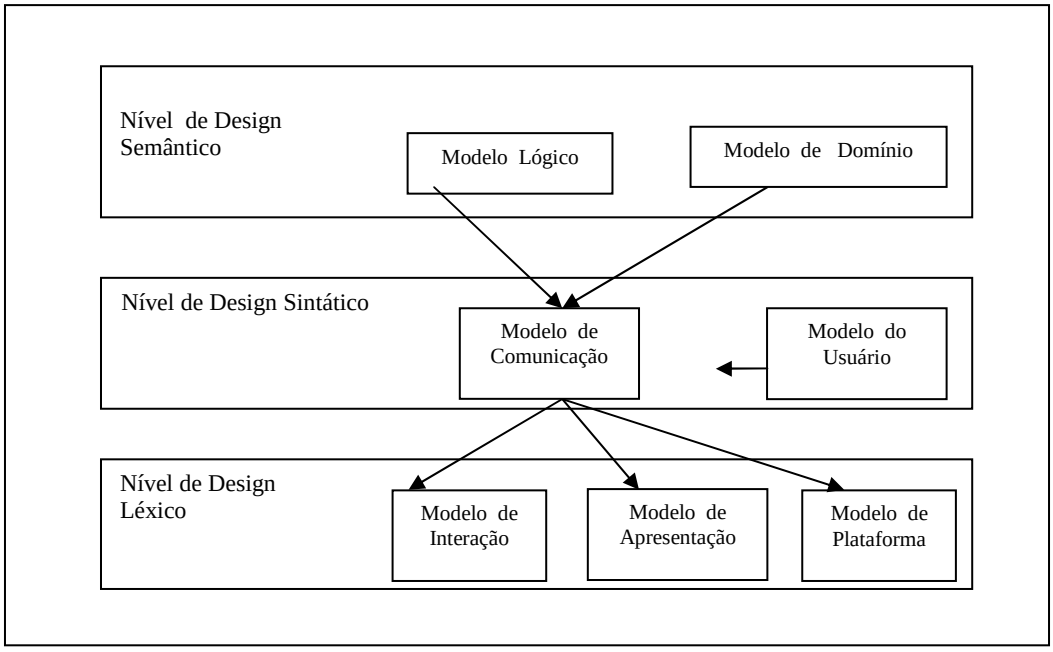


FIGURA 3.3 - Níveis para o Processo de Design da interface
Fonte: Adaptada de Foley (1990)

3.5.1 Nível de Design Semântico

O nível semântico é dividido em duas partes: modelo das atividades e modelo do domínio. O primeiro descreve as atividades que o usuário pode realizar, bem como a aplicação lógica do sistema e o segundo define entidades, atributos, métodos e relacionamentos.

3.5.2 Nível de Design Sintático

O nível de *design* sintático descreve a estrutura e os procedimentos de interação da interface do usuário. É composta por modelo de comunicação e modelo do usuário. No primeiro, é descrito a seqüência sintática da interação entre homem-máquina, sem especificar a representação visual da interface. Sua estrutura descreve o momento em que o usuário poderá realizar solicitações, invocar comandos e quando o computador poderá consultá-lo e apresentar o domínio dos dados.

O perfil do usuário descreve os tipos de usuários e suas principais características. Essa camada tem como objetivo coordenar a geração da interface de acordo com as propriedades do perfil do usuário, podendo esse ser classificado em diversas categorias.

3.5.3 Nível de Design Léxico

O nível de *design* léxico consiste na descrição detalhada da parte visual da interface. É composta por três partes: modelos de interação, plataforma e apresentação. O modelo de interação tem a função de integrar as camadas de apresentação e comunicação, bem como a aplicação lógica. A camada da plataforma contém as características do ambiente em que será executada a interface do usuário, incluindo restrições físicas.

O modelo de apresentação é composto pela parte visual da interface e suas características, como *layout*, organização, fontes, cores, entre outras. Nas linguagens como HTML e XML as camadas de comunicação e apresentação são agrupadas.

Existem diversas alternativas para a implementação da interface do usuário. A idéia é a introdução de elementos da interface genéricos e independentes de plataforma, que podem ser implementados em linguagens como HTML ou WML ou também em linguagens de programação apropriadas.

3.6 Adaptação Estática da Interface do Usuário

A adaptação estática é realizada durante o desenvolvimento do *software.* Nessa etapa o conceito de adaptabilidade está relacionado à manutenção, à modificação e à expansão do sistema.

3.7 Adaptação Dinâmica da Interface do Usuário

A adaptação dinâmica é realizada durante o tempo de execução do *software*. É considerada mais flexível, permitindo ao desenvolvedor a modificação de código em tempo de execução. De acordo com Menkhaus (2002) esse tipo de adaptabilidade pode ser originado de diferentes aspectos, quais são:

- **Mudanças no Perfil do Usuário:** requer uma adaptação da interface do usuário de acordo com seus objetivos, partindo do pressuposto da forma de utilização da aplicação;
- **Mudanças de Funcionalidade:** requer mudanças da interface desde os formulários aos componentes de interação para um conjunto de funcionalidades específicas;
- **Mudanças de Contexto do Sistema:** refere-se a mudanças relacionadas a aspectos tecnológicos do ambiente onde será instalado o *software*;

A adaptação da interface do usuário durante a execução do *software* pode ter diferentes dimensões, segundo Menkhaus (2002). O primeiro fator se refere às funcionalidades de cada interface como, por exemplo, *layout* ou modelo de comunicação e segundo é o tempo no qual ela é adaptada.

Existem quatro modos diferentes de adaptação da interface, os quais ocorrem em tempos também diferentes. São elas: adaptação por seleção, manipulação, colaboração e modificação. A seguir são descritas as principais características inerentes a cada modelo.

3.7.1 Adaptação por Seleção

A adaptação por seleção consiste na seleção de um modelo da interface, como mostra a Figura 3.4 (a). Pode ocorrer de forma global ou local.

Adaptação Global: O desenvolvedor deve conhecer as características de cada dispositivo móvel que será utilizado, pois nesse modelo, deve-se criar diversas interfaces, uma para cada aparelho. Durante o processo de desenvolvimento é feito um conjunto de alternativas de modelos de interface. Um mapeamento entre o perfil do dispositivo móvel e o modelo de apresentação é estabelecido, e a escolha é feita em tempo de execução, de acordo com o modelo mais apropriado para cada aparelho.

Adaptação local: Essa técnica é processada localmente nos dispositivos móveis que possuem características semelhantes (mas não em sua totalidade), sem analisar a estrutura do modelo de apresentação global. Pode ser considerada uma adaptação de código, consistindo na seleção de um conjunto pré-definido e semanticamente equivalente de código.

3.7.2 Adaptação por Manipulação

Nessa forma de adaptação, o usuário manipula apenas uma única camada de apresentação ou comunicação, a qual é usada para promover representação visual de acordo com suas preferências ou com algum ambiente ou dispositivo específico, como mostra a Figura 3.4 (b).

3.7.3 Adaptação por Colaboração

Nessa forma de adaptação, o sistema analisa outros sistemas no mesmo ambiente e tenta formar uma relação de parceria para encontrar a melhor forma de representação para sua interface. O dispositivo entra em um processo de cooperação com outros aparelhos e se une para realizar uma tarefa específica, como ilustra a Figura 3.4 (c).

3.7.4 Adaptação por Modificação

Essa forma de adaptação é mais flexível, pois pelo modelo de comunicação é possível realizar uma mudança da interface do usuário em tempo de execução, para a adaptação em um aparelho específico. Isto significa que não existe um modelo de apresentação para um ambiente, mas genéricos que cobrem uma grande quantidade de aparelhos. Isso faz com que adaptação por modificação seja mais complexa para implementar, como mostra a Figura 3.4 (d).

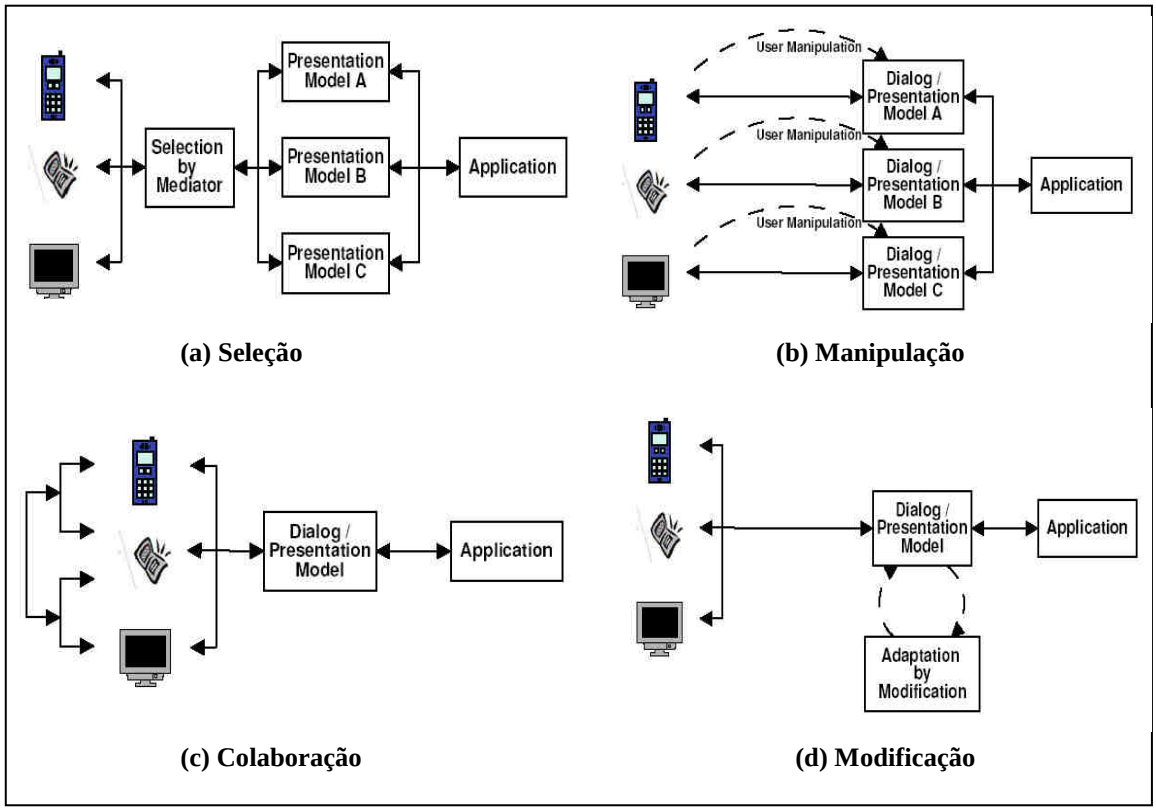


FIGURA 3.4 - Formas de Adaptação Dinâmica
Fonte: Menkhaus (2002)

3.8 Identificação de Dispositivos Móveis

Um aspecto importante com relação a uma arquitetura adaptável na computação móvel são aplicações cientes de contexto, que devem ser capazes de adquirir informações de contexto do usuário de modo automatizado, disponibilizando-as em um ambiente computacional em tempo de execução.

Os desenvolvedores desse tipo de aplicação têm a tarefa de decidir se as informações obtidas são realmente relevantes e como manipulá-las, de acordo com Dey e Abowd (1999). Para os autores, contexto é qualquer informação relevante que possa ser utilizada para caracterizar a situação de uma entidade. Uma entidade pode ser uma pessoa, um lugar ou um objeto, relevantes para a interação entre o usuário e a aplicação. Existem vários tipos de parâmetros para categorizar um contexto, como, por exemplo, tempo, conectividade, comunicação, segurança, tipo de dispositivo ou preferências do usuário.

Hanumansetty (2004) define parâmetros contextuais em duas categorias: estáticos e dinâmicos, que podem ser classificados de acordo com a natureza da aplicação. Como parâmetros estáticos podem-se incluir a plataforma, as regras e as preferências do usuário, que geralmente podem ser gerenciados durante a fase de *design* da interface, enquanto os parâmetros dinâmicos requerem suporte de adaptação em tempo de execução da geração da interface. Como exemplo pode-se citar a localização do usuário, largura de banda da rede de comunicação e o tempo.

O contexto dos dados pode ser de três tipos, dependendo da origem. São eles: contexto do cliente, sensorial e de sistema. O primeiro indica o contexto que é informado pelo usuário para o servidor, indicando o comportamento do usuário, suas preferências, capacidade e plataforma do aparelho e regras associadas. O segundo refere-se à localização, condições ambientes como som, luz e atividades relacionados ao usuário. O contexto de sistema inclui rede de comunicação e tempo.

Para a representação dos dois primeiros contextos, podem-se utilizar formas de identificação do cliente as quais permitem que equipamentos portáteis, se comuniquem com servidores Web e troquem informações.

De acordo com Butler (2005) existem três locais diferentes onde adaptação pode ocorrer: servidor, *proxy* e cliente. Quando esse processo ocorre exclusivamente no cliente pode-se ter algumas desvantagens como, por exemplo, a consistência dos dados,

o consumo desnecessário de banda, a utilização de espaço excessivo no cliente, além do baixo poder de processamento do dispositivo.

3.9 Identificação do Cliente Móvel pelo User-Agent

Uma das formas para a identificação do perfil do cliente que acessa uma aplicação pode ocorrer por meio do *user-agent*, um componente do cabeçalho do protocolo HTTP que realiza a comunicação entre o cliente e servidor, por meio de mensagens.

Uma mensagem de requisição pode conter dados que serão transmitidos diretamente pelo usuário ou um arquivo que será enviado para o servidor. Em uma resposta, o corpo é o recurso que foi requisitado pelo cliente, ou ainda um aviso de erro, caso esse recurso não seja possível.

Uma mensagem é composta por uma linha inicial, nenhuma ou mais linhas de cabeçalhos, uma linha em branco obrigatória e o corpo podendo ser opcional em determinados casos, como mostra a Figura 3.5.

```
1 GET /talesbv/index.html HTTP/1.0
2 User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
3 Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
     application/vnd.ms-powerpoint, application/vnd.ms-excel,
     application/msword, */*
4 Accept-Language: pt-br
5 Host: gravatai.ulbra.tche.br
6 Proxy-Connection: Keep-Alive
```

FIGURA 3.5 - Cabeçalho HTTP

O protocolo no nível da aplicação para a transferência de hipertexto HTTP opera sobre o TCP/IP (*Transmission Control Protocol/ Internet Protocol*) para estabelecer um mecanismo de serviço com estrutura requisição-resposta. Uma de suas características é a composição flexível do cabeçalho, composto por diversas linhas o que permite sua utilização como integrador de diversos formatos e não apenas de documentos HTML.

Para a identificação do perfil do cliente, as informações podem ser recuperadas por meio do *user-agent*, pois o cliente Web identifica-se quando envia uma requisição ao servidor. Pode-se visualizá-lo diretamente no *browser*, digitando-se javascript:alert(navigator.userAgent) na barra de endereço, como mostra a Figura 3.6.

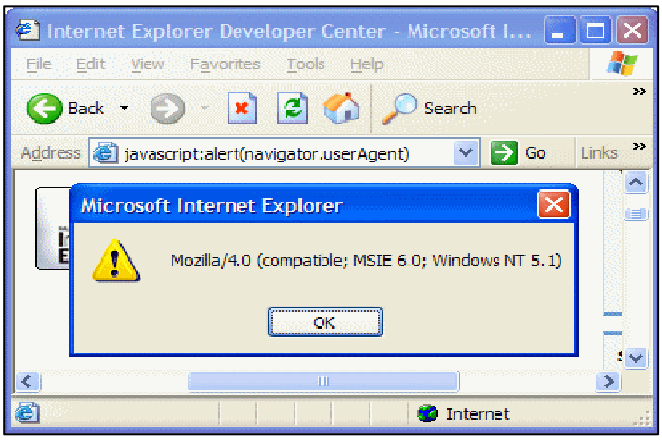


FIGURA 3.6 - *User Agent* do Internet Explorer

Antigamente, os desenvolvedores utilizavam o *User Agent* para determinar o envio de *frames* para o *browser*. Isto acontecia apenas para navegadores com a identificação Mozilla, pois foi o primeiro a permitir esse recurso. Conseqüentemente, a Microsoft fez com que o Internet *Explorer* fosse identificado como Mozilla, pois foi a única maneira de exibir páginas Web com *frames*.

O *user-agent* fornece apenas as informações, tais como a versão do Mozzila, compatibilidade, versão do *token* e o sistema operacional, detalhadas a seguir e ilustradas na Figura 3.7.

- A compatibilidade (*compatible*) é usado pelos *browser*s para indicar se é compatível com um conjunto comum de características;
- A versão do *token* identifica o *browser* e o número da versão. Por exemplo, (MSIE 6.0) identifica o Windows Internet Explorer 6;
- A plataforma (Windows NT 5.1) identifica o sistema operacional;

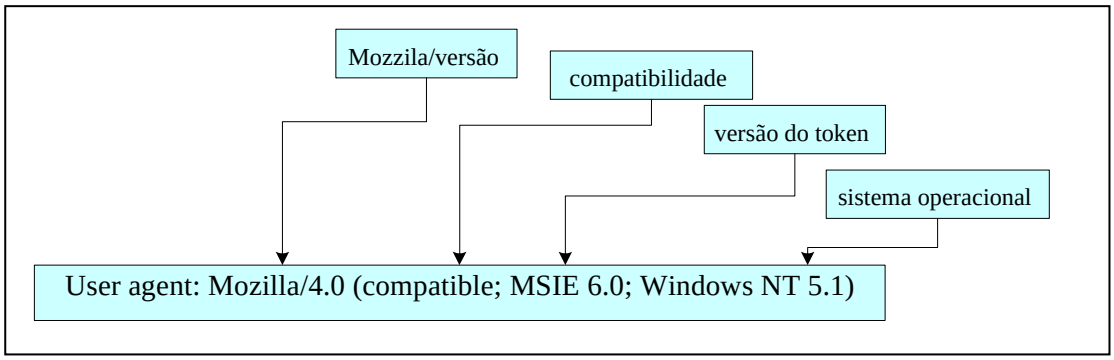


FIGURA 3.7 - Características Capturadas pelo *User Agent*

As tentativas de adaptação por meio de *proxies* e junto ao servidor HTTP são as mais comuns. Como alternativa para garantir que as características dos mais variados tipos de dispositivos sejam capturadas, existem quatro esquemas propostos para especificação das potencialidades dos aparelhos: o *composite capability/preferences profile* (CC/PP), o WAP *User Agent Profile* (UAPROF), o SyncML *Device Information* (DevInf) e o *Universal Plug and Play Standard* (UPnP).

A arquitetura GIA apresentada no capítulo 5 utiliza o CC/PP e o UAPROF, como descrevem as próximas seções.

3.10 Composite Capability / Preferences Profile (CC/PP)

O CC/PP é uma descrição das potencialidades do dispositivo e das preferências do usuário de acordo com Klyne et al. (2005). Tem como objetivo proporcionar um mecanismo estruturado e universal para descrever e transmitir informações sobre as capacidades de um cliente Web para um servidor, de forma que o conteúdo seja direcionado a essas características. O perfil CC/PP é construído em uma hierarquia de dois níveis: o primeiro é chamado de componentes de um perfil e o segundo de atributos de um perfil, como apresenta a Figura 3.8.

Um componente é composto de pelo menos um atributo, podendo ser de *hardware*, de *software* ou de *browser*. Como exemplo de *hardware* pode-se citar tamanho da tela,

processador, memória, entre outras. De *software*, a versão do sistema operacional e de *browser* o nome, versão, fabricante, entre outras.


```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ccpp="http://www.w3.org/2002/11/08-ccpp-schema#"
         xmlns:ex="http://www.example.com/schema#">

  <rdf:Description
      rdf:about="http://www.example.com/profile#MyProfile">

    <ccpp:component>
      <rdf:Description
          rdf:about="http://www.example.com/profile#TerminalHardware">
        <rdf:type
            rdf:resource="http://www.example.com/schema#HardwarePlatform" />
        <ex:displayWidth>320</ex:displayWidth>
        <ex:displayHeight>200</ex:displayHeight>
      </rdf:Description>
    </ccpp:component>

    <ccpp:component>
      <rdf:Description
          rdf:about="http://www.example.com/profile#TerminalSoftware">
        <rdf:type
            rdf:resource="http://www.example.com/schema#SoftwarePlatform" />
        <ex:name>EPOC</ex:name>
        <ex:version>2.0</ex:version>
        <ex:vendor>Symbian</ex:vendor>
      </rdf:Description>
    </ccpp:component>

    <ccpp:component>
      <rdf:Description
          rdf:about="http://www.example.com/profile#TerminalBrowser">
        <rdf:type
            rdf:resource="http://www.example.com/schema#BrowserUA" />
        <ex:name>Mozilla</ex:name>
        <ex:version>5.0</ex:version>
        <ex:vendor>Symbian</ex:vendor>
        <ex:htmlVersionsSupported>
          <rdf:Bag>
            <rdf:li>3.2</rdf:li>
            <rdf:li>4.0</rdf:li>
          </rdf:Bag>
        </ex:htmlVersionsSupported>
      </rdf:Description>
    </ccpp:component>

  </rdf:Description>
</rdf:RDF>
```

FIGURA 3.8- Estruturas CC/PP
Fonte: Adaptada de Klyne et al. (2004)

3.10.1 Resource Description Framework (RDF)

De acordo com Lassila et al. (2005), o *CC/PP* é apresentado em RDF (*Resource Description Framework*), uma linguagem para representação de informação na Web, utilizada pela W3C para modelagem de metadados. Sua função é criar um modelo

simples de dados, com uma semântica formal, permitindo o uso de XML, bem como proporcionar uma melhor interoperabilidade entre aplicações.

O RDF é implementado em XML, que segundo Bray et al. (2005) é a descrição de uma classe de objetos de dados chamada XML *documents* que descrevem parcialmente o comportamento dos programas que os processam.

3.10.2 *User Agent* Profile (UAProf)

O UAProf (WAP-UAProf) é uma especificação definida pelo grupo WAP Fórum, cujo objetivo é descrever as características de telefones celulares. Sua representação é baseada em RDF e, portanto, seu vocabulário utiliza o mesmo formato básico que o CC/PP.

No vocabulário UAPROF, cada propriedade do agente usuário pertence a um dos componentes que o formam, a saber: plataforma de *hardware*, *software*, características WAP, *browser* do agente usuário e características de rede. O formato do vocabulário UAPROF é apresentado na Figura 3.9.

```
<?xml version="1.0"?>
<vocabspec>
        <attribute>
                <name>CcppAccept</name>
                <component>SoftwarePlatform</component>
                <collectionType>Bag</collectionType>
                <attributeType>Literal</attributeType>
                <resolution>Append</resolution>
        </attribute>
</vocabspec>
```

FIGURA 3.9 - Vocabulário UAPROF

As questões em torno da adoção desse vocabulário estão relacionadas, primeiramente, ao fato de que é muito especializado para um tipo de dispositivo móvel. O UAProf só representa características de cliente em detrimento de características de servidores, da rede de acesso e preferências do usuário.

O documento WAP-UAProf define detalhadamente a estrutura desse esquema no que diz respeito à definição de classe e semântica de atributos para dispositivos voltados para padrão WAP. Apenas a *Uniform Resource Locator* (URL) do perfil do aparelho é transmitido do terminal móvel para o servidor. Isto é feito se adicionado um cabeçalho X-Wap-profile para o pedido HTTP, como apresenta a Figura 3.10, que indica onde o servidor encontrará o arquivo de perfil.

```
X-Wap-Profile: http://nds1.nds.nokia.com/uaprof/N6230r100.xml
```

FIGURA 3.10 - Cabeçalho X-Wap-profile

3.10.3 Delivery Context Library (DELI)

O *Delivery Context Library* (DELI) é uma biblioteca desenvolvida pela *Hewlett Packward* (HP), que permite a um *servlet* tratar requisições de dispositivos que possuam perfis CC/PP ou Uaprof.

Uma vez que o DELI estiver habilitado, automaticamente determinará o melhor conjunto de características do dispositivo solicitante e as disponibilizará para o sistema. De acordo com Butler (2005), os perfis podem estar armazenados em um servidor Web local ou externo, conforme descrevem as próximas seções.

3.10.4 Repositório de Perfis Armazenado Localmente

Um perfil é armazenado localmente quando se encontra em uma base de dados no próprio servidor. A utilização dessa arquitetura tem como principal vantagem a velocidade de resposta na requisição dos perfis CC/PP, porém o banco de dados local torna-se obsoleto muito rapidamente, tendo em vista que o surgimento de novos dispositivos cresce de forma acelerada. Para que o reconhecimento de dispositivos torne-se eficaz é necessário que a cada dispositivo lançado seu perfil seja adicionado no repositório local.

Um exemplo de um repositório local de perfis pode ser visto na Figura 3.11. Quando o cliente faz a conexão com o servidor, inicia-se uma busca no repositório local de perfis por meio do seu u*ser agent*, com o objetivo de encontrar o perfil CC/PP.

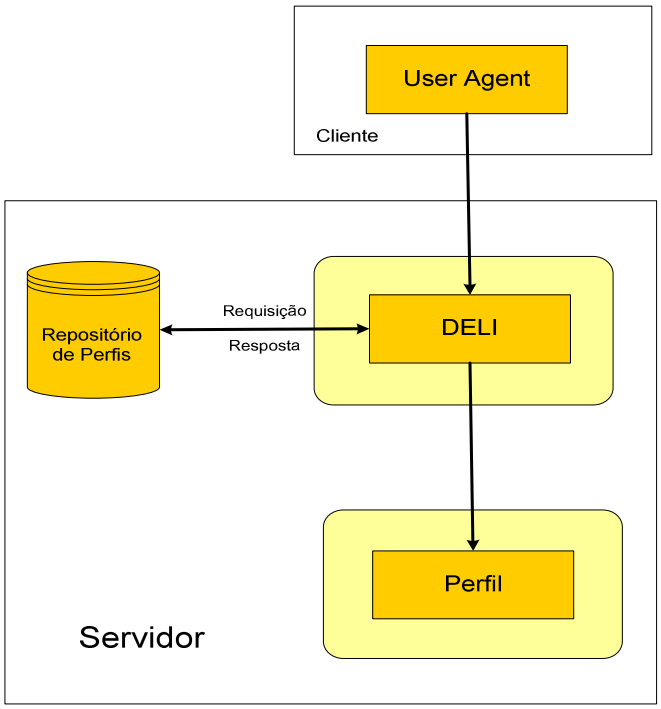


FIGURA 3.11 - Utilização de um Repositório Local de Perfis
Fonte: Adaptada de Hinz e Fiála (2004)

3.10.5 Repositório Externo de Perfis

Um repositório de perfis é considerado externo quando não se encontra no contexto da aplicação. As informações sobre os dispositivos são armazenadas em um Servidor Web e podem ser recuperadas por meio das *Uniform Resource Identifier* (URIs) que são disponibilizadas e atualizadas pelos fabricantes. Apesar de ser uma arquitetura mais lenta, ao contrário de um repositório de perfis armazenado localmente, ela proporciona uma atualização de perfis menos trabalhosa para o programador, pois a responsabilidade de informar novos tipos de aparelhos fica a cargo do fabricante.

A Figura 3.12 ilustra um repositório externo de perfis. Quando o cliente acessar a aplicação o servidor envia juntamente com a requisição, o cabeçalho do *user-agent* que contém a URI referente ao perfil do seu dispositivo.

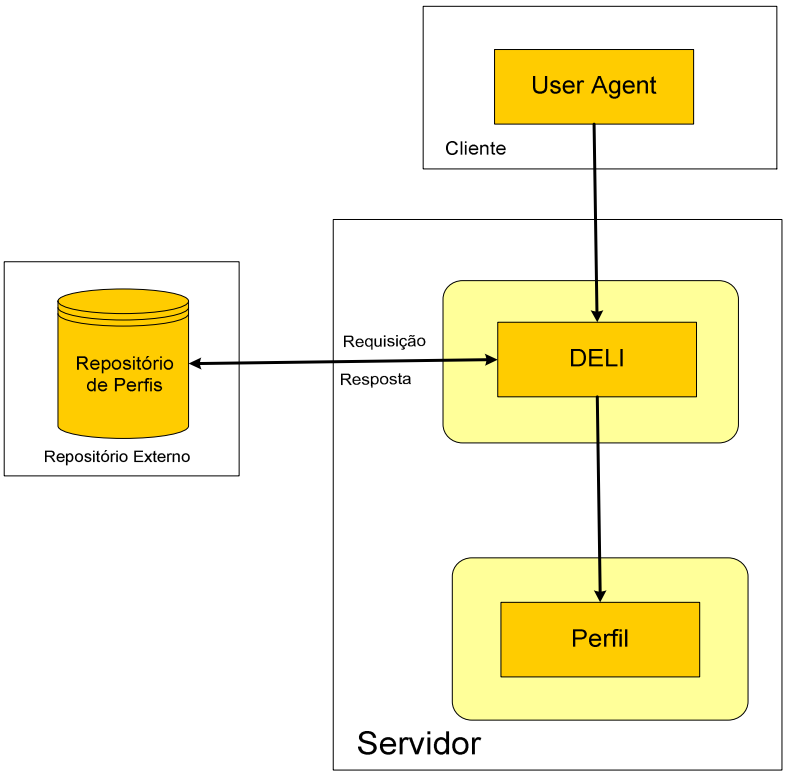


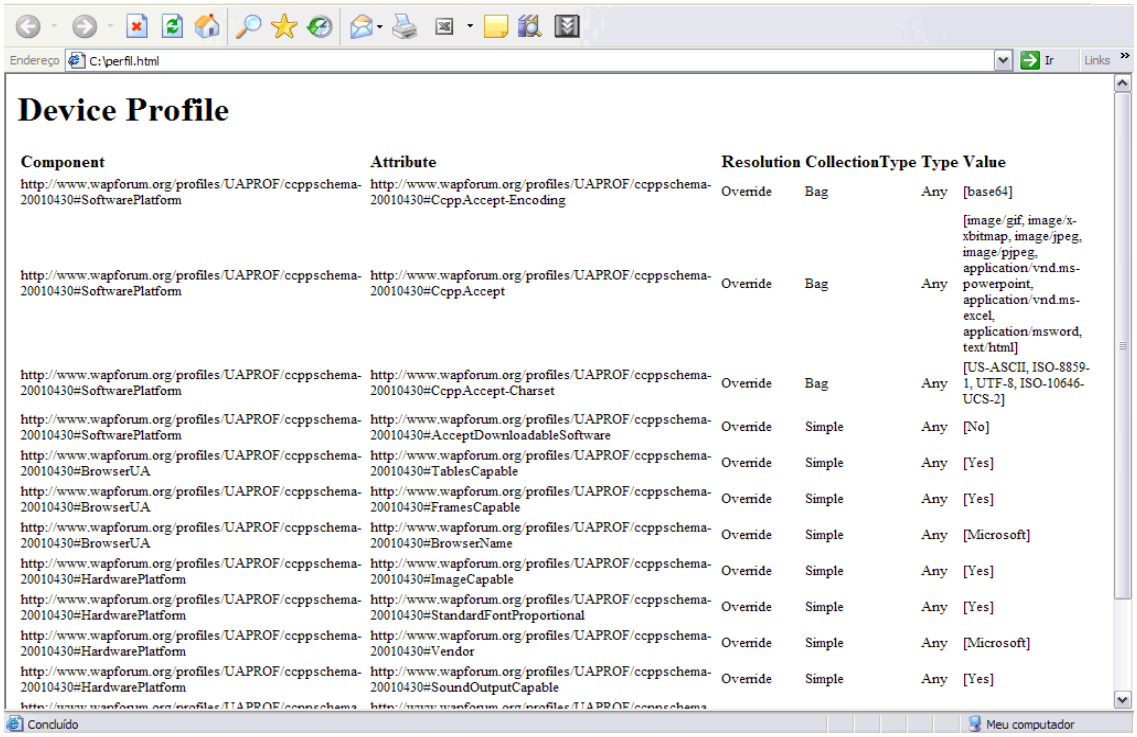
FIGURA 3.12 - Utilização de um repositório externo de perfis
Fonte: Adaptada de Hinz e Fiála (2004)

3.10.6 Perfis de Dispositivos Móveis

A fim de exemplificar características sobre os dispositivos móveis foram capturados dois perfis de simuladores: *Openwave Phone Simulator* e *PocketPC 2002*.

Visualizando-se a Figura 3.13, pode-se perceber que o perfil do simulador openwave contém os componentes, atributos e seus respectivos valores. O componente indicado pelo endereço <http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#HardwarePlatform> contém o atributo que indica o tamanho da tela do dispositivo, 120x160 *pixels*.

A Figura 3.14 representa o perfil do PocketPC, que informa, por exemplo, que o *browser* utilizado pelo cliente aceita *frames*.



Endereço C:\perfil.html

Ir Links

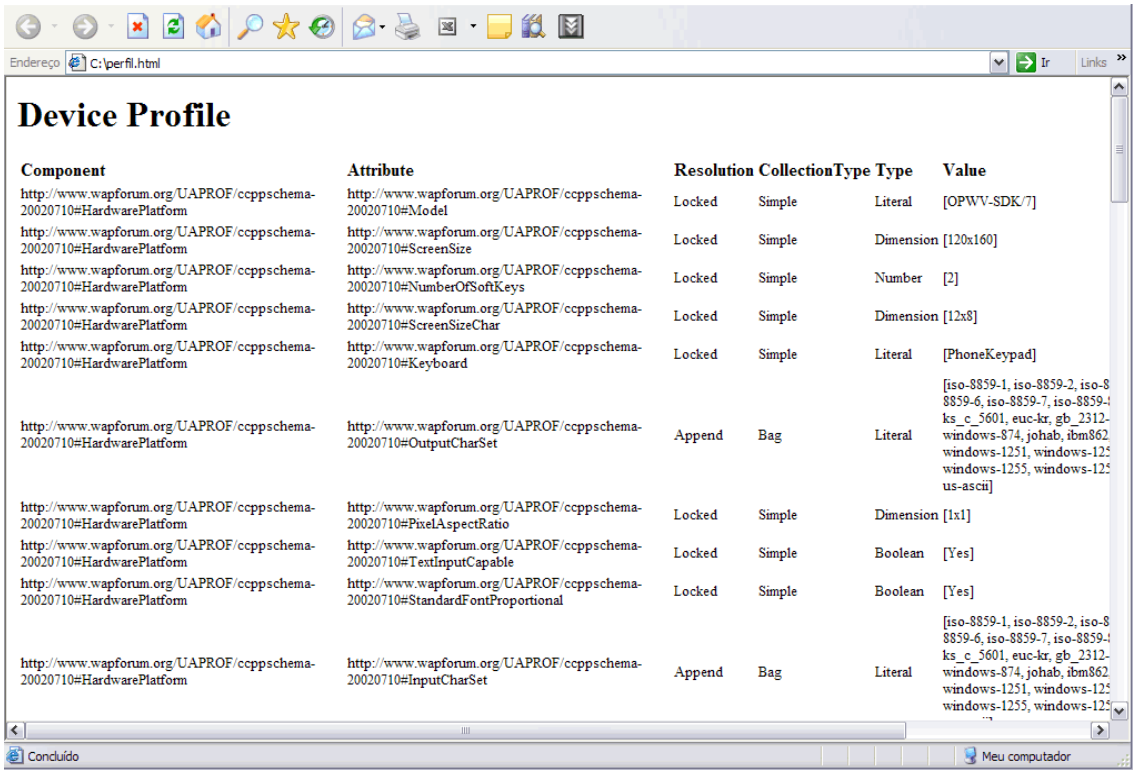
Device Profile

Component	Attribute	Resolution	CollectionType	Type	Value
http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#SoftwarePlatform	http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#CcppAccept-Encoding	Override	Bag	Any	[base64]
http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#SoftwarePlatform	http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#CcppAccept	Override	Bag	Any	[image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, text/html]
http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#SoftwarePlatform	http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#CcppAccept-Charset	Override	Bag	Any	[US-ASCII, ISO-8859-1, UTF-8, ISO-10646-UCS-2]
http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#SoftwarePlatform	http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#AcceptDownloadableSoftware	Override	Simple	Any	[No]
http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#BrowserUA	http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#TablesCapable	Override	Simple	Any	[Yes]
http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#BrowserUA	http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#FramesCapable	Override	Simple	Any	[Yes]
http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#BrowserUA	http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#BrowserName	Override	Simple	Any	[Microsoft]
http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#HardwarePlatform	http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#ImageCapable	Override	Simple	Any	[Yes]
http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#HardwarePlatform	http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#StandardFontProportional	Override	Simple	Any	[Yes]
http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#HardwarePlatform	http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#Vendor	Override	Simple	Any	[Microsoft]
http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#HardwarePlatform	http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#SoundOutputCapable	Override	Simple	Any	[Yes]

Concluído

Meu computador

FIGURA 3.14- Arquivo Gerado com as Características do Emulador de Celular



Endereço C:\perfil.html

Ir Links

Device Profile

Component	Attribute	Resolution	CollectionType	Type	Value
http://www.wapforum.org/UAPROF/ccppschema-20020710#HardwarePlatform	http://www.wapforum.org/UAPROF/ccppschema-20020710#Model	Locked	Simple	Literal	[OPWV-SDK/7]
http://www.wapforum.org/UAPROF/ccppschema-20020710#HardwarePlatform	http://www.wapforum.org/UAPROF/ccppschema-20020710#ScreenSize	Locked	Simple	Dimension	[120x160]
http://www.wapforum.org/UAPROF/ccppschema-20020710#HardwarePlatform	http://www.wapforum.org/UAPROF/ccppschema-20020710#NumberOfSoftKeys	Locked	Simple	Number	[2]
http://www.wapforum.org/UAPROF/ccppschema-20020710#HardwarePlatform	http://www.wapforum.org/UAPROF/ccppschema-20020710#ScreenSizeChar	Locked	Simple	Dimension	[12x8]
http://www.wapforum.org/UAPROF/ccppschema-20020710#HardwarePlatform	http://www.wapforum.org/UAPROF/ccppschema-20020710#Keyboard	Locked	Simple	Literal	[PhoneKeypad]
http://www.wapforum.org/UAPROF/ccppschema-20020710#HardwarePlatform	http://www.wapforum.org/UAPROF/ccppschema-20020710#OutputCharSet	Append	Bag	Literal	[iso-8859-1, iso-8859-2, iso-8 8859-6, iso-8859-7, iso-8859- ks_c_5601, euc-kr, gb_2312- windows-874, johab, ibm862 windows-1251, windows-125 windows-1255, windows-125 us-ascii]
http://www.wapforum.org/UAPROF/ccppschema-20020710#HardwarePlatform	http://www.wapforum.org/UAPROF/ccppschema-20020710#PixelAspectRatio	Locked	Simple	Dimension	[1x1]
http://www.wapforum.org/UAPROF/ccppschema-20020710#HardwarePlatform	http://www.wapforum.org/UAPROF/ccppschema-20020710#TextInputCapable	Locked	Simple	Boolean	[Yes]
http://www.wapforum.org/UAPROF/ccppschema-20020710#HardwarePlatform	http://www.wapforum.org/UAPROF/ccppschema-20020710#StandardFontProportional	Locked	Simple	Boolean	[Yes]
http://www.wapforum.org/UAPROF/ccppschema-20020710#HardwarePlatform	http://www.wapforum.org/UAPROF/ccppschema-20020710#InputCharSet	Append	Bag	Literal	[iso-8859-1, iso-8859-2, iso-8 8859-6, iso-8859-7, iso-8859- ks_c_5601, euc-kr, gb_2312- windows-874, johab, ibm862 windows-1251, windows-125 windows-1255, windows-125

Concluído

Meu computador

FIGURA 3.14- Arquivo Gerado com as Características do Emulador do PocketPC

Uma vez que o dispositivo móvel esteja conectado à Internet pode-se reconhecer o seu perfil diretamente pelo aparelho, como ilustra a Figura 3.15, contendo o perfil dos dispositivos Nokia6273, *Openwave Phone Simulator* e Nokia6230i. Isso é possível por meio do serviço de comunicação da arquitetura para a Geração de Interfaces Adaptativas (GIA) que é detalhado no capítulo 5.

FIGURA 3.15 - Perfis dos Simuladores Capturados com o DELI integrado a GIA

O próximo capítulo apresenta as principais arquiteturas relacionadas ao ambiente móvel, bem como suas características, vantagens e restrições. Os principais trabalhos estudados, MUSA, Dygimes, *Negotiation and Adaptation Core* (NAC), Arquitetura Reflexiva, *Mobile Application Servers* (MAS) e o modelo MVC, também são detalhados no próximo capítulo.

CAPÍTULO 4

TRABALHOS RELACIONADOS

Embora os projetos relacionados ao ambiente móvel já venham sendo explorados há algum tempo, muitos fatores ainda apresentam-se longe de oferecer qualidade de serviço aos usuários.

Um aspecto importante com relação à computação móvel é a utilização de uma arquitetura adaptável, por meio de aplicações cientes de contexto, que devem ser capazes de adquirir informações sobre o usuário de modo automatizado, disponibilizando-as em um ambiente computacional em tempo de execução.

Os principais trabalhos estudados, MUSA descrito por Menkhaus (2002), Dygimes por Coninx et al. (2003), *Negotiation and Adaptation Core* (NAC) proposto por Lemlouma e Layaida (2004), Arquitetura Reflexiva descrita por Sendin e Lores (2004), *Mobile Application Servers* (MAS) sugerido por Viana et al. (2005) e o modelo MVC, são detalhados nas próximas seções.

Neste capítulo também são descritas as principais arquiteturas relacionadas ao ambiente móvel, bem como suas características, vantagens e restrições. Analogamente, apresenta-se a arquitetura adaptativa GIA, que tem como principal objetivo atender a diversos usuários de forma dinâmica, abrangendo uma grande diversidade de dispositivos.

4.1 Arquiteturas para Renderização de Conteúdo Adaptativo

Prabhu (2003) comparou várias arquiteturas para renderização de conteúdo adaptativo, dentre elas as arquiteturas *Transcoding, Ad hoc,* convencional e baseada em dados. O autor apresenta limitações e vantagens na utilização de cada arquitetura e propôs uma arquitetura adaptativa, como apresentam as próximas seções.

4.1.1 Arquitetura *Transcoding*

A arquitetura *Transcoding* é uma das primeiras a abordar a questão de adaptação a múltiplos dispositivos empregando o conceito de *transcoding.* De acordo com Britton et. al (2001), *transcoding* que é o processo de filtragem, tradução ou conversão de um formato para outro. Como exemplos podem-se citar a conversão de formatos de vídeo, conversão de HTML para outras linguagens como WML, transformação de textos PDF para HTML, conversão de imagens como BMP para WBMP e GIF para JPEG.

Essa arquitetura tem o objetivo de adequar-se às capacidades de um dispositivo específico, como ilustra a Figura 4.1. A camada central atua como um *proxy* redirecionando solicitações e filtrando respostas. Ao acessar a aplicação o sistema processa o pedido e renderiza a resposta. O *Proxy* a interpreta e a transforma em um formato compatível com o dispositivo do cliente.

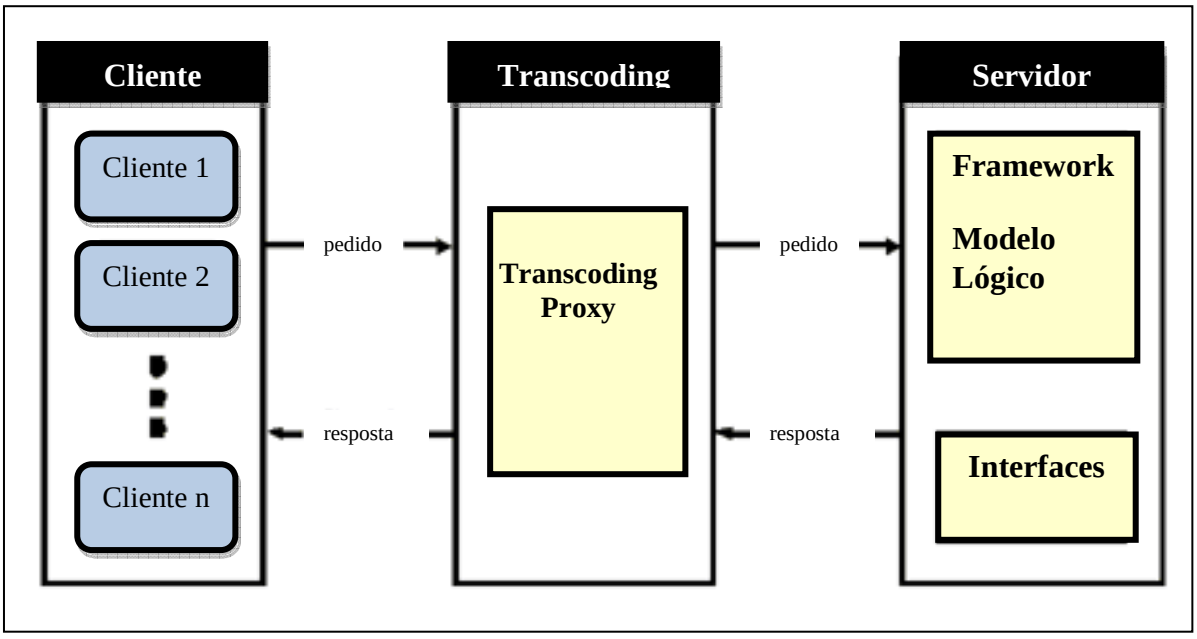


FIGURA 4.1- Arquitetura Transcoding
Fonte: Adaptada de Prabhu (2003)

As limitações que essa arquitetura apresenta referem-se a uma técnica indicada para atender a múltiplos dispositivos, desde que as mudanças no sistema não aconteçam com muita freqüência. Fatores como precisão da informação podem ser considerados

problemáticos nesta arquitetura. Quando um elemento de entrada, por exemplo, em HTML não encontra o seu correspondente em outro formato, no caso WML, os elementos não podem ser mapeados, acarretando em problemas na exibição da interface.

4.1.2 Arquitetura *Ad hoc*

A arquitetura *Ad hoc* apresenta diversos subsistemas de acordo com os dispositivos que acessarem a aplicação. Nesse modelo, o desenvolvedor deve implementar vários sistemas paralelos para atender as solicitações dos clientes, como ilustra a Figura 4.2.

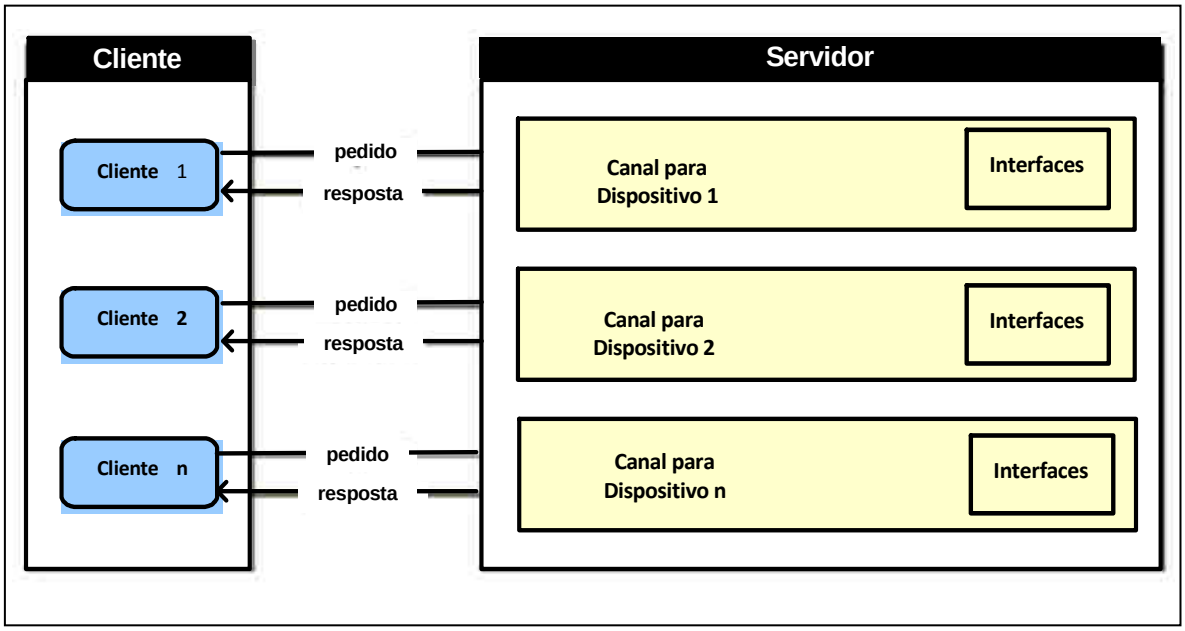


FIGURA 4.2 - Arquitetura Ad Hoc
Fonte: Adaptada de Prabhu (2003)

As limitações que a arquitetura *ad hoc* apresenta referem-se ao fato de que os subsistemas são independentes um do outro, causando, dessa forma, excesso de código duplicado no cliente e no servidor.

Esse tipo de arquitetura pode apresentar alto custo de manutenção, pois para cada aparelho deve-se criar um mecanismo de gerenciamento diferente. Com grande quantidade de código e funcionalidade duplicada no sistema, as falhas podem se tornar

constantes. Fatores como a expansão do sistema são problemáticos, uma vez que ao surgirem novos tipos de dispositivos o sistema deve ser implementado novamente.

4.1.3 Arquitetura Convencional

A arquitetura convencional utiliza a implementação baseada no modelo MVC e provê uma única entrada para o servidor para qualquer tipo de dispositivo. O sistema consiste de um módulo de detecção dos dispositivos que utiliza a informação obtida para renderizar o conteúdo apropriado para cada tipo de aparelho que solicitar uma aplicação Web.

As interfaces são mapeadas para arquivos JSP, sendo que cada arquivo é associado a um único aparelho, causando, dessa forma, duplicidade de conteúdo e múltiplos arquivos de interface. A arquitetura convencional é ilustrada na Figura 4.3:

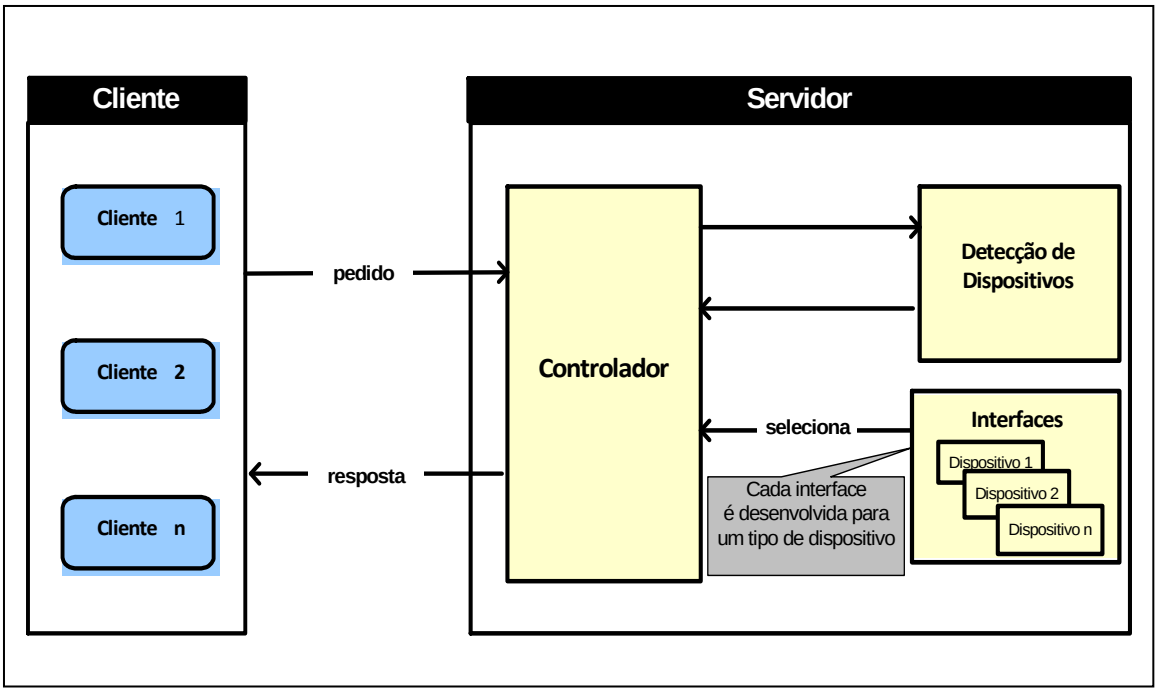


FIGURA 4.3 - Arquitetura Convencional
Fonte: Adaptada de Prabhu (2003)

4.1.4 Arquitetura Baseada em XML e XSL

Essa arquitetura utiliza a linguagem XML para descrever as interfaces do sistema. Nesse contexto, a XML desempenha um importante papel, visto que um documento XML também possui uma estrutura hierárquica e seu conteúdo é mantido separado do *layout* de apresentação.

A XSL, aplicada no contexto de adaptação de interfaces, funciona como uma descrição de como XML deverá aparecer para o cliente. Ela funciona em conjunto com uma linguagem de programação Web, como HTML ou WML. Assim tem-se a idéia de que uma descrição em XSL pode ser aplicada para adaptar uma interface em um dispositivo que tenha suporte a qualquer uma dessas linguagens, ou seja, uma XSL corresponde a um determinado dispositivo, sendo que um computador pessoal irá conter uma folha de estilos para formatar a página à sua resolução.

Dessa maneira, com o uso das folhas de estilo XSL ou da *Application Programming Interface* (API) *Document Object Model* (DOM) torna-se possível gerar diferentes interfaces para um mesmo conteúdo. A arquitetura correspondente pode ser visualizada conforme a Figura 4.4:

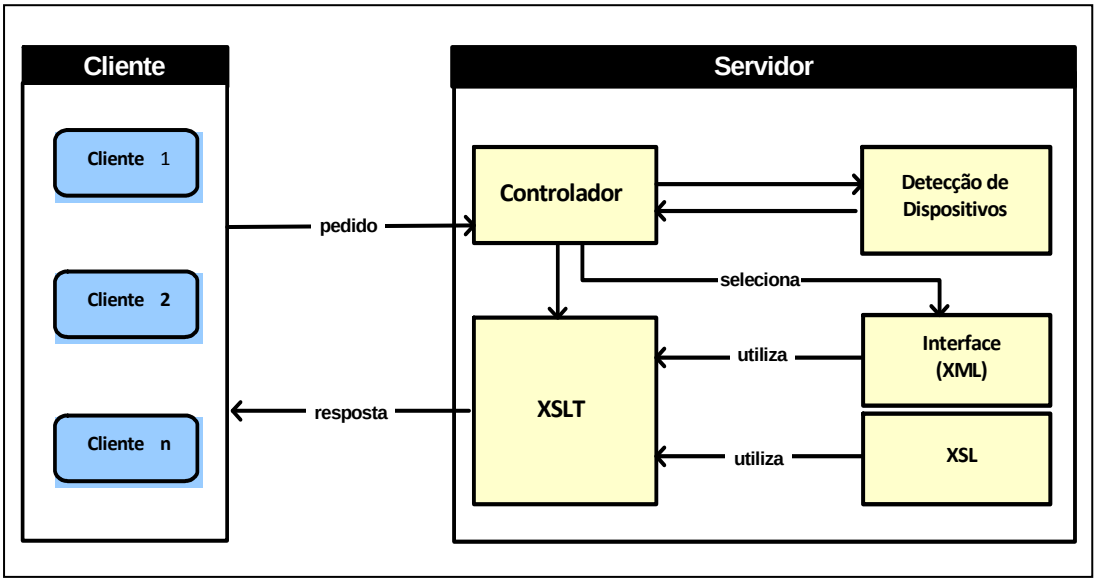


FIGURA 4.4 - Arquitetura Baseada em XML e XSL
Fonte: Adaptada de Prabhu (2003)

O funcionamento da arquitetura baseada em XML e XSL é descrito nos passos a seguir:

- Uma solicitação é feita pelo cliente;
- O modulo detecção de dispositivos identifica o tipo de aparelho;
- O modelo lógico é executado pelo controlador;
- O controlador seleciona a interface em formato XML e um arquivo XSL apropriado para ela;
- Uma transformação é realizada para renderizar uma resposta para o cliente;

As interfaces são complementadas por uma biblioteca de arquivos XSL, sendo implementadas de acordo com o número de dispositivos, pois cada aparelho deve ter uma XSL correspondente. Essa é umas das principais limitações dessa arquitetura, pois diversos arquivos das interfaces são gerados.

4.1.5 Arquitetura Adaptativa

A arquitetura adaptativa é uma extensão da convencional. A principal diferença está na forma como os arquivos são gerenciados. Cada interface corresponde a um único arquivo JSP e o sistema é responsável por localizar um conteúdo apropriado para renderizar o código desse arquivo.

Nesse contexto, *tags libraries* (capítulo 2), podem ser desenvolvidas para averiguar as propriedades do dispositivo e prover uma ligação com o modelo lógico.

A arquitetura adaptativa apresenta a vantagem de causar menos duplicidade de conteúdo, pois uma única interface será alterada, caso sejam feitas alterações no sistema. Uma restrição dessa arquitetura refere-se ao fato de que como todos os dispositivos compartilham uma mesma interface, mudanças para um dispositivo específico não poderão ser feitas de forma separada.

A arquitetura adaptativa pode ser visualizada conforme ilustra a Figura 4.5:

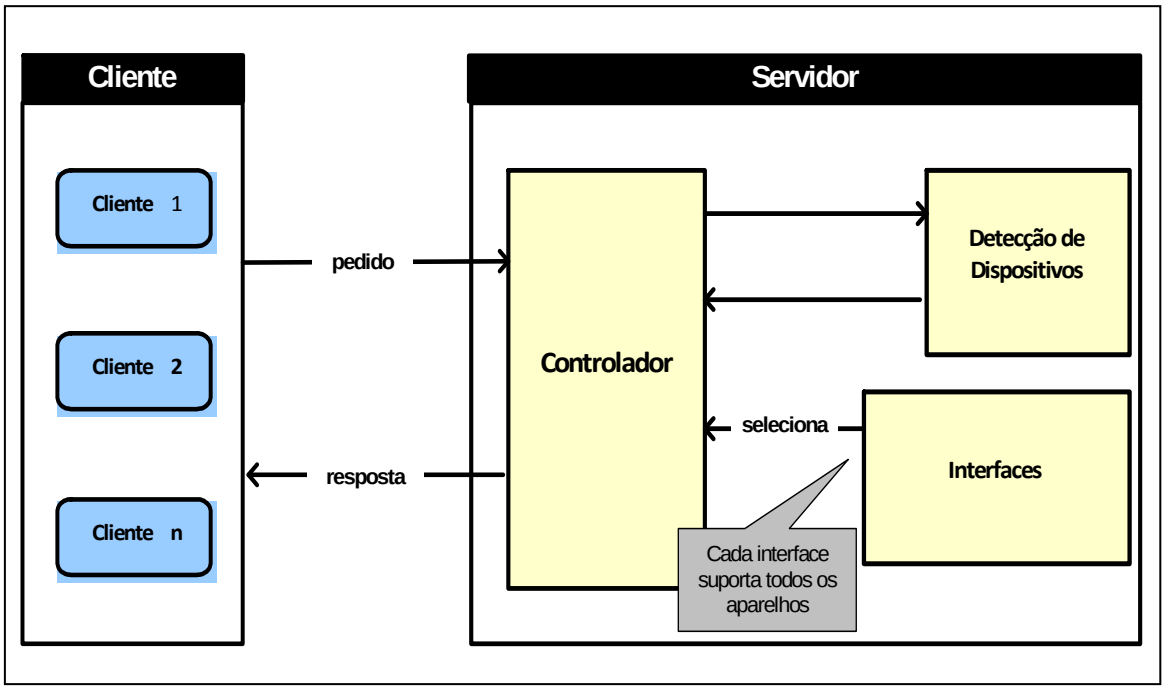


FIGURA 4.5 - Arquitetura Adaptativa
Fonte: Adaptada de Prabhu (2003)

4.2 Trabalhos Relacionados

As próximas seções apresentam as arquiteturas relacionadas ao ambiente móvel que foram estudadas e analisadas para o desenvolvimento da arquitetura GIA.

4.2.1 Arquitetura Dygimes

A arquitetura *Dynamically Generating Interfaces for Mobile and Embedded Systems* (Dygimes) elaborada por Coninx et al. (2003), é um *framework* que tem como finalidade a criação e *design* de um sistema interativo para diversos tipos de dispositivos móveis e embutidos, utilizando o conceito de separação entre as camadas lógica e de apresentação, como ilustra a Figura 4.6.

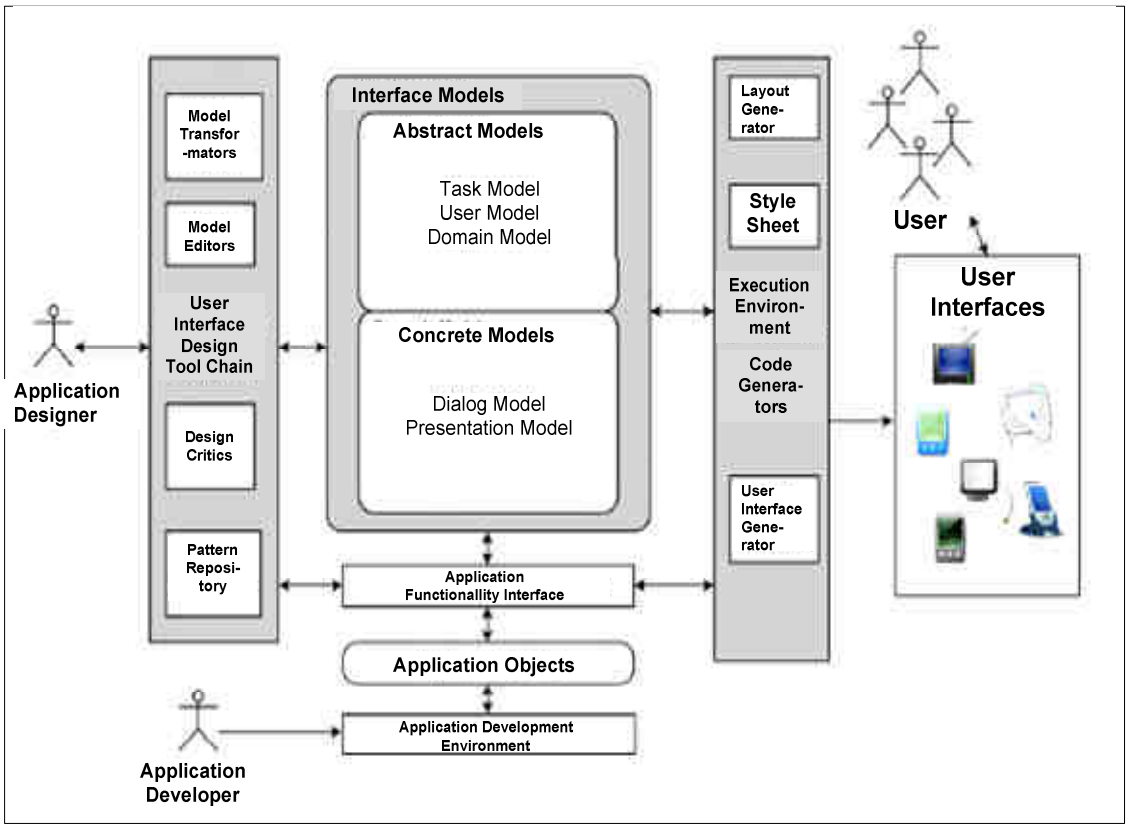


FIGURA 4.6 - Arquitetura Dygimes
Fonte: Coninx et. al. (2003)

A arquitetura Dygimes incorpora várias técnicas de modelo de desenvolvimento de interface do usuário, linguagens como XML, gerenciamento automático de *layout* e transparência de localização.

O *design* começa com um modelo lógico, que uma vez criado, outros poderão estar relacionados ou gerados a partir dele. O *framework* integra a informação do contexto ao *design* da interface do usuário para gerá-las para diversos tipos de dispositivos móveis em tempo de execução. Apresenta os objetivos listados a seguir:

- criar uma especificação das atividades sensíveis ao contexto, utilizando o *ConcurTaskTree* - CTT;

- criar interface do usuário baseada em blocos para separar as atividades;

- relatar a interface do usuário por meio da construção de blocos com as atividades a serem desenvolvidas pelo usuário;

- definir um *layout* da interface e propriedades para a aparência da interface do usuário, gerar um protótipo e avaliá-lo;

O Dygimes utiliza a especificação de atividades, CTT, proposta por Mori (2003), definida como uma especificação de atividades para a modelagem da interface do usuário em alto nível e determinar o conjunto de interações necessárias. A ferramenta TERESA (*Transformation Environment for inteRactive Systems representAtions*) proposta pelo mesmo autor, oferece uma sintaxe gráfica e uma estrutura hierárquica, como uma notação específica entre as atividades do sistema. O formalismo CTT, nesse caso, é combinado com nós de decisão e regras para permitir que a interface do usuário se adapte ao contexto, como ilustra a Figura 4.7.

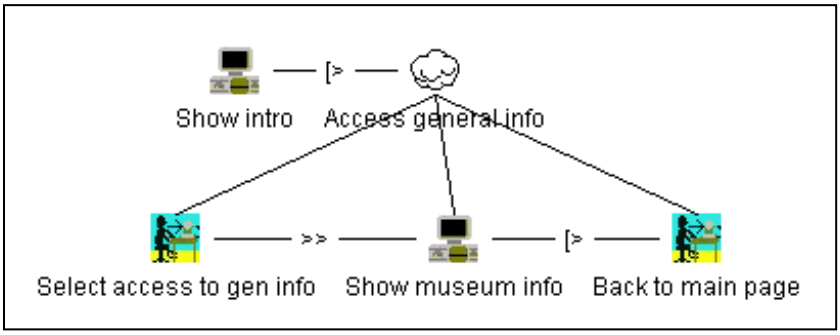


FIGURA 4.7 - Notação CTT na Ferramenta TERESA

A arquitetura Dygimes permite mudanças no contexto, providenciando um modelo de comunicação apropriado, incluindo as transições e o diálogo entre diferentes aparelhos. O modelo de apresentação também suporta comunicação que são distribuídas para diversos tipos de aparelhos.

Uma das limitações da arquitetura Dygimes, segundo o autor, é que ela não garante uma apresentação visual agradável da interface perante sua migração para dispositivos móveis, alegando que dessa forma o sistema apresenta maior flexibilidade.

4.2.2 Arquiteturas em Camadas

O modelo MVC estabelece uma separação da estrutura da interface em três partes distintas: Modelo, Visão e Controle (MVC) como ilustra a Figura 4.8.

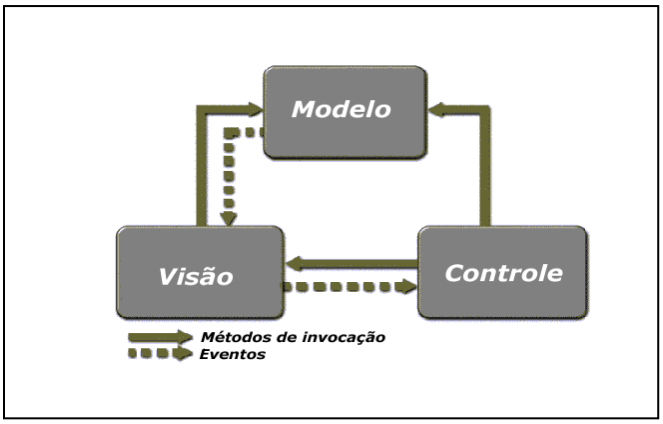


FIGURA 4.8 - Arquitetura *Model View Controll* (MVC)

A visão gerencia a saída gráfica e textual da parte da aplicação visível ao usuário, ou seja, implementa a lógica de apresentação dos dados em um formato apropriado para os clientes. A mesma informação pode ser apresentada de maneiras diferentes para grupos de usuários com requisitos diferentes.

O controle interpreta as entradas do *mouse* e do teclado, comandando a visão e o modelo para se alterarem de forma apropriada, baseado nas preferências do usuário. O modelo representa a estrutura de dados e operações que atuam sobre eles. Em uma aplicação orientada a objetos, constitui as classes da aplicação.

O modelo Seeheim foi um dos primeiros a implementar a separação de três camadas: aplicação, comunicação e apresentação, tendo como seu sucessor o MVC.

Com base nos modelos MVC e o Seeheim surge o *Presentation – Abstraction - Controller* (PAC), que é considerado um refinamento dos modelos anteriores. Baseado também no modelo Seeheim, é construído o *Arch Slinky Model*. Todos os modelos utilizam à idéia de separação em três camadas. As diferenças entre os

modelos é a forma como eles implementam a separação destes níveis e como gerenciam as modificações dos componentes do sistema.

O MCV separa a aplicação da interface, mas a camada de comunicação é integrada à aplicação. O PAC separa os três níveis: apresentação, abstração e aspecto de controle. Cada interface é criada para uma aplicação específica. Coutaz et. al. (1995) definem uma estrutura hierárquica de cooperação de agentes, onde cada um é representado pelas três camadas. De acordo com Menkhaus (2002), a manutenção deste modelo é difícil e o *design* apresenta alta complexidade.

4.2.3 Arquitetura MUSA

Um dos trabalhos estudados sobre interfaces adaptativas é o *Multi User Interfaces Single Application* (MUSA), desenvolvido por Menkhaus (2002), que tem como objetivo a redução no tempo de desenvolvimento, a melhora da manutenção e a flexibilidade de utilização. O núcleo do sistema MUSA é um componente de comunicação, que é representado por um evento gerenciador gráfico, ilustrado na Figura 4.9. O evento é descrito em EG-XML (*event graph* XML) que fica entre a interface do usuário e a aplicação lógica.

A arquitetura MUSA utiliza a linguagem XML e a XSL, para a geração da interface. O serviço adaptável está na camada de comunicação que é implementado a partir do modelo MUSA dividindo os níveis de apresentação, aplicação lógica e comunicação, podendo ser acessado por diferentes classes de aparelhos.

Quando um usuário solicita um serviço, a informação do contexto dispara um mecanismo de adaptação para o gerenciador gráfico juntamente com o conteúdo do serviço, que é feita em tempo de execução. O evento gerenciador gráfico permite a manipulação direta pelo usuário, (seção 1.7.2 do capítulo 3) para escolher uma interface específica de acordo com seu perfil. Isto é possível graças às várias versões da interface, geradas pela XSL.

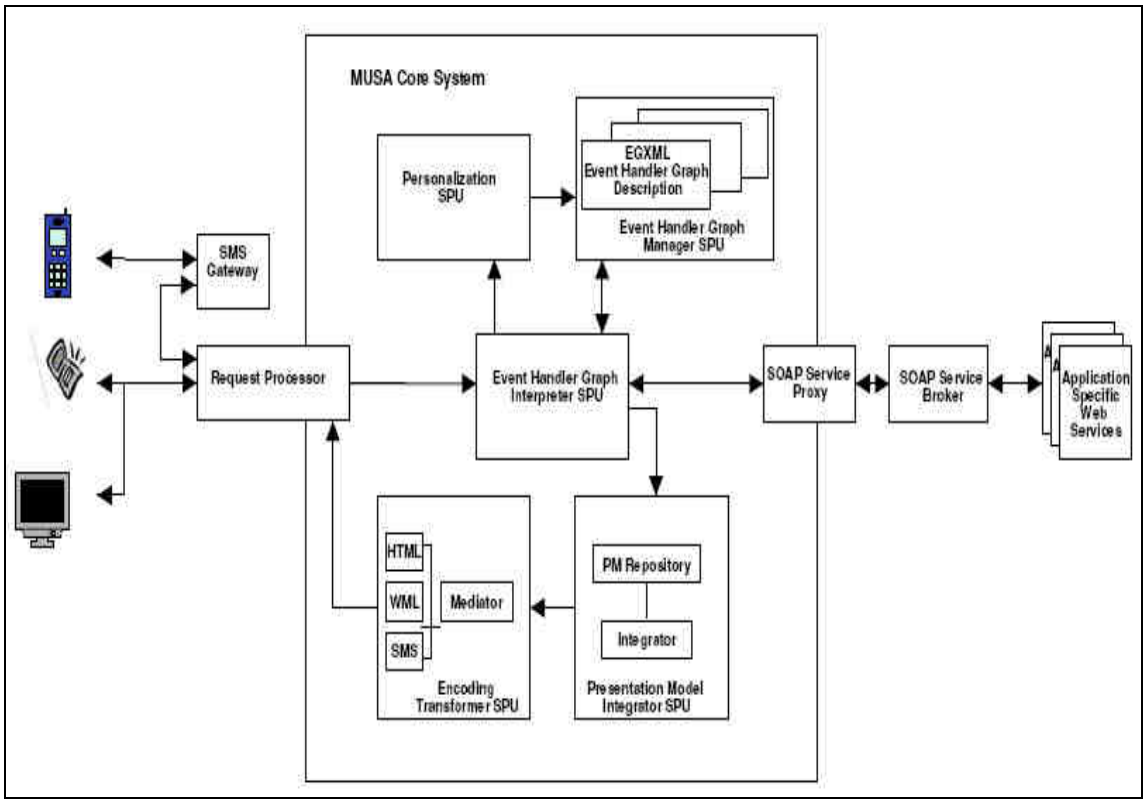


FIGURA 4.9 - Arquitetura MUSA
Fonte: Menkhaus (2002)

Para a identificação do tipo de dispositivo, a arquitetura MUSA utiliza o *user agent* do *browser* do cliente, (seção 3.9 do capítulo 3). Utilizar essas informações para desempenhar a adaptação do conteúdo pode ser problemático, pois apresentam-se limitadas a apenas algumas características como a versão do *browser*, compatibilidade, versão do *token* e o sistema operacional.

A fim de garantir a adaptação a diferentes tamanhos de tela, a arquitetura MUSA utiliza o conceito de hierarquia de grafos, baseada em mecanismos de agrupamento, separação, divisão e re-união de componentes.

4.2.4 Arquitetura Reflexiva

Para Sendin e Lores (2004), arquiteturas para a antecipação de mudanças contextuais são umas das principais lacunas na literatura. Por este fato, o autor desenvolveu uma arquitetura que proporciona uma adaptação dinâmica, utilizando uma estrutura baseada no conceito de computação reflexiva, representada na Figura 4.10.

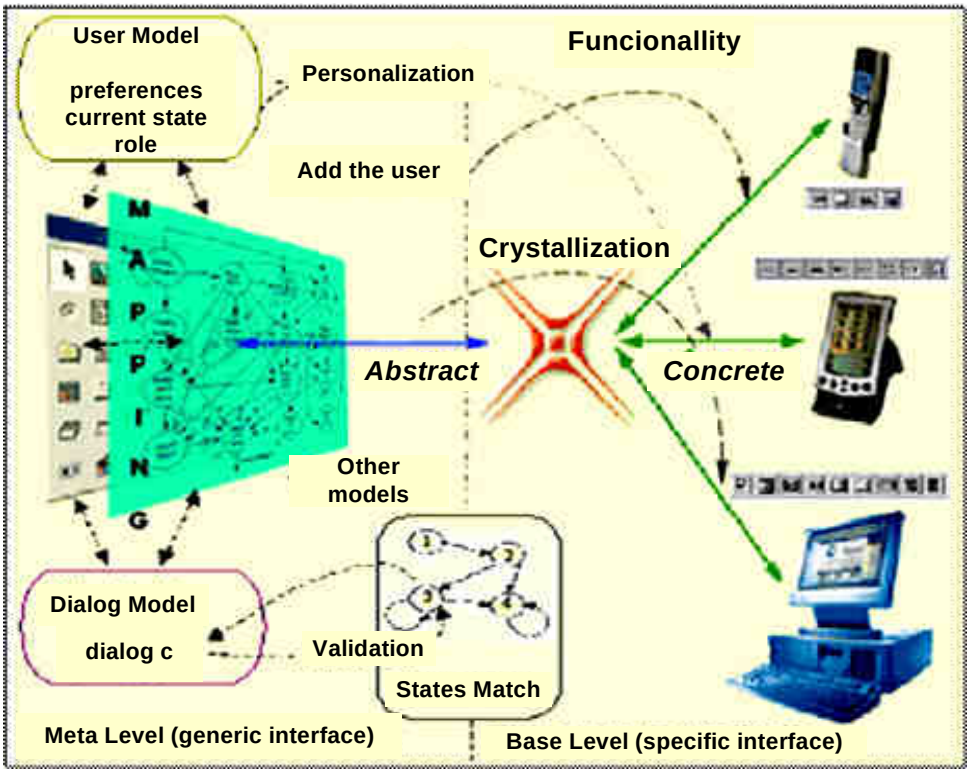


FIGURA 4.10 - Arquitetura Reflexiva
Fonte: Sendin e Lores (2004b)

O modelo da computação reflexiva sugere um paradigma decomposto em pelo menos dois níveis: nível-base, onde estão agrupadas as funções relacionadas exclusivamente ao domínio do problema e o nível reflexivo, que agrupa o código que supervisiona, adapta e retorna informações sobre o nível-base, de acordo com Stehling (1999).

Na arquitetura proposta por Sendin e Lores (2004) utiliza-se a separação entre conceitos, em que o desenvolvedor foca apenas na funcionalidade da aplicação, sem se preocupar com a interface. O desenvolvimento de um sistema é capaz de adaptar a

interface do usuário de acordo com um conjunto de condições relatadas no contexto de uso, que deverão ser modeladas isoladamente no meta-nível, para prover a adaptação desejada. Esta técnica permite ao meta-nível obter informações a respeito do próprio programa, com o objetivo de monitorá-lo, adicionar novas funcionalidades e mesmo fazer alterações adaptativas em tempo de execução.

A funcionalidade do sistema está no nível-base da arquitetura. Ambos os níveis são independentes, pois, por meio deste princípio, nível-base funciona sem a geração da interface do usuário. Isso é feito no meta-nível em tempo de execução, determinando quais componentes concretos da interface representarão a funcionalidade descrita pelos componentes abstratos, dependendo da situação contextual.

De acordo com Sendin e Lores (2004), a principal característica deste tipo de técnica é o fato de que todos os aspectos relevantes da interface do usuário são formalizados explicitamente e representados em modelos declarativos que englobam toda a diversidade de exigências de cada contexto de uso, armazenando o caminho da representação conceitual da interface.

4.2.5 Arquitetura NAC

Lemlouma e Layaida (2004) propôs o NAC (*Negotiation and Adaptation Core*), uma arquitetura voltada a serviços multimídia para ambientes heterogêneos. Oferece uma forma de adaptação estrutural, como XHTML para WML, SMIL e HTML, proporcionando a adaptação de imagens e textos. Os documentos também podem ser adequados para o idioma do cliente solicitante.

Outra proposta da NAC refere-se à apresentação dos documentos que podem ser convertidos de XHTML para WML, caso sejam acessados por celulares. A adaptação do conteúdo é realizada por meio do reconhecimento dos dispositivos utilizando o repositório de perfis CC/PP. Os perfis são armazenados no formato XML e são acessados por meio de *Web Services*.

A arquitetura NAC emprega um conjunto de regras que são utilizadas por um *proxy* para realizar a transformação dos documentos, como ilustra a Figura 4.11.

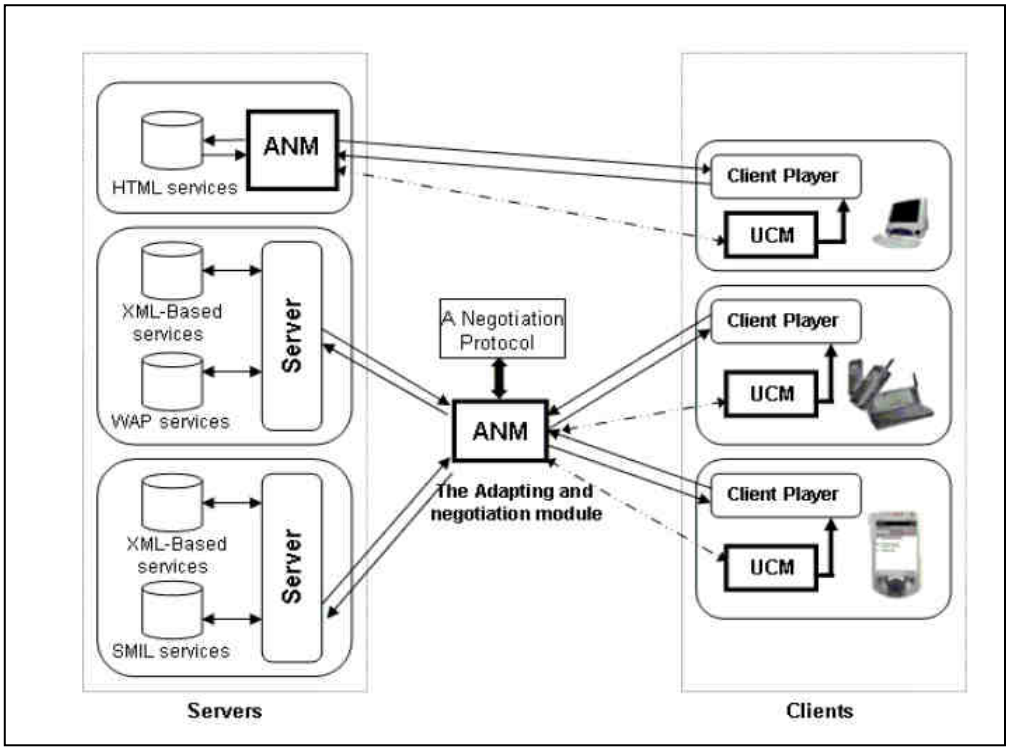


FIGURA 4.11 - Arquitetura NAC
Fonte: Lemlouma e Layaida (2004)

Dessa forma, quando um dispositivo deseja acessar uma página Web, requisita ao *Proxy*, o qual obtém o documento original dos servidores de conteúdo e o documento CC/PP com o perfil do dispositivo. O *proxy* realiza a adaptação baseada nas regras pré-estabelecidas e devolve ao dispositivo o documento transformado.

4.2.6 Ambiente de Desenvolvimento MAS

Viana et al. (2005) apresentaram um ambiente para o desenvolvimento de aplicações com as plataformas Superwaba, J2ME MIDP 1.0/CLDC 1.0 e J2ME MIDP 2.0/CLDC 1.1. Uma das vantagens da ferramenta proposta é escrever uma única vez o código e exibí-lo nas três plataformas citadas.

O trabalho fornece uma infra-estrutura Web e uma coleção de clientes móveis que, em conjunto, oferecem ao *Mobile Application Servers* (MAS) as informações para a realização da adaptação. Estas informações são descritas utilizando as especificações CC/PP e UAProf. O *framework* apresenta restrições como permitir apenas algumas plataformas de programação. O cliente precisa ter um módulo instalado para capturar as informações do dispositivo.

4.3 Arquitetura Proposta

A arquitetura para Geração de Interfaces Adaptativas (GIA) proposta nesta tese foi idealizada como uma arquitetura adaptativa, descrita na seção 4.1.5 deste capítulo. Um dos objetivos dessa abordagem é atender a diversos usuários de forma dinâmica, abrangendo uma grande diversidade de dispositivos.

A arquitetura GIA pretende reduzir o tempo de desenvolvimento e duplicidade de conteúdo por meio de uma metodologia menos dependente de propriedades de um dispositivo único. Dessa forma, almeja-se uma implementação voltada a um ambiente multiplataforma e que seja capaz de adquirir informações de contexto do usuário de modo automatizado. Nesse contexto, o processo de adaptação é realizado integralmente no servidor, sem a necessidade de instalação de nenhum módulo específico no dispositivo móvel, facilitando, dessa forma, o acesso de qualquer aparelho.

A GIA modela a aplicação para o desenvolvimento de interfaces baseando-se no modelo MCV, o qual estabelece uma separação da estrutura da interface em três partes distintas: modelo, visão e controle (seção 4.2.2). A arquitetura proposta é composta de quatro camadas: cliente, serviço de comunicação, serviço de adaptação e serviço de configuração e editoração, ilustrada na Figura 4.12.

Para solucionar a problema de detecção do dispositivo móvel em tempo de execução, a arquitetura GIA apresenta o serviço de comunicação responsável por interceptar a conexão realizada pelo usuário para identificar as características do dispositivo. A fim de garantir a adaptação a novos tipos de dispositivos, a arquitetura GIA utiliza o serviço

de configuração ao qual o desenvolvedor pode adicionar e gerenciar novos recursos.

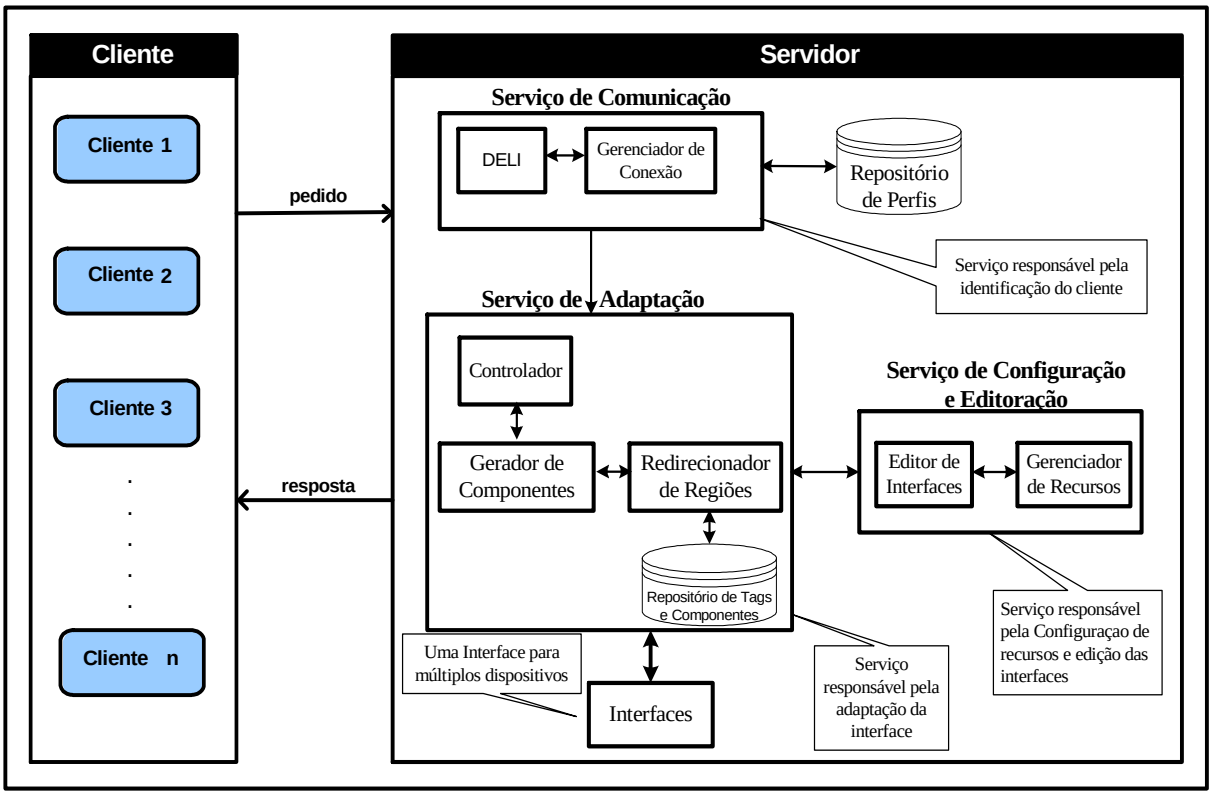


FIGURA 4.12 - Arquitetura GIA

Para realizar a adaptação ao contexto do usuário, o serviço de adaptação recebe as informações do serviço de comunicação, pertinente ao tipo de dispositivo que solicitou a aplicação e, a partir dessas informações, realiza o processo de verificação e transformação para que a interface se ajuste a múltiplos dispositivos.

Isto significa que, nessa arquitetura, os objetos do domínio do problema, por exemplo: componentes e propriedades da interface, dispositivo, *browser*, fabricante, entre outras, ao invés de estarem localizados no código da aplicação, estão implementados em um banco de dados relacional para que possam ser instanciados em tempo de execução a partir de um modelo genérico.

O serviço de configuração e editoração tem o objetivo de auxiliar desenvolvedores no processo de construção de interfaces para dispositivos móveis por meio da arquitetura

GIA, utilizando um ambiente de desenvolvimento com visualização em tempo real da interface em vários tamanhos de telas.

Pode-se dizer ainda que a arquitetura proposta é considerada adaptável, porque é capaz de acomodar possíveis mudanças no domínio do problema por meio da configuração apropriada dos metadados. Isto permite acompanhar a evolução dos requisitos do domínio, e adaptando-se às necessidades dos usuários.

Dessa forma, os especialistas do domínio e os *designers* podem adaptar o sistema para acomodar novas classes de aparelhos por meio da criação, em tempo de execução, dessas classes e seus atributos.

A arquitetura GIA é descrita detalhadamente no capítulo 5.

4.4 Comparação da Arquitetura GIA e Trabalhos Relacionados

As arquiteturas *transcoding*, *ad hoc*, baseada em XML/XSL e convencional, apresentam duplicidade de código no processo de geração de interfaces. Outro fator refere-se à limitação de realizar mudanças no sistema de forma dinâmica, uma vez que ao surgirem novos tipos de dispositivos, o sistema deve ser reestruturado para suportá-los, gerando alto custo de manutenção. Com grande quantidade de código e funcionalidade duplicada no sistema, podem ocorrer problemas como inconsistência e falhas constantes.

Os trabalhos estudados, MUSA descrito por Menkhaus (2002) e Dygimes por Coninx et al. (2003), utilizaram o conceito de separação por camadas, fazendo uma clara distinção entre o modelo de implementação de cada nível: interface do usuário e modelo lógico. Ambos têm como objetivo a adaptação da interface do usuário no ambiente móvel, utilizando técnicas diferentes.

Tanto o projeto MUSA como Dygimes utilizam como linguagem de programação o XML, geração da interface em tempo de execução e Web *services* como forma de

comunicação. Uma das limitações da utilização da abordagem XML refere-se ao código mais complexo, já que é preciso verificar a sintaxe com um analisador, a fim de garantir que os programas usados posteriormente funcionarão sem erros, tornando mais difícil sua codificação. A estrutura é bem definida, mas a semântica um tanto limitada. Os documentos XML definem a estrutura de dados, mas não descrevem como manipular estes dados.

Outro fator diz respeito à transformação da XML pela linguagem XSL, pois ela exige que cada aparelho tenha um arquivo XSL correspondente, ocasionando diversos arquivos para uma interface, ou seja, um para cada modelo de dispositivo.

A arquitetura GIA utiliza a linguagem JSP, que permite ao desenvolvedor Web produzir aplicações que permitam acesso a banco de dados e a arquivos-texto, a captação de informações a partir de formulários, a captação de informações sobre o visitante e o servidor, o uso de variáveis, entre outras. Por meio da utilização da linguagem JSP pode-se separar a programação lógica da programação visual. Outra característica do JSP é produzir conteúdos dinâmicos que possam ser reutilizados.

A arquitetura MUSA utiliza o u*ser agent* para identificar o perfil do dispositivo. Nesse contexto o *user agent* limita-se a informar apenas algumas características (seção 3.9, do capítulo 3). A arquitetura GIA utiliza o CC/PP, considerado uma alternativa para garantir que as características dos mais variados tipos de dispositivos sejam capturadas, ampliando, dessa forma, a capacidade de adaptação ao contexto do usuário.

Sendin e Lores (2004) propôs uma arquitetura reflexiva para solucionar o problema de adaptação da interface em dispositivos móveis. Um aspecto a ser observado nesta arquitetura é o grande número de exigências que devem ser obedecidas para a implementação de todos os requisitos estabelecidos. Apesar do potencial da abordagem reflexiva, seu uso para tempo real pode ser questionado com relação aos aspectos de desempenho e previsibilidade. A arquitetura GIA é baseada em uma arquitetura

adaptativa e fundamentada no modelo MCV, o qual estabelece uma separação da estrutura da interface em camadas.

O modelo MVC visa implementar com maior facilidade e clareza a persistência de dados, controle de segurança, comunicação em rede e fluxo de visualização. Desenvolvedores podem trabalhar separadamente e integrar as camadas por meio de interfaces bem definidas. Esta separação permite que alterações do domínio possam ser mais facilmente modificadas e estendidas para atender a novas exigências, bem como possibilita que a interface com o usuário apresente várias visões de um só modelo, sem interferir com a lógica de negócio.

Outro trabalho estudado foi o NAC. A limitação apresentada nesta arquitetura diz respeito a necessidade de descrever cada novo dispositivo em *Universal Profiling Schema* (UPS), uma forma para minimizar a troca de perfis entre clientes, servidores ou *proxies*. Nessa arquitetura algumas vezes necessita-se de vários tipos de documentos para cada solicitação do cliente. A arquitetura GIA gera apenas um arquivo da interface.

No trabalho apresentado em Viana et al. (2005), o cliente precisa ter um módulo instalado para capturar as informações do dispositivo. Caso o cliente não possua esse módulo, o processo de adaptação não acontece. Na arquitetura GIA o processo de adaptação é realizado integralmente no servidor, sem a necessidade de instalação de nenhum módulo específico no dispositivo móvel, facilitando, dessa forma, o acesso de qualquer aparelho sem uma preocupação prévia com instalação de *software* adicional no cliente.

Com relação ao .NET *framework* da Microsoft (seção 2.7.4 do capítulo 2), pode-se ressaltar que apresenta menos flexibilidade devido a limitação de sua utilização quanto à plataforma, pois uma aplicação deve ser executada em dispositivos que tenham o sistema operacional Windows. Diversos telefones celulares e PDAs não aceitam o sistema da Microsoft.

Prabhu (2003) sugere uma arquitetura adaptativa para solucionar os problemas das arquiteturas utilizadas atualmente, contudo não aborda as metodologias para implementá-la.

A arquitetura GIA propõe o desenvolvimento de aplicações por meio de uma metodologia menos dependente de propriedades de um dispositivo específico. Dessa forma, a arquitetura proposta proporciona uma implementação voltada a um ambiente multi-plataforma, abrangendo uma grande diversidade de aparelhos existentes e que possam ser lançados futuramente, como detalhada o capítulo a seguir.

O próximo capítulo apresenta detalhadamente a arquitetura para Geração de Interfaces Adaptativas (GIA), por meio da descrição e definição do escopo e do funcionamento de seus serviços: serviço de comunicação, serviço de adaptação e serviço de configuração e editoração. É definida também a sua relação com as tecnologias de armazenamento, comunicação e distribuição que foram abordadas. São apresentados ainda os atores e diagramas de caso de uso e de atividades referentes à arquitetura GIA.

CAPÍTULO 5

ARQUITETURA PARA GERAÇÃO DE INTERFACES ADAPTATIVAS

Esse capítulo descreve a arquitetura para Geração de Interfaces Adaptativas (GIA) que tem como proposta principal a criação de uma interface que preserve a consistência, a usabilidade e a facilidade para migrar de um dispositivo para outro. Essas interfaces podem automaticamente se adaptar a novas classes de aparelhos, oferecendo a mesma funcionalidade, sem a necessidade de programação adicional.

O objetivo da arquitetura GIA é permitir que uma implementação esteja voltada a um ambiente multi-plataforma, partindo de uma descrição da interface genérica que seja independente de dispositivo e que atenda às solicitações de diversos usuários a uma mesma aplicação considerando o tipo de dispositivo móvel em tempo de execução. Na arquitetura proposta, todo processo acontecerá sem que seja necessário instalar nenhum módulo no cliente, deixando a responsabilidade da adaptação para o servidor.

Dessa forma, pretende-se reduzir o tempo de desenvolvimento, por meio de uma metodologia menos dependente de propriedades de um dispositivo único, justificando, dessa forma, a necessidade do conceito de adaptação da interface do usuário, detalhada no capítulo 3.

Assim como os trabalhos estudados, MUSA descrito por Menkhaus (2002) e Dygimes por Coninx et al. (2003), a arquitetura GIA utiliza o conceito de separação por camadas, fazendo uma clara distinção entre o modelo de implementação de cada nível: interface do usuário e modelo lógico. É baseado no PAC *Amodeus* e *Arch Slinky Model* descritos no capítulo 2.

A arquitetura GIA possui uma infra-estrutura para adaptação da interface do usuário de duas formas: estática e dinâmica, conforme detalhadas no capítulo 3. A primeira é

realizada durante o desenvolvimento do *software* pela metodologia de fragmentação de código utilizada pela arquitetura GIA, apresentada no próximo capítulo.

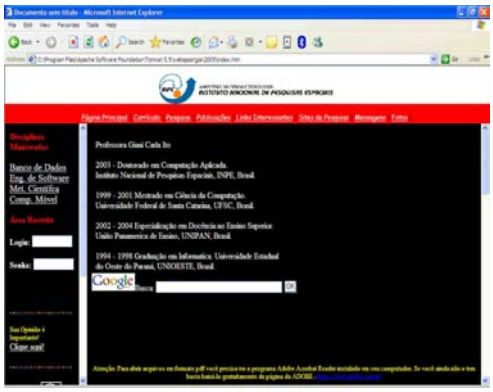
A adaptação dinâmica é realizada durante tempo de execução do *software*, classificada em três tipos:

- Adaptação para diversas classes de aparelhos: proporcionada pelo serviço de comunicação;
- Adaptação dinâmica para diferentes tamanhos de tela: proporcionada pelo serviço de adaptação;
- Adaptação por modificação da interface: proporcionada pelo serviço de configuração e editoração;

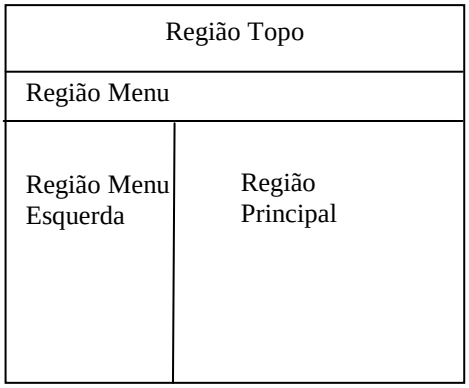
Para validar a arquitetura proposta, esse trabalho integra os serviços de comunicação, por meio do reconhecimento de dispositivos móveis em tempo de execução, o serviço de adaptação que utiliza uma metodologia de fragmentação da interface e o serviço de configuração e editoração, por meio de um ambiente de desenvolvimento, detalhado no capítulo 7.

5.1 Arquitetura para Geração de Interfaces Adaptativas (GIA)

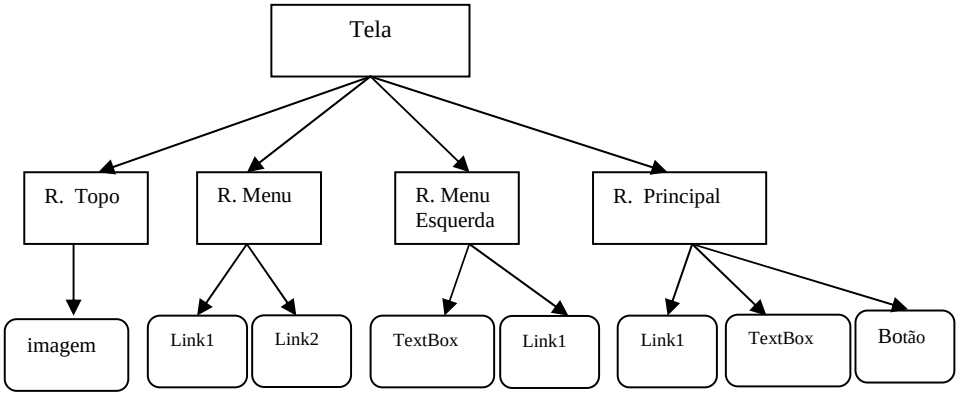
A arquitetura GIA, baseia-se na segunda abordagem, pela proposta da divisão de uma interface em diversas partes. Isso é possível por meio da delimitação de trechos do código em tamanhos pré-definidos. Cada trecho da interface particionada é denominado região. Cada região pode ser composta por sub-regiões e também por componentes visuais como textos, botões, *links*, *combos*, arquivos, entre outras, como ilustra a Figura 5.1.



(a) Site Exemplo



(b) Site Mapeado em Regiões



(c) Estrutura Hierárquica do Site

FIGURA 5.1 - Regiões de uma Interface

A Figura 5.1 (a) ilustra uma interface original contendo componentes como *links*, caixas de texto e botão, enquanto a Figura 5.2 (b) mostra a sua divisão em regiões. A Figura 5.1 (c) ilustra todos os componentes de um mesmo grupo de forma hierárquica.

Para o funcionamento da arquitetura GIA as regiões foram delimitadas em 9 tamanhos numa escala aritmética de 100 a 500 *pixels* com coeficiente de variação de 50 *pixels*. A definição dos tamanhos mínimo e máximo foi baseada em uma análise em que o menor tamanho da tela dos dispositivos móveis atuais, incluindo celulares e PDAs, aproxima-se de 100 *pixels* e o maior de 500 *pixels*.

Caso o tamanho da tela ultrapasse 500 *pixels*, a interface não será fragmentada, sendo mostrada na íntegra. Nesse caso, para viabilizar o processo de adaptação, será mostrada a menor região que mais se aproxima da resolução do aparelho. Por exemplo, para uma tela 230 *pixels*, será carregada a região declarada com 200 *pixels*.

O processo de planejamento da interface e tamanho de suas regiões é fundamental para garantir a usabilidade e consistência dos dados utilizando a arquitetura GIA. A quantidade de regiões é determinada pelo desenvolvedor de acordo com o tamanho da interface, bem como o conteúdo que será exibido. A Figura 5.2 ilustra uma interface acessada por um *desktop* com resolução de 1024x768 *pixels*.

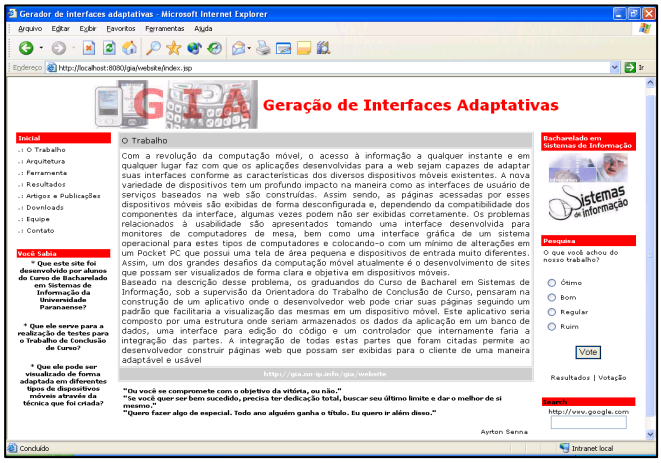


FIGURA 5.2 - Interface sem Regiões

A fim de exemplificar o processo de fragmentação dessa interface, as linhas pontilhadas indicam que ela foi delimitada em 7 regiões, como apresenta a Figura 5.3.

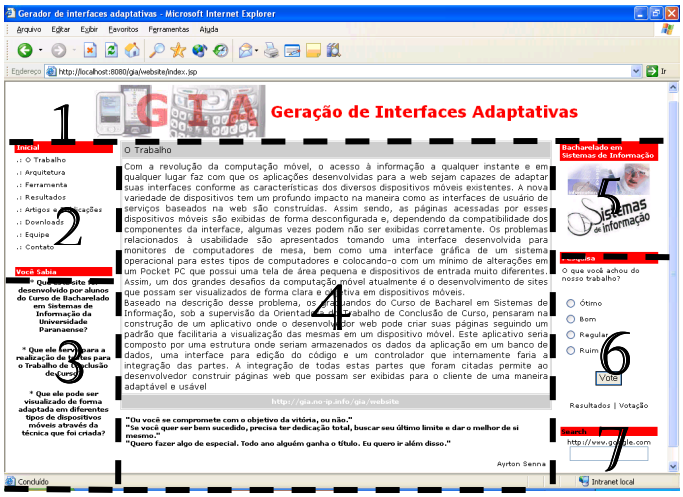


FIGURA 5.3 - Interface com Delimitação de Regiões

A Figura 5.4, apresenta as 7 regiões no simulador de celular *Openwave*, com tamanho de tela 120x160 *pixels*.

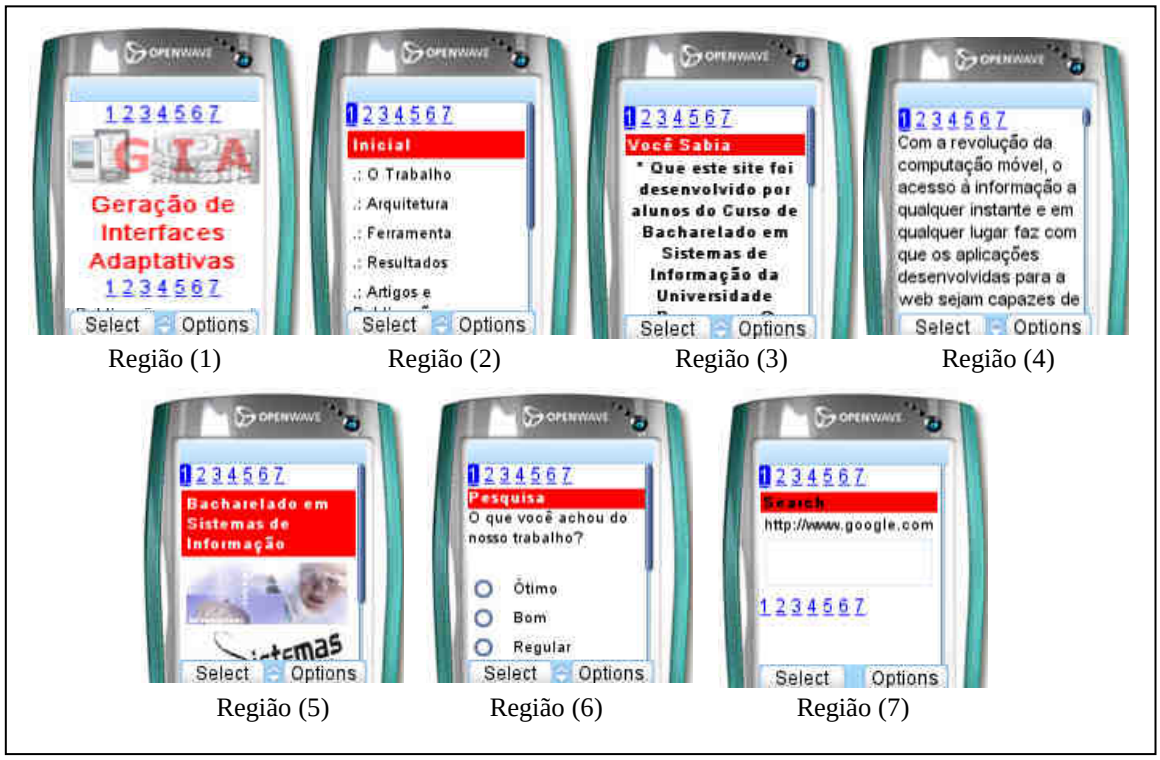


FIGURA 5.4 - Interface Fragmentada em Regiões no Simulador Openwave

5.2 Descrição dos Serviços da Arquitetura GIA

A Figura 5.5 ilustra a arquitetura GIA composta por quatro níveis: cliente, serviço de comunicação, serviço de adaptação e serviço de configuração e editoração, descritos a seguir.

5.2.1 Cliente

O cliente pode solicitar serviços à arquitetura GIA por meio de diversos tipos de PDAs dentre eles Palms, PocketPCs, *Smartphones* ou *desktops*. Não há necessidade de nenhum *software* adicional instalado no cliente, apenas um *browser* compatível com o sistema operacional. A comunicação com a arquitetura é realizada por meio da Internet, por uma rede com ou sem fio. Após a solicitação do cliente, a arquitetura faz a sua

identificação e com base nessas informações, processa os pedidos e os retorna por meio de uma interface adaptada.

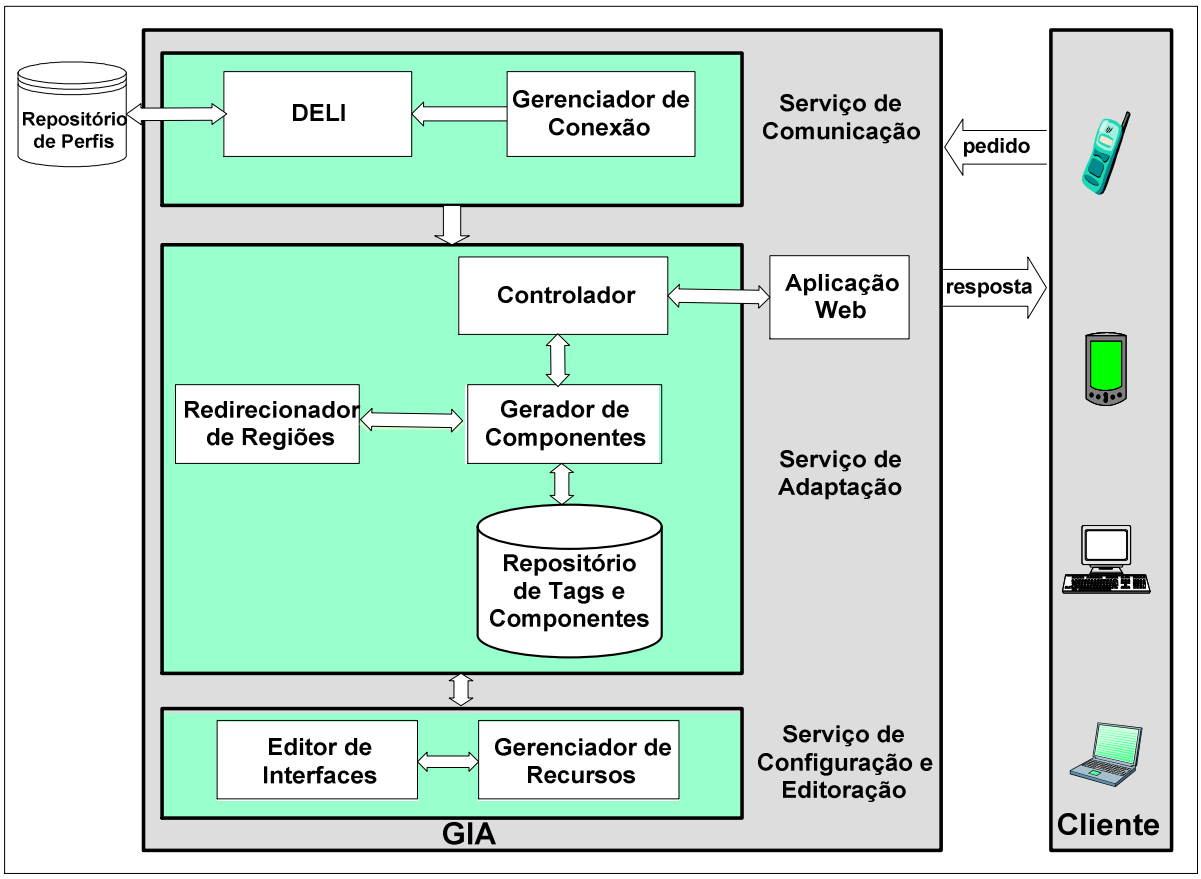


FIGURA 5.5 - Arquitetura GIA

5.2.2 Serviço de Comunicação

O serviço de comunicação é o módulo responsável por interceptar a conexão realizada pelo usuário com o objetivo de identificar as características do dispositivo. É composto pelo *framework* DELI, repositório de perfis e o gerenciador de conexão. Detalhes de implementação podem ser visualizados no apêndice A.

- **Gerenciador de Conexão:** tem a função de acionar o DELI e se comunicar com o repositório externo de perfis, o CC/PP, a fim de obter os dados do dispositivo cliente e retorná-las para o módulo controlador;

- **DELI:** é responsável por realizar a comunicação entre o gerenciador de conexão e o repositório de perfis CC/PP, como detalhado no capítulo 4;

- **Repositórios de Perfis**: contém as informações referentes ao dispositivo móvel com a finalidade de realizar a identificação, como descrito no capítulo 4;

5.2.3 Serviço de Adaptação

O serviço de adaptação recebe as informações da camada de comunicação, pertinentes ao tipo de dispositivo que solicitou a aplicação. O tamanho da tela e o *browser* do cliente são utilizados para buscar no banco de dados qual a linguagem suportada pelo aparelho do usuário. Detalhes de implementação podem ser visualizados no apêndice A.

Com base nessas informações, a arquitetura GIA realiza o processo de mapeamento entre os componentes cadastrados no repositório e a linguagem do aparelho do usuário. Após esse processo, a arquitetura faz a verificação para analisar se o código da interface apresenta *tags* de delimitação de regiões. Caso positivo, os componentes são gerados e exibidos no dispositivo solicitante, como apresenta a Figura 5.6.

Fazem parte do serviço de adaptação o redirecionador de regiões, o gerador de componentes, o controlador e o repositório de tags e componentes, descritos a seguir.

- **Controlador:** é o módulo responsável pelo gerenciamento da arquitetura. Associa funcionalidades do *framework* DELI e do CC/PP para obter as características do cliente que está requisitando a interface. Dessa forma, é possível reconhecer a linguagem que o dispositivo consegue interpretar. O controlador também faz um controle sobre a interface para verificar se está particionada ou não. Esse procedimento é detalhado no capítulo 6;

- **Aplicação Web:** é uma interface Web composta por componentes textuais e gráficos implementados utilizando a metodologia proposta no capítulo 6;

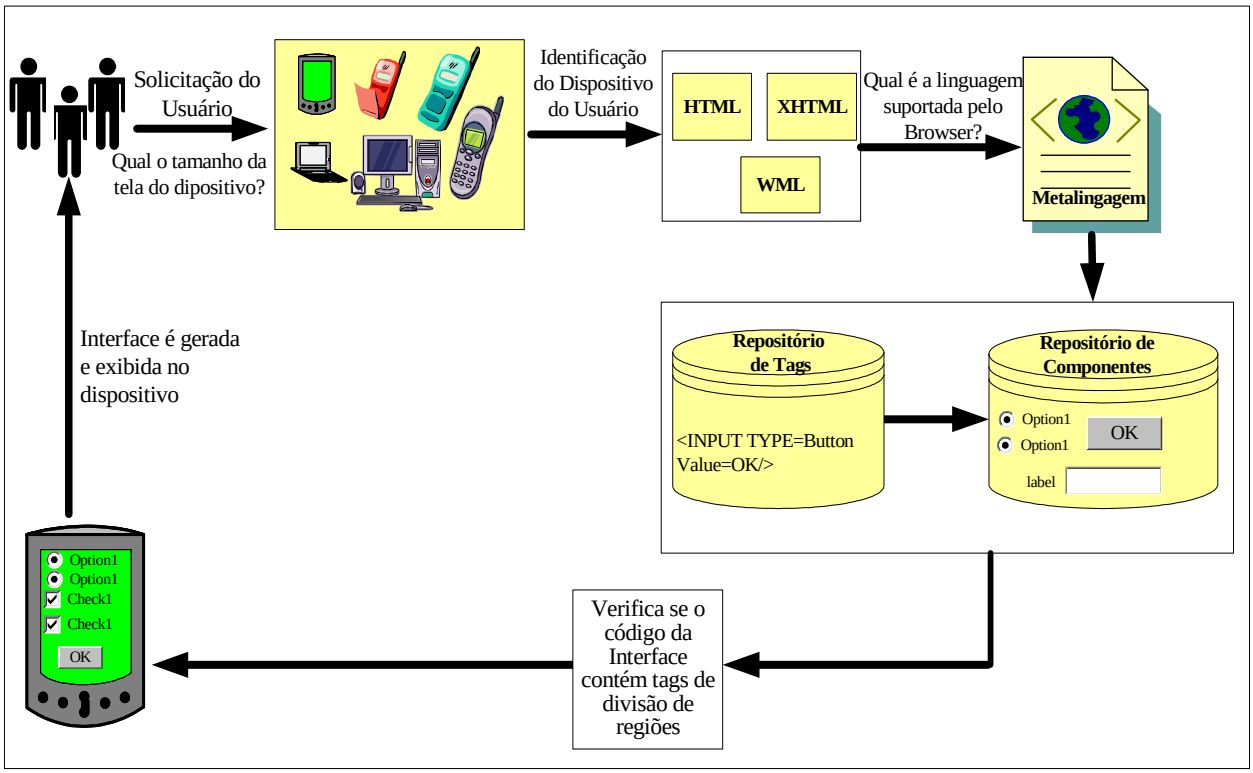


FIGURA 5.6 - Processo de Adaptação da Arquitetura GIA

- **Redirecionador de Regiões**: é o módulo responsável por gerar um menu hierárquico que possibilita a navegação entre a interface. Para tal, o controlador verifica o tamanho de tela do dispositivo solicitante e se a interface está fragmentada, a fim de utilizar ou não as regiões declaradas no código. Caso essas duas condições sejam verdadeiras, o menu é gerado. O redirecionamento de regiões permite ao desenvolvedor reduzir o conteúdo que será exibido no dispositivo móvel, pois as restrições relacionadas ao tamanho da tela podem exigir a classificação das informações. As formas de exibição de conteúdo são apresentadas no capítulo 6;

- **Gerador de Componentes**: é o módulo que tem a função de converter componentes da aplicação Web para a linguagem que o dispositivo solicitante consiga interpretar, como apresenta o capítulo 7;

- **Repositório de Tags e Componentes:** é utilizado para tornar a arquitetura GIA mais flexível. Contém informações sobre *browser*, fabricante, perfis, componentes, *tag-library* e dispositivos. Caso surjam novos modelos, estes podem ser atualizados. O repositório também é utilizado pelo editor de interfaces Web que compõe o ambiente de desenvolvimento GIA. Mais detalhes são apresentados no capítulo 7.

5.2.4 Serviço de Configuração e Editoração

O serviço de configuração e editoração é responsável por manter os dados do sistema e as configurações dos componentes e *tags* da arquitetura GIA. Proporciona aos especialistas do domínio e desenvolvedores de interfaces para dispositivos móveis uma interface adequada para que eles realizem tal tarefa. O serviço proporciona um ambiente para implementar aplicações e gerenciar recursos do sistema. É composto pelo editor de interfaces e gerenciador de recursos.

- **Editor de Interfaces:** proporciona um ambiente para a criação de interfaces Web, permitindo que o desenvolvedor possa estruturar, fragmentar e visualizar interfaces para *desktops* e dispositivos móveis de diversos tamanhos. O capítulo 7 apresenta o editor e seu funcionamento;

- **Gerenciador de Recursos:** é o módulo responsável por gerenciar as informações cadastradas no repositório de tags e componentes. Coordena dados fundamentais para o funcionamento do serviço de adaptação. Seu acesso é realizado pelo editor de interfaces. Esse componente é detalhado no capítulo 7;

5.3 Atores e Funcionalidades da Arquitetura GIA

Nesta seção são apresentados os diagramas de casos de uso usados para visualizar, especificar, construir e documentar o comportamento dos sistemas baseados na arquitetura proposta.

No contexto do funcionamento geral da arquitetura GIA, dois atores apresentam-se como responsáveis por interagir com o sistema: o cliente e o desenvolvedor, como ilustra a Figura 5.7.

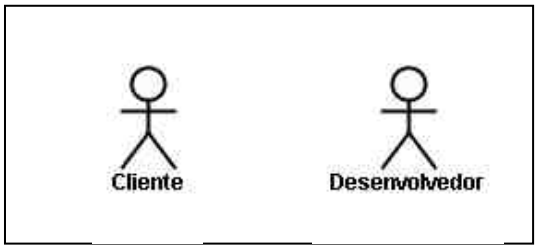


FIGURA 5.7 - Atores da Arquitetura

5.3.1 Funcionalidades do Ator Cliente

O cliente interage com a arquitetura GIA acionando todo o mecanismo de adaptação. Isso acontece quando munido do seu dispositivo móvel ou computador pessoal o cliente solicita uma aplicação Web.

Após a solicitação do cliente, vários processos são acionados, como ilustra o diagrama de caso de uso da Figura 5.8. São eles:

- **Capturar conexão**: processo que tem a função de acionar a interceptação da conexão realizada pelo cliente;
- **Acionar o DELI:** processo responsável por ativar a biblioteca DELI;
- **Identificar o dispositivo:** processo que realiza a identificação do dispositivo por meio do repositório de perfis CC/PP;
- **Enviar informações para o controlador**: tem a função de enviar as informações capturadas do processo capturar conexão;

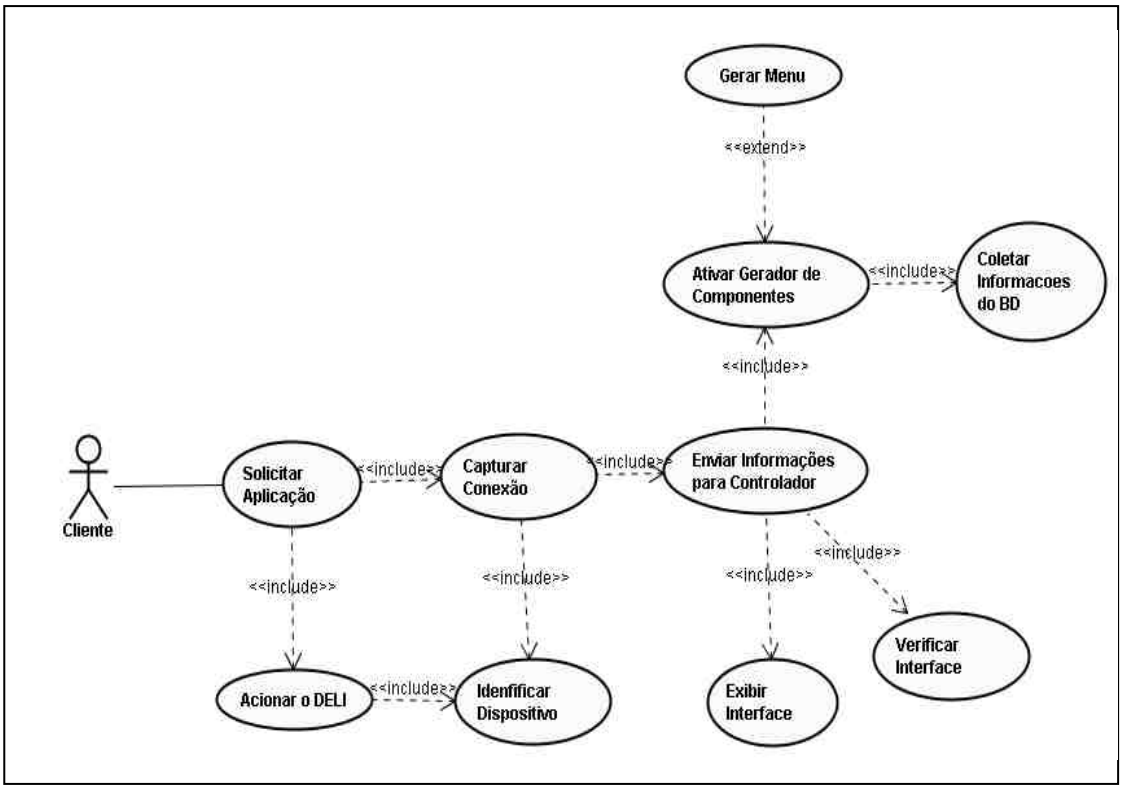


FIGURA 5.8 - Diagrama de Caso de Uso do Cliente

- **Ativar gerador de componentes:** processo que aciona o gerador de componentes para transformá-los na linguagem suportada pelo dispositivo solicitante;

- **Coletar informações do repositório de tags e componentes:** compara as informações do código da interface com as cadastradas no repositório de tags e componentes;

- **Verificar interface:** processo que verifica se o código da interface essa dividido em regiões ou não;

- **Gerar menu:** processo que aciona o procedimento para a geração do menu hierárquico que será exibido nas telas fragmentadas;

- **Exibir interface:** mostra a interface no dispositivo, de acordo com os resultados obtidos pelos processos anteriores;

5.3.2 Funcionalidades do Ator Desenvolvedor

O desenvolvedor tem como principal função acionar ações sobre o serviço de configuração e editoração da arquitetura GIA. É responsável por criar a aplicação e configurá-la no ambiente de desenvolvimento GIA, podendo atualizar recursos do sistema, quando necessário. A Figura 5.9 apresenta o diagrama de caso de uso referente ao ator desenvolvedor.

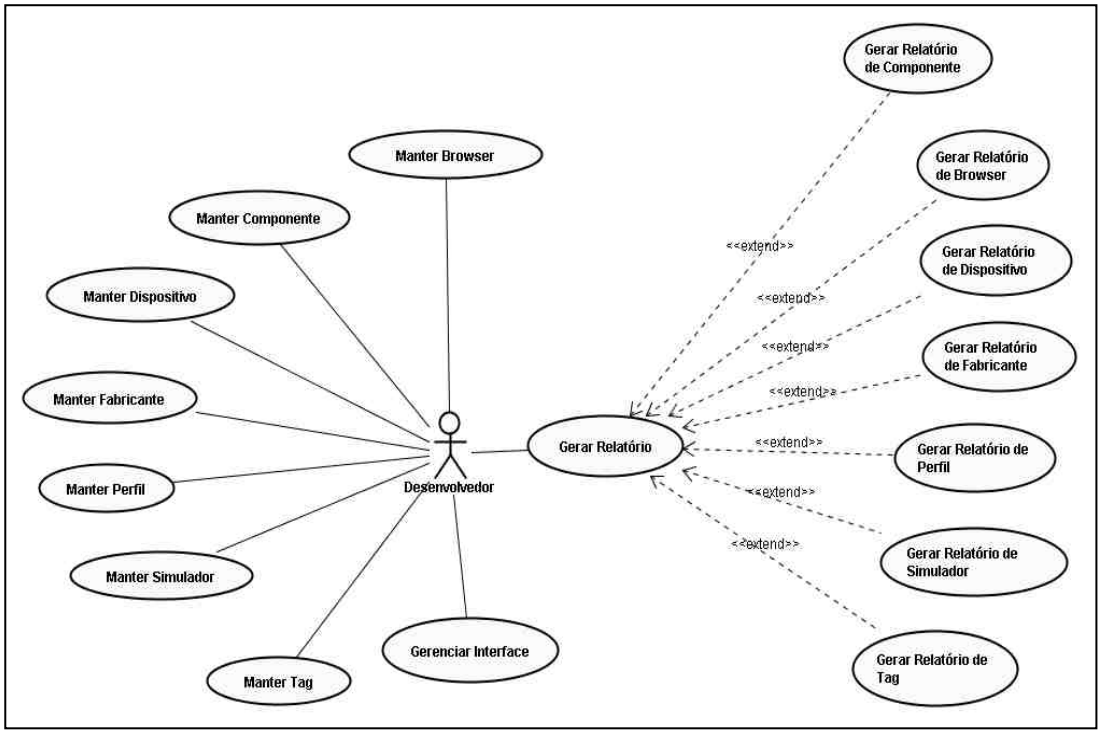


FIGURA 5.9 - Diagrama de Caso de Uso do Desenvolvedor

As funcionalidades executadas pelo desenvolvedor são apresentadas nos processos a seguir:

- **Manter Browser:** é o processo que aciona o cadastro de *browser*, utilizado pela arquitetura GIA, como forma de reconhecer as características do dispositivo solicitante e, dessa forma, gerar a interface de acordo com a linguagem por ele suportada;

- **Manter Componente:** componente é todo elemento que faz parte de uma determinada linguagem de programação, como, por exemplo: a linguagem HTML possui os elementos IMG, TITLE, BODY, entre outras. Esse processo aciona os cadastros desses componentes e suas propriedades;

- **Manter Dispositivo**: processo que ativa o cadastro dos diversos tipos de dispositivos móveis;

- **Manter Fabricante**: processo que ativa o cadastro dos fabricantes de dispositivos móveis;

- **Manter Simulador:** processo utilizado para acionar o cadastro de simuladores, pois durante a codificação da interface, o desenvolvedor poderá visualizar a parte já implementada em um simulador de dispositivo móvel;

- **Manter Tag:** processo que aciona o cadastro das *tags*[1] e suas propriedades;

- **Manter Perfil:** processo que aciona o cadastro dos perfis[2] dos dispositivos móveis;

- **Gerar Relatório:** processo responsável por ativar a geração dos relatórios da arquitetura GIA, tais como: relatório dos componentes e suas propriedades, relatório dos *browsers*, relatório dos dispositivos móveis, relatório dos fabricantes, relatório dos perfis, relatório dos simuladores e relatório das *tags* e suas propriedades;

- **Gerenciar Interface:** processo responsável por gerenciar o arquivo da interface e enviá-lo ao controlador da arquitetura.

---

[1] T*ag* é todo elemento que faz parte da linguagem HTML, como por exemplo, os elementos IMG, TITLE, BODY, entre outras. As *tags* possuem propriedades, no caso do elemento IMG, os atributos SIZE e COLOR.

[2] Perfil é nome da linguagem de programação suportada por um browser

A fim de detalhar o fluxo de controle de uma atividade para outra, a Figura 5.10 ilustra o diagrama de atividades representando as atividades realizadas pela arquitetura GIA descrita na seção 5.1 desse capítulo.

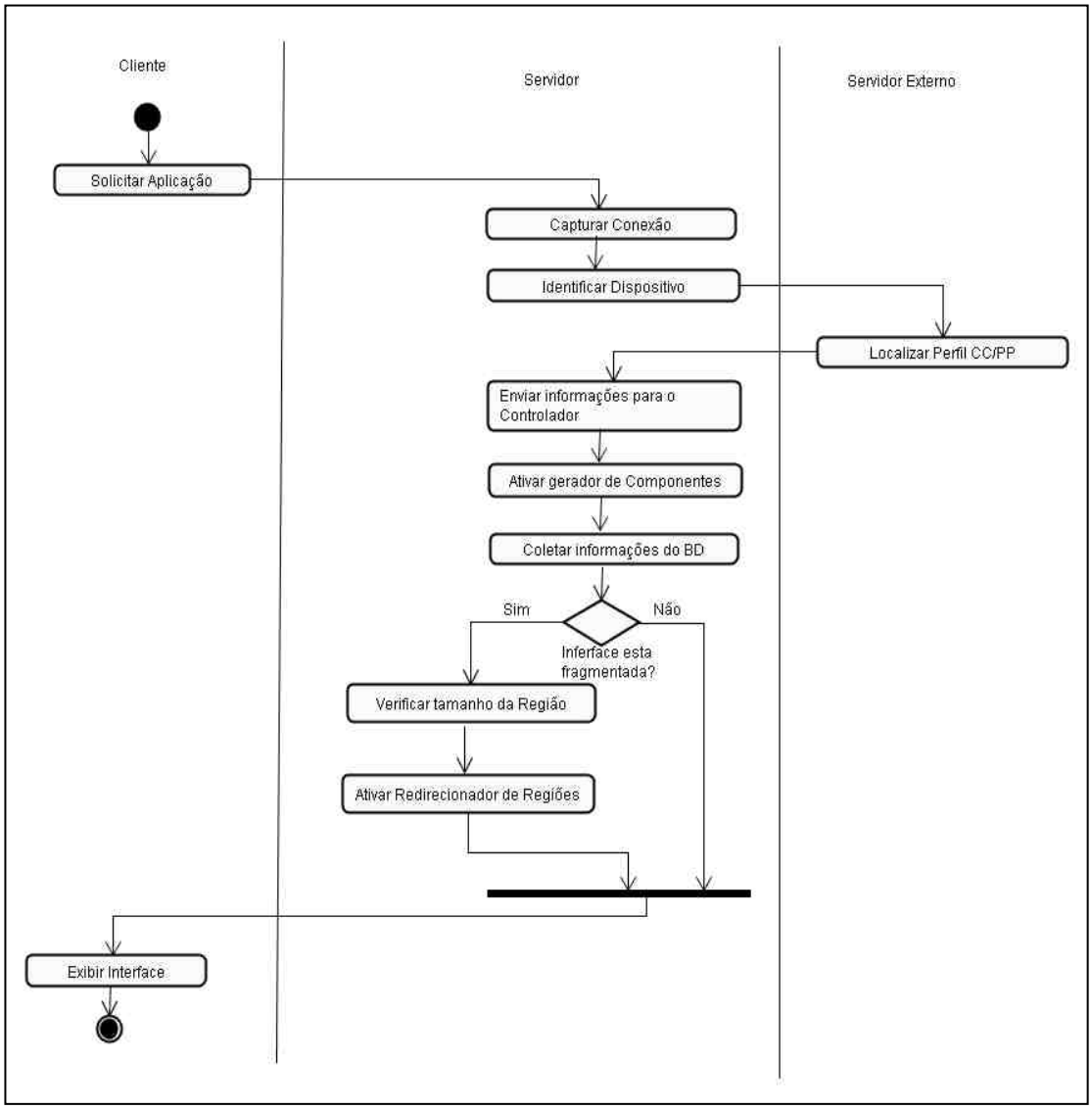


FIGURA 5.10 - Diagrama de Atividades da arquitetura GIA

Quando o usuário solicitar uma aplicação por meio de um dispositivo móvel ou computador pessoal, o gerenciador de conexão captura-a e faz uma chamada para o DELI, para identificar o dispositivo e localizar no repositório externo de perfis as

informações relevantes, tais como o tamanho de tela, *browser*, sistema operacional, entre outras.

Após esse processo, o gerenciador de conexão envia os parâmetros para o *controlador* que, por sua vez, ativa o gerador, o qual utiliza as informações dos componentes armazenados em banco de dados para converter as *tag-libraries* declaradas no código em uma linguagem que o *browser* do dispositivo consiga interpretar.

Em seguida, o controlador analisa se a interface está fragmentada. Caso positivo, verifica qual a região em cuja faixa de tamanhos de tela o dispositivo se encaixa. Uma vez determinado o tamanho mais adequado, a página é fragmentada e exibida de acordo com as regiões definidas pelo desenvolvedor, permitindo ao usuário navegar entre as telas. Caso a interface não esteja fragmentada, ela é exibida na forma original.

Quando uma solicitação vier de um *desktop,* tem-se apenas uma região definida, contendo toda a página. Dessa forma, o sistema detecta a resolução de tela de 800x600 *pixels* e seleciona automaticamente as delimitações para esse tamanho. Portanto, a página inteira retorna ao *browser*.

Com o objetivo de descrever a metodologia de fragmentação de regiões adotada pela arquitetura GIA, o próximo capítulo descreve detalhadamente o processo de particionamento de uma interface Web em diversas partes, para que seja possível a geração de interfaces adaptativas em múltiplos dispositivos.

CAPÍTULO 6

METODOLOGIA DE DESENVOLVIMENTO PARA GERAÇÃO DE INTERFACES ADAPTATIVAS

O desenvolvimento de interfaces para dispositivos móveis envolve vários desafios, pois a grande diversidade de dispositivos como telefones celulares, *smartphones*, PDAs, entre outros, são lançados constantemente. O ambiente heterogêneo, limitações físicas do aparelho, formas de entrada e sincronização de dados, plataforma e contexto no qual ele será utilizado são elementos cruciais para o sucesso de uma aplicação móvel.

Outro fator que pode influenciar no processo de desenvolvimento é a capacidade que alguns dispositivos possuem para apresentar ou não figuras, imagens ou vídeos, bem como a quantidade variada de linhas da tela e as diferentes formas de conectividade de rede.

Várias pesquisas têm sido realizadas para avaliar a usabilidade de interfaces para dispositivos móveis com o objetivo de facilitar o processo de desenvolvimento relacionado a esse ambiente, como detalhado na seção 2.4 do capítulo 2. De acordo com Xie et. al (2005), os trabalhos atuais focam em duas abordagens: a primeira é transformar páginas Web existentes e a segunda introduz novos formatos e mecanismos para adaptar-se a diferentes tamanhos de tela.

A solução adotada pela arquitetura GIA consiste em fragmentar uma interface em um conjunto de partes, bem como gerar um índice do conteúdo. Isso é possível por meio da delimitação de trechos do código em tamanhos pré-definidos. Cada trecho da interface particionada é denominado região. Cada região pode ser composta por sub-regiões como detalhado no capítulo 5.

O processo de estruturação da interface e tamanho de suas regiões é fundamental para garantir a usabilidade e consistência dos dados utilizando a arquitetura GIA.

Para a aplicação da metodologia proposta nas próximas seções o *designer* pode optar pelos vários tipos de interfaces para o ambiente móvel, detalhados no capítulo 3.

6.1 Formas de Exibição da Interface

A arquitetura GIA proporciona ao desenvolvedor várias formas de implementação da interface, dentre elas a do *layout* original e *layout* reduzido, apresentado em Roto (2006). Nesse trabalho é proposto o *layout* com seleção de conteúdo e o *layout* fragmentado.

6.1.1 *Layout* Original

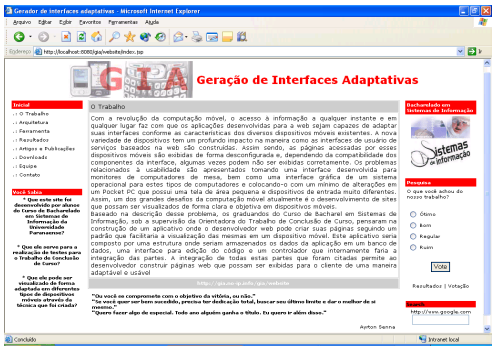
Nessa forma de apresentação, o *layout* original será exibido por completo, ou seja, a página é mostrada como foi projetada, como apresenta no *desktop* a Figura 6.1(a) e no computador de mão em 6.1(b). Esse tipo de exibição pode ser problemática pela quantidade de informações que devem ser exibidas nos dispositivos móveis, causando a falta de usabilidade da aplicação, dificuldade de navegação e visualização do conteúdo, em telas menores, como exemplo, os celulares e PDAs.

6.1.2 *Layout* Reduzido

O *layout reduzido* apresenta o conteúdo formatado ao tamanho da tela. Todas as páginas são apresentadas uma após a outra em uma única coluna, sem a necessidade da barra de rolagem horizontal. Dessa forma, o conteúdo é visualizado conforme a estrutura da página HTML. Pela Figura 6.1(c), pode-se verificar o *layout* reduzido

6.1.3 Proposta de Seleção de conteúdo

Na exibição da interface por meio da seleção de conteúdo, as informações são consideradas de acordo com o nível de relevância para a aplicação. As restrições ligadas ao tamanho de tela dos dispositivos móveis tornam necessárias que as informações visualizadas nesses aparelhos sejam classificadas, ou seja, reduzidas de forma que seja mostrado somente o conteúdo principal do *site*.



(a) Layout Original no Desktop

(b) Layout Original no PDA

(c) Layout Reduzido

FIGURA 6.1 - Formas de Exibição de Layout

Um método que pode ser adotado para realizar a seleção de conteúdo são as prioridades. A metodologia proposta pela arquitetura GIA utiliza o atributo *seq* para indicar a seqüência em que o bloco de código deve ser exibido, como ilustra a parte em negrito da Figura 6.2.

O atributo *seq* 1 indica a primeira região a ser exibida, já a *seq* 2 a segunda, e assim por diante. Durante a fase de *design* da interface o desenvolvedor pode determinar quais

regiões devem ser mostradas, ou quais são mais relevantes, acrescentando ao código o nível de prioridade.

```
<%@taglib uri="gia" prefix="gia"%>
<gia:html>
<body>
<table width="100%" height="100%" border="1">
 <tr>
  <td colspan="3">
        <gia:regiao200 id="regiao200" seq="1" comentario="Titulo">
            <gia:regiao150 id="regiao150" seq="1" comentario="Titulo">
             <gia:regiao100 id="regiao100" seq="1" comentario="Titulo">
                <font color="#006666" size="2" face="Verdana, Arial, Helvetica, sans-serif">
                    <strong> GERADOR DE INTERFACES ADAPTATIVAS </strong>
                </font>
             </gia:regiao100>
            </gia:regiao150>
        </gia:regiao200>
  </td>
 </tr>
 <tr>
  <td width="20%">
        <gia:regiao200 id="regiao200" seq="2" comentario="Menu">
            <gia:regiao150 id="regiao150" seq="2" comentario="Menu">
             <gia:regiao100 id="regiao100" seq="2" comentario="Menu">
                    <img src="imagens/palm.gif" align="center"> <br>
             </gia:regiao100>
            </gia:regiao150>
        </gia:regiao200>
```

FIGURA 6.2 - Método Seleção de Conteúdo

6.1.4 Layout Fragmentado

Para que seja possível a exibição de uma interface que foi projetada para a resolução de um *desktop*, sem que seja necessário reescrevê-la novamente, a arquitetura GIA propõe

uma forma de fragmentação do código em diversos tamanhos, de acordo com os dispositivos para os quais se pretende exibir a interface, como detalhado no capítulo 5.

Por meio da utilização do *layout* fragmentado uma interface poderá ser dividida e apresentada em diversas partes, contendo menus que permitem navegar entre uma tela e outra. Essa forma de visualização permite a exibição de uma interface em múltiplos dispositivos.

A Figura 6.3 apresenta a interface da Figura 6.1 (a), fragmentada em regiões. Percebe-se no menu gerado pela arquitetura GIA, que nas regiões de 100 a 200 *pixels*, houve uma divisão da interface em 7 partes.

Nas regiões de 100 a 300 *pixels* a imagem do logotipo do *site* não aparece, pois o tamanho da tela é muito pequena e, nesse caso, o desenvolvedor pode optar por exibi-la ou não. Para as regiões seguintes o logotipo aparece adaptado.

Observa-se, na Figura 6.3, que à medida que o tamanho das regiões vai aumentando, a quantidade de opções do menu diminui. Por exemplo, a região de 100 *pixels* apresenta um menu com 7 opções, enquanto a região de 500 apenas 4.

Esse processo acontece de forma automática por meio do serviço de adaptação e de comunicação da arquitetura GIA apresentado no capítulo 5.

FIGURA 6.3 - Método Layout Fragmentado

6.2 Metodologia para Desenvolvimento de Interfaces para Dispositivos Móveis

Esse trabalho propõe uma metodologia de programação que divide a interface em regiões, pela intercalação do código. Essas regiões serão interpretadas somente quando a interface for visualizada em um dispositivo móvel.

Cada região poderá ter vários tamanhos que serão determinados de acordo com as telas para as quais se pretende exibir a interface, podendo ser visualizados e alterados conforme desejado.

6.2.1 Delimitação de Regiões

A arquitetura GIA utiliza 9 tamanhos de regiões numa escala aritmética de 100 a 500 *pixels,* como detalhado no capítulo 5.

Para garantir o funcionamento da metodologia proposta, primeiramente a interface deve conter a referência à arquitetura GIA conforme a sintaxe apresentada na Figura 6.4. O conteúdo estruturado dentro das *tags* <gia:html> e </gia:html > são tratados pela arquitetura durante a requisição a uma aplicação.

```
<%@taglib uri="gia" prefix="gia"%>
<gia:html>
    Exemplo de delimitação de tags da ferramenta GIA
</gia:html>
```

FIGURA 6.4 - Delimitação de *Tags*

Para que as regiões sejam reconhecidas pela aplicação por meio da arquitetura GIA devem ser identificadas e seqüenciadas pelas propriedades *id, seq e comentário,* demonstrados no código pela Figura 6.5.

```
<gia:regiao300 id="Topo300" seq="1" comentario="Topo">
```

FIGURA 6.5 - Propriedades para Identificação de Regiões

- **Id:** é o nome que identifica o trecho da região;
- **Seq:** indica o nível de prioridade de cada região dentro de sua hierarquia. A seqüência determinada por esse atributo indica a ordem que as telas aparecerão;
- **Comentário:** é a descrição da região. É exibido por um menu durante a navegação para possibilitar ao usuário um melhor entendimento do mapeamento da interface;

A fim de exemplificar a utilização das delimitações de regiões apresenta-se a seguir dois trechos de códigos, um com aplicação da metodologia proposta e outro sem.

A Figura 6.6 exibe um trecho de código, sem a metodologia de fragmentação.

```
<%@taglib uri="gia" prefix="gia"%>
<gia:html>
<body>
<table width="100%" height="100%" border="1">
 <tr>
  <td colspan="3">
        <gia:componente  tipo="IMG"
        propriedade="SRC=imagens/logo_do_site.jpg;ALIGN=middle"/>
        <font color="#006666" size="2" face="Verdana, Arial, Helvetica, sans-  serif">
              <strong> GERADOR DE INTERFACES ADAPTATIVAS </strong>
       </font>
  </td>
 </tr>
 <tr>
```

FIGURA 6.6 - Código Sem Utilização de *Tags*

Para que cada bloco de código seja delimitado dentro de uma região, o conteúdo deve estar estruturado dentro das *tags* <gia:regiao> e </gia:regiao >.

A Figura 6.7 apresenta o mesmo código da Figura 6.6, delimitado por uma região 100 *pixels*.

```
<%@taglib uri="gia" prefix="gia"%>
<gia:html>
<body>
<table width="100%" height="100%" border="1">
 <tr>
  <td colspan="3">

    <gia:regiao100 id="regiao100" seq="1" comentário="Titulo">
        <gia:componente  tipo="IMG"
        propriedade="SRC=imagens/logo_do_site.jpg;ALIGN=middle"/>
        <font color="#006666" size="2" face="Verdana, Arial, Helvetica, sans-  serif">
              <strong> GERADOR DE INTERFACES ADAPTATIVAS </strong>
        </font>
    </gia:regiao100>

  </td>
 </tr>
  <tr>
```

FIGURA 6.7 - Delimitação de Código utilizando *Tags*

A Figura 6.8 exibe o mesmo código delimitado por 3 regiões de 100 a 200 *pixels.* A imagem logo_do_site.jpg só será visualizada nos dispositivos com tamanho superior a 150 *pixels*.

```
<%@taglib uri="gia" prefix="gia"%>
<gia:html>
<body>
<table width="100%" height="100%" border="1">
 <tr>
  <td colspan="3">
        <gia:regiao200 id="regiao200" seq="1" comentario="Titulo">
          <gia:componente  tipo="IMG"
          propriedade="SRC=imagens/logo_do_site.jpg;ALIGN=middle"/>
         <gia:regiao150 id="regiao150" seq="1" comentario="Titulo">
          <gia:regiao100 id="regiao100" seq="1" comentario="Titulo">
           <font color="#006666" size="2" face="Verdana, Arial, Helvetica, sans-serif">
                <strong> GERADOR DE INTERFACES ADAPTATIVAS </strong>
           </font>
          </gia:regiao100>
         </gia:regiao150>
```

FIGURA 6.8 - Delimitação de Código em Regiões de 100 a 200 *pixels*

6.3 Modelos de Interface da Arquitetura GIA

Para exemplificar a metodologia proposta pela arquitetura GIA, dois modelos de interface foram elaborados: *templates* simples e comercial. No contexto dos modelos desenvolvidos, os trabalhos exploram a proposta da arquitetura GIA contribuindo para maximizar a adaptação e propiciar uma maior usabilidade.

No *template* simples foram utilizadas duas dimensões de tamanhos: Regiao300 e Regiao100, os quais permitem duas faixas de largura em resoluções distribuídas entre 300 a 500 *pixels* e entre 100 a 299 *pixels*

Esse *template* apresenta um *layout* mais recomendado para a utilização de conteúdo textual, sendo composto por poucas imagens, atendendo principalmente aparelhos que aceitem a linguagem HTML.

O modelo simples é dividido em formato de tabela, composto por cinco partes: cabeçalho, menu, conteúdo, complemento, rodapé e imagens. A fim de melhorar a visibilidade do *site* em dispositivos inferiores a 300 *pixels*, foi selecionado apenas o conteúdo textual e retiradas as imagens do cabeçalho, do conteúdo e complemento intercalando os limites das regiões, conforme ilustra a Figura 6.9.

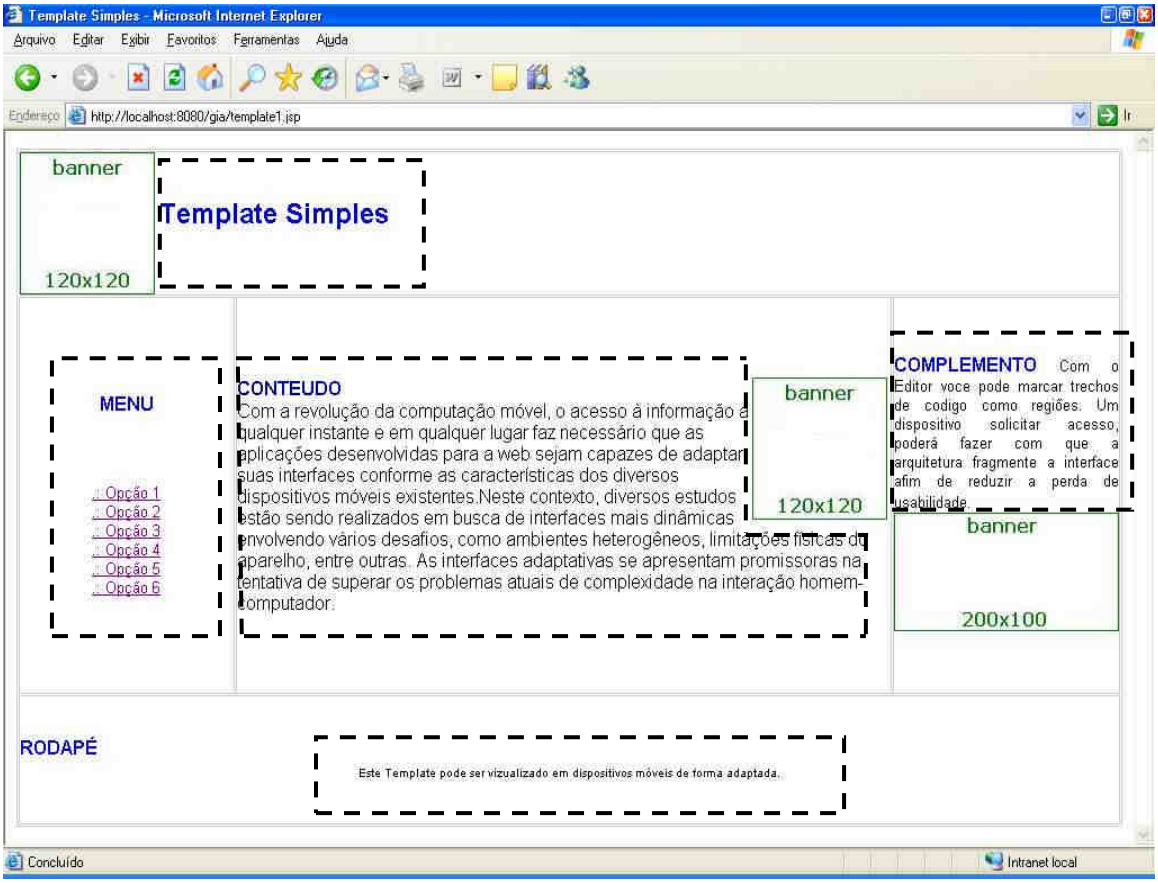


FIGURA 6.9 - Template Simples

Já para dispositivos que possuem dimensões acima de 300 *pixels*, a interface deverá exibir todo o conteúdo em 5 regiões, incluindo as imagens conforme o código completo desse modelo apresentado no apêndice B.

A fim de exemplificar as regiões referentes ao cabeçalho do *template* simples, seu código é descrito na Figura 6.10.

```
<gia:regiao300 id="Topo300" seq="1" comentário="Topo">
        <img src="imagens/img_120x120.jpg" alt="logotipo do site" width="120"
        height="120" align="middle">

       <gia:regiao100 id="Topo" seq="1" comentário="Topo">
            <font color="#0000CC" size="5" face="Geneva, Arial, Helvetica, sans-
            serif"><strong> Template Simples</strong></font>
        </gia:regiao100>

</gia:regiao300>
```

FIGURA 6.10 - Célula do topo da interface simples

O *template c*omercial é um modelo que utiliza uma distribuição de imagens com tamanhos entre 250 a 120 *pixels* de largura, componentes de formulários, atendendo dispositivos com capacidade gráfica superior em termos de resolução.

Esse modelo apresenta-se adaptativo a diferentes arquiteturas utilizando a classificação de conteúdo a serem exibidas durante a execução. Explora o uso de formulários e botões a fim de abranger todos os tamanhos tratados pela arquitetura GIA.

A partir de uma solicitação ao servidor, o tamanho da tela é identificado automaticamente. O sistema analisa os fragmentos do código e verifica em qual faixa de tamanho ele se encaixa. A seleção do conteúdo a ser exibido é determinada de acordo com as delimitações das regiões.

O *template* comercial é ilustrado na Figura 6.11.

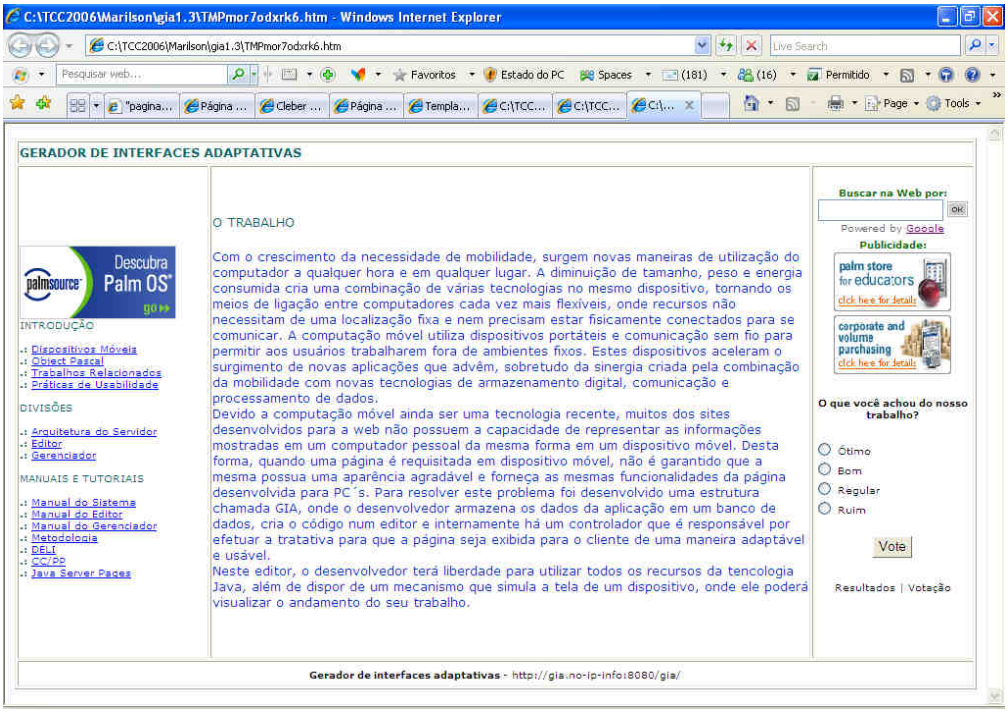


FIGURA 6.11 - Template Comercial da Arquitetura GIA

O *template* comercial é dividido em 6 faixas de exibição distribuídas entre os valores: 400 a 100 *pixels*, como ilustra a Figura 6.12.

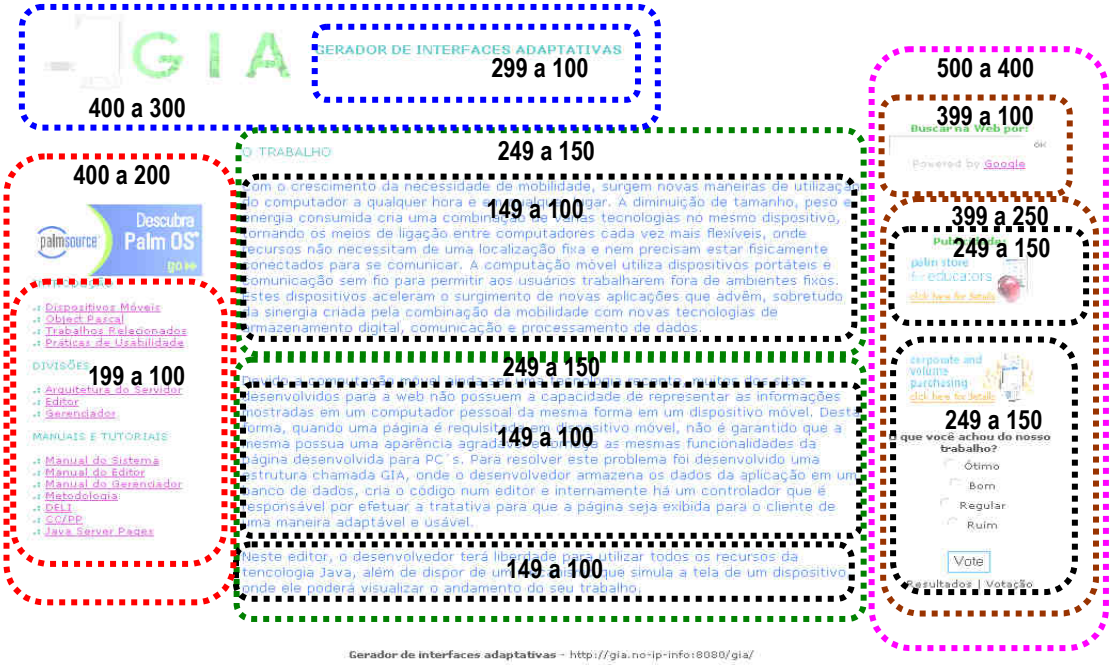
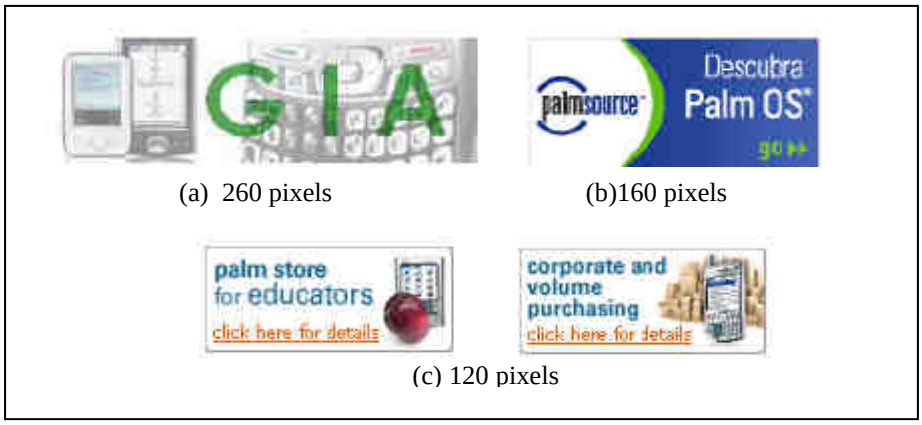


FIGURA 6.12 - Template Comercial dividido em Regiões

As imagens do *template* comercial estão distribuídas nas regiões de acordo com a dimensão da tela do dispositivo móvel considerando a largura, conforme é ilustrado na Figura 6.13 (a),(b) e (c).

A Figura 6.13 (a) ilustra a imagem da parte superior do *template* comercial com 260 *pixels* de largura podendo ser visualizada em dispositivos de no mínimo 300 *pixels*. Na Figura 6.13(b), a imagem apresentada contém a largura de 160 *pixels* disponibilizada para dimensões de até 200 *pixels*.

Já na Figura 6.13 (c), as imagens possuem uma largura de 120 *pixels*, podendo ser exibidas em dispositivos de até 150 *pixels* de largura, de acordo com as delimitações das regiões.



(a) 260 pixels (b)160 pixels

(c) 120 pixels

FIGURA 6.13 - Imagens do Template

No modelo comercial foram aplicados, além das *tag-libraries* de regiões para estruturação da interface, objetos genéricos que são modificados para a linguagem do dispositivo em tempo de execução.

A Figura 6.14 descreve o código, o cabeçalho do *template* comercial, delimitado pelas regiões de 100 a 400 *pixels*.

O código *<propriedade="SRC=imagens/logo_do_site.jpg;ALIGN=middle"/>* na região300 indica que a imagem logo do *site* será exibida apenas em dispositivos com resolução acima de 300 *pixels*.

O código completo dessa interface pode ser visualizado no apêndice B.

```
<%@taglib uri="gia" prefix="gia"%>
<gia:html>
<head>
<title> Template Comercial </title>
</head>
<body>
<table width="100%" height="100%" border="1">
<tr>
 <td colspan="3">
          <gia:regiao400 id="regiao400" seq="1" comentario="Titulo">
           <gia:regiao300 id="regiao300" seq="1" comentario="Titulo">
           <gia:componente tipo="IMG"
           propriedade="SRC=imagens/logo_do_site.jpg;ALIGN=middle"/>
            <gia:regiao250 id="regiao250" seq="1" comentario="Titulo">
             <gia:regiao200 id="regiao200" seq="1" comentario="Titulo">
              <gia:regiao150 id="regiao150" seq="1" comentario="Titulo">
               <gia:regiao100 id="regiao100" seq="1" comentario="Titulo">
                    <font color="#006666" size="2" face="Verdana, Arial, Helvetica, sans-
                    serif">
                    <strong> GERADOR DE INTERFACES ADAPTATIVAS </strong>
                    </font>
               </gia:regiao100>
              </gia:regiao150>
             </gia:regiao200>
            </gia:regiao250>
          </gia:regiao300>
         </gia:regiao400>
 </td>
```

FIGURA 6.14 - Código do Cabeçalho do Template Comercial

O próximo capítulo apresenta o ambiente de desenvolvimento GIA, que tem como objetivo integrar o reconhecimento de dispositivos móveis em tempo de execução e o desenvolvimento de interfaces, utilizando a metodologia descrita nesse capítulo, bem

como o gerenciador de recursos do sistema. Ainda no capítulo 7 são apresentados os diagramas de classe referentes à arquitetura GIA, bem como detalhes referentes à implementação do ambiente proposto de desenvolvimento.

CAPÍTULO 7

AMBIENTE DE DESENVOLVIMENTO PARA A GERAÇÃO DE INTERFACES ADAPTATIVAS

Nesse capítulo apresenta-se o ambiente de desenvolvimento GIA com o objetivo de validar a proposta desta tese. São descritas as estratégias utilizadas para a implantação da arquitetura GIA, ou seja, a metodologia adotada para a construção dos serviços propostos no capítulo 5 desse trabalho. A seguir, essas estratégias são sumarizadas:

1. Inicialmente, foram estabelecidos modelos independentes para a arquitetura GIA. Essa etapa compreendeu a construção de diagramas de casos de uso para o sistema, a construção de um diagrama de pacotes para o sistema e a construção de diagramas de classes representando as classes do domínio do problema para o sistema;

2. Construídos os modelos para o sistema, cada serviço que compõe a arquitetura foi detalhado sob o ponto de vista de seu funcionamento. Para cada serviço foi construído um modelo representado por um diagrama de classes para aquele serviço;

3. Em seguida, foram estabelecidas as formas de comunicação e integração da arquitetura GIA por meio do *framework* DELI e o repositório de perfis CC/PP;

4. Para a validação da arquitetura GIA, implementou-se um ambiente de desenvolvimento, com o objetivo de integrar o reconhecimento de dispositivos móveis em tempo de execução e o desenvolvimento de interfaces utilizando a metodologia proposta no capítulo anterior. Para facilitar o processo de desenvolvimento, formas de visualização do código da interface em diferentes

tamanhos de telas de dispositivos são disponibilizados no momento da implementação;

5. Aliado ao editor de interfaces, o gerenciador de recursos do sistema foi anexado ao ambiente GIA, com a função de cadastrar e atualizar dados do sistema como fabricante, *browser*, perfis de dispositivos, componentes e propriedades de interfaces, *tag-library*, entre outras;

6. O ambiente GIA foi implementado utilizando-se para o serviço de adaptação a linguagem de programação Java. Para o serviço de configuração e editoração a linguagem *Object Pascal* por meio do *Borland Delphi* 7.0 e o banco de dados relacional *PostgreSql* 8.1.4.2;

7. Informações sobre os componentes foram inseridas pelo gerenciador de recursos para que testes pudessem ser realizados, como apresenta o capítulo 8. Por meio das informações cadastradas pelo gerenciador é possível fazer um mapeamento entre componentes, propriedades com seus respectivos *browsers* e dispositivos. Posteriormente, a interface é transformada pela arquitetura na linguagem suportada pelo dispositivo móvel, que está cadastrado previamente no repositório de tags e componentes, como exemplo, HTML, XHTML ou WML, garantindo, dessa forma, a adaptação em múltiplos *browsers*;

Na Figura 7.1 pode-se ter uma visão geral da arquitetura proposta pelo diagrama de pacotes, no qual cada serviço é visto como um pacote. Os inter-relacionamentos entre esses módulos são evidenciados pelas linhas tracejadas.

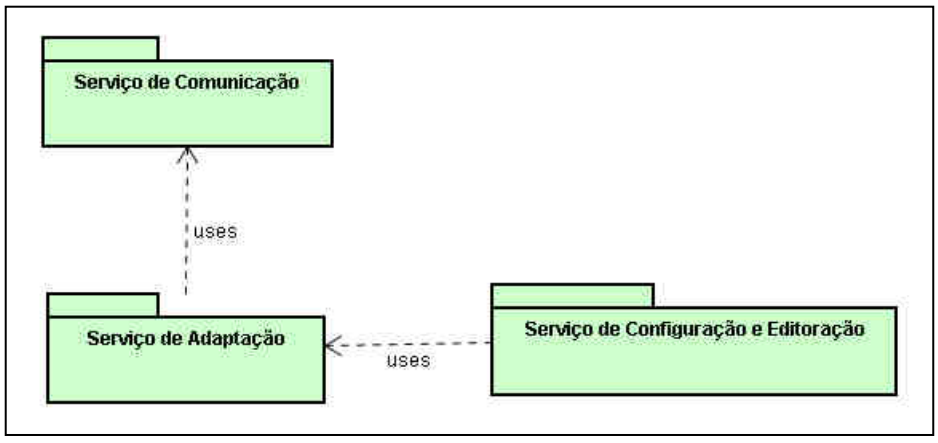


FIGURA 7.1- Diagrama de pacotes da arquitetura GIA

As classes apresentadas nos diagramas a seguir compõem os pacotes que representam os serviços apresentados no capítulo 5. São eles: serviço de comunicação, serviço de adaptação e serviço de configuração e editoração.

7.1 Serviço de Comunicação

A Figura 7.2, ilustra o diagrama de classes do pacote do serviço de comunicação. Fazem parte desse diagrama três classes: Dbase, Tdeli e Tbrowser, descritas a seguir:

- **DBase:** a classe DBase contém as configurações da conexão com o banco de dados;
- **Tdeli**: a Tdeli é uma classe em *servlet* responsável por interceptar a conexão realizada pelo usuário com o objetivo de identificar as características do dispositivo;
- **TBrowser**: essa classe armazena informações sobre o tipo do *browser* do cliente, capturada por meio do *user-agent* (conforme capítulo 3);

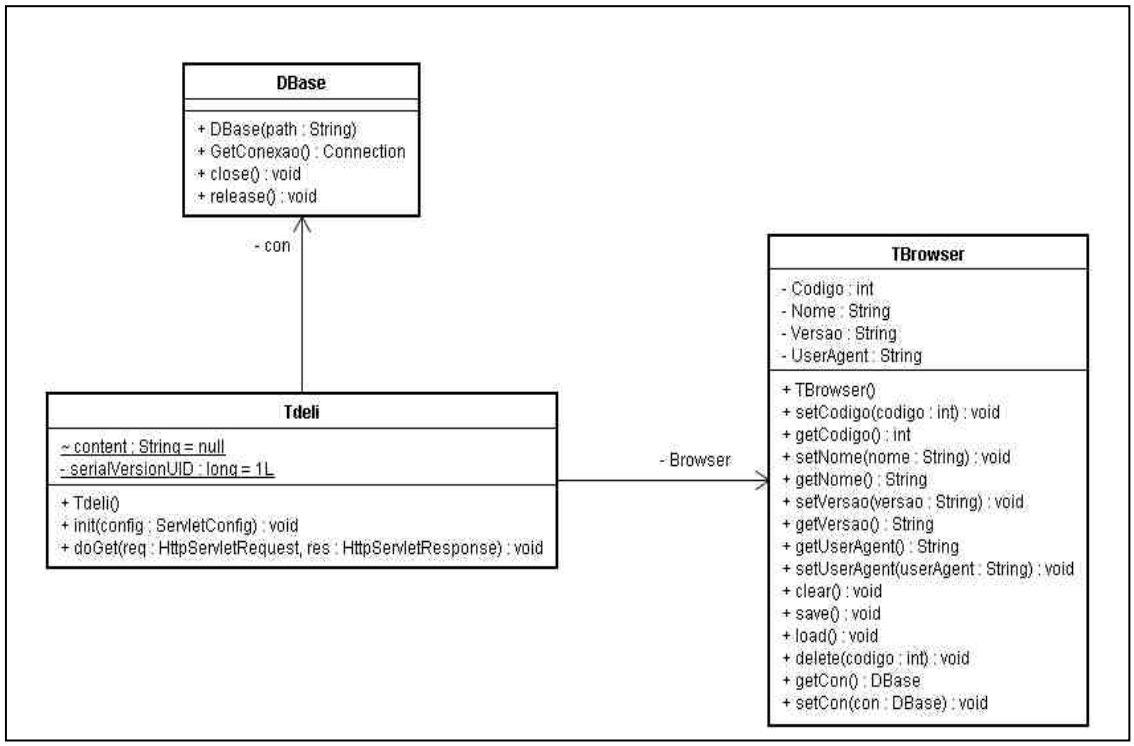


FIGURA 7.2 - Diagrama de Classes do Serviço de Comunicação

Todas as classes referentes ao serviço de comunicação são detalhadas no apêndice C.

7.2 Serviço de Adaptação

Pela Figura 7.3 pode-se visualizar o diagrama de classes referente ao serviço de adaptação. As classes que fazem parte desse serviço são descritas a seguir:

- **THtml:** a classe THtml realiza o gerenciamento da arquitetura e informa a necessidade de exibir ou não o menu com as descrições das regiões para o dispositivo;

- **TQuery:** os dados são capturados por meio da classe TQuery, utilizada para manipular o banco de dados. A conexão é realizada pela classe TConexão;

- **TComponente:** a classe TComponente é responsável por transformar as *tags* genéricas cadastradas no banco de dados em componentes reconhecidos pelos dispositivo que acessam a aplicação em tempo de execução;

- **TRegiao:** a classe TRegiao é responsável por gerar as regiões por meio do nível de prioridade indicado pelo atributo *seq* e também pelo tamanho, que pode variar de 100 a 500 *pixels*, de acordo com a metodologia proposta no capítulo 6;

- **TPerfil:** a classe TPerfil é responsável por manter as linguagens permitidas pelos *browsers*, como por exemplo, HTML, XHTML, entre outras;

As classes Tfabricante, TBrowser e TDispositivo são detalhadas na próxima seção.

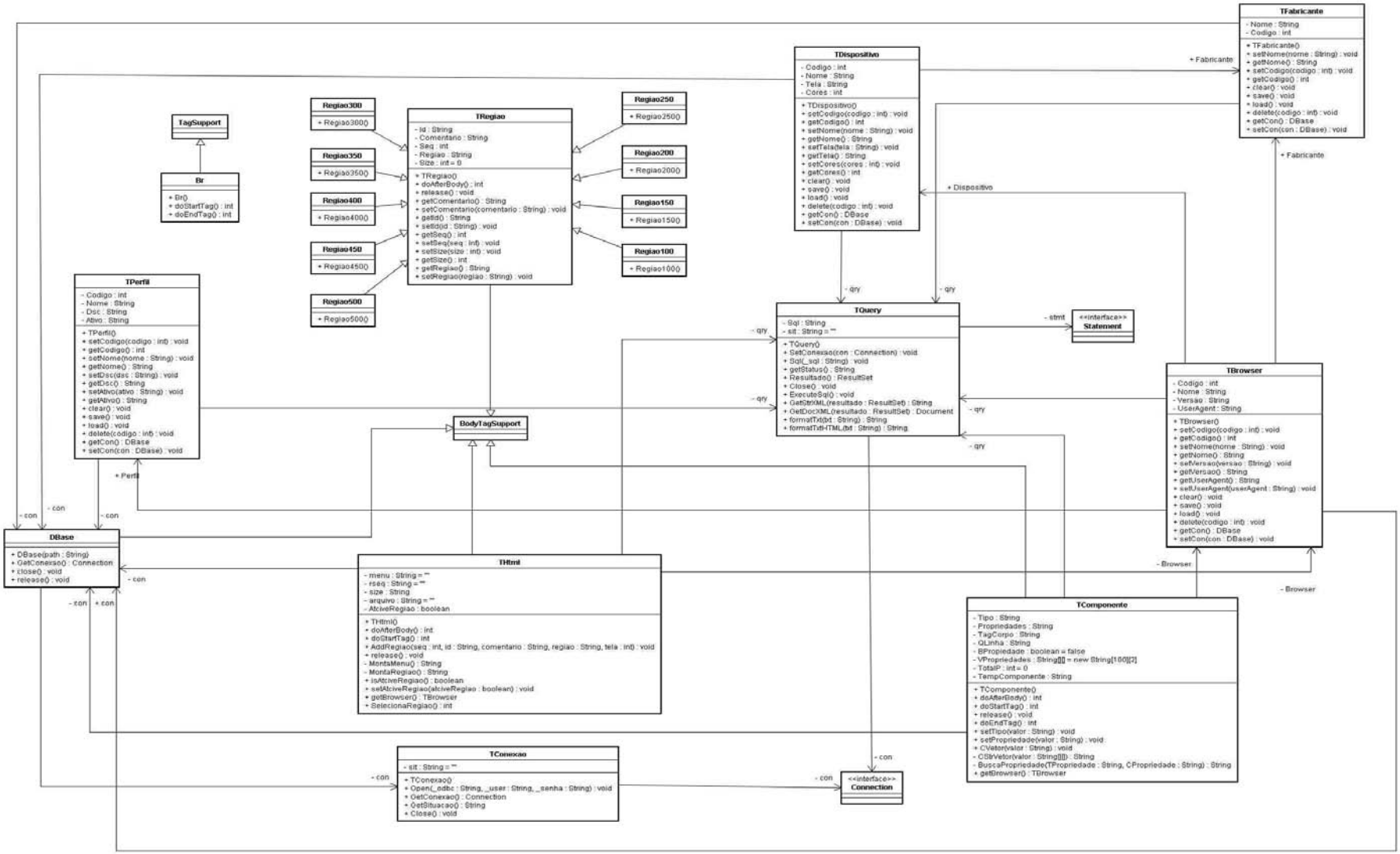


FIGURA 7.3 – Diagrama de Classes do Serviço de Adaptação

7.2.1 Classes Persistentes do Serviço de Adaptação

Nessa seção apresenta-se o diagrama de classes persistentes do serviço de adaptação da arquitetura GIA. Essas classes são gerenciadas pelo módulo do gerenciador dos recursos do sistema (detalhado na seção 7.4 desse capítulo), que tem a função de armazenar e recuperar dados fundamentais para proporcionar o processo de adaptação por meio do mapeamento de perfis cadastrados e relacioná-los com os componentes e propriedades dos dispositivos móveis. Todas as classes referentes ao serviço de comunicação são detalhadas no apêndice C.

O diagrama da Figura 7.4 é composto por 9 classes persistentes, que se relacionam entre si, exceto a do simulador que é isolada. Abaixo serão descritas as características de cada classe:

- **Fabricante:** a classe fabricante armazena as informações sobre o nome de fabricantes disponíveis no mercado, como exemplo, Nokia, Palmone, HP, entre outras;

- **Dispositivo:** essa classe armazena as informações sobre dispositivos existentes, tais como: nome, fabricante, tamanho da tela e quantidade de cores suportada;

- **Simulador:** a classe simulador contém as informações com o nome e o caminho físico em disco, do executável do simulador. Durante o processo de codificação da interface, o desenvolvedor poderá visualizar a parte já implementada em um simulador de dispositivo móvel, usando as informações cadastradas nessa classe para fazer a chamada de tal utilitário;

- ***Browser:*** essa classe registra as informações sobre os *browsers*, tais como: nome, versão, dispositivo, fabricante e a qual perfil está relacionado. Esses dados são utilizados para que a arquitetura GIA reconheça as características do *browser* do dispositivo solicitante e possa gerar a interface de acordo com a linguagem por ele aceita;

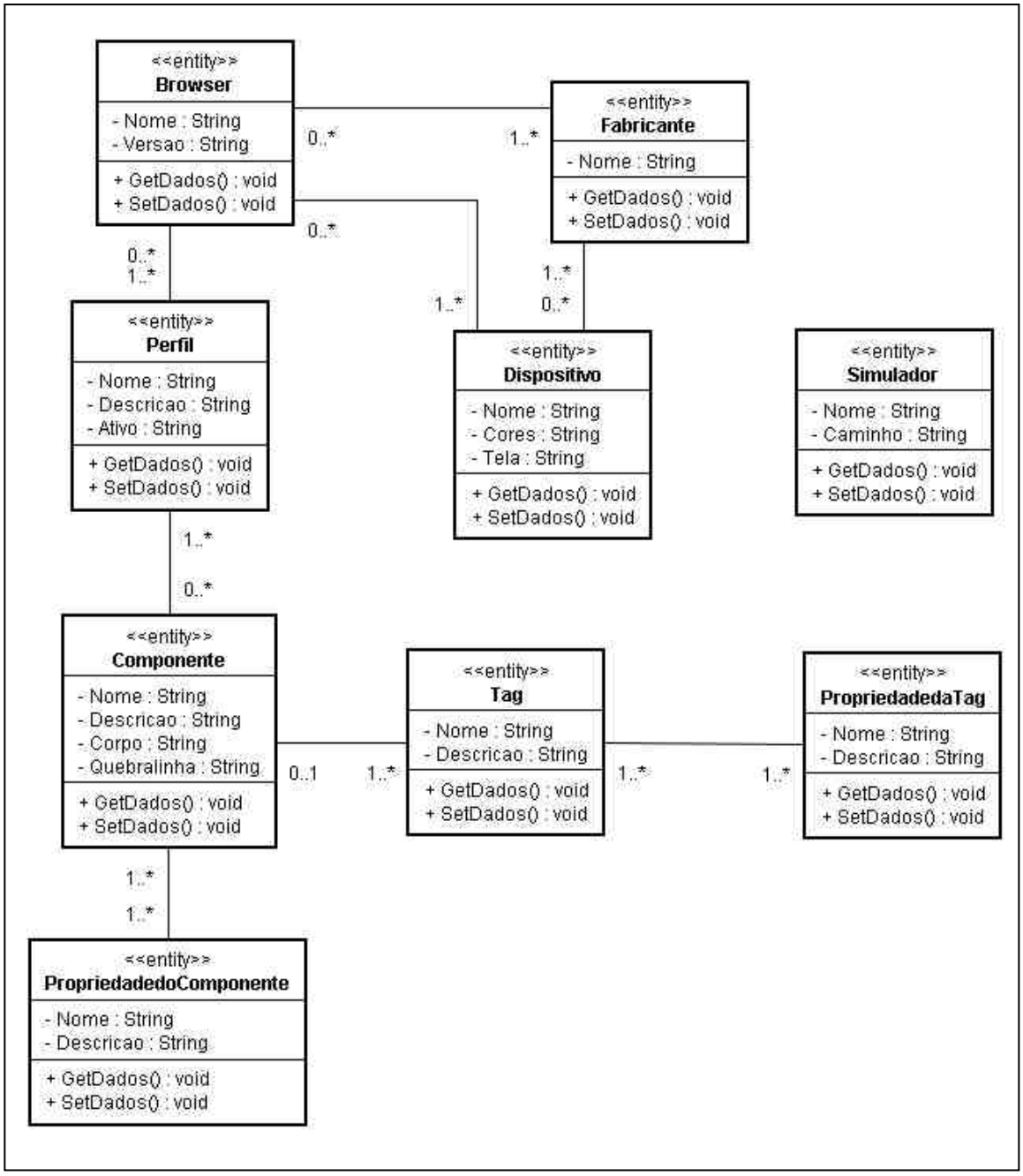


FIGURA 7.4 - Diagrama de Classes Persistentes

- **Perfil:** identifica o tipo de linguagem de programação aceita pelo *browser*. Nessa classe, estão armazenadas as informações sobre o nome do perfil (por exemplo, HTML ou WML), uma descrição sobre esse perfil e se ele está ou não ativo;

- ***Tag*:** essa classe contém as informações sobre os elementos que compõem a linguagem HTML, como por exemplo, *img*, *font*, *table*, entre outras;

- **Tpropriedade:** a classe Tpropriedade armazena as informações sobre os atributos da linguagem HTML, como por exemplo, *href*, *size*, *color*, entre outras;

- ***Tag*xTPropriedade*:*** nessa classe, será efetuada a associação entre os dados da classe *tag* com Tpropriedade, por exemplo, o elemento *font* que possui os atributos *size* e *color*. Essas informações serão utilizadas para a codificação da interface por meio da ferramenta GIA;

- **Componente:** a classe componente armazena as informações sobre os elementos de várias linguagens de programação para a Web, por exemplo: a linguagem HTML possui os elementos *body*, *br* e *div*; a linguagem WML possui os elementos *head*, *card* e *anchor*;

- **Cpropriedade**: essa classe contém as informações sobre os atributos de diversas linguagens de programação para Web, por exemplo: a linguagem HTML possui os atributos *class, style e title*; a linguagem WML possui os atributos *name, type e title*;

- **ComponentexCPropriedade:** nessa classe será efetuada a associação entre os dados de componente com Cpropriedade, ou seja, o relacionamento de um elemento com sua propriedade. Para cada associação será definida a equivalência com a correspondente no código da interface. Quando a arquitetura GIA identifica a linguagem do dispositivo, é necessário que o código da interface seja convertido para o perfil que o *browser* do dispositivo móvel suporte, ou seja, caso seja XHTML em tempo de execução, a arquitetura converterá a interface descrita em HTML para a linguagem XHTML, por meio da equivalência que foi definida no banco.

7.3 Serviço de Configuração e Editoração

O serviço de configuração e editoração disponibiliza, para os objetos persistentes da aplicação, os serviços de acesso ao banco de dados e à base de configurações (armazenamento, recuperação e exclusão), bem como a edição das interfaces detalhada na seção 7.6. É composto por dois módulos: editor de interfaces e gerenciador de recursos.

7.3.1 Editor para Criação de Interfaces

A linguagem a ser utilizada pelo *designer* para implementação da interface original no ambiente GIA é a *Java Server Pages* (JSP), por meio da metodologia de desenvolvimento proposta no capítulo 6. Para melhor visualização das regiões o editor permite visualizar as interfaces fragmentas em diversos tamanhos, a partir de 100 *pixels* até a interface completa. A Figura 7.5 apresenta o *layout* gráfico do editor.

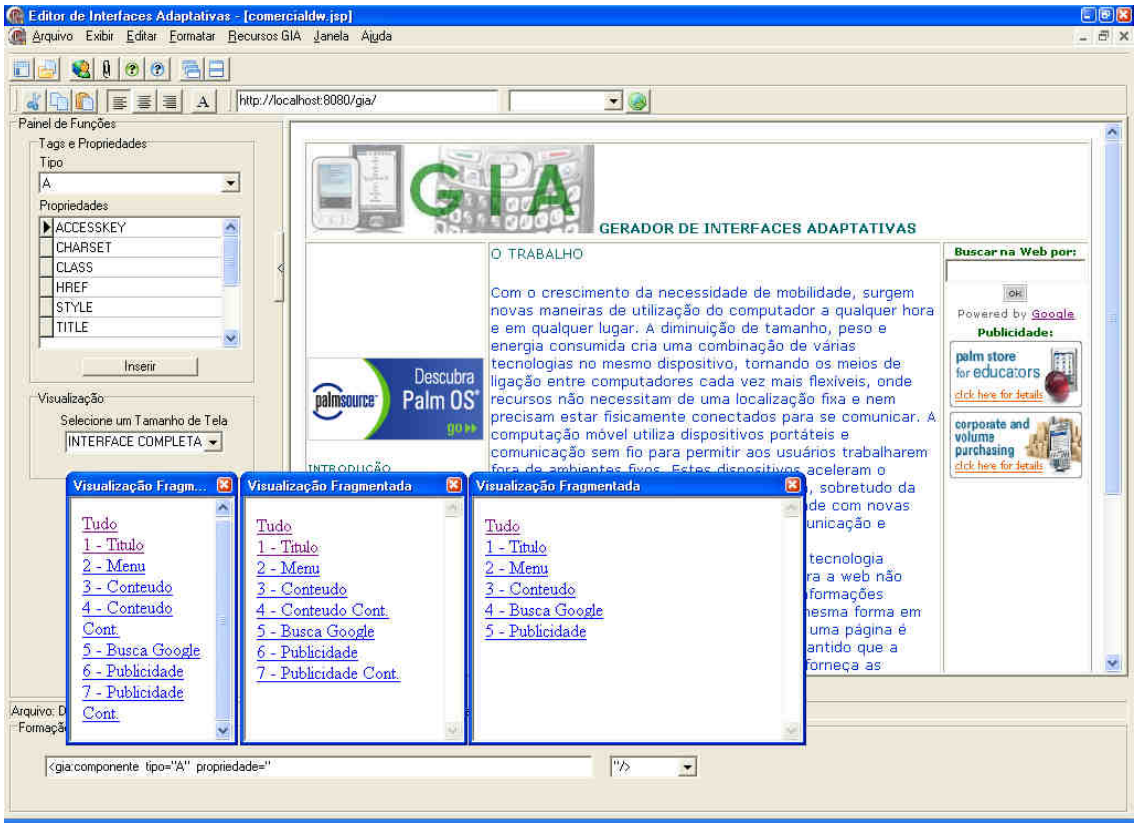


FIGURA 7.5 - Ambiente de Desenvolvimento GIA

Para possibilitar o uso de interfaces *Web* na ferramenta GIA, foram desenvolvidos dois *templates* e disponibilizados por meio do editor de interfaces para maximizar a adaptação e propiciar uma maior usabilidade, conforme detalhados no capítulo 6.

Para a construção de uma nova interface, o desenvolvedor pode optar pela escolha de três tipos de modelos: nenhum, simples ou comercial, como mostra a Figura 7.6. Ao escolher a opção *Nenhum* o desenvolvedor poderá criar seu código livremente por meio de *tags*, utilizando a metodologia de delimitação de regiões.

Ao escolher um dos *templates* (comercial ou simples), o desenvolvedor tem modelos de interface prontos, os quais têm opção de editá-los e visualizá-los de forma completa ou fragmentada em diversos tamanhos de tela, conforme seção 6.3 do

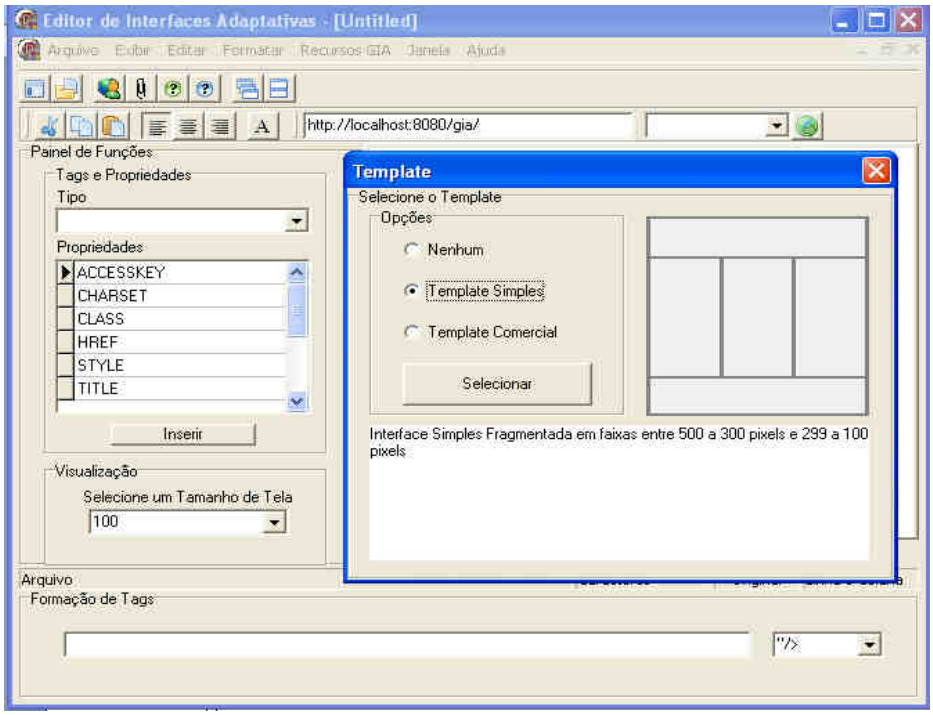


FIGURA 7.6 - Opções de Desenvolvimento da Interface

capítulo 6.

7.3.2 Painel de Funções e Menus

Após selecionar uma das opções de *template*, algumas opções de menus tornam-se disponíveis, como mostra a Figura 7.7. São elas:

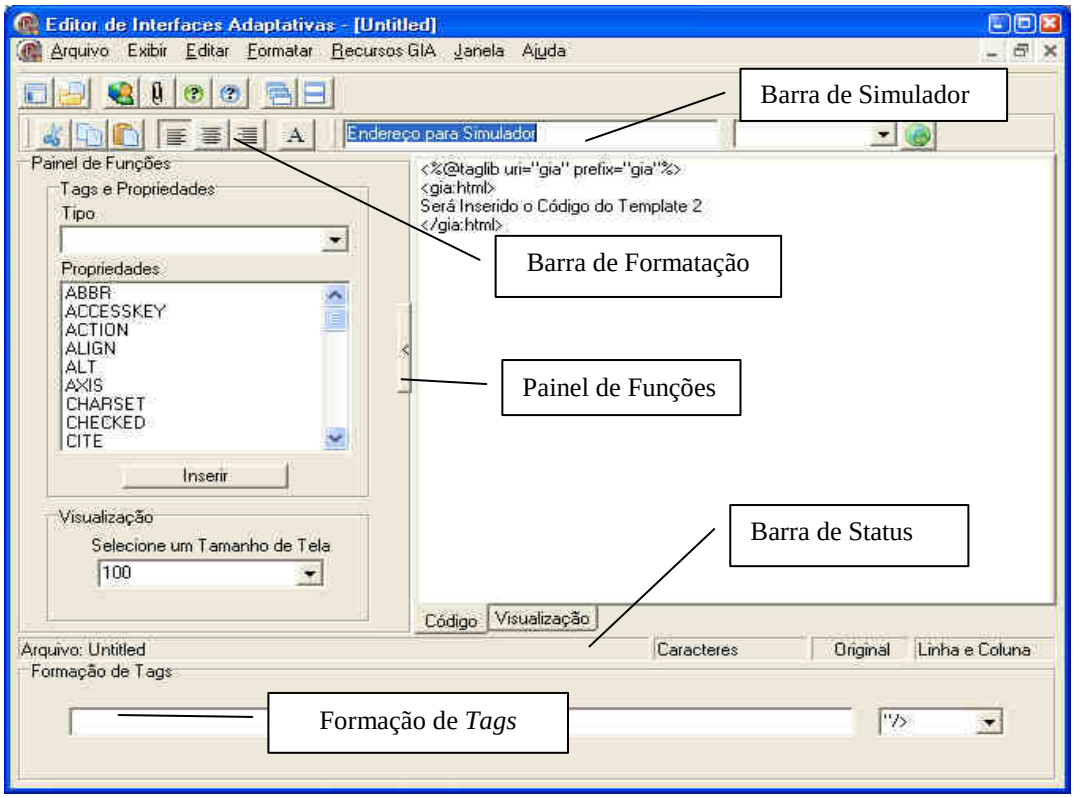


FIGURA 7.7 - Painel de Funções e Menus

- **Menu Arquivo**: serão adicionadas as opções Fechamento da interface ativa, Salvar, Salvar Como, Imprimir e Configurar Impressora;

- **Menu Exibir**: permite Ocultar/Exibir as Barras de Edição, Simulador, Funções, *Status* e *Tags;*

- **Menu Editar**: contém funções para trabalho com a estrutura. As funções de Recortar, Colar, Copiar, Apagar e Selecionar Tudo;

- **Menu Formatar**: contém funções de texto, como Alinhar à Esquerda, Alinhar à Direita, Centralizar, Quebra de Linha e propriedades de Fonte;

- **Menu Janela**: é adicionado o apontamento da janela ativa, podendo alternar entre vários projetos durante o trabalho;

Após a seleção dos *templates*, juntamente com os menus, são incluídas novas barras de ferramentas, quais são:

- **Barra de Formatação**: contém as opções recortar, copiar e colar do menu editar e também do menu formatar os itens alinhar à esquerda, à direita, centralizar e fonte;

- **Barra de Simulador**: contém um campo onde é inserido o endereço do arquivo de interface que está sendo editado e um campo de seleção de simuladores cadastrados para que sejam usados na visualização;

- **Painel de Funções**: contém um campo de seleção de *tags* e um campo para seleção das propriedades e a seleção do tamanho de tela para visualização da interface;

- **Barra de Status**: exibe informações tais como nome do arquivo, quantidade de caracteres utilizados, se o arquivo foi modificado e em qual posição se encontra o cursor;

- **Formação de *Tags***: é formada uma linha de código onde o desenvolvedor completa com valores desejados. Ao selecionar o fechamento da linha, ela aparece na área de edição.

7.3.3 Coloração de Código

Para o trabalho com a estrutura de código, várias colorações de código são utilizadas pela ferramenta GIA, com as seguintes cores:

- Vermelho para a estrutura comum da sintaxe GIA;

- Verde para os parâmetros incluídos na sintaxe, como tipos e propriedades;

- Azul para o conteúdo externo às *tags*, como texto.

Exemplo:

<gia:componente tipo=”TIPO” propriedade=”PROPRIEDADE=1”>

Texto da Tag de Exemplo

</gia:componente>

7.3.4 Desenvolvimento da Interface

Para iniciar a construção de uma interface posiciona-se o cursor no código existente e em seguida selecionam-se as *tags* e propriedades pelo Painel de Funções. Pela Barra de Formação de *Tags*, o desenvolvedor preenche os valores das propriedades conforme desejado e seleciona o tipo de fechamento da linha, como mostra a Figura 7.8.

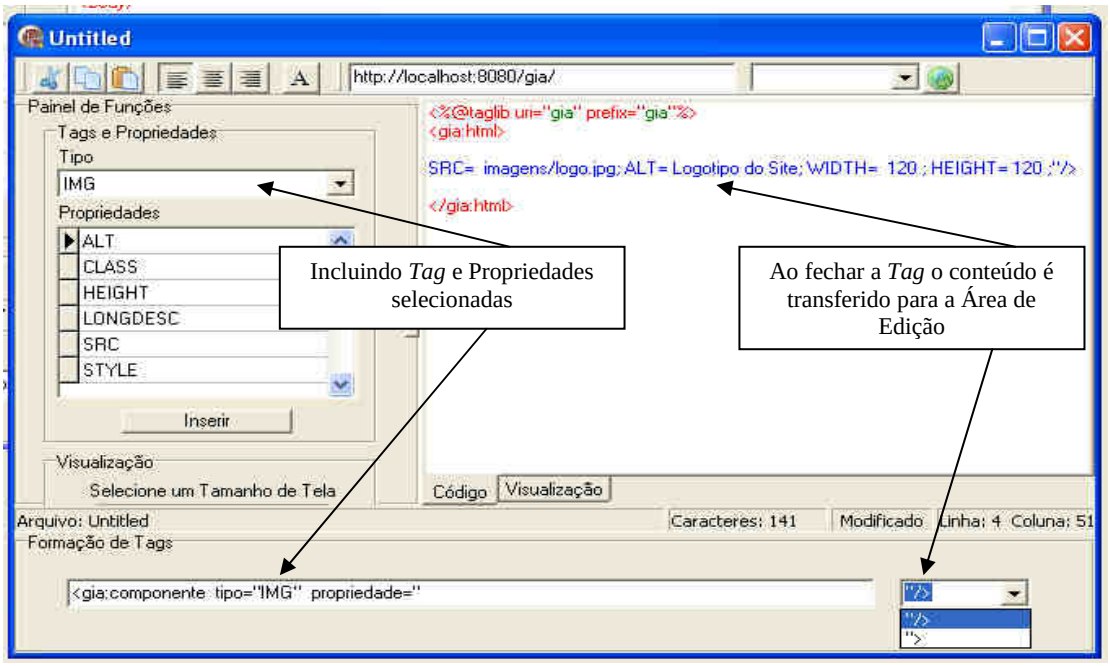


FIGURA 7.8 - Formação de *Tags*

No campo Formação de *Tags*, o fechamento deve ser efetuado conforme a sua funcionalidade. Para *tags* que contenham conteúdo adicional como textos, é utilizado o fechamento **“>** (sem a barra), o que implica que será feito em outra *Tag*, conforme o exemplo:

<gia:componente tipo=”A” propriedade=”href=index.jsp”>

Esse Texto é envolvido pela TAG

</gia:componente>

Para *tags* que não envolvem informações externas, utilizam o fechamento **“/>** (coma barra) como o exemplo:

<gia:componente tipo=”IMG” propriedade=”SRC=imagem.jpg”/>

7.3.5 Fragmentação da Interface em Regiões

Pelo uso do botão direito do mouse é realizada a fragmentação do código, selecionando-se o tamanho desejado para o trecho de código, conforme ilustra a Figura 7.9.

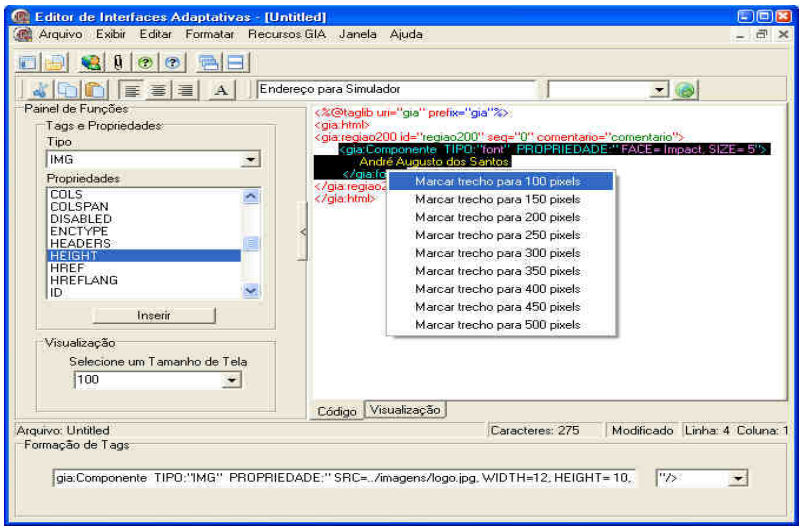


FIGURA 7.9 - Fragmentação da Interface em Regiões

7.3.6 Visualização da Interface

A interface pode ser visualizada por tamanho de região ou de forma completa. No modo completo deve-se selecionar pelo Painel de Funções o item visualização *Visualização Completa* ou, caso contrário, no tamanho desejado. Também poderá ser utilizada a aba de visualização junto à área de edição, que exibe a interface completa, como ilustra a Figura 7.10.

7.4 Gerenciador de Recursos do Sistema

A implementação do gerenciador de recursos da arquitetura GIA foi realizada utilizando-se a linguagem *Object Pascal* por meio do *Borland Delphi* 7, o banco de

dados relacional Postgresql 8.1.4.2, conforme o diagrama de classes persistentes da Figura 7.3. A opção *Recursos GIA*, proporciona o acesso ao gerenciador, referente aos cadastros e a opção para a geração de relatórios.

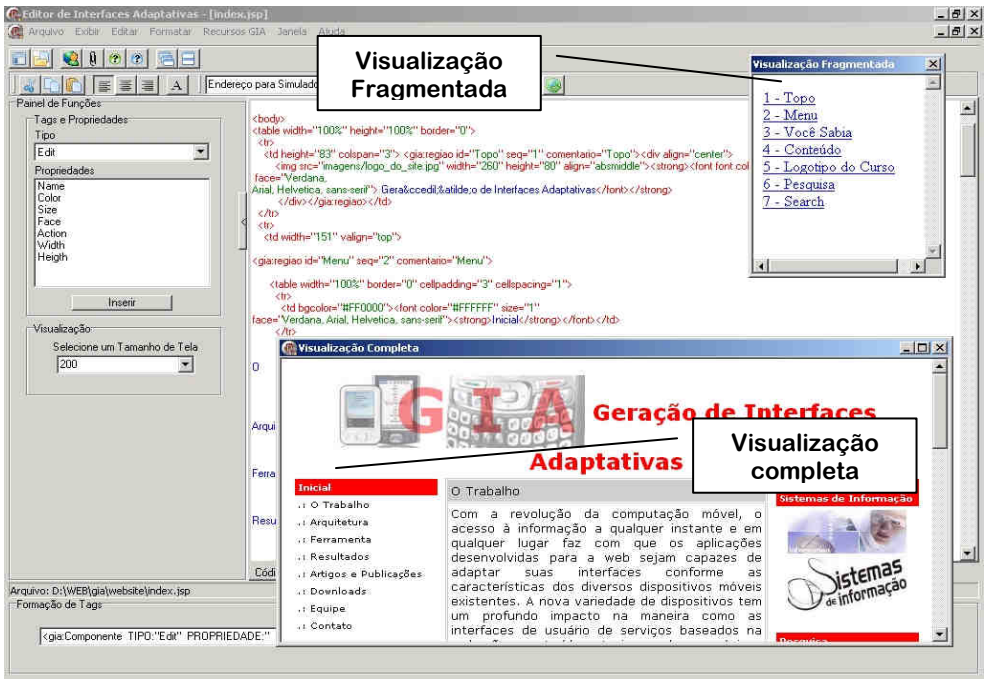


FIGURA 7.10 - Formas de Visualização da Interface

A Figura 7.11 ilustra o cadastramento de perfis, ou seja, o tipo de linguagem de programação aceito pelo *browser* dos dispositivos móveis.

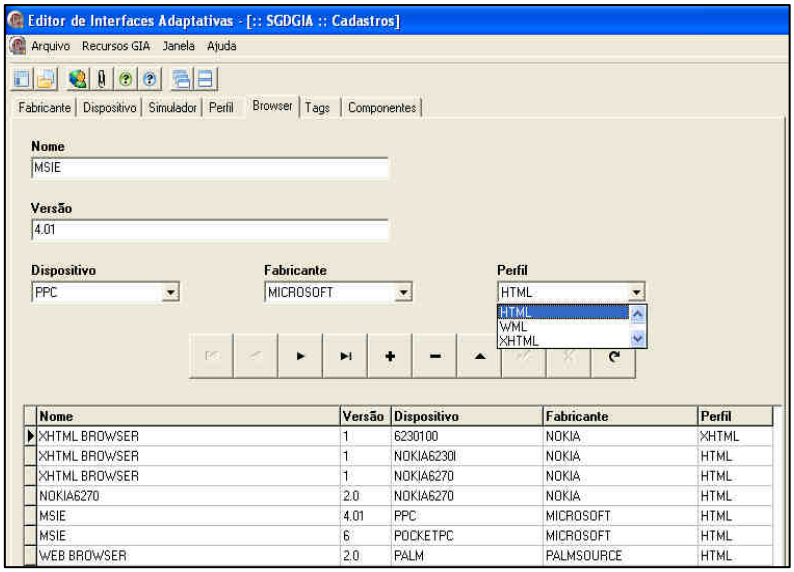


FIGURA 7.11- Cadastramento de Perfis

Após o cadastramento das *tags* e suas propriedades, deve-se fazer a associação entre ambas, como apresenta a Figura 7.12.

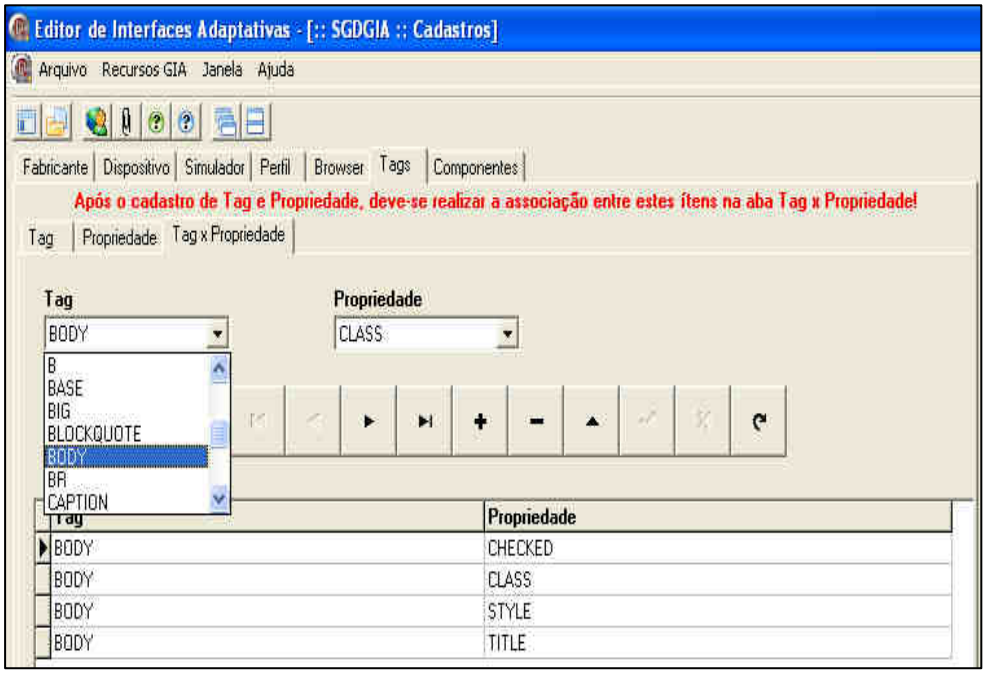


Figura 7.12 – Associação de Tags e Suas Propriedades

Analogamente após o cadastramento de componentes e suas propriedades, a associação entre ambos deve ser realizada, ou seja, o relacionamento de um elemento HTML com uma propriedade HTML, como ilustra a Figura 7.13.

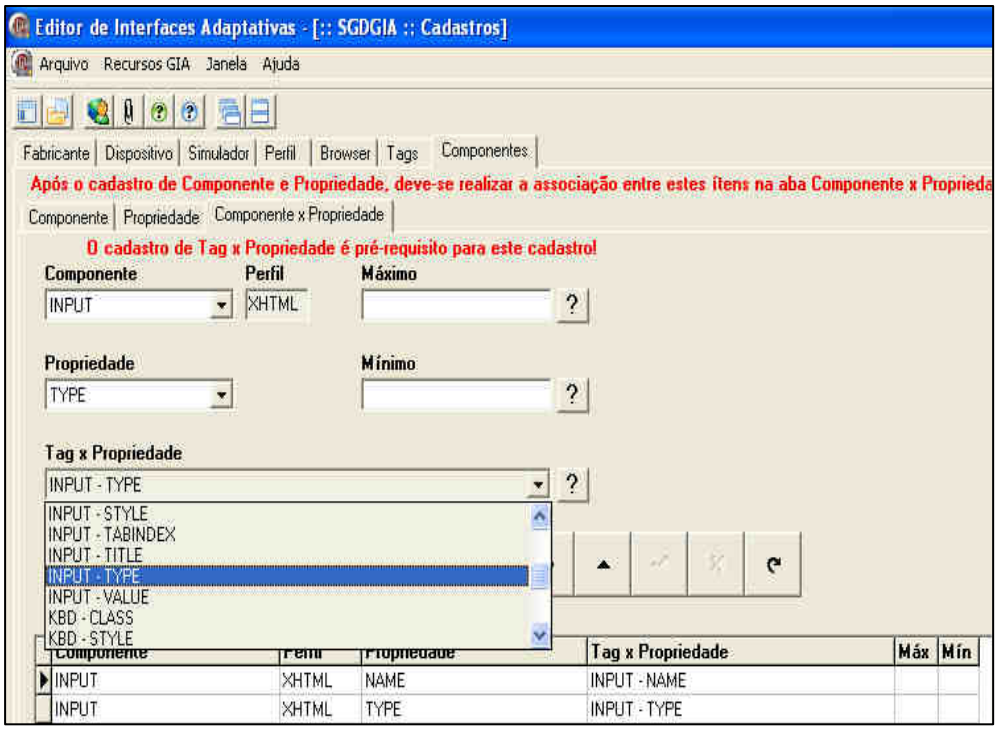


FIGURA 7.13– Associação de Componentes e suas Propriedades

CAPÍTULO 8

ESTUDO DE CASO

Para a validação da arquitetura GIA, bem como da metodologia proposta nesse trabalho, realizou-se uma análise por meio de simuladores de dispositivos móveis como telefone celular e PDAs. As interfaces utilizadas foram apresentadas no capítulo 6, intituladas *template* simples e *template* comercial. Ambas foram implementadas utilizando-se o ambiente de desenvolvimento proposto no capítulo 7.

Para a realização dos testes foi configurado o servidor Web Apache TomCat 5.5 e o banco de dados PostgresSql 8.1.4 com o sistema operacional Windows XP. Para acesso das interfaces na máquina local utilizou-se o navegador Internet Explorer 6.0 e, para testes em dispositivos móveis, os simuladores e aparelhos apresentados na tabela 8.1.

TABELA 8.1 – Perfil dos Simuladores de Dispositivos Móveis

	Modelo	**Categoria**	**Dimensão da tela**	**Sistema Operacional**	**Browser**	**Linguagem**
1	Palm OS Cobalt Simulator 6.0.1	PDA	320x320	Palm OS Cobalt 6.0.1	Web Browser	HTML/WML/ XHTML
2	Openwave Simulator 7.0	Celular	120x160	OPWV-SDK UP	Openwave Mobile Browser	WML/XHTML
3	Microsoft PocketPC 2003	PDA	320x240	Windows Mobile	Internet Explorer	HTML/XHTML
4	Nokia 6230i	Celular	208x208	Nokia OS	Nokia	XHTML
5	Nokia 6270	Celular	240x320	Nokia OS	Nokia Mobile Internet Client	XHTML
6	Palm zire 72	PDA	320x320	Palm OS Garnet 5.2.8	WebPro	HTML/XHTML

Os *templates* desenvolvidos exploram a proposta da arquitetura GIA contribuindo para maximizar a adaptação e propiciar uma maior usabilidade. O *template* simples é composto por cinco partes: cabeçalho, menu, conteúdo, complemento e rodapé. O

modelo é mais indicado para conteúdo textual, definido por poucas imagens, como descrito no capítulo 7.

O *template* comercial utiliza uma distribuição de imagens com tamanhos entre 250 a 120 *pixels* de largura, componentes de formulários como botão, caixa de texto e radio *button*, atendendo dispositivos com capacidade gráfica superior em termos de resolução.

No primeiro teste, o *template* simples foi acessado por meio do Internet Explorer 7.0, com resolução de 1024x768 *pixels*, como apresenta a Figura 8.1

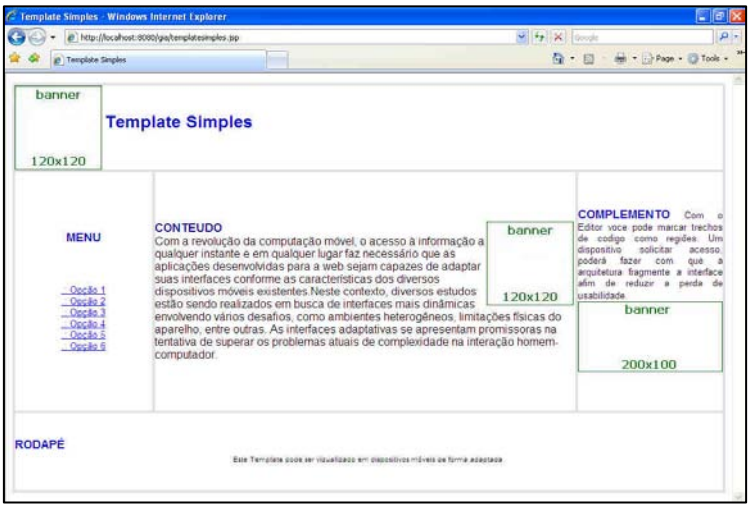
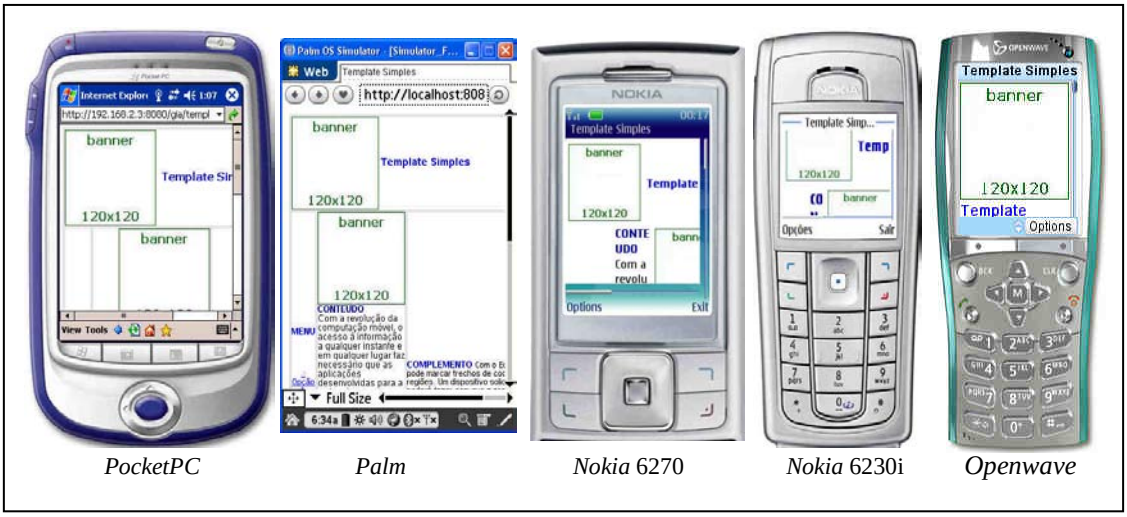


FIGURA 8.1 - *Template* Simples Acessado pelo Internet Explorer

Em seguida, o modelo citado foi acessado por simuladores, sem a utilização da arquitetura GIA. Os resultados são apresentados na Figura 8.2.



PocketPC *Palm* *Nokia* 6270 *Nokia* 6230i *Openwave*

FIGURA 8.2 - *Template* Simples acessado sem a Arquitetura GIA

Analogamente o mesmo processo foi realizado com o *template* comercial, apresentado nas Figuras 8.3 e 8.4.

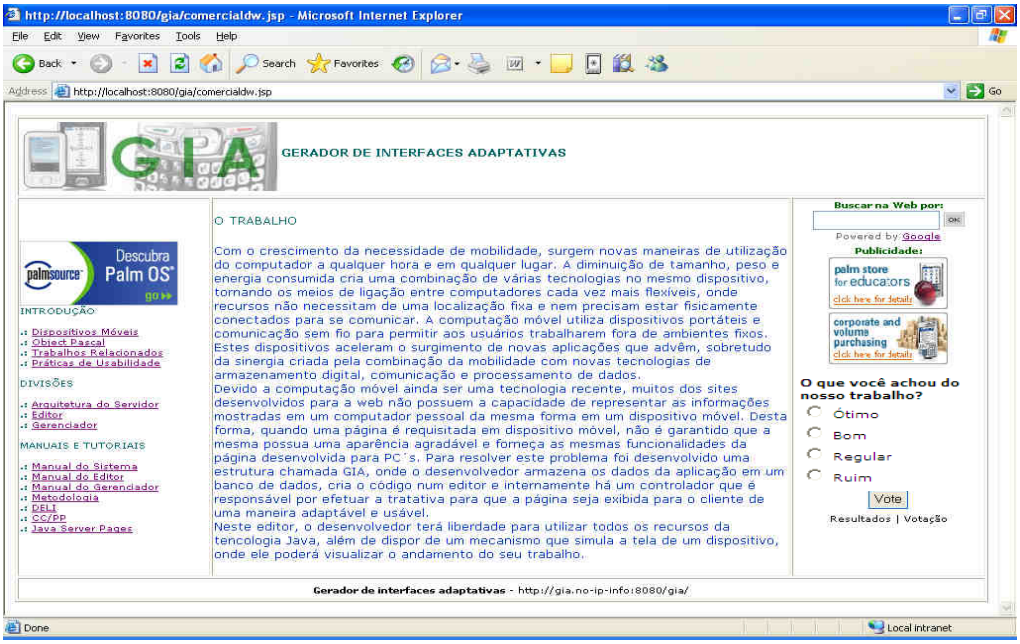


FIGURA 8.3 - *Template* Comercial Acessado pelo Internet Explorer

FIGURA 8.4 - *Template* Comercial acessado sem a Arquitetura GIA

Como a maioria dos *sites* é desenvolvida exclusivamente para *desktops*, a interface é exibida de forma desconfigurada em dispositivos com telas menores. A navegação se torna difícil e o usuário tem mais dificuldade em encontrar informações, uma vez que imagens e textos são visualizados de forma parcial utilizando-se a barra de rolagem. Esses problemas são comumente encontrados na maioria dos Web *sites* disponíveis na Internet devido, ao desenvolvimento voltado ao *desktop*.

8.1 Formas de Implementação da Interface

No capítulo 3, foram descritas as formas de exibição do *layout* original e reduzido. O primeiro exibe a página completa, ou seja, como foi projetada e o no segundo, a interface é apresentada em uma longa coluna vertical.

Exibir uma interface completa em um dispositivo móvel dificulta a navegação como citado anteriormente. Navegar em uma coluna vertical pode causar exaustão ao usuário, pois as páginas se tornam extensas demais.

Nesse trabalho foram propostas as formas de exibição de *layout* fragmentado e seleção de conteúdo, descritas no capítulo 3. Por meio da seleção e classificação de informações apenas o conteúdo principal do *site* será exibido. No *layout* fragmentado uma interface é dividida e apresentada em diversas partes, contendo menus que

permitem navegar entre uma tela e outra. A solução consiste em criar um *design* com o mínimo possível de conteúdo e elementos gráficos com hierarquias de navegação.

A arquitetura GIA proporciona ao desenvolvedor utilizar varias formas de exibição de *layout* e determinar qual se encaixa melhor para o projeto em questão. Durante a fase de desenvolvimento, torna-se necessário conhecer o perfil dos usuários que utilizarão a aplicação, pois, a partir dessas informações, pode-se delimitar qual a melhor forma para que a interface seja exibida.

8.1.1 Menu para a Navegação

A arquitetura GIA gera um menu no topo e no final de cada tela, com o objetivo de facilitar a navegação, como ilustra a Figura 8.5.

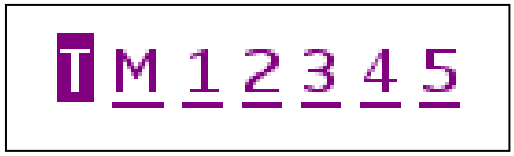


FIGURA 8.5 - Menu Gerado pela GIA

A opção **T** apresenta o *layout* reduzido, ou seja, adaptado ao tamanho da tela em uma longa coluna vertical. Essa opção é utilizada caso o usuário deseje ver o conteúdo completo do *site*. Pelo uso da letra **M** pode-se visualizar um menu de opções da interface fragmentada. As demais regiões são numeradas em ordem ascendente com seus respectivos títulos, definidos durante a fase de implementação.

A Figura 8.6 ilustra dois exemplos de menus gerados pela arquitetura GIA referentes à mesma interface, com 6 e 7 itens. O menu gerado com 6 opções não mostra a parte referente à publicidade, pois nesse caso não foi considerada relevante. Dependendo do tamanho da tela do dispositivo e da forma como foram fragmentadas as regiões, tem-se a quantidade de opções. Cabe ao desenvolvedor da aplicação determinar qual conteúdo será ou não exibido.

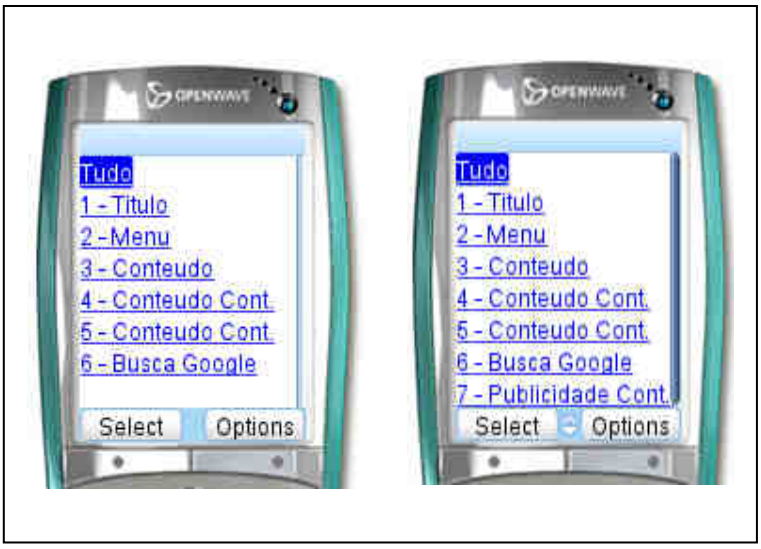


FIGURA 8.6 - Número de Itens do Menu GIA

8.2 Cenários do Estudo de Caso

Para a realização desse estudo de caso, têm-se dois cenários: No primeiro cenário a aplicação é desenvolvida desde o projeto até a implementação utilizando os serviços da arquitetura GIA. Deve-se considerar, nessa fase, que uma interface Web que se propõe a funcionar bem em qualquer resolução é normalmente baseada em texto, com poucas imagens, enquanto que *sites* com *design* mais apurado, que usam imagens extensivamente, normalmente são projetados para uma resolução fixa.

O segundo cenário consiste em modificar uma aplicação Web existente, implementada nas linguagens HTML ou JSP. Nesse caso, é possível conseguir um nível de adaptação da interface em múltiplos dispositivos, por meio de um processo de reengenharia em que a metodologia de fragmentação de regiões é aplicada e validada pela arquitetura GIA.

Logo, quanto mais complexo o requisito de *design*, menor a adaptabilidade. Caso o *designer* queira uma interface mais elaborada, deve fazê-lo para cada um dos tamanhos de tela desejados. Assim, na fase de projeto, devem-se considerar vários aspectos, inclusive que a interface será tão flexível a ponto de ser exibida em diversos tipos de dispositivos, inclusive no *desktop*.

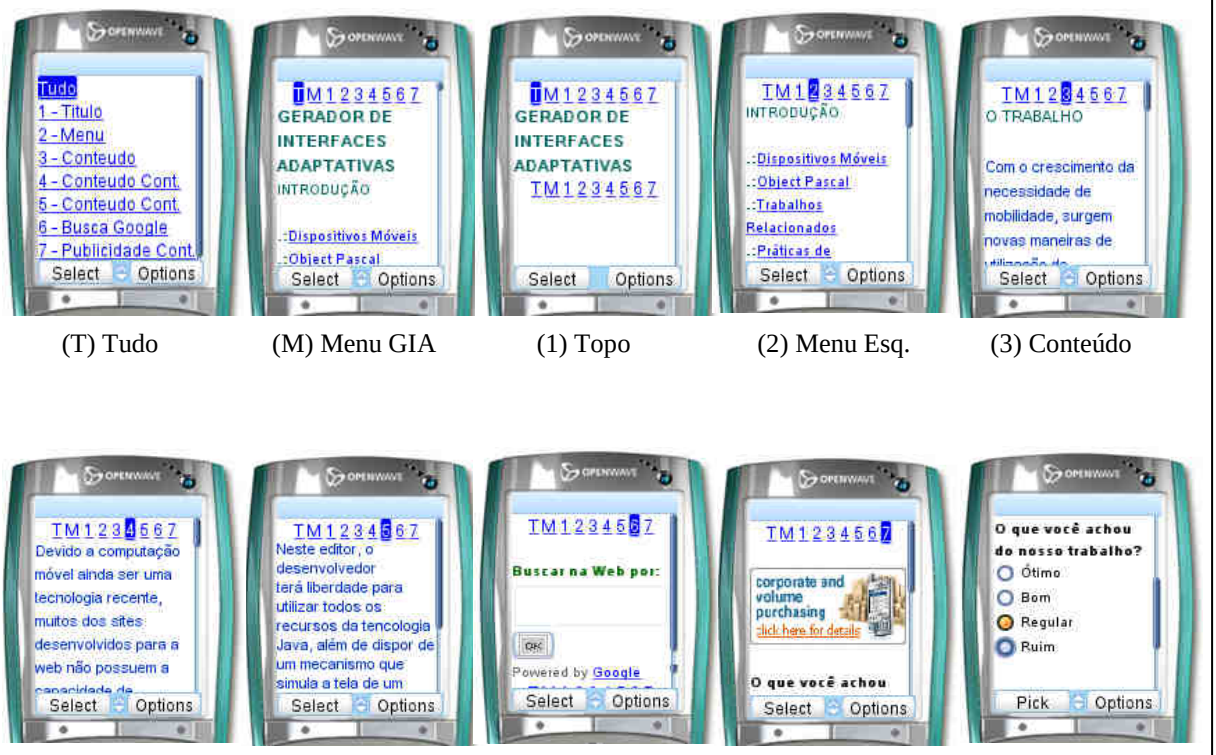
8.2.1 Cenário 1: Do Projeto a Implementação

Para iniciar a implementação de uma interface adaptativa por meio da arquitetura GIA, primeiramente o *designer* deve criar um projeto da interface de forma que sejam analisados vários aspectos, dentre eles:

- Realizar uma pesquisa sobre as classes de dispositivos que acessarão a aplicação, envolvendo o *browser*, tamanho de tela, entre outras;
- A equipe de desenvolvimento deve conhecer as linguagens JSP, bem como a metodologia de fragmentação de código proposta nessa tese;
- O projeto deve ser flexível, pois será visualizado em múltiplos dispositivos, inclusive *desktops;*
- Posteriormente à realização do projeto da aplicação, o desenvolvedor deve analisar em quantas regiões uma interface deve ser fragmentada de forma a preservar a sua usabilidade;
- Durante a implementação da interface no ambiente de desenvolvimento GIA, o *designer* pode visualizar a interface fragmentada em diversos tamanhos de tela e, caso seja necessário, o número de regiões pode ser alterado;

Após a definição dessas etapas, pode-se iniciar o processo de implementação no ambiente de desenvolvimento (descrito no capítulo 7). A fim de exemplificar esse cenário, o *template* c*omercial* ilustrado na Figura 8.3 foi dividido em 6 faixas de exibição, variando entre 100 a 500 *pixels*. O código completo dessa interface é apresentado no Apêndice B.

A Figura 8.7 ilustra o *template* comercial acessado do *Openwave Phone Simulator*. A opção *Tudo* mostra a interface reduzida em uma única coluna vertical. A opção *Menu GIA* exibe o menu gerado pela arquitetura e as demais partes são as regiões que foram delimitadas pelo desenvolvedor.



(T) Tudo (M) Menu GIA (1) Topo (2) Menu Esq. (3) Conteúdo

(4)Conteúdo Cont. (5)Conteúdo cont. (6)Busca Google (7)Publicidade (7)Publicidade

FIGURA 8.7 - Interface Comercial visualizada no *Openwave Phone Simulator*

O mesmo processo foi realizado para o acesso ao simulador do PocketPC, como apresentado na Figura 8.8.

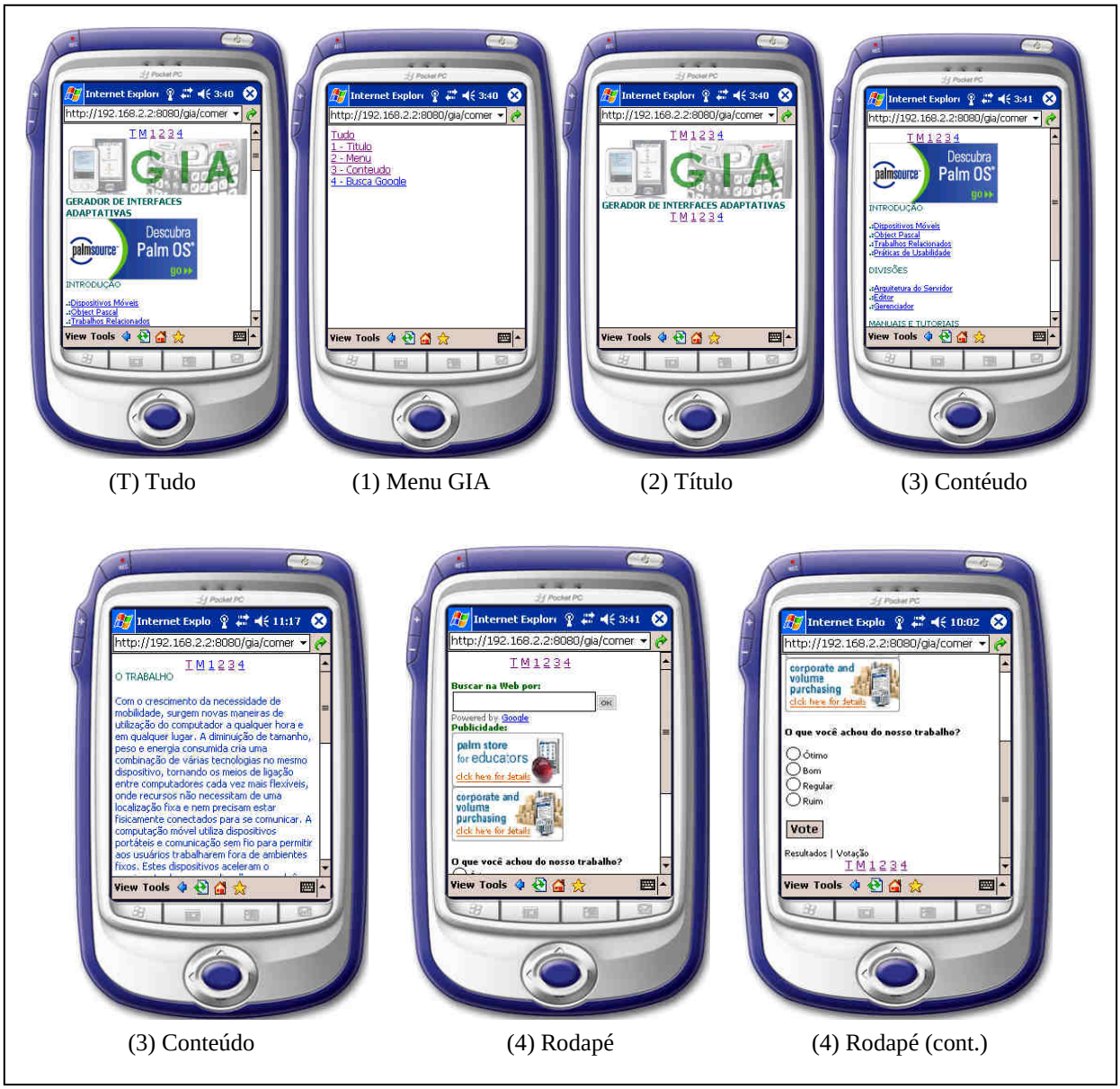


FIGURA 8.8- Interface Comercial visualizada no PocketPC

8.2.2 Cenário 2: Solução Proposta para uma Interface Existente

Para exemplificar o cenário em que uma interface já esteja implementada, mas que não seja adaptativa torna-se necessário que seja analisado o uso de imagens, textos e componentes, como detalhado na seção 1.4.1 do capítulo 2, bem como os aspectos citados na seção anterior, para em seguida aplicar a fragmentação do código em regiões. A Figura 8.9 apresenta o código HTML do *template* simples.

```
<html> <head>
<title> Template Simples </title> </head>
<body>
<table width="100%" height="100%" border="1" cellpadding="0" cellspacing="1"
bordercolor="#CCCCCC"> <tr>
<td height="20%" colspan="3">
    <img src="imagens/img_120x120.jpg" alt="logotipo do site" width="120" height="120" align="middle">
    <font color="#0099CC" size="5" face="Geneva, Arial, Helvetica, sans-serif"><strong> Titulo do Site ou
      Página</strong></font> </td>
<td width="20%" height="60%">
                  <font color="#0099CC" size="1" face="Geneva, Arial, Helvetica, sans-serif">
                  <strong>Menu</strong><br><br>
                     Opção 1 <br>
                     Opção 2 <br>
                     Opção 3 <br>
                     Opção 4 <br>
                     Opção 5 <br>
                     Opção 6 <br>
                  </font>
  </td>
  <td height="60%">
              <img src="imagens/img_120x10.jpg" width="120" height="100" align="right">
 <font color="#0099CC" size="1" face="Geneva, Arial, Helvetica, sans-serif">
      Com a revolução da computação móvel, o acesso à informação a qualquer instante e em qualquer lugar faz necessário que as
      aplicações desenvolvidas para a web sejam capazes de adaptar suas interfaces conforme as características dos diversos
      dispositivos móveis existentes.Nesse contexto, diversos estudos estão sendo realizados em busca de interfaces mais dinâmicas
      envolvendo vários desafios, como ambientes heterogêneos, limitações físicas do aparelho, entre outras. As interfaces
      adaptativas se apresentam promissoras na tentativa de superar os problemas atuais de complexidade na interação homem-
      computador.
</font>
</td>
<td width="20%" height="60%">
<p align="justify">
                  <font color="#0099CC" size="1" face="Geneva, Arial, Helvetica, sans-serif">
                  Com o Editor voce pode marcar trechos de codigo como regiões.
                  <img src="imagens/img_100x120.jpg" alt="logotipo do site" width="100" height="120"
align="left">
                  Um dispositivo solicitar acesso, poderá fazer com que a arquitetura fragmente
                  a interface a fim de reduzir a perda de usabilidade.</font> <br> </p></td>
  <td height="20%" colspan="3">
 <font color="#0099CC" size="1" face="Geneva, Arial, Helvetica, sans-serif"> <center>Esse Template pode
          ser   visualizado em dispositivos móveis de forma adaptada</center></font>
</td> </tr> </table> </body>
```

FIGURA 8.9 – Código HTML do Template Simples

Após o processo de reestruração da interface, aplica-se a metodologia de fragmentação do código, como descrito no capítulo 6. A quantidade de regiões para a qual a interface deve ser fragmentada depende dos tipos de dispositivos móveis que virão a acessá-la.

A Figura 8.10 apresenta o código do cabeçalho do *template* simples fragmentado em 4 regiões, permitindo duas faixas de largura em resoluções distribuídas entre 500 a 300 *pixels* e entre 299 a 100 *pixels*.

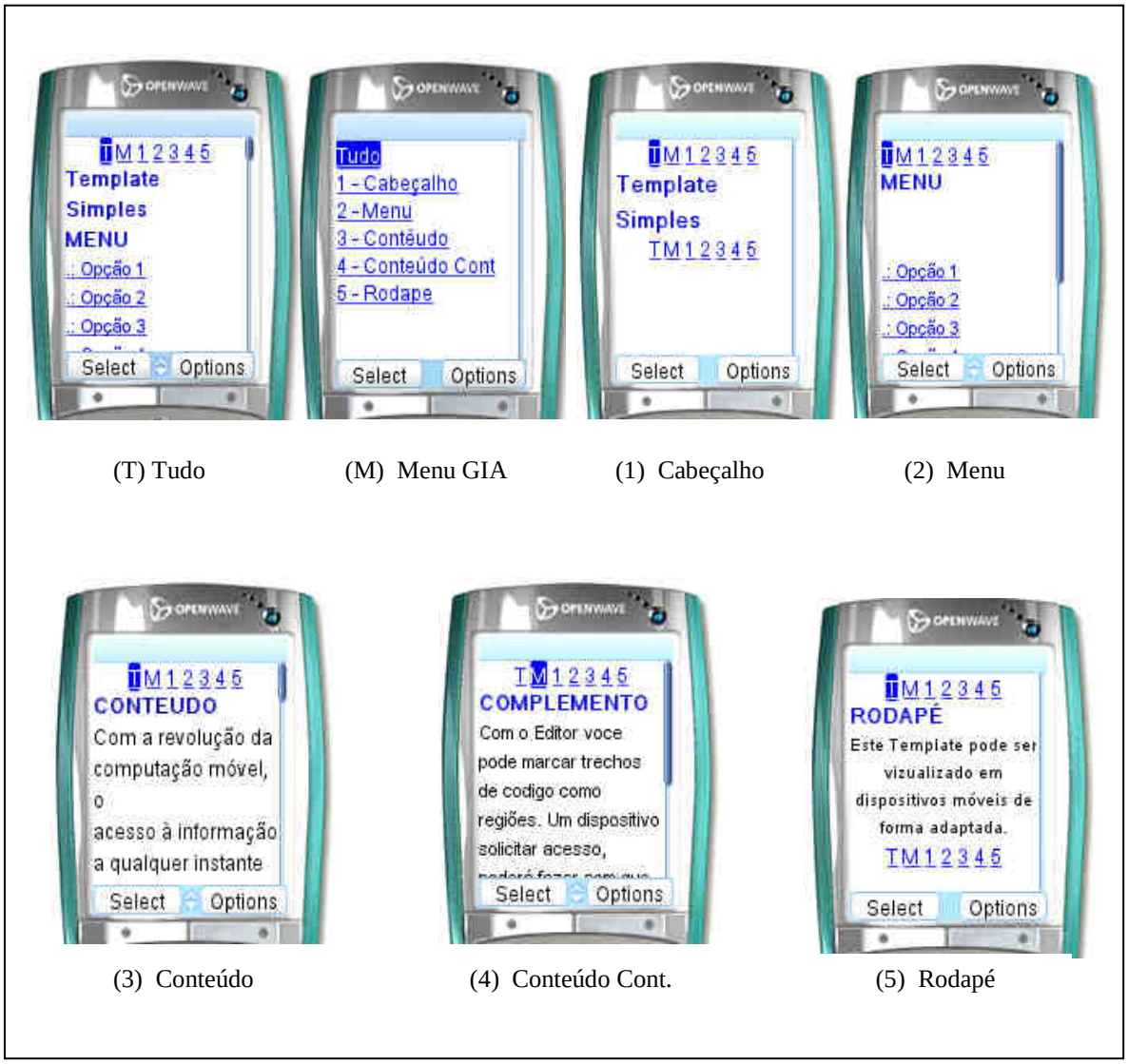
Dessa forma, o restante do código HTML deve ser delimitado pelas *tags* e seus respectivos tamanhos. O código completo é apresentado no Apêndice B.

```
<%@taglib uri="gia" prefix="gia"%>
<gia:html>
<head>
<title> Template Simples </title> </head>
<body>
<table width="100%" height="100%" border="1" cellpadding="0" cellspacing="1"
        bordercolor="#CCCCCC"> <tr>
<td height="20%" colspan="3">
gia:regiao500 id="Topo500" seq="1" comentario="Topo">
  <gia:regiao400 id="Topo400" seq="1" comentario="Topo">
    <gia:regiao300 id="Topo300" seq="1" comentario="Topo">
      <gia:regiao200 id="Topo200" seq="1" comentario="Topo">
        <img src="imagens/img_120x120.jpg" alt="logotipo do site" width="120"
            height="120" align="middle">
          <gia:regiao100 id="Topo100" seq="1" comentario="Topo">
            <img src="imagens/img_120x120.jpg" alt="logotipo do site" width="120"
                height="120"  align="middle">
            <font color="#0099CC" size="5" face="Geneva, Arial, Helvetica, sans-
                serif"><strong> Titulo do Site ou     Página</strong></font> </td>
          </gia:regiao100>
      </gia:regiao200>
    </gia:regiao300>
  </gia:regiao400>
</gia:regiao500>
```

FIGURA 8.10 – Fragmentação da Interface do Template Simples

Para demonstrar o processo de adaptação realizado pela arquitetura GIA, o *template* simples foi acessado pelos simuladores *Openwave Phone Simulator* 7.0, *Nokia*6270, *Palm Simulator* e *PocketPC*. Por meio do perfil CC/PP, obtiveram-se as informações necessárias para identificar o perfil dos dispositivos que acessaram a aplicação.

A Figura 8.11 ilustra a interface sem a exibição das imagens que fazem parte do *layout* original, devido ao tamanho da tela do dispositivo, conforme tabela 8.1.



(T) Tudo (M) Menu GIA (1) Cabeçalho (2) Menu

(3) Conteúdo (4) Conteúdo Cont. (5) Rodapé

FIGURA 8.11– *Template* Simples acessado pelo *Openwave Simulator*

O *template* simples acessado sem a utilização da arquitetura GIA, como ilustrado na figura 8.2, exibiu a interface sem nenhuma usabilidade para navegação. Já, pela Figura 8.9 percebe-se maior facilidade para encontrar o conteúdo por meio do menu GIA. O desenvolvedor pode optar por declarar um número maior ou menor de regiões comparado as implementadas nesse estudo de caso.

Analogamente, a Figura 8.12 ilustra a interface do *template* simples no simulador Nokia6270, sem a exibição das imagens.

(M) Menu Gia (1) Tudo (2) Cabeçalho (3) Menu

(3) Conteúdo (4) Conteúdo Cont. (5) Rodapé

FIGURA 8.12 - *Template* Simples acessado pelo *Nokia6270 Simulator*

Por meio do acesso realizado pelo *PocketPC*, a interface do *template* simples foi particionada em 6 regiões. Nesse caso as imagens são exibidas, pois o tamanho da tela é maior, como apresenta a Figura 8.13.

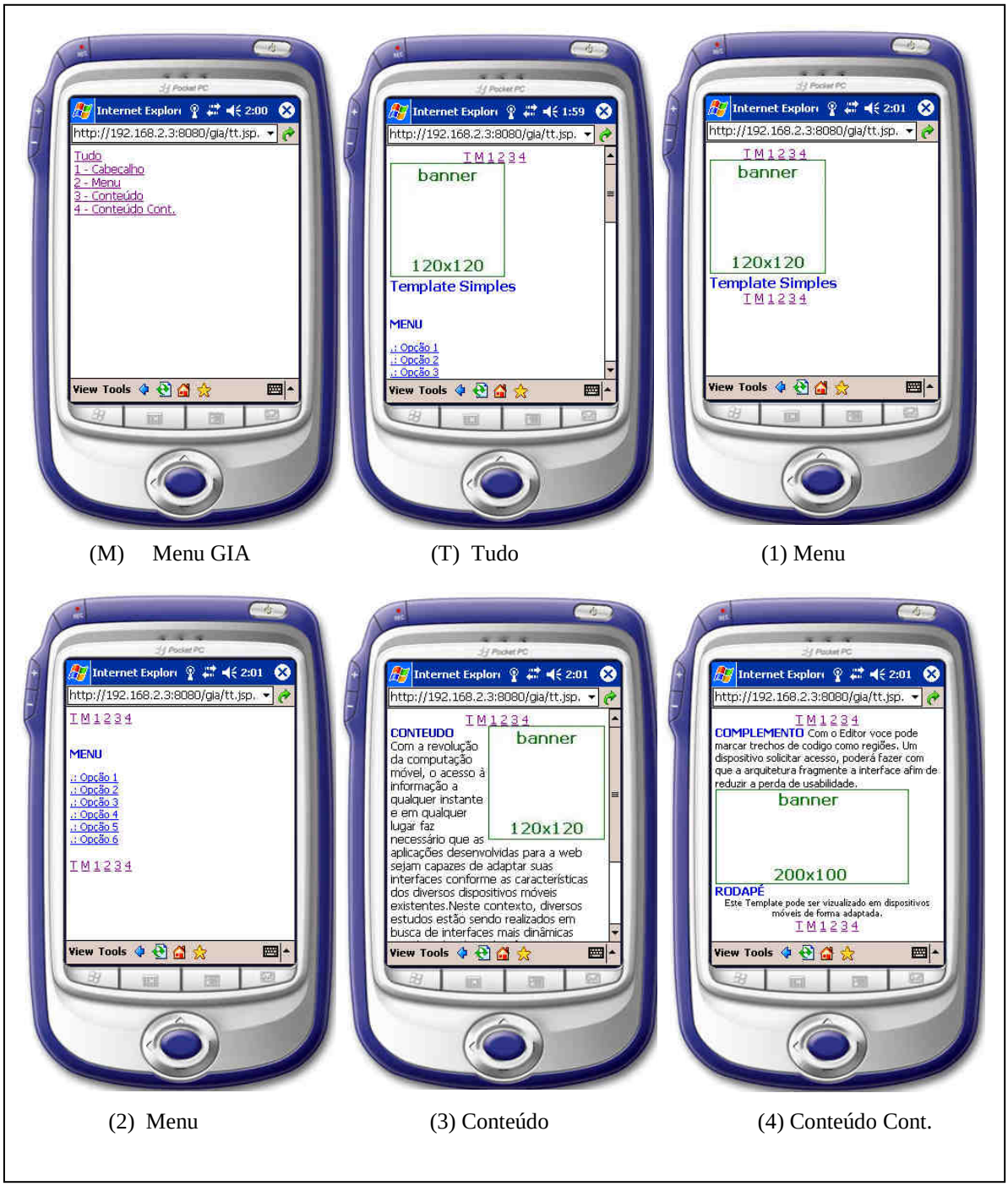


FIGURA 8.13 - *Template* Simples acessado pelo Simulador *PocketPC*

8.3 Considerações sobre os Cenários Apresentados

A partir dos testes realizados nos simuladores citados na tabela 8.1, observa-se que os modelos de interface simples e comercial adaptaram-se ao tamanho da tela de cada dispositivo, gerando para as telas menores até 8 regiões, enquanto para as maiores apenas 5.

Com a geração do menu descrito na seção 8.1.1, um sistema de navegação hierárquica é estabelecido. Apesar de exigir um número maior de cliques para navegar, esse fator é compensado pela maior clareza do conteúdo apresentado, uma vez que as telas geradas são menores que a original.

Além disso, procurou-se particionar regiões que pudessem reduzir o uso da barra de rolagem horizontal, o que representa um grande problema de navegabilidade em dispositivos móveis. Um fator a ser observado é que, ao definir um particionamento exagerado, pode-se afetar a usabilidade da interface.

Testes realizados com um *Palm Zire* 72 apresentaram resultados positivos com relação à adaptação. Para esse dispositivo, o perfil CC/PP, detalhado no capítulo 3, não foi encontrado. A solução para esse problema foi o desenvolvimento de um perfil baseado em informações disponibilizadas pelo fabricante.

Nesse caso pode-se optar por criar um *rdf-schema* proposto em Brickley (2000) no qual são especificados os componentes e os atributos, ou criá-lo baseado em um esquema já existente, restando ao desenvolvedor apenas informar o valor dos atributos referente ao dispositivo.

Para elaborar o perfil do *Palm Zire* 72 utilizou-se o *rdf-schema* localizado em http://www.inrialpes.fr/opera/people/Tayeb.Lemlouma e Layaida /NegotiationSchema/ClientProfileSchema-03012002#. O código completo do perfil pode ser visualizado no apêndice E.

No próximo capítulo apresentam-se as conclusões e resultados obtidos com o trabalho de pesquisa, bem como algumas sugestões para a realização de trabalhos futuros.

CAPÍTULO 9

CONCLUSÃO

As ferramentas atuais de desenvolvimento abordam um tipo de implementação voltada ou para *desktops* ou para dispositivos móveis. O ambiente da computaçao móvel cria a necessidade de uma maior flexibilidade de *software*, impondo novos desafios para a modelagem e desenvolvimento de programas.

Este trabalho buscou, a partir dos desafios identificados inicialmente e considerando-se o escopo e as restrições estabelecidas, elaborar uma Arquitetura para Geração de Interfaces Adaptativas (GIA), com o intuito de contribuir para uma maior facilidade de desenvolvimento para múltiplos dispositivos.

Os trabalhos propostos por Menkhaus (2002) e Dygimes por Conix et al. (2003), utilizam a linguagem XML para geração das interfaces em tempo de execução e Web *services* como forma de comunicação. A transformação da XML por meio da linguagem XSL, exige que cada aparelho tenha um arquivo XSL correspondente, ocasionando diversos arquivos para uma interface, ou seja, um para cada modelo de dispositivo.

A arquitetura GIA utiliza a linguagem JSP, que permite ao desenvolvedor Web produzir aplicações que possibilitam acesso a banco de dados e a arquivos-texto, à captação de informações a partir de formulários, à captação de informações sobre o cliente e o servidor, ao uso de variáveis, entre outras. Pela utilização da linguagem JSP pode-se separar a programação lógica da programação visual. Outra característica do JSP é produzir conteúdos dinâmicos que possam ser reutilizados.

Na arquitetura GIA, o processo de adaptação é realizado integralmente no servidor, sem a necessidade de instalação de nenhum módulo específico no dispositivo móvel, facilitando, dessa forma, o acesso de qualquer aparelho sem uma preocupação prévia com instalação de *software* adicional no cliente.

Um aspecto importante com relação às arquiteturas estudadas refere-se ao fato de que a arquitetura GIA gera aplicações cientes de contexto com capacidade de adquirir informações do usuário de modo automatizado, disponibilizando-as em tempo de execução por meio do CC/PP e Uaprof, detalhados no capítulo 4. Vale ressaltar que o processo de implementação da interface é realizado sem duplicidade de código. Outro fator refere-se à dinamicidade da arquitetura, uma vez que, ao surgirem novos tipos de dispositivos, o sistema não precisa ser reestruturado para suportá-los.

Com a utilização da arquitetura GIA, pretende-se reduzir o tempo de desenvolvimento e duplicidade de conteúdo por meio de uma metodologia menos dependente de propriedades de um dispositivo único. Dessa forma, almeja-se uma programação voltada a um ambiente multiplataforma e que seja capaz de adquirir informações de contexto do usuário de modo automatizado, disponibilizando-as em um ambiente computacional em tempo de execução.

A metodologia de desenvolvimento proposta é um diferencial deste trabalho, pois permite ao programador fragmentar o código, de acordo com diversos tamanhos de tela, bem como ao fato de que o sistema seleciona, em tempo de execução, o tamanho mais adequado ao dispositivo visitante da página. Vale salientar que um único código é gerado e exibido em diversos aparelhos, não sendo necessário desenvolvimento específico para cada tipo de dispositivo.

Pode-se dizer que a arquitetura GIA é adaptativa porque é possível, em tempo de execução, alternar entre os metadados de diversos dispositivos, ocasionando uma instanciação de um novo modelo de objetos com base em um modelo genérico cada vez que uma troca de contexto desse tipo for requisitada pelo usuário, ou seja, cada vez que a aplicação for acessada por um dispositivo.

Pode-se dizer ainda que a arquitetura proposta é considerada adaptável, porque é capaz de acomodar possíveis mudanças no domínio do problema por meio da configuração apropriada dos metadados, permitindo que se possa acompanhar a evolução dos requisitos do domínio, e adaptando-se às necessidades dos usuários.

Assim sendo, os especialistas do domínio e os *designers* podem adaptar o sistema para acomodar novas classes de aparelhos por meio da criação, em tempo de execução, dessas classes e seus atributos.

Dessa forma, considerando as vantagens provenientes de *softwares* adaptativos, a arquitetura apresentada visou obter, como especificado no capítulo 5 deste trabalho, os seguintes objetivos:

- reconhecer os dispositivos por meio do CC/PP, uma especificação reconhecida pela W3C, que em tempo de execução fornece o perfil do aparelho que realiza uma requisição a aplicação;
- gerar interfaces de usuário que se adaptam às características de dispositivos móveis;
- retornar dinamicamente elementos da interface de acordo com as características de cada dispositivo;
- adaptar-se em tempo de execução ao tipo de dispositivo móvel que solicitar o serviço, considerando a classe de dispositivos, o tipo de plataforma, aspectos de contexto do usuário entre outros fatores;
- permitir uma implementação que esteja voltada a um ambiente multiplataforma;
- realizar o processo de adaptação no servidor, sem a necessidade de instalação de nenhum módulo específico no dispositivo móvel;
- propor uma metodologia de desenvolvimento de interfaces para dispositivos móveis;
- propor um ambiente de desenvolvimento com a finalidade de integrar o reconhecimento de dispositivos móveis em tempo de execução a uma metodologia de fragmentação de código.

Com a utilização da ferramenta GIA aliada à metodologia de desenvolvimento, pretende-se contribuir para a diminuição da complexidade do projeto das interfaces Web e melhorar a usabilidade em múltiplos dispositivos. Vale salientar que um único código é gerado e exibido em diversos aparelhos, sendo esse o principal diferencial da arquitetura GIA comparada às ferramentas de desenvolvimento disponíveis.

9.1 Resultados Obtidos

Por meio do desenvolvimento deste trabalho, e especialmente após a realização do estudo de caso pôde-se constatar que a Arquitetura para Geração de Interfaces Adaptativas mostrou-se viável e adequada aos propósitos estabelecidos inicialmente.

Para a validação da arquitetura GIA, bem como da metodologia proposta neste trabalho, realizou-se uma análise por meio de simuladores de dispositivos móveis como telefone celular e PDAs. As interfaces utilizadas foram apresentadas no capítulo 6, intituladas *template* simples e *template* comercial. Ambas foram implementadas utilizando o ambiente de desenvolvimento proposto no capítulo 7.

A partir dos testes realizados nos simuladores, observou-se que os modelos adaptaram-se ao tamanho da tela de cada dispositivo, gerando para as telas menores até 8 regiões enquanto para as maiores, apenas 5.

Com a geração do menu descrito no capítulo 8, um sistema de navegação hierárquica foi estabelecido. Apesar de exigir um número maior de cliques para navegar, esse fator é compensado pela maior clareza do conteúdo apresentado, uma vez que as telas geradas são menores que a original.

Além disso, procurou-se particionar regiões que pudessem reduzir o uso da barra de rolagem horizontal, o que representa um grande problema de navegabilidade em dispositivos móveis. Um fator a ser observado é que, ao definir uma fragmentação exagerada, pode-se afetar a usabilidade da interface.

9.2 Trabalhos Futuros

A conclusão deste trabalho não encerra as considerações a serem feitas em relação ao desenvolvimento de interfaces adaptativas para dispositivos móveis. Na realidade, este trabalho abre uma série de questões que ainda podem ser exploradas, como exemplo:

- Pode-se, futuramente, acrescentar um recurso utilizando inteligência artificial que particione o código em regiões de qualquer interface que for acessada pelo usuário;

- Para facilitar ainda mais o processo de desenvolvimento pode-se adicionar à ferramenta um ambiente gráfico para o desenvolvimento de interfaces, no qual o *designer* poderá inserir componentes sem precisar fazê-lo utilizando código, mas por meio do recurso arrastar-soltar;

- Outro recurso que pode ser adicionado à ferramenta GIA é um módulo para tratar especificamente a adaptação de imagens.

9.3 Considerações Finais

Ao longo deste documento, foram abordadas questões referentes ao contexto de uso dos dispositivos móveis, sendo consenso entre os autores revisados a extrema relevância das informações contextuais para adaptação de interfaces adaptativas.

As idéias deste trabalho já foram publicadas, até o momento, em Ito et al. (2006a), Ito et al. (2006b), Ito et al.(2006c), Ito et al. (2006d), Ito et al. (2006e), Ito et al. (2005a), Ito et al. (2005b), Ito et al. (2004) e Ito et al. (2003).

Espera-se, com o desenvolvimento deste trabalho, colaborar para o estudo de interfaces para dispositivos móveis, oferecendo uma nova alternativa para o desenvolvimento para múltiplos aparelhos e, principalmente, abrindo novos campos de estudo em direção a novas formas de implementação que proporcionem usabilidade e consistência das informações.

REFERÊNCIAS BIBLIOGRÁFICAS

AL-BAR, A.; WAKEMAN, I. A survey of adaptive applications in mobile computing. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS WORKSHOPS, 21, 2001. Phoenix. **Procedings…** Phoenix: IEEE, 2001.

ALUR, D.; CRUPI, J.; MALKS, D. **Core J2EE Patterns**. Rio de Janeiro: Elsevier, 2004

BORNING, A.; LIN, R. K.; MARRIOTT, K. Constraint-Based document layout for the Web. In: ACM INTERNATIONAL MULTIMEDIA CONFERENCE, 5, 1997, Seattle. **Proceedings...** 1997, Seattle. Disponível em: <http://www.csse.monash.edu.au/~marriott/BorLinMar00.pdf>. Acesso em: 05 de jun. 2004

BRAY, T.; PAOLI, J.; SPERBERG-MCQUEEN, C.; MALER, E.; YERGEAU, F. **Extensible markup language (XML) 1.0** 3.ed. W3C Recommendation. Disponível em <http://www.w3.org/TR/REC-xml/>. Acesso em: 20 jul. 2005.

BRICKLEY, D.; GUHA, R.V. E**. Resource description framework (rdf) schema specification 1.0. w3c candidate recommendation 27 march 2000**. Disponível em: <http://www.w3.org/TR/2000/CR-rdf-schema-20000327>. Acesso em: 01 jun. 2006

BRITTON, K.H.; CASE, R.; CITRON, A.; FLOYED, R.; LI,Y.; SEEKAMP C.; TOPOL, B.; TRACEY, K. Transcoding extending e-business to new environments. **IBM Systems Journal**, v.40, n.1, p.153-178, 2001. Disponível em <http://researchweb.watson.ibm.com/journal/sj/401/britton.html>. Acesso em: 01 maio 2005.

BRUSILOVSKY, P.; KARAGIANNIDIS, C.; SAMPSON, D. Adaptive user interfaces models and evaluation. In: CONFERENCE ON HUMAN-COMPUTER INTERACTION, 2001, Patras. **Proceedings…** Greece: ACM, 2001.

BUTLER. M. H. **DELI**: a delivery context library for CC/PP and UAProf. Disponível em: <http://www.hpl.hp.com/personal/marbut/DeliUserGuideWEB.htm>. Acesso em: 19 ago. 2005.

CHEN, Y.; MA,W. Y.; ZHANG, H., J. Detecting web page structure for adaptive viewing on small form factor devices. In: INTERNATIONAL WORLD WIDE WEB CONFERENCE, 12, 2003, Budapest, Hungary. **Proceedings…** Budapest, Hungary: ACM, 2003.

CONINX, K.; LUYTEN, K.; VANDERVELPEN, C.; VAN DEN BERGH, J.; and CREEMERS, B. Dygimes: dynamically generating interfaces for mobile computing

devices and embedded systems. In: HUMAN-COMPUTER INTERACTION WITH MOBILE DEVICES AND SERVICES, INTERNATIONAL SYMPOSIUM, MOBILE HCI 2003, 5, 2003, Udine, Italy. **Proceedings…** Udine, Italy: Springer, 2003. p. 256-270

COUTAZ, J.; NIGAY, L.; SALBER, D. Agent-based architecture modelling for interactive systems. In: PALANQUE, P. ; BENYON, D. **Critical issues in user interface engineering**. London: Spring Verlag, 1995, p. 191-209. ISBN: 3-540-19964-0.

DEY, A. K.; ABOWD, G. D. **A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications.** Disponível em: <http://www.cc.gatech.edu/fce/ctk/pubs/HCIJ16.pdf>. Acesso em: 23 jul. 2005.

DEY, K.; ABOWD ,G. D. Towards a better understanding of context and context-awareness. In: INTERNATIONAL SYMPOSIUM ON HANDHELD AND UBIQUITOUS COMPUTING, 1, 1999, Karlsruhe, Germany. **Proceedings…** London: Spring Verlag, 1999.

DUNHAM, M. H; HELAL, A. Mobile computing and databases: anything new?. **SIGMOD Record**, v. 24, n. 4, 1995, p. 5-9.

FOLEY, J. D.; DAM, A. V.; FEINER, S. K.; HUGHES, J. F. **Computer graphics** - principles and practices. Massachusetts: Addison-Wesley Publishing Co.,1990.

GU, X. D.; CHEN, J. L.; M. A, W. Y.; CHEN, G. L. Visual based content understanding towards web adaptation. In: INTERNATIONAL CONFERENCE ON ADAPTIVE HYPERMEDIA AND ADAPTIVE WEB BASED SYSTEMS, 2, 2002, Malaga, Spain. **Proceedings…** Malaga, Spain: Springer, 2002, p. 164–173.

HAN, R.; PERRET, V.; NAGHSHINEH, M. WebSplitter: a unified xml framework for multi-device collaborative web browsing. In: ACM CONFERENCE ON COMPUTER SUPPORTED COOPERATIVE WORK, PHILADELPHIA, 2000, Philadelphia **Proceedings…** Philadelphia, USA: ACM, 2000, p. 221–230.

HANUMANSETTY, R. **Model based approach for context aware and adaptive user interface generation, 2004.** Disponível em: <http://citeseer.ist.psu.edu/717362.html>. Acesso em: 10 nov. 2005.

HARRINGTON, R. A. **Utilizing bayesian techniques for user interface intelligence.** 1996. Tese de Mestrado. Faculty of the School of Engineering, Air Force Institute of Technology, Air University, Ohio, EEUU.

HASSANEIN, K.; HEAD, M. Ubiquitous Usability: exploring mobile interfaces within the context of a theoretical model. In: CONFERENCE ON ADVANCED INFORMATION SYTEMS ENGINEERING, 15, 2003, Klagenfurt/Velden, Áustria. **Proceedings...** Klagenfurt/Velden, Áustria: University of Klagenfurt , 2003.

HENRICKSEN, K.; INDULSKA, J. Adapting the web interface: an adaptive web browser. In: AUSTRALASIAN USER INTERFACE CONFERENCE (AUIC.01), 4, 2001, Gold Coast, Queensland, Austrália. **Proceedings...** Gold Coast, Queensland Austrália: IEEE, 2001. (0-7695-0969-X/01)

HINZ, M.; FIÁLA, Z. **Context modeling for device- and location-aware mobile web applications**. Disponível em: http://www.medien.ifi.lmu.de/permid2005 /pdf/MichaelHinz_Permid2005.pdf. Acesso em: 10 ago. 2005.

ITO, G. C.; FERREIRA, M.; SANT'ANNA, N. Computação movél: aspectos de gerenciamento de dados. In: WORKSHOP DOS CURSOS DE COMPUTAÇÃO APLICADA DO INPE, 3. (WORCAP), 26-27 nov. 2003, São José dos Campos. **Anais...** São José dos Campos: Instituto Nacional de Pesquisas Espaciais, 2003. p. 253 - 258. CD-ROM, On-line. Disponível em: <http://hermes2.dpi.inpe.br:1905/rep-/lac.inpe.br/worcap/2003/11.04.09.51>. Acesso em: 13 set. 2007. rep: lac.inpe.br/worcap/2003/11.04.09.51.

ITO, G. C.; FERREIRA, M.; SANT' ANNA, N. Utilização de Interfaces Adaptativas para a Computação Móvel. In: WORKSHOP DOS CURSOS DE COMPUTAÇÃO APLICADA DO INPE, 4. (WORCAP), 20 e 21 out. 2004, São José dos Campos. **Anais...** São José dos Campos: Instituto Nacional de Pesquisas Espaciais, 2004. Resumos Extendidos. CD-ROM, On-line. Disponível em: <http://hermes2.dpi.inpe.br:1905/rep-/lac.inpe.br/worcap/2004/09.30.13.53>. Acesso em: 13 set. 2007. rep: lac.inpe.br/worcap/2004/09.30.13.53.

ITO, G. C.; FERREIRA, M. G.; SANT'ANNA, N. Uma arquitetura para geração de interfaces adaptativas para dispositivos móveis. In: INTERNATIONAL INFORMATION AND TELECOMMUNICATION TECHNOLOGIES SYMPOSIUM, 4, 2005, Florianópolis**. Proceedings...** Florianópolis: [s,n], 2005a.

ITO, G. C.; FERREIRA, M.; SANT'ANA, N.; SILVA, R. R. Geração de Interfaces Adaptativas para Dispositivos Móveis. In: WORKSHOP DOS CURSOS DE COMPUTAÇÃO APLICADA DO INPE, 5. (WORCAP), 26 e 27 out. 2005, São José dos Campos. **Anais...** São José dos Campos: Instituto Nacional de Pesquisas Espaciais, 2005. CD-ROM, On-line. Disponível em: <http://hermes2.dpi.inpe.br:1905/rep-dpi.inpe.br/hermes2@1905/2005/10.03.16.14>. Acesso em: 13 set. 2007. rep: dpi.inpe.br/hermes2@1905/2005/10.03.16.14.

ITO, G. C.; FERREIRA, M., SANT'ANNA, N. Uma ferramenta para geração de interfaces adaptativas. In: CONFERÊNCIA IBERO AMERICANA WWW/INTERNET, 2006. Murcia – Espanha. **Anais...** Murcia – Espanha: IADIS Press, 2006a.

ITO, G. C.; FERREIRA, M. G.; SANT'ANNA, N.; SANTOS, A. A.; FINKLER, D. T.; SANTOS, M.M.. **Adaptação de interfaces web para dispositivos móveis**. Hífen, v.30, n. 58, Uruguaiana, PUCRS, 2006b, p. 41-47.

ITO, G. C.; FERREIRA, M.G.; SANT' ANNA, N. A Proposed Architecture for the Generation of Adaptive Interfaces in Mobile Devices. In: WORKSHOP DOS CURSOS DE COMPUTAÇÃO APLICADA DO INPE, 6. (WORCAP), 8 e 9 nov. 2006, São José dos Campos. **Proceedings...** São José dos Campos: Instituto Nacional de Pesquisas Espaciais (INPE), 2006. inscrito no doutorado - proposta defendida. On-line. Disponível em: <http://hermes2.dpi.inpe.br:1905/rep-/dpi.inpe.br/hermes2@1905/2006/10.17.14.15>. Acesso em: 13 set. 2007. rep: dpi.inpe.br/hermes2@1905/2006/10.17.14.15.

ITO, G. C.; FERREIRA, M.; SANT'ANA, N.; SILVA, R. R. Geração de Interfaces Adaptativas para Dispositivos Móveis. In: WORKSHOP DOS CURSOS DE COMPUTAÇÃO APLICADA DO INPE, 5. (WORCAP), 26 e 27 out. 2005, São José dos Campos. **Anais...** São José dos Campos: Instituto Nacional de Pesquisas Espaciais, 2005. CD-ROM, On-line. Disponível em: <http://hermes2.dpi.inpe.br:1905/rep-dpi.inpe.br/hermes2@1905/2005/10.03.16.14>. Acesso em: 13 set. 2007. rep: dpi.inpe.br/hermes2@1905/2005/10.03.16.14.

ITO, G. C.; FERREIRA, M. G.; SANT'ANNA, N.; FINKLER, D. T.; SANTOS, A. A. dos; SANTOS, M.M. dos. Interfaces adaptativas para dispositivos móveis: um estudo de caso. In: SEMINÁRIO DE INFORMÁTICA, 5, 2006, Torres-RS. **Anais…**Torres–RS: Ulbra, 2006e.

KAIKKONEN, A.; VIRPI, R. Navigating in a mobile XHTML application. In: Conference on Human factors in computing systems. In: CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 2003, Lauderdale, Florida, USA. **Proceedings...** Lauderdale, Florida, USA: ACM, 2003. (ISBN 1-58113-630-7)

KLYNE, G.; REYNOLDS, F.; WOODROW, C.; OHTO, H.; HJELM, J.; BUTLER, M.; TRAN, L. **Composite capability/preference profiles (cc/pp):** structure and vocabularies 1.0. Disponível em: <http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/>. Acesso em: 15 ago. 2005.

KRISTOFFERSEN, S.; LJUNGBERG, F. Designing interaction styles for a mobile use context. In: INTERNATIONAL SYMPOSIUM ON HANDHELD AND UBIQUITOUS COMPUTING, 1, 1999, Karlsruhe, Germany. **Proceedings…** London: Spring Verlag, 1999. p. 208-221.

LASSILA, O. ; SWICK, R. **Resource description framework (rdf) model and syntax specification.** Disponível em: http://www.w3.org/1999/.status/REC-rdf-syntax-19990222/status. Acesso em: 04 out. 2005.

LEMLOUMA, T.; LAYAIDA, N. Context-aware adaptation for mobile devices. In: INTERNATIONAL CONFERENCE ON MOBILE DATA MANAGEMENT, 2004, Saint Martin, France. **Proceedings..**, Saint Martin, France:IEEE, 2004. p.106-111

LIEBERHERR, K.; LOPES C. Adaptable and adaptive software workshop report. In: Conference on Object-Oriented Programming Systems, Languages, and Applications, 1995, Austin, Texas, USA. **Procedings…** Austin, Texas, USA:ACM Digital Library, 1995.

LIEBERMAN, H. Integrating User interface agents with conventional applications. In; IUI'99 Conference, CA, USA, 1999, Redondo Beach, CA, USA. **Proceedings…**New York: ACM Press, 1999.

MALLICK, M. **Mobile and wireless design essentials**. Indiana: Wiley Publishing, 2003

MCKINLEY, P. K.; SADJADI S. M.; KASTEN E. P.; CHENG, B. H. C. Composing adaptive software. **IEEE Computer**, v.37, n.7, 2004. p.56-64

MCTEAR, M. **Intelligent interface technology: from theory to reality? interacting with computers,** 2000, v.12, n.4, 2000, p. 323-336.

MENKHAUS, G. **Adaptive user interface generation in a mobile computing environment**. PhD Thesis, University of Salzburg, Salzburg, Austria, 2002

MENNECKE, B. E.; STRADER, T. J. **Mobile commerce:** technology, theory and applications. London: Idea Group Publishing, 2003

MILIC-FRAYLING, N.; SOMMERER, R. SmartView: flexible viewing of web page contents. In: WWW'02. 2002, Honolulu, USA, **Proceedings…**Honolulu,USA: [s,n], 2002

MORI, G., PATERNÒ, F., SANTORO, C. Design and development of multidevice user interfaces through multiple logical descriptions. **IEEE Transactions on Software Engineering,** v. 30, n. 8, ago. 2004. p.507-520.

MÜLLER, M. E. **Inducing conceptual user models**. Phd Thesis, University of Osnabrück, Germany, 2002

MURPHY, G.; WALKER, R., BANIASSAD, E. Evaluating emerging software development technologies: lessons learned from assessing aspect-oriented programming. **IEEE Transactions on Software Engineering**, v. 25, n.4, 1999. p. 438- 455.

NIELSEN, J. **Usabilidade no celular e em buscas**. 2006. Disponível em: <http://www.jumpexec.com.br/index.php?sub=3&land=ler&idArtigo=360>. Acesso em: 19 jan. 2007.

PASCOE, J.; RYAN, N. S.; MORSE, D. R.Using while moving: hci issues in fieldwork environments.**ACM Transactions on Computer Human Interaction**, v.7, p.417-437, 2000.

PRABHU, A. **Adaptive content rendering strategies**. DM Direct Newsletter, 2003. Disponível em <http://www.dmreview.com/authors/author_sub.cfm?AuthorID=32057> Acesso em: 05 de out. 2005

RIBEIRO, N.M. A utilidade das tecnologias de computação móvel no ensino: o caso da gestão de informação pessoal do professor. In: Gouveia, L. B., Gaio, S., (Orgs), **Sociedade da informação: balanço e implicações**. Porto, Portugal: Edições Universidade Fernando Pessoa, cap. 14, 2004.

ROTO. V. **Web browsing on mobile phones -** characteristics of user experience. Tese de Doutorado. Helsinki University of Technology. Espoo, Finlândia. 2006

SENDIN, M.; LORES, J. Plasticity in mobile devices: a dichotomic and semantic view**.** In: INTERNATIONAL CONFERENCE ON ADAPTIVE HYPERMEDIA AND ADAPTIVE WEB-BASED SYSTEMS, 3, 2004, Eindhoven, Holanda. **Proceedings…** Eindhoven, Holanda: [s,n], 2004a. p. 58-67.

SENDIN, M.; LORÉS, J. Plastic user interfaces: designing for change. In: International Workshop on Making Model-based User Interface Design Practical, 1, 2004, Lleida, SPAIN. **Procedings…** Lleida, SPAIN:[s,n], 2004b. Disponível em: <http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-103/sendin-et-al.pdf>. Acesso em: 01 de março 2006

SSU, K.; YAO, B.; FUCHS, K.; NEVES, N. F. **Adaptive checkpointing with storage management for mobile environments**, manuscript, Dez. 1998. Disponível em http://citeseer.nj.nec.com/context/582387/0. Acesso em: 07 de abril 2005.

STAYANARAYANAN, M. Mobile information access. **IEEE personal communications**, v. 3, n. 1, Pittsburgh, 1996, p.1-8

STEHLING, R. O. **Projeto e Implementação de uma arquitetura de software reflexiva para linguagem xchart.** Dissertação (Mestrado em Computação)- Instituto de Computação da Universidade Estadual de Campinas, Campinas. UNICAMP , 1999

TEMPLE, A.; MELLO, R. F.; CALEGARI, D. T.; SCHIEZARO. M. **Programação web com JSP, servlets e J2EE**. Creative Commons, Califórnia. 2004.

TEWARI, R.; GRILLO, P. Data management for mobile computers in internet. In: ACM COMPUTER SCIENCE CONFERENCE, 23, 1995, Tennessee, United States. **Proceedings.**.. Tennessee, United States: ACM Press, 1995, p. 246-252.

VIANA,W.; TEIXEIRA R.; CAVALCANTE, P.; ANDRADE, R. **Mobile adapter:** uma abordagem para a construção de mobile application servers adaptativos utilizando as especificações CC/PP e UAProf, 2005. Disponível em: <www.sbc.org.br/bibliotecadigital/download.php?paper=173>. Acesso em: 08 fev. 2006.

WEISER, M. The Computer for the twenty-first century. **Scientific American,** v. 265, n. 3, 1991, p. 94-104

XIE, X.; WANG, C.; CHEN, L.; MA W. An adaptive web page layout structure for small devices**. ACM Multimedia Systems Journal**, v.11, n. 1, p. 34-44, 2005.

APÊNDICE A

CÓDIGOS DA CLASSE DOS SERVIÇOS DA ARQUITETURA GIA

A.1 Código da Classe do Serviço de Comunicação

A.1.1 Classe Tdeli para Capturar a Conexão do Dispositivo

```
package gia.deli;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.hp.hpl.deli.Profile;
import com.hp.hpl.deli.Workspace;
public class Tdeli extends HttpServlet {

	static String content=null;
	private static final long serialVersionUID = 1L;
	Workspace workspace;
	protected ServletContext servletContext;
	public Tdeli() {
		super();
	}
	public void init(ServletConfig config) throws ServletException {
		super.init(config);
		try {
		   servletContext = config.getServletContext();
		   Workspace.getInstance().configure(servletContext, "config/deliConfig.xml");
		} catch (Exception e) {
		   try {
		      Workspace.getInstance().configure((ServletContext) null, "config/deliConfig.xml");
		   } catch (Exception f) {
		      System.out.println(f.toString());
		   }
		}
	}
	public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
		Profile perfil = new Profile(req);
		String ScreenSize;
		ServletContext app = getServletContext();
	   req.setAttribute("ScreenSize",perfil.getAttribute("ScreenSize").get().toString());
	   req.setAttribute("OSName",perfil.getAttribute("OSName").get().toString());
	   req.setAttribute("OSVendor",perfil.getAttribute("OSVendor").get().toString());
	   req.setAttribute("BrowserName",perfil.getAttribute("BrowserName").get().toString());
	   req.setAttribute("BrowserVersion",perfil.getAttribute("BrowserVersion").get().toString());
	   req.setAttribute("Vendor",perfil.getAttribute("Vendor").get().toString());
	   req.setAttribute("Model",perfil.getAttribute("Model").get().toString());
	   req.setAttribute("Url", req.getRequestURI().replace(".gia","").substring(req.getRequestURI().indexOf("/",1)));
	   req.setAttribute("QueryString",req.getQueryString());
	   RequestDispatcher disp =
app.getRequestDispatcher(req.getRequestURI().replace(".gia","").substring(req.getRequestURI().indexOf("/",1)));
```

```
            disp.forward(req,res);
        }
}
```

A.2 Código da Classe do Serviço de Adaptação

A classe TComponente transformar as tags genéricas em componentes reconhecidos pelos dispositivos.

```
package gia.tag;
import java.io.IOException;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.servlet.jsp.JspTagException;
import javax.servlet.jsp.tagext.BodyContent;
import javax.servlet.jsp.tagext.BodyTagSupport;
public class TComponente extends BodyTagSupport  {
private String Tipo;
private String Propriedades;
private String TagCorpo;
private String QLinha;
private boolean BPropiedade = false;
private String[][] VPropriedades = new String[100][2];
private int TotalP = 0;
private String TempComponente;
private TBrowser Browser;
private DBase con;
private TQuery qry;
public TComponente() {
        }
                public int doAfterBody() throws JspTagException {
                BodyContent bc = getBodyContent();
                String t = bc.getString();
                try {
                        getPreviousOut().print( t);
                } catch (IOException e) {
                        throw new JspTagException("TransformTag: " +e.getMessage());
                }
                release();
                return SKIP_BODY;
        }
        public int doStartTag() {
                StringBuffer sb = new StringBuffer();
                con = new DBase(pageContext.getServletContext().getRealPath("/WEB-INF"));
                qry = new TQuery();
                qry.SetConexao(con.GetConexao());
                String strSql;
                boolean BTipo= false;
                int Perfil;
                THtml Html = (THtml)findAncestorWithClass(this,THtml.class);
                if (Html == null){
                        TComponente Comp = (TComponente)findAncestorWithClass(this,TComponente.class);
                        Browser = Comp.getBrowser();
                }else{
                        Browser = Html.getBrowser();
                }
                strSql = " select " +
                                "  c.Nome as ComponenteNome," +
                                "  c.Corpo, " +
                                "  c.QuebraLinha, "+
                                "  tp.Nome as TagPropriedade," +
```

```
                                        "  cp.Nome as ComponentePropriedade" +
                                        " from componentexcpropriedade ccp" +
                                        "  inner join componente c on (c.codComponente = ccp.CodComponente)" +
                                        "  inner join CPropriedade cp on (cp.CodCpropriedade = ccp.CodCPropriedade)" +
                                        "  inner join tagxTPropriedade ttp on(ttp.CodTagTPropriedade =
ccp.CodTagTPropriedade )" +
                                        "  inner join tag t on(t.codtag = ttp.codTag)" +
                                        "  inner join tpropriedade tp on (tp.codtpropriedade = ttp.codtpropriedade)" +
                                        " where c.CodPerfil = "+ Browser.Perfil.getCodigo() +" and upper(t.Nome) = upper('"+
Tipo +"')";

                        if(BPropiedade){
                                strSql = strSql + " and upper(tp.Nome) in("+ CStrVetor(VPropriedades) +")";
                        }

                        qry.Sql(strSql);

                        ResultSet rs = qry.Resultado();
                        try {
                                sb.append("<");
                                while(rs.next()){
                                        if(!BTipo){
                                          BTipo = true;
                                          TempComponente = rs.getString("ComponenteNome");
                                          TagCorpo = rs.getString("Corpo");
                                          QLinha = rs.getString("QuebraLinha");
                                          sb.append(TempComponente);
                                        }

        sb.append(BuscaPropriedade(rs.getString("TagPropriedade"),rs.getString("ComponentePropriedade")));
                                }
                                sb.append(">");
                        } catch (SQLException e) {
                                e.printStackTrace();
                        }

                        try{
                                pageContext.getOut().write(sb.toString());
                        }catch(IOException ioe){
                                System.out.println("Erro Tag: " + ioe.getMessage());
                        }
                        release();
                        return EVAL_PAGE;
                }

                public void release() {
                        super.release();
                }
                public int doEndTag() {
                        StringBuffer sb = new StringBuffer();
                        if(TagCorpo.equalsIgnoreCase("S")){
                                sb.append("</"+ TempComponente +">");
                        }
                        if(QLinha.equalsIgnoreCase("S")){
                                sb.append("<br>");
                        }
                        if(sb.length() > 0){
                                try {
                                        pageContext.getOut().write(sb.toString());
                                } catch (Exception e) {
                                        e.printStackTrace();
                                }
```

```
		}
		return EVAL_PAGE;
	}
	 public void setTipo(String valor) {
		Tipo = valor;
	}
	public void setPropriedade(String valor) {
		Propriedades = valor;
		if(Propriedades == null){
			BPropiedade = false;
		}else{
			if(valor.trim().length() == 0){
				BPropiedade = false;
			}else{
				BPropiedade = true;
				CVetor(Propriedades);
			}
		}
	}
	 public void CVetor(String valor) {
		boolean sair = false;
		String TProp;
		int cont = 0;
		while(!sair){
			if(valor.indexOf(";") == -1){
				sair = true;
				TProp = valor.trim();
			}else{
				TProp = valor.substring(0,valor.indexOf(";"));
			}
			VPropriedades[cont][0] = TProp.substring(0,TProp.indexOf("=")).trim();
			VPropriedades[cont][1] = TProp.substring(TProp.indexOf("=")+1,TProp.length()).replace("'","");
			valor = valor.substring(valor.indexOf(";")+1,valor.length());
			cont++;
		}
		TotalP = cont-1;
	}
	 private String CStrVetor(String[][] valor) {
		String strReturn = "";
		for(int i=0;i<=TotalP;i++){
			strReturn =strReturn + "upper('" + valor[i][0].trim() + "'),";
		}
		return  strReturn.substring(0,strReturn.length()-1);
	}
	 private String BuscaPropriedade(String TPropriedade, String CPropriedade) {
		String strReturn = "";
		for(int i=0;i<=TotalP;i++){
			if(TPropriedade.equalsIgnoreCase(VPropriedades[i][0])){
				strReturn = " "+ CPropriedade+ "='"+ VPropriedades[i][1] +"'";
				break;
			}
		}
		return strReturn;
	}
	public TBrowser getBrowser() {
		return Browser;
	}

}
```

APÊNDICE B

CÓDIGOS DA INTERFACE DOS TEMPLATES DA FERRAMENTA GIA

B.1 Código do Template Simples

```
<%@taglib uri="gia" prefix="gia"%>
<gia:html>
<head>
<title> Template Simples</title>
</head>
<body>
<table width="100%" height="100%" border="1" cellpadding="0" cellspacing="1"
bordercolor="#CCCCCC">
  <tr>
    <td height="20%" colspan="3">
      <gia:regiao300 id="Topo300" seq="1" comentario="Cabeçalho">
        <img src="imagens/img_120x120.jpg" alt="logotipo do site" width="120" height="120"
align="middle">
        <gia:regiao100 id="Topo" seq="1" comentario="Cabeçalho">
        <br>
        <font color="#0000CC" size="5" face="Geneva, Arial, Helvetica, sans-serif"><strong> Template
Simples</strong></font><br>
      </gia:regiao100>
      </gia:regiao300>
    </td>
  </tr>
  <tr>
    <td width="20%" height="60%" align="center">
    <gia:regiao300 id="Menu300" seq="2" comentario="Menu">
       <gia:regiao100 id="Menu100" seq="2" comentario="Menu">
         <br>
         <strong><H4><font color="#0000CC" face="Geneva, Arial, Helvetica, sans-
serif">MENU</font></H4></strong>
          <font size="2" face="Geneva, Arial, Helvetica, sans-serif">
                                  <a href="template1.jsp">.: Op&ccedil;&atilde;o 1</a><br>
            <a href="template1.jsp">.: Op&ccedil;&atilde;o 2</a><br>
            <a href="template1.jsp">.: Op&ccedil;&atilde;o 3</a><br>
            <a href="template1.jsp">.: Op&ccedil;&atilde;o 4</a><br>
            <a href="template1.jsp">.: Op&ccedil;&atilde;o 5</a><br>
            <a href="template1.jsp">.: Op&ccedil;&atilde;o 6</a><br>
          </font>
        <br>
       </gia:regiao100>
     </gia:regiao300>
           </td>
    <td width="60%" height="60%">
             <gia:regiao300 id="Centro300" seq="3" comentario="Conteúdo">
               <img src="imagens/img_120x120.jpg" width="120" height="120" align="right">
                      <gia:regiao100 id="Centro100" seq="3" comentario="Contéudo">
          <font size="3" face="Geneva, Arial, Helvetica, sans-serif" color="#000099">
                                    <strong>CONTEUDO</strong>
                                  </font>
                                  <br>
          <font size="3" face="Geneva, Arial, Helvetica, sans-serif">Com a
     revolu&ccedil;&atilde;o da computa&ccedil;&atilde;o m&oacute;vel, o acesso
     &agrave; informa&ccedil;&atilde;o a qualquer instante e em qualquer lugar
     faz necess&aacute;rio que as aplica&ccedil;&otilde;es desenvolvidas para
```

```
    a web sejam capazes de adaptar suas interfaces conforme as caracter&iacute;sticas
    dos diversos dispositivos m&oacute;veis existentes.Neste contexto, diversos
    estudos est&atilde;o sendo realizados em busca de interfaces mais din&acirc;micas
    envolvendo v&aacute;rios desafios, como ambientes heterog&ecirc;neos, limita&ccedil;&otilde;es
    f&iacute;sicas do aparelho, entre outras. As interfaces adaptativas se apresentam
    promissoras na tentativa de superar os problemas atuais de complexidade
    na intera&ccedil;&atilde;o homem-computador.
                    </font>
                        </gia:regiao100>
            </gia:regiao300>
   </td>
   <td width="20%" height="60%">
    <p align="justify">
              <gia:regiao300 id="Direito300" seq="4" comentario="Conteúdo Cont.">
                        <gia:regiao100 id="Direito100" seq="4" comentario="Conteúdo Cont">
          <font color="#0000CC" size="3" face="Geneva, Arial, Helvetica, sans-serif">
            <strong>COMPLEMENTO</strong>
                                  </font>
                                  <font size="2" face="Geneva, Arial, Helvetica, sans-serif">
        Com o Editor voce pode marcar trechos de codigo como regiões. Um dispositivo solicitar
acesso,
        poderá fazer com que a arquitetura fragmente a interface afim de reduzir a perda de
usabilidade.</font>
                        </gia:regiao100>
                        <img src="imagens/img_200x100.jpg" alt="logotipo do site" width="200"
height="100">
     <br>
     </p>
           </td>
  </tr>
  <tr>
    <td height="20%" colspan="3">

       <gia:regiao100 id="Rodape100" seq="5" comentario="Rodape">
                          <font color="#0000CC" size="3" face="Geneva, Arial, Helvetica, sans-serif">
            <strong>RODAPÉ</strong>
                                  </font>
          <center>
            <font size="1" face="Geneva, Arial, Helvetica, sans-serif">Este Template pode ser
visualizado em dispositivos móveis de forma adaptada.</font>
          </center>
       </gia:regiao100>
   </gia:regiao300>
           </td>
  </tr>
</table>
</body>
</gia:html>
```

B.2 Código do Template Comercial

```
<%@taglib uri="gia" prefix="gia"%>
<gia:html>
<body>
<table width="100%" height="100%" border="1">
  <tr>
    <td colspan="3">
           <gia:regiao400 id="regiao400" seq="1" comentario="Titulo">
            <gia:regiao300 id="regiao300" seq="1" comentario="Titulo">
             <gia:componente  tipo="IMG"
propriedade="SRC=imagens/logo_do_site.jpg;ALIGN=middle"/>
```

```
            <gia:regiao250 id="regiao250" seq="1" comentario="Titulo">
             <gia:regiao200 id="regiao200" seq="1" comentario="Titulo">
              <gia:regiao150 id="regiao150" seq="1" comentario="Titulo">
               <gia:regiao100 id="regiao100" seq="1" comentario="Titulo">
                  <font color="#006666" size="2" face="Verdana, Arial, Helvetica, sans-serif">
                     <strong> GERADOR DE INTERFACES ADAPTATIVAS </strong>
                  </font>
               </gia:regiao100>
              </gia:regiao150>
             </gia:regiao200>
            </gia:regiao250>
           </gia:regiao300>
          </gia:regiao400>
  </td>
 </tr>
 <tr>
  <td width="20%">
          <gia:regiao400 id="regiao400" seq="2" comentario="Menu">
           <gia:regiao300 id="regiao300" seq="2" comentario="Menu">
            <gia:regiao250 id="regiao250" seq="2" comentario="Menu">
             <gia:regiao200 id="regiao200" seq="2" comentario="Menu">
              <img src="imagens/palm.gif" align="center"> <br>
              <gia:regiao150 id="regiao150" seq="2" comentario="Menu">
               <gia:regiao100 id="regiao100" seq="2" comentario="Menu">
                  <font color="#006666" size="1" face="Verdana, Arial, Helvetica, sans-
serif">INTRODUÇÃO <br><br>
                  <strong>.:</strong><a href="index.jsp">Dispositivos Móveis</a> <br>
                  <strong>.:</strong><a href="index.jsp">Object Pascal</a> <br>
                  <strong>.:</strong><a href="index.jsp">Trabalhos Relacionados</a> <br>
                  <strong>.:</strong><a href="index.jsp">Práticas de Usabilidade</a> <br>
                  <br> DIVISÕES<br> <br>
                  <strong>.:</strong><a href="index.jsp">Arquitetura do Servidor</a> <br>
                  <strong>.:</strong><a href="index.jsp">Editor</a> <br>
                  <strong>.:</strong><a href="index.jsp">Gerenciador</a> <br>
                  <br> MANUAIS E TUTORIAIS <br><br>
                  <strong>.:</strong><a href="index.jsp">Manual do Sistema</a> <br>
                  <strong>.:</strong><a href="index.jsp">Manual do Editor</a> <br>
                  <strong>.:</strong><a href="index.jsp">Manual do Gerenciador</a> <br>
                  <strong>.:</strong><a href="index.jsp">Metodologia</a> <br>
                  <strong>.:</strong><a href="index.jsp">DELI</a> <br>
                  <strong>.:</strong><a href="index.jsp">CC/PP</a> <br>
                  <strong>.:</strong><a href="index.jsp">Java Server Pages</a> <br>
                  </font>
               </gia:regiao100>
              </gia:regiao150>
             </gia:regiao200>
            </gia:regiao250>
           </gia:regiao300>
          </gia:regiao400>
  </td>
  <td width="60%">
          <gia:regiao400 id="regiao400" seq="3" comentario="Conteudo">
           <gia:regiao300 id="regiao300" seq="3" comentario="Conteudo">
            <gia:regiao250 id="regiao250" seq="3" comentario="Conteudo">
             <gia:regiao200 id="regiao200" seq="3" comentario="Conteudo">
              <gia:regiao150 id="regiao150" seq="3" comentario="Conteudo">
               <gia:regiao100 id="regiao100" seq="3" comentario="Conteudo">
                <font color="#006666" size="2" face="Verdana, Arial, Helvetica, sans-serif">
                   O TRABALHO <br>
                </font><br>
                 <font color="#0033CC" size="2" face="Verdana, Arial, Helvetica, sans-serif">
                  Com o crescimento da necessidade de mobilidade, surgem novas
```

```
                maneiras de utilização do computador a qualquer hora e em qualquer lugar.
                A diminuição de tamanho, peso e energia consumida cria uma combinação de
                várias tecnologias no mesmo dispositivo, tornando os meios de ligação entre
                computadores cada vez mais flexíveis, onde recursos não necessitam de uma
                localização fixa e nem precisam estar fisicamente conectados para se comunicar.
                A computação móvel utiliza dispositivos portáteis e comunicação sem fio
                para permitir aos usuários trabalharem fora de ambientes fixos. Estes dispositivos
                aceleram o surgimento de novas aplicações que advêm, sobretudo da sinergia
                criada pela combinação da mobilidade com novas tecnologias de armazenamento
                digital, comunicação e processamento de dados.
                                </font>
              </gia:regiao100>
             </gia:regiao150>
            </gia:regiao200>
            <gia:regiao200 id="regiao200" seq="4" comentario="Conteudo Cont.">
             <gia:regiao150 id="regiao150" seq="4" comentario="Conteudo Cont.">
              <br>
              <gia:regiao100 id="regiao100" seq="4" comentario="Conteudo Cont.">
                <font color="#0033CC" size="2" face="Verdana, Arial, Helvetica, sans-serif">
                Devido a computa&ccedil;&atilde;o m&oacute;vel ainda ser uma tecnologia recente,
                muitos dos sites desenvolvidos para a web n&atilde;o possuem a capacidade de
                representar as informa&ccedil;&otilde;es mostradas em um computador pessoal da
                mesma forma em um dispositivo m&oacute;vel. Desta forma, quando uma
p&aacute;gina
                &eacute; requisitada em dispositivo m&oacute;vel, n&atilde;o &eacute; garantido que a
                mesma possua uma apar&ecirc;ncia agrad&aacute;vel e forne&ccedil;a as mesmas
                funcionalidades da p&aacute;gina desenvolvida para PC&acute;s.
Para resolver este problema foi  desenvolvido uma estrutura chamada GIA, onde o desenvolvedor
armazena os dados da aplica&ccedil;&atilde;o em um banco de                dados, cria o
c&oacute;digo num editor e internamente h&aacute; um controlador                que &eacute;
respons&aacute;vel por efetuar a tratativa para que a p&aacute;gina seja                exibida para o
cliente de uma maneira adapt&aacute;vel e us&aacute;vel.
                </font>
              </gia:regiao100>
                <br>
              <gia:regiao100 id="regiao100" seq="5" comentario="Conteudo Cont.">
                                                <font color="#0033CC" size="2" face="Verdana, Arial,
Helvetica,sans-serif">
                Neste editor, o desenvolvedor ter&aacute; liberdade para utilizar
                todos os recursos da tencologia Java, al&eacute;m de dispor de um
                mecanismo que simula a tela de um dispositivo, onde ele poder&aacute;
                visualizar o andamento do seu trabalho.
                                                </font>
              </gia:regiao100>
             </gia:regiao150>
            </gia:regiao200>
           </gia:regiao250>
          </gia:regiao300>
         </gia:regiao400>
    </td>
     <td width="20%" align="center" valign="top">
     <gia:regiao400 id="regiao400" seq="4" comentario="Busca Google">
     <gia:regiao300 id="regiao300" seq="4" comentario="Busca Google">
      <gia:regiao250 id="regiao250" seq="4" comentario="Busca Google">
       <gia:regiao200 id="regiao200" seq="5" comentario="Busca Google">
        <gia:regiao150 id="regiao150" seq="5" comentario="Busca Google">
         <gia:regiao100 id="regiao100" seq="6" comentario="Busca Google">
           <br>
           <font color="#006600" size="1" face="Verdana, Arial, Helvetica, sans-serif">
            <strong>Buscar na Web por:</strong>
           </font>
           <br>
```

```
          <INPUT TYPE=text name=q2 size=17 maxlength=255 value="" class="form">
          <INPUT TYPE=image name=btnG src="imagens/ok.jpg" border="0">
          <br>
          <font color="#666666" size="1" face="Verdana, Arial, Helvetica, sans-serif">
            Powered by <a href="http://www.google.com.br" target="_blank">Google</a>
          </font>
         </gia:regiao100>
        </gia:regiao150>
       </gia:regiao200>
      </gia:regiao250>
     </gia:regiao300>
      <br>
     <gia:regiao300 id="regiao300" seq="5" comentario="Publicidade">
      <gia:regiao250 id="regiao250" seq="5" comentario="Publicidade">
       <gia:regiao200 id="regiao200" seq="6" comentario="Publicidade">
        <gia:regiao150 id="regiao150" seq="6" comentario="Publicidade">
         <font color="#006600" size="1" face="Verdana, Arial, Helvetica, sans-serif">
          <strong>Publicidade:</strong>
         </font>
         <br>
         <img src="imagens/banner1.gif" width="120" height="60">
        </gia:regiao150>
       </gia:regiao200>
       <gia:regiao200 id="regiao200" seq="7" comentario="Publicidade Cont.">
        <gia:regiao150 id="regiao150" seq="7" comentario="Publicidade Cont.">
         <br>
         <img src="imagens/banner2.gif" width="120" height="60">
         <br>
         <br>
         <div>
         <strong> <font size="1" face="Verdana"> O que você achou do nosso trabalho?
</font></strong>
         <br>
         <input type="radio" name="voteID" value="1">
         <font size="1" face="Verdana">Ótimo</font><br>
         <input type="radio" name="voteID" value="2">
         <font size="1" face="Verdana">Bom</font><br>
         <input type="radio" name="voteID" value="3">
         <font size="1" face="Verdana">Regular</font><br>
         <input type="radio" name="voteID" value="4">
         <font size="1" face="Verdana">Ruim</font><br>
         <br><br>
         <input name="submit" type="submit" value="Vote">
         <br>
         <font size="1" face="Verdana"> Resultados | Votação </font>
        </gia:regiao150>
       </gia:regiao200>
      </gia:regiao250>
     </gia:regiao300>
    </gia:regiao400>
    </tr>
    <tr>
    <td colspan="3" align="center">
      <font size="1" face="Verdana, Arial, Helvetica, sans-serif"><strong>Gerador de interfaces
adaptativas </strong> - http://gia.no-ip-info:8080/gia/</font>
   </td>
  </tr>
</table>
</body>
</gia:html>
```

APÊNDICE C

DESCRIÇÃO DAS CLASSES DOS SERVIÇOS DA ARQUITETURA GIA

C.1 Classes do Serviço de Comunicação

TABELA C.1 - Descrição da Classe Tdeli

Nome	*Tdeli*
Descrição	*Classe responsável por identificar o dispositivo.*
Atributos	
Nome	**Descrição**
content	*Variável utilizada pelo Deli.*
serialVersionUID	*Variável utilizada pelo Deli.*
Métodos	
Nome	**Descrição**
Tdeli()	*Construtor da classe.*
init()	*Busca as configurações do Deli.*
doGet(req:HttpServletRequest, res:HttpServletResponse)	*Método que captura a conexão e identifica o dispositivo.*

TABELA C.2 - Descrição da Classe DBase

Nome	*DBase*
Descrição	*Classe com as configurações da conexão com o banco de dados.*
Métodos	
Nome	**Descrição**
DBase()	*Construtor da classe.*
GetConexao():Connection	*Método utilizado para pegar a conexão com o banco de dados.*
Close()	*Método utilizado para fechar a conexão com o banco de dados.*
Release()	*Método utilizado para servlet.*

TABELA C.3 - Descrição da Classe TBrowser

Nome	*TBrowser*
Descrição	*Classe responsável manter o browser.*
Atributos	
Nome	**Descrição**
Nome	*Nome do browser.*
Codigo	*Código do browser.*
Versao	*Versão do browser.*
UserAgent	*User-Agente do browser.*
Métodos	
Nome	**Descrição**
TBrowser()	*Construtor da classe.*
setCodigo(codigo:int)	*Grava o código do browser.*
getCodigo():String	*Retorna o código do browser.*
setNome(nome:String)	*Grava o nome do browser.*
getNome():String	*Retorna o nome do browser.*
setVersao(versao:String)	*Grava a versão do browser.*
getVersao():String	*Retorna a versão do browser.*
setUserAgent(userAbgent:String)	*Grava o user-agent do browser.*
getUserAgent():String	*Retorna o user-agent do browser.*
clear()	*Limpar as variáveis*
save()	*Salvar os dados*
delete(código:int)	*Deletar os dados*
getCon()	*Retorna a conexão com o banco de dados*
setCon(con:DBase)	*Grava a conexão com o banco de dados.*

C.2 Classes do Serviço de Adaptação

TABELA C.4 - Descrição da Classe Br

Nome	*Br*
Descrição	*Classe utilizada para quebra de linha.*
Atributos	
Nome	**Descrição**
Métodos	
Nome	**Descrição**

TABELA C.5 - Descrição da Classe TComponente

Nome	*TComponente*
Descrição	*Classe responsável pela geração de componentes*
Atributos	
Nome	**Descrição**
Tipo	*Tipo do componente/nome do componente.*
Propriedade	*Propriedade dos componentes.*
TagCorpo	*Identifica se o componente possui corpo ou não.*
QLinha	*Identifica se o componente possui quebra de linha.*
VPropriedade	*Vetor com as propriedades do componente.*
BPropriedade	*Flag utilizada para informar se a tag possui proriedade.*
TotalP	*Informa a quantidade de propriedade do componente.*
TempComponente	*Variável temporária utilizada armazenar o nome do componente.*
Métodos	
Nome	**Descrição**
TComponente()	*Construtor da classe.*
doAfterBody()	*Método utilizado para manipular o conteúdo da tag.*
doStartTag()	*Método utilizado quando o servlet é acionado.*
release()	*Método utilizado pelo servlet.*
doEndTag()	*Método utilizado finalização do servlet.*

setTipo(valor:String)	*Método utilizado para informar o tipo do componente.*
setPropriedade(valor:String)	*Método utilizado para informar a propriedade do cor*
CVetor(valor:String)	*Método utilizado para converter uma string com as propriedade para um vetor.*
CStrVetor(valor:String[][]):String	*Método utilizado para converter um vetor de propriedade em um string.*
BuscaPropriedade(TPropriedade:String,CPropriedade:String):String	*Método utilizado para buscas as propriedades validas no dispositivo baseando-se na propriedade genérica.*
getBrowser()	*Método utilizado para repassar as informações do browser.*

TABELA C.5 – (Conclusão)

(Continua)

TABELA C.6 - Descrição da Classe TConexão

Nome	*TConexao*
Descrição	*Classe de conexão com o banco de dados.*
Atributos	
Nome	**Descrição**
Sit	*Informa o situação da conexão do banco.*
Métodos	
Nome	**Descrição**
TConexao()	*Construtor da classe.*
Open(_odbc:String,_user:String,_senha:String)	*Método utilizado para abrir conexão com o banco de dados.*
GetConexao():connection	*Método utilizado para pegar a conexão com o banco de dados.*
GetSituacao():String	*Método utilizado para buscar a situação do banco se ele esta aberto, fechado ou a conexão caiu.*
Close()	*Método utilizado para fechar a conexão com o banco de dados.*

TABELA C.7 - Descrição da Classe TQuery

Nome	*TQuery*
Descrição	*Classe utilizada para manipular o banco de dados*
Atributos	
Nome	**Descrição**
Sql	*Comando SQL.*
Sit	*Informa a situação da classe.*
Métodos	
Nome	**Descrição**
TQuery()	*Construtor da classe.*
SetConexao(con: Connection)	*Método utilizado para informar a classe TQuery qual banco poderá ser utilizado.*
Sql(_sql:String)	*Método utilizado para passar os comandos de manipulação do banco de dados.*
GetStatus ():String	*Método utilizado para buscar a situação/status da classe.*
Resultado():Result Set	*Método utilizado para pegar o resultado da consulta no banco.*
Close()	*Método utilizado para fechar os objetos de manipulação do banco.*
ExcuteSql()	*Executa comando SQL sem esperar resultado.*
GetStrXML():String	*Mostra o resultado da consulta em uma string no formato XML.*
GetDocXML():Document	*Mostra o resultado da consulta no tipo Document utilizado para manipular XML.*
formatTxt(txt:String):String	*Método utilizado para retirar acentuações das palavras.*
formatTxtHTML(txt:String):String	*Método utilizado para transformar palavras acentuadas em um padrão HTML.*

TABELA C.8 - Descrição da Classe THtml

Nome	*THtml*
Descrição	*Classe responsável pelo gerenciamento do GIA*
Atributos	

Nome	TABELA C.8 – Conclusão **Descrição**
menu	*Informa a necessidade de aparecer ou menu com as [illegible] das regiões para o dispositivo.*
rseq	*Informa qual a região deverá aparecer.*
size	*Informa o tamanho da tela do dispositivo*
arquvio	*Informa o nome da página ".jsp" que esta sendo acessada.*
ActiveRegiao	*Informa se a região está ativa ou não.*
Métodos	
Nome	**Descrição**
THtml()	*Construtor da classe.*
doAfterBody ()	*Método utilizado para mostrar o resultado na tela.*
doStartTag ()	*Método utilizado pelo servlet, também utilizado para pegar as informações do dispositivo.*
addRegiao(_seq:int ,_id:String,Coment ário:String,regiao: String,tela: int)	*Método utilizado para gravar os dados da região.*
release()	*Método utilizado pelo servlet.*
MontaMenu():Strin g	*Método utilizado para montar um menu com as informações das regiões.*
MontaRegiao():Stri ng	*Método utilizado para montar as regiões.*
isActiveRegiao():b oolean	*Método utilizado para verificar se a região está ou não ativa.*
setActiveRegiao(ac tiveRegiao:boolean)	*Método utilizado para gravar o status da região.*
SelecionaRegiao()	*Seleciona a região que será apresentada na tela do dispositivo.*
getBrowser()	*Retorna as informações do dispositivo.*

(Continua)

TABELA C.9 - Descrição da Classe TRegiao

Nome	*TRegiao*
Descrição	*Classe responsável por gerar as regiões.*
Atributos	
Nome	**Descrição**
Id	*Identificador da região.*

(Continua)

TABELA C.9 – Conclusão

Comentario	*Comentário da região.*
Seq	*Seqüência da região.*
Região	*Conteúdo da região.*
Size	*Tamanho da região.*
Métodos	
Nome	**Descrição**
TRegiao()	*Construtor da classe.*
doAfterBody ()	*Método utilizado pelo servlet.*
doStartTag ()	*Método utilizado pelo servlet, também utilizado para pegar as informações do dispositivo.*
getComentario():String	*Método utilizado para retornar o comentário da região.*
release()	*Método utilizado pelo servlet.*
setComentario(comentário:String)	*Método utilizado para gravar o comentário na região.*
getId():String	*Método utilizado para retornar o Id da regiões.*
setId(id:String)	*Método utilizado para gravar o Id da região.*
getSeq():int	*Método utilizado para retornar a seqüência da região.*
setSeq(seq:int)	*Método utilizado para gravar a seqüência da região.*
getSize():int	*Método utilizado para retornar o tamanho da região*
setSize(size:int)	*Método utilizado para gravar o tamanho da região.*
getRegiao	*Método utilizado para retorna o conteúdo da região.*
setRegiao(região:String)	*Método utilizado para gravar o conteúdo da região.*

TABELA C.10 - Descrição da Classe TFabricante

Nome	*TFabricante*
Descrição	*Classe responsável manter o fabricante.*
Atributos	
Nome	**Descrição**
Nome	*Nome do fabricante.*
Codigo	*Código do fabricante.*
Métodos	
Nome	**Descrição**

(Continua)

TABELA C.10 – (Conclusão)

TFabricante()	*Construtor da classe.*
setNome(nome:String)	*Grava o nome do fabricante.*
getNome():String	*Retorna o nome do fabricante*
setCodigo(código:int)	*Grava o código do fabricante*
getCodigo():int	*Retorna o código do fabricante*
clear()	*Limpar as variáveis*
save()	*Salvar os dados*
delete(código:int)	*Deletar os dados*
getCon()	*Retorna a conexão com o banco de dados*
setCon(con:DBase)	*Grava a conexão com o banco de dados.*

TABELA C.11 - Descrição da Classe TPerfil

Nome	*TPerfil*
Descrição	*Classe responsável manter o Perfil do dispositivo.*
Atributos	
Nome	**Descrição**
Codigo	*Código do perfil.*
Nome	*Nome do perfil*
Dsc	*Descrição do perfil*
Ativo	*Status do perfil.*
Métodos	
Nome	**Descrição**
TPerfil()	*Construtor da classe.*
setCodigo(código:int)	*Grava o código do perfil.*
getCodigo():int	*Retorna o código do perfil.*
setNome(nome:String)	*Grava o nome do perfil.*
getNome():String	*Retorna o nome do perfil.*
setDsc(dsc:String)	*Grava descrição do perfil.*
getDsc():String	*Retorna descrição do perfil.*

(Continua)

TABELA C.11 – (Conclusão)

setAtivo(ativo:String)	*Grava status.*
getAtivo():String	*Retorna status.*
clear()	*Limpar as variáveis*
save()	*Salvar os dados*
delete(código:int)	*Deletar os dados*
getCon()	*Retorna a conexão com o banco de dados*
setCon(con:DBase)	*Grava a conexão com o banco de dados.*

TABELA C.12 - Descrição da Classe TDispositivo

Nome	*TDispositivo*
Descrição	*Classe responsável manter o dispositivo.*
Atributos	
Nome	**Descrição**
Codigo	*Código do dispositivo.*
Nome	*Nome do dispositivo*
Tela	*Tamanho da tela do dispositivo*
Cores	*Quantidade de cores do dispositivo.*
Métodos	
Nome	**Descrição**
TDispositivo()	*Construtor da classe.*
setCodigo(código:int)	*Grava o código do dispositivo.*
getCodigo():int	*Retorna o código do dispositivo.*
setNome(nome:String)	*Grava o nome do dispositivo.*
getNome():String	*Retorna o nome do dispositivo.*
setTela(tela:String)	*Grava o tamanho da tela do dispositivo.*
getTela():String	*Retorna o tamano da tela do dispositivo.*
setCores(cores:String)	*Grava a quantidade de cores do dispositivo.*
getCores():String	*Retorna a quantidade de cores do dispositivo.*
clear()	*Limpar as variáveis*

(Continua)

TABELA C.12 – (Conclusão)

save()	*Salvar os dados*
delete(código:int)	*Deletar os dados*
getCon()	*Retorna a conexão com o banco de dados*
setCon(con:DBase)	*Grava a conexão com o banco de dados.*

APÊNDICE D

MANUAL DE INSTALAÇÃO DO AMBIENTE GIA

Este manual tem a finalidade de orientar o usuário sobre os requisitos do sistema e os procedimentos para a sua instalação.

D.1 Requisitos do Sistema

Requisitos de Hardware	
Descrição	A máquina a qual será instalado o sistema deverá possuir os seguintes componentes de hardware: um processador de 1500Mhz ou superior, 128 MB de memória RAM ou superior e espaço em disco (HD) de 100 MB.

Requisitos de Software	
Descrição	A máquina a qual será instalado o sistema deverá possuir os seguintes componentes de software: sistema operacional Windows 2000 ou superior, Postgres SQL 8.1.4 ou superior, servidor Apache Tomcat 5.0 ou superior.

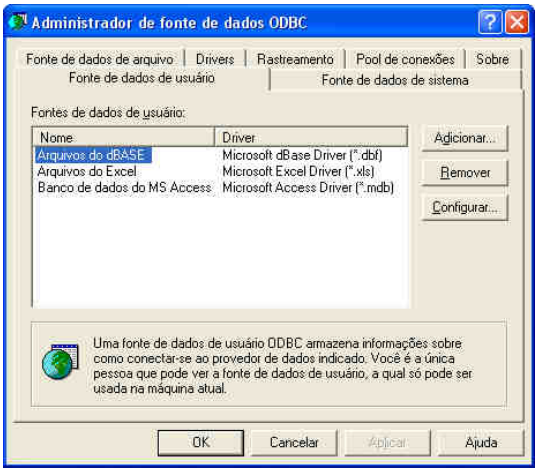
D.2 Instalação

A instalação da ferramenta é composta por três passos, descritos a seguir:

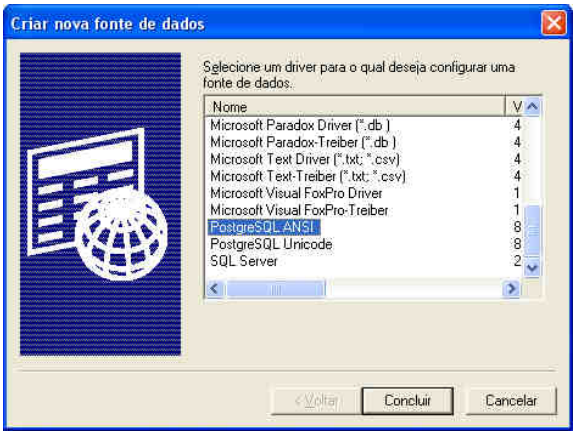
1. Criar e configurar uma conexão ODBC;
2. Criar o banco de dados, através de um script;
3. Instalar o programa de instalação.

D.2.1 Criação e configuração de uma conexão ODBC

1. O usuário deverá selecionar no menu iniciar, a opção painel de controle.
2. Após, deverá selecionar a opção Ferramentas Administrativas.
3. Selecionar a opção Fonte de Dados (ODBC).
4. Clicar no botão adicionar.



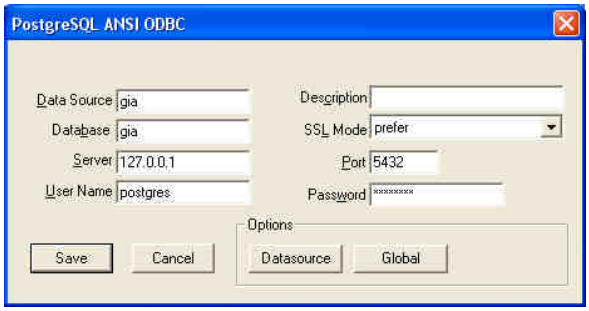
5. Selecionar na lista exibida, a opção PostgreSQL ANSI e clicar em concluir.



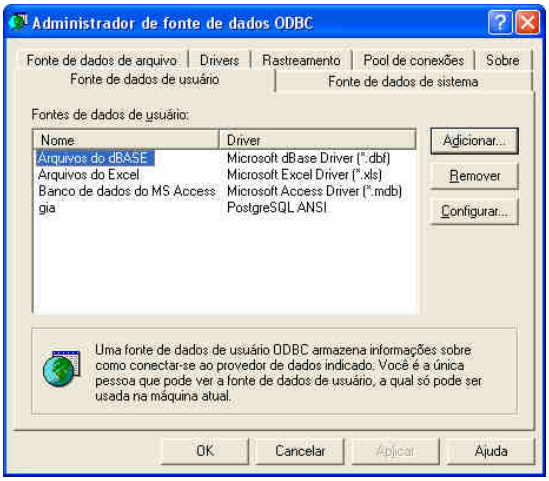
6. Preencher os campos com os seguintes dados:

Data Source: gia
Database: gia
SSL Mode: Prefer
Server: 127.0.0.1
User Name: postgres
Password: deverá ser informada a senha do usuário postgres do banco PostgreSQL.

Após o preenchimento destes campos deverá clicar no botão Save.

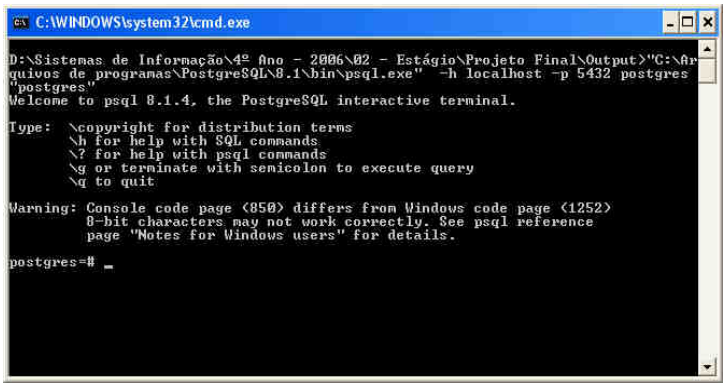


7. Concluído esta configuração, o usuário deverá clicar no botão OK.



D.1.2 Criação do Banco de Dados

1. O usuário deverá abrir o arquivo script.txt, disponível na pasta inicial do CD de instalação, e copiar todo o conteúdo deste arquivo.
2. Logo após, executar o arquivo postgres.bat, que está disponível na pasta inicial do CD de instalação. Será exibida a tela a seguir.



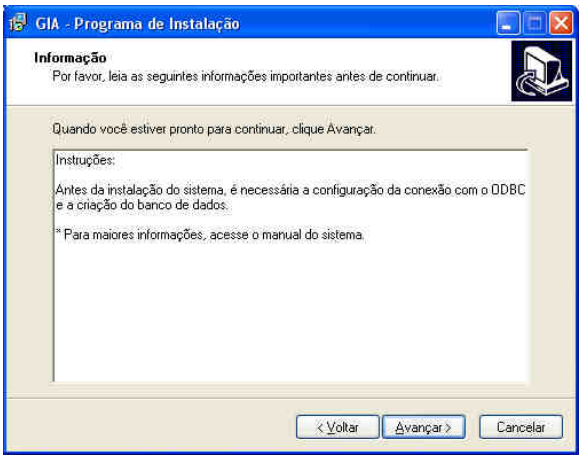
3. Feito isso, o usuário deverá clicar com o botão direito do mouse e selecionar a opção colar (é colado o script que foi copiado no passo 1.

4. O banco é criado em alguns segundos; após este procedimento o usuário deverá pressionar a tecla ENTER;

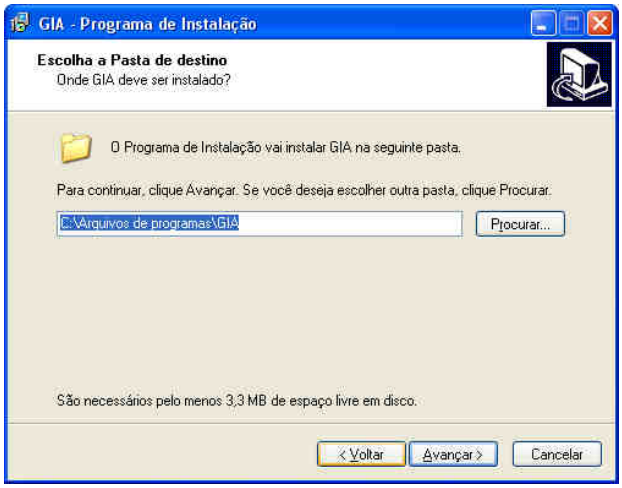
D.1.3 Instalação do Programa

1. O usuário deverá selecionar o arquivo executável GIA Setup, que está disponível na pasta inicial do CD de instalação.
2. Será exibida a tela a seguir, onde o usuário deve clicar no botão avançar.

3. Na próxima tela clicar no botão Avançar.

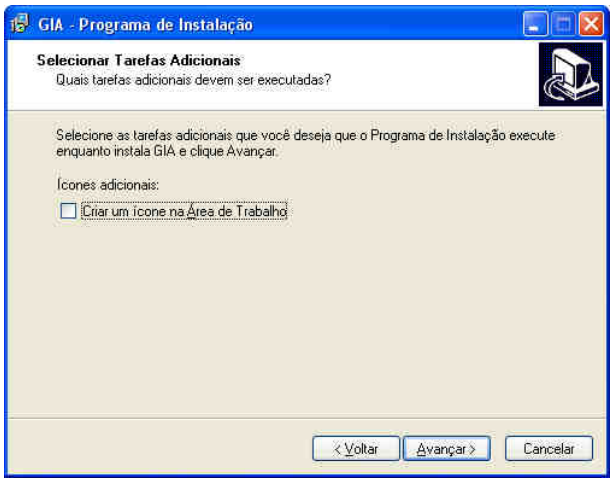


4. Na próxima tela, o usuário deverá escolher o local onde será instalado o programa. Clicar no botão Avançar.

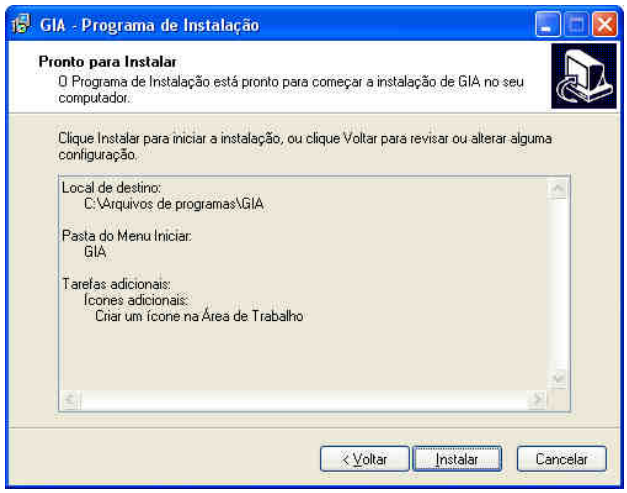


5. Na próxima tela, escolher qual o nome para a pasta do programa que será criada no Menu Iniciar – Programas. Clicar em Avançar.

6. Em seguida, o usuário pode escolher se deseja instalar um atalho para o programa na área de trabalho. Após escolher, deverá clicar no botão Avançar.



7. Serão exibidos as configurações para a instalação do programa. Caso elas estejam corretas, clicar no botão Instalar.



8. O programa será instalado e é exibida uma tela que informa a finalização da instalação do programa. O usuário deverá clicar no botão Concluir.

APÊNDICE E

DESCRIÇÃO DE PERFIL DE DISPOSITIVO MÓVEL

E.1 Perfil do Dispositivo Palm Zire 72

```
<?xml version="1.0" ?>
 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:ccpp="http://www.w3.org/2000/07/04-ccpp#"
xmlns:neg="http://www.inrialpes.fr/opera/people/Tayeb.Lemlouma/NegotiationSchema/Client
ProfileSchema-03012002#">
   <rdf:Description rdf:ID="ClientProfile">
    <ccpp:component>
     <rdf:Description rdf:ID="HardwarePlatform">
          <rdf:type
rdf:resource="http://www.inrialpes.fr/opera/people/Tayeb.Lemlouma/NegotiationSchema/Clie
ntProfile-03012002#HardwarePlatform" />
          <neg:DeviceType>Palm</neg:DeviceType>
          <neg:DeviceName>Zire 72s</neg:DeviceName>
          <neg:DeviceConstructor>Paml Inc.</neg:DeviceConstructor>
          <neg:screen>320x320Pixels</neg:screen>
          <neg:screenColor>yes</neg:screenColor>
          <neg:RAMSize>32Mo</neg:RAMSize>
          <neg:systemProcessor>PXA270</neg:systemProcessor>
     </rdf:Description>
   </ccpp:component>
   <ccpp:component>
    <rdf:Description rdf:ID="SoftwarePlatform">
     <rdf:type
rdf:resource="http://www.inrialpes.fr/opera/people/Tayeb.Lemlouma/NegotiationSchema/Clie
ntProfile-03012002#SoftwarePlatform" />
          <neg:PlatformName>Palm OS Garnet</neg:PlatformName>
          <neg:PlatformVersion>5.2.8</neg:PlatformVersion>
     </rdf:Description>
   </ccpp:component>
   <ccpp:component>
    <rdf:Description rdf:ID="BrowserUA">
     <rdf:type
rdf:resource="http://www.inrialpes.fr/opera/people/Tayeb.Lemlouma/NegotiationSchema/Clie
ntProfile-03012002#BrowserUA"/>
        <neg:UsedPlayerName>WebPro</neg:UsedPlayerName>
        <neg:UsedPlayerVersion>3.5</neg:UsedPlayerVersion>
        <neg:OnlySupportedResources>
            <rdf:Bag>
             <rdf:li rdf:parseType="Resource">
                    <neg:type>HTML</neg:type>
                    <neg:version>4.0.1</neg:version>
                    <neg:format>html</neg:format>
                    <neg:format>htm</neg:format>
            </rdf:li>
        </rdf:Bag>
       </neg:OnlySupportedResources>
```

```
        <neg:NonSupportedResources>
         <rdf:Bag>
              <rdf:li rdf:parseType="Resource">
                        <neg:type>DHTML</neg:type>
                        <neg:type>VBScript</neg:type>
                        <neg:type>Java VM</neg:type>
                        <neg:type>VRML</neg:type>
              </rdf:li>
         </rdf:Bag>
        </neg:NonSupportedResources>
        <neg:PrederredSupportedResources>
         <rdf:Bag>
              <rdf:li rdf:parseType="Resource">
                        <neg:type>language</neg:type>
                        <neg:format>English</neg:format>
                        <neg:priorityLevel>1</neg:priorityLevel>
                        <neg:type>language</neg:type>
                        <neg:format>Spanish</neg:format>
                        <neg:priorityLevel>2</neg:priorityLevel>
                        <neg:type>language</neg:type>
                        <neg:format>Protuguese</neg:format>
                        <neg:priorityLevel>3</neg:priorityLevel>
              </rdf:li>
         </rdf:Bag>
        </neg:PrederredSupportedResources>
      </rdf:Description>
    </ccpp:component>
</rdf:Description>
</rdf:RDF>
```

PUBLICAÇÕES TÉCNICO-CIENTÍFICAS EDITADAS PELO INPE

Teses e Dissertações (TDI)

Teses e Dissertações apresentadas nos Cursos de Pós-Graduação do INPE.

Manuais Técnicos (MAN)

São publicações de caráter técnico que incluem normas, procedimentos, instruções e orientações.

Notas Técnico-Científicas (NTC)

Incluem resultados preliminares de pesquisa, descrição de equipamentos, descrição e ou documentação de programa de computador, descrição de sistemas e experimentos, apresentação de testes, dados, atlas, e documentação de projetos de engenharia.

Relatórios de Pesquisa (RPQ)

Reportam resultados ou progressos de pesquisas tanto de natureza técnica quanto científica, cujo nível seja compatível com o de uma publicação em periódico nacional ou internacional.

Propostas e Relatórios de Projetos (PRP)

São propostas de projetos técnico-científicos e relatórios de acompanhamento de projetos, atividades e convênios.

Publicações Didáticas (PUD)

Incluem apostilas, notas de aula e manuais didáticos.

Publicações Seriadas

São os seriados técnico-científicos: boletins, periódicos, anuários e anais de eventos (simpósios e congressos). Constam destas publicações o Internacional Standard Serial Number (ISSN), que é um código único e definitivo para identificação de títulos de seriados.

Programas de Computador (PDC)

São a seqüência de instruções ou códigos, expressos em uma linguagem de programação compilada ou interpretada, a ser executada por um computador para alcançar um determinado objetivo. São aceitos tanto programas fonte quanto executáveis.

Pré-publicações (PRE)

Todos os artigos publicados em periódicos, anais e como capítulos de livros.