Universidade Federal do Ceará
Departamento de Computação
Mestrado em Ciência da Computação

Dissertação de Mestrado

Um Ambiente de Desenvolvimento de Aplicações Multi-Plataformas e Adaptativas para Dispositivos Móveis

Windson Viana de Carvalho

Fortaleza-CE, 2005

Um Ambiente de Desenvolvimento de Aplicações Multi-Plataformas e Adaptativas para Dispositivos Móveis

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Windson Viana de Carvalho e aprovada pela Banca Examinadora.

Fortaleza, 14 de junho de 2005

Rossana Maria de Castro Andrade
(Orientadora)

Dissertação apresentada ao Mestrado de Ciência da Computação da Universidade Federal do Ceará (UFC), como requisito parcial para a obtenção do título de Mestre em Ciência da Computação

RESUMO

A heterogeneidade dos dispositivos móveis e a integração com as tecnologias de comunicação sem fio impõem desafios para o desenvolvimento de aplicações e de serviços, dos quais podemos citar os seguintes: a descrição, independente de dispositivo e plataforma, da interface com usuário; a adaptação do conteúdo acessado pelas aplicações; e o desenvolvimento de aplicações multi-plataformas.

Diversos trabalhos foram propostos para solucionar cada um desses desafios. Contudo, não foi encontrada na literatura nenhuma abordagem satisfatória que permitisse a um engenheiro de software construir aplicações multi-plataformas utilizando uma descrição inicial da interface e dos seus dados, de forma que essa descrição seja independente de um dispositivo específico ou de uma plataforma de programação. Ressalta-se, ainda, que nenhuma solução facilita a descrição da aplicação e a sua integração com as arquiteturas de adaptação de conteúdo.

Portanto, é necessária a criação de um ambiente para integrar e aprimorar essas soluções existentes para os desafios expostos. Esta dissertação propõe então um ambiente para o desenvolvimento de aplicações para dispositivos móveis composto pelos *frameworks* XFormUI, Requisitor, Mobile Adapter e MobAC e pela ferramenta de geração de código User Interface Generator (UIG). Esse ambiente propicia o desenvolvimento rápido de aplicações multi-plataformas e adaptáveis à descrição e oferece mecanismos eficientes para captura, gerenciamento e manipulação dos perfis do dispositivo, do usuário e do contexto em que executam, o que auxilia a construção de *Mobile Application Servers* (MAS) para realizar adaptação de conteúdo.

Um estudo de caso, que consiste em um sistema para adaptação de imagens fotográficas é também apresentado para ilustrar como os *frameworks* e a ferramenta UIG podem ser utilizados para construir sistemas complexos para dispositivos móveis.

ABSTRACT

The heterogeneity of mobile devices and their integration with wireless communication technologies impose challenges for the development of applications and services, such as: device and platform independent user interface description; content adaptation for mobile applications; and development of multi-platform applications.

Several works have been proposed to solve these challenges. However, no satisfactory approach has been found in the literature that allows a software engineer to specify multi-platform applications using an initial description of their interface and data as well as using an independent description of a specific device or a programming platform. Moreover, it has not been found in the literature solutions that facilitate the description of an application and its integration with content adaptation architectures.

Therefore, it is necessary the creation of an environment to integrate and improve these existing solutions. This master thesis proposes an environment for mobile device applications composed by the XFormUI, Requisitor, Mobile Adapter and MobAC frameworks and the User Interface Generator (UIG), which is a code generation tool. This environment offers the fast development of multi-platform and description adaptable applications. Moreover, it provides efficient mechanisms for capture, management and manipulation of device, user and context profiles. These mechanisms help the construction of Mobile Application Server (MAS) that executes content adaptation.

A case study, which consists in a system for adaptation of photographic images, is also presented in this work to illustrate how the frameworks and the UIG tool is used to build complex systems for mobile devices.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CC/PP	*Composite Capability/Preference Profiles*
CLDC	*Connected Limited Device Configuration*
CSS	*Cascading Style Sheets*
DM	*Dispositivo Móvel*
EDGE	*Enhanced Data GSM Environment*
GPL	*GNU General Public License*
GPRS	*General Packet Radio Service*
GSM	*Global System for Mobile Communications*
HTML	*HyperText Markup Language*
IDE	*Integrated Development Environment*
IEEE	*Institute of Electrical and Electronic Engineers*
IP	*Internet Protocol*
J2EE	*Java 2 Enterprise Edition*
J2ME	*Java 2 Micro Edition*
J2SE	*Java 2 Standard Edition*
KVM	*Kilobyte Virtual Machine*
LAN	*Local Area Network*
LGPL	*Lesser General Public License*
MAS	*Mobile Application Server*
MIDP	*Mobile Information Device Profile*
MobAC	*Mobile Adapter Client*
NAC	*Negotiation and Adaptation Core*
PDA	*Personal Digital Assitant*
RTP	*Real-Time Transport Protocol*
SDK	*Software Development Kit*
SMIL	*Synchronized Multimedia Integration Language*
SQL	*Structured Query Language*
UAProf	*User Agent Profiling*

UID	*User Interface Description*
UIG	*User Interface Generator*
UPnP	*Universal Plug and Play*
VoiceXML	*Voice Extensible Markup Language*
W3C	*World Wide Web Consortium*
WAP	*Wireless Application Protocol*
WLAN	*Wireless Local Area Network*
WML	*Wireless Markup Language*
WTK	*Wireless Toolkit*
XHTML	*Extensible HyperText Markup Language*
XML	*eXtended Markup Language*
XSL	*eXtensible Stylesheet Language*
XSLT	*eXtensible Stylesheet Language Transformation*

1. Introdução

Esta dissertação apresenta um ambiente para desenvolvimento de aplicações multi-plataforma e adaptativas para dispositivos móveis. Neste capítulo, é apresentada na Seção 1.1 uma visão geral sobre o desenvolvimento de aplicações para dispositivos móveis e a caracterização do problema que gerou a motivação deste trabalho. Na Seção 1.2, são descritos os objetivos principais da dissertação, e, por fim, na Seção 1.3 é apresentada a estrutura na qual está organizado o restante do trabalho.

1.1. Caracterização do Problema e Motivação

O desenvolvimento de aplicações para dispositivos móveis tornou-se uma área de grande interesse para a computação devido ao aumento da disponibilidade desses dispositivos no mercado e a sua crescente integração com as tecnologias de comunicação sem fio [40]. Celulares com suporte a WAP, GPRS ou EDGE e *handhleds*, como Palms e Pocket PCs, com Bluetooth [42] e IEEE 802.11 [73] fazem crescer a demanda por novos tipos de serviços e aplicações nas mais diversas áreas (e.g., sistemas de vendas em campo, coletores de informação, aplicações de entretenimento).

Essa integração dos dispositivos móveis (DMs) com as redes de comunicação de dados, permitiu aos sistemas corporativos e aos sistemas *Web,* que antes interagiam somente com computadores *desktops,* serem acessados também por dispositivos móveis [9]. Contudo, esses DMs apresentam diferentes restrições de processamento, memória, bateria e largura de banda. Eles possuem ainda diferenças nas formas de interação do usuário com as aplicações e nas dimensões e cores dos *displays*. Além disso, o suporte a plataformas de programação varia de um dispositivo para outro, dificultando a adoção de uma única plataforma para o desenvolvimento das aplicações.

Desta forma, a heterogeneidade dos dispositivos e a integração com as tecnologias de comunicação sem fio têm imposto desafios para o desenvolvimento de aplicações. Entre os desafios encontrados na pesquisa bibliográfica realizada nesta dissertação [22][76][12][25][26], destacam-se os seguintes: a descrição, independente de dispositivo e

plataforma, da interface com usuário; a adaptação do conteúdo acessado pelas aplicações; e o desenvolvimento de aplicações multi-plataformas.

O problema da descrição das interfaces das aplicações consiste na dificuldade que o engenheiro de software encontra em projetar as interfaces com usuário para que estas possam se adequar a diferentes tipos de dispositivos e executar em diferentes plataformas de programação. Para solucionar esse desafio, podemos citar os seguintes trabalhos: o UIML (*User Interface Markup Language*) [14], o *Icrafter* [21], o UID (*User Interface Description*) [16] , o MAB (*Mobile Application Builder*) [17] e MultiMad [75]. Esses trabalhos propõem a definição da interface utilizando documentos XML (i.e., *eXtended Markup Language*) [36]. Essa definição é feita através de marcações (i.e., *tags*) pré-estabelecidas para a criação dos componentes de interface. Depois, geradores de código transformam a descrição inicial em código executável ou em documentos de uma linguagem de marcação (e.g., XHTML). No caso do MAB e do MultiMad, ferramentas gráficas são disponibilizadas para facilitar a criação das interfaces das aplicações.

Outros trabalhos foram propostos para prover a adaptação de conteúdo baseada nas características dos dispositivos, nas preferências dos usuários e no contexto do ambiente em que o dispositivo se encontra (e.g., mudança na largura de banda). Algumas dessas soluções utilizam *proxys ou gateways* para realizar a tarefa de adaptação, como em [10], [11] e [12]. Contudo, esses trabalhos concentram toda a decisão da adaptação em uma arquitetura geral, podendo não ser satisfatória para certos tipos de aplicações que não consistam na simples requisição de um conteúdo Web. Além disso, essas soluções não apresentam formas eficientes de captura e gerenciamento dos perfis do usuário e do dispositivo.

Conforme exposto anteriormente, existem soluções para os desafios citados no desenvolvimento de aplicações para DMs. Entretanto, não foi encontrada na literatura nenhuma abordagem satisfatória que permita a um engenheiro de software construir aplicações multi-plataformas utilizando uma descrição inicial da interface e dos seus dados, de forma que essa descrição seja independente de um dispositivo específico ou de uma plataforma de programação. Vale ressaltar ainda que nenhuma solução vislumbra simultaneamente a descrição da aplicação e a sua integração com as arquiteturas de

adaptação de conteúdo. Portanto, é necessária a criação de um ambiente para integrar e aprimorar essas soluções existentes para os desafios expostos.

1.2. Objetivos

Na Seção 1.1, foi destacada a necessidade de integrar e aprimorar as soluções existentes para os desafios no desenvolvimento de aplicações para DMs. Com o objetivo principal de suprir essa necessidade, este trabalho apresenta um ambiente para a programação em dispositivos móveis a fim de facilitar, para o engenheiro de software, o desenvolvimento de aplicações multi-plataformas e adaptativas.

Esse ambiente é composto pelos *frameworks* XFormUI, Mobile Adapter, Mobile Adapter Cliente (MobAC), Requisitor e pela ferramenta de geração de código User Interface Generator (UIG). Com esse ambiente, o engenheiro de software pode construir mais rapidamente tanto aplicações que executam somente no dispositivo, como aplicações que trocam dados com servidores remotos, denominados Mobile Application Servers (MAS) [74]. Nos dois tipos de aplicações, o ambiente permite a construção de protótipos utilizando uma descrição independente de uma plataforma de programação. E no caso da aplicação que acessa um MAS, o ambiente provê um conjunto de *frameworks* que auxiliam a construção do MAS e do cliente que executa no dispositivo móvel.

Para atingir o objetivo de construir aplicações multi-plataformas com interfaces adaptáveis à descrição, o ambiente proposto permite ao engenheiro descrever os formulários da aplicação, suas validações e o estilo das interfaces em XForms [15]. Com esses documentos, o engenheiro utiliza a ferramenta, User Interface Generator (UIG), para produzir código em uma plataforma de programação. Esse código gerado utiliza o *framework* multi-plataforma XFormUI para a composição dos formulários, o que torna o protótipo da aplicação multi-plataforma. Ao final do processo de geração, o engenheiro complementa a codificação acrescentando a lógica de negócio inerente à aplicação. Na Figura 1, uma visão geral desse processo de codificação é apresentado.

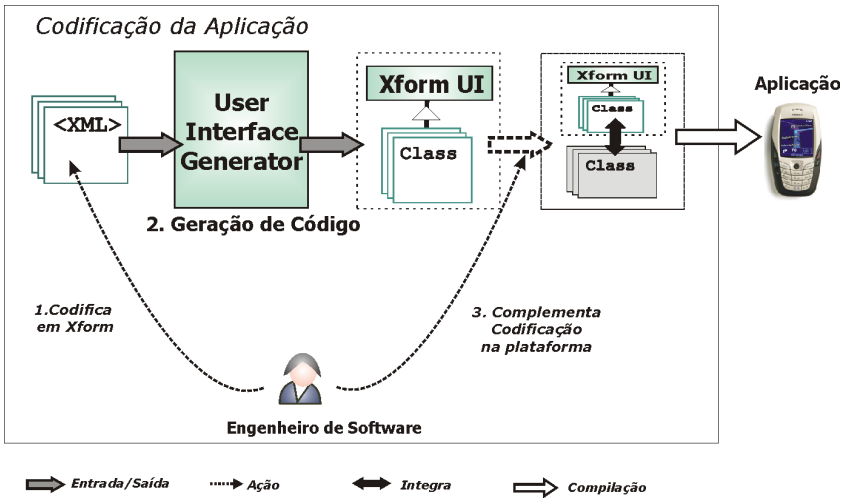


Figura 1 - Visão geral da codificação da aplicação que executa no dispositivo móvel

Nos casos em que o engenheiro precisa criar aplicações que acessam um MAS, o ambiente fornece dois *frameworks*: Mobile Adapter e MobAC. O Mobile Adapter facilita a construção de MAS adaptativos, pois esse *framework* provê de forma eficiente informações sobre os perfis do dispositivo, do contexto e do usuário que acessa o MAS. O MobAC permite a construção das aplicações clientes que executam no dispositivo móvel e acessam os MAS construídos com o Mobile Adapter.

Na Figura 2, é apresentada uma visão geral do processo de construção dos MAS adaptativos usando os *frameworks* propostos. Para a construção da aplicação cliente, o engenheiro utiliza o código dos formulários gerados pela UIG (1) e codifica a integração com o MAS usando o *framework* MobAC (2). Na construção do servidor, o engenheiro codifica o MAS utilizando o Mobile Adapter (I) e constrói os pacotes de adaptação de conteúdo baseados nas informações fornecidas pelo *framework* (II). Ao final do processo, o engenheiro construiu uma aplicação multi-plataforma adaptável a descrição para DMs que é capaz de coletar informações dinâmicas sobre o dispositivo e enviá-las ao MAS. Além disso, o MAS construído pode recuperar as informações estáticas do dispositivo e do usuário e as características dinâmicas do contexto, o que possibilita a realização da adaptação do conteúdo requisitado pela aplicação do DM.

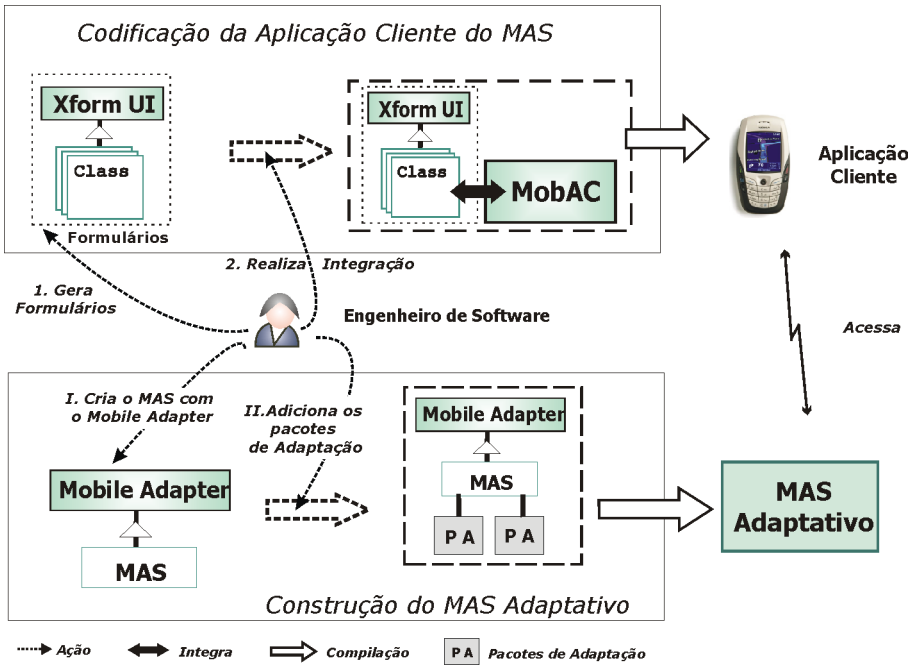


Figura 2 - Visão geral do processo de construção de MAS Adaptativos

Para prover o desenvolvimento de aplicações multi-plataformas e adaptativas, os *frameworks* MobAC e XFormUI são implementados em três plataformas de programação J2ME MIDP 1.0/CLDC 1.0 [43], J2ME MIDP 2.0/CLDC 1.1 [43] e Superwaba [44]. Além disso, o ambiente proposto é projetado para facilitar a construção de novas implementações dos *frameworks* em outras plataformas.

1.3. Organização da Dissertação

Essa dissertação está dividida em mais 8 capítulos que serão descritos a seguir:

- No Capítulo 2, é apresentado um conjunto de definições para adaptação e são detalhados os desafios e as soluções existentes descritos na Seção 1.1 no desenvolvimento de aplicações para dispositivos móveis.
- No Capítulo 3, é apresentada uma introdução sobre *frameworks* e sobre as plataformas de programação investigadas para a construção e a implementação do ambiente proposto na dissertação.

- No Capítulo 4, são enumerados os requisitos desejados para a construção de um *framework* de interface gráfica e é apresentado o projeto e os detalhes de implementação do *framework* XFormUI para a construção de interfaces gráficas multi-plataformas.
- No Capítulo 5, é apresentada a ferramenta de geração de código User Interface Generator (UIG) proposta nesta dissertação para agilizar o desenvolvimento de aplicações para dispositivos móveis.
- No Capítulo 6, são apresentados os *frameworks* Mobile Adapter, MobAC e Requisitor que, em conjunto, permitem a construção de *Mobile Aplication Servers* (MAS) que adaptam conteúdo e seus clientes para DMs.
- No Capítulo 7, é descrito um estudo de caso construído para exemplificar como o ambiente proposto pode ser utilizado para construir um MAS que adapta conteúdo e uma aplicação para DM multi-plataforma e adaptável a descrição. Uma avaliação da eficiência do uso da adaptação de imagens contida no estudo de caso é também apresentada.
- No Capítulo 8, a conclusão da dissertação é apresentada com as principais contribuições e os resultados alcançados, além de sugestões de trabalhos futuros.

2. Adaptação para DMs: Desafios e Soluções

Neste capítulo, são detalhados os desafios e as soluções existentes no cenário atual do desenvolvimento de aplicações para DMs. Como descrito no Capítulo 1, esses desafios são impostos pela heterogeneidade e pela integração desses dispositivos com as tecnologias de comunicação sem fio.

Na Seção 2.1, são apresentadas a definição e os diferentes modos de adaptação. Na Seção 2.2, soluções para a descrição, independente de dispositivo e de plataforma, da interface das aplicações são descritas. Por fim, na Seção 2.3, são apresentadas arquiteturas de adaptação de conteúdo encontradas na literatura.

2.1. Definição e Características de Adaptação

Adaptação, de forma mais geral, é definida como o relacionamento entre um organismo e seu ambiente tal que uma mudança ou uma coleção de mudanças do organismo resultam em uma melhor adequação em um contexto particular [76]. Portanto, ser adaptável significa ter a habilidade de modificar o comportamento para responder a eventuais mudanças de um ambiente. Analogamente, em ciência da computação, uma aplicação é adaptável quando têm a capacidade de se adequar, de forma ótima, a mudanças nas circunstâncias em que executa [22]. Assim, se o ambiente de execução mudar, a aplicação é capaz de se modificar para executar de uma forma mais eficiente nas novas condições.

No caso de aplicações para DMs, a heterogeneidade e a dinamicidade do ambiente de computação móvel tornam a adaptação da aplicação uma técnica necessária. M. Wieiser denomina de *calm computing* [23] a adaptação de uma aplicação sem a intervenção humana. No desenvolvimento de aplicações adaptativas esse é o objetivo a ser alcançado [24]. Contudo, a construção desse tipo de aplicação representa um obstáculo ainda maior para o engenheiro de software.

A adaptação de uma aplicação para DMs pode ocorrer de várias formas e em diferentes momentos. Entre os modos de adaptação, destacam-se os seguintes [81][27][26][25][76]:

- **Adaptação à descrição**: é a capacidade de descrever uma aplicação permitindo sua adequação ou transformação em diferentes linguagens ou plataformas de programação.

Nesse caso, a aplicação é descrita em uma meta-linguagem e é transformada em uma linguagem de programação (e.g., Java, C++) ou em uma linguagem de marcação de hipertexto como *Extensible HyperText Markup Language* (XHTML) ou *Wireless Markup Language* (WML).

- **Adaptação ao dispositivo**: é a habilidade de uma aplicação de adequar seu modo de execução às características do dispositivo. Essas características podem ser estáticas, como número de cores e dimensões da tela, e dinâmicas, como a quantidade de memória e bateria disponíveis. Essa adaptação pode ser específica para um dispositivo ou ocorrer também em relação a uma classe deles (e.g., celulares, *handhelds*).
- **Adaptação ao contexto:** é a propriedade de uma aplicação de adequar-se a mudanças no contexto em que executa. As mudanças no contexto podem ser decorrentes, por exemplo, das alterações da localização do dispositivo, do interesse do usuário e da largura de banda de comunicação.

Esses modos de adaptação podem acontecer tanto em tempo de construção como em tempo de execução. A **adaptação em tempo de construção** ocorre quando uma aplicação é transformada para se adequar à mudança de um ambiente antes de executar nele. Esse tipo de adaptação exige a intervenção do engenheiro de software e é caracterizada pela construção de novas versões da aplicação. Por exemplo, uma aplicação que tem uma versão para executar em celulares e uma versão para executar em *handhelds*.

No caso da **adaptação em tempo de execução**, a aplicação adaptável modifica seu comportamento, sem a intervenção do engenheiro de software, para atender as mudanças no ambiente em que já está executando. Por exemplo, uma aplicação de visualização de *streams* de vídeos capaz de modificar as propriedades do vídeo para se adequar às variações na banda passante do canal de comunicação.

Os modos de adaptação apresentados anteriormente não estão restritos a uma mudança no modo de execução das aplicações nos DMs. A adaptação pode também estar relacionada à adequação dos dados que a aplicação acessa. Nesse caso, a adaptação é denominada de **adaptação de conteúdo**.

2.2. Descrição, independente de dispositivo e plataforma, da interface das aplicações

Como foi mencionado anteriormente, uma das formas de adaptação é a adaptação à descrição. Nesse tipo de adaptação, a aplicação tem que ser descrita em uma meta linguagem, independente de uma linguagem de programação ou de marcação, que executa em um determinado dispositivo. Para prover esse tipo de adaptação, foram propostos trabalhos [14] [21] [16] [17] que utilizam documentos XML para definir a interface da aplicação e, em alguns casos, definir também seus dados.

Descrever uma interface de uma aplicação independente de uma plataforma de programação e de um dispositivo, é uma tarefa complexa, já que esses dispositivos possuem diversas formas de exibição de dados e de interação com o usuário. Portanto, descrever, por exemplo, que uma interface de uma aplicação é composta por botões e reage a eventos de *click,* pode não ser adequado.

Com o objetivo de solucionar a questão da definição da interface, os trabalhos descritos nas subseções seguintes sugerem uma outra forma de descrição dos componentes e dos eventos das aplicações. Cada trabalho será apresentado através de um resumo contendo seus objetivos, resultados alcançados, propostas de trabalhos futuros e uma classificação quanto a tipo de adaptação de acordo com as definições apresentadas na Seção 2.1. No final dessa seção, é apresentado um resumo comparativo das soluções.

2.2.1. "Adaptive Applications for Ubiquitous Collaboration"

Em [27], é descrita uma arquitetura para a provisão de adaptação para aplicações móveis em ambientes colaborativos heterogêneos. Um ambiente colaborativo é caracterizado pela existência de vários usuários colaborando e interferindo sobre uma determinada coleção de objetos ou de dados. No caso de [27], as aplicações executam em diferentes dispositivos e devem ser capazes de acessar, interferir e perceber alterações do ambiente colaborativo.

Os autores identificam três requisitos para garantir a adaptação nesses ambientes: a descrição, independente de dispositivo, da interface da aplicação que possa ser facilmente mapeada para uma implementação real; a adaptação dinâmica da interface com usuário, baseada em mudanças contextuais, tais como, alterações no interesse do usuário ou na

largura de banda; e o suporte à visualização heterogênea dos dados, na qual a visualização é adaptada às potencialidades do dispositivo.

No trabalho é apresentada uma arquitetura para suportar o desenvolvimento de ferramentas para facilitar a colaboração em ambientes heterogêneos. Essa arquitetura define uma linguagem para a descrição genérica das aplicações e mapeia essa descrição em uma implementação específica. A linguagem de descrição é composta de um conjunto de componentes chamados de *interactors*, semelhante à abordagem do XWeb[28], que correspondem à visualização de um objeto ou de um recurso do mundo compartilhado (e.g., um robô em uma sala, um conjunto de lâmpadas controladas por vários dispositivos).

Na arquitetura, foi definida uma linguagem baseada em XML para a especificação desses *interactors.* XSL (*eXtensible Stylesheet Language*) [36] e processadores XSLT (XSL *Transformation*) mapeiam essa especificação em uma visualização adaptada a classe do dispositivo alvo. No caso de um computador, é gerada uma visualização tri-dimensional, e no caso de um PDA, são geradas plantas bidimensionais.

A arquitetura foi definida para possibilitar que eventos externos ou eventos indiretos das aplicações de visualização (e.g., mover o robô pra frente) sejam refletidos em todas as visualizações. Dessa forma, toda a lógica e os dados da aplicação concentram-se no servidor. Esse, por sua vez, gera a aplicação a ser enviada aos dispositivos quando esses requisitam ou quando ocorre alguma alteração nos objetos do ambiente colaborativo.

Como foi mencionado, o processo de geração da aplicação acontece no servidor, que é responsável por mapear a descrição da aplicação e dos dados em uma visualização Java que é enviada para o dispositivo que fez a requisição da aplicação. Desta forma, a arquitetura provê **adaptação em tempo de execução em relação à descrição e a classes de dispositivos**. Na Figura 3, é ilustrado esse mecanismo de mapeamento.

Para testar a arquitetura, foi construída uma aplicação, o SlowTetris, que possui quatro objetos no ambiente colaborativo. Os dados dos objetos, como dimensões e localização, foram descritos utilizando os *interactor*s. Esses eram mapeados em *cWorld,* um *Software Development Kit* (SDK) de Java 3D, e em PocketEscape, um ambiente de visualização bidimensional em Pocket PC. O SlowTetris possibilita apenas a visualização dos quatro objetos, sem eventos de interação da aplicação móvel com eles.

Essa abordagem torna a aplicação móvel uma visualização do mundo compartilhado permitindo poucos eventos e limitando as possibilidades de aplicações que possam ser construídas. Entretanto, o uso de XML para descrever a interface da aplicação e o uso de XSLT para mapeá-la representam boas estratégias para o provimento de adaptação à descrição. Como trabalhos futuros, foi apresentada a possibilidade de serem geradas aplicações J2ME MIDP [43].

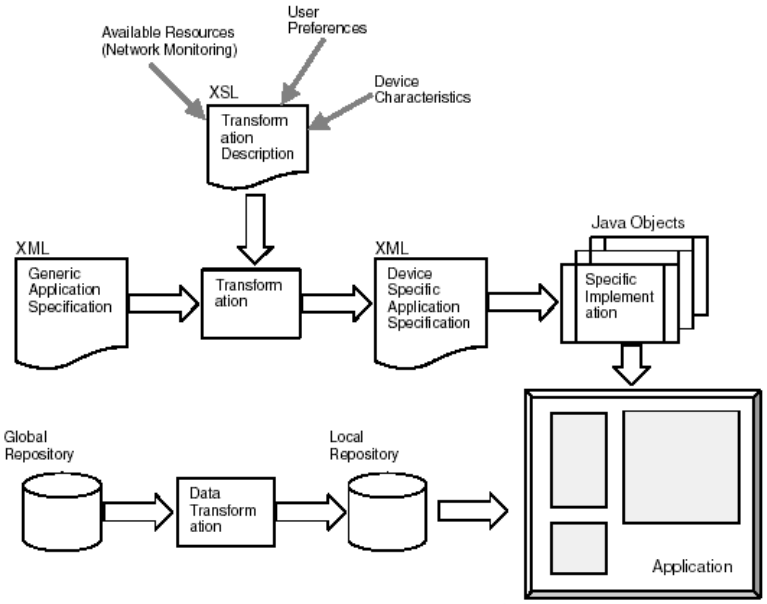


Figura 3 - Processo de mapeamento das aplicações (extraído de [27])

2.2.2. "ICrafter: A Framework for Ubiquitous Computing Environments"

Em [21], é apresentado o ICrafter, um *framework* para o desenvolvimento de serviços e de suas interfaces com usuário para uma categoria de ambientes ubíquos, os espaços interativos (i.e., *interative workspaces*). Um espaço interativo é um espaço físico rico em tecnologia. Ele pode conter computadores interconectados (e.g., *desktops*, *laptops*, *handhelds*.), dispositivos de uso geral (e.g., *scanners*, impressoras) e dispositivos de I/O (e.g., projetores, microfones, alto-falantes). Nesses espaços interativos, os usuários, utilizando os seus dispositivos, podem interagir com o ambiente e com outros usuários para

compartilhar dados e colaborar na realização de tarefas (e.g., apresentações, revisões de projeto). Os espaços de trabalho interativos podem ser salas de reuniões, de conferências ou bibliotecas.

Baseado nas características dos espaços interativos citadas anteriormente, a arquitetura do ICrafter propõe a centralização das aplicações em um servidor que possui informações sobre o contexto do ambiente. Na estrutura do ICrafter, os clientes devem requisitar um serviço através de um pedido ao servidor de UIs (i.e., interfaces de usuários) específicas para o dispositivo utilizado. O pedido é recebido por uma entidade chamada de *Interface Manager* (IM). O IM, ao receber um pedido para uma UI de um serviço, seleciona primeiramente um gerador baseado no dispositivo, que realizou o pedido, e no serviço para qual a UI foi pedida. Em seguida, o IM executa o gerador que cria uma descrição da aplicação para ser enviada ao dispositivo para que ele monte a interface (e.g., HTML para um *notebook* com navegador).

Para criar as UIs, os geradores necessitam de acesso à informação dos serviços, do dispositivo, e do contexto do espaço interativo. Essas informações estão armazenadas no servidor, num espaço chamado de *Context Memory*, sendo esse, compartilhado por todos os dispositivos que fazem parte do ambiente.

Os dispositivos ubíquos, ao receberem a descrição das UIs do serviço, devem ser capazes de transformá-las em interfaces permitidas pelo dispositivo. No ICrafter, foram implementadas: a geração de VoiceXML, HTML e SUIML (*Swing UI Markup Language*). O SUIML é uma linguagem também proposta no ICrafter para ser utilizada em dispositivos que suportam Java Swing (i.e., uma biblioteca gráfica do Java) através da instalação de um *middleware* chamado de SUIML *Interpreter*. Esse *middleware* transforma uma interface descrita em SUIML em componentes gráficos da API do Java.

O ICrafter é semelhante, em termos de concepção, ao trabalho descrito na Seção 2.2.1, com a diferença de exigir no dispositivo que requisita um serviço a existência de uma aplicação capaz de transformar os UIs em uma interface exibível no dispositivo. Desta forma, também provê **adaptação da interface das aplicações, em tempo de execução, adaptação à descrição e às classes de dispositivos.**

2.2.3. *"Using XML to semi-automatically derive user interfaces"*

Em [16], é apresentada uma abordagem para geração semi-automática de interfaces de aplicações a partir de XML. O artigo apresenta uma linguagem própria para descrição da interface, baseada em XML, denominada de UID (*User Interface Description*) que permite a descrição da interface da aplicação através dos UIOs (*User Interface Objects*) que representam componentes de entrada e saída de dados. O engenheiro de software deve especificar um mapeamento de cada UIO, descrito na interface da aplicação, para um componente de uma linguagem de programação (e.g., Java) ou uma linguagem de marcação (e.g.,VoiceXML) através de um documento chamado UIM (*User Interface Mapping*).

No trabalho [16], ao contrário dos trabalhos [27] e [21], não foram especificadas regras para mapear a linguagem UID para uma linguagem de marcação ou de programação. Essa tarefa deveria ser realizada pelo engenheiro de software e esse poderia também utilizar a linguagem DEVDEF proposta para especificar as características do dispositivo alvo da aplicação. Dessa forma, usando a definição inicial e as regras escritas pelo engenheiro de software, o processo de mapeamento seria realizado.

Um protótipo inicial de uma ferramenta que realiza o mapeamento para WML foi desenvolvido. Contudo, a ferramenta não realizava o processo completo de geração do código WML, sendo a sua finalização apresentada como um trabalho futuro. A ferramenta gera em tempo de construção o WML a partir dos UIDs descritos no XML. Assim, a ferramenta provê **adaptação, em tempo de construção, à descrição da interface das aplicações.**

É importante mencionar que na conclusão do trabalho foi destacado que padrões como *Composite Capability/Preference Profiles* (CC/PP) [47] e *Universal Plug and Play* (UPnP) [29] podem ser mais adequados para a descrição do dispositivo do que o DEVDEF proposto pelo trabalho.

2.2.4. *"XML-based unified user interface system under J2EE architecture"*

Em [30], é apresentada uma arquitetura para o desenvolvimento de aplicações para internet. A arquitetura é projetada para prover diferentes visualizações de dados e serviços contidos em um servidor Web, permitindo o acesso a esses serviços por diferentes clientes

(e.g., PCs, PDAs). A arquitetura, intitulada de *XML-Based Unified User Interface System*, é dividida em três camadas: a cliente, que possui um navegador com suporte a linguagens de marcação WEB (e.g., XHTML); a *middlelayer*, que contém a lógica de programação; e a camada de dados na qual estão as bases de dados legadas ou os bancos de dados. A arquitetura é proposta para ser desenvolvida utilizando J2EE [51], tecnologia Java para servidores.

As aplicações devem ser desenvolvidas utilizando J2EE e o acesso aos dados deve ser realizado com XML. Documentos XML são utilizados na comunicação com qualquer base de dados, inclusive as legadas. Para realizar essa operação, é sugerida a criação do *XML DB Converter*, uma camada da arquitetura responsável por mapear os dados contidos nos documentos XML, provenientes da aplicação, em um formato conhecido pelos banco de dados (e.g., inserções em SQL). O mapeamento foi implementado de XML para tabelas cujos relacionamentos eram um para um.

A arquitetura sugere que a interface das aplicações seja descrita também em XML e que conversores devem ser escritos para mapear essa descrição em uma linguagem de marcação aceita pelos navegadores dos computadores ou dos PDAs (e.g., XHTML, WML), assim, a arquitetura permite **adaptação, em tempo de execução, à descrição e a classes de dispositivos.**

Na arquitetura são apresentados como trabalhos futuros: o desenvolvimento de uma linguagem para descrever as aplicações e a construção de conversores para linguagens de marcação (e.g., HTML, WML).

2.2.5. "MAB - Mobile Application Builder"

Em [17], é apresentada uma solução para o problema de descrição de aplicações independentes de dispositivo. O trabalho propõe que a descrição da aplicação seja realizada em XML e apresenta a possibilidade de gerar código nativo ou intermediário para um dispositivo alvo, assim como, a possibilidade de acréscimo de códigos na linguagem de programação (e.g., J2ME MIDP).

No trabalho, foi apresentada uma ferramenta, intitulada MAB (*Mobile Application Builder*), que possibilita a geração de código J2ME MIDP para executar em PalmOS utilizando KVM/CLDC [43], assim como a geração de código WML para executar em

celulares através do uso de WAP [114]. A ferramenta possibilita a criação de aplicações cliente/servidor que utilizam três componentes de interface: listas, texto e formulários. A interface da aplicação é construída através de um módulo gráfico que possibilita a montagem dos componentes que compõem a interface, bem como a relação entre eles. A ferramenta gera uma descrição da aplicação em XML que é depois convertida no código alvo. Assim, a ferramenta provê **adaptação, em tempo de construção, à descrição da interface das aplicações.**

A incorporação e a importação de dados são feitas através do uso de XML, mas o acesso a dados no dispositivo não é possibilitado pela ferramenta, sendo esse, um papel adicional do programador que deve utilizar o recurso de inserção de códigos externos através da invocação de métodos de uma classe abstrata. A ferramenta é construída em Delphi e a tecnologia utilizada na aplicação servidor é Java *Servlets* [51] rodando em um servidor TomCat. Uma aplicação teste de m-Commerce é criada para validar o uso da ferramenta.

2.2.6. "UIML – User Interface Markup Language"

A UIML [52] é uma linguagem declarativa baseada em XML para especificação de interfaces de usuário independente de dispositivo. A UIML permite a descrição da interface em cinco partes: *description*, *structure*, *data*, *style* e *events*. Na Figura 4, é apresentada uma aplicação que segue essa estrutura do UIML.

Em <*description*>, declara-se os elementos que compõem a interface e a classe a qual pertence cada elemento. Em <*structure*>, descreve-se como os componentes se agrupam para definir a interface (e.g. um botão dentro de um painel dentro de um formulário). Em <*data*>, é descrita a informação que deve ser exibida pela interface para cada componente (e.g. "OK" para um botão). Em <*style*>, é definido o mapeamento de cada classe para um componente de uma linguagem real e em <*events*> é permitido associar eventos em linguagens reais (e.g. Java) aos componentes descritos.

A plataforma proprietária LiquidUI [113] fornece uma aplicação servidor para realizar mapeamentos de aplicações escritas em UIML para WML, VoiceXML e HTML dependendo da requisição realizada por um navegador via HTTP. Essa ferramenta provê, portanto, **adaptação, em tempo de execução, à descrição da interface das aplicações.**

O UIML apresenta como desvantagens: a permissão para que quaisquer componentes de interface possam ser definidos, a exigência de que cada aplicação possua uma folha de estilo para mapeamento, a definição de eventos dependentes de uma implementação real e a ausência de grande quantidade de documentação.

```
<?xml version="1.0" standalone="no"?>
  <uiml version="2.0">
   <interface name="Apliteste" class="MyApps">
     <description>...</description>
     <structure>...</structure>
     <data>...</data>
     <style>...</style>
    <events>...</events>
   </interface>
  </uiml>
```

Figura 4 - Estrutura de uma aplicação UIML (extraído de [14])

2.2.7. Resumo comparativo

As soluções de adaptação à descrição expostas anteriormente têm em comum o uso de uma meta linguagem para descrever as interfaces das aplicações em XML. Essa meta linguagem em um determinado momento é convertida em código de uma linguagem de programação ou de uma linguagem de marcação. Essa conversão pode ocorrer tanto em tempo de construção quanto em tempo de execução.

A ferramenta User Interface Generator (UIG) proposta nesta dissertação e detalhado no Capítulo 5 utiliza a mesma abordagem de descrição das aplicações em XML. E realiza, em tempo de construção, a conversão desse XML para um código executável nas plataformas de programação para DMs.

Na Tabela 1, é apresentada um resumo das principais características das soluções descritas e da ferramenta UIG possibilitando uma melhor visualização das características comuns e das diferenças existentes entre as soluções. As características apresentadas são as seguintes:

I - Tipos de Aplicações alvo;

II - Plataformas de programação das aplicações geradas;

III - Componentes gráficos das interfaces das aplicações;

IV - Definição de estilos para as interfaces;

V - Geração de comportamento esperado para as interfaces;

VI - Exigência de *middlewares* para execução das aplicações;

VII - Existência de uma ferramenta gráfica para auxiliar a edição;

VIII - Integração com arquiteturas de adaptação de conteúdo;

IX - Estratégia de adaptação.

Para facilitar a construção da tabela foram criados acrônimos para dois trabalhos: AAUC referente ao trabalho "*Adaptive Applications for Ubiquitous Collaboration*" e UID referente ao trabalho "*Using XML to semi-automatically derive user interfaces*". Os demais acrônimos já são adotados nos próprios trabalhos expostos anteriormente.

	AAUC	ICrafter	UID	MAB	UIG
I	Aplicações para ambientes colaborativos	Aplicações para espaços interativos	Aplicações com formulários	Aplicações com formulários	Aplicações com formulários
II	PockectSpace e J2SE com 3D	VoiceXML, HTML e SUIML	WML	WML e J2ME MIDP 1.0	Superwaba, J2ME MIDP 1.0 e 2.0
III	*interactors*	UIs criados pelo engenheiro de software para cada serviço do espaço interativo	O engenheiro cria os componentes UIO de entrada de dados	Listas, Textos, Formulários com campos de entrada de dados e datas	10 componentes do XForms e o *image* do XHtml
IV	-	-	-	-	Uso de CSS (i.e., *Cascading Style Sheets*) para definição de cores e atributos para definir alinhamento
V	Uma ação em um *interactor* reflete em uma mudança no modelo compartilhado	Uma ação em um UI invoca um serviço no servidor	-	Passagem de parâmetros entre formulários	Uso de XML Schema para definição da validação da entrada de dados
VI	A aplicação no PDA exige o PockectSpace	SUIML Interpreter para geração em SUIML	-	-	-
VII	-	-	-	Ferramenta para definições dos formulários e da interação entre eles	*Plugin* para o Eclipse que facilita a configuração da

					geração de código
VIII	A arquitetura adapta o modelo em relação à visualização	-	-	-	O código gerado é facilmente integrável ao MobAC
IX	XSLT pré-estabelecidos para a transformação do XML	O engenheiro cria *templates* para mapear o resultado dos serviços nos UIs	O engenheiro escreve um UIM com as regras de mapeamento	A ferramenta contém os mapeamentos definidos	XSLT pré-estabelecidos para a transformação do XML

Tabela 1 – Resumo comparativo de soluções para descrição da interface com usuário

2.3. Arquiteturas de Adaptação de Conteúdo

Na Seção 2.2, foram apresentadas soluções para o problema de definição, independente de dispositivo, da interface de uma aplicação. Contudo, a adaptação de uma aplicação não está restrita a sua interface com usuário. Como mencionado no Capítulo 1, a integração dos dispositivos móveis com as redes de comunicação de dados permitiu aos sistemas corporativos e aos sistemas *Web,* que antes interagiam somente com computadores *desktops,* serem acessados também por dispositivos móveis [9]. Essa integração impôs a exigência de um outro tipo de adaptação para as aplicações: a adaptação de conteúdo, já que as informações dos sistemas corporativos e dos sistemas *Web*, em geral, não estão preparadas para serem exibidas por esses dispositivos [12].

Nesses casos em que um sistema é acessado por diferentes dispositivos e por computadores *desktops*, o conteúdo deve ser cuidadosamente selecionado e adaptado às condições restritivas do equipamento de destino (e.g. dimensões do *display*, quantidade de cores, memória disponível). Obviamente, que o mesmo conteúdo direcionado para um computador fixo (i.e. *desktop*) não deve ser enviado a um celular, por exemplo. Nesse caso, tanto deve haver redução das informações, como modificação da forma de apresentação dos dados.

Contudo, a adaptação de conteúdo não, necessariamente, baseia-se somente nas características dos dispositivos que acessam as informações. As preferências e o contexto do usuário que utiliza a aplicação também podem ser relevantes nessa adaptação. Informações sobre a localização, o perfil e os parâmetros de qualidade de serviço do usuário são importantes para a decisão de filtragem de informação [9]. Essa adaptação é

chamada em [12], de adaptação semântica. Essa pode considerar inclusive a língua materna do usuário para converter as informações.

As estratégias para adaptação de conteúdo podem ser divididas em:

- **Adaptação de conteúdo baseada em agentes móveis**. Nesse tipo de adaptação de conteúdo, a aplicação no DM contém um agente móvel capaz de coletar dados sobre o dispositivo e usuário e migrar para uma rede (e.g a Internet) com intuito de buscar informações nos servidores para a aplicação. Dentre os trabalhos que descrevem soluções para esse tipo de adaptação podemos citar o PUMAS [38] , o KODAMA [35] e o CONSORT [39] .

- **Adaptação de conteúdo baseada em *proxys* ou *gateways***. Nesse tipo de adaptação de conteúdo, imagens, vídeos, áudio e páginas *Web*, são requisitadas pelas aplicações através de *gateways* ou *proxies* que são responsáveis por adaptar as informações utilizando as características do dispositivo ou do usuário que as acessa.

Nas subseções seguintes são descritos três trabalhos que utilizam a estratégia de adaptação de conteúdo através de *proxys* e *gateways.* Esses trabalhos são fortemente relacionados à solução de adaptação de conteúdo do ambiente proposto nesta dissertação.

2.3.1. "Adaptation d'une application multimédia par un code Mobile"

No trabalho [10], são descritas as vantagens de utilizar adaptação de conteúdo através de *proxys*. Dentre as vantagens, são citadas as seguintes: a adequação das informações ao dispositivo e a diminuição do tráfego de informação entre o *proxy* e o DM que, na maioria das vezes, utiliza uma conexão sem fio instável e de baixa largura de banda.

No trabalho, uma arquitetura para adaptação de vídeo é apresentada. Um *proxy* é usado para intermediar e adaptar um *stream* de vídeo de acordo com as dimensões da tela do dispositivo, consistindo, de acordo com as definições de adaptação descritas na Seção 2.1, em uma **adaptação de conteúdo, em tempo de execução, ao dispositivo**. O *proxy* utiliza HTTP para se comunicar com o servidor no qual se encontra o vídeo requisitado e utiliza RTP (i.e., *Real Time Protocol*) para entregar o vídeo ao dispositivo móvel. Durante a conversão, o *proxy* converte o vídeo de MPEG para o formato H.263 e reduz as imagens brutas que compõem o vídeo, assim como pode modificar o número de quadros por

segundo no vídeo adaptado. Entretanto, o trabalho não apresenta uma abordagem para coletar as informações sobre os dispositivos móveis e enviar essas características ao *proxy* para realizar a adaptação.

2.3.2. "Context-aware adaptation for mobile devices"

Em [12], é apresentada uma arquitetura de adaptação de documentos Web para dispositivos móveis, intitulada de NAC (*Negotiation and Adaptation Core*). Nessa arquitetura, a adaptação do conteúdo modifica o layout dos documentos baseado nas características do dispositivo cliente e das preferências do usuário. Para realizar a adaptação, um conjunto de regras de conversão é estabelecido na arquitetura. Essas regras são utilizadas por um *proxy* para realizar a transformação dos documentos.

Um repositório dos perfis dos usuários e dos dispositivos é criado para fornecer as características necessárias para adaptação. Esses perfis encontram-se descritos no padrão proposto pelo *World Wide Web Consortium* (W3C) [36] para a descrição de perfis conhecido como *Composite Capability/Preference Profiles* (CC/PP) [46]. Os vocabulários para a construção dos documentos CC/PP utilizam a linguagem UPS (*Universal Profiling Schema*) proposta nesse mesmo trabalho.

O acesso do *proxy* aos perfis é realizado utilizando a tecnologia SOAP [55] para acessar *WebServices.* Assim, quando um dispositivo deseja acessar um documento *Web*, ele requisita ao *proxy*, que 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.

Os documentos *Web* podem ser adaptados para a língua de preferência do usuário. Neste caso, o documento original contido no servidor *Web* deve ser descrito em SMIL [10], uma tecnologia XML que permite a descrição de conteúdo com alternativas, por exemplo, textos em inglês e francês. Assim, o *proxy* retira o conteúdo descrito na língua de preferência do usuário e o devolve ao dispositivo. Nesse caso, uma **adaptação de conteúdo, em tempo de execução, ao contexto** (i.e., preferências do usuário e perfil do dispositivo)**.**

Outra forma de adaptação exposta refere-se à apresentação dos documentos que podem ser convertidos de XHTML para WML caso sejam acessados por celulares. Nesse caso, uma **adaptação de conteúdo, em tempo de execução, ao dispositivo.**

Uma importante desvantagem desta arquitetura é a necessidade de descrever cada novo dispositivo em UPS, que dificulta a escalabilidade dessa solução. Outro problema decorre do fato da adaptação, em alguns casos, necessitar de documentos alternativos para cada requisição. Na Figura 5, é ilustrada essa arquitetura.

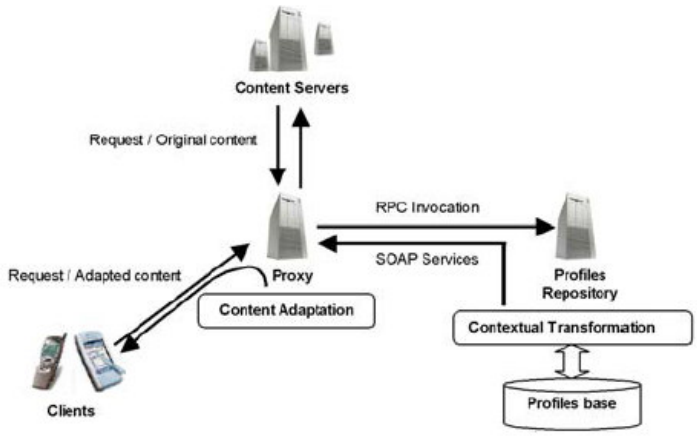


Figura 5 - Arquitetura NAC, retirado de [12]

2.3.3. "Adaptation in Mobile Computing"

Em [11], é apresentada uma arquitetura geral para adaptação de conteúdo multimídia Web utilizando um módulo cliente para monitoramento e realização de tarefas de descompressão e um gerenciador de adaptação localizado em um servidor *proxy*. Esse gerenciador de adaptação repassa as requisições recebidas para os módulos responsáveis pelas adaptações específicas de um serviço.

No trabalho, módulos, parâmetros e estratégias para a realização de adaptação de imagem, vídeo e áudio são definidos. Na Figura 6, é ilustrada a arquitetura proposta em [11], na qual um módulo cliente de adaptação é capaz de monitorar e armazenar informações sobre as preferências do usuário e informações sobre o dispositivo. Essas informações são enviadas em cada requisição da aplicação ao gerenciador de adaptação localizado no servidor. Esse servidor, por sua vez, repassa as informações para o módulo de

adaptação responsável pelo serviço, por exemplo, no caso de requisição de um vídeo, ela é repassada ao módulo de adaptação de vídeo.

O trabalho apresenta a simulação da adaptação de *stream* de vídeo baseada nas mudanças da banda passante da simulação. O que consiste em uma **adaptação de conteúdo, em tempo de execução, ao contexto.** Contudo, as formas de adquirir a informação necessária para realizar a adaptação não são apresentadas.

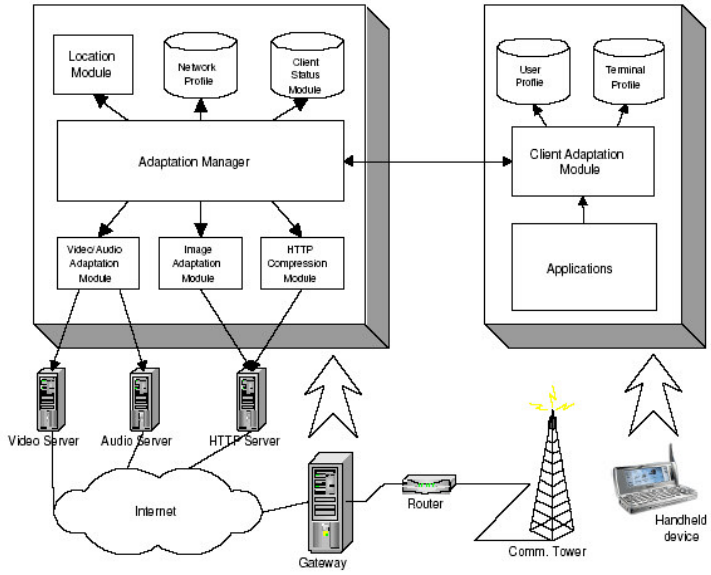


Figura 6 - Arquitetura proposta em [11]

2.3.4. Resumo Comparativo

Na Tabela 2, é apresentada um resumo das principais características das arquiteturas de adaptação de conteúdo descritas anteriormente. As arquiteturas são comparadas ao *framework* Mobile Adapter proposto nesta dissertação possibilitando uma melhor visualização das características comuns e das diferenças existentes entre as soluções. As características abordadas no resumo comparativo são:

I - Tipos de Aplicações alvo;

II - Processo de captura das características dos dispositivos;

III - Descrição dos perfis do dispositivo e do usuário;

IV - Integração com aplicações para dispositivos móveis;

V - Tipos de Conteúdo Adaptado;

VI - Estratégias de adaptação.

	Adaptation par un code mobile	NAC	Mobile Computing	Mobile Adapter
I	Aplicações multimídia Web	Aplicações multimídia para Web	Aplicações multimídia para Web	*Mobile Application Servers* (MAS)
II	-	Um monitor no dispositivo captura as características dinâmicas e os identificadores dos perfis do usuário e do dispositivo	Um monitor no dispositivo captura as características e envia a cada requisição	Um monitor no dispositivo captura as características dinâmicas, o identificador do usuário e a URI do documento UAProf que descreve o dispositivo
III	-	Utiliza CC/PP para descrição dos perfis usando os vocabulários UPS	-	Utiliza CC/PP para descrição dos perfis do usuário e UAProf para descrição das características estáticas do dispositivo
IV	-	Um simulador executa em Pocket PC	-	O MobAC é implementado em Superwaba, Personal Profile, J2ME MIDP 1.0 e 2.0
V	*Streams* de Vídeo	Documentos SMIL e HTML	Imagens, *Streams* de áudio e de vídeo.	Recursos estáticos (e.g. imagens, texto)
VI	O vídeo é adaptado pelo *proxy* para as dimensões do dispositivo	Extração de conteúdo e conversão de formatos de HTML para WML	*Streams* de Vídeo são adaptados à banda passante da simulação	As estratégias de adaptação são delegadas ao MAS

Tabela 2 – Resumo comparativo dos trabalhos de adaptação de conteúdo

2.4. Conclusão

Neste capítulo, foram apresentadas arquiteturas de adaptação de conteúdo e soluções para a construção de aplicações adaptáveis à descrição. No caso das arquiteturas de adaptação de conteúdo, os trabalhos não apresentam formas eficientes de captura, manipulação e gerenciamento dos perfis do dispositivo, do contexto e do usuário, o que dificulta a construção de sistemas reais que realizem adaptação de conteúdo.

Os trabalhos para a construção interfaces com usuário adaptáveis à descrição restringem os tipos de aplicações que podem ser construídas. E, em certos casos, exigem do engenheiro software a escrita das regras de mapeamento e dos componentes de interface com usuário. No caso do MAB, existem poucos componentes gráficos para a construção dos formulários.

Além disso, como mencionado no Capitulo 1, é necessária a criação de um ambiente para integrar e aprimorar essas soluções existentes para os desafios do desenvolvimento de aplicações para DMs. A abordagem de construção de interfaces adaptáveis proposta por Allan [27] e a arquitetura de adaptação de conteúdo NAC [12] foram utilizadas como base para atingir este objetivo e construir o ambiente proposto nesta dissertação.

3. Plataformas de programação para DMs e FrameWorks

Neste capítulo, é apresentada uma introdução sobre *frameworks* e sobre as plataformas de programação investigadas para a construção e a implementação dos *frameworks* XFormUI, MobAC e Requisitor propostos nesta dissertação.

Na Seção 3.1, são descritas as características das plataformas Superwaba, J2ME MIDP 1.0/CLDC 1.0 e MIDP 2.0/CLDC 1.1. Na Seção 3.2, são apresentadas as definições de *framework* , a sua correlação com padrões de software e como essas tecnologias podem ser utilizadas para facilitar a construção de aplicações adaptativas e multi-plataformas.

3.1. Plataformas de Programação

No Capítulo 1, o desenvolvimento de aplicações multi-plataformas foi descrito como um dos desafios no desenvolvimento de aplicações para DMs. Para construir aplicações para esses dispositivos, um engenheiro de software pode utilizar uma diversidade de linguagens e plataformas de programação, como: Brew [59], J2ME [43], C++ [56], Superwaba [44], Mophun [57] e CodeWarrior C++ [58]. Essas aplicações podem também ser criadas utilizando linguagens de marcação, como HTML, XHTML [36] e WML quando os dispositivos móveis possuem navegadores que suportam essas tecnologias.

O desafio no desenvolvimento de aplicações multi-plataformas decorre do suporte heterogêneo a essas plataformas de programação e a essas linguagens de marcação. Em sua maioria, as plataformas para DMs executam apenas em uma classe de dispositivos (e.g., Sattellite Forms [86] para *handhelds*), ou em um determinado sistema operacional (e.g., CodeWarrior C++ para Palm OS [48]). Esse mesmo problema acontece com as linguagens de marcação, pois apenas algumas classes de dispositivos suportam WML (e.g., celulares), HTML (e.g., Pockect PCs) e XHTML (e.g., *smartphones*). Portanto, essa heterogeneidade inviabiliza a escolha de uma única linguagem para o desenvolvimento de uma aplicação que tenha como requisito executar em uma grande quantidade de diferentes dispositivos.

Assim, em muitos casos, a única solução para o engenheiro de software é reescrever todo o código da aplicação para que essa execute em uma outra plataforma de programação. A reescrita do código pode ocorrer mesmo nos casos em que o engenheiro escolhe plataformas baseadas na mesma linguagem de programação. Isto acontece com Superwaba [44] e J2ME [43]. Essas plataformas são baseadas na linguagem de programação Java, mas possuem *APIs* (*Application Programming Interface*) diferentes para interface gráfica, armazenamento e comunicação de dados. Até mesmo o suporte aos tipos básicos de dados é diferente nessas plataformas.

Portanto, existe uma necessidade de construir soluções que permitam ao engenheiro de software desenvolver, com uma menor quantidade de reescrita de código, aplicações que executem em muitos dispositivos. Com objetivo de criar um ambiente que atenda a essa necessidade, as plataformas Superwaba, J2ME MIDP 1.0/CLDC 1.0 e J2ME MIDP 2.0/CLDC 1.1 foram investigadas.

Essas plataformas foram escolhidas por apresentarem alguns conceitos de adaptação à descrição, porque em conjunto são suportadas por uma grande quantidades de DMs e o processo de desenvolvimento pode ser realizado, em sua maioria, utilizando ferramentas livres e de código aberto. Além disso, essas plataformas são baseadas em Java, uma linguagem orientada a objeto, com curva de aprendizagem relativamente baixa em relação a linguagens como C++ e já amplamente adotada [20] [75].

Nas subseções seguintes, são descritos detalhes de cada uma dessas plataformas e ao final, um resumo comparativo entre elas é apresentado.

3.1.1. J2ME

Java 2 Mobile Edition (J2ME) [43] é uma coleção de tecnologias e especificações que foram projetadas pela Sun Microsystems[tm] para o desenvolvimento de aplicações para dispositivos de pequeno porte, especialmente sem fio. J2ME utiliza a linguagem de alto nível Java que foi projetada para executar independentemente de um sistema operacional específico e é sintaticamente semelhante ao C++. As especificações do J2ME são divididas em configurações (i.e., *Configurations*) e perfis (i.e., *Profiles*). O conjunto perfil-configuração determina quais as funcionalidades disponíveis para uma aplicação e a forma

com que ela deve se estruturar. Na Figura 7, são ilustradas as especificações e as configurações do J2ME e a sua relação com as outras tecnologias Java: J2SE e J2EE.

As configurações são especificações que detalham uma máquina virtual Java e um conjunto básico de APIs que podem ser usadas por um certo conjunto de dispositivos. Para dispositivos de maior poder de processamento existe a especificação CDC (i.e., *Connected Device Configuration*), voltada para dispositivos com mais de 512 KB de memória, conexão permanente e taxa de transmissão relativamente alta. Na Figura 7, pode-se ver que o CDC é uma especificação cuja implementação se faz sobre a máquina virtual Java convencional, que naturalmente demanda maior capacidade de processamento e memória.

Para dispositivos de pequeno porte, tais como PDAs e celulares, com menos de 512 KB de memória disponível para as aplicações e sem conexão permanente, usa-se a especificação CLDC (i.e., *Connected, Limited Device Configuration*). A máquina virtual Java para essa especificação é conhecida como KVM (*Kilo Virtual Machine*) e seu conjunto de APIs é o núcleo de funcionalidades básicas de uma aplicação (e.g tipos de dados, conexão). O CLDC possui duas versões: a 1.0, que não contém tipos de dados ponto flutuante e a versão 1.1, que incorpora esses tipos de dados e outras funções matemáticas.

Os perfis (i.e., *Profiles*) são as especificações que criam um ambiente completo para desenvolvimento das aplicações. Eles contêm bibliotecas de classes adaptadas a uma determinada família de dispositivos. Nessas bibliotecas estão os componentes de interface visual e acesso a armazenamento de dados.

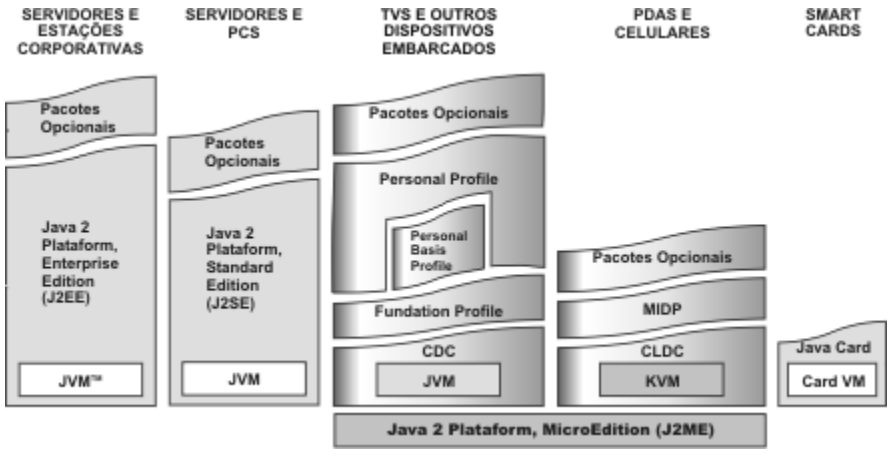


Figura 7 - Visão Geral das tecnologias Java da Sun [62]

O MIDP é construído sobre a especificação CLDC, como pode ser visto na Figura 7. A unidade básica de um programa MIDP é um *Midlet.* Um *Midlet* executa no dispositivo em um ambiente de execução chamado de *MidletSuite* que limita as permissões de acesso do *Midlet* ao dispositivo, semelhante ao mecanismo de *sandbox* dos *Applets* [62]. O MIDP adiciona APIs com diversas funcionalidades, tais como: armazenamento de dados permanentes, comunicação e interface com o usuário.

O MIDP foi projetado inicialmente para executar em celulares e *pagers.* Entretanto, já existem implementações para PDAs como a máquina virtual MIDP/CLDC para PalmOS da IBM [63]. O MIDP possui duas versões: a 1.0, que possui muitas limitações dos componentes visuais e a 2.0, com maior quantidade de componentes e que executa sobre o CLDC 1.1. As seguintes APIs estão presentes no MIDP:

- javax.microedition.lcdui, que apresenta um conjunto de classes para implementação de interface com o usuário;
- javax.microedition.rms, que fornece um mecanismo para armazenamento de dados permanentes;
- javax.microedition.io, que fornece suporte básico para rede e
- javax.microedition.midlet, que faz a interface entre a aplicação e o ambiente em que ele está executando.

As aplicações J2ME MIDP/CLDC são extremamente limitadas com relação a gerenciamento de eventos disponibilizados para programador. Apenas os eventos de ações em *Commands*, semelhantes a botões, e seleção em um *List,* são disponibilizados. A construção de interface nesta plataforma é bem distinta de outras plataformas de programação, pois não permite ao programador estabelecer a posição exata no *display* do dispositivo na qual um componente de interface irá se posicionar.

O posicionamento de componentes em J2ME MIDP foi assim projetado para possibilitar adaptação ao dispositivo da interface da aplicação. Na Figura 8, é ilustrado esse processo de adaptação provido pela plataforma, na qual uma mesma aplicação se comporta da forma mais adequada em cada tipo de dispositivo. Por exemplo, o componente que lista opções é apresentado no Palm OS como um *ComboBox*, no qual um *click* dispara a abertura de uma pequena janela com as opções. Já nos outros dispositivos as opções já são

apresentadas na tela. Além disso, a forma como os *commands* são exibidos na tela modifica-se dependendo da máquina virtual do dispositivo. No caso do Palm OS, os *commands* são apresentados diretamente no *display* do dispositivo, já em outros, são acessáveis através de um Menu (*options* em certos celulares).

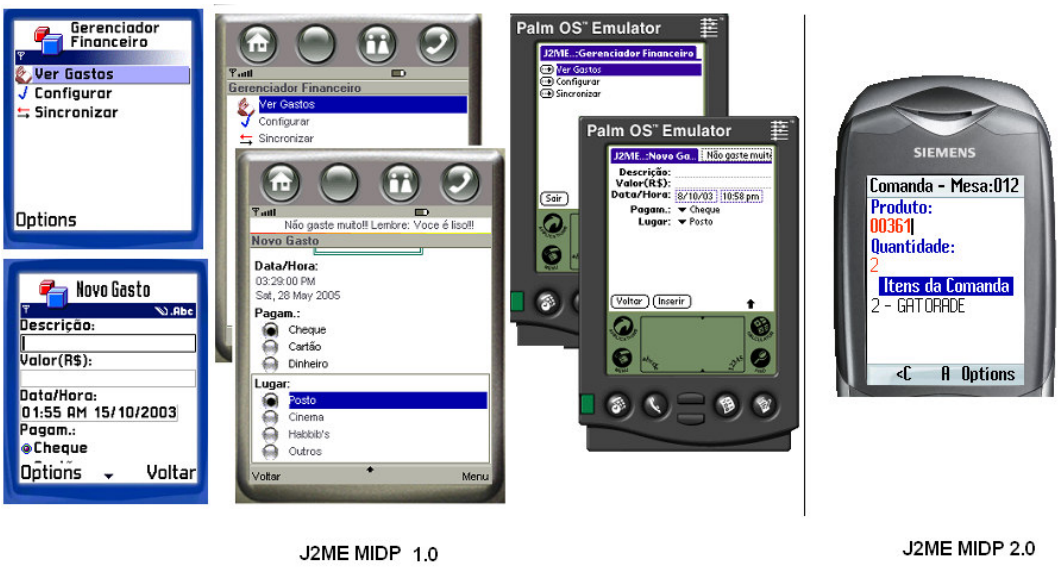


Figura 8 - Adaptação ao dispositivo oferecida pela plataforma J2ME MIDP

No MIDP 2.0, novos componentes são disponibilizados e existe a possibilidade de determinar alinhamento aos componentes (e.g. centro, esquerdo, direita). Outra vantagem do MIDP 2.0 é permitir a construção de componentes costumizáveis através da classe *CustomItem*. Esse componente foi utilizado para a elaboração do *ColorLabel* que será apresentado no *framework* XFormUI e que possibilita atribuir cores aos *labels* dos componentes. Na Figura 8, é ilustrada uma aplicação MIDP 2.0 utilizando o *ColorLabel* e as funções de alinhamento.

Para começar a desenvolver aplicativos em J2ME/MIDP é necessária a instalação das seguintes ferramentas:

- Editor de texto: necessário para editar o código fonte do aplicativo;
- Java 2 Standard Edition (J2SE) SDK: utilizado para compilar, pré-verificar e empacotar os códigos escritos para J2ME .
- J2ME Wireless Toolkit (WTK): um conjunto de ferramentas que oferece ao engenheiro de software um ambiente de gerenciamento dos projetos, compilação,

emulação, documentação e exemplos necessários para o desenvolvimento de aplicações usando a especificação CLDC/MIDP [37].

- Emuladores de dispositivos reais: os fabricantes de dispositivos móveis disponibilizam SDKs com emuladores de modelos comerciais que podem ser integráveis ao WTK.

Existem versões distintas do *Wireless Toolkit* e dos SDKs dos fabricantes para MIDP 1.0 e MIDP 2.0. Uma aplicação MIDP 1.0 pode executar em um dispositivo MIDP 2.0, contudo o contrário não é possível. E na maioria dos dispositivos, não existe a possibilidade de realizar atualização das máquinas virtuais para suportar a nova versão do MIDP.

Existem ambientes de desenvolvimento completos, os IDEs, com os quais o engenheiro de software pode escrever, testar e depurar aplicações em um único ambiente. Um bom exemplo de um IDE completo é Sun Java[tm] Studio Mobility [79], da própria Sun. Outra IDE bastante utilizada é o ambiente Eclipse [37] que através do uso do *plugin* eclipseme [78] oferece todo o processo de desenvolvimento: edição, compilação, depuração, pré-verificação, empacotamento e emulação no WTK e nos SDKs dos fabricantes.

Neste trabalho, iremos considerar o J2ME MIDP 2.0/CLDC 1.1 e J2ME MIDP 1.0/CLDC 1.0 como plataformas distintas, já que o MIDP 2.0 apresenta diferenças significativas, como mencionadas anteriormente, em relação ao MIDP 1.0 e exige uma nova máquina virtual e uma ferramenta distinta para compilação.

3.1.2. Superwaba

O Superwaba [44] é uma plataforma de software livre para desenvolvimento de software para PDAs. O Superwaba foi criado no início de 2000, por Guilherme Campos Hazan, bacharel em Ciência da Computação, pela Universidade Federal do Rio de Janeiro e é uma evolução de um outro projeto de software livre chamado Waba. A plataforma é disponibilizada sob duas versões: GPL (gratuita) e LGPL (subscrição anual), que permite tanto o desenvolvimento de aplicações livres quanto comerciais, e ao mesmo tempo impõe que a plataforma continue como um software de código aberto.

O Superwaba SDK define uma linguagem, uma máquina virtual, um formato de arquivo para armazenagem de classes e um conjunto de classes padrão. Um programa

Superwaba é multi-plataforma, pode executar, com as devidas restrições, tanto em PalmOS, Pocket PC e Symbian OS, além de poder executar como um *applet* em Linux e Windows.

Devido ao modo com que o Waba e o Superwaba foram projetados, os programadores podem usar as ferramentas Java, como compiladores e editores, para desenvolver programas em Superwaba. Entretanto, o Superwaba não implementa nenhuma parte das especificações do Java e também não tem nenhuma conexão com a detentora da marca, a Sun Microsystems [62].

As principais APIs disponíveis no Superwaba são:

- java.lang, que contém os tipos básicos de dados e a classe *Class*;
- waba.ui, que contém os principais componente para construção de interface gráfica;
- waba.fx, que engloba a classe *Graphics* para desenhos e a classe *Color* para mudança de cores;
- waba.io, para entrada/saída e classes de comunicação;
- waba.sys, com classes que oferecem uma interface com o sistema operacional;
- Superwaba.ext, com classes para acesso de periféricos.

Além disso, o Superwaba fornece um conjunto completo de APIs para desenvolvimento em PDAs, como classes para exibição de informações GPS (i.e. *Global Positioning System*), classes para acessar dados pessoais nas plataformas Pocket PC e Palm OS e classes que permitem armazenamento de dados em arquivos PDBs (i.e. *Palm Record Database*) também no Pocket PC, o que torna o PDB, específico do Palm OS, multi-plataforma. Nas versões mais recentes, foram disponibilizadas classes para realizar consultas SQL sobre PDBs, o pacote PDBDriver.

O Superwaba permite a criação de bibliotecas nativas em C e tem suporte à escala de cinza, cores e alta-resolução em todos os PDAs. O Superwaba utiliza toda a memória disponível para as aplicações, ao contrário do J2ME MIDP.

Ele permite ainda acessar banco de dados para PDAs como o IBM Db2EveryPlace [65] e o PDB SQL [44], além de suportar tipos de dados ponto flutuante, fontes customizáveis, detecção de colisão para jogos, janelas *popup* e arrastáveis, *Threads*, USB e IrDA.

O Superwaba ao contrário do J2ME MIDP, possibilita ao engenheiro de software determinar a posição dos componentes no *display* e colocar mais de um componente em uma mesma linha do dispositivo. Esta característica, apesar de possibilitar uma maior flexibilidade para a criação de interfaces, pode impedir que uma aplicação se adapte a diferentes dispositivos com resoluções distintas. Para evitar esse problema, o Superwaba permite posicionamento relativo dos componentes usando diretivas de posicionamento (e.g. AFTER, BEFORE, RIGHT).

A unidade básica de uma aplicação Superwaba é a classe *MainWindow*. Todo programa Superwaba inicia-se através da execução dessa classe. O Superwaba possibilita o gerenciamento de uma grande quantidade de eventos para interface: eventos de *click* com as canetas dos PDAs, eventos de *focus* e de saída dos componentes e eventos de seleção. Com Superwaba, pode-se construir janelas arrastáveis, modificar as propriedades de cores dos componentes, criar componentes customizáveis e adicionar menus às aplicações.

Para iniciar o desenvolvimento de aplicações móveis utilizando a plataforma Superwaba, é necessário utilizar os seguintes componentes:

- Editor de texto: para editar o código fonte do aplicativo;
- Java 2, Standard Edition (J2SE) Software Development Kit (SDK): para a compilação, empacotamento e execução no computador *desktop.*
- Superwaba SDK [11]: contém as classes que compõem a plataforma;
- Emuladores ou simuladores: Palm OS Emualtor (POSE), Palm OS Simulator, UIQ SDK para Symbian OS.

No desenvolvimento em Superwaba, a ferramenta Eclipse também pode ser utilizada para a compilação, a emulação e a geração de instaladores das aplicações.

3.1.3. Resumo comparativo entre as plataformas

Conforme descrito anteriormente, para construir os *frameworks* do ambiente proposto, as funcionalidades distintas e comuns das plataformas de programação para DMs foram investigadas. Na Tabela 3, é apresentado um resumo comparativo das características dessas plataformas que foram mais relevantes para a criação do *framework* XFormUI de interface gráfica e do *framework* Requisitor de comunicação de dados.

Critério	MIDP 1.0	MIDP 2.0	Superwaba
Executa em Palm OS	SIM	SIM	SIM
Executa em Windows CE	NÃO	NÃO	SIM
Executa em celulares	Na maioria dos modelos a partir de 2003	Na maioria dos modelos a partir de 2005	Só em alguns modelos Symbian OS
Interface Adaptável ao dispositivo	SIM	SIM	Depende do engenheiro de software
Custos	Máquinas virtuais são pagas pelos fabricantes	Máquinas virtuais são pagas pelos fabricantes	Depende da licença
Documentação disponível	Livros, *sites* e fóruns	Livros, *sites* e fóruns	Restrita ao *site* e aos fóruns
Acesso a arquivos do sistema	NÃO	Sim, usando a PIM API	SIM
Suporte a TCP/IP	Só HTTP	HTTP e Sockets	HTTP e Sockets
Suporte a IrDA	NÃO	Depende do dispositivo	SIM
Suporte a imagens	Formato PNG	Formato PNG, GIF e JPG	Formato BMP na versão GPL e PNG, GIF e JPG na versão LGPL
Controle do layout	Os componentes são adicionados um por linha	Controle de alinhamento e adição de componentes	Posicionamento relativo dos componentes
Suportes a cores nos componentes de interface	NÃO	Só nos componentes CustomItems	SIM
Suporte a ponto flutuante	NÃO	SIM	SIM
Quantidade de componentes	12 componentes	18 componentes e componentes *customizáveis*	Mais 100 componentes disponíveis na API e no *site*

Tabela 3 – Resumo comparativo entre as plataformas de programação

Como pode ser visto, cada uma delas atende a uma classe de dispositivos e apresentam funcionalidades distintas, principalmente em relação à composição das interfaces gráficas. Além disso, a estrutura do código da aplicação varia de J2ME MIDP para Superwaba, o que dificulta mais ainda a portabilidade de aplicações entre as plataformas.

3.2. Frameworks e Padrões de Projeto

O conceito de *framework* é apresentado na literatura de diferentes modos. Uma definição mais geral é apresentada em [66]: "um *framework* é um projeto abstrato para um tipo de aplicação". Segundo Viljamaa [44], *framework* é um conjunto de objetos reutilizáveis que engloba conhecimento de determinadas áreas e se aplica a um domínio específico. *Framework* é definido também como um projeto de reutilização que descreve como o sistema está decomposto em um conjunto de objetos que interagem entre si, podendo ser uma aplicação inteira ou apenas um subsistema [87].

Nessa dissertação, consideraremos um *framework* como um elemento para o reuso de componentes de arquiteturas orientadas a objetos. De um *framework* podem ser especializadas partes significantes de uma aplicação ou até mesmo, uma aplicação completa. Assim, um *framework* determina a arquitetura da aplicação e predefine parâmetros de projeto, permitindo ao engenheiro concentrar-se nos detalhes específicos da aplicação.

Vale ressaltar que um *framework* orientado a objeto é diferente de um pacote de classes ou de uma API, pois indica o modo como as classes devem ser instanciadas para colaborar na construção de uma aplicação. *Frameworks* têm como objetivo facilitar para o engenheiro de software a construção de aplicações com a adição de menor quantidade de código escrito possível, além de propiciar que boas soluções possam ser reutilizadas dentro de um mesmo domínio de aplicações.

Frameworks são construídos para serem flexíveis e permitir a adição de novas funcionalidades. Pontos de flexibilização, chamados de *Hot Spots,* são disponibilizados para propiciar a incorporação dessas novas características.

Para utilizar *frameworks,* os engenheiros de softwares escrevem subclasses das classes contidas na estrutura (*frameworks* caixa branca) ou utilizam classes existentes (*frameworks* caixa preta). Com essas classes, criam objetos para colaborarem juntos na realização das funcionalidades.

De um modo geral, o processo de desenvolvimento de um *framework* é realizado em três grandes fases [20] que são descritas a seguir:

- análise de domínio – Nessa fase, é identificado e organizado o conhecimento sobre o domínio para o qual o *framework* é proposto. O objetivo principal dessa fase é capturar os requisitos do domínio, encontrar os pontos comuns para compor o *framework* e identificar os pontos de flexibilização (i.e., os *Hot spots*).
- projeto de *framework* – Nessa fase, concentram-se os esforços na criação das abstrações do *framework* e na identificação dos relacionamentos entre as classes que irão compô-lo;
- instanciação do *framework* – Nessa fase, o engenheiro de software constrói classes a partir das classes disponíveis (*framework* caixa branca) ou escolhe classes fornecidas na estrutura (*framework* caixa preta) para criar as aplicações.

Em geral, como *frameworks* focam na reutilização de boas soluções, outro conceito de engenharia de software, os padrões de software, são utilizados durante a fase de projeto de um *framework*. De acordo com Coplien [68], um padrão é um pedaço de literatura que descreve um problema de projeto e uma solução geral para o problema num contexto particular. Desta forma, um padrão é uma entidade que documenta uma solução, um problema recorrente e o contexto em que se deve aplicar tal solução.

Padrões de software são utilizados em diferentes estágios do desenvolvimento de um software. E podem ser classificados de acordo como a fase em que são aplicados (e.g. padrões de análise, padrões de projeto) [19]. Nesta dissertação, diversos padrões de projeto foram utilizados para a construção dos *frameworks* e da ferramenta de geração de código do ambiente proposto. Além de facilitar a construção dos *frameworks,* os padrões de projeto são também adequados para documentá-los, pois o uso de padrões permite o entendimento do *framework*, sem forçar o conhecimento pleno do seu código fonte [45].

Além de permitir o reuso de boas soluções, *frameworks* são utilizados também para abstrair para o engenheiro de software como APIs específicas de uma plataforma de programação são utilizadas para realizar tarefas comuns. Por exemplo, uma implementação de um *framework* caixa preta pode conter classes que usam APIs de uma plataforma de programação, porém, o engenheiro de software ao instanciar essas classes não sabe quais APIs são usadas e nem como elas são acessadas.

Assim, caso um *framework* seja implementado em diferentes plataformas de programação, os códigos das aplicações que utilizam o *framework* poderão executar nas

diferentes plataformas. Desta forma, *frameworks* podem ser construídos para abstrair funcionalidades como construção de interface gráfica, persistência, comunicação de dados e segurança.

Uma outra forma de realizar essa abstração seria o uso de *middlewares* que oferecessem às aplicações essas funcionalidades. Esses *middewares* seriam implementados nos diferentes dispositivos móveis e poderiam ser requisitados pelas aplicações [60]. Contudo, o uso de *middlewares* pode não ser possível devido à pequena quantidade de memória de alguns dispositivos, além do desperdício de espaço e do atraso no tempo de execução que os *middlewares* podem causar, como foi comprovado no trabalho [61]. Portanto, *frameworks* podem ser mais adequados para realizar a abstração das APIs, além de permitir a sua alteração e extensão por parte do engenheiro de software.

Dentro desse contexto, um *framework* de persistência de dados para DMs, o FramePersist, foi proposto em [1] com objetivo de prover a transparência do armazenamento de objetos para a aplicação e fornecer um mecanismo eficiente de busca de objetos. Na construção do FramePersist, formas de armazenamento de dados das plataformas J2ME Personal Profile, J2ME MIDP e em Superwaba foram estudadas e generalizadas no *framework* para permitir sua implementação nas diferentes plataformas.

Essa mesma abordagem de abstração das APIs é utilizada na construção do *framework* XFormUI e do *framework* Requisitor propostos nesta dissertação. As funcionalidades comuns de cada plataforma investigada são incorporadas aos *frameworks* e as diferenças nas instanciações das classes que compõem as APIs das plataformas são ocultadas das aplicações. Por exemplo, um engenheiro de software escreve um código que realiza uma conexão HTTP utilizando o *framework* Requisitor, sem saber que cada implementação desse *framework* instancia classes distintas das APIs das plataformas para realizar a conexão.

3.3. Conclusão

Neste Capítulo, foram apresentados os principais conceitos relacionados a *frameworks* e às plataformas de programação para DMs.

Como visto na seção 3.1, as plataformas J2ME MIDP 1.0/CLDC 1.0, MIDP 2.0/CLDC 1.1 e Superwaba apresentam diferenças quanto a forma de estruturar as

interfaces das aplicações e contêm APIs distintas para sua construção. Além disso, cada plataforma é mais adequada para um grupo de dispositivos. Desta forma, para que um engenheiro de software possa construir aplicações que executem em grande quantidade de dispositivos, boa parte do código dessas aplicações teria que ser reescrito.

Os *frameworks* XFormUI e Requisitor propostos nesta dissertação têm o objetivo de diminuir essa reescrita na construção de interfaces com usuário e na realização de comunicação de dados. O *framework* XFormUI será apresentado no próximo capítulo, assim como a ferramenta UIG que realiza a geração de código que estende esse *framework* a partir de documentos XML.

4. Framework XFormUI

Neste capítulo é apresentado o *framework* proposto nesta dissertação para a construção de interfaces gráficas multi-plataformas para DMs. Na seção 4.1, é apresentada a tecnologia XForms utilizada como fonte de nomes e de componentes para o *framework*. Na seção 4.2, são apresentados os requisitos identificados para a construção do XFormUI. Na seção 4.3, é apresentada uma visão geral do *framework* e como ele atende aos requisitos.

4.1. XForms

XForms [53][15] é um esforço do *World Wide Web Consortium* (W3C) [36] para substituir formulários HTML por um formato mais avançado que permita a definição de formulários *Web* multi-plataformas e independente de dispositivos. XForms é uma linguagem baseada em XML que permite a definição de formulários, eventos e dados iniciais dos formulários.

A estrutura de um documento XForms é baseada no modelo MVC (i.e., *Model/Viewer/Controller*) e é dividida em três partes: *Instance*, *Model* e *User Interface*. Essa estrutura é composta pelos dados iniciais, pelos componentes de interface e pelos eventos. Dessa forma, a aplicação XForms é construída de modo mais abstrato, com a separação de dados, lógica e apresentação. Essa abordagem é definida em [15] como a forma ideal para criação de aplicações independente de dispositivo.

A descrição dos componentes de interface e dos eventos do XForms foi projetada para ser independente de dispositivo. Os campos de entrada de dados, por exemplo, são descritos como "*inputs*" e não como campos de texto que ocupam determinadas posições na tela. Outro exemplo é o elemento que dispara um evento de ação. Em XForms, em vez de descrevê-lo como um botão que recebe um evento de *click,* esse elemento é descrito como um gatilho (i.e., *trigger*) que recebe um evento de ativação. Isto permite que dispositivos que não possuam interface visual possam executar também a aplicação XForms.

Outra grande vantagem de XForms é permitir a integração com outras tecnologias XML como CSS (*Cascading Style Sheets*) [36] e XML Schema [36]. Em [15], é apresentado um navegador chamado X-Smiles que suporta parte da linguagem XForms e sua integração com as tecnologias citadas anteriormente. O navegador executa apenas em PCs e foi programado utilizando a linguagem Java.

Na Figura 9, é apresentado o código de um pequeno formulário que contém dois campos de entradas de dados e dois gatilhos de ações. São utilizados os componentes de interface *input*, *secret* e *trigger* do XForms. Cada componente descrito possui um *label.* No caso do *input* e do *secret* usa-se XPath [36] para acessar os dados contidos na parte *Model* do documento. Na aplicação, o campo de entrada de dados tfLogin é descrito como um *input,* possui o *label* "Login" e a referência ao valor "/data/login" que nesse caso é "Windson". Na Figura 10, é ilustrada a visualização desse formulário, utilizando o navegador X-Smiles [15].

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<XHTML xmlns="http://www.w3.org/1999/xhtml"
   xmlns:xforms="http://www.w3.org/2002/xforms"
   xmlns:ev="http://www.w3.org/2001/xml-events"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <HEAD>
   <link href="css.css" rel="stylesheet" type="text/css"/>
      <title>Formulário</title>
      <xforms:model id="FormLogin">
         <xforms:instance xmlns="" id="instance1">
            <data>
                 <faixa>Exemplo XForms</faixa>
                 <login>Windson</login>
                 <senha>senha</senha>
              </data>
           </xforms:instance>
        </xforms:model>
    </HEAD>
        <BODY>
        <xforms:output ref="/data/faixa" id="faixa">
         <xforms:label/>
        </xforms:output>
         <xforms:input ref="/data/login" id="tfLogin">
         <xforms:label>Login:</xforms:label>
           </xforms:input>
        <xforms:secret ref="/data/senha"id="pwSenha">
            <xforms:label>Senha:</xforms:label>
          </xforms:secret>
        <xforms:trigger id="btOK">
            <xforms:label>OK</xforms:label>
          </xforms:trigger>
        <xforms:trigger id="btLimpar">
            <xforms:label>Limpar</xforms:label>
          </xforms:trigger>
     </BODY>
  </XHTML>
```

Figura 9 - Código em XForms de uma tela de login

O XForms é utilizado nessa dissertação para servir como fonte de nomes e de componentes para o *framework* XFormUI. Além disso, documentos XForms são utilizados pela ferramenta UIG, descrita no Capítulo 5, para a geração de código nas plataformas de programação. A escolha do XForms deve-se ao fato dessa tecnologia ter qualidades desejadas para uma linguagem de descrição de interface. Dentre essas qualidades, podemos enumerar: a descrição dos componentes e dos eventos da interface independente de dispositivo e a grande quantidade de detalhes que as aplicações podem conter. Além disso,

o *XForms* tem boa quantidade de documentação [54], se comparado aos outros trabalhos mencionados no Capítulo 2.

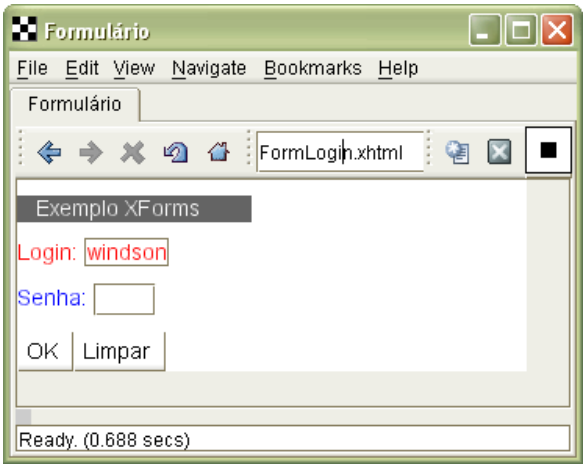


Figura 10 - Visualização da tela de *login* no X-Smiles [15]

4.2. Análise de domínio

Como mencionado no Capítulo 3, a primeira fase no processo de desenvolvimento de um *framework* consiste na análise do seu domínio. Desta forma, para a concepção do *framework* XFormUI, foi investigada a construção de interfaces gráficas nas plataformas de programação para DMs. Essa investigação ocorreu com o estudo apresentado no Capítulo 3 das plataformas de programação e com o desenvolvimento de software para DMs apresentado nos trabalhos [3], [4] e [1].

Nesta análise do domínio, foi identificada, uma lista, não exaustiva, de requisitos desejáveis para um *framework* de construção de interfaces gráficas para DMs. Esses requisitos são descritos a seguir:

1. **Disponibilidade de uma variedade de componentes de interface.** É importante para um engenheiro de software ter a sua disposição uma variedade de componentes de interface para compor os formulários, por exemplo, componentes de entrada de texto, números, datas e senhas; componentes para escolha de opções; gatilhos para disparar ações; componentes para exibição de textos e imagens; e componentes para exibição de tabelas de dados (i.e., os *grids*).

2. **Suporte a mecanismos de restrição e validação da entrada de dados em um formulário.** Em geral, os dados que são preenchidos por um usuário de uma aplicação têm

restrições estabelecidas pelas regras de negócio e devem ser validados durante o seu preenchimento. Assim, o *framework* deve conter mecanismos de restrição e validação desses dados preenchidos para auxiliar o engenheiro de software no desenvolvimento das aplicações.

3. Suporte a funções para troca de formulários e exibição de mensagens. O *framework* deve permitir a fácil mudança entre os formulários de uma aplicação para agilizar os processos de prototipação e desenvolvimento. Outra importante função é a exibição de mensagens que são usadas nas aplicações para informar erros de entrada de dados e exibir alertas com informações sobre o sistema.

4. Suporte ao Gerenciamento de Eventos. O gerenciamento de eventos é uma das partes mais importantes do desenvolvimento de uma aplicação, pois define a mudança de fluxo entre formulários e dispara as ações realizadas pela aplicação. Com o *framework,* o engenheiro de software deve ser capaz de gerenciar eventos que disparam ações (e.g., *clicks* em um botão), eventos de seleção de opções e eventos de mudança de estado dos componentes (e.g., entrada e saída, alteração do valor).

5. Execução adaptativa nos dispositivos. O *framework* deve possibilitar a construção de interfaces gráficas adaptativas para dispositivos móveis. As interfaces devem se adequar a diferentes dispositivos e tentar exibir sempre a melhor forma de apresentação. Por exemplo, a aplicação construída com o *framework* deve ser capaz de expandir sua área de visão e usar cores quando executada em um dispositivo com um *display* maior e colorido.

6. Flexibilidade na composição dos componentes dos formulários. É importante que o *framework* contenha funcionalidades que permitam ao engenheiro de software definir o *layout* dos componentes, como cores e alinhamento. Desta forma, ele pode construir interfaces mais amigáveis com usuário.

7. Extensibilidade para criação de novos componentes gráficos. O *framework* deve conter *Hot Spots* que possibilitem a fácil inserção de novos componentes para atender a novas funcionalidades desejadas pelo engenheiro de software.

8. **Facilidade no uso.** A nomenclatura dos componentes de interface do *framework* deve ser intuitiva para facilitar a instanciação dos componentes. Além disso, as funções de composição de formulários devem ser de fácil uso e o *framework* deve ser documentado pra facilitar a aprendizagem.

O *framework* XFormUI atende parcial ou totalmente cada um desse requisitos apresentados, como será descrito na próxima seção.

4.3. *Framework* XFormUI

Para atender os requisitos apresentados e facilitar o processo de geração de código da ferramenta UIG, o *framework* XFormUI foi projetado para conter componentes que tivessem o mesmo nome e comportamento dos componentes do XForms. Essa escolha deve-se ao fato de o XForms já atender os requisitos **1**, **2**, **4**, **6** e **8**, o que facilita a construção do *framework* XFormUI.

No Capítulo 3, foi explicado que o estudo e generalização das características comuns de um domínio deve ser utilizado na construção de um *framework*. Contudo, somente esta estratégia de generalização é insuficiente para que o XFormUI possa satisfazer todos os requisitos apresentados anteriormente. Assim, além da generalização, outras três estratégias foram utilizadas na construção do *framework* XFormUI: transparência, especificidade e combinação de componentes . Essas estratégias e a generalização são descritas, a seguir:

- **Generalização**. Para que o *framework* pudesse ser implementado em várias plataformas, as características comuns foram capturadas e generalizadas. A generalização acontece principalmente em relação à estrutura das classes principais das plataformas que são instanciadas no início da aplicação (e.g., *Midlet* em J2ME MIDP, *MainWindow* em Superwaba). Outro exemplo de generalização usado no *framework* é o gerenciamento de eventos que contém apenas os eventos comuns às plataformas investigadas.

- **Transparência**. Todos os métodos contidos nas classes do *framework* abstraem da aplicação o modo pelo qual são instanciadas as APIs de cada plataforma para criar os componentes, montar a interface, disparar eventos e exibir mensagens de alerta. Como mencionado no Capítulo 3, essa estratégia permite a construção de aplicações para executar em qualquer plataforma na qual o *framework* é implementado.

- **Especificidade**. Para que o *framework* pudesse atender aos requisitos ***5*** e **6,** funções que não correspondem a características comuns das plataformas foram adicionadas ao *framework.* Estas funções estão principalmente relacionadas ao gerenciamento de *layout* que pode não ser possível de ser executado em certas plataformas de programação como J2ME MIDP 1.0. No caso em que funcionalidades não são possíveis de serem

implementadas por completo em uma plataforma de programação, as implementações dos métodos correspondentes no *framework* podem não executar nada ou executar aquilo que se predispõe de forma parcial. Um exemplo dessa estratégia são os métodos para mudança de cores dos componentes que, na implementação em J2ME MIDP 1.0, não têm nenhum código para execução. Isto acontece devido a essa plataforma não suportar mudança de cores paras os componentes da API. A vantagem do uso dessa estratégia é que a aplicação é executada da melhor forma em cada uma das plataformas (e.g., se a plataforma suporta cores, os componentes da aplicação ficam coloridos).

- **Combinação de Componentes**. Nem todos os componentes existentes no XForms têm componentes correspondentes que executam o mesmo comportamento nas plataformas de programação. Nesses casos, componentes das plataformas são combinados para construir o componente do XFormUI.

Na Figura 11, é apresentada uma visão geral do XFormUI que é dividido em quatro partes: Formulários; Núcleo e Eventos; Componentes; e Validação e Restrição.

Três tipos de formulários podem ser construídos com o *framework* XFormUI: *TextArea*, *Select1Full* e *XForm.* O *TextArea* corresponde a um formulário com uma única caixa de texto que deve ocupar toda a tela do dispositivo. O *Select1Full* é um formulário com um menu para escolha de uma única opção. O *XForm*, por sua vez, corresponde ao formulário padrão ao qual podem-se adicionar vários componentes (e.g., campos de entrada de dados, componentes de escolha de opções).

A parte de Núcleo e Eventos contém as três principais classes do *framework*: *MainXFform, XFormItem* e *Trigger.* O *MainXForm* é a classe que define a estrutura principal da aplicação e é a primeira classe a ser executada no início de uma aplicação. Todos os componentes que podem ser adicionados a um formulário *XForm* herdam da classe *XFormItem*. O *Trigger* define um gatilho para disparar ações e pode ser adicionado aos três tipos de formulários descritos anteriormente.

A parte Validação e Restrição é composta pelas classes que permitem ao engenheiro de software definir restrições à entrada de dados nos componentes e testar se os dados não violam essas restrições. As classes *StringRestriction, StringValidator, BindRestriction, BindValidator*, *NumberRestriction* e *NumberValidator* compõem esta parte e são descritas em detalhes nas próximas sub-seções.

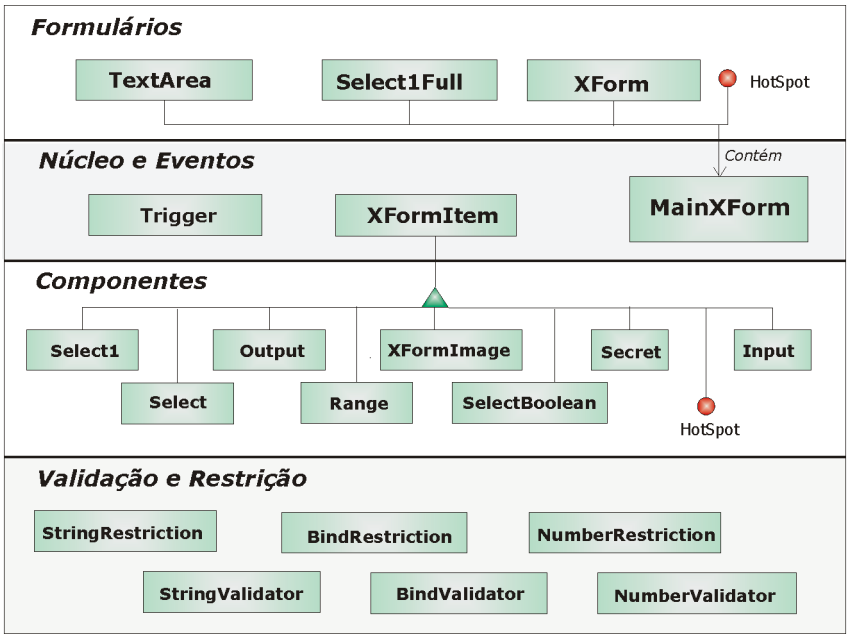


Figura 11 - *Framework* XFormUI

A parte Componentes é composta pelas oito classes disponibilizadas pelo framework para construção de formulários do tipo *XForm*. Essas classes em conjunto permitem a entrada de dados, como texto, datas e senhas; a exibição de textos e imagens, e a seleção de opções. Esses componentes, juntamente com a classe *Trigger*, atendem quase totalmente ao requisito **1** apresentado na seção 4.2, pois o *framework* XFormUI só não contém componentes para exibição de tabelas de dados.

Nas sub-seções seguintes são descritas em detalhes cada uma dessas classes e seus mapeamentos e implementações nas plataformas Superwaba, J2ME MIDP 1.0 e 2.0.

4.3.1. MainXForm

Como mencionado no Capítulo 2, a estrutura de uma aplicação em J2ME MIDP 1.0 e 2.0 é diferente da estrutura de uma aplicação em Superwaba. Em J2ME MIDP, a classe principal de uma aplicação herda da classe *Midlet* e deve reescrever os métodos *startApp()* e *destroyApp(boolean typeExit)* que correspondem, respectivamente, a inicialização da aplicação e as funções a serem executadas antes de sua saída. Já em Superwaba, a classe principal de uma aplicação herda da classe *MainWindow* e deve reescrever o método *onStart()* que é disparado no início da execução. Portanto, para criar uma estrutura que

possa ser implementada nessas plataformas, o *framework* XFormUI disponibiliza a classe MainXForm que generaliza os métodos dessas plataformas.

A classe MainXForm é uma classe abstrata que contém os métodos *onStart()* e *onExit(boolean typeExit)* que devem ser reescritos pelo engenheiro de software para definir o que ocorre ao iniciar aplicação e ao sair dela. A classe MainXForm disponibiliza, também, o método *setCurrentForm()* que permite a troca rápida entre formulários da aplicação e abstrai como ocorre a mudança de formulários nas plataformas.

A classe MainXForm implementa os métodos obrigatórios das estruturas de cada plataforma e mapeia suas chamadas para os métodos abstratos que devem ser escritos pelo engenheiro de software. Na Tabela 4, podem ser vistos os códigos de implementação do MainXForm nas plataformas. No caso do J2ME MIDP, os métodos *startApp()* e *destroyApp()* invocam, respectivamente, os métodos abstratos *onStart()* e *onExit()*. E no caso do Superwaba, apenas o método *onExit()* é mapeado, uma vez que a própria plataforma já exige o método *onStart()*. Na Tabela 4, pode ser visto também, como a classe MainXForm abstrai a forma de mudança de formulários.

Implementação em J2ME MIDP	Implementação em SuperWaba

`public abstract class MainXForm extends MIDlet{`
`protected void startApp(){`
`  onStart();`
`}`
`protected void destroyApp(boolean arg0) {`
`  onExit(arg0);`
`}`
`public void setCurrentForm(XForm xform){`
`  xform.addComponents();`
`  Display.getDisplay(this).setCurrent(xform);`
`}`
`public abstract void onStart();`
`public abstract void onExit(boolean arg);`
`}`
	`public abstract class MainXForm extends MainWindow{`
	`public MainXForm(){`
	`  super("",TAB_ONLY_BORDER);`
	`  this.setDoubleBuffer(true);`
	`}`
	`public void setCurrentForm(XForm xform){`
	`  swap(xform);`
	`}`
	`public void onExit(){`
	`  onExit(false);`
	`}`
	`public abstract void onExit(boolean arg);`
	`}}`

Tabela 4 - Códigos do MainXForm nas plataformas

4.3.2. Formulários

Como mencionado anteriormente, o framework XFormUI disponibiliza ao engenheiro de software três tipos de formulários: *Select1Full, XForm* e *TextArea*. Esses formulários têm, em comum, um método para exibição de mensagens de alerta, o *showAlert()*. Além disso, como mencionado anteriormente, a classe *MainXForm* disponibiliza um método que possibilita a fácil troca entre esses formulários para estabelecer o formulário corrente da

aplicação. Desta forma, esse método, em conjunto com o *showAlert(),* faz com que o *framework* XFormUI atenda ao requisito **3** especificado na seção 4.2.

Outra característica comum entre esses formulários é a possibilidade de adição de *Triggers* para disparar eventos de ação. Os *Triggers* são mapeados para a classe *Button* na implementação em Superwaba e para a classe *Command* em J2ME MIDP 1.0 e 2.0. No caso do método *showAlert()*, a implementação em Superwaba instancia e exibe um *MessageBox*, enquanto em J2ME MIDP a implementação utiliza uma instância da classe *Alert*. A seguir, é descrito em detalhes cada um desses formulários.

4.3.2.1. Select1Full

O *Select1Full* é um formulário para exibição de opções das quais o usuário pode escolher somente uma. O *Selec1Full* pode receber em seu construtor uma lista de opções e um título para ser exibido no topo do formulário. A esta classe, apenas um tipo de componente pode ser adicionado, o *Trigger*. Essa adição acontece através do método *add* que pode receber também diretivas de *layout* como alinhamento e mudança de linha no *display*.

Para que o engenheiro de software possa utilizar um *Select1Full*, ele precisa escrever uma subclasse dessa classe. Essa subclasse deve implementar os métodos *triggerEvent()* e *selectEvent()* que permitem o gerenciamento dos eventos de ações, respectivamente, sobre o *Trigger* e sobre a lista de opções.

O Select1Full foi mapeado em J2ME MIDP 1.0 e 2.0 para a classe *List* que têm exatamente o mesmo comportamento de escolha de opções e eventos de seleção. No caso do Superwaba, um componente foi criado através da combinação das classes *Container e ListBox.* Vale ressaltar que as diretivas de *layout* dos *Triggers* funcionam apenas em Superwaba, já que não se pode atribuir alinhamento aos *Commands* em J2ME.

4.3.2.2. TextArea

O *TextArea* é um formulário para exibição e edição de textos. Ele pode receber em seu construtor um texto e um título. À esta classe, assim como acontece com Select1Full, apenas um tipo de componente pode ser adicionado, o *Trigger*. Para que o engenheiro de software possa utilizar um *TextArea*, ele precisa escrever uma subclasse dessa classe. Ao contrário do *Select1Full*, essa subclasse deve implementar apenas o método *triggerEvent*. O *TextArea* foi mapeado em J2ME MIDP 1.0 e 2.0 para a classe *TextBox* que permite a edição

e exibição de textos. No caso do Superwaba, um componente foi criado através da combinação das classes *Container* e *Edit.*

Na Tabela 5, é apresentado um código de um formulário que utiliza o framework XFormUI para construir um *TextArea*. Uma subclasse foi criada e o método *triggerEvent* foi escrito. Nesse evento, um alerta é exibido caso uma ação seja disparada sobre o *Trigger* de alerta e o tamanho do texto no *TextArea* seja maior que 50 caracteres. Na Tabela 5, também podem ser visualizadas as telas do processo descrito.

Código em XFormUI	Telas no J2ME MIDP 2.0

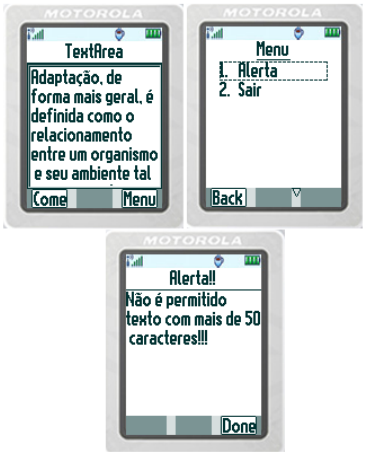
`import org.great.xform.ui.TextArea;`
`import org.great.xform.ui.Trigger;`
 	
`public class FormComentar extends TextArea{`
`//Trigger(s)`
`Trigger trgAlerta;`
 	
`FormComentar(MainXForm mainXForm){`
`…`
`public void triggerEvent(Trigger trig) {`
 	
`if(trig==trgAlerta){`
`  if(this.getValue().length()>50){`
`  this.showAlert("Não é permitido texto`
`  com mais de 50 caracteres!!!");`
`}}`
`}`
	

Tabela 5 - *TextArea*, *Triggers* e o *showAlert()* em J2ME MIDP 2.0

4.3.2.3. XForm

O *XForm* é o formulário padrão correspondente ao comportamento de um formulário XForms. A ele podem ser adicionados *Triggers* e qualquer componente que herde de *XFormItem*. Assim como nos outros formulários, a adição acontece através do método *add* que recebe diretivas de *layout* como alinhamento e mudança de linha no *display*. A diferença é que essas diretivas funcionam para os componentes que herdam de *XFormItem* tanto em Superwaba como em J2ME MIDP 2.0.

O *XForm* é implementado em J2ME MIDP estendendo a classe *Form*. No caso do MIDP 2.0, ele utiliza as diretivas de *layout* da classe *Item* para adicionar os componentes. Essas diretivas permitem alinhar os componentes à esquerda, ao centro e à direta e determinam se um ou mais componentes estão na mesma linha do formulário. No Superwaba, o *XForm* estende a classe *Container* e utiliza as diretivas da classe *Control* para adicionar os componentes.

4.3.3. Validações e Restrições

Para atender o requisito de disponibilizar mecanismos para validação e restrição das entradas de dados nos formulários (i.e., requisito **2)**, o *framework* contém as classes *StringRestriction, StringValidator, BindRestriction, BindValidator*, *NumberRestriction* e *NumberValidator*.

A classe *StringRestriction* permite atribuir aos componentes de entrada de dados *Input*, *Secret* e *TextArea* as seguintes restrições: número exato, tamanho máximo e mínimo de caracteres. A classe *StringValidator* disponibiliza um método que valida se uma cadeia de caracteres atende às restrições impostas em um objeto *StringRestriction*.

A classe *BindRestriction* permite associar restrições de valor requerido (i.e., *required*) ou de proibição de entrada de dados (i.e., *readonly*) para os componentes *Input*, *Secret* e *TextArea*. Essas restrições são passadas no início da construção desses componentes, pois influenciam na forma como eles são mapeados para as plataformas de programação. A classe *BindValidator* testa se a restrição de *required* está cumprida em um componente.

A classe *NumberRestriction* permite atribuir ao componente *Input* as seguintes restrições na entrada de dados numéricos: se o número é inteiro ou decimal, o número exato de dígitos, valores máximos e mínimos que o número pode assumir. A classe *NumberValidator* testa se uma cadeia de caracteres (i.e., *String*) segue as restrições impostas pelo *NumberRestriction*.

Os componentes *Input*, *Secret e TextArea* disponibilizam uma função que dispara a checagem das restrições, o método *isValid()*. Este método retorna verdade (i.e., *true*) se o valor contido no campo de entrada de dados satisfaz todas as restrições. Na Tabela 6, é apresentado o código reescrito do *TextArea* descrito na Tabela 5, utilizando as funções de validação disponibilizadas pelo *framework*. No construtor do formulário *FormComentar,* são chamados os métodos de restrição do tipo *BindRestriction* e *StringRestriction* (i.e., *setRequired* e *setMaxlength*). A checagem de validação acontece agora através do uso do método *isValid()* implementado pelo *framework*.

```
public class FormComentar extends TextArea{
//Trigger(s)
Trigger trgAlerta;

FormComentar(MainXForm mainXForm){
…
    //Restrições
    this.setRequired(true);
```

```
    this.setMaxLength(50);
}
public void triggerEvent(Trigger trig) {

if(trig==trgAlerta){
  if(this.isValid()){
      this.showAlert("Não é permitido texto vazio ou
                  com mais de 50 caracteres!!!");
}}}
```

Tabela 6 - Código do TextArea apresentado na Tabela 5 usando as classes de validação

4.3.4. *XFormItem* e Componentes

O *framework* XFormUI disponibiliza componentes para a construção de formulários do tipo *XForm*. Todos esses componentes são subclasses da classe abstrata *XFormItem* e devem implementar os métodos: *getValue(), setValue(String s), setLabelForeColor()* e *setLabelBackColor(),* definidos como abstratos na classe *XFormItem*. Os dois primeiros métodos definem modificadores sobre o conteúdo do componente, permitindo alterar e recuperar o valor existente. A semântica desse valor depende da implementação de cada componente que herda de *XFormItem.* Por exemplo, em um *Input*, esses métodos modificam e acessam o texto inserido no componente pelo usuário, enquanto no *Select1*, esses métodos modificam e acessam o item selecionado.

Os métodos *setLabelForeColor()* e *setLabelBackColor()* devem permitir que as cores dos *labels* dos componentes possam ser modificadas. Esses métodos recebem como parâmetros três inteiros que representam as componentes de uma cor no modelo de cores aditivo RGB (i.e., *red*, *green* e *blue*). Como mencionado anteriormente, na plataforma J2ME MIDP 1.0, esses métodos não possuem nenhum código, uma vez que, não é possível modificar as cores dos *labels* dos componentes nessa plataforma.

Na plataforma J2ME MIDP 1.0, o *XFormItem* contém uma instância de *Item*, classe da API padrão do J2ME. No caso do J2ME MIDP 2.0, o componente *XFormItem* contém um *ColorLabel,* uma classe também proposta nessa dissertação, e um *Item*. A plataforma J2ME MIDP 2.0 não contém nenhuma classe que permita a adição de cores a componentes, contudo, disponibiliza a classe *CustomItem* para a construção de componentes costumizáveis.

Assim, o *ColorLabel* foi construído para exibir cadeias de caracteres coloridos no *display* do dispositivo. No Superwaba, o componente *XFormItem* contém um *Label* e um *Control,* classes já disponibilizadas na API da plataforma. Os métodos de mudança de cores

são mapeados para os métodos disponibilizados pela plataforma para mudança de cores dos componentes: *setForeColor()* e s*etBackColor()*.

O *framework* XFormUI disponibiliza oito componentes que herdam de *XFormItem*: *Secret*, *Output*, *SelectBoolean*, *Select1*, *Select*, *XFormImage, Range e Input.* Esses componentes são descritos em detalhes a seguir.

4.3.4.1. Secret e Output

O *Secret* permite a entrada de textos e exibe para cada caractere digitado um "*" e é utilizado para entrada de senhas. Como mencionado anteriormente, a esse componente podem ser relacionadas às restrições do tipo *BindResctriction* e *StringRestriction*.

O *Secret* contém em J2ME MIDP 1.0 e 2.0 uma instância da classe *TextField* com a restrição *TextField.PASSWORD,* enquanto em Superwaba, o *Secret* contém um *Edit* com a restrição *Edit.PASSWORD.*

O *Output,* por sua vez, permite a exibição de textos que não podem ser editados. Esse componente é o único do XFormUI que permite que toda a cor de seu conteúdo seja modificada pelo engenheiro de software.

Em J2ME MIDP 1.0, o *Output* instancia um *StringItem*, enquanto no MIDP 2.0 instancia o *ColorLabel* provido pelo *framework*. Em Superwaba, a classe instancia um *Label*.

4.3.4.2. SelectBoolean, Select1, Select

Esses três componentes são utilizados para escolha de opções. No *SelectBoolean*, é apresentada uma única opção de escolha. No *Select1* e no *Select* são disponibilizadas várias opções. No caso do *Select1*, apenas uma pode ser escolhida.

Em J2ME MIDP 1.0 e 2.0, esses componentes são mapeados para o *ChoiceGroup.* O *SelectBoolean* e *Select1* utilizam a restrição *Choice.Exclusive* para explicitar que apenas uma opção pode ser escolhida. Enquanto, o Select usa a restrição *Choice.Multiple.* Em Superwaba, esses componentes são mapeados para, respectivamente, o *Check*, o *ComboBox* e o *ListBox*.

4.3.4.3. *XFormImage e Range*

O *XFormImage* define um componente para exibição de imagens. No J2ME MIDP 1.0 e 2.0, o *XFormImage* é composto pelas classes da API do J2ME, *ImageItem* e *Image*. No MIDP 2.0, elas permitem a exibição de imagens dos tipos JPG, PNG e GIF. No MIDP 1.0, como mencionado no Capítulo 2, apenas o formato de imagem PNG é suportado. No caso do *SuperWaba*, o *XFormImage* contém as classes da API *Image* e *Button* que permitem a exibição de imagens nos formatos BMP, PNG, JPG e GIF.

O Range permite a escolha de um valor dentro de um intervalo de valores. Em J2ME MIDP, ele instancia a classe *Gauge* e em Superwaba, instancia a classe *ScroollBar*. A implementação do método *getValue()* do *Range* retorna o valor escolhido dentro do intervalo que é estabelecido durante a instanciação de um *Range*.

4.3.4.4. Input

Esse componente permite a entrada de dados numéricos, texto, datas e horas. Durante a sua construção, o engenheiro de software informa que tipo de restrição é aplicada para filtrar a entrada dos dados. Ele pode escolher as seguintes restrições: *NUMERIC*, *DECIMAL*, *STRING*, *DATE*, *DATE_TIME*, *TIME.*

Contudo, essas restrições de filtragem podem não ser possíveis de executar em certas plataformas de programação. Nesses casos a checagem ocorre durante a execução do método *isValid()*.

Essas restrições também influenciam em quais componentes o *Input* instancia durante a execução da aplicação. Na Tabela 7, são apresentadas as formas como essas instanciações acontecem em cada plataforma na qual o *framework* foi implementado. Entre o J2ME MIDP 1.0 e MIDP 2.0 uma única diferença acontece na instanciação, no MIDP 1.0 a restrição *DECIMAL* do *Input* se transforma em uma restrição do tipo *TextField.ANY,* pois não existem restrições em MIDP 1.0 para entrada somente de valores decimais nos campos do tipo *TextField*.

Como mencionado no Capítulo 3, isso acontece devido a inexistência de suporte a ponto flutuante na plataforma. Como em MIDP 2.0, já existe o suporte a ponto flutuante, uma restrição para entrada de valores decimais nos *TextField* é disponibilizada, a

TextField.DECIMAL. No caso da restrição está relacionada a tempo ou a datas, o *Input* instancia um objeto da classe *DateField* disponibilizada pela API do J2ME MIDP.

Restrição	J2ME MIDP 1.0	J2ME MIDP 2.0	Superwaba
NUMERIC	*TextField* com restrição *TextField.NUMERIC*	*TextField* com restrição *TextField.NUMERIC*	*Edit com restrição Edit.CURRENCY e caracteres válidos* "0123456789-+"
DECIMAL	*TextField* com restrição *TextField.ANY*	*TextField* com restrição *TextField.DECIMAL*	*Edit com restrição Edit.CURRENCY e caracteres válidos* "0123456789.,-+"
STRING	*TextField* com restrição *TextField.ANY*	*TextField* com restrição *TextField.ANY*	*Edit com restrição Edit.NORMAL*
DATE	*DateField* com restrição *DateField. DATE*	*DateField* com restrição *DateField. DATE*	*Edit com restrição Edit.DATE*
DATE_TIME	*DateField* com restrição *DateField. DATE_TIME*	*DateField* com restrição *DateField. DATE_TIME*	*Edit com restrição Edit.NORMAL e caracteres válidos* "0123456789/:"
TIME	*DateField* com restrição *DateField. TIME*	*DateField* com restrição *DateField. TIME*	*Edit com restrição Edit com restrição Edit.NORMAL e caracteres válidos* "0123456789:"

Tabela 7- Restrições de filtragem e componentes instanciados nas plataformas

No caso do Superwaba, em todos os tipos de restrição, apenas um tipo de componente da plataforma é instanciado, o *Edit.* As restrições são implementadas na plataforma através do tipo do componente *Edit* e da máscara de filtragem aplicada aos caracteres. As máscaras para cada restrição do Input são apresentadas na Tabela 7. Vale ressaltar, que ao contrário do J2ME MIDP, no qual já existe um componente para permitir entrada de data e hora que exibe um calendário, no Superwaba a classe *Edit* apenas filtra a entrada de texto. Assim, para que um *Input* do tipo *DATE* tivesse o mesmo comportamento nas três plataformas,

uma chamada automática ao *Calendar,* classe da API do Superwaba que exibe um calendário, foi escrita dentro da classe *XForm*.

Na Figura 12, podem ser vistos um formulário com os componentes *Output, Secret, Select1, SelectBoolean* e um *Input* com a restrição do tipo *DATE* sendo executados nas três plataformas. É ainda exibido na figura o uso de cores e alinhamento dos componentes, assim como a execução da abertura de uma janela para escolha de uma data no componente *Input*.

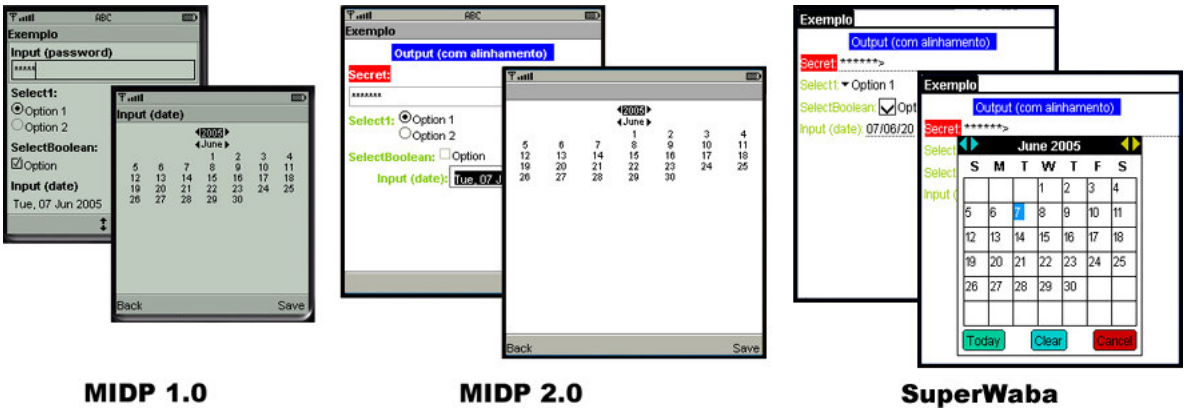


Figura 12 - Telas dos componentes nas plataformas

4.3.5. *Hot Spots* e Classes do XFormUI

Na Tabela 8, são apresentadas as classes que compõem o *framework* XFormUI e uma visão geral do seus mapeamentos para as três plataformas. Essas classes em conjunto oferecem uma boa quantidade de componentes que permitem a construção da maioria dos formulários desejados em uma aplicação para dispositivos móveis. Além disso, *Hot Spots* são disponibilizados para o engenheiro de software construir seus próprios componentes. Para que esse novo componente seja adicionado a um XForm, é suficiente que ele estenda a classe XFormItem e implemente os métodos abstratos definidos nela. Desta forma, o *framework* também atende ao requisito **7** de extensibilidade apresentado na seção 4.2.

XFormUI	J2ME MIDP 1.0	J2ME MIDP 2.0	Superwaba
MainXFform	*Midlet*	*Midlet*	*MainWindow*
XForm	*Form*	*Form*	*Container*
Select1Full	*List*	*List*	*Container* com um *ListBox*
TextArea	*TextBox*	*TextBox*	*Container* com um *ListBox*
XFormItem	Contém um *Item*	Contém um *Item* e um	Contém um *Control* e

		ColorLabel	um *Label*
Secret	Contém um *TextField*	Contém um *TextFied* e um *ColorLabel*	Contém um *Edit* e um *Label*
Input	Pode conter um *TextField* ou um *DateField* dependendo da restrição	Contém um *ColorLabel e* pode conter um *TextField* ou um *DateField* dependendo da restrição	Contém um *Edit* e um *Label*. No caso da restrição de tipo ser date, um *Calendar* é aberto ao se clicar no *Input*
Output	Contém um *StringItem*	Contém um *StringItem* e um *ColorLabel*	Contém um *Label*
Select1	Contém um *ChoiceGroup* com restrição *Exclusive*	Contém um *ColorLabel* e um *ChoiceGroup* com restrição *Exclusive* ou *Popup*	Contém um *Label* e um *ComboBox* ou um *RadioGroup ou um ListBox* dependendo do tipo do Select1
Select	Contém um *ChoiceGroup* com restrição *Multiple*	Contém um *ColorLabel* e um *ChoiceGroup* com restrição *Multiple*	Contém um *Label* e um *ListBox*
SelectBoolean	Contém um *ChoiceGroup* com restrição *Exclusive* e um único item	Contém um *ColorLabel* e um *ChoiceGroup* com restrição *Exclusive* e um único item	Contém um *Label* e um *Check*
XFormImage	Contém um *ImageItem* e um *Image*	Contém um *ImageItem* e um *Image*	Contém um *Image* e um *Button*
Trigger	Estende *Command*	Estende *Command*	Estende *Button*
Range	Contém um *Gauge*	Contém um *ColorLabel* e um *Gauge*	Contém um *ScrollBar*
triggerEvent()	*commandAction()*	*commandAction()*	*onEvent()*

Tabela 8 - Classes que compõem o XFormUI e seus mapeamentos nas plataformas

4.4. Conclusão

Neste Capítulo foi apresentado o *framework* XformUI que permite a construção de interfaces gráficas adaptativas e multi-plataformas para dispositivos móveis. O *framework* contém vários componentes de interface e facilita a construção de formulários com recursos de validação de dados e *layout* para os componentes. As implementações do *framework* em J2ME MIDP1.0/CLDC 1.0, J2ME MIDP2.0/CLDC1.1 e Superwaba proporcionam a execução das aplicações em três plataformas distintas.

Assim, de acordo com as definições de adaptação apresentadas no Capítulo 2, o *framework* provê **adaptação à descrição em tempo de construção da interfaces das aplicações**. Contudo, o engenheiro de software ainda tem que escrever código para compor os formulários e realizar as ações de validação. Além disso, algumas funcionalidades do *framework* não funcionam em certas plataformas de programação (e.g., mudança de cores no J2ME MIDP 1.0), o que representa um desperdício de código, já que chamadas de métodos são executadas em vão (e.g., *setLabelForeColor*() no J2ME MIDP 1.0/CLDC 1.0).

Dessa forma, para evitar que métodos que não executam código sejam invocados e para agilizar a criação de interfaces que utilizem o *framework* XFormUI é apresentada a ferramenta UIG no próximo capítulo. Essa ferramenta permite ao engenheiro de software descrever as interfaces de uma aplicação usando uma linguagem declarativa, e com essa descrição, a ferramenta gera código que utiliza o *framework* XformUI.

5. User Interface Generator

Neste capítulo, é apresentada a ferramenta de geração de código User Interface Generator (UIG) proposta nesta dissertação. Essa ferramenta gera código dos formulários de uma aplicação a partir de uma descrição inicial desses formulários em uma linguagem de marcação.

Na Seção 5.1, é apresentada uma visão geral da ferramenta UIG. Na Seção 5.2, é descrita a linguagem suportada pela ferramenta. Na Seção 5.3, são apresentadas as regras de mapeamento dos documentos para o *framework* XFormUI. Na Seção 5.4, é descrito o processo de geração de código. Na Seção 5.5, é detalhada a estrutura do código gerado. Por fim, na Seção 5.6 é apresenta o *plugin* do eclipse construir para auxiliar o uso da ferramenta UIG.

5.1. Visão geral da ferramenta User Interface Generator

Conforme mencionado no Capítulo 2, um dos desafios do desenvolvimento de aplicações para DMs consiste na descrição, independente de plataforma ou dispositivo, da interface com usuário das aplicações. O *framework* XFormUI soluciona em parte esse desafio, visto que, possibilita a construção de interfaces vinculadas ao *framework* e não a uma determinada plataforma. Além disso, as diretivas de cores e alinhamento disponibilizadas e implementadas pelo XFormUI são independentes de dispositivo.

Contudo, para utilizar o *framework* o engenheiro de software ainda escreve código para montar as interfaces e para disparar os eventos que executam a validação dos dados. Com o objetivo de agilizar esse processo de construção de interfaces e permitir que o engenheiro de software utilize uma linguagem declarativa para descrever as restrições dos dados e as interfaces, essa dissertação apresenta a ferramenta User Interface Generator (UIG).

Na Figura 13, é apresentada uma visão geral dessa ferramenta. A UIG permite ao engenheiro de software descrever os formulários de uma aplicação utilizando o XForms para definir os componentes, o CSS para definir o estilo (i.e., cores) e o XML Schema para definir restrições aos campos de entrada de dados do formulário. O engenheiro cria um documento XML, o *Manifest.xml*, que define quais formulários serão mapeados e para qual

plataforma o código será gerado. Ele dispara a ferramenta UIG e esta, por sua vez, gera o código dos formulários utilizando o framework XFormUI.

Para criar esse *Manifest.xml,* o engenheiro pode utilizar o *plugin* para a ferramenta livre Eclipse [37] disponibilizado pelo ambiente proposto nesta dissertação. Esse *plugin* permite a construção do *Manifest.xml* de forma gráfica e também dispara o processo de geração de código da ferramenta.

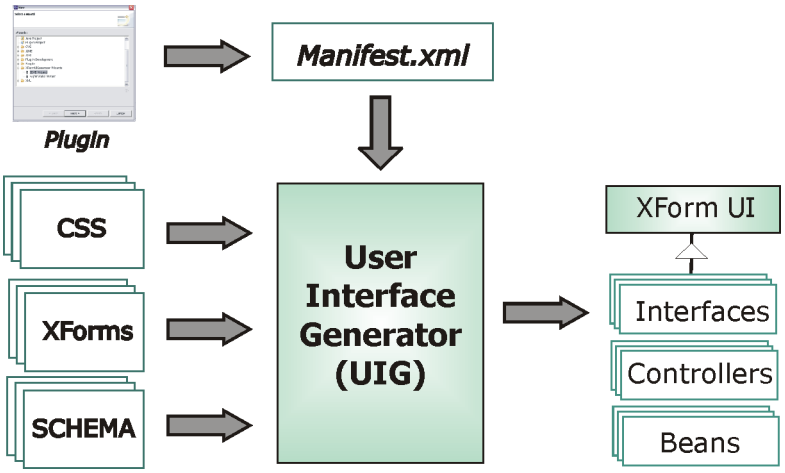


Figura 13 - Visão geral da ferramenta UIG

A ferramenta realiza essa geração de código utilizando XSL [36] (i.e., *eXtensible Stylesheet Language*). Essa linguagem de marcação baseada em XML permite a criação de folhas de estilo ou documentos de transformação chamados de XSLT (i.e., XSL *Transformation*). O XSLT permite a definição de regras de mapeamento das *tags* de um documento XML para uma outra descrição. Com XSLT, é possível converter um documento XML em outros documentos XML, em arquivos texto ou em documentos HTML. A vantagem do uso do XSLT decorre da existência de vários processadores ou *parsers* XSL, capazes de realizar a transformação a partir de dois documentos: o documento XML original e o documento XSLT de transformação.

O código dos formulários gerados pela ferramenta seguem um modelo semelhante ao MVC (i.e., *Model/Viewer/Controller*) [41]. Para cada formulário, três classes são geradas: *Interface*, *Bean* e *Controller*.

A classe *Interface* define a visão do formulário na qual estão descritos os componentes gráficos e como eles se compõem para constituir o formulário. Esses

componentes gráficos utilizados são instâncias das classes que fazem parte do *framework* XFormUI apresentado no Capítulo 4.

A classe *Bean* contém atributos relacionados aos campos de seleção e de entrada de dados do formulário, assim, essa classe corresponde ao modelo na qual estão os dados relacionados ao formulário. Por sua vez, a classe *Controller* contém métodos de execução para cada *Trigger* existentes nos formulários.

Esse modelo de código gerado permite o desenvolvimento mais rápido das aplicações, pois o engenheiro se concentra na escrita dos métodos da classe controladora que são referentes às regras de negócio da aplicação. Na Seção 5.5, as estruturas dessas classes são detalhadas.

5.2. Linguagem suportada pela ferramenta

Como descrito anteriormente, a ferramenta utiliza como entrada, para a geração de código, documentos XForms, CSS e XML Schema. Para utilizar essas tecnologias na descrição de um formulário, o engenheiro de software utiliza um documento XHTML, no qual estão incluídos as *tags* do XForms e as referências aos arquivos do CSS e do XML Schema.

Desta forma, além de suportar essas três tecnologias, a ferramenta UIG também mapeia certas *tags* do XHTML, listadas a seguir:

- *xhtml:html,* define o início do documento;
- *xhtml:link*. , onde está o caminho para arquivo que define o estilo do formulário;
- *xhtml:head,* descreve o início do cabeçalho do XHTML que pode ter dois filhos:
  - *xhtml:title,* é o título do formulário;
  - *xforms:model,* é a parte *Model* do XForms;
- *xhtml:body,* contém os componentes XForms e as *tags* XHTML;
- *xhtml:img,* adiciona uma imagem ao formulário;
- *xhtml:p,* é o parágrafo para separar os componentes em linhas.

Como pode ser visto, além das *tags* que definem a estrutura do documento XHTML, a ferramenta suporta a *tag xhtml:img* para adicionar imagens a um formulário e o *xhtml:p* para definir o *layout* dos componentes.

5.2.1. Suporte ao XForms

Como mencionado no Capítulo 4, um formulário XForms é dividido em 3 partes: *Model, User Interface e Instance*.

O XForms *Model* suportado pela UIG é semelhante ao *Model* original do XForms, com a restrição de que apenas um único modelo exista em cada formulário. Para descrever o esquema do documento XML contido no modelo, o engenheiro de software pode relacionar a ele um arquivo XML Schema. Essa relação acontece através do atributo *schema* do *Model*.

Além das restrições impostas pelo XML Schema, que serão descritas em detalhes nas próximas subseções, a UIG suporta também a *tag Bind* para definir restrições. Esta *tag* permite ao engenheiro de software associar uma restrição a um campo do modelo. As restrições suportadas pela UIG são:

· *Required*: pode ser "*false()*" ou "*true()*" e define se um valor do modelo é obrigatório.

· *Readonly*: pode ser "false()" ou "true()" e define se um valor do modelo pode ser modificado. Neste caso, o componente relacionado a esse valor do modelo torna-se *readonly.*

Com relação à parte *User Interface* do XForms, a UIG suporta todos os componentes XForms que são descritos nessa parte do documento, exceto o *upload.* Os componentes suportados, juntamente com o *xhtml:img*, disponibilizam ao engenheiro de software uma variedade de componentes para a construção das interfaces de uma aplicação.

Na Tabela 9, são listados os componentes XForms suportados, sua sintaxe e a imagem no navegador XSmiles [15].

Componente	Descrição	Sintaxe	Imagem no XSmiles

Input	Componente para entrada de dados	`<input id="inp" ref="">`
		`  <label>Input:</label>`
		`</input>`
			Input: valor

Secret	Componente para entrada de senhas	`<secret id="sec" ref="">`
		`  <label>Secret:</label>`
		`</secret>`
			Secret: *****

Select1	Componente para escolha de um entre	`<select1 id="sel" ref="">`
		`  <label>select1:</label>`
		`  <item>`	

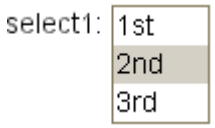
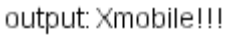
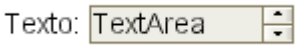
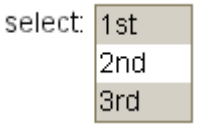
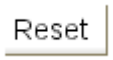
	alguns itens	<value>1</value> <label>1st</label> </item> </select1>	
Output	Componente para exibição de dados	<output id="out" ref=""> <label>output:</label> </output>	
Textarea	Componente para entrada de cadeias grandes de caracteres	<textarea id="tex" ref=""> <label>Texto:</label> </textarea>	
Select	Componente para escolha de itens	<select id="sel" ref=""> <label>select:</label> <item> <value>1</value> <label>1st</label> </item> </select1>	
Selectboolean	Componente para marcar ou não um item	<selectboolean id="sb" ref=""> <label>Selectboolean:</label> </selectboolean>	
Range	Componente para escolha de valores em um intervalo	<range end="20" ref="" id="rg" start="1" step="1"> <label>Range:</label> </range>	
Trigger	Gatilho para disparar ações	<trigger id="trig"> <label>Reset</ label> <reset/> </trigger>	
Submit	Gatilho que dispara a ação de submissão	<submit id="sub"> <label>Submit</ label> </submit>	

Tabela 9 - Componentes *XForms* Suportados

Como pode ser visto na tabela, cada componente obrigatoriamente contém o atributo *id* e o nó filho *label*, respectivamente, o identificador e o texto que é exibido ao lado de cada componente. Os componentes, exceto o *trigger* e o *submit*, possuem o atributo *ref* com o caminho XPath referente ao a seu valor correspondente no *Model* do XForms.

Os componentes *input*, *secret* e *textarea* podem possuir o nó filho *alert*. O nó *alert* contém uma mensagem que é exibida no caso do texto inserido não corresponder à restrição estabelecida no *Bind* ou no *Schema* para o nó do modelo ao qual o componente se refere (e.g. o texto não satisfaz a restrição *integer*). O componente *trigger,* por sua vez, pode conter o nó filho *reset* que indica que sua ação é reinicializar os valores de todos os componentes do formulário.

A UIG suporta também dois atributos adicionais ao *XForms*:

- O atributo *maxlength* para limitar o tamanho do texto a ser inserido no caso dos componentes *input*, *secret* e *textarea*;
- O atributo *align* que pode ser *left*, *right* e *center* que indica a posição relativa dos componentes no *display*.

5.2.2. Suporte ao XML Schema

Como mencionado anteriormente, o XML Schema é utilizado no XForms para associar restrições aos campos de dados do modelo. Quando o engenheiro de software associa, usando o atributo *ref,* um componente XForms a um valor do modelo, a restrição referente a esse valor é repassada para o componente.

No caso da UIG, essas restrições se tornam as restrições das classes de validação e restrição do *framework* XFormUI. As restrições do XML Schema são especficificadas através do estabelecimento de tipos para os dados. Esses tipos podem ser básicos, simples ou complexos. Os tipos básicos são os tipos já suportados pelo XML Schema (e.g., *xs:string*). Os tipos simples são aqueles definidos a partir dos tipos básicos aplicando-se restrições. Os tipos complexos são definidos usando expressões regulares ou uma árvore de nós XML. Os tipos XML Schema suportados pela UIG, são os seguintes:

- Tipos básicos:
  - *xs:string*;
  - *xs:date*, *xs:datetime*, *xs:time*;
  - *xs:integer*, *xs:nonNegativeInteger*, *xs:positiveInteger*, *xs:integer*, *xs:nonPositiveInteger*, *xs:negativeInteger*;
  - *xs:decimal*, *xs:double*.
- Tipos simples:

- Base *xs:string* com as restrições *length*, *maxLength* e *minLength* .
- Base *xs:integer, xs:nonNegativeInteger, xs:positiveInteger, xs:integer, xs:nonPositiveInteger, xs:negativeInteger* com as restrições *minExclusive*, *maxExclusive*, *minInclusive*, *maxInclusive.*
- Base *xs:decimal, xs:double* com as restrições *minExclusive*, *maxExclusive*, *minInclusive*, *maxInclusive*, *fractionDigits*.

5.2.3. Suporte ao Cascading Style Sheets (CSS)

O CSS é utilizado no XForms para formatação de estilo das páginas e dos componentes. Na UIG, são suportadas as modificações das cores de todo componente, a mudança nas cores do *label* e a modificação das cores do *value* (e.g., o texto contido em *input*). O uso do CSS pode ser realizado de duas formas, listadas a seguir:

- Com modificadores para todos os componentes. Nesse caso, o estilo inicia pelo tipo de componente ao qual ele está associado (e.g., input{...} para todos os inputs);
- Com a criação de classes. Nesse caso, um novo estilo é criado no CSS (e.g., texto {...}). No XForms, o engenheiro de software associa essa nova classe de estilo, usando o atributo *class*, em cada componente que desejar.

Na Tabela 10, são apresentados exemplos de como o CSS pode ser usado para modificar estilo de cores nos componentes XForms.

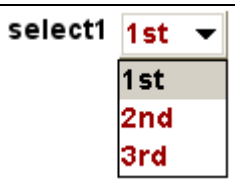
Descrição	Atributos	Sintaxe	Imagem no XSmiles
Modificar atributos de todos os componentes de um tipo (e.g., todos os *inputs*)	Background-color e Color dos componentes	`input{ background-color: rgb(255,255,100); color: #AAAAAA }`	
Modificar atributos do *label* de todos os componentes de um tipo	Background-color e Color do label	`input>label{ background-color: #000000; color: rgb(255,255,255); }`	
Modificar atributos do *value* de todos os componentes de um tipo	Background-color e Color do value	`select1::value{ background-color: #FFFFFF; color: rgb(155,0,0); }`	

Tabela 10 - Exemplo de estilos do CSS suportados pela UIG

5.3. Regras de mapeamento para as plataformas

Como mencionado no Capítulo 4, o *XForms* foi escolhido como fonte de nomes dos componentes XFormUI para facilitar o mapeamento realizado pela ferramenta UIG. Dessa forma, os mapeamentos, em sua maioria, acontecem de componentes XForms para componentes de mesmo nome em XFormUI.

Os casos em que esse mapeamento não acontece de forma direta são os seguintes:

- Componente *submit*. Esse componente XForms é mapeado para o componente XFormUI *Trigger* e uma função que checa se todos os dados do formulário são válidos é chamada no evento desse gatilho de ação.

- Componente *textarea.* Nos casos em que o formulário contém um *textarea* e outros componentes diferentes do *trigger*, o seu mapeamento depende da plataforma de programação. Se as plataformas desejadas forem J2ME MIDP 1.0 ou 2.0, o componente é mapeado para o formulário *TextArea* e os componentes diferentes de *triggers* são descartados na geração do código. Se o mapeamento for para Superwaba, o *textarea* é mapeado para um *Input* e os outros componentes são adicionados ao formulário. O mapeamento de textarea para *TextArea* só ocorre em Superwaba quando apenas *triggers* e um *textarea* são descritos no XForms.

- Componente *select1* com atributo *appearance="full"*. O mapeamento desse componente depende da plataforma de programação quando o formulário contém um *select1* com atributo *appearance="full"* e outros componentes diferentes do *trigger*. Se as plataformas desejadas forem J2ME MIDP 1.0 ou 2.0, o componente é mapeado para o formulário *Select1Full* e os componentes diferentes de *triggers* são descartados na geração do código. Se o mapeamento for para Superwaba, o *select1* é mapeado para um Select1 e os outros componentes são adicionados ao formulário. O mapeamento de Componente *select1* com atributo *appearance="full"* para Select1Full só ocorre em Superwaba quando apenas *triggers* e um *select1* são descritos no XForms.

As restrições impostas pelo XML Schema e pela Bind se transformam em chamadas dos métodos dos componentes do XFormUI que implementam essas restrições. Por exemplo, se existe um *Bind* de *required="true"* no XForms, essa restrição é mapeada para

a chamada do método *setRequired()* do componente associado a essa restrição (e.g., um *Input*).

As diretivas de alinhamento adicionadas ao XForms (i.e., o atributo *align*) e os estilos do CSS são mapeados para as funções do *framework* XFormUI que permitem esse mesmo comportamento. Caso o mapeamento seja feito para uma plataforma que não possuí diretivas de *layout* e os métodos do *framework* não contenham nenhum código a ser executado (e.g., mudança de cor no J2ME MIDP 1.0), o mapeamento das diretivas de *layout* não é realizado.

5.4. Processo de geração de código

Na Figura 14, é apresentado o processo de geração de código da ferramenta UIG. Como mencionado anteriormente, a ferramenta utiliza XSLT para realizar as transformações dos documentos XMLs para código Java que estende o *framework* XFormUI. Contudo, uma das entradas da ferramenta é um documento CSS que não é baseado no padrão XML.

Assim, para realizar o processo de mapeamento, o primeiro passo é transformar o CSS em um documento XML. Para isso, foi criado um *Document Type Definition (DTD)* que especifica um documento XML para suportar os parâmetros de um documento CSS, o XCSS. Após a geração do XCSS, esse documento, em conjunto com o documento XForms e o XML Schema, são transformados para um documento intermediário o XMLMiddleForm.

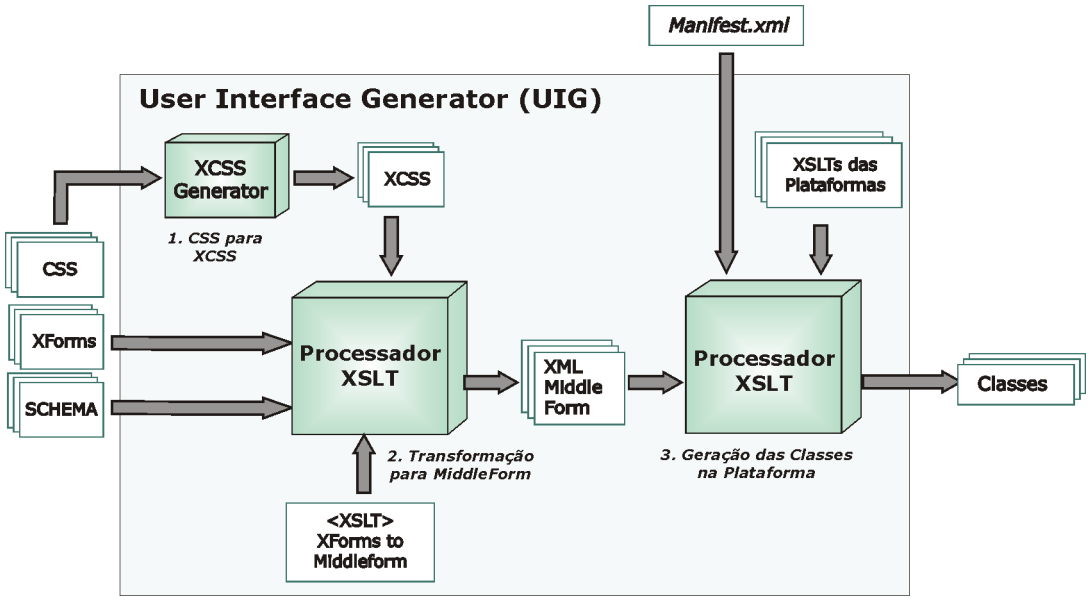


Figura 14 - Processo de geração de código

O XMLMiddleForm é a forma canônica definida pelo UIG para facilitar o mapeamento para as plataformas de programação. Os documentos são primeiramente transformados para XmlMiddleForm e depois de XMLMiddeForm para as plataformas de programação. Essa abordagem é a aplicação do padrão de geração de código *Meta-Model*, descrito em [102], que tem como vantagem facilitar a construção de novos tipos de mapeamentos (e.g., para uma outra plataforma de programação).

Após a geração do XMLMiddleForm, o *Manifest.xml* é lido e identificado a plataforma desejada pelo engenheiro de software para realizar a transformação. Assim, a UIG passa para o processador XSLT os *templates* que mapeiam de XMLMiddleForm para classes em Java.

A construção da ferramenta UIG foi realizada em Java e utilizou o JAXP [71] (i.e., *Java API for XML Processing*). O JAXP contém várias classes que facilitam a manipulação de documentos XML e um processador XSLT capaz de realizar as transformações desejadas pela ferramenta.

5.5. A estrutura do código gerado

Como mencionado anteriormente, para cada formulário, a ferramenta gera três classes: a classe *Interface*, a classe *Bean* e a classe *Controller*.

A classe *Interface* é dividida em quatro partes: declaração, instanciação, adição dos componentes e controle de eventos. Na declaração, são gerados os códigos que declaram os componentes da interface do formulário, todos eles são declarados como ***protected***. Na parte de instanciação, dois construtores são disponibilizados, um que constrói o formulário com os dados estáticos preenchidos no documento XForms e um outro construtor, no qual é passado como parâmetro o *Bean* com novos dados para preencher os campos do formulários.

Assim, esses construtores possibilitam ao engenheiro de software tanto criar um formulário que já possua alguns valores preenchidos, quanto criar um formulário com valores dinâmicos, modificados a cada instanciação.

Na parte de controle de eventos, são criados métodos que tratam os eventos dos *Triggers* e repassam o fluxo para um método correspondente da classe *Controller*. Esses métodos da classe *Controller* recebem como parâmetro uma instância do *Bean* com os valores atuais dos campos dos formulários.

Por exemplo, se um formulário XForms, chamado de *FormLogin*, contém um *trigger* cujo o *id* é tgOk, o mapeamento da UIG gera as classes *FormLogin*, *FormLoginBean* e *FormLoginController*. Na classe *FormLogin*, dentro do método *triggerEvent()* é gerado um código que testa se o alvo do evento é o *tgOK*. Caso seja verdade, o método *tgOkAction()* da classe *FormLoginController* é acionado e nele é passado como parâmetro uma instância de *FormLoginBean* com os valores atuais dos campos preenchidos do formulário (e.g., *login* e senha). Desta forma, o engenheiro de software necessita apenas escrever código na classe controladora. Nos casos em que está definido um *submit* no XForms, um método que checa todas as restrições que compõem o formulário é chamado antes de repassar o fluxo pra o método do controlador.

5.6. Plugin do eclipse

O UIG *plugin* é projetado e construído no ambiente Eclipse. Funcionalmente, o UIG *plugin* utiliza *wizards* para simplificar a definição dos diversos atributos da transformação contidos no *Manifest.xml*, tais como:

- A plataforma de programação alvo do mapeamento (e.g., J2ME MIDP 1.0 ou Superwaba);

- A lista de arquivos que devem ser transformados;
- A pasta de destino no sistema de arquivos para onde os arquivos gerados devem ser copiados.

Em um *wizard*, é definida uma seqüência de passos pela qual o usuário pode navegar através dos botões "*NEXT*" e "*BACK*", finalizando, enfim, com o botão "*FINISH*".

Os *wizards* do *plugin* seguem os seguintes passos:

1. O usuário escolhe qual *wizard* irá utilizar, de acordo com a plataforma alvo. Esse passo é apresentado na Figura 15.

2. O usuário decide se utilizará definições de uma transformação já realizada pelo *plugin* ou irá realizar uma nova transformação, especificando novas definições.

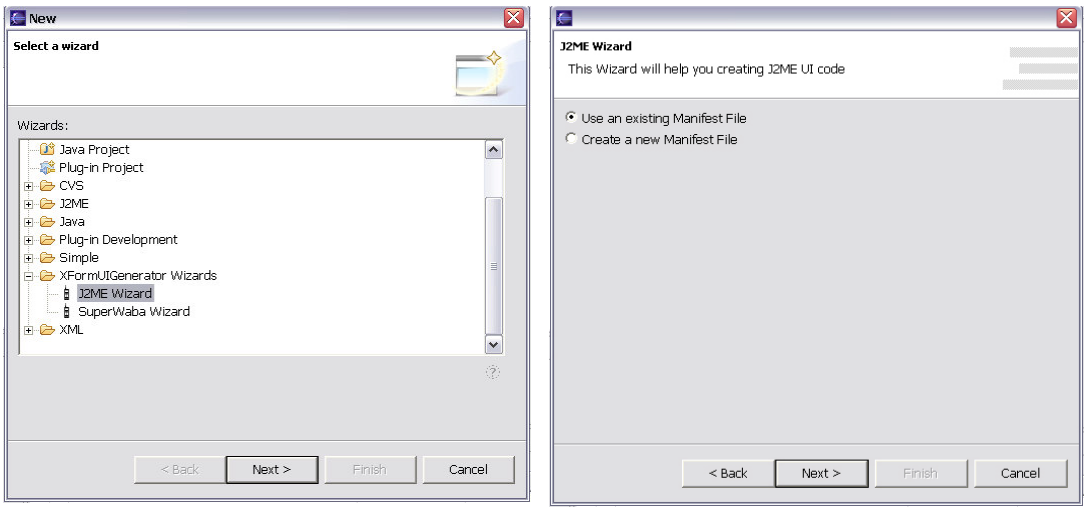


Figura 15 – Passos 1 e 2 do *wizard*

3. Um dos dois passos seguintes é realizado, de acordo com a escolha do passo anterior.

   a. Se escolher por utilizar definições de uma transformação já realizada, o usuário deve informar onde o *plugin* deve buscar o arquivo *Manifest.xml*, gerado anteriormente.

   b. Se escolher por uma nova transformação, deverá informar o nome do projeto a ser criado e os dados específicos de transformação de cada plataforma, como a versão do J2ME.

4. Nesse passo, os arquivos a serem transformados, seus atributos de transformação e seus componentes são apresentados, para que o engenheiro de software possa confirmá-los

e, caso seja necessário, adicionar ou remover mais arquivos. Esse passo é mostrado na Figura 16.

5. Finalmente, o usuário deve escolher a pasta onde os arquivos serão transformados, dando início à transformação propriamente dita. A transformação é transparente para o usuário e uma barra de progresso mostra o andamento do processo. Todos os passos da transformação são detalhados em um arquivo de *log*, salvo também na pasta escolhida para gerar os arquivos.

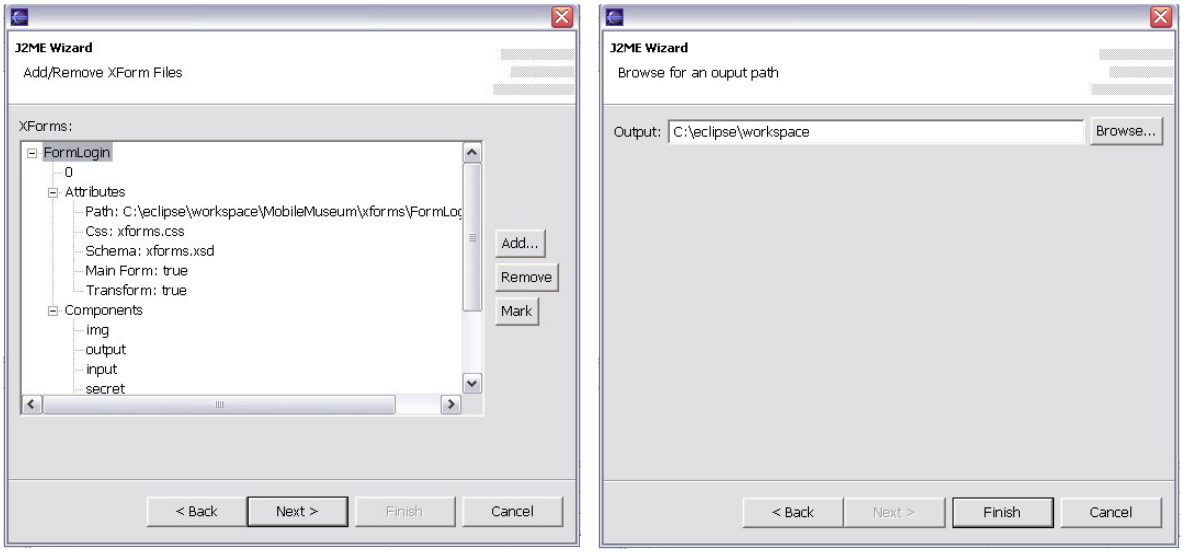


Figura 16 - Passos 4 e 5 do *wizard*

5.7. Conclusão

Neste capítulo, foi descrita a ferramenta User Interface Generator (UIG) que facilita a construção de aplicações que utilizam o *framework* XFormUI. Com a ferramenta, o engenheiro de software pode construir formulários, as restrições dos dados e o estilo dos componentes usando linguagens declarativas que agilizam o processo de desenvolver. Esses formulários descritos são mapeados pela ferramenta em código executável nas plataformas de programação para dispositivos móveis J2ME MIDP 1.0, J2ME MIDP 2.0 e Superwaba.

No próximo capítulo são apresentados os *frameworks* que permitem a construção de MAS que adaptem conteúdo e seus clientes para dispositivos móveis. Vale ressaltar que o código gerado pela UIG é facilmente integrável a esses *frameworks*.

6. *Frameworks* Mobile Adapter, Requisitor e MobAC

Neste capítulo são descritos os *frameworks* Mobile Adapter, Requisitor e MobAC que, em conjunto, facilitam a construção de *Mobile Application Servers* que adaptam conteúdo e aplicações para dispositivos móveis que acessam essas informações adaptadas.

Na seção 6.1, são descritas as especificações XML CC/PP e UAProf utilizadas pelos *frameworks* como documentos para troca e descoberta das informações do usuário, do dispositivo e do contexto. Na seção 6.2, é apresentado o Mobiler Adapter e seu relacionamento com o MobAC. Na seção 6.3, é descrita a estrutura do *framework* Mobile Adapter. Na seção 6.4, é apresentado o *framework* multi-plataforma Requisitor para realização de comunicação de dados do tipo *Request-Response.* Por fim, na seção 6.5, é descrito em detalhes o processo de monitoração e montagem do documento CC/PP pelo MobAC e como ele utiliza o *framework* Requisitor para trocar dados com o Mobile Adapter.

6.1. CC/PP e UAProf

A adaptação de conteúdo para dispositivos móveis pode basear-se nas características do dispositivo, nas preferências do usuário e nas condições de contexto em que ambos se encontram. Para fornecer essas informações para os MASs realizarem a adaptação de conteúdo, as especificações *Composite Capability/Preference Profiles* (CC/PP) e *User Agent Profiling Specification* (UAProf) podem ser utilizadas.

O CC/PP [47] é uma especificação do W3C [36] para expressar características dos dispositivos e das preferências de usuários. O CC/PP é baseado no *framework* RDF [36] para descrição de documentos XML que permite uma maior flexibilidade na criação de novos vocabulários. O CC/PP especifica uma estrutura geral de um documento XML para descrever as características dos dispositivos e dos usuários. Nos perfis dos dispositivos, o documento pode ser dividido em componentes (e.g., TerminalHardware, TerminalSoftware) e fornece alguns vocabulários padrões para descrever características básicas dos dispositivos (e.g., dimensões do *display*, versão do sistema operacional).

Contudo, o CC/PP não evita que cada fabricante ou engenheiro de software crie um vocabulário próprio, o que dificulta a integração de sistemas para obtenção das características dos dispositivos.

O *Open Mobile Alliance* [96] criou a especificação UAProf com o objetivo de padronizar os vocabulários CC/PP entre os fabricantes de dispositivos móveis. UAProf consiste em um conjunto de vocabulários para descrever as características dos dispositivos e dos *User Agent Browsers* existentes nos mesmos. Para cada dispositivo criado pelos fabricantes, um documento UAProf é criado e armazenado no repositório *Web* do *site* do fabricante. Esse documento contém informações como características do *display* do dispositivo, informações sobre sistema operacional e softwares disponíveis. Na Figura 17, é apresentada uma parte do documento UAProf do dispositivo P900 disponível no *site* da Sony Ericsson [97]. No documento podem ser vistas informações sobre a quantidade de cores do dispositivo, dimensões da tela, número de linhas e colunas de texto.

```
<prf:component>
  <rdf:Description rdf:ID="HardwarePlatform">
    <prf:BitsPerPixel>16</prf:BitsPerPixel>
    <prf:ColorCapable>Yes</prf:ColorCapable>
    <prf:ScreenSize>208x320</prf:ScreenSize>
    <prf:ImageCapable>Yes</prf:ImageCapable>
    <prf:ScreenSizeChar>20x15</prf:ScreenSizeChar>
    <prf:TextInputCapable>Yes</prf:TextInputCapable>
    <prf:Vendor>Sony Ericsson</prf:Vendor>
  </rdf:Description>
</prf:component>
```

Figura 17 - Documento UAProf do P900 retirado do site da Sony Ericsson [97]

No ambiente proposto nessa dissertação, os *frameworks* Mobile Adapter e MobAC utilizam o UAProf como fonte de informação das características estáticas dos dispositivos. Assim, ao contrário da arquitetura NAC [8], os engenheiros de softwares poderão criar os MAS sem a necessidade de descrever cada dispositivo. Já os documentos CC/PP são utilizados para especificar as características dinâmicas do contexto e as preferências do usuário.

6.2. Mobile Adapter e MobAC

Para construir aplicações que adaptam conteúdo, certas questões devem ser respondidas [26]: (i) como os perfis dos dispositivos e dos usuários podem ser obtidos; (ii) que políticas de adaptação são empregadas; e (iii) quando a adaptação deve ser executada. Em [10], [11]

e [12], uma única entidade, um *proxy*, responde a essas questões. Um *proxy* adapta os recursos a cada requisição e é o responsável por delegar como e quando realizar a adaptação. Contudo, como mencionado no Capítulo 2, essa abordagem concentra toda a decisão da adaptação em uma arquitetura geral que pode não ser satisfatória para outros tipos de aplicações que não consistam na simples requisição de um conteúdo *Web*.

No Mobiler Adapter e no MobAC, as decisões relacionadas com a adaptação são delegadas para os MAS e para as aplicações no dispositivo móvel. As implementações desses *frameworks* se encarregam então de capturar, manipular e gerenciar os perfis dos dispositivos, dos usuários e do contexto. Desta forma, permite-se a criação de sistemas mais flexíveis quanto à adaptação, uma vez que a obtenção das informações é separada dos métodos de adaptação.

O Mobile Adapter consiste em um *framework Web* para criação de MAS adaptativos. O Mobile Adapter Client (MobAC) é um cliente multi-plataforma para dispositivos móveis que é capaz de trocar informações com o Mobile Adapter. Na Figura 18, uma visão geral das classes que compõem esses *frameworks* e seus relacionamentos com um MAS e com os repositórios de especificações UAProf são apresentados.

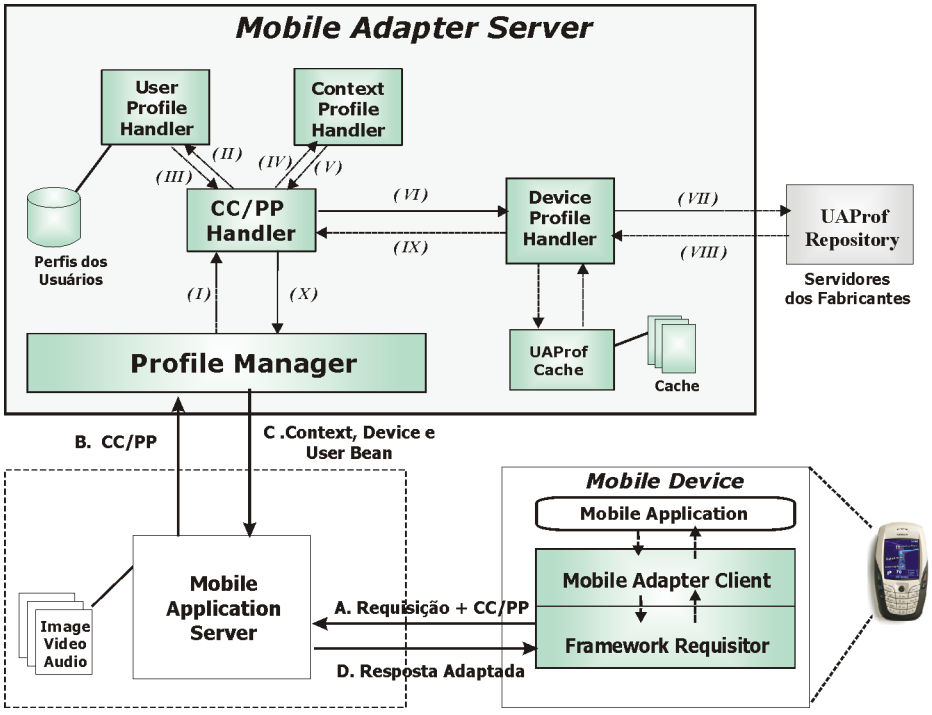


Figura 18 - Visão geral do Mobile Adapter e do MobAC

O MobAC é responsável por capturar as informações dinâmicas do contexto referentes ao dispositivo e ao ambiente de execução (e.g., memória livre, localização), assim como é responsável por capturar a identificação do dispositivo ou a URI da especificação UAProf e a identificação do usuário. De posse desses valores, a cada requisição, o MobAC gera um documento CC/PP que é enviado juntamente com a requisição da aplicação (A). Ao contrário do trabalho [113], somente os atributos dinâmicos do dispositivo são enviados a cada requisição, o que reduz o tamanho da informação a ser enviada.

Após a geração do documento CC/PP, o MobAC envia ao MAS o documento e a requisição (A). Esse envio é realizado utilizando uma implementação do *framework* Requisitor que é descrito na seção 6.4. O MAS repassa o documento ao Mobile Adapter (B) para que este forneça as informações estáticas e dinâmicas do dispositivo, do usuário e do contexto. A entidade central do Mobile Adapter, chamada de *ProfileManager*, instancia o *CC/PPHandler* para processar o documento CC/PP (I).

Durante o processamento, o *CC/PPHandler* separa as informações contidas no documento em três partes: ID do usuário, informações do contexto e URI do documento UAProf. As partes são repassadas para os manipuladores específicos de cada informação, respectivamente: *UserProfileHandler*, *ContextProfileHandler* e *DeviceProfileHandler*.

A classe *UserProfileHandler* lê o ID do usuário (II) e recupera o seu perfil na base de dados XML do *framework*. Estes perfis dos usuários devem ser previamente cadastrados pelo MAS, utilizando o *ProfileManager* e também devem estar descritos em CC/PP. De posse do perfil, o *UserProfileHandler* gera um objeto (III) que contém todos os dados do usuário descritos no documento, com métodos de acesso que seguem a especificação Java *Beans* [118]. O *ContextProfileHandler*, de forma semelhante, após receber as informações relacionadas ao contexto (IV) gera um objeto Java *Beans* povoado (V) com os dados contidos no documento CC/PP enviado pela aplicação do DM.

O *DeviceProfileHandler*, por sua vez, recebe do *CC/PPHandler* a URI do documento UAProf que descreve as propriedades estáticas do dispositivo (VI). O *DeviceProfileHandler* utiliza o *UAProfCache* para checar se esse documento já se encontra no *cache* do *framework*. Caso não seja encontrado, o *download* do documento é realizado junto ao *site* dos fabricantes (VII e VIII). O *DeviceProfileHandler*, de posse do documento

UAProf, gera também um objeto Java *Beans* correspondente aos dados estáticos que serão utilizados para adaptação (IX).

Assim, após os processo de geração dos *Beans*, o *CC/PPHandler* retorna um *UserBean*, um *DeviceBean* e um *ContextBean* ao *ProfileManager* (X). Este, por sua vez, repassa os objetos para o MAS (C). Em seguida, o MAS executa as políticas de adaptação e retorna a resposta adaptada para a aplicação no dispositivo móvel (D).

6.3. Arquitetura do Mobile Adapter

Na Figura 19, é apresentado uma visão geral das classes que compõem o *framework* Mobile Adapter. O *framework* é dividido em três partes: Núcleo, Manipuladores de Perfis e *Beans*.

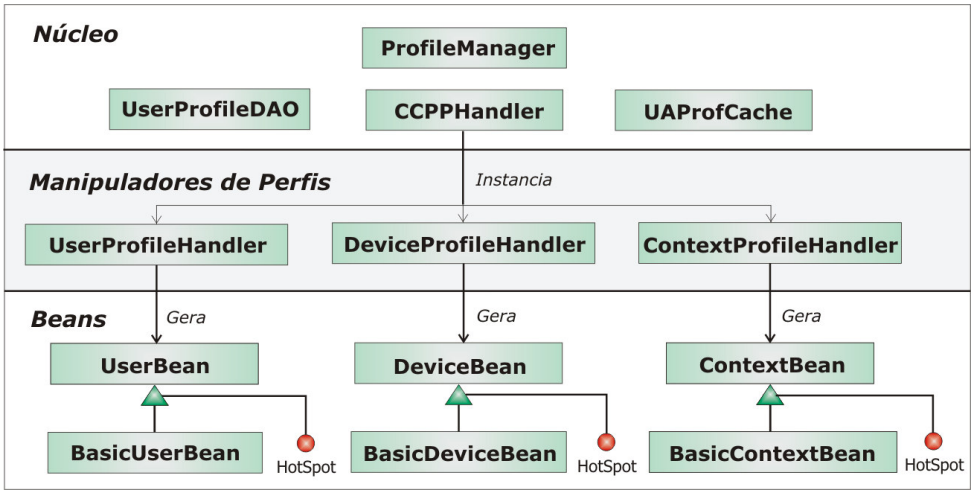


Figura 19 - Visão geral do *framework* Mobile Adapter

Na parte Núcleo, encontram-se as principais classes do *framework*: *ProfileManager*, *CCPPHandler*, *UserProfileDAO* e *UAProfCache.* A classe *ProfileManager,* como descrito anteriormente, é a principal classe do *framework* e deve ser instanciada pelo MAS para a sua utilização. A classe *UserProfileDAO* é a responsável por armazenar e recuperar os perfis dos usuários descritos em CC/PP que são adicionados pelo MAS usando o método *addUserProfile* da classe *ProfileManager*.

A classe *UAProfCache* é a responsável por gerenciar o *cache* dos documentos UAProf. O *cache* utilizado consiste em uma tabela de dispersão (i.e., *hashtable*) cuja chave é a URI e o valor é o documento UAProf. As tabelas de dispersão são eficientes para esse tipo de operação, pois possuem complexidade média O(1) para acesso [118]. O uso de

cache é necessário devido aos tamanhos que os documentos UAProf podem ter, em geral acima de 14KB. Dessa forma, se o *cache* não for utilizado, *downloads* repetidos do UAProf causam atraso no processo de adaptação.

Na parte Manipuladores de Perfis, encontram-se as classes: *UserProfileHandler*, *ContextProfileHandler* e *DeviceProfileHandler.* Essas classes são responsáveis por mapear, para instâncias de objetos da parte Beans do *framework,* os parâmetros contidos, respectivamente, nos documentos CC/PP do usuário, CC/PP enviado pelo MobAC e UAProf do dispositivo.

Para que o engenheiro de software possa utilizar os parâmetros que desejar nos documentos CC/PP relacionados ao contexto e ao usuário, as classes *UserProfileHandler* e *ContextProfileHandler* criam os objetos *Beans* utilizando o conceito de *Reflection* que é a capacidade de em tempo de execução descobrir métodos e atributos de um objeto ou de uma classe [100]. Dessa forma, esses *Handlers* descobrem o nome de cada atributo contido nos *Beans* e procuram nos documentos XMLs, *tags* com o mesmo nome. Caso encontrem, esses *Handlers* utilizam os métodos *sets* para atribuir aos *Beans* os valores contidos no documento XML.

Para facilitar o uso do Mobile Adapter, o *framework* já disponibiliza *Beans* com atributos gerais, que são descritos a seguir:

- *BasicUserBean* – contém o atributo *language* que define a língua de preferência do usuário;
- *BasicContextBean* – contém os atributos *battery, fMem e fStorMem* que correspondem à quantidade de bateria disponível, memória livre para a execução e o armazenamento no dispositivo.
- *BasicDeviceBean* - contém os atributos *displayColor, displayWidth, displayHeight, displayNumberOfColors, displayCharsLines, displayCharsColumns* e *playSound* referentes às propriedades do dispositivo.

Para monitorar novos atributos relacionados ao contexto e ao usuário, o engenheiro de software deve utilizar os *Hot Spots* disponibilizados no *framework* (ver Figura 19).Vale ressaltar que a instanciação do *BasicDeviceBean* pelo *DeviceProfileHandler* ocorre sem o uso de *Reflection*, pois os dados estáticos do dispositivo são retirados de um documento UAprof com *tags* pré-estabelecidas. Assim, caso o engenheiro deseje monitorar novos

atributos do dispositivo, ele deve, além de criar uma subclasse de *DeviceBean*, modificar o código do *DeviceProfileHandler*.

O *framework* Mobile Adapter foi implementado em Java utilizando a API livre *Common Bean Utils* para realizar as operações de *Reflection.* E a classe Java *HashTable* foi utilizada na implementação do *cache*.

6.4. *Framework* Requisitor

Como mencionado no Capítulo 3, as plataformas de programação para dispositivos móveis, mesmo as baseadas em Java, possuem diferentes APIs para conexão de dados. Esta característica dificulta a construção de aplicações multi-plataformas que necessitam trocar informações com servidores ou outras aplicações remotas.

Nesse contexto, apresentamos um *framework* multi-plataforma, intitulado de Requisitor, para permitir que aplicações em dispositivos móveis possam realizar comunicação de dados do tipo *Request-Response*. Este *framework* permite a comunicação transparente de dados e foi projetado para suportar troca de dados através de *socket* TCP-IP e do protocolo HTTP.

Na analise de domínio do *framework*, foi realizada uma investigação de como pode ser estabelecida a comunicação de dados nas plataformas de programação J2ME MIDP, Superwaba e J2ME Personal Profile, uma outra versão de J2ME baseada na configuração CDC [43]. A partir dessa investigação, foram identificadas as principais classes de cada plataforma que realizam a comunicação de dados e uma generalização dos seus métodos foi criada.

Como mencionado no Capítulo 3, padrões de projeto podem ser utilizados na fase de projeto de um *framework*. No caso do Requisitor, os padrões de software *Foward-Receiver* [41], *Strategy* [41] e *Factory Method* [41] foram utilizados para permitir a transparência dos protocolos de comunicação e das APIs de conexão.

O padrão *Forward-Receiver* introduz um modelo de comunicação ponto-a-ponto (i.e., *peer-to-peer*), que permite a separação dos remetentes (*Forwarders*) e receptores (*Receivers*), dos mecanismos de comunicação utilizados para a entrega e recebimento das informações, provendo, assim, uma comunicação interprocesso transparente. O padrão

Forwarder-Receiver é utilizado como base para a construção do *framework* Requisitor. O *framework* utiliza a mesma abordagem de *Fowarders*, *Receivers* e *Peers*.

O padrão *Strategy* é utilizado quando se quer manter a independência do algoritmo utilizado para realizar uma determinada tarefa. Sendo assim, ele define uma família de algoritmos e os encapsula, permitindo ao objeto utilizador realizar a permuta entre os algoritmos sempre que esse achar conveniente. O padrão Strategy foi utilizado na definição do *framework* para montar a hierarquia das classes do *Forwarder*, do *Receiver* e do *Entry*, nas quais as estratégias de envio, de recebimento e de mapeamento das entradas ficam transparentes para o *Peer*.

Além desses padrões, o *framework* utilizou o padrão *Factory Method* para geração dos *Forwarders* e *Receivers* dependentes do protocolo de comunicação que o *Peer* utilizará para se comunicar com outro *Peer* (e.g., *ForwarderHTTP*, *ReceiverSocketTcpIp*).

Na Figura 20, é apresentada uma visão geral do *framework* que é dividido em três partes principais: *Requisitor Interface*, *Requisitor Factory* e *Requisitor Communication Protocols*.

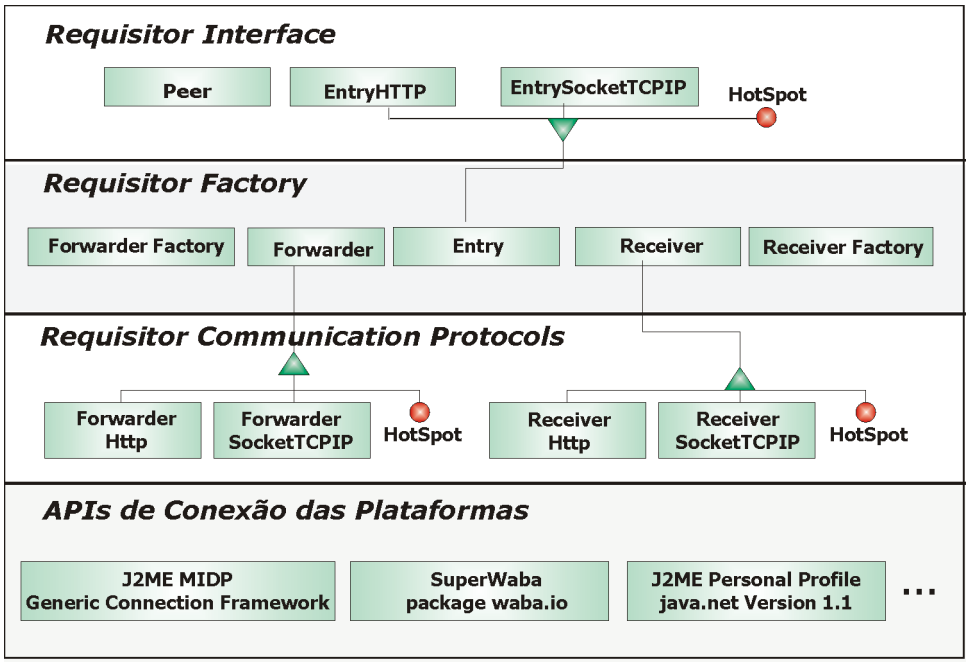


Figura 20 - *Framework* Requisitor

A parte *Requisitor Interface* disponibiliza para as aplicações as classes *Peer* e as classes que herdam de *Entry*. A classe *Peer* representa a interface do *framework* com a

aplicação. Nessa classe são disponibilizados os métodos para envio e recebimento de dados. Ela também possui uma tabela de dispersão (i.e., *hashtable*) na qual são armazenados os nomes das partes comunicantes e seus respectivos objetos das classes que herdam de *Entry*. Em um objeto *Entry* está contido o tipo de protocolo que deve ser utilizado para a comunicação, assim como os dados específicos dessa comunicação (e.g., URI na classe *EntryHTTP*; endereço IP e porta na classe *EntrySocketTCPIP*). A aplicação utiliza o método público *addEntry*, pertencente ao *Peer*, para adicionar novas partes comunicantes e seus respectivos dados específicos de comunicação.

Para que a aplicação possa enviar dados utilizando a classe *Peer,* basta informar a mensagem e o nome dado ao outro *Peer* comunicante. E para receber os dados, a aplicação deve informar o nome do outro *Peer* comunicante. O *Peer* disponibiliza dois tipos de envio: em cadeia de *bytes* e em cadeia de caracteres (i.e., *String*). Além disso, o *Peer* possui três formas de recebimento de dados: em cadeia de *bytes*, em cadeia de caracteres e um objeto da classe *ReceiverBytePerByte* que permite manipular *byte* a *byte* à medida que os dados são recebidos.

A parte *Requisitor Factory* do *framework* é responsável por criar o *Forwarder* e o *Receiver* específicos para estabelecimento da comunicação. Na parte *Requisitor Communication Protocols*, as classes específicas de cada protocolo, como por exemplo, o *ForwarderHTTP*, são associadas a cada API de comunicação da plataforma de programação na qual o *framework* é implementado. Assim, para portarmos o *framework* para outra plataforma de programação é necessário apenas reimplementar partes das classes da terceira parte do *framework*. Caso o engenheiro de software deseje adicionar um novo protocolo de comunicação (e.g., envio por SMS - *Short Message Service*), ele deve utilizar os *Hot Spots* para criar novas classes que herdam de *Forwarder*, *Receiver* e *Entry*.

O *framework* Requisitor foi implementado em J2ME MIDP 1.0 e 2.0, J2ME Personal Profile e Superwaba. Como mencionado anteriormente, estas implementações são usadas pelo MobAC para enviar o documento CC/PP para um MAS que utiliza o Mobile Adapter. Contudo, vale ressaltar que o *framework* Requisitor pode ser utilizado sem o uso do MobAC para envio e recebimento de dados. Na Figura 21, é ilustrado um trecho do código de uma aplicação J2ME MIDP 1.0 que utiliza o *framework* Requisitor para realizar o *download* de uma imagem de um servidor *Web*.

```
EntryHTTP entryImagem=new EntryHTTP("GET",
   "http://www.lia.ufc.br/~windson/imagens/ ");
    Peer peer1=new Peer();
    peer1.addEntry("ServImagem",entryImagem);
    //...
    //Na parte que controla os eventos dos Botões
    peer1.sendString("logomidp.png ","ServImagem");
    byte[] imagem=peer1.readBytes("ServImagem");
    if(imagem!=null){
    Image imagemAux=Image.createImage(imagem,0,imagem.length);
    //Imagem criada
     }
```

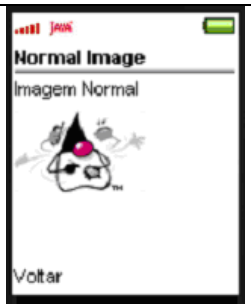


Figura 21 – Código que utiliza o *framework* Requisitor

6.5. Arquitetura do MobAC

Como mencionado anteriormente, o MobAC é o *framework* responsável por capturar as variáveis dinâmicas do contexto e montar o documento CC/PP que é enviado juntamente com as requisições da aplicação para o MAS.

Para permitir a flexibilidade do MobAC e a sua implementação em diferentes plataformas, ele é dividido em quatro processos básicos: a interface com a aplicação, a monitoração, a geração do documento CC/PP e o envio da requisição.

Entre esses quatro processos, o envio de requisição e a monitoração são as partes mais complexas a serem portadas para várias plataformas. Para facilitar esse procedimento, toda parte de comunicação do MobAC é realizada utilizando o *framework* Requisitor.

Na Figura 22, são apresentadas as classes que compõem o *framework* e seu relacionamento com o *framework* Requisitor. A classe *MobAC* é a classe a ser utilizada pela aplicação e contém métodos para envio e recebimento de vetores de *bytes* e cadeias de caracteres (i.e., *Strings*).

A classe abstrata *CCPPAssembler* é a responsável por montar o documento CC/PP que é adicionado à requisição. Na instanciação da classe *MobAC*, o engenheiro de software deve passar como parâmetro uma instância de uma subclasse do *CCPPAssembler* que implemente o método que monta o conteúdo do documento CC/PP.

O *framework* já disponibiliza uma classe que implementa este método, a *CCPPBasicAssembler*. Esta classe monta um documento CC/PP com as seguintes propriedades: o ID do usuário, a URI do UAProf, a quantidade de energia disponível na bateria, memória livre de execução e de armazenamento do dispositivo. O ID do usuário é

passado pela aplicação ao realizar uma requisição, a URI do documento UAProf que descreve o dispositivo é capturada pela classe *UAProfGetter* e os outros parâmetros são obtidos a partir de uma instância da classe *Monitor*.

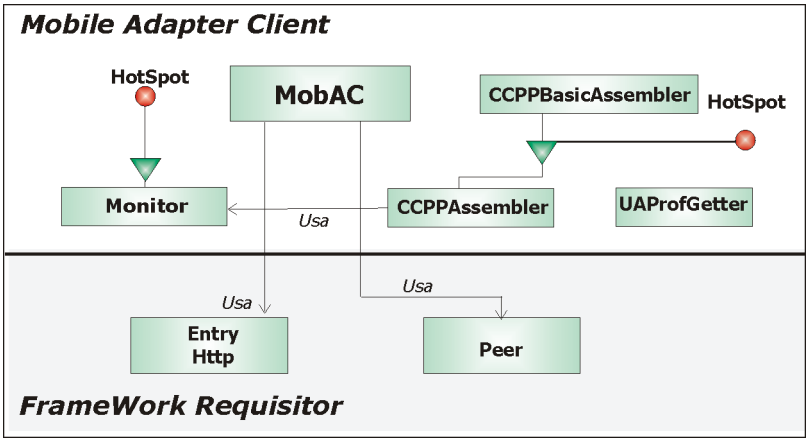


Figura 22 - *Framework* MobAC e a interação com *framework* Requisitor

Vale ressaltar, que os atributos do contexto contidos no documento CC/PP gerado pela classe *CCPPBasicAssembler* correspondem aos atributos do *bean ContextBasicBean* disponibilizados pelo *framework* Mobile Adapter. Na Figura 23, é apresentada uma parte do documento CC/PP gerado pelo framework MobAC durante uma requisição no P900 utilizando a classe *CCPPBasicAssembler*.

Entretanto, caso o engenheiro de software deseje monitorar e enviar outros parâmetros sobre o contexto de execução, ele deve criar o seu próprio *CCPPAssembler* e seu *Monitor* utilizando os *Hot Spots* disponibilizados pelo *framework* (Figura 22), assim como criar uma nova classe que herda de *ContextBean* no *framework* Mobile Adapter conforme mencionado na seção 6.2.

```
<ccpp:component>
 <rdf:Description>
  <great:uaProfURI>wap.sonyericsson.com/UAprof/P900R101.xml</great:uaProfURI>
  <great:userID>UserDefault</great:userID>
  <great:fMem>465812</great:fMem>
  <great:fStorMem>145800</great:fStorMem>
  <great:battery>0.8</great:battery>
 </rdf:Description>
</ccpp:component>
```

Figura 23 - Documento CC/PP gerado durante uma requisição no dispositivo P900

O MobAC foi implementado em J2ME MIDP e Superwaba, o que facilita a integração com os códigos gerados pela UIG. Como essas plataformas são baseadas em

Java e o processo de requisição utiliza o *framework* multi-plataforma Requisitor, apenas o processo de monitoração foi implementado em cada plataforma.

6.6. Conclusão

Neste Capítulo foi apresentado o Mobile Adapter e MobAC que, em conjunto, propiciam uma forma eficiente de captura, manipulação e gerenciamento dos perfis do dispositivo, do usuário e do contexto em que executam. Como mencionado no Capítulo 2, a ausência destes tipos de mecanismos dificulta a construção de sistemas reais que executam adaptação de conteúdo.

Além disso, o MobAC foi construído para ser facilmente integrável ao código gerado pela ferramenta UIG apresentada no Capítulo 5. Dessa forma, o ambiente proposto pode ser facilmente utilizado tanto para construir aplicações adaptáveis à descrição e multi-plataformas, quanto para construir aplicações que acessam MAS que adaptam conteúdo.

No capítulo, também foi apresentado o *framework* multi-plataforma Requisitor que permite o estabelecimento de comunicação de dados entre servidores remotos e as aplicações nos dispositivos móveis. Esse *framework* foi utilizado no MobAC para facilitar sua implementação em várias plataformas.

No próximo capítulo, é apresentado um estudo de caso que ilustra como os *frameworks* e a ferramenta propostos nessa dissertação podem ser integrados para construir aplicações complexas para dispositivos móveis.

7. Estudo de Caso

Neste Capítulo é apresentado um estudo de caso que ilustra como o ambiente proposto nesta dissertação pode ser utilizado no desenvolvimento de aplicações adaptativas e multi-plataformas para dispositivos móveis. O estudo de caso consiste em um sistema para a visualização e a postagem de imagens fotográficas e comentários. O sistema, chamado de M-Flog, é composto de uma servidor *Web* e de um cliente para DMs.

Na Seção 7.1, são apresentados os requisitos e a arquitetura geral do M-*Flog*. Na Seção 7.2, é detalhado o processo de construção da aplicação para DMs do M-*Flog*. Na Seção 7.3, é descrita a construção do MAS Adaptativo e dos pacotes de adaptação de imagem e de texto. E, por fim, na Seção 7.4 é a apresentado uma avaliação da eficiência do uso de adaptação de imagens.

7.1. M-Flog

Uma das aplicações de entretenimento que tem se tornado popular na Internet são os *Flogs.* Um *Flog* é um diário de fotos na forma de um *site* na Internet. Esses sistemas *Web* permitem ao usuário adicionar imagens fotográficas a cada dia e colocar comentários sobre elas. Além disso, outros usuários podem adicionar comentários nos *Flogs* dos amigos.

Já existem *Flogs* [88] que permitem o envio, através de MMS (*Multimedia Message Service*), de imagens retiradas com câmera de um celular. Contudo, o objetivo do M-Flog é construir um *Flog* acessível por dispositivos móveis, tanto para visualizar imagens e comentários, quanto para o usuário adicionar comentários aos *Flogs* dos seus amigos.

Para que o M-Flog possa ser acessado por uma grande quantidade de usuários e o para que sistema tenha um tempo satisfatório de resposta às requisições, dois requisitos não funcionais devem ser satisfeitos e são listados a seguir:

- **Portabilidade da aplicação cliente**. Diferentes usuários irão utilizar o M-Flog em dispositivos com características diferentes entre si. Desta forma, a aplicação deve ser capaz de ser executada em uma grande diversidade de dispositivos e plataformas de programação.
- **Adaptação das imagens e dos comentários.** A adaptação das imagens é necessária devido à dificuldade que um dispositivo móvel possa vir a ter para conseguir acessar uma imagem fotográfica da *Web*, por causa do tamanho ou do formato da mesma. A adaptação

dos comentários é importante para possibilitar que mesmo em dispositivos com pequenos *displays,* um usuário possa ter acesso a um comentário ou a uma parte dele.

Portanto, para satisfazer esses requisitos não funcionais e diminuir o tempo e o custo de construção da aplicação, foi utilizado o ambiente proposto nesta dissertação. Na Figura 24, é ilustrada uma visão geral do M-Flog e seus relacionamentos com os *frameworks* propostos na dissertação. Para construir a aplicação do DM, a ferramenta UIG foi utilizada para gerar as interfaces, os controladores e os *beans* dos formulários. Desta forma, as interfaces da aplicação utilizam o *framework* XFormUI e executam em três plataformas de programação.

Para realizar a comunicação com o MAS do M-Flog, códigos foram escritos nos controladores dos formulários. Quando o conteúdo a ser requisitado não necessita de adaptação, esses códigos utilizam o *framework* Requisitor para enviar a requisição (e.g., na checagem de *login* e senha). Já nos casos em que as requisições podem ter respostas adaptadas, o MobAC é utilizado (e.g., requisição de imagens).

O MAS do M-Flog tanto pode ser acessado por dispositivos móveis quanto por um *browser* de um PC. Para sua construção, o *framework* Mobile Adapter foi utilizado. Assim, o M-Flog pode recuperar facilmente os perfis do usuário, do dispositivo e do contexto, permitindo a realização da adaptação dos textos e das imagens.

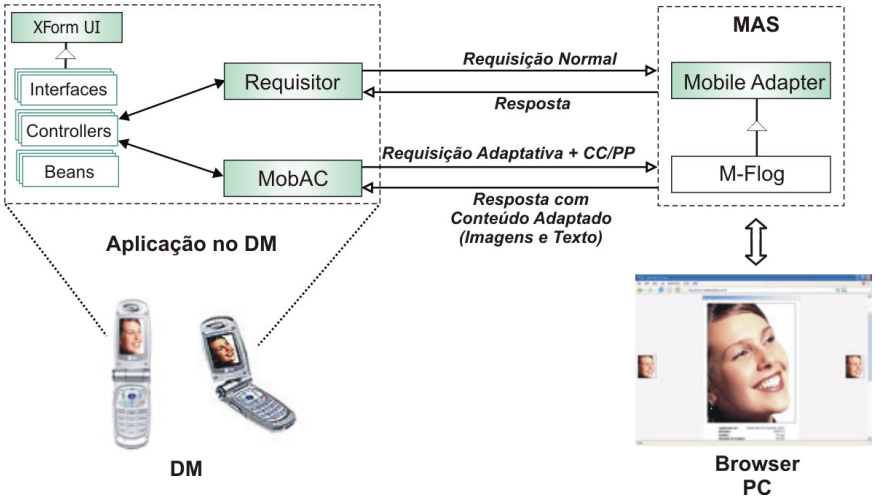


Figura 24 - Visão geral do M-Flog

7.2. Aplicação no DM

Para a construção da aplicação no dispositivo móvel, os formulários foram especificados em XForms, CSS e XML Schema utilizando o Eclipse e o *plugin* para edição de documentos XML, o XMLBuddy [69] . Os formulários criados são usados para entrada de dados de *login*, visualizações de imagens e comentários, escolha de opções, busca de imagens, mudança das preferências do usuário e adição de novos comentários. Na Figura 25, são apresentadas as telas da aplicação de *login*, de preferências do usuário e de leitura de comentários, em XForms executadas no *browser* X-Smiles [15].

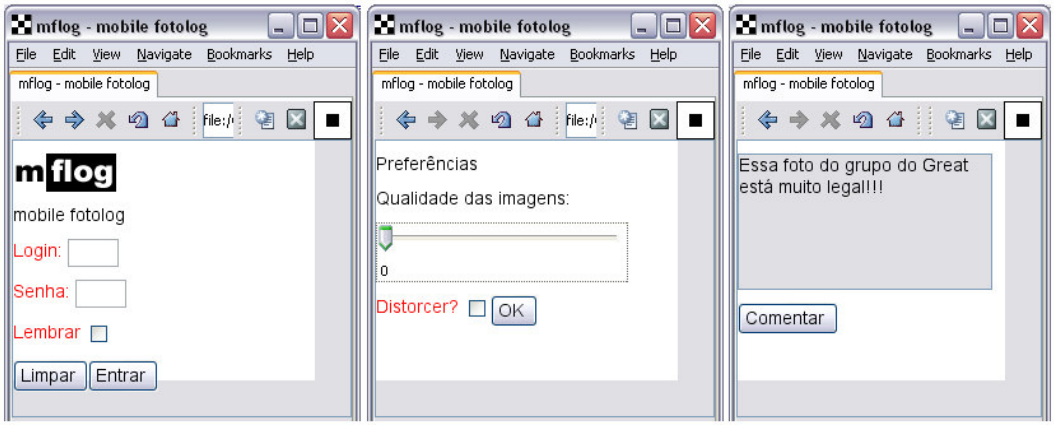


Figura 25 - Telas da aplicação no XSmiles

Na construção do *layout* dos formulários da aplicação, os atributos de cores do CSS foram utilizados para modificar as cores dos *labels* das interfaces e o atributo *align* e a *tag* *</xhtml:p>* foram usados para determinar o posicionamento dos componentes nos formulários. O XML Schema e o *Bind* do XForms foram usados para determinar as restrições para entrada de dados em três formulários, listados a seguir:

- **Formulário de *login***. Neste formulário, o usuário preenche *login* e senha para obter autorização para entrar no sistema. Um tipo simples do XML Schema foi associado ao campo senha para que a mesma contenha de 3 a 10 dígitos. Para o campo *login* foi criado um *Bind* de "*required=true*" o qual define o campo como obrigatório.

- **Formulário de busca de imagens**. Neste formulário, o usuário informa um período para serem buscadas as imagens que ele deseja visualizar. Assim, o tipo básico *xsd:date* do XML Schema foi associado aos *Inputs,* o que permitiu a seleção de datas para realizar a busca.

❑ **Formulário de exibição de comentário.** Neste formulário, é exibido o comentário adicionado a uma imagem por um amigo do usuário. Como o usuário não pode modificar esse texto, foi criado um *Bind* de "*readonly=true*" para o *TextArea*.

Na Tabela 11, podem ser vistos trechos do arquivo XML Schema utilizado nas telas de *login* e busca de imagens para aplicar as restrições na entrada dos dados. O CSS usado em todos os formulários para definir as cores dos *labels* também é apresentado nessa tabela.

Schema

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!-- Simple types -->
<xsd:element name="login" type="xsd:string"/>
<xsd:element name="data" type="xsd:date"/>

<!-- Restricted simple types -->
<xsd:simpleType name="senha">
   <xsd:restriction base="xsd:string">
        <xsd:maxLength value="10"/>
        <xsd:minLength value="3"/>
   </xsd:restriction>
</xsd:simpleType>
```

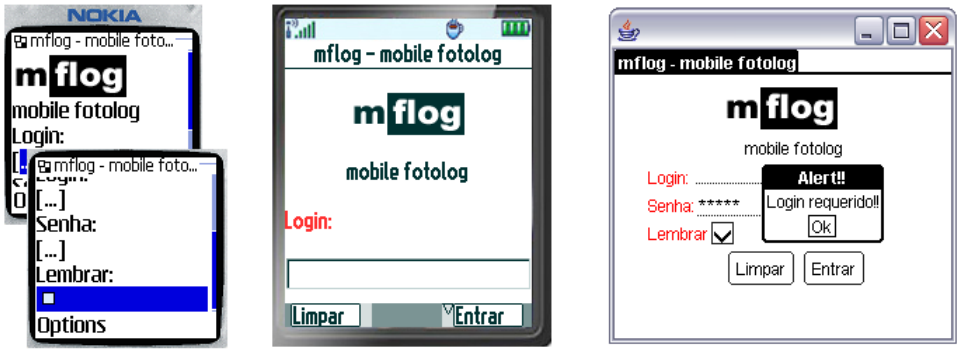
CSS

```
input {
        background-color: #FFFFFF;
        color: #FF0000;
}
secret {
        background-color: #FFFFFF;
        color: #FF0000;
}
selectboolean {
        background-color: #FFFFFF;
        color: #FF0000;
}
```

Tabela 11 - Códigos do XML Schema da tela de *login* e do CSS usado nos formulários

Após a escrita dos formulários, o *Manifest*.xml foi criado e os formulários foram gerados nas três plataformas de programação utilizando o *plugin* do Eclipse também provido pelo ambiente. O tempo máximo decorrido para a geração de código de todos formulários para as três plataformas foi de 9,52 segundos. Vale ressaltar que a UIG foi executada em uma máquina AMD Atlhon 2400Hz, 512 MB de RAM com Windows XP Pro.

Na Figura 26, é ilustrada a tela de *login* gerada pela UIG sendo executada nas três plataformas de programação. Na aplicação que executa em J2ME MIDP 1.0, pode ser notada a ausência de alinhamento centralizado da imagem e do *Output* se comparada à aplicação em J2ME MIDP 2.0. Além disso, apesar de o dispositivo ser colorido, não existem cores nos *labels* do *Input* e do *Secret*. Na aplicação em Superwaba, é exibida a execução do código gerado automaticamente, a partir do XML Schema e do *Bind*, para a validação da entrada de dados nos campos senha e *login*. Como o campo *login* não está preenchido, ao se acionar o *Trigger* "Entrar", a janela de alerta com a mensagem "Login requerido!!" é exibida.



Nokia 5100
MIDP 1.0

Motorola V600
MIDP 2.0

Palm OS
Applet SuperWaba

Figura 26 - Tela de *login* nas três plataformas

Cinco formulários gerados pela ferramenta têm dados iniciais dinâmicos que se modificam a cada execução da aplicação. Desta forma, esses formulários são instanciados através do construtor gerado pela UIG que permite a passagem de atributos de inicialização usando um objeto *Bean* do formulário. Na Tabela 12, é ilustrado o código em XForm do formulário de preferências do usuário, assim como o código correspondente em XFormUI para o construtor parametrizado.

Código em XForms	Código em XFormUI

`<xforms:output ref="/data/titulo"`
`id="outTitulo" align="center">`
`<xforms:label/>`
`</xforms:output>`
`<!-- Output -->`
`<xforms:output ref="/data/output"`
`id="outQualidade" align="center">`
`<xforms:label/>`
`</xforms:output>`
`<!-- SelectBoolean -->`
`<xforms:selectboolean`
`ref="/data/selectboolean"`
`id="sbDistorcer" align="center">`
`<xforms:label>Distorcer?</xforms:label>`
`</xforms:selectboolean>`
`<!-- Trigger -->`
`<xforms:submit id="trgOK">`
`<xforms:label>OK</xforms:label>`
`</xforms:submit>`
	`FormPreferencias(MainXForm mainXForm,`
	`FormPreferenciasBean xformBean){`
	`super("mflog - mobile fotolog",mainXForm);`
	`// Output(s)`
	`this.outTitulo =`
	`new Output("","+xformBean.titulo);`
	`this.outQualidade =`
	`new Output("","+xformBean.output);`
	`// SelectBoolean(s)`
	`this.sbDistorcer =`
	`new SelectBoolean("Distorcer?"," ");`
	`this.sbDistorcer.setValue("`
	`+xformBean.selectboolean);`
	`// Trigger(s)`
	`this.trgOK = new Trigger("OK");`
	`} // End FormPreferencias`

Tabela 12 – Código do formulário em XForms e o correspondente em XFormUI

Para realizar a comunicação com o MAS do M-Flog e receber as respostas não adaptadas, códigos que utilizam o *framework* Requisitor para enviar e receber informações foram escritos nos controladores dos formulários.

Essas informações são enviadas através de requisições *GET* do protocolo HTTP. O MAS responde às requisições e no caso de a resposta ser uma lista de *Strings* (e.g., lista dos *flogs* dos amigos) , as informações são enviadas para o DM em uma única *String* com caracteres especiais de separação.

No caso das requisições adaptativas de texto e imagem, códigos que utilizam o *framework* MobAC foram escritos nos controladores dos formulários. Durante uma requisição, a implementação do *framework* MobAC adiciona um parâmetro que contém o documento CC/PP com o ID do usuário, a URI do documento UAProf e as seguintes variáveis de contexto monitoradas: memória livre do dispositivo e o tipo de imagem suportado (e.g., JPG, GIF e PNG). Esse documento é processado pelo MAS através do Mobile Adapter o que permite a realização das adaptações. Esse processo é detalhado na próxima seção.

Todos esses códigos escritos nos controladores executam nas três plataformas de programação, já que elas são baseadas em Java e os códigos utilizam os *frameworks* implementados nas três plataformas. Desta forma, todas as funcionalidades de validação dos dados, troca de formulários e comunicação com o MAS da aplicação foram escritas apenas uma vez pelo engenheiro de software.

Na Figura 27, é apresentado um fluxo de execução da aplicação no J2ME MIDP 2.0, iniciando na tela de *login* e senha e ilustrando as telas de listagem de amigos, busca de imagens, a imagem adaptada, visualização de comentários e preferências do usuário.

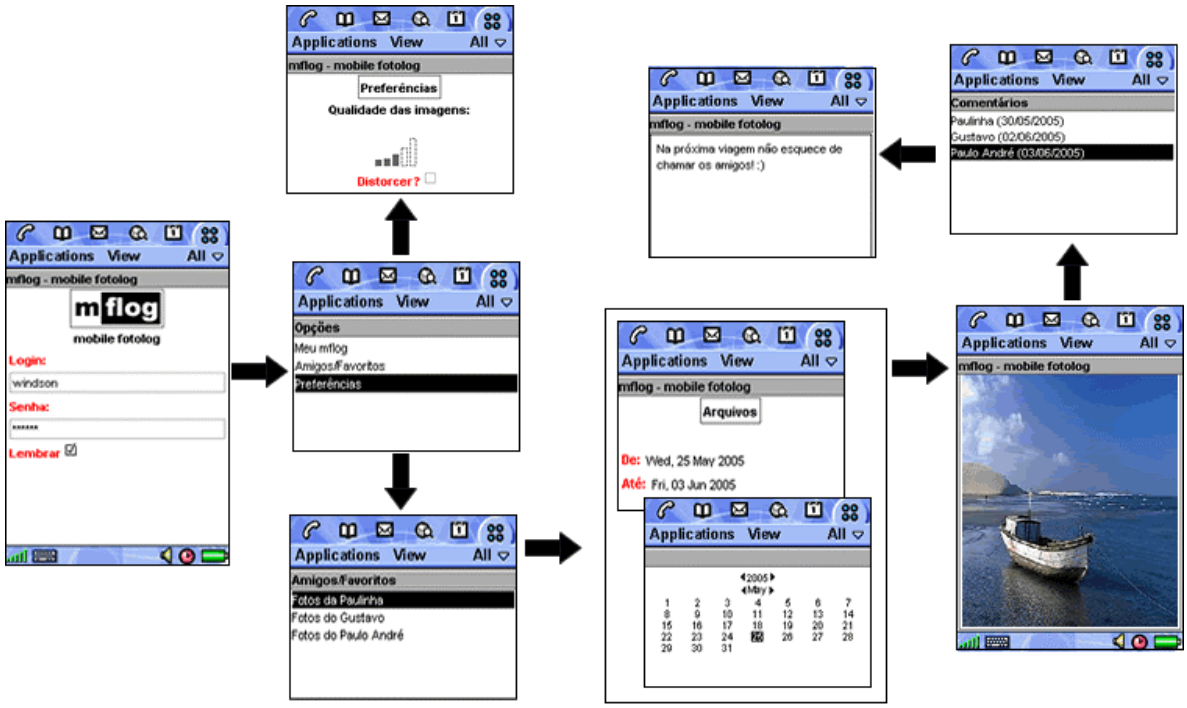


Figura 27 - Um fluxo de execução da aplicação cliente do M-Flog

7.3. MAS Adaptativo

Como mencionado anteriormente, o *Mobile Application Server* do M-Flog responde tanto a requisições normais quanto a requisições que exigem respostas adaptativas. Dois recursos do sistema são adaptados: imagens e textos dos comentários.

A adaptação de imagens no M-Flog é necessária, pois caso um usuário utilize um navegador tradicional de um dispositivo móvel para acessar o sistema, as imagens fotográficas originais são enviadas por completo para o dispositivo. O envio dessas imagens ocupa desnecessariamente a banda de comunicação, já que o dispositivo tem que converter a imagem recebida em dimensões e cores adequadas para a sua exibição. Essa conversão exige memória e processamento do dispositivo que muitas vezes pode não conseguir realizá-la em um tempo aceitável para o usuário. Além disso, nos casos em que o usuário é tarifado pela quantidade de pacotes recebidos (e.g., GPRS, EDGE), ele paga pelo tamanho inicial da imagem.

Outro problema relacionado ao acesso de imagens pelas aplicações, decorre da diferença nos formatos de imagens suportados pelas plataformas de programação. Como

mencionado no Capítulo 3, a plataforma J2ME MIDP 1.0 suporta apenas PNG, o que exige uma mudança no formato das imagens acessadas por aplicações escritas nessa plataforma.

Para oferecer as informações sobre o usuário, o dispositivo e o contexto em que ambos executam, o M-Flog foi construído utilizando o *framework* Mobile Adapter. Ao receber uma requisição de uma imagem ou de um comentário, o M-Flog repassa para o ProfileManager o documento CC/PP enviado pelo MobAC. O ProfileManager, como descrito no Capítulo 6, dispara os processos de descoberta do perfil do usuário e do dispositivo e de mapeamento dos perfis para os objetos *Beans*. De posse desses objetos, o M-Flog realiza a adaptação.

Os parâmetros considerados pelo MAS dos perfis do usuário, do dispositivo e do contexto para realizar as adaptações são os seguintes:

- **Perfil do Usuário**: qualidade desejada e realização de distorção (i.e., *stretch*). O parâmetro de qualidade determina qual a qualidade da imagem desejada pelo usuário, já que em alguns formatos de imagens, como JPEG e PNG, é possível aplicar filtros para redução da qualidade e conseqüentemente do tamanho da imagem. O parâmetro de realização de distorção informa se o usuário deseja ou não realizar a operação de *stretch* de uma imagem. Uma operação de *stretch* é uma mudança nas dimensões da imagem (i.e., redução ou aumento), que não preserva as proporções iniciais da imagem, causando uma distorção.
- **Perfil do Dispositivo**: largura e altura do *display*, número de linhas e de colunas de caracteres e número de cores ou tons de cinza do dispositivo.
- **Perfil do Contexto**: memória livre do dispositivo e tipo de imagem suportada

Como pode ser visto, foram acrescentados dois parâmetros ao Perfil do Usuário que não são suportados de início pelo *framework* Mobile Adapter. Essa adição de parâmetros foi realizada através de uma nova classe, a UserMFlogBean que estende a classe BasicUserBean provida pelo *framework.* Desta forma, as instâncias da classe UserMFlogBean são povoadas automaticamente pelo Mobile Adapter através do processo de *Reflection* explicitado no Capítulo 6. Vale ressaltar que estes parâmetros do perfil do usuário são preenchidos quando este se cadastra ao sistema e podem ser alterados, inclusive utilizando o cliente para DMs do M-Flog.

Para atender as requisições do cliente do M-Flog, o MAS foi implementado em J2EE [51] utilizando o *framework* Maverick. O Maverick é uma implementação do modelo MVC (i.e., *Model-View-Controller*) para facilitar o desenvolvimento de aplicações Web. Assim, para cada tipo de requisição foi associada uma classe *Servlet* controladora. Para armazenar os comentários e os relacionamentos entre os usuários, o banco de dados livre *MySql* foi utilizado.

Na Figura 28, são apresentadas as classes do M-Flog responsáveis por participar do processo de adaptação e o seus relacionamentos com as classes existentes no Mobile Adapter. As classes ComController e o ImageController são os *Servlets* que recebem as requisições adaptativas provenientes dos clientes dos DMs. As classes TextAdapter e Img Adapter são responsáveis por realizar a adaptação e são descritas nas próximas sub-seções.

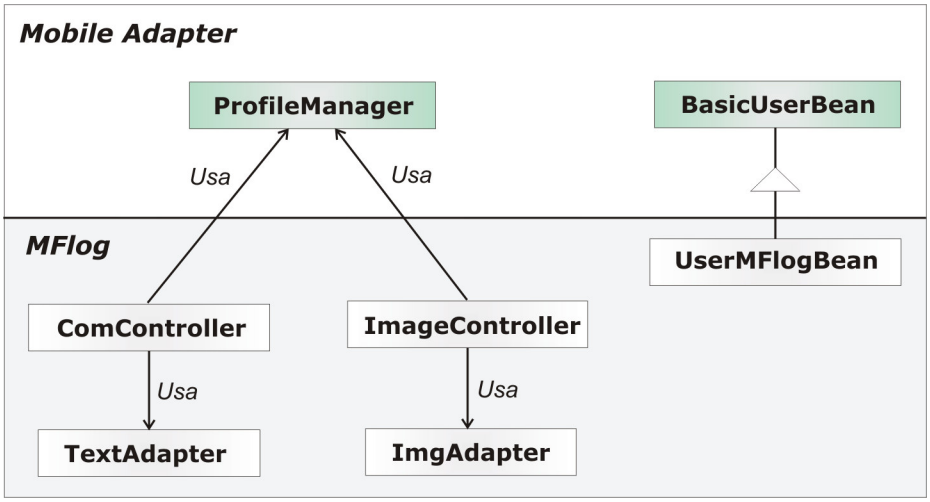


Figura 28 – Classes que participam dos processos de adaptação do M-Flog

7.3.1. Adaptação de Textos

Para realizar a adaptação de texto dois parâmetros do perfil do dispositivo são usados: o número de linhas e o número colunas de caracteres. Com esses dois parâmetros é calculado o número máximo de caracteres exibíveis na tela do dispositivo. Esse número calculado é utilizado para classificar o dispositivo em três categorias: **pequeno**, **médio** e **grande**. O dispositivo é classificado como **pequeno** quando o número de caracteres é menor que 360 (e.g., celular), **médio** quando o número de caracteres está entre 360 e 1600 (e.g., Smartphones) e **grande** quando tem mais de 1600 caracteres (e.g., Pocket PC).

A adaptação dos textos é realizada usando duas estratégias: *versões* e *cortes*. A adaptação baseada em versão consiste em escolher a mais adequada entre diferentes versões de um mesmo texto. Essa adaptação é semelhante à estratégia usada em [12] para documentos SMIL. A estratégia de corte consiste em recortar um texto para que este contenha uma quantidade de caracteres que não ultrapasse um número determinado.

Assim, quando a aplicação no DM faz uma requisição pedindo ao *Servlet* ComController um comentário, este retira o documento CC/PP da requisição e repassa pra ProfileManager. Depois de receber os *Beans* povoados pelo ProfileManager, o Com Controller instancia o TextAdapter. O TextAdapter classifica o dispositivo em **pequeno**, **médio** e **grande**. Baseado na classificação, o TextAdapter consulta a base de dados para checar se existe uma versão do comentário para essa classe de dispositivo. Caso exista uma versão, o comentário é retornado ao dispositivo. Caso uma versão não esteja disponível, o TextAdapter retira os primeiros 360 ou 1600 caracteres, dependendo da classificação do dispositivo, da versão existente do comentário. Vale ressaltar que o usuário, ao cadastrar um comentário, pode criar três versões para cada um deles ou tem a opção de escrever uma única versão.

Desta forma, a adaptação de textos apresentada consiste em uma **adaptação de conteúdo, em tempo de execução, a classes de dispositivos**. Na Figura 29, é apresentado o caminho lógico dessa adaptação utilizando a notação descrita em [89], na qual cada seta representa um tipo de adaptação ou uma ação realizada para executar a adaptação.

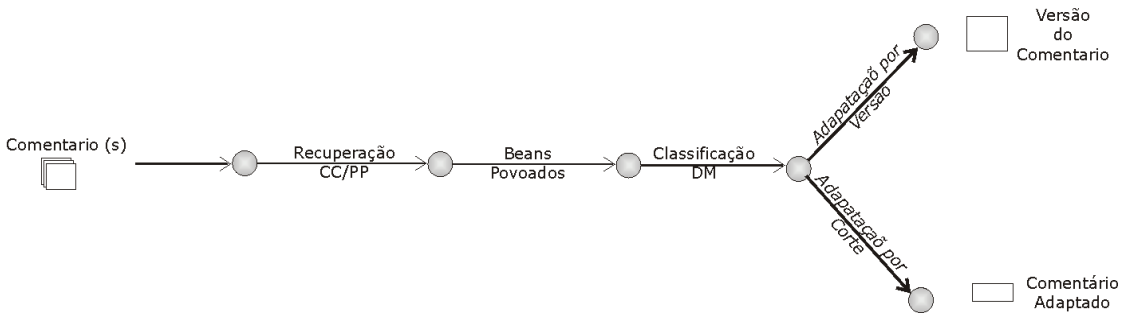


Figura 29 - Caminho Lógico da Adaptação de Textos

7.3.2. Adaptação de Imagens

Para realizar a adaptação de imagens são utilizados parâmetros dos perfis do dispositivo, do usuário e do contexto. Uma imagem pode ser adaptada de quatro modos, listados a seguir:

- **Redução**. O ImgAdapter pode diminuir a imagem para se adequar às dimensões do dispositivo. O parâmetro *stretch*, do perfil do usuário, é consultado para saber se a redução deve ou não manter as proporções iniciais da imagem.
- **Mudança de Cores**. O ImgAdapter modifica as cores da imagem para as cores suportadas pelo dispositivo. No caso do dispositivo não ser colorido, a imagem é convertida para os tons de cinza suportados pelo dispositivo.
- **Conversão do Formato**. O ImgAdapter altera o formato da imagem para se adequar ao formato suportado pela plataforma de programação do módulo cliente que requisitou a imagem.
- **Alteração da qualidade**. O ImgAdapter modifica os parâmetros de qualidade da imagem para reduzir mais ainda o seu tamanho.

O ImageController ao receber uma requisição de uma imagem de um cliente em um dispositivo móvel, retira o documento CC/PP da requisição e repassa para o ProfileManager. Depois de receber os *Beans* povoados pelo ProfileManager, o módulo ImgAdapter é acionado. O ImgAdapter inicialmente realiza a conversão entre os tipos de imagem, por exemplo, se a imagem requisitada é um JPG e o dispositivo executa J2ME MIDP 1.0, a imagem é convertida de JPG para PNG.

Após a conversão de formatos, o ImgAdapter monta uma chamada de sistema para o programa Convert que faz parte da coleção da ferramentas para manipulação de imagens ImageMagick [72]. Essa chamada de sistema contém o caminho do arquivo de imagem original, os parâmetros do usuário, de qualidade e distorção e os parâmetros do dispositivo de número de cores e de dimensões. Após a realização da conversão da imagem, o ImgAdapter recupera a imagem adaptada e checa se o tamanho da imagem não é maior que a quantidade de memória disponível no dispositivo quando este fez a requisição. Caso seja maior, uma pequena imagem com a frase: "Memória disponível insuficiente para a imagem" é retornada. Assim, é evitada uma possível falha na execução da aplicação no dispositivo móvel.

Desta forma, a adaptação de imagens apresentada consiste em uma **adaptação de conteúdo, em tempo de execução, ao dispositivo e ao contexto**. Na Figura 30, é apresentado o caminho lógico da adaptação de imagens.

Figura 30 - Caminho Lógico da Adaptação de Imagens

7.4. Análise de Eficiência

Para mostrar as vantagens da adaptação de imagens para dispositivos móveis e validar os pacotes de adaptação do M-Flog, testes de eficiência da adaptação foram realizados. A aplicação cliente do M-Flog foi executada em quatro dispositivos: Motorola A388, Nokia 6230, HP Ipaq h4100 e Sony Ericsson P900. Esses dispositivos apresentam diferenças quanto ao suporte a cores, dimensões do display e plataforma de programação testada. Além disso, a aplicação foi executada em cenários distintos para cada dispositivo: testes reais numa rede Wi-Fi e numa rede GPRS, e um teste simulado de uma conexão EDGE. Na Tabela 13, são apresentadas as tecnologias de conexão sem fio utilizadas nos cenários de simulação.

Dispositivos	Cor	Nº Cores	Display	Plataforma	Cenário
Motorola A388	Não	4 tons	260x200	J2ME MIDP 1.0	GPRS
Nokia 6230	Sim	65535	128x128	J2ME MIDP 1.0	GPRS
HP Ipaq h4100	Sim	65535	240X320	Superwaba	IEEE 802.11b
Sony Ericsson P900	Sim	65535	208x320	J2ME MIDP 2.0	Simulação EDGE

Tabela 13- Características dos dispositivos testados

Cinco imagens foram escolhidas para ilustrar o processo de adaptação, elas possuem tamanhos que variam de 123KB até 660KB. Essas imagens possuem diferentes dimensões e formatos (i.e., PNG e JPG) e ilustram as imagens que serão colocadas no ambiente real do M-Flog.

Dois tipos de adaptação de imagem foram executados baseados nos parâmetros do usuário: adaptação com qualidade alta e distorção; e adaptação com qualidade baixa e sem distorção. Duas métricas são utilizadas para quantificar a avaliação: o tempo de latência para realizar o *download* da imagem adaptada; e o tamanho da imagem adaptada. Cada teste foi realizado dez vezes em cada dispositivo e após o descarte do maior e do menor valor, a média dos tempos foi calculada. Os resultados são comparados ao tempo de latência para realizar o *download* da imagem original e o tamanho original da imagem. Na Figura 31, encontram-se as telas dos processos de adaptação de imagens simulados nos emuladores dos dispositivos P900 e Nokia 6230. No dispositivo P900 é apresentada a diferença no uso ou não de distorção na adaptação, nos dois casos as imagens têm tamanho maior do que a imagem adaptada para o dispositivo Nokia 6230.

Figura 31 - Adaptação de imagens nos emuladores dos dispositivos

Na Figura 32, são ilustrados os resultados dos testes no dispositivo Ipaq executando em uma rede real IEEE 802.11b. São apresentados dois gráficos, um referente ao tempo de latência de *download* e outro referente ao tamanho final das imagens. Podemos notar que nos dois gráficos, os dois tipos de adaptação apresentaram ganhos significativos. Com a imagem real, o tempo de latência de *download* variou de 150 a 400 segundos. Já com as imagens adaptadas, o tempo não passou de 35 segundos em todas as imagens. Além disso, as imagens foram reduzidas em seu tamanho para menos de 90 Kb. Desta forma, o tempo de latência e o tamanho final de cada imagem reduziram pelo menos 95%. A adaptação com menor qualidade e sem distorção apresentou ganhos ainda maiores.

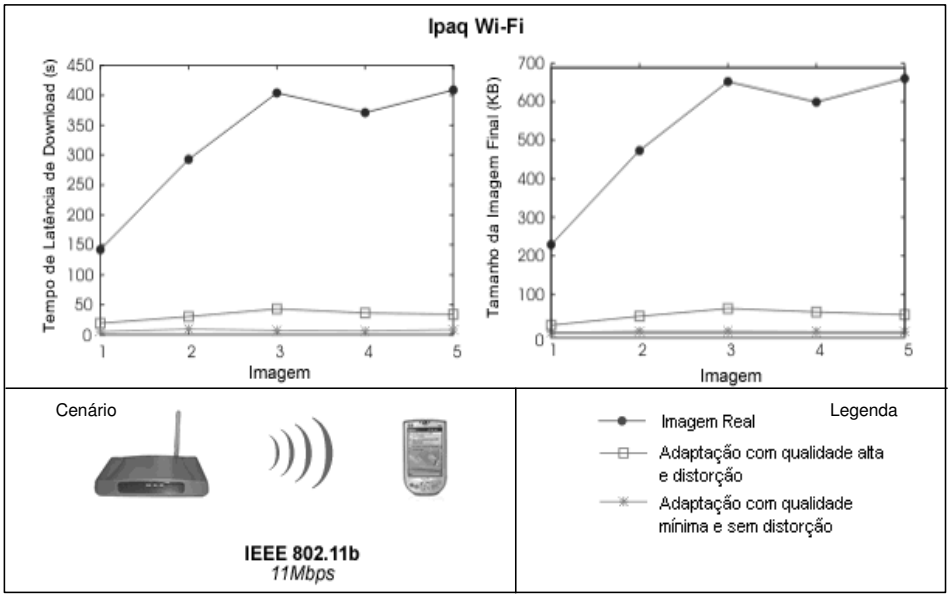


Figura 32 - Gráficos do Cenário IEEE 802.11b

Na Figura 33, são ilustrados os gráficos dos testes com o dispositivo A388 na rede GPRS. Esse dispositivo apresentou os maiores tempos de *downloads*, mesmo quando comparado ao dispositivo Nokia 6230 que também executou os testes no cenário GPRS. O tempo de latência da adaptação teve o valor máximo de 107,08 segundos, que pode ser considerado inaceitável para alguns usuários. Contudo, o tempo de *download* das imagens originais passava de dez minutos e causava *overflow* de memória no dispositivo. Assim, a adaptação das imagens apresentou também ganhos sensíveis, apesar de o tempo de espera ainda ser grande.

Na Figura 34, são ilustrados também os testes com o dispositivo P900 simulado no computador com uma conexão do tipo EDGE. Este dispositivo deve a velocidade da rede apresentou o menor tempo de latência para *download* da imagem original. Contudo, a adaptação ainda obteve ganho de mais de 80% no tempo e de 90% no tamanho.

O tempo de atraso somente da conversão das imagens variou de 1,44 a 4,57 segundos em todas as execuções. Esses tempos podem diminuir se o M-Flog for instalado em uma máquina com maior capacidade de processamento. O M-Flog foi testado em um Pentium III de 600 MHz com 128 MB de RAM. Com os resultados dos testes apresentados, podemos constatar que a adaptação realmente possibilitou que os dispositivos possam

acessar as imagens do portal M-Flog em tempo satisfatório e no caso do GPRS e do EDGE com custo bem menor para os usuários.

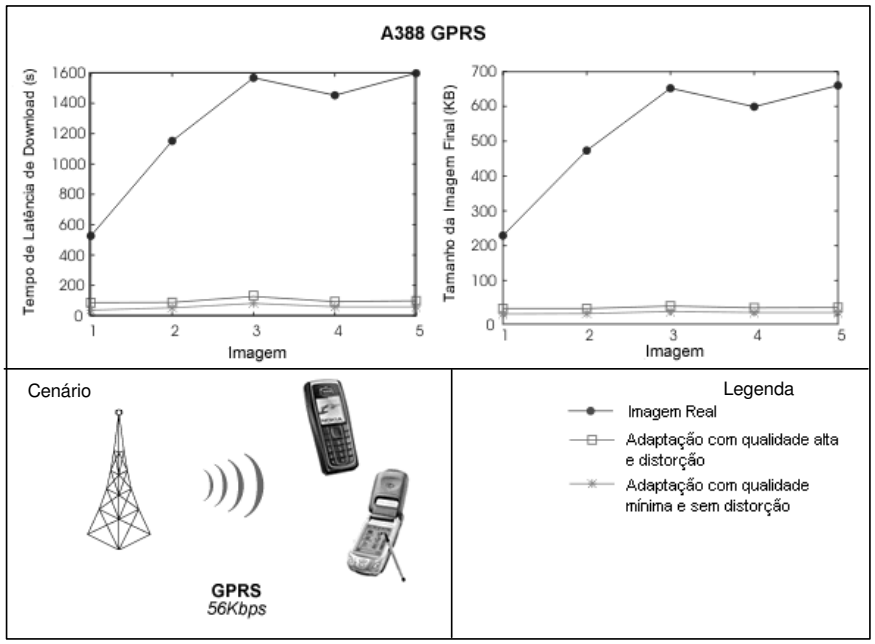


Figura 33 - Gráficos do cenário GPRS com o A388

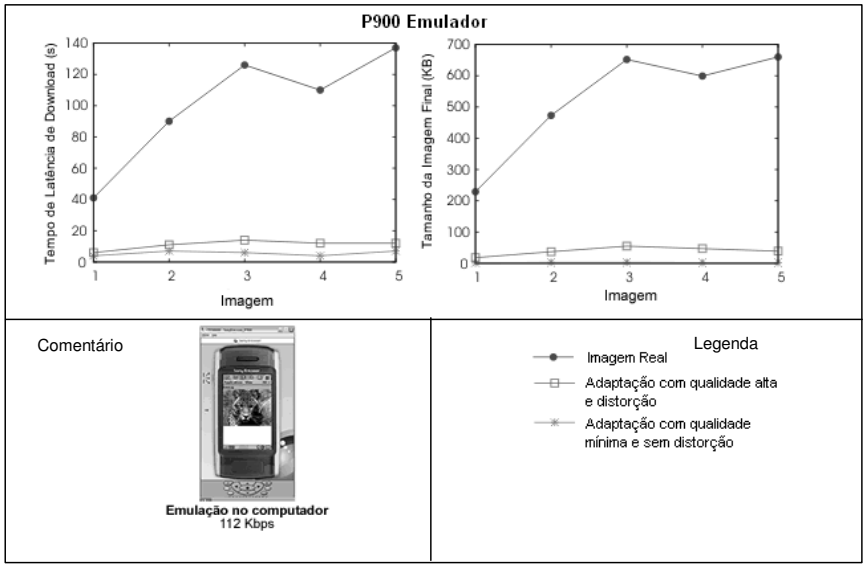


Figura 34 - Gráficos da simulação no cenário EDGE

7.5. Conclusão

Neste capítulo, foi apresentado como o ambiente proposto nesta dissertação pode ser utilizado para desenvolver aplicações multi-plataformas e adaptativas para dispositivos móveis.

A ferramenta UIG facilita a geração de código para as plataformas e os *frameworks* Requisitor, XFormUI e MobAC diminuem a quantidade de código escrito pelo engenheiro de software. Uma demonstração deste fato é que a aplicação cliente, apresentada no estudo de caso, é escrita uma única vez e é executada nas três plataformas. Além disso, essa execução é adaptada para cada plataforma com relação ao *layout* de apresentação das interfaces.

Os *frameworks* Mobile Adapter e MobAC oferecem uma forma eficiente de captura e manipulação dos perfis do usuário, do dispositivo e do contexto. Essas informações são indispensáveis e fundamentais para que se possam criar sistemas que realizam adaptação de conteúdo, o que é ilustrado no M-Flog.

A avaliação de eficiência em ambientes reais e simulados demonstra que mesmo inserindo um passo a mais de conversão entre o dispositivo e o recurso desejado, a adaptação possibilita ganhos significativos no tempo de latência de *download* dos recursos e na diminuição dos pacotes trafegados.

No próximo capítulo são apresentadas as principais contribuições dessa dissertação e trabalhos futuros que podem estender o ambiente proposto.

8. Conclusão

Esta dissertação apresentou um ambiente para o desenvolvimento de aplicações adaptativas e multi-plataformas para dispositivos móveis. Esse ambiente, composto pelos *frameworks* XFormUI, Requisitor, Mobile Adapter, MobAC e pela ferramenta UIG, facilita a construção das interfaces gráficas das aplicações e o desenvolvimento de Moblie Application Servers (MAS) que realizam adaptação de conteúdo.

Na seção 8.1, são apresentadas as principais contribuições da dissertação e os resultados alcançados durante o mestrado. Na seção 8.2, são descritos os trabalhos futuros que possam estender o ambiente apresentado nesta dissertação.

8.1. Contribuições e Resultados Alcançados

No contexto do desenvolvimento de aplicação para dispositivos móveis, o ambiente proposto atende ao objetivo de aprimorar e integrar as soluções para os desafios de descrição, independente de dispositivo e plataforma, da interface com usuário; a adaptação do conteúdo acessado pelas aplicações; e o desenvolvimento de aplicações multi-plataformas.

O *framework* XFormUI possibilita a construção de interfaces gráficas com recursos de validação de dados e *layout* de componentes. Suas implementações em Superwaba, J2ME MIDP 1.0 e J2ME MIDP 2.0 permitem ao engenheiro de software escrever uma única vez as interfaces gráficas e executá-las de forma adaptativa nas três plataformas.

A ferramenta UIG de geração de código diminui o tempo de construção das aplicações já que permite a definição das interfaces, dos estilos e da validação dos dados usando as linguagens declarativas XForms, CSS e XML Schema. Outra importante contribuição dessa ferramenta é o modelo de código gerado que possibilita ao engenheiro de software escrever apenas os controladores dos formulários e dessa forma, concentrar-se nas regras de negócio da aplicação, já que as interfaces já estão construídas. Além disso, o *plugin* do eclipse agiliza a configuração para a geração de código.

Outra importante contribuição é o *framework* Requisitor que facilita a escrita de código para realizar a comunicação de dados. Além disso, as suas implementações em

Superwaba e J2ME MIDP permitem a escrita multi-plataforma de código que envia e recebe informações usando HTTP.

Os *frameworks* Mobile Adapter e MobAC possibilitam a captura, o gerenciamento e a manipulação das informações necessárias para a construção de MAS que realizam adaptação de conteúdo. O uso de CC/PP e UAProf para servir como fonte de informação permitem a escalabilidade da abordagem de captura e tornam possível a implementação de MAS adaptativos em ambientes reais. O uso de *Reflection* para a geração dos *Beans* do usuário, do dispositivo e do contexto torna a implementação do *framework* flexível de forma a permitir através dos *Hot Spots* a adição de novos parâmetros nos perfis.

O estudo de caso, o M-Flog, além de ilustrar como o ambiente proposto na dissertação é utilizado, apresenta estratégias e implementações de adaptação de imagens e textos que podem ser reutilizadas em outras aplicações. Além disso, a avaliação da eficiência da adaptação apresentada no Capítulo 7 comprova as vantagens do uso de adaptação de conteúdo para dispositivos móveis.

Além do ambiente proposto, uma importante contribuição desta dissertação é o estudo e a classificação dos trabalhos de adaptação descritos no Capítulo 2. As definições apresentadas permitem uma melhor compreensão dos desafios e das soluções existentes para a adaptação de aplicações para DMs.

Finalmente, vale ressaltar que durante o desenvolvimento desta dissertação, foram publicados artigos relacionados ao desenvolvimento de aplicações para DMs no *International Conference on Telecommunications* - ICT 2004 [3], Simpósio Brasileiro de Banco de Dados - SBBD 2004 [1], *Mobility Aware Technologies and Applications* - MATA 2004 [4], Salão de Ferramentas do Simpósio Brasileiro de Redes de Computadores – SBRC 2005 [6]. Além disso, foram aceitos artigos no *Advanced International Conference on Telecommunications* - AICT 2005 [2] e também um artigo diretamente relacionado ao trabalho no Semish do Congresso da Sociedade Brasileira de Computação - CSBC 2005 [5].

8.2. Trabalhos futuros

O *framework* XFormUI pode ser estendido para suportar uma maior quantidade de eventos e permitir o uso de expressões regulares na definição de restrições. Além disso,

esse *framework* pode ser implementado em outras plataformas de programação, como as baseadas na linguagem C++ (e.g., BREW [59]) .

Em relação à ferramenta UIG, o plugin eclipse da UIG pode ser estendido para permitir a construção das interfaces gráficas das aplicações, criando uma ferramenta RAD (*Rapid Application Developer*). Além disso, a ferramenta pode ser estendida para suportar dados complexos e expressões regulares do XML Schema, assim como mapear maior quantidade de atributos do CSS.

Para melhorar o código gerado pela UIG e facilitar ainda mais a prototipação de aplicações, uma forma de especificar em XML as transições entre os formulários e mapeá-las para código XFormUI pode ser investigada. O modelo de transição usando máquinas de estados finitas proposto em [94] pode ser utilizado como ponto de partida. Outro trabalho futuro, é a investigação do uso de palavras reservadas (i.e., *inline code atributes*) no XForms e transformá-las em códigos pré-determinados. Estes códigos podem ser, por exemplo, chamadas de comunicação de dados, acesso a camadas de persistência e envio de dados seguros. Desta forma, a UIG geraria código automaticamente integrado com os *frameworks* Requisitor, FramePersist [1] e FrameSec [50].

Outro importante trabalho futuro é geração automática de interface gráfica para realizar operações de cadastro, remoção, busca e alteração (i.e., CRUD - *Create, Retrieve, Update e Delete*), a partir de uma modelagem dos dados realizadas em UML (Unified Modeling Language) [91]. Os dados seriam modelados, seguindo uma especificação a ser investigada, em uma ferramenta de modelagem de UML (e.g., JUDE [93]) e exportados para o padrão XMI (XML *Metadata Interchange*) [92]. Na ferramenta UIG, seriam criados XSLTs para mapear de XMI em XMLMiddleForm. Dessa forma, utilizando os processos de mapeamento apresentados nessa dissertação, esse XML seria transformado em códigos executáveis de uma linguagem de programação. Esses códigos, tanto conteriam as interfaces gráficas, como as validações dos dados e as operações de CRUD implementadas.

Com relação aos *frameworks* Mobile Adapter e MobAC, a generalização dos pacotes de adaptação apresentados no estudo de caso pode ser realizada para construir um *framework* de adaptação de conteúdo. A esse *framework* podem ser adicionados outros tipos de adaptação, como adaptação de *streams* de vídeo e de áudio. Além disso, o processo de captura, gerenciamento e manipulação dos perfis apresentados nesses *frameworks* pode

ser integrado ao protocolo ICAP (*Internet Content Adaptation Protocol*) [95] para permitir a construção de *proxies* adaptativos reais.

9. Bibliografia

1. MAGALHÃES, Katy; VIANA, Windson; LEMOS, Fabrício; CASTRO, Javam de; ANDRADE, Rossana. “Um Framework de Persistência de Objetos em Aplicações para Dispositivos Móveis”. In: The Brazilian Symposium on Databases (SBBD 2004). Brasilia-DF, Brasil, Outubro 2004 , 2004.

2. BRINGEL FILHO, Jose Ribamar; VIANA, Windson; CASTRO, Rossana Maria Andrade de. FRAMESEC: a FRAMEwork for the application development with end-to-end SECurity provision in the mobile computing environment. Advanced International Conference on Telecommunications - AICT 2005.

3. VIANA, Windson, CASTRO, R. M. C., MACHADO, Javam, FILHO, Bringel, MAGALHAES, Katy, GIOVANO, Carlo. Mobis: A Solution For The Development Of Secure Applications For Mobile Device. In: ICT: International Conference on Telecommunications, 11th, Fortaleza-CE, Brazil, 2004.. Lecture Notes in Computer Science. , v.3124, p.1015 - 1022, 2004.

4. FILHO, Bringel; VIANA, Windson; CASTRO, R. M. C. PEARL: a PErformance evaluaAtor of cRyptographic aLgorithms for mobile devices. In: The First International Workshop on Mobility Aware Technologies and Applications (MATA 2004). Florianópolis-PR, Brasil, Outubro 2004 . Lecture Notes in Computer Science. MATA 2004 Conference: 275-284, v.3284, 2004.

5. VIANA, Windson; FERNANDES, Paula; TEIXEIRA, Robson; CASTRO, R. M. C. "Mobile Adapter: Uma abordagem para a construção de Mobile Application Servers adaptativos utilizando as especificações CC/PP e UAProf". XXXII Seminário Integrado de Software e Hardware (SEMISH 2005). No XXV Congresso da Sociedade Brasileira de Computação (CSBC). São Leopoldo-RS, Brasil, 2005.

6. VIANA, Windson; FILHO, Bringel; CASTRO, R. M. C. PEARL Tools: um Conjunto de Ferramentas para Avaliação da Eficiência de Algoritmos de Criptografia em Dispositivos Móveis. Salão de Ferramentas do Simpósio Brasileiro de Redes de Computadores (SBRC 2005). Fortaleza-CE, Brasil. 2005.

7. PINHEIRO, Manuele Kirsch; VILLANOVA-OLIVER, M.; GENSEL, J.; MARTIN, H, "Representing Context for an Adaptative Awareness Mechanism", in X International Workshop on Groupware (CRIWG 2004), San Carlos, Costa Rica, Setembro, 2004

8. PINHEIRO, Manuele Kirsch; VILLANOVA-OLIVER, M.; GENSEL, J.; MARTIN, H, "A Context-Based Awareness Mechanism for Mobile Cooperative Users", in International Conference on Cooperative Information Systems (CoopIS 2004), Larnaca, Cyprus, 25 – 29. Outubro, 2004.

9. PINHEIRO, Manuele Kirsch; VILLANOVA-OLIVER, M.; GENSEL, J.; MARTIN, H, "Awareness on Mobile Groupware Systems", in the First International Workshop on Mobility Aware Technologies and Applications (MATA 2004), , Florianópolis, Brazil, 20-22. Outubro, 2004.

10. HAGIMONT, Daniel; LAYAÏDA, Nabil. Adaptation d'une application multimédia par un code mobile. In Technique et Science Informatique (TSI), numéro spécial "Agents et code mobile", Volume 21, Issue 6/2002, Junho, 2002.

11. CASTRO, Márcio; LOUREIRO, A. A. F. Adaptation in Mobile Computing. In: XXII Simpósio Brasileiro de Redes de Computadores, 2004, Gramado, RS. Anais do XXII Simpósio Brasileiro de Redes de Computadores, p.439 – 452, 2004.

12. LEMLOUMA, T. LAYAIDA, N. Context-aware adaptation for mobile devices. This paper is from the WAM Project, INRIA, Saint Martin, France. In: Proceedings of IEEE International Conference on Mobile Data Management, p.106 - 111, 2004.

13. Projeto Sourceforge da linguagem SuperX++. Uma linguagem orientada a objeto baseada em XML. Disponível em: <http://xplusplus.sourceforge.net/index.htm>. Acessado em Setembro de 2004.

14. Site do LiquidUI, ferramenta de UIML. Disponível em: <http:// www.harmonia.com>. Acessado em Março, 2004.

15. HONKALA, M.; VUORIMAA, P.; XForms in X-Smiles. Anais do International Conference on Web Information Systems Engineering, 2. p203 – 211. Dezembro, 2001.

16. MUELLER, A.; MUNDT, T.; LINDNER, W.; Using xml to semi-automatically derive user interfaces. Anais do Uidis 2001. International Workshop on User Interfaces to Data Intensive Systems, 2., p.91 – 95, Junho, 2001.

17. MINELLI, ANDRÉ C.O.; LOULEIRO, ANTÔNIO A.F. Uma ferramenta para desenvolvimento de aplicações para dispositivos móveis. SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES (SBRC), 20., Búzios 2002.

18. Site oficial de O-XML. Disponível em: <http://www.o-xml.org> . Acessado em março de 2004.

19. SANTOS, Misael da Silva. Integração de Modelos de Padrões de Software com Ferramentas de Apoio ao Desenvolvimento de Sistemas: Rose - um estudo de caso. 2004. Dissertação (Ciência da Computação) - Universidade Federal do Ceará.

20. MAGALHAES,Katy. Framepersist: Um Framework de Persistência de Objetos para o Desenvolvimento de Aplicações para Dispositivos Móveis. 2005. Dissertação (Engenharia Elétrica - DETI) - Universidade Federal do Ceará.

21. SHANKAR, R.; LEE, Brian; FOX, Armando; HANRAHAN, P; ICrafter: A Service Framework for Ubiquitous Computing Environments. Anais do Ubiquitous Computing Conference (UBICOMP), 21., 2001.

22. RAATIKAINEN, Kimmo E. E.; Middleware for Mobile Aplications Beyond 3G. International Conference on Intelligence in Networks (SMARTNET 2002),7.,p3-18, Abril, 2002.

23. WIEISER, M.; BROWN, J. “Designing Calm Technology,” PowerGrid J., vol. 1, 1996.

24. WIEISER, M., “The Computer for the 21st Century,” Scientific American, , p. 94–104, vol. 265, no. 3. Setembro, 1991.

25. KINDBERG, Tim; FOX, Armando. System Software for Ubiquitous Computing, IEEE Pervasive Computing, v.1 n.1, p.70-81. Janeiro, 2002.

26. BANAVARF, G. Software Infraestructure and Design Challenges for Ubiquitous Computing Applications. Communications of the ACM, vol.45, no.12, p.92-96, Dezembro, 2002.

27. ALLAN MENG KREBS; MARSIC, I.; Adaptative Applications For Ubiquitous Collaboration In Mobile Environments. Annual Hawaii International Conference On System Sciences, 37., p.31 – 40, 2004.

28. OLSEN, D. R., JEFFERIES, S., NIELSEN, T., MOYES, W., FREDRICKSON, P. Cross-modal Interaction using Xweb. Anais do UIST '00, p.191-200, Novembro de 2000. San Diego, CA, USA.

29. Site do Fórum do UPnP - Universal Plug in Play. Disponível em: <http://www.upnp.org/>. Acessado em Março de 2004.

30. CHI-HSING Chu; CHIEN-HSUAN Huang; LEE, M.; Building an XML-based unified user interface system under J2EE architecture. Anais do International Symposium on Multimedia Software Engineering, p.208 – 214, Dezembro, 2000

31. VILLANOVA-OLIVER, M.; GENSEL, J. ; MARTIN, H. Five models for integrating a progressive access approach into Web-based Information Systems. The Twelfth International World Wide Web Conference, Poster session, 20-24 May 2003, Budapest, Hungary.

32. VILLANOVA-OLIVER, M.; GENSEL, J. ; MARTIN, H. Progressive Views for Adaptable Web-based Information Systems. Object-Oriented Information Systems OOIS'02, LNCS, Montpellier, Setembro 2-5, 2002.

33. GENSEL, J.; FREIRE, J.C. Junior, BARBEIRO, E.; VILLANOVA-OLIVER, M; MARTIN, H. Adaptable Web Teaching with KIWIS. Int. Conference on Engineering Education (ICEE 2002). Manchester, UK . 18 - 22 Agosto, 2002.

34. VILLANOVA-OLIVER, M.; GENSEL, J. ; MARTIN, H. Exploiting a progressive access model to information. 2nd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, LNCS 2347, Ed. Springer, Malaga, Spain, Maio 29-31, 2002.

35. KURUMATANI, K.: Mass User Support by Social Coordination among Citizen in a Real Environment.In: Chen, Shu-Heng; Ohuchi, Azuma (eds.): Multi-Agent for Mass User Support. International Workshop (MAMUS 2003). Acapulco, Mexico, Agosto 10, 2003. LNAI, Vol. 3012. Springer-Verlag, Berlin Heidelberg (2004) 1–16.

36. W3C- WWW CONSORTIUM, Site do fórum de desenvolvimento de tecnologias para Web. Disponível em:<http::\\www.w3c.org >. Acessado em Janeiro de 2004.

37. ECLIPSE, Site da ferramenta eclipse. Disponível em:<http::\\www.eclipse.org >. Acessado em Janeiro de 2004.

38. RAMOS, Angela Carrillo, VILLANOVA-OLIVER, M.; GENSEL, J.; MARTIN, H. "Modelling with Ubiquitous Agents a Web-based Information System Accessed through Mobile Devices", in International Conference on Cooperative Information Systems (CoopIS 2004), Larnaca, Cyprus, 25 – 29, Outubro, 2004.

39. TAKAHASHI, K., AMAMIYA, S., IWAO, T.: An Agent-based Framework for Ubiquitous Systems. In: Challenges in Open Agent Systems '03 Workshop (Challenge03). Melbourne, Australia, July 14-15 2003. (2003) http://www.agentcities.org/Challenge03/papers.php. Janeiro, 2004.

40. DORNAN, ANDY. The Essential Guide to Wireless Communication Applications, Prentice Hall Inc., 2001. ISBN 0-13-031716-0.

41. POSA.Buschmann et al, **Pattern-Oriented Software Architecture**, Wiley, 1996, ISBN: 0471958697.

42. BlueTooth. Disponível em: <http://www.bluetooth.com>. Acessado em Agosto de 2003.

43. Java 2 Platform, Micro Edition (J2ME). Disponível em <http://wireless.java.sun.com>. Acessado em Março de 2004.

44. Superwaba: The Java VM for PDAs. Disponível em < http://www.Superwaba.com.br>. Acessado em Março de 2004.

45. VILJAMAA, A.. Pattern-Based Framework Annotation and Adaptation – A systematic Approach, 2001. 118 f. Thesis (Licenciate Thesis in Computer Science) University of Helsinki, Finland.

46. GAMMA, E. R. HELM; JOHNSON, R.; VLISSIDES J. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison Wesley, 1995.

47. Site da tecnologia XML Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularie. Disponível em: <http://www.w3.org/Mobile/CCPP/> . Acessado em Outubro de 2004.

48. PalmOne Aliance. Site da empresa que fabrica os Palms e os *handhelds* HandSpring. Disponível em: <http://www.palmone.com>. Acessado em Outubro de 2004.

49. SANTOS, Misael da Silva. Uma Proposta para a Integração de Modelos de Padrões de Software com Ferramentas de Apoio ao Desenvolvimento de Sistemas. 104 p, Dissertação (Mestrado em Ciência da Computação), Universidade Federal do Ceará, Fortaleza, 2004.

50. BRINGEL FILHO, José de Ribamar Martins. FRAMESEC: Um *Framework* para a provisão de segurança fim-a-fim para aplicações no ambiente de Computação Móvel. 131 p, Dissertação (Mestrado em Ciência da Computação), Universidade Federal do Ceará, Fortaleza, 2004.

51. Java 2 Plataform Enterprise Edition (J2EE). Disponível em: < http://www.java.sun.com/j2ee/index.jsp >. Acessado em Março de 2004.

52. ABRAMS, M.; PHANOURIOU, C.; BATONGBACAL, A.L.; WILLIAMS, S.M.; SHUSTER, J.; UIML: An Appliance-Independent XML User Interface Language. International World Wide Web Conference, 18.,Toronto, Canada, Maio, 1999. Disponível em: <http://www8.org/w8-papers/5b-hypertextmedia/uiml/uiml.html>.

53. KHARE, R.;Can XForm transform the Web? Transcending the Web as GUI II.Internet Computing, IEEE, Vol 4, p.103 - 106, Abril ,2000.

54. DUBINKO, Micah. **XForms Essential.** Editora O'Reilly & Associates. ISBN: 0596003692. Agosto, 2003.

55. Site de WebServices do Apache, WebServices utilizando SOAP. Disponível em: <http://ws.apache.org/soap/>. Acessado em Outubro de 2004.

56. Desenvolvimento de aplicaçõe para PalmOS utilizando C++. Disponível em <http://www.palmos.com/dev/tools/gcc/>. Acessado em Outubro de 2004.

57. Mophun: plataforma de desenvolvimento de jogos para celulares. Disponível em <http://www.mophun.com/ >. Acessado em Outubro de 2004.

58. CODEWARRIOR. CodeWarrior for Palm OS v8.0. Disponível em: <http://www.metrowerksstore.com/palmcomputing.htm>. Acesso em: Setembro 2004.

59. BREW – Ambiente de execução de aplicativos binários sem fio. Disponível em: <http://www.qualcomm.com/brew/>.Acesso em Outubro de 2003.

60. CAPRA, Licia. Mobile Computing Middleware for Context-Aware Applications. In: Proceddings of International Conference on Software Engineering, 24. p.723-726. Maio, 2002

61. OMOJOKUN, O.; DEWAN, P.; Experiments With Mobile Computing Middleware For Deploying Appliance Uis. International Conference On Distributed Computing Systems Workshops, 23., p.375–380, 2003.

62. SUN MICROSYSTEMS. Writing code for the JECF - Java Electronic Commerce Framework. CA, USA: Sun Microsystems, 1996. Disponível em: <http://java.sun.com/products> . Acessado em Setembro 2004.

63. Site da Máquina Micro Java para PalmOs. Disponível em: <http://www.palmone.com/java>. Acessado em outubro de 2004.

64. Site do fabricante da Jeode, máquina virtual Personal Profile. Disponível em: < http://www.esmertec.com/>. Acessado em outubro de 2004.

65. Banco de dados para PDAs: IBM DB2 Everyplace. Disponível em: <http://www-306.ibm.com/software/data/db2/everyplace/> . Acessado em Outubro de 2004.

66. JOHNSON, R.; FOOTE, B. Designing Reusable Classes. Journal of Object-Oriented Programming. SIGS, 1, 5 (Junho/Julho. 1988), 22-35. Disponível em: <http://wwwlifia.info.unlp.edu.ar/poo2001/DRC.pdf> .

67. ANDRADE, R. Capture, Reuse, and Validation of Requirements and Analysis Patterns for Mobile Systems. Ph.D. Thesis, School of Information Technology and Engineering (SITE), University of Ottawa, Ottawa, Ontario, Canada, Maio, 2001.

68. COPLIEN, J. O. Software Patterns. SIGS books and Multimedia, Junho, 1996.

69. Site do XMLBuddy. *Plugin* para edição de XML para o Eclipse. Disponível em: <http://xmlbuddy.com> . Acessado em Outubro de 2004.

70. Site do JavaML. Linguagem de marcação em XML para descrever programas Java. Disponível em: <http://www.cs.washington.edu/research/constraints/web/badros-javaml-www9/>. Acessado em Outubro de 2004.

71. Site do JAXP. Java API for XML Processing. Disponível em: <http://java.sun.com/xml/jaxp/index.jsp> . Acessado em Outubro de 2004

72. Site do ImageMagick. Disponível em: <http://www.imagemagick.org/> . Acessado em Outubro de 2004.

73. IEEE Std 802.11. “Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications”, Março, 1999.

74. UMAR, Amjad. E-Business and Distributed Systems Handbook: Platforms Module,2003, ISBN: 0972741496.

75. FIGUEIREDO, Thiago H. de Paula, LOUREIRO, A. A. F.MultiMAD: Uma Ferramenta Multimodelo de Desenvolvimento de Aplicações para Dispositivos Móveis In: Salão de Ferramentas, XXIII Simpósio Brasileiro de Redes de Computadores, 2005, Fortaleza. XXIII Simpósio Brasileiro de Redes de Computadores. 2005.

76. MENKHAUS, Guido. Adaptive User Interface Generation in a Mobile Computing Environment. PhD thesis, Salzburg University, 2002.

77. JOHNSON, Ralph E. How to Design Framework. Notes for Proceedings of the Conference on Object-oriented programming system, languages, and applications - OOPSLA93, University of Ilinois, 1993. Disponível em: <ftp://st.cs.uiuc.edu/pub/papers/frameworks/OOPSLA93-frmwk-tut.ps>. Acessado em Maio de 2004.

78. Site do Projeto Eclipse ME, *plugin* para construção de aplicações J2ME no Eclipse. Disponível em: <http://eclipseme.org/index.html>. Acessado em Maio de 2005.

79. Site da IDE Sun Java[tm] Studio Mobility. Disponível em: <http://developers.sun.com/prodtech/javatools/mobility/downloads/index.jsp>. Acessado em Junho de 2005.

80. SOUZA, W. L.; BOCHMANN, G. V., EL-KHATIB K., “Adaptacao de conteúdo de mensagens HTTP baseada em perfis de dispositivo, conteúdo, usuário e servico de rede (Content adaptation of HTTP messages based on device, content, user, and network service profiles)", Proceedings of the 20th Brazilian Symposium of Computer Networks, Vol II, pp. 554-568, Buzios-RJ (Brazil), Maio 20-24, 2002.

81. LEMLOUMA, T.; LAYAIDA, N. "Content Adaptation and Generation Principles for Heterogeneous Clients". In : W3C Workshop on Device Independent Authoring Techniques,SAP University, St. Leon-Rot, Germany, 25-26 Setembro, 2002.

82. LEMLOUMA, T.; LAYAIDA, N. “Media Resources Adaptation for Limited Devices”. Proceedings of the 7th ICCC/IFIP International Conference on Electronic Publishing ELPUB 2003 - Universidade deo Minho, Portugal, Junho 25-28, 2003.

83. LEMLOUMA, T.; LAYAIDA, N. “Adaptation et multimedia mobile sur le Web”. Mcube 2004, first national conference on mobile multimedia, Montbéliard, France, 30-31. Março, 2004.

84. RAVI, J.; STEFANO, P.; WULLERT, John. “The Mobile Application Server (MAS): An Infrastructure Platform for Mobile Wireless Services”. Information Systems Frontiers. Volume 6, Number 1, 23-34, Março, 2004.

85. BERHE, G., BRUNIE, L.; PIERSON, J.M. “Modeling service-based multimedia content adaptation in pervasive computing”. Conf. Computing Frontiers, 60-69, 2004.

86. Satellite Forms. Plataforma RAD para o desenvolvimento em Palm OS e Pocket PC. Disponível em: <http://www.satelliteforms.net/>. Acessado em Abril de 2004.
87. JOHNSON, R. E.; FOOTE, B. Designing Reusable Classes. Journal of Object-Oriented Programming, 1988. Disponível em:<ftp://st.cs.uiuc.edu/pub/papers/frameworks/designing-reusable-classes.ps>. Acessado em Maio de 2003.
88. Site Internacional do Phlog. Disponível em:http://www.phlog.net/. Acessado em Maio 2005.
89. BERHE, Girma; BRUNIE, Lionel; PIERSON, Jean-Marc, "Modeling Service-Based Multimedia Content Adaptation in Pervasive Computing", ACM Proceedings of the first conference on computing frontiers (CF'04). Ischia , Italy , Abril 14 - 16, 2004 pp. 60 – 69.
90. Site do Framework MVC Maverick. Disponível em: <http://mav.sourceforge.net/>. Acessado em Novembro de 2004.
91. Site do UML - Unified Modeling Language. Disponível em: <http://www.uml.org>. Acessado em Junho de 2005.
92. Tecnologia XMI XML Metadata Interchange. Disponível em: <http://www.omg.org/technology/documents/formal/xmi.htm>. Acessado em Abril de 2005.
93. Site da ferramenta livre Jude de modelagem de UML. Disponível em: <http://www.esm.jp/jude-web/en/index.html> Acessado em Março de 2005.
94. Jalal Kawash. Declarative user interfaces for handheld devices. In WISICT '04: Proceedings of the winter international synposium on Information and communication technologies, pages 1–6. Trinity College Dublin, 2004.
95. ICAP: Internet Content Adaptation Protocol. Disponível em: <http://www.i-cap.org/home.html>. Acessado em Março de 2005.
96. Site da especificação XML User Agent Profilling Specfication (UAProf). Disponível em: <http://www.openmobilealliance.org/ >. Acessado em Novembro de 2004.
97. P900 UAProf. Disponível em:<http://wap.sonyericsson.com/UAprof/P900R101.xml>. Acessado em Novembro de 2004.
98. Site da especificação Java Beans. Disponível em: <http://java.sun.com/products/javabeans/docs/spec.html>. Acessado em Janeiro de 2005.
99. SZWARCFITER, Jayme L., MARKENSON, Lilian. Estrutura de Dados e seus Algoritmos. Ed. LTC - Livros Técnicos e Científicos, 1994.
100. GUIMARÃES, Renato. Tutorial Web sobre a técnica de *Reflection*. Disponível em: <http://www.linhadecodigo.com.br/artigos.asp?id_ac=264&sub=0 >. Acessado em Março de 2005.
101. API Commom Bean Utils para uso de *Reflection.* Disponível em: <http://jakarta.apache.org/commons/beanutils/>. Acessado em Março de 2005.
102. VOELTER, M., A Catalog of Patterns for Program Generation, EuroPlop'2003. Disponível em: http://www.voelter.de/data/pub/ProgramGeneration.pdf. Acessado em março de 2005.