Uma proposta para a geração semi-automática de aplicações adaptativas para dispositivos móveis

Windson Viana[1, i], Rossana M. C. Andrade[2]

[1]LSR-IMAG, équipe SIGMA, Université Joseph Fourier (UJF), France

[2]Departamento de Computação (DC), Grupo de Redes, Engenharia de Software e Sistemas (GREat), Universidade Federal do Ceará (UFC)

Windson.Viana-de-Carvalho@imag.fr, rossana@ufc.br

Abstract. *Ubiquitous Computing promises seamless access to information anytime, anywhere with different and heterogeneous devices. When executing an application, this kind of environment demands adaptation of its user interface in order to provide transparent and consistent access. This paper proposes a construction-time approach for semi-automatic generation of adaptive applications to pervasive computers such as mobile phones and PDAs. The approach includes a device-independent user interface framework and a code generation tool for providing fast development of multi-platform and adaptable applications according to device and platform features. A case study is also presented to illustrate how the approach can be used for constructing an application to heterogeneous devices.*

Resumo. Computação Ubíqua promete acesso transparente à informação através de dispositivos diferentes e heterogêneos. A concepção de uma aplicação que executa neste tipo de ambiente apresenta como um dos desafios a adaptação da sua interface com usuário. Com o objetivo de prover o rápido desenvolvimento de aplicações para esse ambiente, este artigo propõe uma abordagem para a geração semi-automática de aplicações para dispositivos móveis multi-plataformas e com diferentes níveis de adaptação. Ela é composta por um *framework* de componentes de interface independentes de dispositivo e por uma ferramenta de geração de código. No artigo, é também apresentado um estudo de caso para ilustrar como o ambiente pode ser utilizado para a construção de uma aplicação para dispositivos heterogêneos e como o código gerado pode ser integrado com outros *frameworks* de desenvolvimento para DMs.

[i] Este trabalho foi financiado pela CAPES no Programa de Mestrado em Ciência da Computação da Universidade Federal do Ceará durante dois anos e pela LG Electronics, convênio UFC/FCPC N° 1472 de março/2005 a setembro/2005.

1. Introdução

A evolução na capacidade de processamento e na miniaturização dos dispositivos computacionais, assim como, o aumento da disponibilidade dos meios de comunicação sem fio para esses dispositivos (e.g., *WiFi*, WiMax, UMTS) torna possível uma visão ubiqüitária da computação [19], na qual os dispositivos computacionais estão imersos no ambiente e permitem a um usuário o acesso à informação, em qualquer lugar e através de diferentes tipos de modalidades de interação (e.g., visual, auditiva).

Um exemplo de uma aplicação ubíqua é um sistema inteligente para transportes urbanos, onde um usuário, José, utiliza um painel interativo de uma estação de metrô para obter informações sobre as linhas e seus horários e, em seguida, para escolher sua estação de destino. A aplicação do painel, por sua vez, detecta a presença dos dispositivos pessoais de José, por exemplo, um *smartphone*, que são celulares com maior capacidade de processamento e recursos multimídia, e um relógio de pulso, ambos equipados com Bluetooth e transfere para estes dispositivos uma aplicação para ajudá-lo em seu itinerário. José, ao entrar em um vagão, consulta seu relógio que informa o horário previsto para sua chegada. Ele, então, decide utilizar seu *smartphone* para consultar informações sobre as linhas que fazem correspondência na estação destino e modifica a parada desejada. Durante o trajeto, José usa seu *smartphone* para ouvir MP3 e dedica-se a leitura de um jornal. Após vinte minutos, a música é interrompida e o seu relógio vibra para indicar que a parada desejada está próxima.

Aplicações como estas já se encontram em discussão em projetos europeus [7,11] e um dos aspectos críticos da construção desses sistemas é a premissa de que as aplicações e os serviços devem ser capazes de executar e de se adaptar à heterogeneidade dos dispositivos computacionais do usuário e do ambiente no qual ele está imerso [9,13]. No caso especial dos dispositivos móveis (DMs) como PDAs, celulares e *smartphones*, essa heterogeneidade é caracterizada por diferentes restrições de processamento, memória, bateria e largura de banda de comunicação. Eles possuem ainda diferenças nas formas de interação com o usuário e nas dimensões e na quantidade de 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 [13].

Dessa forma, a concepção e o desenvolvimento de aplicações que executam em dispositivos heterogêneos tornam-se desafios para o engenheiro de software [9]. Por exemplo, no sistema ubíquo descrito anteriormente, diferentes versões da aplicação de auxílio ao itinerário teriam que ser criadas, de forma a adequar sua interface, seu comportamento de execução e suas funcionalidades ao relógio e ao *smartphone* de José. Contudo, a criação de versões pode se tornar inviável e não escalável para codificação humana devido à variedade de plataformas e dispositivos disponíveis. Assim, existe a necessidade da construção de ferramentas que possam automatizar o processo de criação dessas aplicações [9].

Dentro desse contexto, este artigo propõe uma abordagem para a geração semi-automática de aplicações adaptativas para dispositivos móveis, com o objetivo de reduzir o tempo de prototipagem e facilitar a construção de aplicações com diferentes modos de conectividade à redes. Esta abordagem é composta por um *framework* de componentes de interface e por uma ferramenta para geração de código, que em

conjunto permitem o rápido desenvolvimento de aplicações multi-plataformas com diferentes níveis de adaptação de interface.

O restante deste artigo está organizado da seguinte forma: a seção 2 apresenta conceitos sobre geração interfaces adaptativas; a seção 3 apresenta a abordagem proposta; as seções 4 e 5 apresentam, respectivamente, o *framework* XformUI e o gerador de interface; a seção 6 discute os trabalhos relacionados; a seção 7 apresenta um estudo de caso que ilustra como o ambiente pode ser utilizado para construção de uma aplicação para dispositivos heterogêneos; e, finalmente, na seção 8 são apresentadas as considerações finais e possíveis trabalhos futuros.

2. Geração de Interfaces Adaptativas

A geração automática de interfaces de usuários (UIs) é extremamente útil no desenvolvimento de aplicações, pois proporciona a separação entre a descrição da interface e a lógica da aplicação [4]. Em geral, os ambientes para construção e geração de UIs utilizam uma hierarquia de camadas ou de modelos de descrição da interface para dissociar sua definição da forma como ela é renderizada [16]. Esses modelos de interface permitem a modelagem com níveis distintos de abstração, a reutilização das especificações das interfaces em diferentes fases de um projeto de aplicação, além de prover uma infra-estrutura para a construção de métodos e ferramentas para geração automática da apresentação final da interface [18]. Por exemplo, uma interface pode ser construída em três modelos: um modelo mais abstrato para descrever os componentes e as funcionalidades da interface; um segundo modelo para descrever a composição e o *layout*; e um terceiro modelo concreto no qual os componentes são mapeados para *widgets* reais de uma plataforma.

Em computação ubíqua, o uso da descrição da interface em níveis de abstração permite que para uma mesma interface possam ser definidos mais de um modelo de composição e *layout*, assim como modelos distintos de mapeamentos para *widgets* reais. A escolha de quais modelos são utilizados pode ser associada a diferentes contextos de utilização do usuário e, desta forma, a interface pode ser renderizada de forma adaptada ao dispositivo, à plataforma de programação, às preferências do usuário e às características do ambiente onde ele se encontra.

2.1. Linguagens de Descrição de Interface

Na literatura, são encontradas várias linguagens baseadas em XML concebidas com o objetivo de dissociar a definição da interface da sua apresentação final, tais como: UIML [22], XIML [16], UID [17], AWD [2], Teresa XML [10]. Essas linguagens possuem formas distintas de estruturação da interface e diferem entre si em relação ao poder de expressão, ao conjunto de componentes disponíveis e às técnicas de mapeamento da descrição abstrata para a apresentação final da interface. Por exemplo, UIML divide a definição da interface em cinco níveis: *description*, *structure*, *data*, *style* e *events*. Em <*description*>, <*structure*> e <*data*> descrevem-se os componentes, a sua composição e os dados a serem exibidos. Em <*style*> e <*events*> definem-se os mapeamentos dos componentes e seus eventos em uma linguagem real (e.g., Java). Já XIML utiliza uma estrutura de representação de ontologias, que é o par atributo-valor, para descrever os componentes, as relações e os atributos da interface, assim como as regras de mapeamento para uma determinada plataforma.

Neste trabalho, iremos utilizar XForms [24], um esforço do W3C para substituir formulários HTML por um formato mais adequado para execução em dispositivos heterogêneos. Seus componentes foram criados independentes do paradigma de acesso via *mouse* e da exibição visual em um monitor, o que torna a descrição do formulário independente da modalidade de interação (e.g., visual ou auditiva) e das características do dispositivo onde a aplicação executa. Por exemplo, os campos de entrada de dados, são descritos como "*inputs*" e não como campos de texto que ocupam determinadas posições na tela e os elementos que disparam eventos de ação são descritos como gatilhos (i.e., *triggers*) ao invés de descrevê-los como botões que recebem eventos de *click*. 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, uma interface XForms já é construída com a separação dos dados, da lógica e da apresentação. Outra vantagem do XForms é permitir a integração com outras tecnologias como CSS, XML Schema, XPath e SVG, ampliando e enriquecendo as possibilidades de descrição da interface.

2.2. Estratégias de Geração de Código

Os trabalhos para geração automática de interfaces adaptativas para dispositivos móveis em geral utilizam duas estratégias para mapear a definição abstrata da interface na interface final a ser apresentada:

- **Geração em tempo de construção**. Neste caso as interfaces descritas são mapeadas antes da execução da aplicação e várias versões da interface da aplicação são geradas para diferentes contextos de execução. Exemplos de trabalhos que utilizam essas estratégias são MultiMad [1], SEFAGI [2], UID [17] e Teresa [10].

- **Geração em tempo de execução**. Neste caso, a definição da interface é transformada durante a execução da aplicação. Essa abordagem é usada, sobretudo, para adaptação de sistemas Web baseados em requisição/resposta, nos quais as interfaces são mapeadas para tecnologias Web como WML, XHTML e VoiceXML dependendo do dispositivo que faz a requisição. Os trabalhos LiquidUI [22], e GIA [6] utilizam essa estratégia.

A geração em tempo de execução limita os tipos de aplicações que podem ser construídas, já que exige que a aplicação esteja sempre conectada. E, em geral, com a utilização das tecnologias WML, XHTML e VoiceXML, essa estratégia restringe o acesso a funcionalidades dos dispositivos móveis [15]. No exemplo do sistema de auxílio a itinerário citado na Seção 1, usando essas tecnologias não seria possível acionar a função de vibração do relógio de José, assim como interromper a execução de música em seu *smartphone*. Neste trabalho é utilizada a estratégia de geração em tempo de construção, visto que o uso de geração em tempo de construção permite ao desenvolvedor acrescentar funcionalidades específicas pra cada versão da aplicação. Além disso, ela permite aos engenheiros de software a concepção de aplicações com tecnologias como J2ME MIDP, SuperWaba, Mophun e Doja I-Mode. Essas tecnologias apresentam maior interatividade, com a customização da entrada de dados e do acesso a diversas novas funcionalidades dos DMs (e.g.; *bluetooth*, câmera fotográfica, gps, mp3). Finalmente, elas também oferecem diferentes modos de

conectividade (e.g.; sempre conectado, desconectado com sincronização a posteriori) e diversos protocolos de comunicação (e.g.; HTTP, HTTPs, XML-RPC, SOAP).

3. Abordagem Proposta

Com o objetivo de auxiliar o processo de desenvolvimento de aplicações para DMs e fornecer mecanismos para amenizar o problema da heterogeneidade das plataformas de programação e dos dispositivos, foi proposta uma abordagem para geração semi-automática de aplicações adaptativas. Essa abordagem foi concebida para ser utilizada em processos de desenvolvimento orientados a prototipação e desta forma permitir rapidamente testes de usabilidade das aplicações.

3. 1. Requisitos

Para a concepção da abordagem foram utilizados alguns requisitos apontados em [8,14,5] como fundamentais para a concepção de ambientes de construção de interfaces adaptativas, a seguir:

A. Orientação ao processo de desenvolvimento e Interoperabilidade. Uma ferramenta de construção de interface deve ser facilmente inserida dentro de um processo de desenvolvimento de software. A sua forma de utilização, suas funções, seus documentos de entrada e saída não devem ser divergentes dos processos de engenharia de software. Além disso, a ferramenta tem que ser integrável com outros ambientes de desenvolvimento, pois a heterogeneidade dos dispositivos e das plataformas de programação torna impraticável a concepção de uma única ferramenta para todo o desenvolvimento da aplicação.

B. Riqueza de componentes e extensibilidade. É importante para um engenheiro de software ter a sua disposição uma variedade de componentes de interface, um conjunto de mecanismos de restrição e validação da entrada de dados, funções para navegação entre interfaces e exibição de mensagens, assim como mecanismos para o gerenciamento de eventos. Contudo, todas essas funcionalidades devem ser implementadas suportando a sua extensibilidade, já que as plataformas de programação e os dispositivos móveis evoluem rapidamente.

C. Independência de plataforma e dispositivo. O ambiente deve permitir a descrição da interface de forma dissociada de uma modalidade, de uma plataforma ou de um dispositivo para facilitar o processo de adaptação. Entretanto, essa descrição não pode ser restringida a um conjunto mínimo de funcionalidades comuns entre os dispositivos ou entre as plataformas de programação, pois isto limitaria o processo da adaptação da interface.

3.2. Componentes

A abordagem é composta pelo *framework* XFormUI de componentes de interface e pela ferramenta User Interface Generator (UIG) de geração de código.

O *framework* XFormUI fornece uma infra-estrutura para a construção de aplicações adaptativas baseadas em formulários. O XFormUI foi projetado para conter componentes que tivessem o mesmo nome e comportamento dos componentes do XForms, facilitando o seu aprendizado pelo engenheiro de software, assim como, auxilia a geração de código realizada pela UIG. Esta ferramenta, por sua vez, permite

ao engenheiro de software a definição das interfaces, dos estilos e da validação dos dados usando as seguintes normas técnicas (i.e.; *standards*) do W3C [24]: XForms, XMLSchema e CSS. A ferramenta utiliza esses documentos para gerar automaticamente código adaptável das interfaces das aplicações. O uso dos *standards* W3C facilita a aprendizagem e proporciona a integração do ambiente XMobile com outras ferramentas de desenvolvimento.

Na Figura 1, é apresentada uma visão geral da abordagem e do relacionamento entre o *framework* XFormUI e a ferramenta UIG. A abordagem permite ao engenheiro de software descrever os formulários de uma aplicação utilizando o XForms para definir os componentes, o CSS para definir o estilo (e.g., cores e *layout*) e o XML Schema para definir restrições aos campos de entrada de dados do formulário. O engenheiro, utilizando um plugin do Eclipse, cria um documento XML, o Manifest.xml, que define a navegabilidade entre as interfaces e as diretivas de mapeamento para um código executável de uma plataforma de programação. Ele dispara a ferramenta UIG e esta, por sua vez, gera o código dos formulários utilizando o *framework* XFormUI.

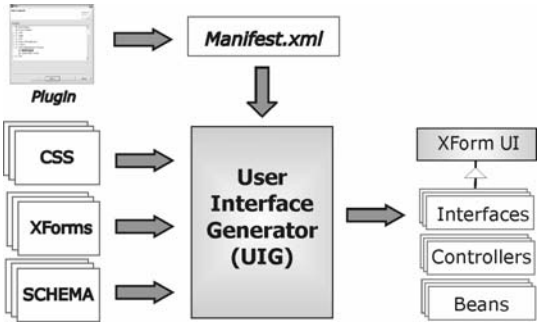

Figura 1 - Visão geral dos componentes

3.3. Tipos de Adaptação Utilizados

As plataformas de programação para DMs, como Superwaba e J2ME MIDP, oferecem algumas funcionalidades de adaptação dinâmica de interface que permitem uma certa independência de um dispositivo específico. Por exemplo, a disposição dos componentes na interface pode ser descrita de maneira relativa entre os componentes (um componente A à esquerda de um componente B). Além disso, os componentes de seleção e de ação podem alterar o estilo de apresentação de acordo com a forma de interação do dispositivo (e.g., via *stylus* ou via teclado do celular). No entanto, como mencionado na Seção 1, essas plataformas de programação não são suportadas de maneira homogênea pelos sistemas de operacionais de DMs, o que dificulta a adoção de uma única plataforma para o desenvolvimento das aplicações. Com o objetivo de solucionar este problema e de reutilizar as funções de adaptação já existentes nessas plataformas, a abordagem proposta neste artigo fornece três tipos de adaptação de interface: i) adaptação às classes de dispositivo, ii) adaptação à plataforma, e iii) adaptação ao dispositivo. Na Figura 2, são apresentados esses três tipos de adaptação e como elas são encadeadas durante o processo de geração de código.

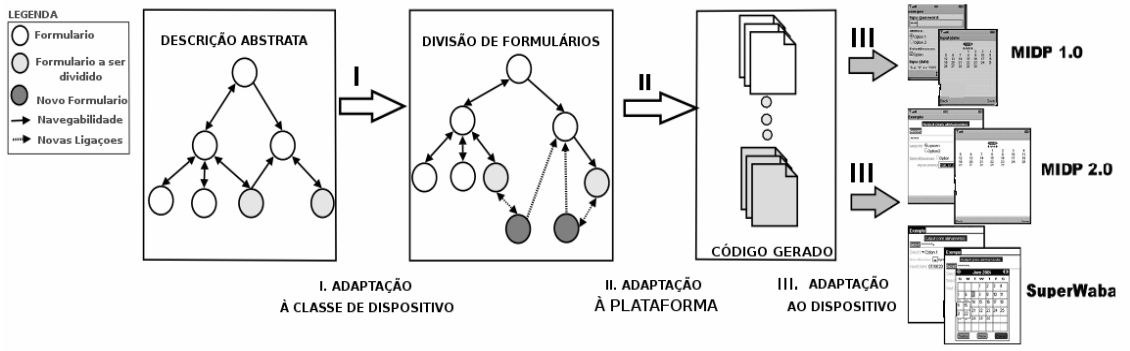

Figura 2 - Os três tipos de adaptação permitidos pelo ambiente

Os dois primeiros tipos de adaptação são fornecidos diretamente pela UIG através da geração de diferentes versões da aplicação. Contudo, **a adaptação ao dispositivo** provém da forma como o código é gerado pela ferramenta UIG. As interfaces geradas utilizam o *framework* XformUI que foi especialmente projetado para reutilizar as funcionalidades de adaptação dinâmica ao dispositivo disponível nas plataformas de programação.

A adaptação à classe de dispositivo consiste em limitar o número de componentes que podem ser apresentados simultaneamentes na interface. Para cada classe de dispositivo, a UIG estabelece um número máximo de componentes por interface. Quando uma descrição em XForm da interface possui mais componentes que o número máximo admitido, uma divisão automática da interface é realizada (o formulário é transformado em vários formulários para que todos os componentes sejam exibidos). Para permitir essa divisão, a descrição do formulário XForm é particionada em diferentes formulários através da utilização do padrão *Wizard Dialog Pattern* [21]. Esse padrão permite a construção de uma seqüência de formulários interligados e foi concebido inicialmente para adequar formulários Web a interfaces de aparelhos celulares. Além disso, aos novos formulários são acrescidos Triggers (i.e.; próximo e anterior) para realizar a interligação entre eles. E os Triggers iniciais contidos na descrição XForm original são colocados no último formulário da seqüência do *Wizard*. Dessa forma, o código gerado garante a similaridade e mantém a consistência da interface descrita pelo engenheiro de software. No ambiente, três classes de dispositivo foram definidas: "mobilephone", "smartphone" e "PDA". Para cada classe de dispositivo, um número máximo de componentes foi definido através de testes de utilização. Por exemplo, para a classe de dispositivo "mobilephone", o número máximo de componentes definido é oito. Assim se uma interface contém mais de oito componentes e se a classe de dispositivo escolhida para a transformação é "mobilephone", então a interface é dividida. Vale ressaltar que o desenvolvedor pode alterar, se desejar, o número máximo de componentes de cada classe de dispositivo.

A adaptação à plataforma de programação é realizada de duas maneiras na ferramenta UIG. A primeira forma consiste também na divisão de interfaces e é utilizada quando a plataforma de programação alvo da geração de código não permite a apresentação simultânea de certos componentes. Por exemplo, J2ME MIDP não permite apresentar numa mesma interface um componente de exibição de texto (TextBox) e outro campo de exibição de dados. Para resolver este problema, a

interface é dividida usando o padrão descrito anteriormente. A segunda forma de adaptação à plataforma é ligada às funcionalidades de estilo de apresentação (cores, *layout*). A ferramenta utiliza as informações descritas pelo CSS, apenas se a plataforma alvo da geração de código suporta funcionalidades de mudança de cores. Esta modificação de regras visa otimizar o código gerado a fim de utilizar da melhor forma possível os recursos de definição de interface disponíveis em cada plataforma.

4. Framework XFormUI

O *framework* XFormUI consiste em um *toolkit* de componentes de interface independente de uma plataforma de programação. Além dos componentes, o XFormUI disponibiliza funções para navegação entre interfaces, funções de restrição de dados de entrada (e.g., limitação de tipos numéricos e quantificadores existenciais), funções para validação automática de dados com exibição de mensagens de alerta e mecanismos para definição de *layout* e estilo dos formulários. Construído através de uma estratégia *bottom-up*, o *framework* oferece abstração das APIs de diferentes plataformas de programação para DMs e utiliza os recursos de adaptação de interface encontrados nestas plataformas. Durante a concepção do *framework* quatro estratégias foram utilizadas para atender os requisitos de riqueza de componentes, extensibilidade, adaptabilidade e independência de plataforma de programação (i.e., partes dos requisitos B e C mencionados na Seção 3.1.). Essas estratégias são listadas a seguir:

- **Generalização**. Para que o XformUI fosse implementado em várias plataformas, um estudo das principais plataformas de programação para DMs foi realizado e 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 e 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. Essa estratégia permite às aplicações construídas executarem em qualquer plataforma na qual o *framework* seja implementado.

- **Especificidade**. Para evitar que o *framework* se limitasse a um conjunto mínimo de características comuns das plataformas de programação investigadas, funções que não correspondem a funcionalidades comuns foram adicionadas ao *framework*. Estas funções estão principalmente relacionadas ao gerenciamento de *layout*, que não é disponibilizado 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 predispõem 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 para 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, então 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 e garantir, em parte, a similaridade entre as implementações.

Na Figura 3, 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 pode-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.* A classe MainXForm é uma classe abstrata que contém os métodos *onStart()* e *onExit()* que devem ser reescritos pelo engenheiro de software para definir o que ocorre ao iniciar aplicação e ao sair dela. Essa classe implementa os métodos obrigatórios das estruturas de cada plataforma e mapeia suas chamadas para os seus métodos abstratos.

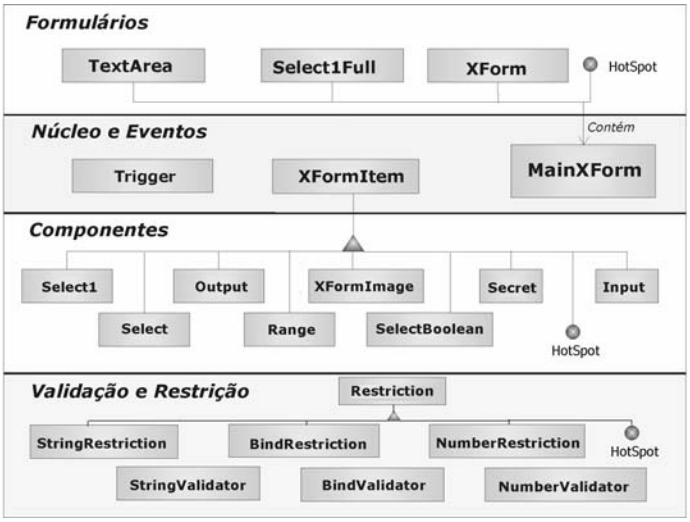

Figura 3 - *Framework* XFormUI

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. 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. Por exemplo, restrições como quantidades de caracteres máximas e mínimas de um texto e intervalo de validade para um valor numérico podem ser expressas usando métodos dessas classes. Na Tabela 1 pode ser

visto como algumas classes e métodos do *framework* são mapeados para as plataformas de programação.

XFormUI	J2ME MIDP 1.0	J2ME MIDP 2.0	Superwaba
MainXFform	*Midlet*	*Midlet*	*MainWindow*
TextArea	*TextBox*	*TextBox*	*Container* com um *ListBox*
XFormItem	*Item*	*Item* e *ColorLabel*	*Control* e *Label*
Input	Pode conter um *TextField* ou um *DateField*	*ColorLabel e* pode conter um *TextField* ou um *DateField*	*Edit* e *Label*. No caso da restrição de datas, um *Calendar* é associado
Output	*StringItem*	*StringItem* e *ColorLabel*	*Label*
Select1	*ChoiceGroup* com restrição *Exclusive*	*ColorLabel* e *ChoiceGroup* com restrição *Exclusive* ou *Popup*	*Label* e *ComboBox* ou *RadioGroup* ou *um ListBox*
triggerEvent()	*commandAction()*	*commandAction()*	*onEvent()*
setForeColor()	-	*ColorLabel.setForeColor()*	*Control.setForeColor()*

Tabela 1- Mapeamento das classes do *framework*

5. Ferramenta de Geração da Interface do Usuário

A ferramenta User Interface Generator (UIG) foi construída com o objetivo de mapear os documentos que descrevem uma aplicação (i.e., XForms, CSS, XMLSchema e Manifest.xml) em um código executável. A ferramenta realiza essa geração de código utilizando XSL [24] (i.e., eXtensible Stylesheet Language). Essa linguagem de marcação baseada em XML permite a criação de folhas de estilo ou documentos de transformação chamados XSLT (i.e., XSL Transformation). O XSLT permite a definição de regras de mapeamento das tags de um documento XML para uma outra descrição. 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.

Na Figura 4 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, o documento XForms e o XML Schema são transformados para um documento intermediário: o XMLMiddleForm.

O XMLMiddleForm é a forma canônica definida pelo UIG para facilitar o mapeamento para as plataformas de programação. Os documentos XForms, XCSS e XMLSchema 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 [23], que tem como vantagem facilitar a extensibilidade da ferramenta para a adição de novos tipos de mapeamentos (e.g., para a plataforma de programação BREW).

Após a geração do XMLMiddleForm, o Manifest.xml é lido e são identificadas a plataforma e a classe de dispositivo desejadas pelo engenheiro de software para realizar a transformação. Assim, a UIG passa para o processador XSLT o *template*

específico da classe do dispositivo que verifica a necessidade da realização da divisão dos formulários. Em seguida, os *templates* que mapeiam de XMLMiddleForm para classes Java que instanciam XFormUI são processados. A construção da ferramenta UIG foi realizada em Java e utilizou o JAXP (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.

O código dos formulários gerados pela ferramenta segue um modelo semelhante ao MVC (i.e., *Model/Viewer/Controller*). 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 a sua composição. Esses componentes gráficos utilizados são instâncias das classes que fazem parte do *framework* XFormUI. Além disso, diferentes construtores são gerados para a classe *Interface* permitindo a sua instanciação com dados estáticos e dados dinâmicos. A classe *Bean* contém atributos relacionados aos campos de seleção e de entrada de dados do formulário, assim, essa classe corresponde ao modelo no qual os dados do formulário estão relacionados. Por sua vez, a classe *Controller* contém métodos de execução para cada *Trigger* existente nos formulários, além dos métodos para validação de dados e de navegabilidade. Esse modelo de código gerado permite o desenvolvimento mais rápido das aplicações, pois o engenheiro se concentra na escrita dos métodos da classe controladora referentes às regras de negócio da aplicação.

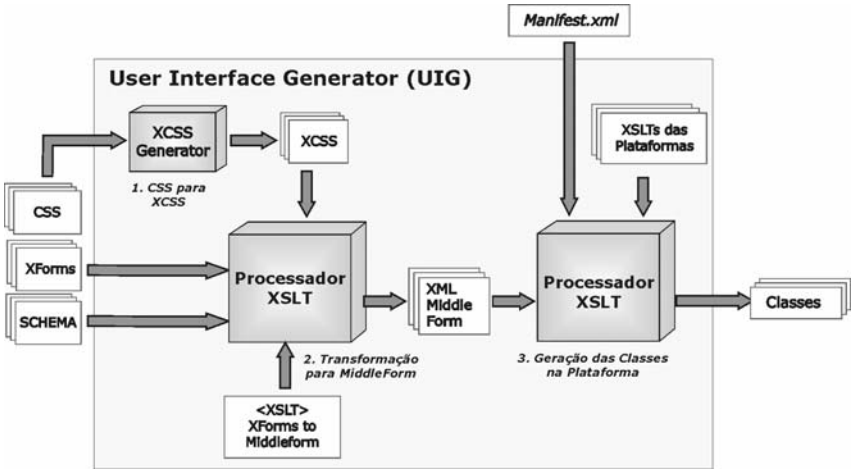

Figura 4 - Processo de mapeamento realizado pela ferramenta UIG

6. Trabalhos Relacionados

Nesta seção, são apresentados os trabalhos que possuem estreita relação com a abordagem apresentada neste artigo. Na Tabela 2, é apresentada uma comparação desses trabalhos com a abordagem discutida nas seções anteriores. Nesta tabela podem ser vistos os tipos de adaptação de interface suportados pelos ambientes e as principais diferenças entre as abordagens, a seguir: tipos de aplicações, plataformas, componentes de interfaces e outras características específicas.

O MultiMad, por exemplo, é uma ferramenta visual para geração de aplicações multi-modelos para dispositivos móveis [1]. Ela permite a construção de quatro tipos de formulários com um conjunto limitado de componentes genéricos. Contudo, a ferramenta não oferece outros mecanismos de adaptação, além da geração de código multi-plataforma. O SEFAGI, por sua vez, é uma arquitetura para geração automática

de interface para aplicações baseadas em Web-Services [2]. As janelas das aplicações são definidas em XML AWD (*Abstract Window Description*) através da composição de painéis (i.e., painel de exibição de vídeo, painel de exibição de uma tabela), na qual cada painel é associado a um serviço Web. Regras de adaptação são definidas para gerar as interfaces em código executável. Essas regras definem o mapeamento de *widget* descrito em AWD para um componente de uma plataforma, assim como, o *layout* da interface (e.g.; posição dos componentes, divisão da janela). A principal desvantagem do uso do SEFAGI decorre da necessidade de criar, para cada novo componente construído, um componente correspondente na descrição do AWD. Além de limitar a aplicação à um cliente que acessa um Web-Service.

	Tipos de Aplicações	***Plataformas***	***Componentes de Interface***	***Tipos de Adaptação de Interface***	***Características Específicas***
MultiMad	Aplicações *online/offline*	WML 1.0/2.0 e J2ME MIDP 1.0 e 2.0	Listas, Textos, Formulários com campos de entrada de dados e datas	Adaptação estática à plataforma	Ferramenta Visual, Passagem de parâmetros e código MobileVC
SEFAGI	Aplicações *online* baseadas em *Web-Services*	J2SE e J2ME	Painéis com grupos de componentes (grid, lista, imagens)	Adaptação estática à plataforma e dinâmica à classe de dispositivos	Tabelas de regras de adaptação para mapear AWD em componentes específicos. Divisão de formulários
Roam	Aplicações moveis	J2SE, Personal Java e J2ME Doja	Componentes semelhantes a API Java Swing	Adaptação dinâmica à classe de dispositivos	Migração de código, recuperação de estado da aplicação
UIG + XFormUI	Aplicações *online/offline*	Superwaba, J2ME MIDP 1.0 e 2.0	10 componentes do XForms e o *image* do XHtml	Adaptação estática à plataforma e à classe de dispositivos. Adaptação dinâmica ao dispositivo	XSLT para geração de código. Suporte a CSS e XML Schema. Plugin Eclipse e Manifest.xml para a navegação entre as interfaces. Divisão de formulários

Tabela 2 –Trabalhos relacionados com a geração de interfaces adaptativas

Roam é um *framework* baseado em componentes para construção de aplicações que executam em dispositivos heterogêneos e que podem migrar entre eles [13]. Roam permite ao desenvolvedor definir a interface da aplicação através de um toolkit de componentes de interface independentes de dispositivos. Esses componentes assemelham-se aos componentes gráficos da API Java Swing. Este *framework* utiliza uma estrutura de agentes para garantir a captura e a transferência do estado da aplicação quando esta migra entre os dispositivos. O engenheiro de software descreve como os componentes devem ser adaptados para um par plataforma-dispositivo e se a lógica da aplicação deve migrar para um dispositivo de maior capacidade (e.g., um computador desktop). Durante a migração da aplicação, um agente Roam é responsável por escolher como a interface será exibida no novo dispositivo através das estratégias de adaptação descritas.

A nossa abordagem tem como principal diferencial o suporte tanto de adaptação estática, com a ferramenta UIG, quanto de adaptação dinâmica, com o *framework* XFormUI. E ao contrário dos trabalhos Roam e SEFAGI, não existe uma limitação dos tipos de aplicação que podem ser construídas. Além disso, o uso de normas técnicas do W3C facilita o aprendizado e interoperabilidade da ferramenta UIG.

7. Estudo de Caso

Para validar a abordagem, foi construído um *mobile fotolog* que permite aos usuários publicarem suas fotos e comentários utilizando um PC ou um dispositivo móvel. O portal, chamado de M-Flog, pode ser acessado por uma grande quantidade de usuários e deve responder em um tempo satisfatório às requisições. Durante a concepção do portal, requisitos não funcionais foram identificados:

- **Adaptação 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 acessadas.** A adaptação das imagens é necessária devido à dificuldade que um dispositivo móvel pode ter para conseguir acessar uma imagem fotográfica da *Web*, por causa do seu tamanho e formato.
- **Diferentes modos de conectividade.** A aplicação deve permitir o armazenamento e o acesso das imagens preferidas dos usuários no próprio dispositivo de forma a evitar a troca excessiva de pacotes.
- **Acesso a funcionalidades multimídia.** O cliente do M-Flog pode utilizar, em caso de existência, a câmera fotográfica do dispositivo móvel e o protocolo MMS para envio de fotos.

Este exemplo simples de aplicação não pode ser construído com os trabalhos [6][22] para adaptação de interface baseados somente em tecnologias Web para celulares (i.e., WML e XHTML) e com os trabalhos baseados em modelo de conectividade *online* [2][13], já que os dois últimos requisitos não podem ser satisfeitos. Desta forma, a proposta deste artigo mostra-se mais adequada. Na Figura 5, uma visão geral do M-Flog é apresentada. Ele é composto de uma aplicação cliente que executa em DMs e de um servidor Web. Para construir o cliente, os formulários foram especificados em XForms, CSS e XML Schema.

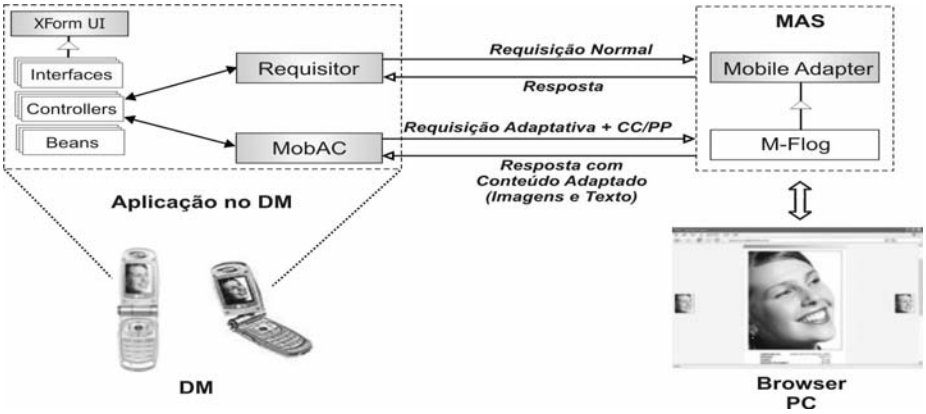

Figura 5 - Visão geral do M-Flog

Após a escrita dos formulários, um Manifest.xml foi criado com as diretivas de mapeamento e a descrição da navegação. Foram realizadas três gerações de código: J2ME MIDP 1.0 com a classe de dispositivo "mobilephone", J2ME MIDP 2.0 com a classe de dispositivo "smartphone" e SuperWaba com a classe de dispositivo "PDA". 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.

O servidor do M-Flog foi construído com a arquitetura MobileAdapter [3] que proporciona a construção de aplicações para a adaptação da apresentação de conteúdo baseada nas preferências do usuário e nas características do dispositivo e do ambiente de execução. Assim, foi construído um módulo para a adaptação de imagens do M-Flog que utiliza as dimensões do dispositivo e as preferências de exibição descritas pelo usuário, mais detalhes sobre esse processo de adaptação são descritos em [3].

Para realizar a comunicação entre as aplicações geradas e o servidor do M-Flog, as classes controladores do código gerado foram associadas aos *frameworks* Requisitor e MobAC [3] que permitem a comunicação e aquisição de contexto nas três plataformas alvo. Esses códigos escritos nos controladores executam nas três plataformas de programação, já que elas são baseadas em Java e os códigos utilizam os *frameworks* que são também implementados nas plataformas. Desta forma, todas as funcionalidades de validação dos dados, troca de formulários e comunicação com o servidor do M-Flog foram escritas apenas uma vez pelo engenheiro de software.

O armazenamento local das imagens preferidas do usuário foi realizado utilizando o modelo de serialização, persistência e busca proposto em [12], que permite sua implementação nas três plataformas através do mapeamento das classes do FramePersist para as APIs RMS e Catalog respectivamente de J2ME MIDP e SuperWaba. O acesso a câmera foi realizado apenas na plataforma J2ME MIDP 2.0 com uso da Mobile Media API (MMA). As versões do cliente M-Flog foram executadas em três dispositivos, um HP iPaq 4100 com SuperWaba, o celular motorola A388 com MIDP 1.0 e no *smartphone* P900 com MIDP 2.0. Na Figura 6, é apresentado um fluxo de execução da aplicação cliente do M-Flog no emulador do P900. Neste fluxo, podem ser vistos os formulários criados 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.

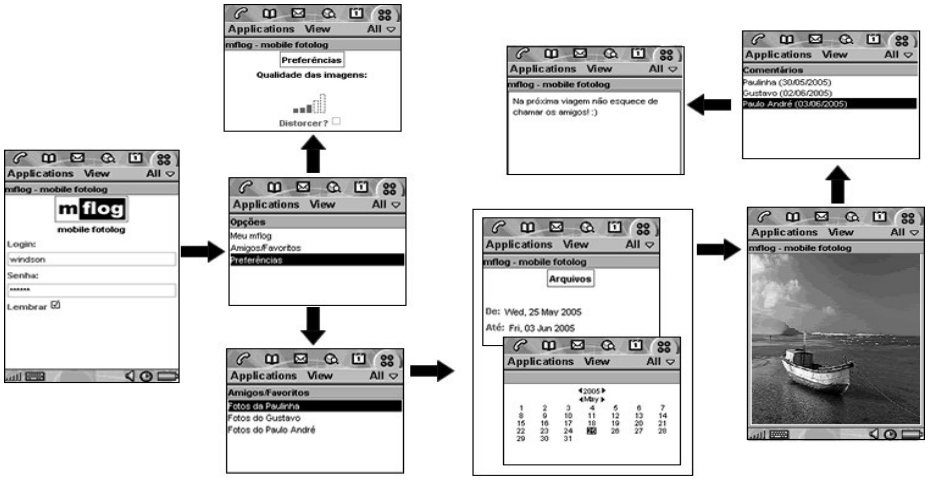

Figura 6 - Fluxo de execução do cliente do M-Flog no emulador do P900

8. Conclusão e Trabalhos Futuros

A abordagem proposta deve facilitar a rápida prototipagem das interfaces de aplicações para aplicações pervasivas em dispositivos móveis heterogêneos e proporcionar

diferentes níveis de adaptação. Por um lado, o *framework* XFormUI possibilita a construção de interfaces gráficas com recursos de validação de dados e *layout* de componentes. Suas implementações permitem ao engenheiro de software escrever uma única vez as interfaces gráficas e executá-las de forma adaptativa em três plataformas. Por outro lado, a ferramenta UIG de geração de código permite a definição das interfaces, dos estilos e da validação dos dados usando as linguagens declarativas XForms, CSS e XML Schema. Esta ferramenta proporciona ainda a adaptação da interface em relação a classes de dispositivos, com a divisão de formulários, e gera automaticamente o código para navegação entre eles. 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, uma vez que as interfaces já estão construídas.

O estudo de caso apresentado ilustra a capacidade da abordagem de construir uma aplicação para dispositivos heterogêneos com diferentes modos de conectividade. Além disso, ela demonstra como o código gerado pela UIG pode ser facilmente integrado a outros *frameworks* de desenvolvimento.

Como trabalhos futuros, a abordagem pode ser estendida com a adição de componentes com outras modalidades de interface (e.g., entrada/saída de dados em áudio). Além disso, um estudo da usabilidade das interfaces geradas pela ferramenta deve ser realizado para identificar o grau de interferência do uso da adaptação na usabilidade da aplicação.

Referências

1. FIGUEIREDO, Thiago H., LOUREIRO, A. A. F. “MultiMAD: Uma Ferramenta Multimodelo de Desenvolvimento de Aplicações para Dispositivos Móveis”. Anais do Salão de Ferramentas, XXIII Simpósio Brasileiro de Redes de Computadores, Fortaleza-CE, Brasil, 2005.

2. CHAARI, T.; LAFOREST, F. “SEFAGI: Simple Environment For Adaptable Graphical Interfaces”. Anais do VII International Conference on Enterprise Information Systems (ICEIS), Miami, USA, 2005.

3. VIANA, Windson; FERNANDES, Paula; TEIXEIRA, Robson; ANDRADE, R. M. C. "Mobile Adapter: Uma abordagem para a construção de Mobile Application Servers adaptativos utilizando as especificações CC/PP e UAProf". Anais do XXXII Seminário Integrado de Software e Hardware (SEMISH 2005). São Leopoldo-RS, Brasil, 2005.

4. NICHOLS, Jeffrey; FAULRING, Andrew. “Automatic Interface Generation and Future User Interface Tools”. Anais do Workshop: The Future of User Interface Design Tools, ACM CHI 2005, Portland, USA, 2005.

5. PUERTA, A. “A Better Future for UI Tools through Engineering”. Anais do Workshop: The Future of User Interface Design Tools, ACM CHI 2005, Portland, USA, 2005.

6. ITO, G.; ROCHA, R.; GONÇALVES, M.; SANTANNA, N. “Uma Arquitetura para Geração de Interfaces Adaptativas para Dispositivos Móveis”. Anais do 4th International Information and Telecommunication Technologies Symposium (I2TS), Florianopolis, Brasil, 2005.

7. COUTAZ, J.; CROWLEY, J. L.; DOBSON, S.; GARLAN, D. “Context is key”. *Communications of the ACM*, vol.48, nº 8, p.49-53. 2005.

8. GAJOS, Krzysztof; WU, Anthony; WELD, Daniel S. "Cross-Device Consistency in Automatically Generated User Interfaces". Anais do Workshop on Multi-User and Ubiquitous User Interfaces (MU3I'05). San Diego, CA, 2005.

9. GAJOS, Krzysztof; WELD, Daniel S. "SUPPLE: automatically generating user interfaces". Anais do Intelligent User Interfaces (IUT'2004). Funchal, Portugal, 2004.

10.PATERNO, F.; MORI, Giulio; SANTORO, Carmen. "Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions". *IEEE Transactions on Software Engineering*, vol. 30, nº 8, p. 507-520, 2004.

11. RIOULT, Jean; GRANSART,Christophe; AMBELLOUIS, Sebastien. « Zut, j'ai loupé mon arrêt ! Un nouveau service d'aide aux déplacements ». Anais do II Colloque Les nouvelles technologies dans la cité (em françês). Lille, França, 2004.

12.MAGALHÃES, Katy; VIANA, Windson; LEMOS, Fabrício; CASTRO, Javam de; ANDRADE, Rossana. "FramePersist: An Object Persistence Framework for Mobile Device Applications". Anais do Brazilian Symposium on Databases (SBBD), Brasilia-DF, 2004.

13.CHU, H.; SONG, H.; WONG, C.; KURAKAKE, S.; KATAGIRI, M.. "Roam, a seamless application framework". *Journal of Systems and Software*, vol. 69, nº 3, p.209-231, 2004.

14.WEGSCHEIDER,Florian; DANGL,Thomas; JANK, Michael; SIMON, Rainer. "A Multimodal Interaction Manager for Device Independent Mobile Applications". Anais do XIII ACM International World Wide Web Conference, New York, USA, 2004.

15.READ, Kris; MAURER, Frank. "Developing Mobile Wireless Applications". *IEEE Internet Computing*, vol. 07, no. 1, p. 81-86, 2003.

16.PUERTA, Angel; EISENSTEIN, Jacob. "XIML: A Universal Language for User Interfaces". Anais do IUI 2002 - Intelligent User Interfaces. San Francisco, USA, 2002.

17.MUELLER, A.; MUNDT, T.; LINDNER, W. "Using xml to semi-automatically derive user interfaces". Anais do Uidis 2001 - II International Workshop on User Interfaces to Data Intensive Systems, Zurique, Suiça, 2001.

18.DA SILVA, P.P. User Interface Declarative Models and Development Environments: a Survey. Anais do DSV-IS'2000, 2000.

19.WEISER, M. The computer for the 21st century. Scientific American, 1991.

20.Open Mobile Aliance. Criadores do Wap e do UAProf. Disponivel em: http://www.openmobilealliance.org/. Acessado em Dezembro de 2005.

21.HUI, Ben. Big Design for Small Devices. Design Patterns for J2ME. Disponivel em: http://www.javaworld.com/javaworld/jw-12-2002/jw-1213-j2medesign.html. Acessado em Agosto de 2005.

22.Site do LiquidUI, ferramenta de UIML. Disponível em: <http:// www.harmonia.com>. Acessado em Março de 2006.

23.VOELTER, M., "A Catalog of Patterns for Program Generation". Anais do, EuroPlop'2003. Alemanha, 2003.

24.W3C- WWW CONSORTIUM, Site do fórum de desenvolvimento de tecnologias para Web. Disponível em:<http::\\www.w3c.org >. Acessado em Janeiro de 2006.