Linguagens Declarativas e Tecnologias da Web no Desenvolvimento de Interfaces de Usuário de Dispositivos Portáteis

João Loureiro Júnior

Trabalho Final de Mestrado Profissional

Instituto de Computação
Universidade Estadual de Campinas

Linguagens Declarativas e Tecnologias da Web no Desenvolvimento de Interfaces de Usuário de Dispositivos Portáteis

João Loureiro Júnior

Dezembro de 2005

Banca Examinadora:

- **Profa. Dra. Maria Cecília Calani Baranauskas (Orientadora)**
Instituto de Computação, UNICAMP

- **Prof. Dr. Ricardo de Oliveira Anido**
Instituto de Computação, UNICAMP

- **Profa. Dra. Lúcia Vilela Leite Filgueiras**
Depto. de Engenharia de Computação e Sistemas Digitais, Escola Politécnica da USP

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Júlia Milani Rodrigues – CRB8a / 2116

Loureiro Júnior, João
L934L Linguagens declarativas e tecnologias da Web no desenvolvimento de interfaces de usuário de dispositivos portáteis / João Loureiro Júnior -- Campinas, [S.P. :s.n.], 2005

Orientador: Maria Cecília Calani Baranauskas
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Interfaces de usuário (Sistema de Computador). 2. Computadores portáteis. 3. Internet. 4. Linguagens de programação funcional. I. Baranauskas, Maria Cecília Calani. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: Developing user interfaces for portable devices with declarative languages and Web technologies.

Palavras-chave em inglês (Keywords): 1. User interfaces (Computer systems). 2. Portable computers. 3. Internet. 4. Functional programming languages.

Área de concentração: Engenharia de Computação

Titulação: Mestre em Ciência da Computação

Banca examinadora: Profa. Dra. Maria Cecília Calani Baranauskas (IC-UNICAMP)

Prof. Dr. Ricardo de Oliveira Anido (IC-UNICAMP)

Profa. Dra. Lúcia Vilela Leite Filgueiras (Escola Politécnica - USP)

Data da defesa: 15/12/2005

Programa de Pós-Graduação: Mestrado profissional em Ciência da Computação

Linguagens Declarativas e Tecnologias da Web no Desenvolvimento de Interfaces de Usuário de Dispositivos Portáteis

Este exemplar corresponde à redação final do Trabalho Final devidamente corrigida e defendida por João Loureiro Júnior e aprovada pela Banca Examinadora.

Campinas, 15 de dezembro de 2005.

Profa. Dra. Maria Cecília Calani Baranauskas
(Orientadora)

Trabalho Final apresentado ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Computação na área de Engenharia de Computação

Agradecimentos

À Professora Maria Cecília Calani Baranauskas pela orientação, compreensão e apoio na realização deste trabalho.

À Área de Pesquisa e Desenvolvimento da Motorola Industrial Ltda pelo patrocínio e apoio na realização deste trabalho.

Resumo

Este trabalho enumera diversos problemas relacionados ao desenvolvimento de interfaces de usuário, observados na indústria atual de dispositivos portáteis. É apresentada uma comparação entre diversas linguagens declarativas relacionadas à especificação de interfaces de usuário, todas baseadas em XML. É feita então uma proposta de uso da linguagem XForms, conjuntamente com CSS, na definição de interfaces de usuário destes dispositivos, como alternativa à solução vigente. Como meio de validar a proposta, é definido um modelo para a interface de usuário, tendo como base um telefone celular existente no mercado. Em seguida é descrito como XForms e CSS podem ser utilizados na especificação da interface de usuário dos dispositivos correspondentes a este modelo. São incluídas uma proposta de arquitetura e as extensões necessárias à linguagem XForms. Este trabalho demonstra a necessidade de um número reduzido de extensões para que XForms e CSS possam ser utilizados com este propósito.

Abstract

This work describes several problems related to user interface development that currently affect the industry of portable devices. We present a brief comparison between existing declarative languages based on XML related to user interface specification. We propose the use of XForms and CSS as an alternative to the current approach. In order to validate the proposal, a model for the user interface is presented, based on existing cellular phones on the market. We show how XForms and CSS could be used as user interface definition languages for the portable devices based on that model. It is included an architecture proposal and the required extensions to the W3C XForms recommendations. We conclude the feasibility of this proposal based on the small number of extensions required for that.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Este trabalho tem por objetivo investigar o uso de padrões e tecnologias da *Web*[1] no desenvolvimento de interfaces de usuário para dispositivos portáteis. Este capítulo descreve o problema, os objetivos do trabalho e a metodologia utilizada.

1.1 Síntese do Problema e Proposta de Solução

Dispositivos eletrônicos portáteis, como telefones celulares, estão em constante evolução, a qual é caracterizada pelo aumento no conjunto de funcionalidades, aparência e usabilidade da interface de usuário, como descrito em [17]. Esta evolução é motivada por diversos fatores, como novas tecnologias, concorrência de mercado e exigências específicas de certos clientes, sendo normalmente acompanhada de um aumento na capacidade técnica dos dispositivos, como quantidade de memória, tamanho da tela e quantidade de cores, velocidade de processamento, etc.

As alterações necessárias no código destes dispositivos para suportar esta evolução são, em geral, difíceis de serem implementadas, consumindo grande tempo e esforço. De acordo com [15], "o código das interfaces de usuário é grande, complexo e dificil de ser implementado, representando aproximadamente 48% de todo o código, e necessitando por volta de 50% do tempo gasto na implementação dos programas". Para agravar mais esta situação, a constante busca por uma maior facilidade de uso das interfaces de usuário, como descrito em [16], implica em uma complexidade do código cada vez maior. Diversos fatores contribuem para isso, como obervado pelo autor ao trabalhar na área de desenvolvimento de *software* para a indústria de telefones celulares:

- A adição de novas funcionalidades normalmente requer a implementação, teste e manutenção do código responsável pela criação e gerenciamento dos elementos da interface de usuário, como campos de entrada, visualização e ação.
- Mudanças profundas na interface de usuário exigem grande re-trabalho no código das aplicações já existentes, devido ao alto acoplamento entre estes componentes, mesmo para o caso das funcionalidades permanecerem inalteradas. No caso de uma mudança radical na interface de usuário, o reuso de código pode ser muito pequeno.

Este problema da falta de separação entre a interface de usuário e a aplicação é também reconhecido em [5].

- A forma como as alterações na interface de usuário são especificadas para os programadores é freqüentemente ambígua e incompleta, exigindo grande esforço de revisão e re-trabalho posterior do código. A validação das alterações exige a execução de testes de usabilidade por meio de simuladores, nem sempre disponíveis quando as mudanças na interface de usuário são significativas. A necessidade de um maior formalismo na especificação da interface de usuário é sugerida em [5], o que viabilizaria formas de criação de interfaces de usuário sem a necessidade de programadores.
- São necessários programadores com experiência no código para realizar as alterações de forma eficiente e com pouco risco de introduzir defeitos, porém isso nem sempre é possível. Este problema é citado em [19]. A necessidade de cada vez mais facilitar a programação e customização dos sistemas por não programadores é reconhecida em [17].
- Os requisitos tendem a usar o máximo da capacidade do dispositivo, incluindo mais e mais funcionalidades e melhorias na interface de usuário, o que leva a problemas de falta de memória e de desempenho.
- Muitas vezes uma grande quantidade de mudanças deve ser realizada em prazos muito curtos ou com recursos insuficientes, o que reduz a qualidade do código, aumentando o risco de defeitos e o esforço de manutenção.

Podemos classificar as mudanças no *software* dos dispositivos, decorrentes desta evolução, da seguinte forma:

1. Mudanças nas funcionalidades do dispositivo.
2. Mudanças no fluxo e composição das telas.
3. Mudanças no tipo e comportamento dos elementos de interface de usuário.
4. Mudanças na apresentação gráfica dos elementos de interface de usuário.

As funcionalidades do dispositivo podem ser vistas como sendo independentes, ou separadas, da representação ou interação permitida pela interface de usuário. Por exemplo, um dispositivo pode não ter nenhuma interface de usuário e mesmo assim permitir realizar todas as suas funcionalidades utilizando um programa de testes (Item 1).

[1] O termo "Web" é utilizado neste trabalho para significar o conjunto de padrões utilizados na Internet.

A adição de novas funcionalidades ao dispositivo em geral acarreta mudanças no fluxo e composição das telas do dispositivo. Por exemplo, novos campos de visualização e entrada de dados são adicionados e novas ações passam a ser suportadas. Isso não necessariamente exige a criação de novos elementos na interface de usuário, ou mudanças na forma como ocorre a interação com os elementos existentes (Item 2).

Maior capacidade de processamento, resolução e cores de tela e novos dispositivos de entrada podem permitir melhorar a apresentação dos elementos na interface de usuário e até mesmo introduzir tipos novos, por exemplo controles de manipulação direta, com telas sensíveis ao toque (Item 3).

Os elementos da interface de usuário podem ser representados de diversas formas, e também suportar diversas maneiras de interação com o usuário. A capacidade de alterar a forma de apresentação gráfica dos elementos usados na interface de usuário é conhecida como "estilos", "temas" ou *skinning*. A apresentação dos elementos da interface de usuário pode assim ser considerada independente das informações manipuladas por eles. É possível que um dispositivo suporte várias formas de apresentação gráfica, as quais podem ser trocadas dinamicamente pelo usuário final (Item 4).

Este estudo propõe o uso de linguagens declarativas na definição[2] de interfaces de usuário para dispositivos portáteis que permitam realizar mudanças no fluxo e composição das telas (Item 2) e mudanças na apresentação gráfica dos elementos (Item 4) sem a necessidade de alterações de código. Algumas vantagens do uso de modelos declarativos na definição de interfaces de usuário são descritos em [3], [20] e [24], dentre as quais destacamos as seguintes:

- Trazem maior agilidade ao desenvolvimento e facilidade de manutenção.
- Facilitam a criação de ferramentas de desenvolvimento automatizado.
- Representam o conhecimento de forma explícita, aumentando a consistência e reuso.
- Suportam o desenvolvimento iterativo e centrado no usuário.

A necessidade de maior separação entre a interface de usuário e a lógica da aplicação

[2] O termo "definição" (do inglês *definition*) é utilizado neste trabalho para significar uma especificação em formato capaz de ser utilizado diretamente por ferramentas ou programas.

que não depende dela, por meio do uso de linguagens declarativas na definição de interfaces de usuário, é discutida em [11] e [19], sendo citado como grande benefício o fato de permitir que programadores se concentrem nas áreas que realmente exigem o desenvolvimento de código, enquanto que especialistas em usabilidade e provedores de conteúdo gráfico podem trabalhar diretamente no desenvolvimento da interface de usuário. Além disto, esta separação permite o reuso da lógica da aplicação com diferentes interfaces de usuário.

1.2 Objetivos e Metodologia

O objetivo deste trabalho é investigar formas de reduzir os problemas aqui relatados, que afetam atualmente a industria de dispositivos portáteis, mais especificamente telefones celulares. Para tanto, propomos o uso de linguagens declarativas na definição de interfaces de usuário, com as seguintes considerações:

- Viabilizar o uso de tecnologias da Internet relacionadas a interfaces de usuário, permitindo o aproveitamento de *software* aberto e ferramentas desenvolvidas para estes padrões, além de estabelecer uma base comum entre o código responsável pela interface de usuário e o dos navegadores de Internet, atualmente presentes na maioria dos dispositivos aqui considerados.
- Aumentar a flexibilidade e facilidade na definição da inteface de usuário, por meio da especificação formal dos requisitos, permitindo que ela seja rapidamente criada ou modificada por clientes e especialistas em usabilidade, que normalmente não possuem conhecimentos em programação ou na arquitetura do dispositivo.
- Desacoplar os elementos da interface de usuário e as funcionalidades do dispositivo, permitindo que a interface de usuário possa ser criada e modificada sem a necessidade de alterações de código.
- Permitir a configuração dinâmica das propriedades de apresentação dos controles da interface de usuário (*skinning*), permitindo que detalhes gráficos de apresentação sejam facilmente modificados de acordo com o tema escolhido tanto pelo usuário final como pelo *designer* da interface de usuário.
- Facilitar o uso de ferramentas de auxilio a especificação, prototipagem e geração automática da interface de usuário, acelerando o desenvolvimento e dando suporte antecipado aos testes de usabilidade da interface de usuário do dispositivo.
- Restringir as alterações de código às funcionalidades do dispositivo, de forma independente da especificação do fluxo e dos elementos da interface de usuário,

reduzindo com isso o número de defeitos introduzidos pelas mudanças.

O escopo deste trabalho não é focado no suporte a multi-modalidade[3], portabilidade e plasticidade[4], porém entende que o uso de controles abstratos possibilita uma maior flexibilidade na criação de diferentes estilos e modalidades de interface.

Para atingir estes objetivos, inicialmente foi feito um estudo comparativo de diversas linguagens de definição de interfaces de usuário existentes, dentro do conjunto daquelas baseadas em XML[5], porém não restrito ao uso em dispositivos portáteis. Em função das considerações feitas, foram destacados, dentre estas linguagens, XForms e CSS, padrões da *Web* utilizados por navegadores na apresentação de formulários. Para demonstrar a possibilidade de uso destes padrões em dispositivos portáteis, foi gerado um modelo para a interface de usuário dos telefones celulares Motorola atualmente no mercado, utilizando um método histórico de análise estática[6], além da experiência do autor no desenvolvimento de *software* para estes dispositivos. Desta modelagem resultaram os seguintes artefatos:

- Modelo de arquitetura do *software* atualmente utilizado.
- Modelo de tarefas representando o fluxo entre as diversas telas resultante das ações do usuário.
- Diagramas de classe descrevendo os componentes da interface de usuário.

Em seguida foi feito um mapeamento deste modelo para os padrões XForms e CSS, observando a necessidade de algumas extensões à especificação original. O caráter destas extensões leva à conclusão de que XForms e CSS seriam uma alternativa viável como linguagem declarativa para a definição da interface de usuário dos dispositivos usados no modelo.

1.3 Estrutura do Texto

Este trabalho é organizado da seguinte forma: no Capítulo 2 são apresentadas diversas linguagens declarativas para definição de interfaces de usuário baseadas em XML, e uma comparação entre elas. O Capítulo 3 propõe um modelo de interface de usuário para dispositivos portáteis baseado em telefones celulares existentes no mercado. O Capitulo 4

[3] Diferentes formas de interação com o usuário, como voz, teclado, telas sensíveis ao toque, etc.
[4] Adaptação automática da interface a contextos variáveis, sem perda de usabilidade.
[5] Os padrões XML, XForms e CSS, além de outros relacionados, são descritos no Apêndice A.
[6] O método histórico de análise estática (*Static Analysis)* é definido em [28].

analisa o uso de XForms, uma tecnologia da *Web*, como linguagem de definição de interfaces de usuário, para o modelo proposto. O Capítulo 5 contém a conclusão e sugestões de trabalhos futuros. O Apêndice A contém uma descrição das tecnologias da *Web,* relacionadas a interfaces de usuário, referenciadas neste trabalho. O Apêndice B contém um exemplo da interface de usuário de uma aplicação, de acordo com a proposta apresentada neste trabalho.

Capítulo 2

Estado da Arte em Linguagens Declarativas para Definição de Interfaces de Usuário

Para que uma linguagem possa ser utilizada no desenvolvimento de aplicações, ela deve permitir capturar não apenas os aspectos estáticos da interface de usuário, como elementos gráficos utilizados e seus respectivos mapeamentos para os dados e funcionalidades do sistema, mas também suas características dinâmicas, como fluxo de telas, restrições e mapeamento de eventos.

Como descrito em [19], linguagens imperativas, ou procedimentais, definem uma seqüência de operações que avaliam e alteram o estado do sistema até que o resultado desejado seja atingido. Elas definem exatamente como uma tarefa deve ser executada. Já as linguagens declarativas definem relações lógicas que levam ao resultado desejado. Em contraste às linguagens imperativas, elas definem "o que é" uma tarefa, e não "como" ela deve ser executada. Estas linguagens têm um maior nível de abstração do que as linguagens imperativas, podendo ser interpretadas durante a execução ou transformadas em linguagens imperativas utilizando geradores de código. Com isso é possível obter resultados que se adaptam a características e estados do sistema, sem que a definição tenha que ser alterada.

O uso de abstrações nas linguagens declarativas permite esconder detalhes de implementação, acelerando o desenvolvimento e reduzindo a necessidade de conhecimento especializado. Outras vantagens são o aumento da portabilidade para diferentes plataformas e dispositivos e seu uso como uma camada entre as ferramentas de geração de interface e os geradores de código, como detalhado em [19].

O uso de linguagens declarativas está mais restrito à definição dos objetos apresentados na interface e os modos de interação com o usuário para realizar as tarefas suportadas pelo dispositivo. As funcionalidades básicas do sistema e a própria parte gráfica da interface de usuário são normalmente implementadas utilizando linguagens imperativas (por exemplo, C ou C++).

2.1 Ambientes de Desenvolvimento Baseados em Modelos

Existem inúmeras propostas de ferramentas para o desenvolvimento de interfaces de

usuário. Como descrito em [19], é possível classificar estas ferramentas de acordo com a forma como a interface é especificada. Uma destas formas é utilizando ferramentas baseadas em modelos, normalmente conhecidas como MB-UIDEs (*Model Based User Interface Development Environments*). O desenvolvimento baseado em modelos parte do princípio de que todas as informações necessárias à criação da interface de usuário estão representadas em modelos declarativos, também conhecidos como UIMs (*User Interface Models*). Em outras palavras, todos os aspectos relevantes da interface de usuário estão capturados em uma linguagem declarativa de modelagem, representando conceitos abstratos e concretos, como modelos de tarefas, diálogos, objetos, usuários, comportamento, apresentação, etc.

As vantagens proporcionadas por esta abordagem, de acordo com [20], são:

- Provêem uma descrição mais abstrata da interface de usuário que outras ferramentas de desenvolvimento existentes atualmente.
- Facilitam a criação de métodos para o desenvolvimento de interfaces de forma sistemática, por meio do uso de diferentes níveis de abstração, reuso de especificações e processos interativos de refinamento.
- Provêem a infraestrutura necessária para automatizar as tarefas relacionadas ao desenvolvimento e implementação de interfaces.

De acordo com [24], as seguintes características são necessárias para que um sistema de desenvolvimento de interfaces seja considerado baseado em modelos:

- Deve incluir um modelo declarativo, abstrato, e de alto nível sobre o sistema interativo a ser desenvolvido.
- Deve explorar uma relação clara e capaz de ser suportada por computadores entre os modelos e a interface concreta.

Estes critérios excluem a maior parte das ferramentas de desenvolvimento de interfaces existentes (*GUI Builders*). As linguagens declarativas analisadas neste trabalho também não seriam, por si só, sistemas de desenvolvimento baseados em modelos, mas apenas representariam modelos de interfaces.

Uma crítica aos sistemas baseados em modelos, feita em [6], é que eles são eficientes apenas em domínios restritos, como formulários. Além disso existe uma dificuldade em

juntar os conceitos abstratos aos concretos, criando um problema de mapeamento (*Mapping Problem*), como descrito em [21].

2.2 Linguagens Baseadas em XML

Uma outra forma de especificar interfaces de usuário, ainda de acordo com [19], é usando linguagens declarativas. Existem atualmente inúmeras destas linguagens, porém neste estudo restringimos o escopo apenas àquelas baseadas em XML, em função deste ser um padrão da Internet, suportado por inúmeras implementações de leitores (*parsers*), além de outras vantagens mencionadas em [19].

2.2.1 XIML

XIML (*Extensible Interface Markup Language*) possui uma especificação fechada, definida pelo XIML Forum, cujos membros são formados por empresas que detém seu direito de uso comercial. XIML é uma linguagem multi-plataforma para definição de interfaces universais. Tem suas origens em trabalhos de ontologia e de especificação de interfaces, como explicado em [22].

As principais características de XIML são:

- Independente de plataforma ou dispositivo.
- Suporte a ferramentas de desenvolvimento e sistemas baseado em modelos.
- Repositório central de dados, contendo toda a informação necessária à definição da interface de usuário.
- Separação entre o modelo abstrato de definição da interface de usuário e a forma concreta de apresentação, visando suporte a multiplas plataformas e dispositivos.
- Suporte à reorganização dinâmica da apresentação (plasticidade), baseado em características e estado do sistema, ou dados de configuração.
- Suporte à distribuição (*download*) de novos componentes para a interface de usuário.
- Suporte à modelagem de tarefas, por meio de conceitos abstratos.

A linguagem XIML tenta definir e relacionar todos os elementos relevantes da interface de usuário, sendo composta dos seguintes componentes:

- Tarefa (*Task*) – Decomposição hierárquica em tarefas e subtarefas e o fluxo

entre elas. Foi Inspirado nas ferramentas de modelagem de tarefas de usuário feitas para MOBI-D e U-TEL[7]. Este componente define tarefas de usuário, objetos do domínio e perfis de usuário.

- Domínio (*Domain*) – Hierarquia (ontologia) de objetos e classes manipulados pelo usuário.
- Usuário (*User*) – Hierarquia de usuários. Contém as características dos indivíduos ou grupos que usam a interface de usuário.
- Dialogo (*Dialog*) – Coleção estruturada de elementos que definem as ações entre o usuário e o sistema, e os fluxos da interface. É baseado em tarefas abstratas de usuário.
- Apresentação (*Presentation*) – Hierarquia de elementos de apresentação visual (*widgets*), táctil (*haptics*), auditiva (sons), etc, representados em alto nível. Este componente define objetos abstratos de interação (AIO), neutros em relação à plataforma, e o mapeamento para objetos concretos de interação (CIO), que representa uma *widget* numa certa plataforma.

A linguagem XIML possui ainda os seguintes conceitos:

- Relações – Ligação entre dois ou mais elementos/atributos, sejam do mesmo componente ou não. Por exemplo, o dado "A" é mostrado pelo elemento de apresentação "B" (relação entre componente de domínio e de apresentação).
- Atributos – Propriedades dos elementos para as quais podem ser atribuídos valores (tipos básicos ou outros elementos existentes).

XIML é uma tentativa de padronizar uma linguagem para suporte a interfaces universais. De acordo com [10], os trabalhos nesta linguagem parecem ter sido descontinuados.

2.2.2 UIML

UIML (*User Interface Markup Language*) é uma meta linguagem de definição de interfaces de usuário, capaz de suportar vários vocabulários, dependendo da plataforma utilizada (por exemplo, Swing/AWT, Qt, Visual Basic, etc). Foi originalmente proposta por pesquisadores da universidade Virginia Tech, como descrito em [19], e desenvolvida pela empresa Harmonia Inc., como parte da organização UIML.org. Atualmente sua

[7] MOBI-D é um ambiente de desenvolvimento baseado em modelos. U-TEL é uma ferramenta de auxílio à criação de modelos de dados e tarefas para interfaces de usuário.

padronização está sendo considerada pelo consorcio denominado OASIS, voltado para o estabelecimento de padrões na área de *e-business*. De acordo com [8], o objetivo de UIML é prover uma representação canônica de qualquer interface de usuário capaz de ser mapeada para linguagens existentes.

Como definido em [19], a interface de usuário pode ser vista como sendo composta de quatro aspectos:

- Estrutura – organização dos objetos da interface.
- Conteúdo – recursos usados na interface, como rótulos, mensagens, etc.
- Estilo – forma de apresentação dos objetos da interface.
- Comportamento – ações que podem ser executadas ao interagir com os objetos da interface.

A linguagem UIML procura separar cada um dos aspectos mencionados. Suas principais características são:

- Independência de plataforma e dispositivo, visto que é uma meta-linguagem.
- Separação entre conteúdo e apresentação.
- Mapemanento dos componentes da interface para os objetos gráficos da plataforma é feito utilizando um vocabulário apropriado.
- Definição do mapemanto para as APIs do sistema.
- Suporte para interfaces dinâmicas e multi-modalidade (por exemplo, interação por meio de voz ou teclado).
- Suporte para tratamento de ações baseado em eventos, sem necessidade de *scripts*.

A linguagem UIML, por si só, não tem suporte para componentes abstratos, necessitando de um vocabulário específico para o domínio em que está sendo usada. Uma possibilidade seria a criação de um vocabulário genérico, permitindo abstrair os controles da interface e posteriormente mapeá-los para uma apresentação concreta, de acordo com a plataforma.

2.2.3 XAML

XAML (*Extensible Application Markup Language*) é uma linguagem declarativa usada no desenvolvimento de aplicações para o sistema operacional da Microsoft,

denominado *Vista* (anteriormente conhecido como *Longhorn*). É parte da camada denominada *Avalon*, responsável pela apresentação gráfica, como descrito em [13].

As principais características da linguagem XAML são:

- Especifica uma hierarquia de objetos, baseados em bibliotecas de classes da plataforma, além de um conjunto de propriedades e a lógica para tratamento de eventos. Os elementos da linguagem correspondem diretamente aos objetos criados durante a execução.
- Separação entre apresentação e lógica, permitindo o uso de diversas linguagens suportadas pela plataforma .NET.
- O código que implementa a lógica pode ser especificado como parte do documento XAML ou em um arquivo compilado em separado (*code-behind*).
- Suporte a modelo de dados definidos pela linguagem XML Schema.
- Normalmente compilada, mas pode ser interpretada, desde que não dependa de uma lógica além da existente nas classes da plataforma (ou seja, o arquivo XAML não contenha código). Com isso os documentos em XAML podem ser obtidos de servidores e apresentados diretamente por navegadores.

A linguagem XAML possui um mecanismo, denominado *Data Services*, que permite realizar a associação (*binding*) declarativa entre elementos da interface de usuário e os dados da aplicação, aumentando a separação entre a interface e a lógica de negócios. Os objetos do tipo `DataItem` expõem qualquer tipo de dado para a interface. Mudanças nos dados são notificadas, permitindo a atualização automática da interface. Esta associação pode ser bidirecional, possibilitando atualizar os dados do sistema a partir da interface. Além disso, os dados podem ser transformados pela associação, de acordo com o formato utilizado na interface e o mantido pela lógica.

Em XAML, os elementos correspondentes ao controles da interface refletem diretamente classes da biblioteca gráfica, e não a intenção por trás dos mesmos, dificultando seu uso com modalidades de interface não gráficas ou que exijam plasticidade. A possibilidade de obter as informações a partir de fontes de dados especificadas por meio de XML Schema é, ate certo ponto, similar aos modelos de XForms.

2.2.4 XUL

XUL (*Extensible User Interface Language*) é uma linguagem declarativa para definição de interfaces gráficas, parte do projeto *Mozilla XPToolkit*, utilizado no desenvolvimento de aplicações multi-plataforma, como descrito em [14].

As principais características da linguagem XUL são:

- Especifica um conjunto de elementos que permite criar janelas, diálogos, menus, barras de ferramentas e *widgets*.
- Permite o uso de Javascript para implementar a lógica das aplicações, alem de dar acesso a um conjunto de APIs independentes de plataforma.
- Estilos podem ser aplicados por meio de várias propriedades CSS associadas aos elementos concretos da interface de usuário.
- Integrado ao navegador Mozilla, permitindo o uso de outras tecnologias da *Web*, como HTML e SVG.

XUL é freqüentemente comparada a XAML, pois ambas são plataformas para desenvolvimentos de aplicações e voltadas principalmente para a definição de interfaces gráficas, não levando em conta aspectos como modelos de tarefas e abstração dos controles. Estas linguagens funcionam mais como uma camada de portabilidade para diferentes plataformas, como descrito em [10].

2.2.5 XForms

XForms (*XML Forms*), definida em [34], é uma recomendação do W3C[8] de uma linguagem de formulários da *Web*, a qual pode ser usada com os mais variados tipos de equipamentos, como computadores de mesa e dispositivos portáteis. XForms ainda não é amplamente suportado pelos navegadores atuais, sendo mais utilizado no ambiente servidor como uma linguagem intermediária, a qual é mapeada para HTML e JavaScript quando o formulário é enviado aos clientes. Contudo, existem atualmente vários clientes XForms em desenvolvimento, incluindo suporte para o navegador *Mozilla*, além de *plug-ins* comerciais e baseados em *software* aberto (a lista completa pode ser consultada em [35]).

As principais características da linguagem XForms são:

- Independente de plataforma ou equipamento, pois especifica controles abstratos

para a interação com o usuário. Isto permite o desenvolvimento de interfaces de usuário baseado na intenção (*Intent-based UI Design*), abstraindo o mapeamento para os componentes de apresentação.

- Totalmente declarativa, baseada em eventos e restrições impostas pelo modelo de dados.
- Separação entre conteúdo e apresentação, utilizando modelos de dados baseados em XML Schema, ligados aos controles da interface por meio de expressões XPath.
- Padrão da Internet suportado por navegadores, permitindo o uso de propriedades CSS para definição do estilo da interface gráfica, ou mesmo de outras tecnologias de apresentação, como SVG.

De acordo com [1], a linguagem XForms tem como objetivo, desde sua concepção, o suporte a dispositivos portáteis, porém pode não ser capaz de atender todos os requisitos necessários à definição de uma interface de usuário. Como sugerido em [19], existe um conjunto limitado de interfaces de usuário que podem ser descritas pela linguagem XForms. Um dos objetivos deste trabalho é justamente mostrar que as interfaces atuais dos dispositivos portáteis aqui analisados podem estar dentro deste conjunto, com pequenas extensões à linguagem XForms.

2.2.6 AUIML

AUIML (*Abstract User Interface Markup Language*) é uma linguagem declarativa, definida pela IBM, que permite descrever a semântica da interação com o usuário, independente do dispositvo ou meio de apresentação. Esta linguagem é implementada por uma ferramenta baseada no ambiente Eclipse, a qual pode ser obtida em [9]. Os objetivos que levaram a criação de AUIML foram reduzir a necessidade de conhecimento em programação, permitir o uso de ferramentas e acelerar o desenvolvimento de interfaces, como indicado em [12]. Não foi possível localizar uma especificação formal da linguagem.

As principais características da linguagem AUIML são:

- Independente de plataforma ou equipamento, definindo elementos abstratos de interação com o usuário. Especifica a intenção por trás de cada interação, e não sua aparência.
- Modelo de dados baseado em tipos simples, como data, hora, cadeias de

[8] Consorcio responsável pelo desenvolvimento de tecnologias para a Web.

caracteres e números, utilizado para entrada e saída de dados.

- Define elementos estruturais, como grupos e tabelas, e de apresentação, como títulos e textos de ajuda.
- Apresentação controlada pelos tipos de dados e propriedades associadas a eles.
- Interação abstraida por meio de eventos e as ações correspondentes.
- Ferramentas permitem realizar o mapeamento para apresentações concretas, como *Swing* e HTML.

Similar a XForms em termos de abstração de controles da interface de usuário. Apesar de possuir um modelo de dados, diferentemente de XForms, ele é bastante simples e não totalmente separado dos controles da interface de usuário. De acordo com [10], os trabalhos nesta linguagem não tem evoluido.

2.2.7 WML

Como descrito em [19] e [25], as restrições impostas pelos equipamentos portáteis, como tela pequena, teclado reduzido e baixa capacidade de processamento, impede que as metáforas utilizadas em interfaces de usuário para computadores de mesa, como a de janelas, sejam aplicadas diretamente. A metáfora de cartões (*Deck of Cards*) é adequada aos dispositivos portáteis, pois quebra as tarefas em uma seqüência de telas simples, onde somente uma é visível por vez, permitindo ao usuário avançar e retroceder entre elas. Um exemplo de linguagem de interface de usuário baseada em cartões é WML (*Wireless Markup Language*), padronizada pelo WAP Forum (OMA) e utilizada em navegadores WAP para telefones celulares.

As principais características da linguagem WML são:

- Voltada para navegadores WAP.
- Utiliza JavaScript.
- Não separação entre conteúdo e apresentação.
- Ausência de um mecanismo para internacionalização dos rótulos utilizados.
- Contexto limitado à tela atual.
- Modelo de dados limitado a variáveis.
- Controles de interface limitados.

WML é uma linguagem bastante simples, com várias limitações. Estas limitações dificultam o uso de WML como uma linguagem de definição de interfaces de usuário.

2.2.8 Considerações Finais

A Tabela 2.1 a seguir sumariza as características das diversas linguagens descritas anteriormente relevantes a este trabalho:

Linguagem	Totalmente declarativa	Utilizada em navegadores	Modelo de dados	Independente de plataforma	Responsável pelo padrão
XIML	Sim	Não	Sim	Sim	XIML Forum
UIML	Sim	Não	Não	Sim	UIML.org[9]
XAML	Não	Sim	Sim	Não	Microsoft
XUL	Não	Sim	Não	Não	Mozilla
XForms	Sim	Sim	Sim	Sim	W3C
AUIML	Não	Não	Sim	Sim	IBM
WML	Não	Sim	Não	Sim	OMA

Tabela 2.1 - Características das Linguagens

O número de linguagens baseadas em XML para definição de interfaces de usuário existentes atualmente é grande. Este trabalho é focado no uso de XForms, em função de ser um padrão da Internet, ser totalmente declarativa, e constituir uma base comum com navegadores. Algumas destas linguagens, como XAML e XUL, são parte de plataformas voltadas para o desenvolvimento de aplicações para computadores de mesa, não sendo diretamente aplicáveis a dispositivos portáteis. XIML vai muito além de uma simples linguagem de definição de interfaces de usuário, porém é definida por uma organização fechada, com uso comercial restrito a seus membros. AUIML tem características semelhantes a XForms, porém apresenta um modelo de dados e um conjunto de controles limitados, além de não ter uma especificação formal disponível. UIML, por ser uma meta-linguagem, necessitaria a criação de um vocabulário próprio adequado. Além disso, ela ainda não é reconhecida como um padrão. Neste sentido XForms é uma especificação mais completa. Recentemente UIML tem avançado na direção de padronizar um vocabulário genérico, como o de XForms.

O próximo capítulo descreve um modelo de interface de usuário para dispositivos portáteis, baseado em telefones celulares existentes no mercado. Este modelo será subsequentemente utilizado na validação de XForms como uma linguagem de definição para a interface de usuário destes equipamentos.

[9] Em processo de aprovação como padrão OASIS.

Capítulo 3

Modelo Proposto para Interfaces de Usuário de Dispositivos Portáteis

A linguagem XForms é voltada para a visualização e entrada de dados por meio de formulários, em ambiente cliente-servidor. Para que seja possível validar seu uso como linguagem de definição de interface de dispositivos portáteis, torna-se necessário estabelecer modelos para a interface destes equipamentos.

Este capítulo define um modelo baseado em um telefone celular comercial. O modelo descrito foi estabelecido utilizando um método histórico de análise estática, realizando a inspeção de um conjunto representativo de telas das aplicações destes equipamentos, além do conhecimento do autor na arquitetura do *software*. Este modelo consiste nos seguintes artefatos:

- Modelo da arquitetura de *software* vigente.
- Modelo dos elementos usados na entrada e saída de dados, como tela e teclado.
- Diagramas de classes para os elementos das diversos tipos de telas, indicando seus relacionamentos e atributos.
- Modelo de tarefas para o fluxo das telas, em função da interação do usuário.

As telas de exemplo apresentadas neste capítulo representam um apanhado dentre várias aplicações existentes, em suas versões mais recentes, procurando cobrir uma amostra significativa dos elementos da interface de usuário.

3.1 Características do Dispositivo Portátil

As interfaces de usuário atualmente usadas nos dispositivos portáteis são, em geral, bastante uniformes, pois seguem um estilo imposto pelo fabricante, baseado em regras de usabilidade, e em função das limitações impostas pelo tamanho da tela e pelo teclado reduzido. As áreas da interface de usuário são bem definidas, com poucas combinações de controles em uma mesma tela. Cada tela é baseada em uma distribuição (*layout*) similar, contendo indicadores de estado e controles que permitem realizar uma funcionalidade (passo) bem específica.

Este modelo é baseado em um telefone celular Motorola, modelo V551, lançado recentemente no mercado americano em 2005. Estes equipamentos possuem teclado alfa-numérico reduzido, tecla de navegação de cinco direções e duas teclas de ação, e tela não sensível ao toque. Inúmeros detalhes sobre como os controles destes equipamentos são representados graficamente estão fora do contexto deste trabalho. É importante observar que os diversos aparelhos portáteis existentes atualmente no mercado podem apresentar diferentes características em relação à interface de usuário, principalmente se utilizarem telas sensíveis ao toque, controles por voz, ou se forem de domínios diferentes, como por exemplo sistemas de posicionamento global (GPS) ou assistentes pessoais (PDA).

3.2 Arquitetura de Software Vigente

Podemos caracterizar a arquitetura de *software* dos equipamentos utilizados como base para este modelo da forma indicada na figura 3.1, a seguir:

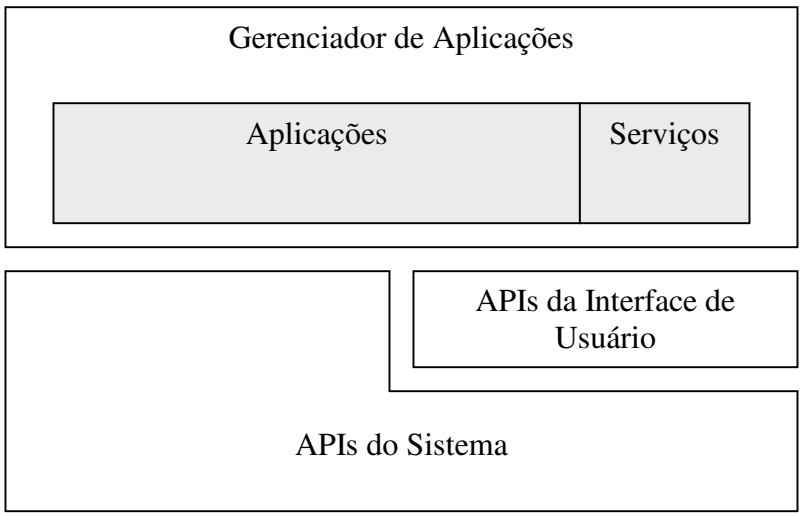

Figura 3.1 - Arquitetura de Software

As aplicações são executadas dentro de um ambiente controlado pelo gerenciador de aplicações, responsável pelo ciclo de vida das aplicações e comunicação entre elas. As aplicações correspondem à lógica responsável por gerenciar os elementos da interface de usuário e pela seqüência (fluxo) das telas. Para tanto, utilizam as APIs (*Application Programing Interfaces*) da interface de usuário para criar os elementos gráficos e receber os eventos resultantes da interação do usuário com o dispositivo. A interface de usuário é responsável pela apresentação gráfica (e sonora) das informações, e pela entrada de dados e comandos do usuário. Os dados que representam o estado do dispositivo (incluindo informações armazenadas pelo usuário) são obtidos através das APIs do sistema e

transferidos para as APIs da interface de usuário pelas aplicações. As APIs do sistema são também utilizadas para executar as funcionalidades básicas do dispositivo, de acordo com as operações solicitadas pelo usuário. Neste modelo, as aplicações dependem diretamente das APIs do sistema e da interface de usuário, sendo afetadas tanto por mudanças de funcionalidade quanto de apresentação.

É importante notar que as aplicações podem também prestar serviços umas às outras, permitindo que uma parte de uma aplicação se torne disponível dentro do contexto de outra aplicação.

3.3 Modelo de Tarefas

A interface de usuário destes dispositivos pode ser definida como um conjunto de tarefas, compostas das operações necessárias para que elas sejam realizadas, como visualizar dados, fornecer informações, tomar decisões, etc. Métodos de análise de tarefas podem ser utilizados durante a especificação da interface de usuário para melhor entender o funcionamento do sistema e o fluxo de informações, identificando e decompondo as tarefas necessárias, e alocando-as de forma adequada, de acordo com a ordem hierárquica (pré-requisito) entre elas. Estes métodos são explicados em [7] e [26].

Dispositivos portáteis, em geral, utilizam uma metáfora diferente da utilizada por computadores de mesa, em função das limitações de tecla e teclado, como já mencionado na sessão 2.2.7 . A interface de usuário dos dispositivos utilizados como base para este modelo utiliza uma variante da metáfora *deck of cards*, porém mais elaborada, pois permite a criação de tarefas. A simplicidade desta metáfora é umas das razões pela qual foi proposto o uso de XForms como uma linguagem de definição da interface de usuário.

Uma tarefa representa os passos que o usuário deve executar para realizar uma determinada operação. Tarefas podem ser decompostas de forma hierárquica, cada uma representando um conjunto de passos da tarefa maior. No contexto deste estudo, uma tarefa é caracterizada por uma seqüência de uma ou mais telas e as ações correspondentes que levam de uma tela a outra, permitindo ao usuário atingir um determinado objetivo. Neste modelo, cada tarefa é tratada como uma entidade independente, contendo um contexto, o qual permance ativo enquanto as subtarefas são executadas. A figura 3.2, a seguir, mostra um exemplo para o modelo de tarefas:

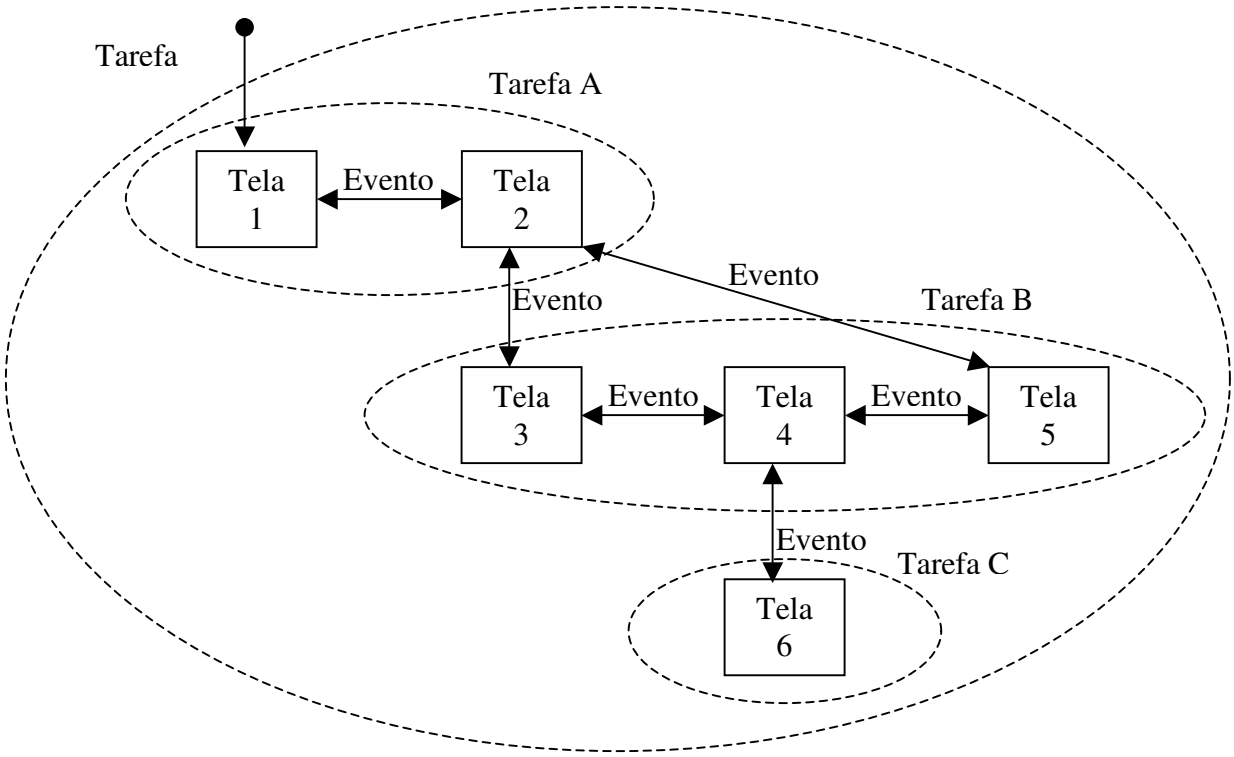

Figura 3.2 - Modelo de Tarefas

Neste exemplo, a tarefa maior é composta pelas tarefas A, B e C. Cada uma destas três tarefas é composta por uma ou mais telas. O usuário pode navegar entre estas telas através da interface de usuário, que gera os eventos correspondentes. Cada tarefa possui um contexto, válido durante o tempo em que a tarefa esta ativa, o qual permite manter o estado dos valores referenciados pelos elementos da interface de usuário. Uma tarefa pode dar início a outra, como no caso da tarefa A que inicia a tarefa B, que por sua vez inicia a tarefa C. Apenas uma tela pode estar "em foco" num determinado momento, no sentido de que o usuário pode interagir apenas com os elementos da interface de usuário que pertencem àquela tela. Ao inicar uma tarefa, o foco vai automaticamente para uma de suas telas, e ao finalizar uma tarefa o foco retorna automaticamente para a tela da tarefa imediatamente anterior.

Ao dar inicio a uma tarefa, dados passados como entrada permitem que ela adquira um contexto relativo à tarefa anterior. Por exemplo, ao selecionar uma determinada imagem de uma lista e executar a tarefa de visualização, o nome da imagem é fornecido como um dado de entrada. Da mesma forma, ao finalizar uma tarefa, dados de retorno podem ser enviados à tarefa que deu origem a ela.

3.4 Texto Formatado

O conceito de texto formatado é fundamental para a interface gráfica de dispositivos portáteis, pois permite ao usuário escolher dentre diversos idiomas. Esta capacidade é conhecida como "internacionalização" e "localização", normalmente abreviados como i18n e L10n, respectivamente. Em geral, as cadeias de caracteres apresentadas nos diversos componentes da tela são formadas por identificadores, parâmetros e caracteres de formatação. Os identificadores permitem especificar um rótulo ou mensagem (*prompt*) nos diversos idiomas suportados pelo dispositivo. Os parâmetros correspondem a valores numéricos, cadeias de caracteres, valores monetários, datas, horas, etc, associados a variáveis de estado ou informações armazenadas no dispositivo. Os caracteres de formatação indicam o alinhamento, tipo de fonte utilizado, formato dos parâmetros, ícones em linha, etc.

Identificadores e parâmetros podem ser combinados de acordo com uma certa ordem, apropriada para a linguagem selecionada no momento, formando uma cadeia de texto que pode ser apresentada para o usuário. Esta combinação é definida por uma cadeia de formatação, a qual indica a posição relativa entre os diversos identificadores e parâmetros, de acordo com a língua selecionada.

As diversas cadeias de caracteres correspondentes a cada identificador, nas línguas suportadas pelo dispositivo, são armazenadas em estruturas de dados. Quando o código da aplicação necessita mostrar um rótulo ou mensagem para o usuário, ele solicita ao sistema que forneça o valor da cadeia de caracteres correspondente, na língua selecionada naquele momento. A figura 3.3, a seguir, exemplifica este mecanismo:

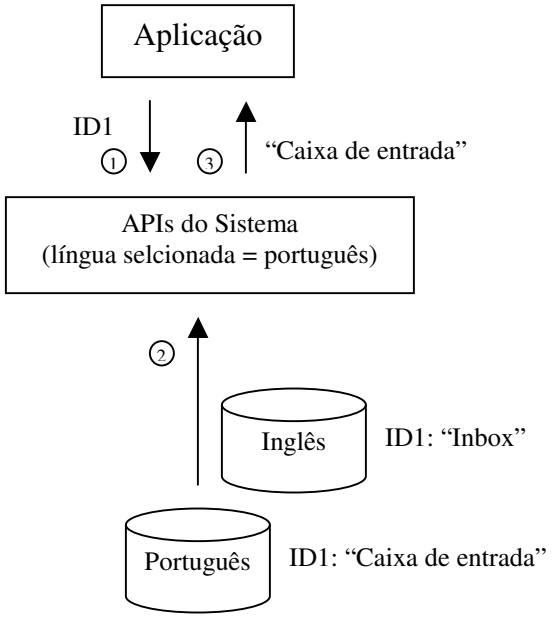

Figura 3.3 - Identificadores de Texto

De forma similar, as cadeias de formatação também estão armazenas em estruturas de dados, tendo uma versão para cada língua suportada pelo dispositivo. O sistema provê à aplicação APIs para converter parâmetros em uma cadeia de caracteres apropriada, de acordo com a cadeia de formatação e a língua selecionada pelo usuário. A figura 3.4, a seguir, exemplifica o processo de conversão do valor de uma variável em um texto formatado:

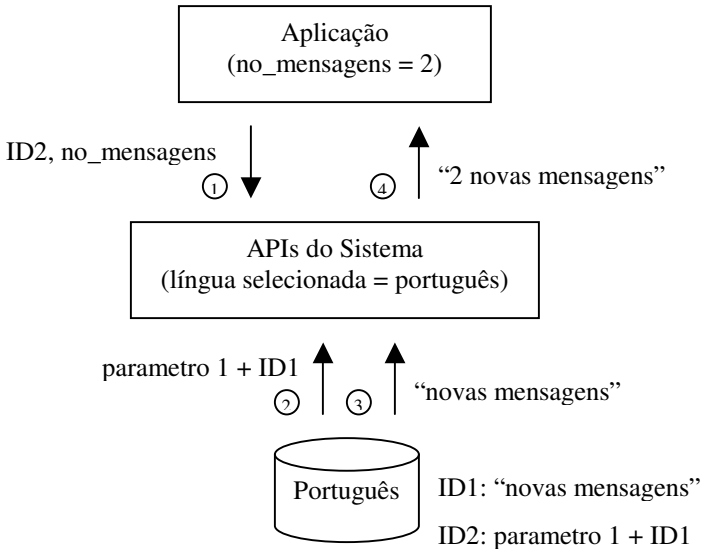

Figura 3.4 - Cadeias de Formatação

Neste exemplo, ID2 é um identificador de cadeia de formatação e "no_mensagens" é uma variável. O conteúdo da cadeia de formatação ID2 indica que a cadeia de caracteres do

identificador ID1 deve ser concatenada ao valor do parâmetro 1, que no caso corresponde a variável “no_mensagens”. O resultado da API do sistema é uma cadeia de caracteres com a mensagem que deve ser mostrada ao usuário. Este mecanismo permite tornar independente do código a forma de apresentação do texto de mensagens, títulos de telas e nomes de campos.

3.5 Elementos da Interface de Usuário

Na figura 3.5, a seguir, estão representados os diversos componentes da interface de usuário do dispositivo. Na parte superior estão os componentes da tela e na parte inferior estão as diversas teclas utilizadas na seleção e entrada de dados:

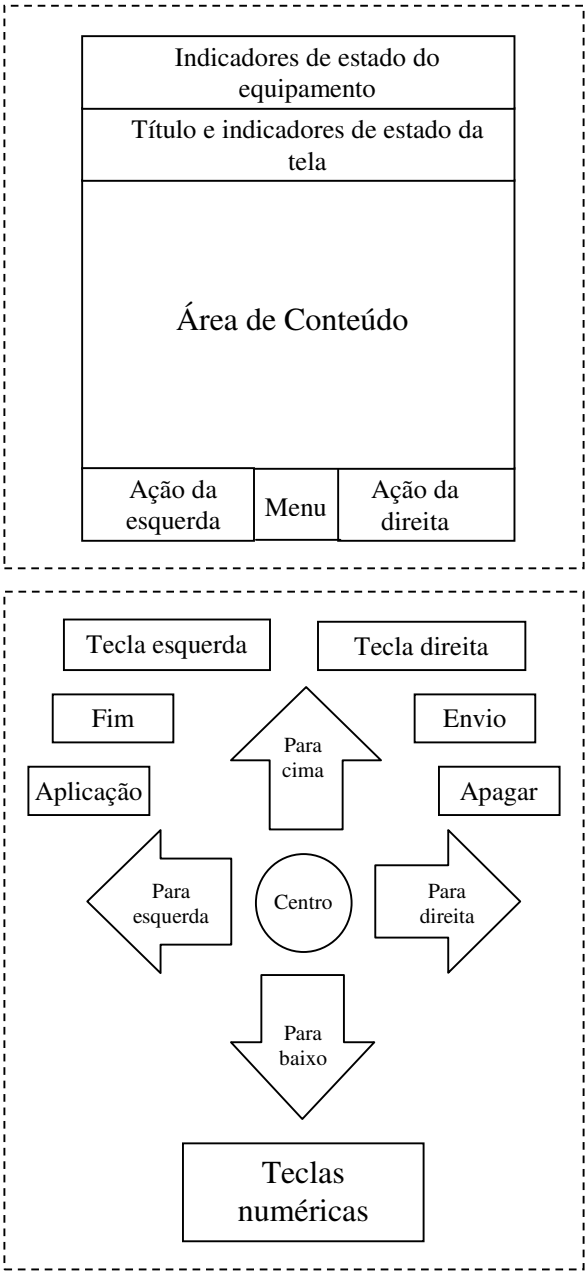

Figura 3.5 - Componentes da Interface

3.5.1 Teclado

As teclas são o mecanismo por meio do qual o usuário consegue interagir com o dispositivo, gerando eventos que são processados pelos elementos da interface de usuário ou pela aplicação. Os componentes do teclado, indicados na Figura 3.5, são:

- Teclas Esquerda e Direita – estão associadas, respectivamente, às ações

esquerda e direita indicadas pelos rótulos na parte inferior da tela. Ao serem pressionadas, permitem executar a ação correspondente.

- Tecla Menu – permite mostrar um menu de opções contextuais, caso disponível para aquela tela.
- Teclas de Navegação e Ação Central – quatro teclas direcionais e uma central de seleção e ação. As teclas direcionais são utilizadas para movimentar o cursor nos editores ou o item em destaque nas listas e menus, ou ainda como atalhos na tela inicial. A tecla central é usada para executar uma ação contextual relacionada ao item em destaque, normalmente a mais importante.
- Teclas de Fim e Envio – teclas especiais do dispositivo, utilizadas normalmente para inicar e terminar uma chamada.
- Tecla Aplicação – utilizada como um atalho para inicar uma determinada aplicação, por exemplo câmera ou mensagens.
- Tecla Apagar – quando utilizando um editor de texto, esta tecla permite apagar o caractere à esquerda do cursor. Em certos casos ela é usada para retornar a tela anterior.
- Teclado Numérico – é utilizado para entrada de números e caracteres nos editores. Os caracteres alfabéticos podem ser inseridos através das teclas numéricas, desde que os editores estejam configurados com um método de entrada alfabético.

3.5.2 Tela

Uma tela contém os diversos componentes gráficos utilizados na visualização e entrada de dados. Com poucas exceções, a tela é composta das regiões indicadas na parte superior da figura 3.5. A tela pode ser representada através do diagrama de classes da Figura 3.6, a seguir:

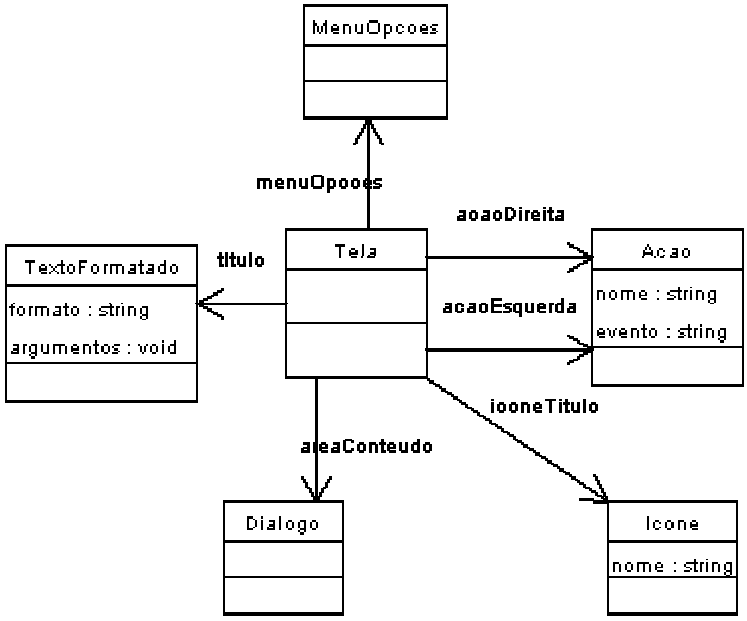

Figura 3.6 - Modelo para Tela

As classes apresentadas nesse diagrama são:

- `Tela` – representa a tela do dispositivo, com seus diversos componentes.
- `Dialogo` – classe base para os diversos tipos de diálogos que podem ser associados à área de conteúdo da tela, como listas, formulários, etc.
- `MenuOpcoes` – conjunto de ações contextuais disponíveis naquela tela.
- `Acao` – representa uma ação que o usuário pode executar.
- `Icone` – imagem gráfica.
- `TextoFormatado` – texto no idioma selecionado, contendo identificadores de textos e ícones em linha, além de seqüências de formatação, como alinhamento, tipo de fonte, etc.

Os atributos e relacionamentos de cada classe estão descritos na Tabela 3.1, a seguir:

Classe	**Atributo/Relac.**	**Descrição**
Tela	titulo	Texto formatado correspondente ao título da tela.
	acaoEsquerda	Ação correspondente à tecla esquerda. Normalmente utilizada para executar a operação mais comum daquela tela.

	acaoDireita	Ação correspondente à tecla direita. Freqüentemente utilizada para cancelar a operação ou retornar à tela anterior.
	iconeTitulo	Ícone associado à tela.
	areaConteudo	Indica o diálogo presente na área de conteúdo da tela.
	menuOpcoes	Menu de opções associado à tela.
Acao	nome	Texto correspondente à ação.
	evento	Nome do evento que deve ser gerado quando a ação for selecionada.
Icone	nome	Nome do arquivo que contém a imagem correspondente ao ícone.

Tabela 3.1 - Atributos de Tela, Acao e Icone

No exemplo ilustrado da Figura 3.7, a seguir, o título da tela, em inglês, é “Settings”, com um ícone representativo a sua esquerda. Logo acima do titulo se encontram, da esquerda para a direita, os ícones indicadores de estado de nível de sinal, campainha e nível da bateria. É importante notar que os indicadores de estado são comuns a todas as telas e portanto foram considerados como uma entidade independente delas. As teclas esquerda e direita estão rotuladas como “Exit” e “Select”, respectivamente. O conteúdo da tela é um diálogo do tipo lista simples:

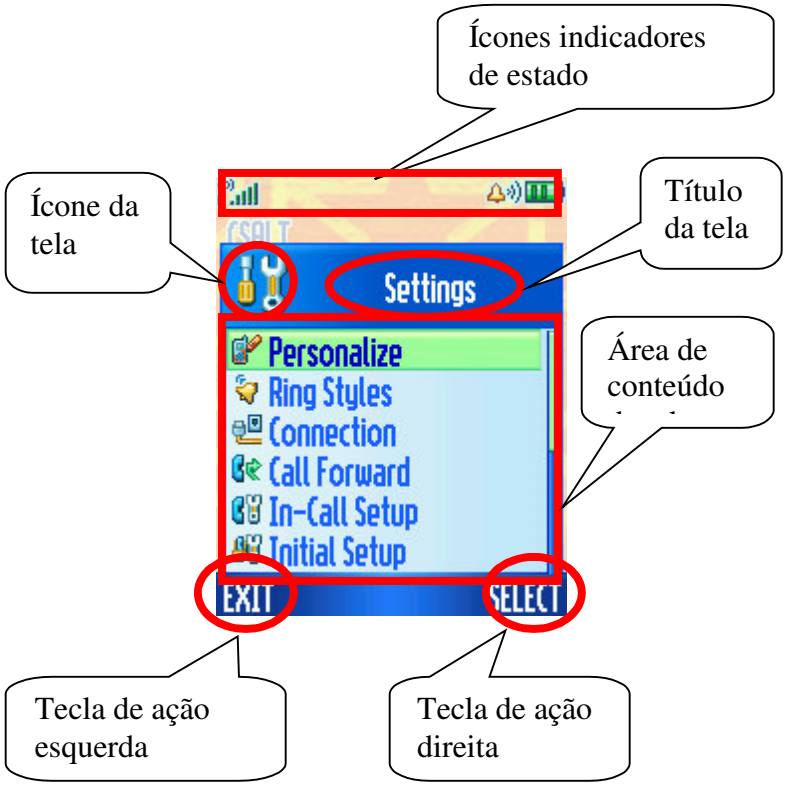

Figura 3.7 - Exemplo de Tela

É importante notar que várias telas podem existir num determinado instante, em função das tarefas que estão ativas, porém apenas uma é mostrada para o usuário (a que está em foco), como descrito na sessão 3.3 .

3.5.3 Indicadores de Estado

Ícones indicadores de estado estão sempre presentes, qualquer que seja a tela. Eles fornecem informações simples sobre o estado do dispositivo. Em função do espaço limitado, alguns indicadores compartilham a mesma região da tela, estando visíveis apenas os de maior prioridade, em função de seus estados.

Os indicadores de estado podem ser representados através do diagrama de classes representado na Figura 3.8, a seguir:

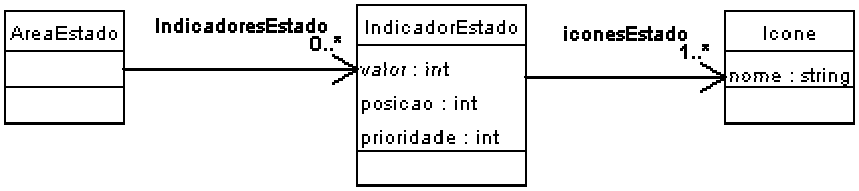

Figura 3.8 - Modelo para Indicadores de Estado

As classes introduzidas nesse diagrama são[10]:

- `AreaEstado` – representa a área no topo da tela do dispositivo que contém os indicadores de estado.
- `IndicadorEstado` – representa um determinado indicador de estado.

Os atributos e relacionamentos estão descritos na Tabela 3.2, a seguir:

Classe	**Atributo/Relac.**	**Descrição**
AreaEstado	indicadoresEstado	Conjunto de indicadores de estado do dispositivo.
IndicadorEstado	iconesEstado	Conjunto de ícones utilizados por este indicador.
	valor	Valor atual do indicador.
	posição	Posição na área de estado ocupada pelo indicador.
	prioridade	Prioridade do indicador, caso exista mais de um na mesma posição.

Tabela 3.2 - Atributos de AreaEstado e IndicadorEstado

Na parte superior da Figura 3.7, da esquerda para a direita é possível ver os indicadores de nível do sinal, tipo de campainha e nível da bateria, respectivamente. Os demais indicadores estão desativados ou tem uma prioridade inferior a estes.

3.5.4 Menu de Opções

O menu de opções contém uma lista de ações associadas à tela atual ou ao item em destaque. O menu, quando disponível numa determinada tela, pode ser ativado através da tecla correspondente. Ao ser aberto, mostra uma lista de itens, de até dois níveis, com os nomes de cada ação disponível. Uma das ações é apresentada em destaque. A ação destacada pode ser escolhida através das teclas de navegação. Algumas das ações podem estar desabilitadas, de acordo com o contexto. Uma vez aberto, a tecla da esquerda permite fechar o menu sem tomar nenhuma ação e a da direita permite ativar a ação em destaque.

O diagrama de classes da Figura 3.9, a seguir, representa o menu de opções:

[10] As classes, atributos e relacionamentos já descritos anteriormente serão omitidos deste ponto em diante.

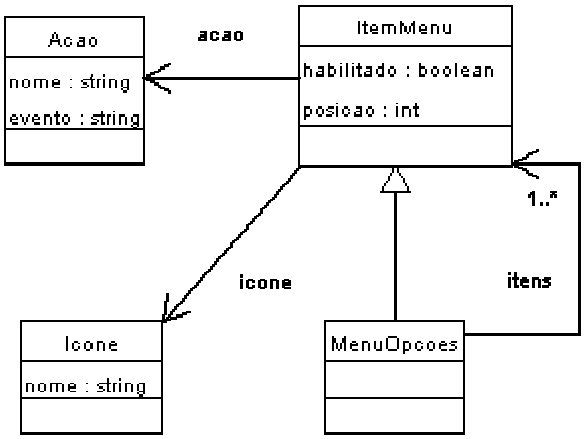

Figura 3.9 - Modelo para Menu de Opções

As classes apresentadas nesse diagrama são:

- `ItemMenu` – representa um item do menu, o qual pode ser uma entrada para um menu de segundo nível.

Os atributos e relacionamentos estão descritos na Tabela 3.3, a seguir:

Classe	**Atributo/Relac.**	**Descrição**
ItemMenu	icone	Ícone correspondente ao item do menu. Pode ser usado para indicar uma opção que esteja selecionada atualmente.
	habilitado	Indica se o item do menu esta disponível ou não.
	acao	Indica a ação correspondente ao item do menu.
	posicao	Define a posição do item no menu.

Tabela 3.3 - Atributos de ItemMenu

Quando uma opção do menu é selecionada, o evento associado à ação correspondente é gerado e o menu é fechado automaticamente. É importante notar que as opções do menu podem estar vinculadas ao contexto definido pelo conteúdo da tela. Por exemplo, no caso de uma lista de seleção, as ações do menu podem depender da entrada em destaque na lista, de forma que apenas um subconjunto das ações aplicáveis aquela entrada fique disponível para o usuário do dispositivo.

A Figura 3.10, a seguir, corresponde ao menu de opções da caixa de entrada de mensagens de texto (SMS), a qual pode ser vista ao fundo. A ação em destaque está

indicada pela barra horizontal verde sobre a ação “Read” (ler), usada para visualizar o conteúdo da mensagem. Algumas das ações possíveis não estão visíveis, como indicado pela barra de rolagem à direita. As ações “Select” (selecionar) e “Back” (voltar) são inerentes ao menu de opções, permitindo, respectivamente, selecionar a ação em destaque e fechar o menu:

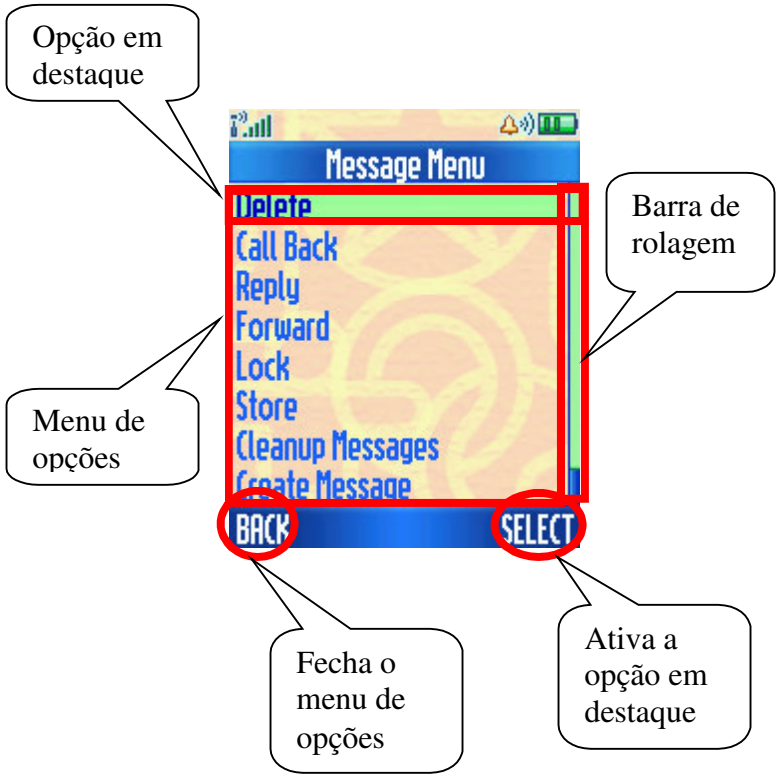

Figura 3.10 - Exemplo de Menu de Opções

3.5.5 Listas de Seleção Simples

Listas de seleção simples são utilizadas para apresentar um conjunto de itens, onde cada item contém um ou mais ícones, além de uma ou duas linhas de texto formatado. Um único item é destacado, de forma que o usuário possa realizar operações contextuais através das teclas de ação ou menu de opções. A lista pode ser “estática”, significando que os itens são sempre os mesmos, ou “dinâmica”, quando a lista pode conter um número de itens igual ou maior do que zero, de acordo com o conjunto de dados associado a ela.

O diagrama de classes da Figura 3.11, a seguir, representa a lista de seleção simples:

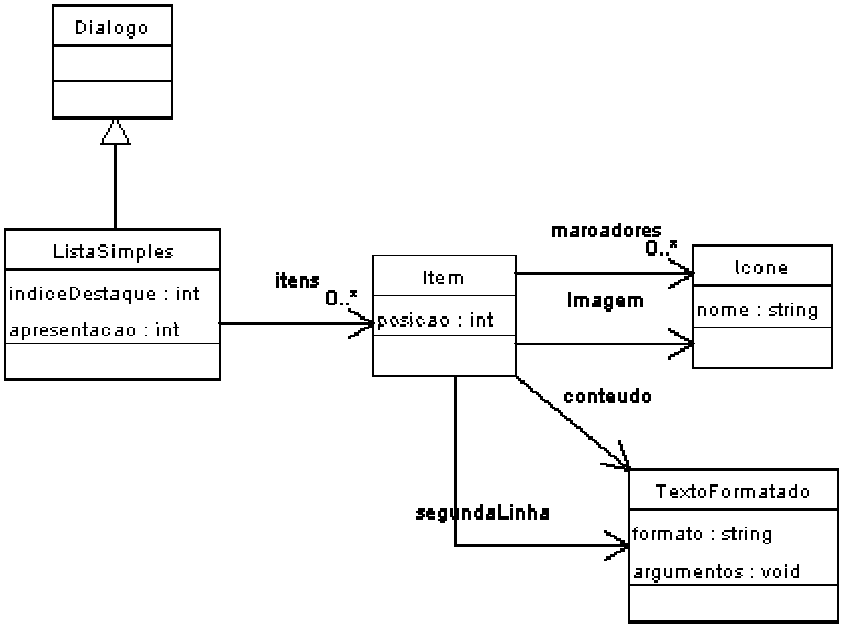

Figura 3.11 - Modelo para Lista de Seleção Simples

As classes introduzidas nesse diagrama são:

- `ListaSimples` – conjunto de itens, dos quais apenas um pode estar destacado em um determinado momento.
- `Item` – representa um item da lista.

Os atributos e relacionamentos estão descritos na Tabela 3.4, a seguir:

Classe	**Atributo/Relac.**	**Descrição**
ListaSimples	indiceDestaque	Indica qual item está selecionado.
	itens	Conjunto de itens associados à lista.
	apresentacao	Indica a forma de apresentacao da lista. Cada item pode ter apenas um texto formatado, apenas uma imagem ou duas linhas de texto formatado e uma imagem. Também especifica se filtros podem ser aplicados através das teclas '*' e '#'.
Item	posicao	Posição (índice) do item em destaque.
	conteudo	Texto formatado correspondente ao item.
	segundaLinha	Texto formatado da segunda linha do item.

	marcadores	Um ou mais ícones que indicam propriedades específicas de cada item.
	imagem	Imagem correspondente ao item.
TextoFormatado	formato	Cadeia de formatação.

Tabela 3.4 - Atributos de ListaSimples, Item e TextoFormatado

A lista de seleção simples gera os seguintes eventos:

- `navegacao` – indica que o item em destaque mudou.
- `itemSelecionado` – indica que o item em destaque foi selecionado.
- `filtroAnterior` – indica que o filtro anterior deve ser aplicado.
- `filtroSeguinte` – indica que o próximo filtro deve ser aplicado.

A Figura 3.12, a seguir, apresenta uma lista simples estática, correspondente ao menu principal de aplicações. Cada item ocupa apenas uma linha, e o item em destaque é indicado por uma barra horizontal. As teclas de navegação permitem alterar o item em destaque. Cada entrada possui um ícone e um conteúdo formatado com o nome de uma aplicação. Alguns dos itens da lista não estão visiveis, como indicado pela barra de rolagem à direita. Quando o usuário seleciona uma opção, a posição do item permite determinar a aplicação correspondente:

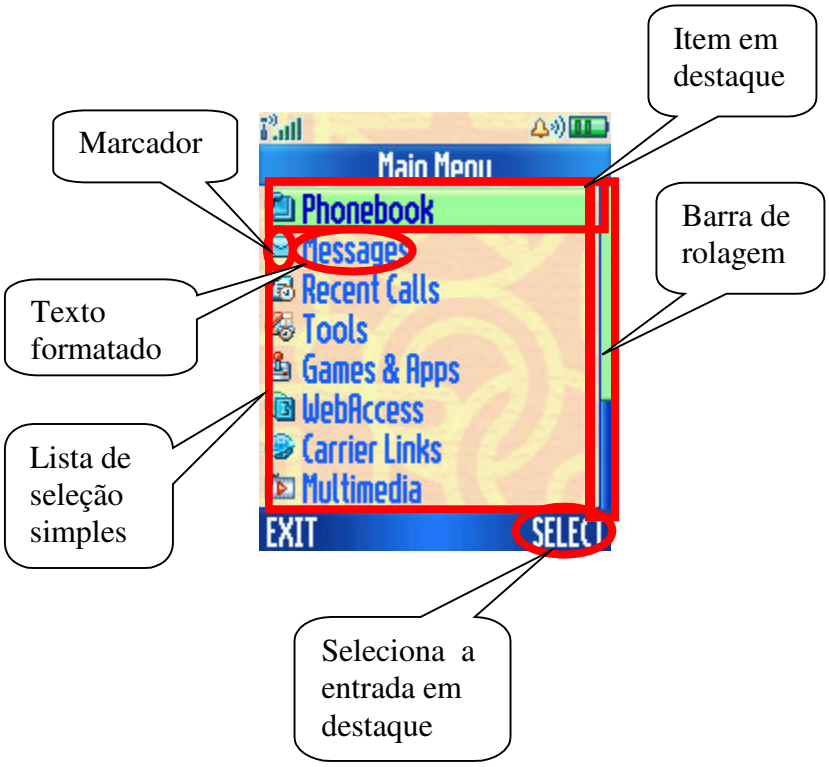

Figura 3.12 - Exemplo de Lista de Seleção Simples Estática

O exemplo da Figura 3.13, a seguir, apresenta outra lista simples estática, correspondente à tela principal da aplicação de mensagens. Nesta lista os itens não têm ícones. O item em destaque permite abrir à caixa de entrada de mensagens, contendo um texto formatado que indica a quantidade de mensagens não lidas e total, alinhadas a direita. O item imediatamente abaixo da entrada em destaque corresponde à ação de criação de uma mensagem, e os demais itens permitem abrir as pastas correspondentes aos demais tipos de mensagens:

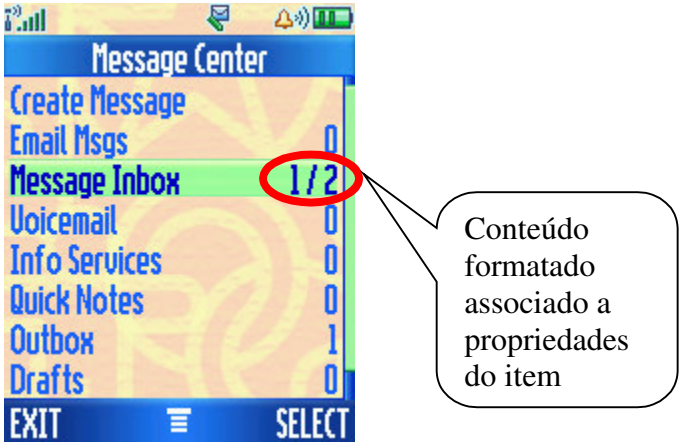

Figura 3.13 - Exemplo de Lista de Seleção Simples com Texto Formatado

O exemplo da Figura 3.14, a seguir, mostra uma lista simples dinâmica correspondente à caixa de entrada de mensagens. O primeiro item representa uma mensagem de texto (SMS). O ícone indica que se ela já foi lida e o texto mostra os primeiros caracteres do corpo da mensagem. Já o último item da lista não possui ícone, apenas um texto formatado para a ação de criação de uma mensagem:

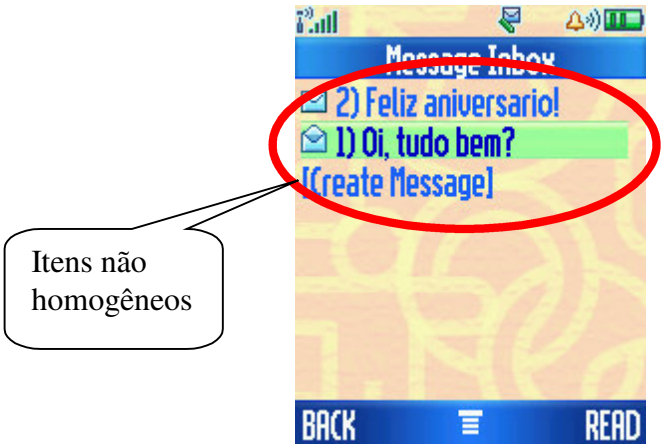

Figura 3.14 - Exemplo de Lista de Seleção Simples com Itens não Homogêneos

No exemplo da Figura 3.15, a seguir, é possível ver uma lista de imagem e texto, onde cada item tem duas linhas:

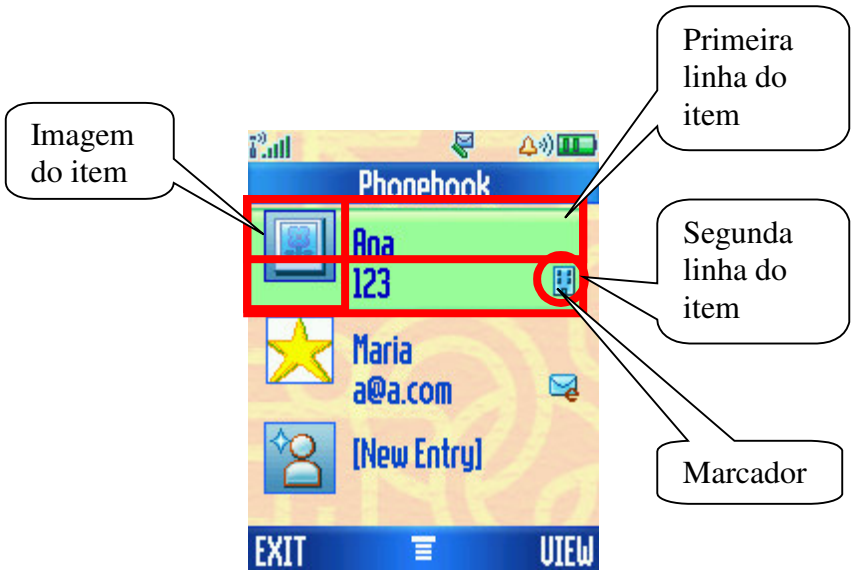

Figura 3.15 - Exemplo de Lista de Seleção Simples com Imagens

O exemplo da Figura 3.16, a seguir, mostra uma lista de imagens. Os indicadores '*' e '#' ao lado do título da tela indicam que estas teclas podem ser utilizadas para alterar o

filtro aplicado aos itens da lista, mostrando apenas um determinado subconjunto dos itens:

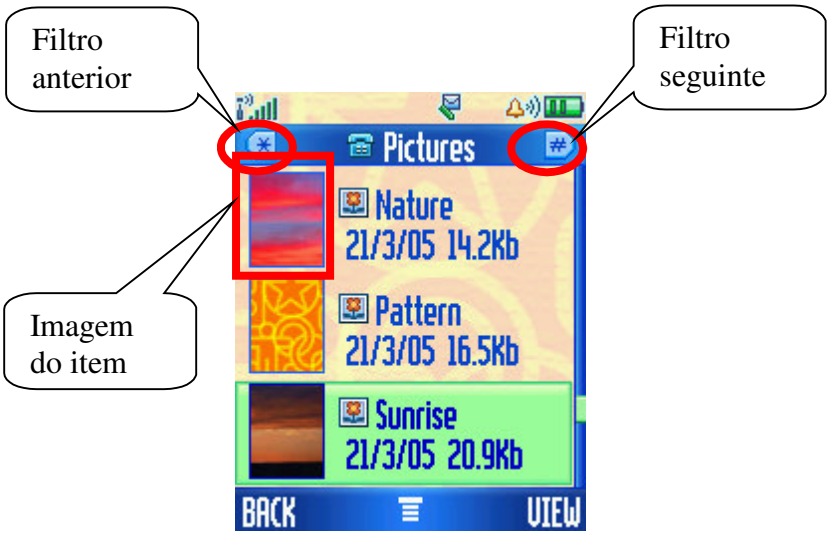

Figura 3.16 - Exemplo de Menu Icônico

3.5.6 Listas de Seleção Múltipla

Listas de seleção múltipla são semelhantes às listas de seleção simples, porém permitem ao usuário marcar um ou mais itens (semelhante aos menus *check-box* usados em computadores de mesa) e realizar uma ação contextual sobre os itens marcados.

O diagrama de classes da Figura 3.17, a seguir, representa a lista de seleção múltipla:

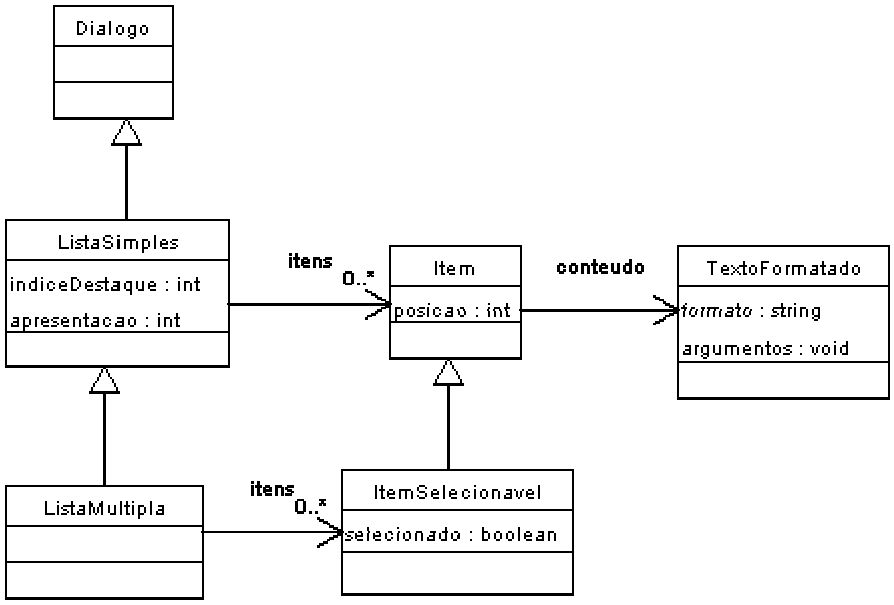

Figura 3.17 - Modelo para Lista de Seleção Múltipla

As classes introduzidas nesse diagrama são:

- `ListaMultipla` – especialização da classe ListaSimples, permitindo que múltiplos itens sejam marcados.
- `ItemSelecionavel` – especialização da classe `Item`, adicionando um atributo para indicar se o item está selecionado ou não.

Os atributos e relacionamentos estão descritos na Tabela abaixo:

Classe	Atributo/Relac.	Descrição
ItemSelecionavel	Selecionado	Indica se o item está selecionando ou não.

Tabela 3.5 - Atributos de ItemSelecionavel

O exemplo da Figura 3.18, a seguir, corresponde a uma lista de seleção múltipla utilizada para selecionar endereços:

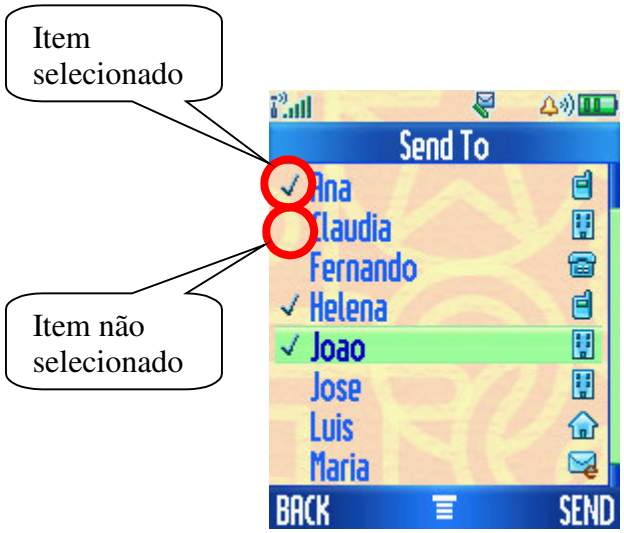

Figura 3.18 - Exemplo de Lista de Seleção Múltipla

3.5.7 Formulários

Formulários são compostos de um ou mais campos de dados dispostos em seqüência. Um dos campos é apresentado em destaque. As teclas de navegação permitem alterar o campo em destaque. Cada campo é composto por um par 'nome' e 'valor'. O valor pode ser apenas para leitura ou também para edição. O valor não pode ser editado diretamente no formulário, devendo a tecla central de navegação ou a tecla de ação esquerda serem utilizadas para abrir um editor correspondente ao tipo do campo que está em destaque. Os

campos podem ser do tipo enumerado, texto, data, hora, inteiro e faixa de valores.

O diagrama de classes da Figura 3.19, a seguir, representa o formulário:

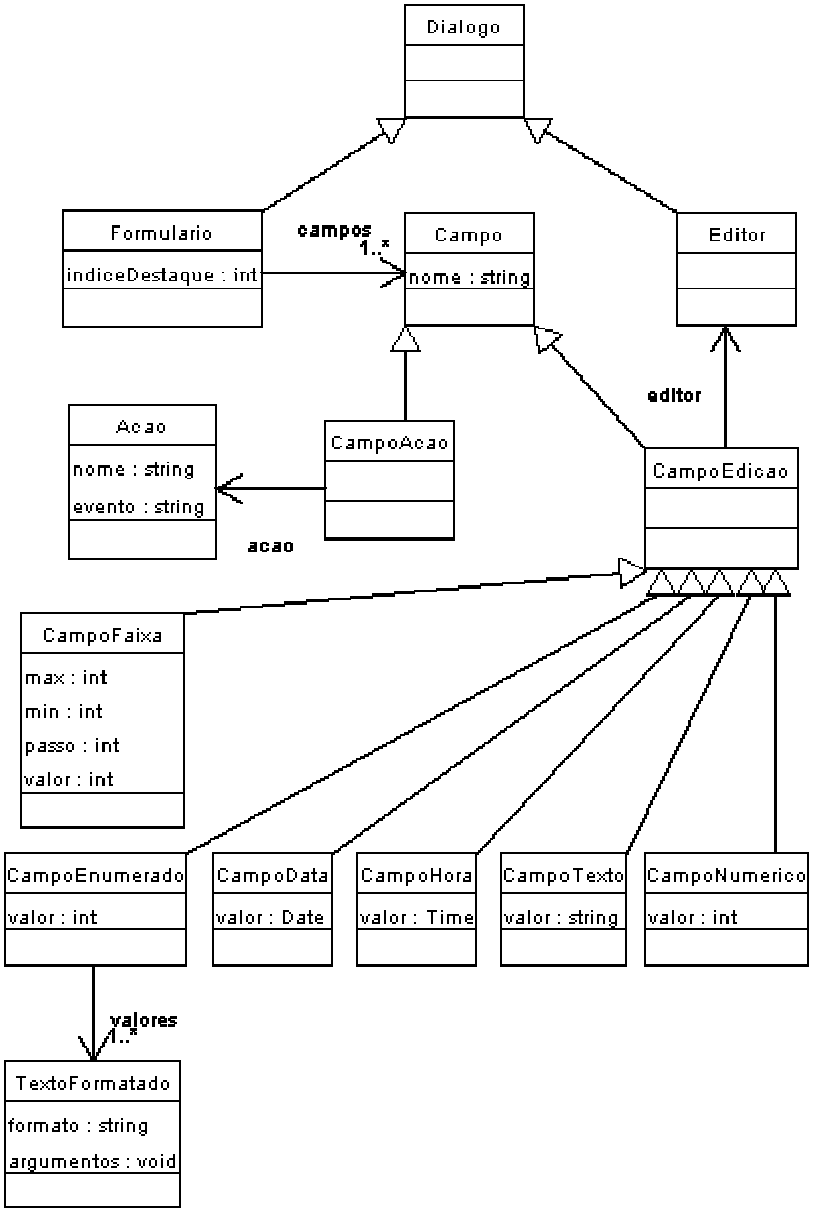

Figura 3.19 - Modelo para Formulários

As classes apresentadas nesse diagrama são:

- `Formulario` – conjunto de campos editáveis.
- `Campo` – classe base para os campos do formulário.

- `CampoEdicao` – classe base para campo de formulário que permite editar um valor.
- `CampoAcao` – campo de formulário que permite executar uma ação.
- `CampoFaixa` – especialização da classe `CampoEdicao`,
- `CampoEnumerado` – especialização da classe `CampoEdicao`,
- `CampoData` – especialização da classe `CampoEdicao`,
- `CampoHora` – especialização da classe `CampoEdicao`,
- `CampoTexto` – especialização da classe `CampoEdicao`,
- `CampoNumerico` – especialização da classe `CampoEdicao`,
- `Editor` – classe base para todos os tipos de editores que podem ser utilizados em um formuário.

Os atributos e relacionamentos estão descritos na Tabela 3.6, a seguir:

Classe	**Atributo/Relac.**	**Descrição**
Formulario	indiceDestaque	Indica qual campo está selecionado.
	campos	Conjunto de campos do formulário.
Campo	nome	Nome do campo.
CampoEdicao	editor	Editor correspondente ao tipo do campo.
CampoAcao	acao	Ação correspondente ao campo.
CampoFaixa	valor	Valor do campo de faixa de valores.
	max	Limite superior.
	min	Limite inferior.
	passo	Valor de incremento/decremento mínimo.
CampoEnumerado	valor	Valor do campo enumerado.
	valores	Texto formatado correspondente a cada valor do enumerado.
CampoData	valor	Valor do campo de data.
CampoHora	valor	Valor do campo de hora.
CampoTexto	valor	Valor do campo de texto.
CampoNumerico	valor	Valor do campo numérico.

Tabela 3.6 - Atributos de Formulario e Campo

O formulário gera os seguintes eventos:

- `navegação` – indica que o campo em destaque mudou.
- `campoAlterado` – indica que o valor do campo foi alterado.

O exemplo da Figura 3.20, a seguir, mostra um exemplo de formulário usado na configuração da caixa de entrada de mensagens do telefone:

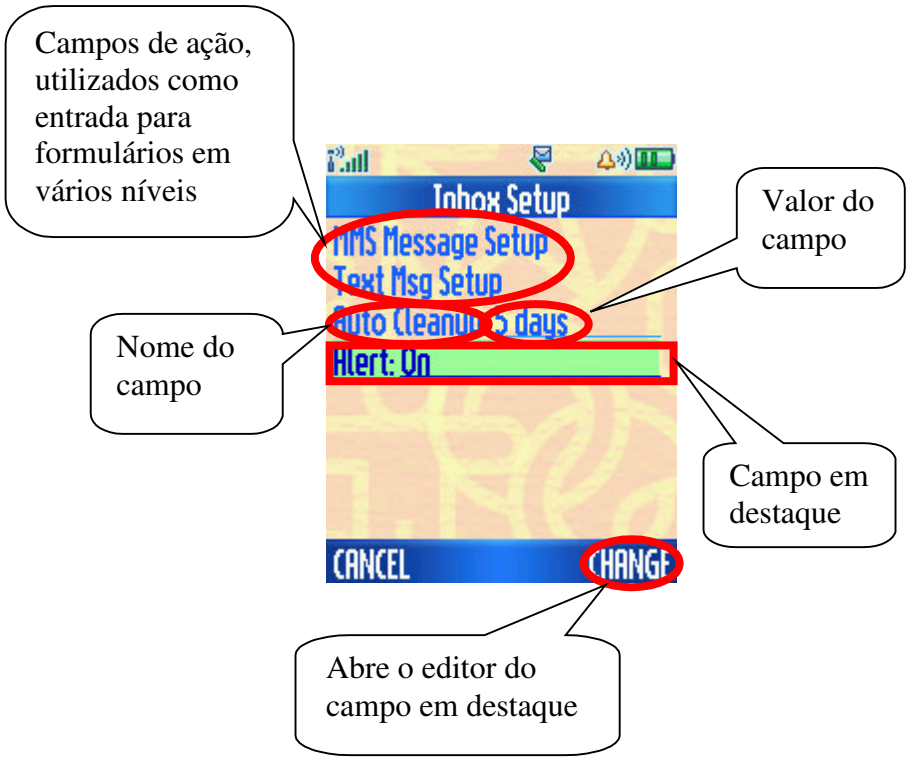

Figura 3.20 - Exemplo de Formulário

3.5.8 Editores

Editores são normalmente utilizados indiretamente a partir de formulários, mas podem ser utilizados diretamente por uma tarefa, como por exemplo na edição de mensagens, onde o conteúdo é composto de texto formatado, imagens e sons.

O diagrama de classes da Figura 3.21, a seguir, representa os editores:

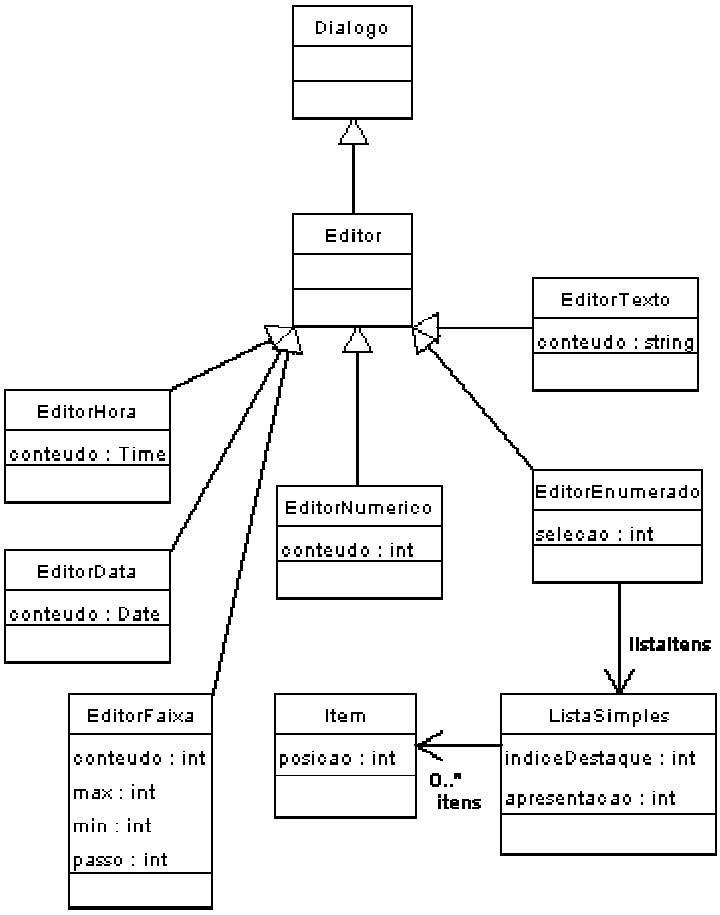

Figura 3.21 - Modelo para Editores

As classes apresentadas nesse diagrama são:

- `EditorTexto` – editor de texto.
- `EditorNumerico` – editor de valores numéricos.
- `EditorEnumerado` – editor de valores enumerados, apresentados sob a forma de uma lista de seleção simples com um marcador para indicar o valor selecionado.
- `EditorFaixa` – editor de faixa de valores, por exemplo um controle de volume.

Os atributos e relacionamentos estão descritos na Tabela 3.7, a seguir:

Classe	**Atributo/Relac.**	**Descrição**
EditorTexto	conteudo	Texto sendo editado.
EditorNumerico	conteudo	Valor numérico sendo editado.

EditorData	conteudo	Valor da data sendo editada.
EditorHora	conteudo	Valor da hora sendo editada.
EditorEnumerado	selecao	Item selecionado.
	itens	Conjunto de valores do enumerado.
EditorFaixa	conteudo	Valor numérico sendo editado.
	max	Limite superior da faixa de valores.
	min	Limite inferior da faixa de valores.
	passo	Incremento/decremento mínimo.

Tabela 3.7 - Atributos de Editores

O exemplo da Figura 3.22, a seguir, mostra um editor numérico aberto a partir de um formulário de configuração das mensagens de texto:

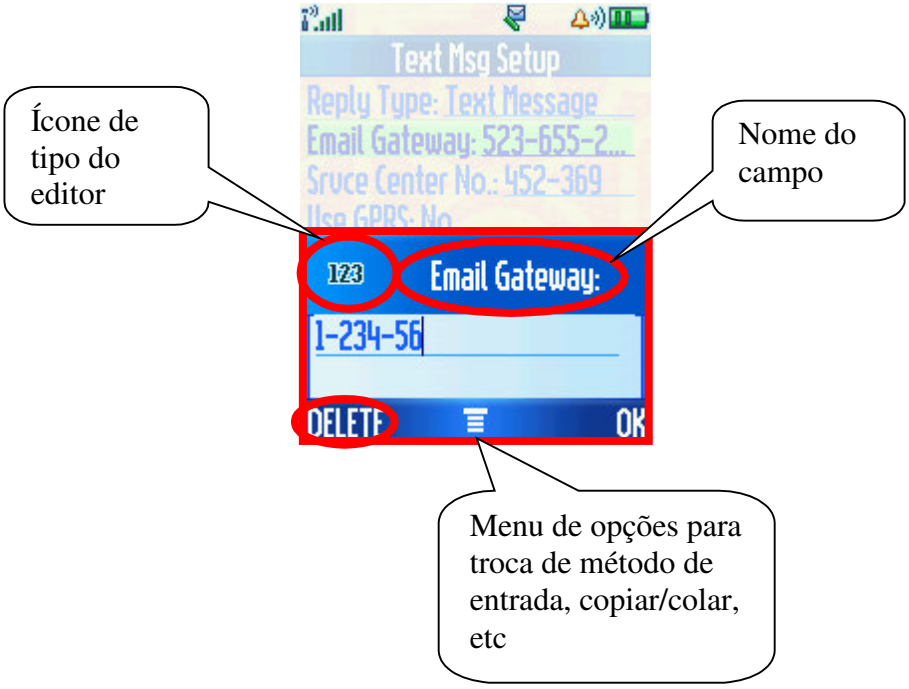

Figura 3.22 - Exemplo de Editor Numérico

O exemplo da Figura 3.23, a seguir, corresponde a um editor de valores enumerados do tipo booleano, neste caso o campo de configuração de alertas do formulário mostrado na Figura 3.20, o qual pode ser ligado e desligado:

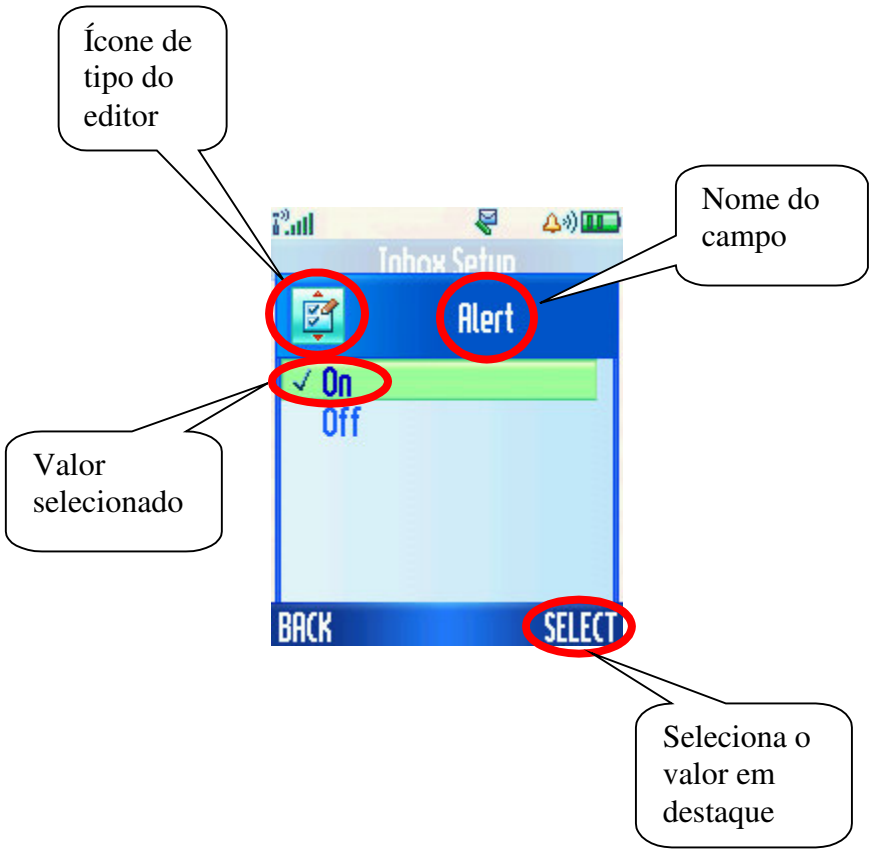

Figura 3.23 - Exemplo de Editor para Booleano

Um outro exemplo de editor de valores enumerados é mostrado na Figura 3.24, a seguir, usado na configuração do período de limpeza automática da caixa de entrada de mensagens:

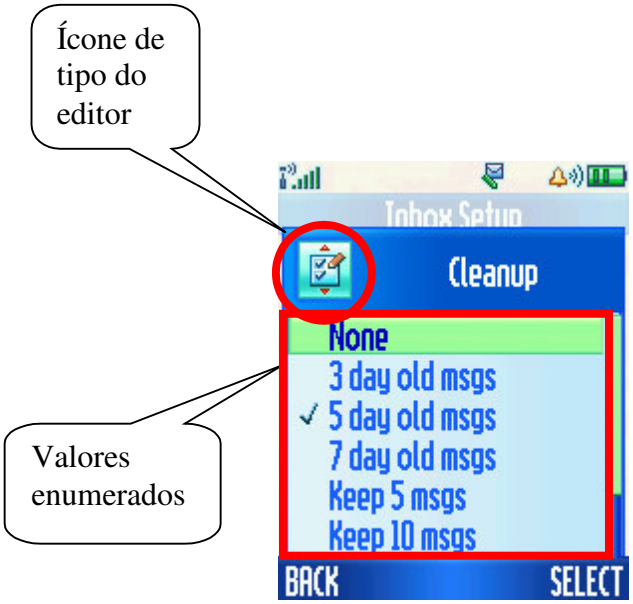

Figura 3.24 - Exemplo de Editor para Enumerado

O exemplo da Figura 3.25, a seguir, mostra um de editor de datas:

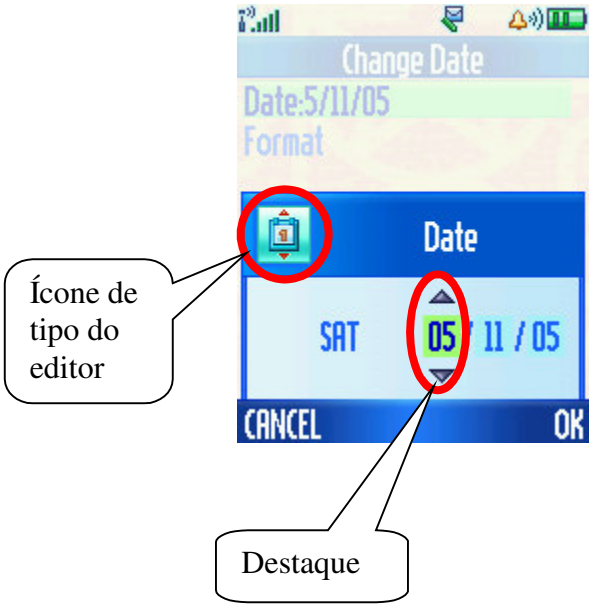

Figura 3.25 - Exemplo de Editor para Data

O exemplo da Figura 3.26, a seguir, corresponde a um editor de horas:

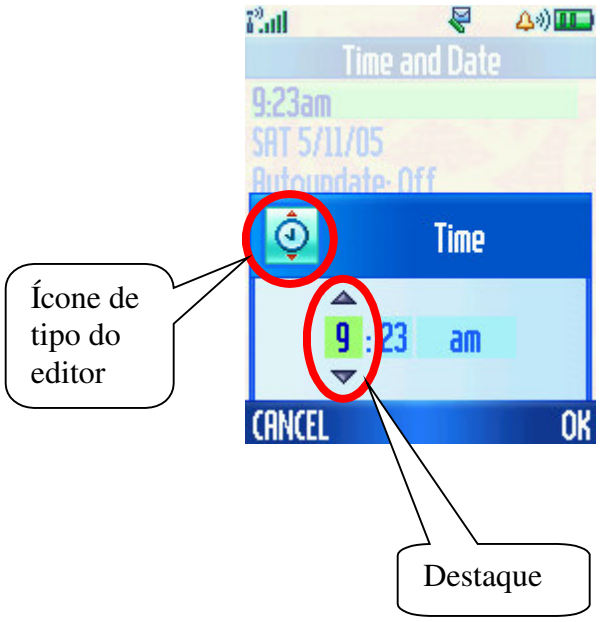

Figura 3.26 - Exemplo de Editor para Hora

3.5.9 Diálogos de Visualização

Diálogos de visualização são utilizados para apresentar dados, como o conteúdo de mensagens, da agenda de compromissos ou dos arquivos de imagem e som.

O diagrama de classes da Figura 3.27, a seguir, representa os diálogos de visualização:

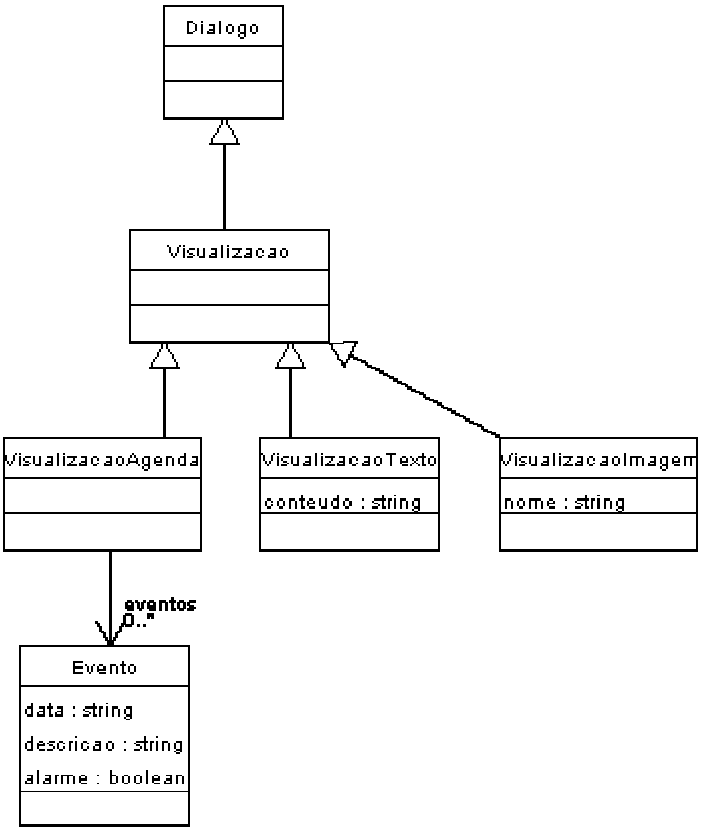

Figura 3.27 - Modelo para Diálogos de Visualização de Texto

As classes apresentadas nesse diagrama são:

- `Visualizacao` – classe base para todos os diálogos de visualização.
- `VisualizacaoTexto` – visualizador de texto.
- `VisualizacaoAgenda` – visualizador da agenda.
- `VisualizacaoImagem` – visualizador de imagem.
- `Evento` – evento da agenda.

Os atributos e relacionamentos estão descritos na Tabela 3.8, a seguir:

Classe	**Atributo/Relac.**	**Descrição**
VisualizacaoTexto	conteudo	Texto sendo mostrado.
VisualizacaoAgenda	eventos	Conjunto de eventos da agenda.
VisualizacaoImagem	nome	Nome do arquivo de imagem.
Evento	data	Data correspondente ao evento.
	descricao	Descricao do evento.

	alarme	Indica se o alarme deve ser tocado no momento do evento ou não.

Tabela 3.8 - Atributos de VisualizadorTexto e VisualizadorAgenda

O exemplo da Figura 3.28, a seguir, mostra um diálogo de visualização de texto utilizado para mostrar o conteúdo de uma mensagem de texto:

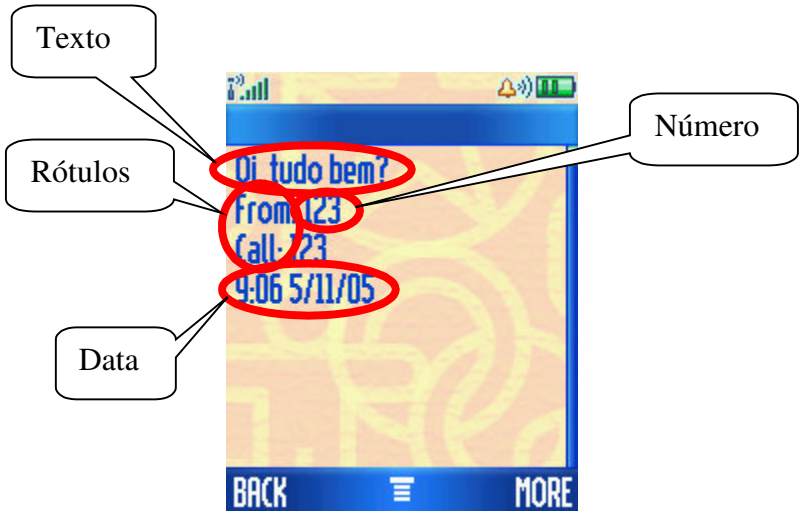

Figura 3.28 - Exemplo de Diálogo de Visualização de Texto

O exemplo da Figura 3.29, a seguir, corresponde a um diálogo de visualização da agenda de compromissos, que mostra a data corrente e os dias em que existem eventos programados:

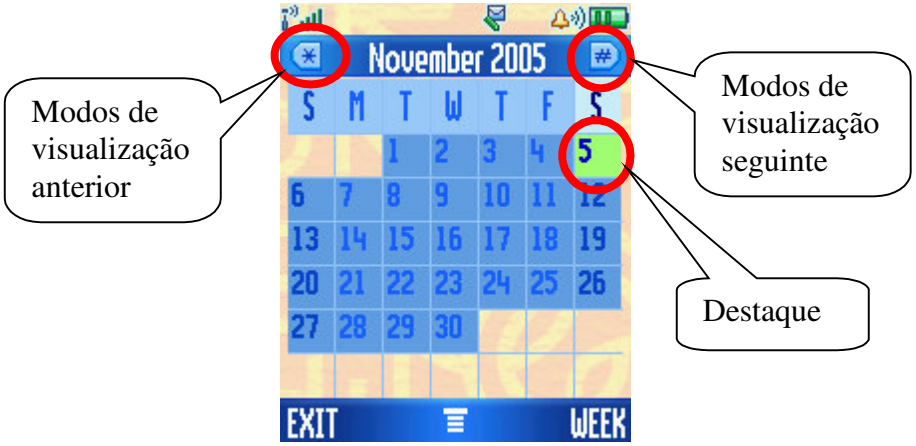

Figura 3.29 - Exemplo de Diálogo de Visualização da Agenda

O exemplo da figura 3.30, a seguir, corresponde a um diálogo de visualização de imagem:

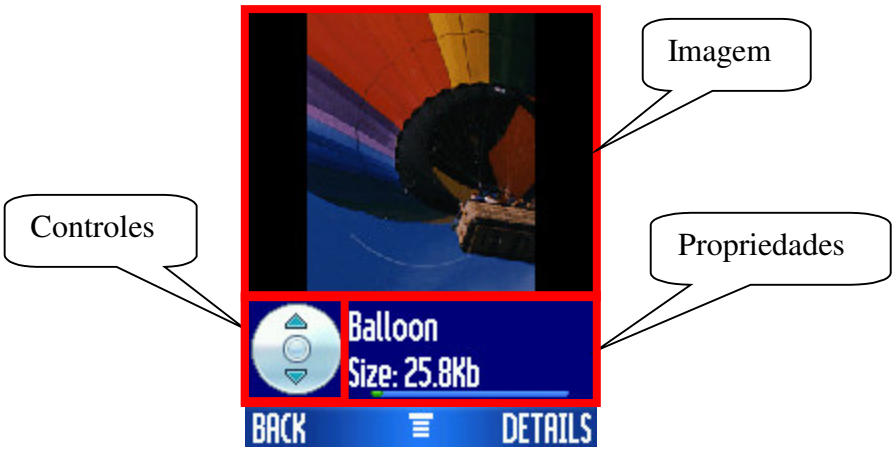

Figura 3.30 - Exemplo de Diálogo de Visualização de Imagem

3.5.10 Diálogos de Notificação e Confirmação

Diálogos de notificação e confirmação são apresentados de forma modal, sobrepondo parcialmente a tela em foco. Em geral possuem um título, texto e ícone indicativo do tipo de informação que esta sendo apresentada, por exemplo indicando um erro. Diálogos de notificação podem ser transientes, desaparecendo automaticamente após um tempo pré-determinado, ou com confirmação, permitindo ao usuário indicar se determinada operação deve ou não ser realizada (utilizando as teclas de ação esquerda e direita para escolher, por exemplo, entre "sim" e "não"), ou simplesmente ter controle sobre quando ele deve ser removido da tela (utilizando uma tecla única de ação, por exemplo "OK").

O diagrama de classes da Figura 3.31, a seguir, representa os diálogos de notificação e confirmação:

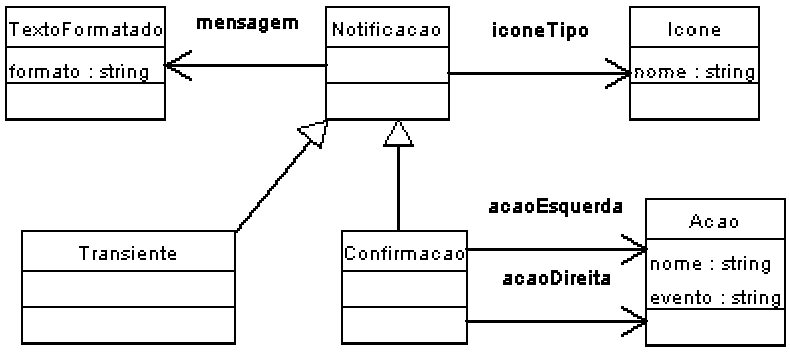

Figura 3.31 - Modelo para Notificação

As classes apresentadas nesse diagrama são:

- `Notificacao` – classe base para os diversos tipos de notificação.
- `Transiente` – notificação temporizada (não possui teclas de ação).
- `Confirmacao` – notificação com confirmação (possui teclas de ação).

Os atributos e relacionamentos estão descritos na Tabela 3.9, a seguir:

Classe	**Atributo/Relac.**	**Descrição**
Notificacao	iconeTipo	Ícone associado ao tipo da notificação.
	mensagem	Texto formatado mostrado pela notificação.
Confirmacao	acaoEsquerda	Ação correspondente à tecla esquerda.
	acaoDireita	Ação correspondente à tecla direita.

Tabela 3.9 - Atributos de Notificação, Transiente e Confirmação

Os diálogos geram os seguintes eventos:

- NotificacaoPositiva – indica que a confirmação foi aceita.
- NotificacaoNegativa – indica que a confirmação foi cancelada.
- NotificaçãoRemovida – indica que a notificação transiente ou de confirmação única foi removida da tela.

O exemplo da Figura 3.32, a seguir, corresponde a um diálogo de notificação transiente associado ao editor mostrado na Figura .23:

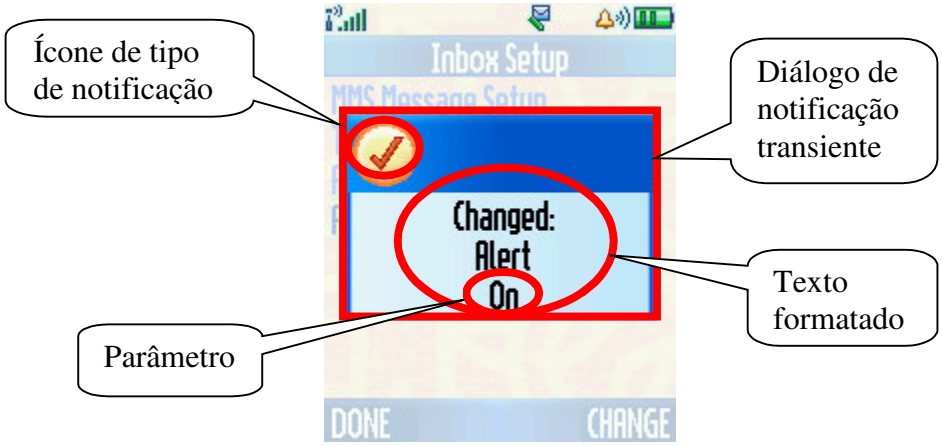

Figura 3.32 - Exemplo de Diálogo de Notificação Transiente

O exemplo da Figura 3.33, a seguir, mostra um diálogo de confirmação para a operação de remoção de uma mensagem da caixa de entrada:

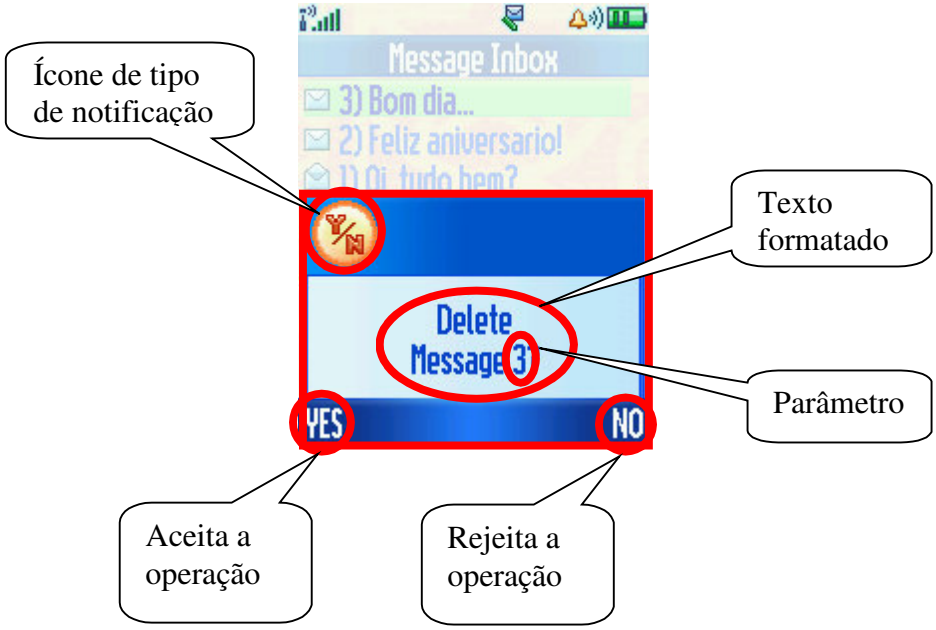

Figura 3.33 - Exemplo de Diálogo de Notificação com Confirmação

O exemplo da Figura 3.34, a seguir, mostra um diálogo de chegada de nova mensagem:

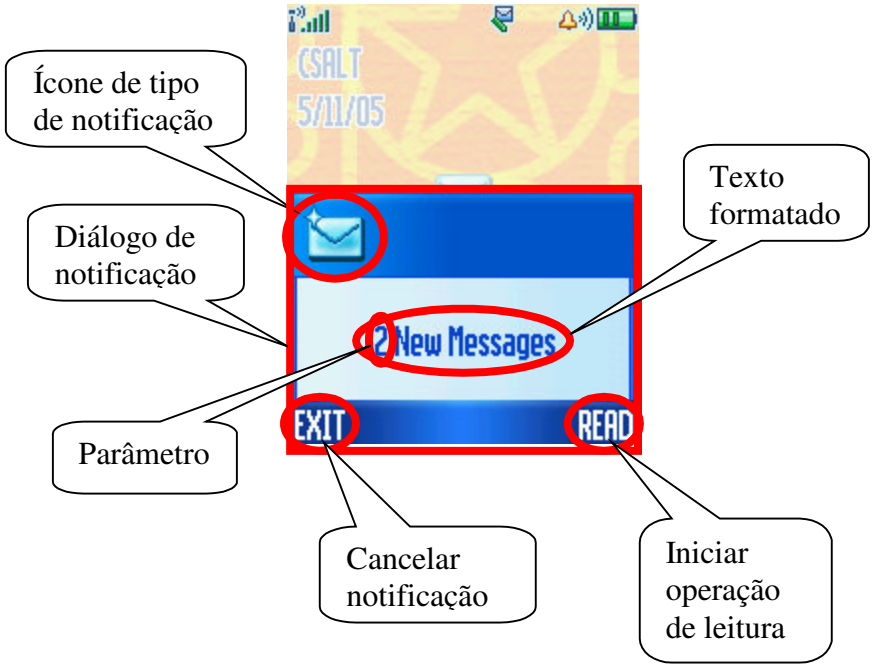

Figura 3.34 - Exemplo de Diálogo de Notificação de Mensagem

3.5.11 Diálogo Inicial

O diálogo inicial indica que o telefone está pronto para uso, sendo o nível mais alto na hierarquia de telas. Não possui título e a região de conteúdo é dividida em várias áreas de informação, como por exemplo o nome da operadora, uma linha de texto para mensagens, relógio, conteúdos dinâmicos recebidos da rede, atalhos para aplicações associados às teclas de navegação, etc. A tecla central de navegação normalmente permite abrir o menu principal de aplicações do dispositivo. As teclas de ação esquerda e direita também são utilizadas como atalhos para aplicações.

O diagrama de classes da Figura 3.35, a seguir, representa os diálogos de notificação e confirmação:

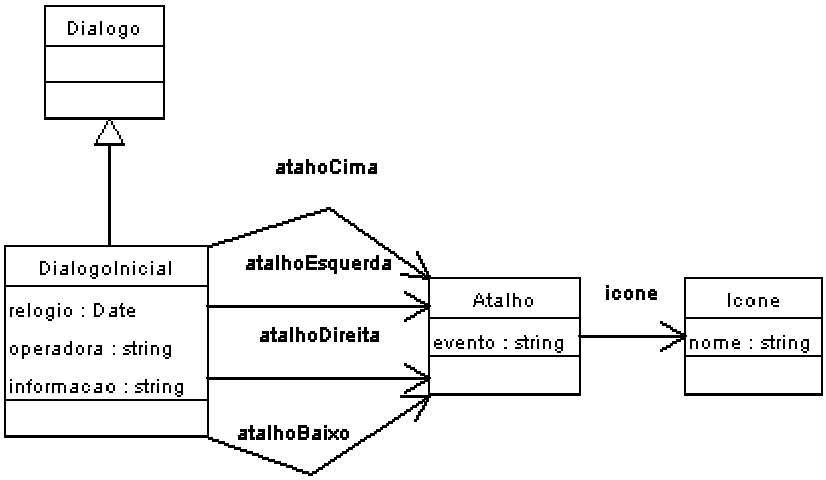

Figura 3.35 - Modelo para DialogoInicial

As classes apresentadas nesse diagrama são:

- DialogoInicial – conteúdo da tela inicial.
- Atalho – aplicação associada a uma das teclas de navegação.

Os atributos e relacionamentos estão descritos na Tabela 3.10, a seguir:

Classe	**Atributo/Relac.**	**Descrição**
DialogoInicial	relogio	Valor da data e hora.
	operadora	Nome da operadora.

	informacao	Informação mostrada abaixo do nome da operadora.
	atalhoCima	Atalho correspondente à tecla para cima.
	atalhoBaixo	Atalho correspondente à tecla para baixo.
	atalhoEsquerda	Atalho correspondente à tecla para a esquerda.
	atalhoDireita	Atalho correspondente à tecla para a direita.
Atalho	evento	Evento associado ao atalho.
	icone	Ícone associado ao atalho.

Tabela 3.10 - Atributos de DialogoInicial

O exemplo da figura 3.36, a seguir, mostra o conteúdo do diálogo inicial:

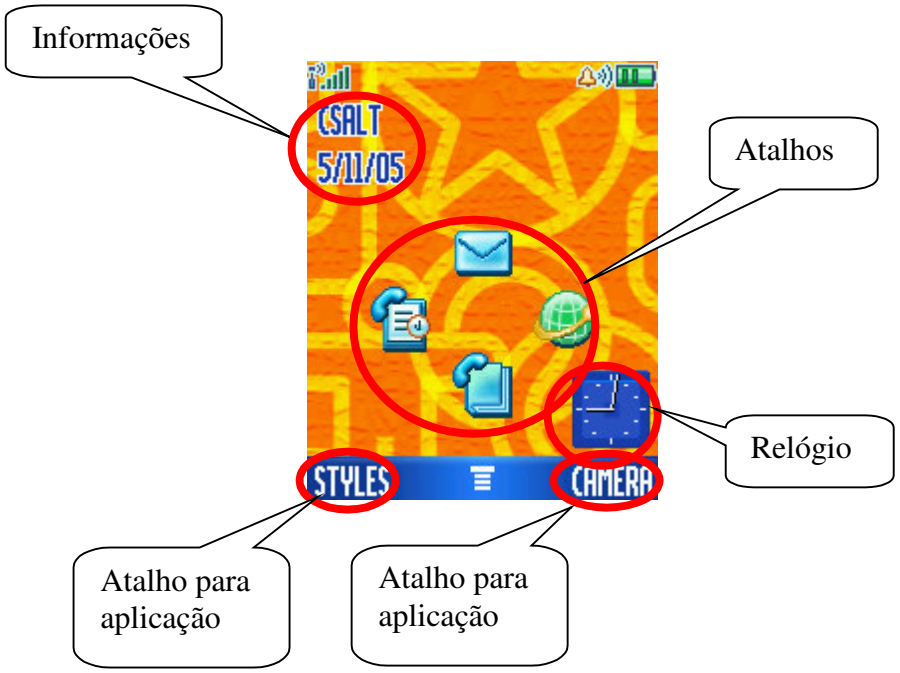

Figura 3.36 - Exemplo de Diálogo Inicial

3.6 Temas e Estilos

Os elementos da interface de usuário descritos anteriormente podem ser apresentados graficamente de diferentes formas, sem que as informações apresentadas ou funcionalidade dos mesmos sejam alteradas. O conjunto de características de cada elemento da interface de usuário que podem ser alteradas representa um tema ou estilo.

O estilo é normalmente definido por uma série de propriedades de cada elemento da interface de usuário, incluindo os arquivos gráficos e de áudio referenciados por aquelas

propriedades. Por exemplo, o tamanho e tipo de fonte utilizado numa lista de seleção são duas propriedades que podem ser alteradas pelo estilo em uso. O conjunto de valores das propriedades de estilo pode ser modificado dinamicamente pelo usuário final, seja por razões estéticas ou de acessibilidade.

O exemplo a seguir corresponde a mesma tela da figura 3.12, porém utilizando um estilo diferente:

Figura 3.37 - Exemplo de estilo

Não é parte do escopo deste trabalho analisar em detalhes as propriedades de estilo de cada elemento da interface de usuário apresentado anteriormente, mas apenas sugerir um mecanismo pelo qual seria possível alterar estes valores.

3.7 Considerações Finais

Neste capítulo foi definido um modelo para interface de usuário de dispositivos portáteis, baseado em telefones celulares atualmente disponíveis no mercado. Este modelo é constituído dos elementos de entrada e saída utilizados pelo dispositivo, um conjunto de diagramas de classes, um modelo de tarefas para o fluxo das telas, e a arquitetura vigente do *software*.

No capítulo seguinte veremos como a linguagem declarativa XForms pode ser utilizada para especificar interfaces de usuário baseadas neste modelo, e quais extensões seriam necessárias em relação à especificação padrão estabelecida pelo W3C.

Capítulo 4

O Uso de XForms e CSS na Definição de Interfaces de Usuário

A seguir é apresentada uma proposta de como as linguagens XForms e CSS podem, em conjunto, serem utilizadas no desenvolvimento de aplicações para os dispositivos portáteis caracterizados pelo modelo descrito no capítulo anterior, indicando quais convenções e extensões seriam necessárias em relação à especificação padronizada pelo W3C, além de propor uma nova arquitetura de *software* em relação à utilizada atualmente nestes dispositivos.

É fundamental para o entendimento deste capítulo que o leitor esteja familiarizado com as diversas tecnologias e especificações relacionadas às linguagens XForms e CSS, incluindo XML, DOM, XML Schema, XML Events e XPath. Uma descrição sumária de cada uma destas tecnologias é apresentada no Anexo A.

É importante notar que o termo *extensão* é utilizado diversas vezes neste capítulo, sempre que uma alteração for necessária em relação às especificações do W3C em questão. Além disso, os vários exemplos dados neste capítulo são apenas fragmentos de documentos XML, não havendo rigor no uso de espaços de nomes.

4.1 Arquitetura de Software Proposta

XForms foi originalmente proposta para o uso em navegadores da Internet, que possuem uma arquitetura cliente/servidor. Para que XForms possa ser utilizado como linguagem de definição da interface de usuário, algumas alterações devem ser feitas na atual arquitetura de *software* vista na sessão 3.2 . A Figura 4.1, a seguir, representa os principais componentes da arquitetura de *software* proposta:

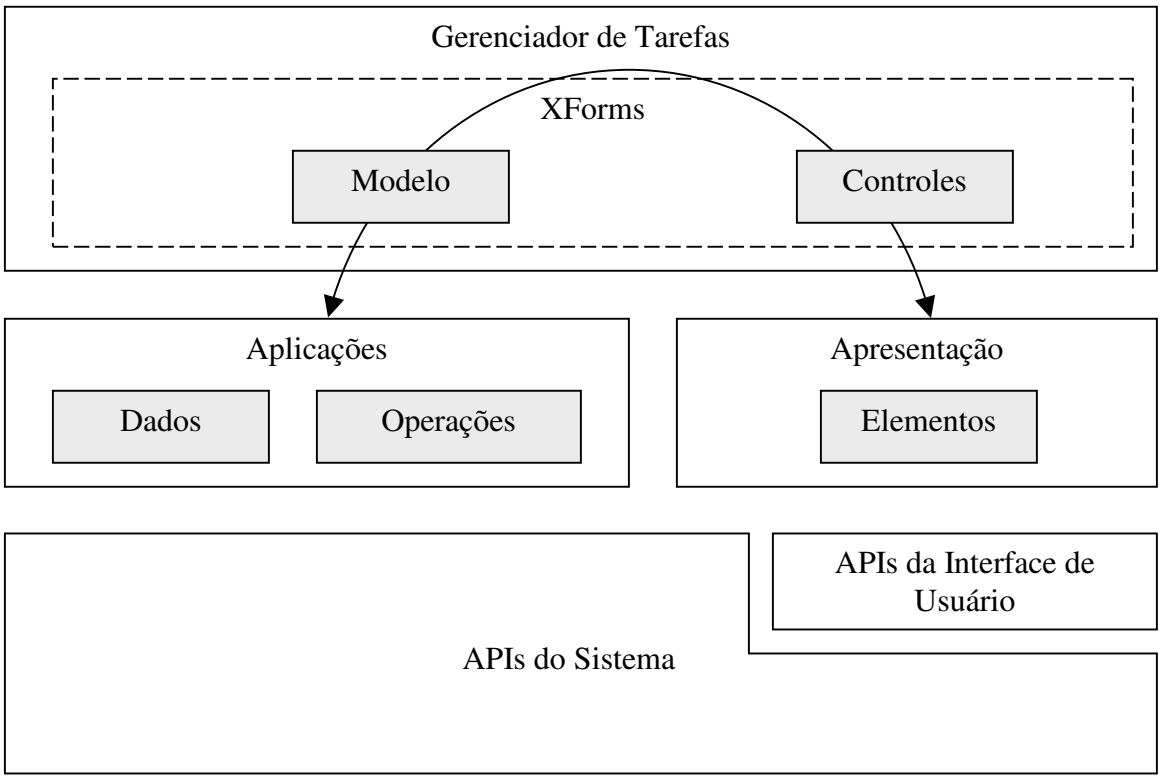

Figura 4.1 - Arquitetura de Software Proposta

Nesta proposta, as aplicações são responsáveis por fornecer dados e executar operações conforme solicitado pelo gerenciador de tarefas, de forma independente da interface de usuário. O gerenciador de tarefas é responsável por ler e interpretar os documentos XForms, mantendo os modelos de dados e solicitando ao componente de apresentação que crie os elementos da interface de usuário, em função dos controles e propriedades CSS especificados. A seta dupla representa o fluxo de dados e de eventos entre apresentação e aplicações. Estes componentes são descritos em maiores detalhes ao final desta sessão.

A arquitetura proposta se aproxima bastante do modelo Arch / Slinky, descrito em [19] e reproduzido na Figura 4.2, a seguir:

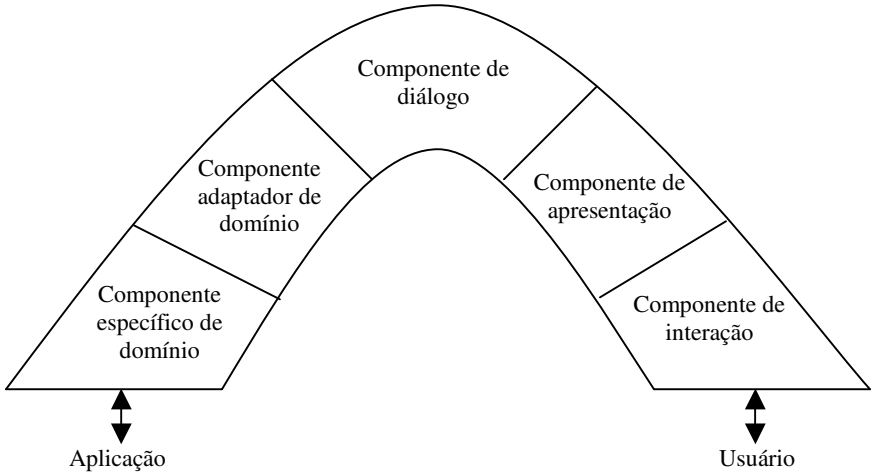

Figura 4.2 - Modelo Arch / Slinky

Este modelo observa o fluxo de dados entre o sistema e a interface de usuário. A responsabilidade de cada componte é:

- Específico de domínio – controla, manipula e recupera dados da aplicação, além de outras funcionalidades pertinentes à aplicação. Corresponde, na arquitetura proposta, às aplicações.
- Adaptador de domínio – atua como um mediador entre o componente de diálogo e o componente específico de domínio, transferindo eventos vindos do sistema e reorganizando dados. Corresponde, na arquitetura proposta, à interface entre as aplicações e o gerenciador de tarefas.
- Diálogo – responsável pelo sequenciamento de tarefas. Corresponde, na arquitetura proposta, ao gerenciador de tarefas.
- Apresentação – atua como um mediador, provendo um conjunto de objetos independentes das APIs gráficas para o componente de diálogo, e criando os objetos correspondentes no componente de interação. Corresponde, na arquitetura proposta, ao componente de apresentação.
- Interação – interação física com o usuário, através da tela, teclado, etc. Corresponde, na arquitetura proposta, ao código que implementa as APIs da interface de usuário.

Este modelo favorece uma maior separação entre a aplicação e a interface de usuário. Na arquitetura proposta, o código das aplicações depende apenas das APIs do sistema, não estando mais acoplada aos componentes da interface de usuário como ocorre atualmente.

Além disso, apenas mudanças de funcionalidade afetam o código das aplicações, assumindo que todos os dados e comandos necessários estão disponíveis.

Parte dos componentes desta proposta de arquitetura pode ser comum ao navegador de Internet do dispositivo, como o suporte para leitura dos arquivos XML e construção do DOM (*parser*). O suporte para XML Schema, XForms e CSS também poderia ser aproveitado, se disponível, modificando apenas a forma como os controles são apresentados e integrando o navegador ao gerenciador de tarefas.

4.1.1 Gerenciador de Tarefas

O gerenciador de tarefas é o principal componente novo na arquitetura, sendo responsável pelas seguintes funcionalidades, de acordo com o mapeamento descrito na sessão 4.2 :

- Leitura e interpretação dos documentos XForms que correspondem a cada tarefa.
- Manutenção de um contexto para cada tarefa ativa e a relação hierárquica entre elas.
- Ativação e encerramento das tarefas.
- Passagem de argumentos e dados de retorno entre tarefas.
- Mapeamento das solicitações (*submission*) para as operações do sistema.
- Construção das instâncias do modelo a partir dos dados contidos no sistema.
- Mudanças de foco entre telas de uma tarefa.

O gerenciador de tarefas é um *framework* para a execução das tarefas, e não requer alterações para que novas funcionalidades ou modos de apresentação sejam adicionados.

4.1.2 Aplicações

Na arquitetura proposta, as aplicações são responsáveis pela execução de operações e acesso aos dados armazenados pelo dispositivo, incluindo variáveis de estado, de acordo com as solicitações feitas pelo gerenciador de tarefas. Desta forma, elas são relativamente simples em comparação com as aplicações da arquitetura vigente, que são responsáveis também por criar os elementos da interface de usuário e controlar o fluxo das telas.

É importante notar que as aplicações são implementadas utilizando uma linguagem imperativa como C ou C++, dependendo de uma interface padrão com o gerenciador de tarefas que permita a adição de novos dados e operações sem que este último precise ser

modificado.

4.1.3 Apresentação

O componente de apresentação é responsável por criar e manter os objetos da interface de usuário correspondentes aos controles presentes no documento XForms, de acordo com o mapeamento descrito na sessão 4.4 . Para tanto, recebe solicitações do gerenciador de tarefas e utiliza as APIs da interface de usuário do dispositivo. Durante este processo, as folhas de estilo CSS são consultadas para que as propriedades sejam aplicadas ao desenhar os controles na tela, conforme descrito na sessão 4.5 .

O componente de apresentação é implementado em uma linguagem imperativa, como C ou C++, somente necessitando ser modificado caso a API da interface de usuário seja alterada ou novos tipos de controles sejam adicionados.

4.1.4 Execução de Operações

No caso de navegadores da Internet, para os quais XForms foi originalmente especificado, o cliente pode submeter dados ao servidor para que sejam processados, conforme definido pelo elemento `submission`. Isto é feito por um pedido assíncrono, utilizando um método adequado, como *post* ou *get* de *http*. Na arquitetura proposta, este mesmo mecanismo é utilizado pelo gerenciador de tarefas para solicitar às aplicações que executem uma operação. Neste caso, o método utilizado corresponde ao envio de um evento, ou chamada de uma função assíncrona, sendo a resposta retornada de forma similar. Por exemplo:

```
<submission id=”apagar-msg” bind=”msg-id” method=”ext:call”
action=”proc://message/delete” replace=”none”>
...
<send submission="apagar-msg"/>
```

No elemento `submission`, o atributo `replace` pode ser usado para especificar o que deve ocorrer após a execução da operação. O valor `instance` permite alterar os valor dos dados da instância do modelo associada ao elemento `submission` com o resultado da operação. O valor `none` indica que nenhum dado é retornado. Além disso, o evento `xforms-submit-done` é enviado ao elemento `submission`, podendo ser associado a quaisquer ações necessárias naquele momento. Uma questão importante é como escolher entre duas possíveis ações dependendo do resultado da operação, por exemplo sucesso e falha. Uma solução seria enviar ao elemento `submission` eventos específicos, de forma

semelhante a `xforms-submit-done`, mas que indiquem cada possível resultado. Estes eventos poderiam então ser associados a elementos `action`, contendo as ações necessárias para cada caso. Esta mesma abordagem poderia ser utilizada nos casos de indicações vindas do sistema, onde não existe uma operação prévia associada, como por exemplo o recebimento de uma mensagem. Neste caso, os eventos seriam enviados ao elemento `model` correspondente àquela indicação.

Extensão 1: Novos eventos enviados aos elementos `submission` e `model`.

Visto que o atributo `action` do elemento `submission` de XForms utiliza um endereço do tipo URI (*Uniform Resource Identifier*) para identificar o destino do pedido, é necessário mapear esta cadeia de caracteres para o evento ou função correspondentes. Isto pode ser feito permitindo que as aplicações registrem um par (URI, evento) ou (URI, função). A URI pode conter um prefixo que facilite a associação com operações executadas pelas aplicações, como `proc://`.

4.1.5 Mapeamento de Dados

Os dados do dispositivo são representados como modelos de dados XForms, como definido pelos elementos `model` e `instance`. O atributo `schema` do elemento `model`, que aponta para um documento XML Schema, permite determinar os tipos de dados presentes no modelo. Este recurso é usado para que os controles da interface, como editores e diálogos de visualização, possam ter uma apresentação adequada ao tipo de dado. Já o atributo `src` do elemento `instance` permite indicar, por meio de um endereço do tipo URI, a fonte dos dados (*data source*).

O mapeamento entre os dados do dispositivo e a respectiva instância do modelo de dados XForms é feito por meio de uma árvore DOM. Os elementos e atributos desta árvore são criados pela aplicação, com o suporte das APIs do dispositivo, conforme solicitado pelo gerenciar de tarefas. Esta árvore pode então ser ligada (*bind*) aos controles da interface do usuário, como definido no documento XForms. Desta forma, o DOM age como um *proxy*, abstraindo a forma como os dados são gerados ou onde estão armazenados e evitando que a aplicação tenha uma dependência da interface de usuário.

Um ponto a ser considerado é a necessidade de atualizar dinamicamente as instâncias do modelo XForms quando os dados do sistema são alterados, em função de eventos não relacionados com a interface de usuário. Por exemplo, os níveis da bateria e do sinal estão constantemente mudando, e os indicadores correspondentes na tela devem permanecer

atualizados. Esta abordagem representa uma maior necessidade de processamento, devendo, portanto, ser utilizada apenas quando necessário.

Também é importante notar que os modelos devem ser estruturados tendo em mente o impacto em memória gerado pelas instâncias. Isto pode ser minimizado através do uso de vários modelos, constituidos pelos menores subconjuntos possíveis de informações do dispositivo, o que resulta em árvores DOM menores.

Uma idéia a ser explorada é o uso de ligações bidirecionais entre as instâncias do modelo e os dados do sistema, de forma que ambos estejam sempre em sincronismo, de forma similar ao suportado por XAML, descrito em 2.2.3 . Com isso não seria necessário submeter pedidos ao sistema para atualização dos dados, bastando alterar a instância do modelo correspondente.

XForms permite que um documento contenha multiplos modelos, e que um modelo contenha multiplas instâncias. A função XPath `instance()` permite selecionar uma determinada instância dentro do modelo definido pelo contexto atual, porém não existe uma forma de referenciar uma instância que pertença a outro modelo. Esta limitação pode ser superada removendo a restrição imposta pela especificação atual do W3C, permitindo que a função `instance()` retorne um conjunto de nós mesmo quando seu argumento corresponder ao atributo `id` de uma instância pertencente a um modelo diferente do utilizado pelo contexto atual.

Extensão 2: Função XPath para permitir selecionar nós de uma instância de um modelo diferente do utilizado pelo contexto atual.

4.2 Modelo de Tarefas

Conforme visto na sessão 3.3, a interface de usuário é formada por um conjunto de tarefas. Cada tarefa por sua vez é composta por um conjunto de telas e os eventos que permitem ao usuário navegar entre elas, executar operações, iniciar outras tarefas e terminar a atual.

Uma tarefa pode ser representada por um documento XForms, contendo a definição das telas, modelos de dados utilizados pelos controles da interface de usuário e ações associadas aos eventos. O documento correspondente a uma tarefa deve conter um

elemento `switch`, o qual por sua vez possui um elemento `case` para cada tela pertencente aquela tarefa, como sugerido em [4] e em [23]. A tela inicialmente selecionada, que possui o foco, é indicada pelo atributo `selected` do respectivo elemento `case`. Um valor apropriado para o atributo `appearance` do elemento `switch` permite caracterizar o mapeamento deste último como uma tarefa. Por exemplo:

```
<switch appearance="ext:task" id="tarefa-a">
  <case id="tela-1" selected="true">
  ...
  </case>
  <case id="tela-2">
  ...
  </case>
</switch>
```

A tarefa ilustrada possui duas telas, como indicado na "Tarefa A" da Figura 3.2. A tela cujo atributo `id` é `tela-1` possui o foco inicialmente.

A navegação entre as telas de uma tarefa é feita através do elemento `toggle`, ativado pelo evento que causa aquela transição. Por exemplo:

```
<toggle ev:event="DOMActivate" case="tela-2"/>
```

O evento `DOMActivate` é o mais comumente gerado pelos controles quando ocorre uma ação do usuário, porém qualquer outro evento poderia ser utilizado. No exemplo acima, quando este evento for gerado ocorrerá uma transição de foco para a tela cujo atributo `id` corresponde ao valor `tela-2`. O elemento `toggle` poderia também ser contido por um elemento `action`, no caso de várias operações estarem associadas a um mesmo evento.

O contexto de uma tarefa pode ser mantido utilizando um modelo de dados interno ao documento XForms. Isto permite que a tarefa exerça um papel de coleta de dados, utilizando um formulário ou mesmo diversas telas para coletar os dados, submetendo-os somente ao final à aplicação para que sejam processados.

O inicio de uma tarefa é equivalente à obtenção de uma página do servidor feita por um navegador. O histórico entre as tarefas é mantido, como ocorre com as páginas de um navegador, de forma a permitir o retorno conforme as tarefas vão sendo finalizadas. O

elemento submission, através de um endereço do tipo URI indicado no atributo action, permite definir o documento correspondente à tarefa que deve ser iniciada. A URI pode conter um prefixo que facilite a associação com o gerenciador de tarefas, como task://. Os argumentos passados para a tarefa sendo iniciada são obtidos da instância do modelo indicada pelo atributo bind, ou ref e model. O atributo replace define o comportamento quando a tarefa é concluída, sendo usado o valor instance quando se deseja que os dados retornados pela tarefa que está sendo finalizada sejam colocados na instância do modelo correspondente ao elemento submission, o valor none quando não é esperado nenhum retorno, ou o valor all para o caso de uma tarefa que deve terminar imediatamente ao iniciar outra. Por exemplo:

```
<submission id="iniciar-a" bind="arg-a" method="ext:call"
action="task://tarefa-a" replace="none">
```

No exemplo apresentado, a tarefa correspondente ao documento XForms de nome tarefa-a é iniciada com os argumentos obtidos da instância do modelo correspondente ao elemento bind cujo atributo id tem o valor arg-a. Estes valores são disponibilizados na tarefa sendo iniciada através de uma instância de modelo contendo um atributo src adequado, como exemplificado a seguir:

```
<model id="tarefa-a-args" schema="tarefa_a_args.xsd">
  <instance src="task://input">
</model>
```

A versão 1.1 de XForms, ainda não finalizada, propõe o uso de um atributo adicional instance, o qual permitiria definir uma instância do modelo para colocar os valores de retorno, diferente da usada para passar os argumentos.

De forma similar à descrita, a finalização de uma tarefa também pode ser feita através do elemento submission, utilizando um endereço do tipo URI apropriado para o atributo action, como indicado no exemplo abaixo:

```
<submission id="terminar-b" bind="ret-b" method="ext:call"
action="task://return">
```

Neste caso, os dados da instância do modelo definido pelo atributo bind são enviados de volta à tarefa anterior. O evento xforms-submit-done é enviado após o

retorno, podendo ser associado a quaisquer ações necessárias naquele momento.

Em ambos os casos descritos, a ação de início ou finalização da tarefa seria invocada por meio do elemento `send`, associado ao evento correspondente. Por exemplo:

```
<send ev:event="DOMActivate" submission="iniciar-a">
```

Nesta abordagem, é possível ter vários documentos XForms ativos em um determinado momento, correspondentes às diversas tarefas que foram iniciadas porém ainda não concluídas. É importante notar que o início de uma tarefa pode ocorrer em função de eventos correspondentes a mudanças no estado do dispositivo, como bateria fraca, perda de sinal, recebimento de uma mensagem, etc. Estes eventos não estão associados às tarefas que possam estar ativas num certo momento, pois independem da intervenção direta do usuário. Por esta razão, torna-se necessário associar um documento para o tratamento destes eventos. Este documento pode corresponder à tela inicial do dispositivo, que está sempre ativa, ou a um ou mais documentos específicos para este fim, carregados quando o sistema é iniciado. Os eventos externos são então enviados aos modelos de dados destes documentos, correspondentes aos estados do equipamento.

4.3 Texto Formatado

O mecanismo utilizado para representar texto formatado deve ser compatível com XML e XForms, permitindo que dados do modelo sejam referenciados, e evitando ou minimizando mudanças na especificação W3C. Por exemplo, é necessário que este tipo de informação possa ser especificada para o elemento `label`, utilizado para rotular nome de controles.

As cadeias de caracteres e formatação possuem identificadores únicos que permitem obter os valores correspondentes, independente de como estão armazenados, como descrito na sessão 3.4 . Para que um identificador possa ser usado em um documento XForms, ele deve ser representado como uma cadeia única de caracteres. Esta última é então usada no documento XForms e mapeada para o valor correspondente do identificador, permitindo o uso das APIs de texto formatado atuais. Por exemplo, considere que o título da tela indicada na Figura 3.7 corresponde ao identificador `titulo_configuracoes`. Este identificador deve ser convertido por uma API do sistema numa na cadeia de texto correspondente, de acordo com a língua selecionada. No documento XForms, esta API pode ser invocada por meio de uma nova função XPath, em adição às descritas nas sessões

A.5.2 e A.6.8, como indicado a seguir:

```
i18n('titulo_configuracoes')
```

Para o caso de cadeias de formatação envolvendo parâmetros, como no caso da Figura 3.33, é necessário, além do identificador da cadeia de formatação, passar uma ou mais referências a instâncias do modelo contendo os valores correspondentes a cada parâmetro. Isto pode ser feito utilizando caminhos de localização XPath, passados como argumentos da função proposta anteriormente. Por exemplo:

```
i18n('confirmacao_apagar', /mensagens/selecionada)
```

O identificador `confirmacao_apagar` aponta para uma cadeia de formatação que define a concatenação entre três elementos: uma cadeia de caracteres "Apagar Mensagem", a representação numérica correspondente à instância do modelo indicado pelo caminho "/mensagens/selecionada" e o caractere '?'. Com isso, o retorno da função `i18n()` é a cadeia de caracteres "Apagar Mensagem 2?".

É importante notar que os elementos `label` e `message` não suportam diretamente que o valor do texto seja especificado por meio de uma expressão XPath, porém eles podem conter um elemento `output`, o qual suporta isso através do atributo `value`. Por exemplo:

```
<label>
  <output value="i18n('titulo_configuracoes')"/>
</label>
```

O mesmo se aplica aos elementos `help`, `hint`, `alert` e `message`.

4.4 Elementos da Interface de Usuário

A seguir é definido o mapeamento entre o modelo da interface de usuário descrito na sessão 3.5 , e os elementos das linguagens XForms e CSS.

4.4.1 Teclado

Os eventos gerados pelas teclas são enviados diretamente à interface de usuário e processados pelo componente de apresentação. O resultado é uma atualização no estado dos controles e o envio, caso necessário, de eventos de mais alto nível ao documento XForms, por exemplo indicando que um item de uma lista foi selecionado. Para o caso de teclas

especiais, como as teclas de fim e envio indicadas na sessão 3.5 , é necessário definir novos eventos que possam ser associados a ações XForms. Estes eventos podem ser direcionados, por exemplo, ao elemento `group` correspondente à tela em foco, ou a uma instância do modelo correspondente ao teclado.

Extensão 3: Novos eventos para indicar que teclas especiais foram ativadas.

4.4.2 Tela

O elemento `case` de XForms pode ser usado para definir os atributos e o conteúdo de uma tela pertencente a uma tarefa, como discutido na sessão 4.2 .

O mapeamento dos atributos e relacionamentos das classes identificadas na Tabela 3.1 está indicado na Tabela 4.1, a seguir:

Classe	**Atributo/Relac.**	**Correspondência em XForms**
Tela	titulo	Definido pelo elemento `label` contido no elemento `case` correspondente à tela.
	acaoEsquerda	Definido por um elemento `trigger` contido no elemento `case` correspondente à tela. A atribuição à ação esquerda ou direita é uma característica gráfica definida por uma propriedade CSS. Alguns controles podem definir automaticamente uma tecla de ação, como "Select" no caso de uma lista de seleção.
	acaoDireita	Idem acima.
	iconeTitulo	Característica gráfica definida por uma propriedade CSS atribuida ao elemento `case` correspondente à tela.
	areaConteudo	Definido pelo controle XForms contido no elemento `case` correspondente à tela.
	menuOpcoes	Definido por um elemento `group` contido no elemento `case` correspondente à tela.
Acao	nome	Definido pelo elemento `label` contido no elemento `trigger` correspondente à ação.

	evento	O evento DOMActivate é enviado ao elemento `trigger` correspondente àquela ação, no momento em que ela é ativada.
Icone	nome	Característica gráfica definida por uma propriedade CSS atribuida ao elemento `trigger` correspondente à ação.

Tabela 4.1 - Mapeamento para Tela, Acao e Icone

Por exemplo, a tela da Figura 3.7 pode ser representada da seguinte forma:

```
<case id="tela_configuracoes">
  <label>
    <output value="i18n('settings')"/>
  </label>
  <select1 ref="/configuracoes/selecao">
    <label/>
    <item class="personilize">
      <label>
        <output value="i18n('personilize')/>
      </label>
      <value>1</value>
    </item>
    ...
  </select1>
</case>
```

O título da tela corresponde ao identificador de texto formatado `settings`, conforme definido pelo elemento `label` logo abaixo do elemento `case`. O controle utilizado é uma lista de seleção simples, conforme indicado pelo elemento `select1`. O elemento `label` do controle está vazio, pois esta informação não é utilizada nesse caso, visto que controles associados à área de conteúdo da tela não possuem título próprio, com exceção de campos de formulário. Cada item da lista é definido por um elemento `item`, que contém um elemento `label` com o nome do item, e um elemento `value` com o valor correspondente aquele item. É importante notar que o controle da lista simples é responsável por atribuir o rótulo "Select" automaticamente. A definição do rótulo "Back" foi omitida do exemplo. Os ícones do título da tela e de cada item da lista devem ser especificados por meio de propriedades CSS, associadas ao documento XForms que descreve esta tela. Um fragmento CSS que exemplifica esta idéia é apresentado a seguir:

```
case#tela_configuracoes {
  title-icon: settings.png;
```

```
}

.personilize {
  list-item-icon: personilize.png;
}
```

As propriedades CSS acima são consultadas ao desenhar o título da tela e os itens da lista.

4.4.3 Indicadores de Estado

Como descrito na sessão 4.2 , os eventos do sistema podem ser tratados por um documento carregado ao iniciar o dispositivo. Isso inclui a definição dos indicadores de estado, que normalmente independem de uma interação do usuário e devem estar sempre visíveis. Por serem dados de saída, eles podem ser representados em XForms através do elemento `output`, contendo um atributo `appearance` que permite assoaciar a apresentação adequada.

O mapeamento dos atributos e relacionamentos das classes identificadas na Tabela 3.2 está indicado na Tabela 4.2, a seguir:

Classe	**Atributo/Relac.**	**Descrição**
AreaEstado	indicadoresEstado	Conjunto de elementos `output`, definidos no documento correspondente aos indicadores de estado, contendo um atributo `appearance` adequado.
IndicadorEstado	iconesEstado	Característica gráfica definida por propriedades CSS associadas ao elemento `output` correspondente ao indicador de estado.
	valor	Fornecido por uma instância do modelo ligada ao elemento `output` correspondente a este indicador.
	posicao	Característica gráfica definida por uma propriedade CSS associada ao elemento `output` correspondente ao indicador de estado.

	prioridade	Característica gráfica definida por uma propriedade CSS associada ao elemento `output` correspondente ao indicador de estado.

Tabela 4.2 - Mapeamento para AreaEstado e IndicadorEstado

Por exemplo, os indicadores de estado podem ser definidos da seguinte forma:

```
<model schema="sistema.xsd">
  <instance src="model://sistema"/>
  <bind id="nivel-sinal" nodeset="sinal/nivel"/>
  <bind id="nivel-bateria" nodeset="bateria/nivel"/>
</model>

...

<output appearance="status" id="estado-sinal" bind="nivel-
sinal"/>
<output appearance="status" id="estado-bateria" bind="nivel-
bateria"/>
```

O atributo `id` permite atribuir propriedades CSS a estes indicadores. O seguinte fragmento CSS poderia ser usado para definir os ícones de um indicador de estado do exemplo anterior:

```
output#estado-sinal {
  status-slot: 0;
  level0-icon: no_signal.png;
  level0-priority: 1;
  level1-threshold: 10;
  level1-priority: 0;
  level1-icon: signal_low.png;
  level2-threshold: 50;
  level2-priority: 0;
  level2-icon: signal_high.png;
}
```

As propriedades CSS não padrão usadas acima indicam em qual região da área de estados deve ser colocodo o ícone (`status-slot`), qual o ícone (`level`*`n`*`-icon`) e qual o limiar (`level`*`n`*`-threshold`) do valor do estado para que ele seja mostrado. Outras propriedades semelhantes poderiam ser definidas para tratar indicadores com valores enumerados, por exemplo mensagem não lida e caixa de entrada cheia.

4.4.4 Menu de Opções

O menu de opções é composto pelo conjunto de ações disponíveis numa determinada tela. Ele pode ser representado em XForms por meio de um ou mais elementos `trigger`, um para cada opção, contidos em um elemento `group`, com um atributo `appearance` que permite assoaciar a apresentação adequada.

O mapeamento dos atributos e relacionamentos das classes identificadas na Tabela 3.3 está indicado na Tabela 4.3, a seguir:

Classe	Atributo/Relac.	Correspondência em XForms
itemMenu	icone	Característica gráfica definida por uma propriedade CSS associada ao elemento `trigger` correspondente ao item do menu.
	habilitado	Definido pela propriedade `relevant` da instância do modelo associada ao elemento `trigger` correspondente ao item do menu.
	acao	Definido por um elemento `trigger` contido no elemento `group` correspondente ao menu.
	posicao	Definido pela ordem de declaração dos elementos `trigger` contidos nos elementos `group` correspondentes ao primeiro ou segundo níveis do menu.

Tabela 4.3 - Mapeamento para ItemMenu

É importante notar que o mapeamento dos itens de menu é uma extensão do usado para as ações, como indicado na Tabela 4.1. Por exemplo, considerando o menu da Figura 3.10:

```
<group appearance=”ext:menu”>
  <trigger>
    <label>
      <output value=”i18n(‘read’)”/>
    </label>
    <action ev:event=”DOMActivate”>
    ...
    </action>
  </trigger>
  ...
```

```
</group>
```

Neste exemplo, o menu corresponde ao elemento `group`. A ação “Read” corresponde ao elemento `trigger`. O rótulo desta ação é indicado pelo elemento `output` contido no elemento `label`. Quando a opção “Read” é selecionada, uma ou mais ações XForms, definidas dentro do elemento `action`, são executadas. As ações mais comuns são `send` para inicar uma tarefa ou operação do sistema, `toggle` para mudar de tela e `setvalue` para alterar um dado da instância do modelo.

O elemento `group` é usado também para representar opções em um segundo nível do menu. Neste caso, ele é inserido dentro do elemento `group` correspondente ao primeiro nível do menu, contendo um elemento `trigger` para cada opção do segundo nível e um elemento `label` para o rótulo da entrada correspondente no primeiro nível.

Para o caso de menus contextuais, onde uma ou mais opções podem ser desabilitadas de acordo com certas condições, os itens devem ser associados a instâncias do modelo, de forma que a propriedade `relevant` possa ser usada para controla-los. Por exemplo:

```
<model id=”dados_tarefa”>
  <instance>
    <dados xmlns=””>
      <acoes>
        <apagar/>
      </acoes>
      <mensagem>
        <protegida/>
      </mensagem>
    </dados>
  </instance>
  <bind id=”acao-apagar” nodeset=”/dados/acoes/apagar”
relevant=”/dados/mensagem/protegida = false()”/>
</model>

...

<trigger bind=”acao-apagar”>
  <label>
    <output value=”i18n(‘apagar’)”/>
  </label>
  ...
</trigger>
```

Neste exemplo, o elemento `trigger`, correspondente ao item de menu "Delete", está associado a uma instância do modelo, conforme indicado pelo atributo `bind`. O elemento da instância correspondente a esta associação é indicado pela expressão XPath, do tipo caminho de localização, definida no atributo `node-set`. A expressão XPath definida pelo atributo `relevant` controla o valor que a propriedade de mesmo nome tem naquele elemento da instância. Assim, o item de menu somente vai estar habilitado quando o elemento `/variables/message/protected`, que indica se a mensagem é protegida, for falso.

4.4.5 Listas de Seleção Simples

A lista de seleção simples (estática ou dinâmica) é composta por um conjunto de itens. Apenas um dos itens da lista aparece destacado, o qual pode ser utilizado para operações contextuais. Ela pode ser representada através do elemento `select1` de XForms.

O mapeamento dos atributos e relacionamentos das classes identificadas na Tabela 3.4 está indicado na Tabela 4.4, a seguir:

Classe	**Atributo/Relac.**	**Descrição**
ListaSimples	indiceDestaque	Indicado pelo valor da instância do modelo ligado ao elemento `select1`.
	apresentacao	Característica gráfica definida através do atributo `appearance` do elemento `select1` ou uma propriedade CSS associada a este elemento.
	itens	Definidos pelos elementos `item` ou `itemset`, contidos no elemento `select1` correspondente à lista.
Item	posicao	Implicito na ordem dos elementos `item`, ou dos dados da instância do modelo referenciados pelo elemento `itemset`.
	conteudo	Definido pelo elemento `label` contido no elmento `item` ou `itemset`.

	segundaLinha	Definido pelo elemento `label` contido no elemento `item` ou `itemset`.
	marcadores	Características gráficas definidas por propriedades CSS associadas ao elemento `item` ou `itemset`, em função do valor de um atributo ou conteúdo do item.
	imagem	Característica gráfica definida por uma propriedade CSS associada ao elemento `item` ou `itemset`, em função do valor de um atributo ou conteúdo do item.
TextoFormatado	formato	Definido pelo atributo `value` do elemento `output`, por meio de uma função XPath não padrão, por exemplo `i18n()`.

Tabela 4.4 - Mapeamento para ListaSimples, Item e TextoFormatado

O mapeamento para os eventos gerados pela lista simples está descrito a seguir:

- `navegação` – eventos `xforms-select` e `xforms-deselect` enviados ao elemento `item`.
- `itemSelecionado` – evento `DOMActivate` enviado ao elemento `item`.
- `filtroAnterior` – evento não padrão enviado ao elemento `select1`.
- `filtroSeguinte` – evento não padrão enviado ao elemento `select1`.

Por exemplo, considerando a lista estática da Figura 3.12:

```
<model id="dados-tarefa">
  <instance>
    <dados xmlns="">
      <selecao/>
    </dados>
  </instance>
  <bind id="selecao-lista" nodeset="/dados/selecao"/>
</model>

<model id="tarefa-gaa" schema="tarefa_gaa.xsd">
  <instance/>
  <submission id="iniciar-gaa" method="ext:call"
action="task://games-and-apps"/>
</model>
```

```
...

<select1 bind="selecao-lista" appearance="ext:simple-list">
  <label/>
    <item class="gaa">
      <label>
        <output value="i18n('games_and_apps')"/>
      </value>
      <value>0</value>
      <action ev:event="DOMActivate">
        <send submission="iniciar-gaa"/>
      </action>
    </item>
    ...
</select1>
```

Neste exemplo, o modelo `dados-tarefa` contém valores temporários da tarefa, sendo usado para armazenar qual item da lista está em destaque, como indicado pelo atributo `bind` do elemento `select1`. O valor de cada item é determinado pelo elemento `value`, podendo ser usado caso haja necessidade de fazer alguma operação contextual. Já o modelo `tarefa-gaa` é usado para permitir que a tarefa associada à aplicação "Games & Apps" seja iniciada, conforme indicado pelo elemento `submission`. A lista simples é definida pelo elemento `select1`, contendo um atributo `appearance` com o valor adequado. Os itens da lista, por ser estática, estão declarados um a um, utilizando elementos `item`. O valor `gaa` do atributo `class` permite que o item receba propriedades CSS específicas, relacionadas à aplicação "Games & Apps", como a imagem utilizada para o ícone do marcador. O texto formatado correspondente a cada item é definido pelo elemento `label`, o qual utiliza uma função XPath associada ao atributo `value` do elemento `output` para definir o valor na língua selecionada.

O alvo do evento `DOMActivate` é definido pela especificação W3C como sendo o controle XForms, no caso o elemento `select1`. Para permitir que uma ação seja associada ao item que estava em destaque no momento da ativação do controle, é necessário que o alvo deste evento seja o elemento `item` correspondente. No exemplo anterior, a ação associada à entrada "Games & Apps" dá inicio a uma tarefa, como definido pelo elemento `send` contido no elemento `action`.

Extensão 4: Elemento `item` como alvo do evento `DOMActivate`.

O exemplo a seguir demonstra o caso de uma lista dinâmica, como a da Figura 3.14:

```
<model id="dados-tarefa">
  <instance>
    <dados xmlns="">
      <selecao/>
    </dados>
  </instance>
  <bind id="selecao-lista" nodeset="/dados/selecao"/>
</model>

<model id="msgs-sumario" schema="msgs_sumario.xsd">
  <instance src="model://mensagens/sumario"/>
</model>

...

<select1 model="dados-tarefa" bind="dados-lista">
  <label/>
  <itemset model="msgs-sumario" nodeset="/mensagens/entrada">
    <label>
      <output value="i18n('msgs_sumario', indice, corpo)"/>
      <output appearance="ext:marker" class="estado"
ref="estado"/>
    </label>
    <value ref="id"/>
  </itemset>
  <item>
    <label ref="I18n('criar_mensagem')"/>
    <value/>
    <action ev:event="DOMActivate">
      <send submission="criar-mensagem"/>
    </action>
  </item>
</select1>
```

Neste exemplo, o modelo `msgs-sumario` contém informações sumarizadas de cada mensagem que está armazenada no dispositivo, como tipo, destinatário, assunto, início do texto, etc. Os itens da lista são obtidos deste modelo, como definido pelo atributo `nodeset` do elemento `itemset`. Os marcadores associados a cada item da lista são definidos por elementos `output`, com um atributo `appearance` adequado, contidos no elemento `label`.

O ícone de um marcador pode ser alterado de forma dinâmica por meio de propriedades CSS, em função do valor de um determinado elemento ou atributo da instância do modelo. Um exemplo de fragmento CSS para associar os ícones de estado é mostrado a seguir:

```
output.estado {
  item-mark1: msg_não_lida.png;
}

output.estado:contains("lida") {
  item-mark1: msg_lida.png;
}
```

4.4.6 Listas de Seleção Múltipla

A lista de seleção múltipla difere da simples pelo fato de permitir que um ou mais itens sejam marcados para seleção, permitindo que o usuário execute uma determinada operação sobre eles. Ela pode ser representada utilizando o elemento `select` de XForms.

O mapeamento dos atributos e relacionamentos das classes identificadas na Tabela 3.5 está indicado na Tabela 4.5, a seguir:

Classe	**Atributo/Relac.**	**Descrição**
ItemSelecionavel	selecionado	Indicado pelo valor da instância do modelo ligado ao elemento `select`.

Tabela 4.5 - Mapeamento para ItemSelecionavel

O exemplo dado anteriormente para a lista simples com conteúdo dinâmico também se aplica a este caso, alterando elemento `select1` por `select` e o atributo `appearance` para um valor adequado, como `mult-sel-list`. Uma diferença de comportamento é que o elemento `selecao` da instância do modelo irá conter multiplos identificadores separados por espaços, correspondentes ao valor de cada item selecionado. É importante que as operações invocadas pelo elemento `submission`, como a que permite apagar um conjunto de mensagens, suportem este tipo de representação em seus argumentos.

4.4.7 Formulários

O formulário é composto por um conjunto de campos de entrada de dados, podendo ser representado em XForms por um elemento `group`, com um atributo `appearance` adequado, contendo um ou mais dos seguintes elementos:

- `input`, para o caso de editores de data, hora, texto, número.
- `select1`, para o caso de valores enumerados que exigem um mapeamento para uma cadeia de caracteres correspondente.
- `trigger`, para o caso de ações.
- `range`, para o caso de faixa de valores.

O mapeamento dos atributos e relacionamentos das classes identificadas na Tabela 3.6 está indicado na Tabela 4.6, a seguir:

Classe	**Atributo/Relac.**	**Descrição**
Formulario	indiceDestaque	Pode ser determinado através do evento `xforms-focus` que é direcionado ao controle correspondente ao campo, no momento em que ele ganha o foco.
	campos	Conjunto de elementos `input`, `select1`, `trigger` ou `range`, contidos no elemento `group` correspondente ao formulário.
Campo	nome	Elemento `label` contido no elemento `input`, `select1`, `trigger` ou `range` correspondente ao campo do formulário.
CampoEdicao	editor	No caso de campos de data, hora, texto, ou númerico, o editor é definido de acordo com o tipo XML Schema do dado da instância ligado ao elemento `input` correspondente ao campo do formulário.
CampoAcao	acao	Elemento `trigger` contido no elemento `group` correspondente ao formulário.
CampoFaixa	valor	Dado da instância do modelo, com um tipo XML Schema numérico, ligado ao elemento `range` correspondente ao campo do formulário.
	max	Atributo `start` do elemento `range` correspondente ao campo do formulário.

	min	Atributo `end` do elemento `range` correspondente ao campo do formulário.
	passo	Atributo `step` do elemento `range` correspondente ao campo do formulário.
CampoEnumerado	valor	Dado da instância do modelo ligado ao elemento `select1` correspondente ao campo do formulário.
	valores	Elementos `item` contidos no elemento `select1` correspondente ao campo do formulário.
CampoData	valor	Dado da instância do modelo, com um tipo XML Schema de data, ligado ao elemento `input` correspondente ao campo do formulário.
CampoHora	valor	Dado da instância do modelo, com um tipo XML Schema de hora, ligado ao elemento `input` correspondente ao campo do formulário.
CampoTexto	valor	Dado da instância do modelo, com um tipo XML Schema de texto, ligado ao elemento `input` correspondente ao campo do formulário.
CampoNumerico	valor	Dado da instância do modelo, com um tipo XML Schema numérico, ligado ao elemento `input` correspondente ao campo do formulário.

Tabela 4.6 - Mapeamento para Formulario, Campo e classes derivadas

Por exemplo, no caso do formulário indicado na Figura 3.20:

```
<model id="tarefa-config-mms" schema="tarefa_config_mms.xsd">
  <instance/>
  <submission id="iniciar-config-mms" method="ext:call"
action="task://config-mms" replace="none"/>
</model>

<model id="dados-config" schema="dados_config.xsd">
```

```
  <instance src="model://configuracao"/>
  <submission id="atualizar-config-alerta"
ref="/config/alerta/estado" method="ext:call"
action="proc://atualizar-config-alerta" replace="none"/>
</model>

...

<group appearance="ext:form">
  <trigger>
    <label>
      <output value="i18n('mms_message_setup')"/>
    </label>
    <action ev:event="DOMActivate"/>
      <send submission="iniciar-config-mms"/>
    </action>
  </trigger>
  ...
  <select1 model="dados-config" ref="/config/alerta/estado">
    <label>
      <output value="i18n('alert')"/>
    </label>
    <item>
      <label>
       <output value="i18n('off')"/>
      </label>
      <value>0</value>
    </item>
    <item>
      <label>
        <output value="i18n('on')"/>
      </label>
      <value>1</value>
    </item>
    <action ev:event="xforms-value-changed"/>
      <send submission="atualizar-config-alerta"/>
    </action>
  </select1>
</group>
```

Neste exemplo, o formulário corresponde ao elemento `group`. O elemento `select1` foi utilizado para o campo de valores enumerados que permite ligar e desligar o alerta. O modelo `dados-config` contém os valores de configuração do dispositivo. É importante notar que os dados do modelo devem ser submetidos ao sistema para que a alteração na configuração seja efetivada. O elemento `trigger` corresponde à ação "MMS

Message Setup", que leva a um outro formulário. O modelo `tarefa-config-mms` é utilizado para dar inicio a tarefa correspondente a esta ação.

Para campos que não requerem um mapeamento de valores, como data e hora, o elemento `input` pode ser usado. Por exemplo, para o caso da Figura 3.25:

```
<group appearance="ext:form">
  ...
  <input model="dados-config" ref="/config/relogio/data">
    <label>
      <output value="i18n('date')"/>
    </label>
    <action ev:event="xforms-value-changed"/>
      <send submission="atualizar-config-data"/>
    </action>
  </input>
</group>
```

Nesse exemplo, o dado de instância `/config/relogio/data` contém a data do relógio do dispositivo, devendo ter um tipo XML Schema adequado para que o editor correspondente a este tipo de dado seja apresentado quando o usuário tentar modifica-lo.

4.4.8 Editores

Os editores são utilizados principalmente por formulários, conforme descrito anteriormente, através dos elementos `input`, `select1` ou `range` de XForms.

O mapeamento dos atributos e relacionamentos das classes identificadas na Tabela 3.7 está indicado na Tabela 4.7, a seguir:

Classe	**Atributo/Relac.**	**Descrição**
EditorTexto	conteudo	Dado da instância do modelo ligado ao elemento `input` correspondente ao editor.
EditorNumerico	conteudo	Dado da instância do modelo ligado ao elemento `input` correspondente ao editor.
EditorData	conteudo	Dado da instância do modelo ligado ao elemento `input` correspondente ao editor.
EditorHora	conteudo	Dado da instância do modelo ligado ao elemento `input` correspondente ao editor.

EditorEnumerado	selecao	Dado da instância do modelo ligado ao elemento `select1` correspondente ao editor.
	itens	Elementos `item` contidos no elemento `select1` correspondente ao editor.
EditorFaixa	conteudo	Dado da instância do modelo ligado ao elemento `input` correspondente ao editor.
	max	Atributo `start` do elemento `range` correspondente ao editor.
	min	Atributo `end` do elemento `range` correspondente ao editor.
	passo	Atributo `step` do elemento `range` correspondente ao editor.

Tabela 4.7 - Mapeamento para Editores

É importante notar que os editores também podem ser utilizados diretamente como conteúdo de uma tela, porém isto normalmente apenas ocorre no caso de editores de texto ou especializados, por exemplo o editor do corpo de mensagens de texto ou *email*. A especificação W3C atual para XForms somente suporta tipos simples para os dados da instância do modelo associado ao elemento `input`, sendo necessárias extensões para que tipos complexos, como o corpo de um email, sejam referenciados. Contudo, estas extensões podem ser feitas sem a adição de novos elementos ou atributos à linguagem.

A sessão 4.4.7 contém exemplos de editores usados em formulários.

4.4.9 Visualizadores

Vsualizadores são utilizados para apresentar dados ao usuário, correspondendo ao elemento `output` de XForms. O tipo apropriado de visualizador é selecionado em função do tipo XML Schema do dado da instância do modelo ligado ao elemento `output`.

O mapeamento dos atributos e relacionamentos das classes identificadas na Tabela 3.8 está indicado na Tabela 4.8, a seguir:

Classe	**Atributo/Relac.**	**Descrição**

VisualizadorTexto	conteudo	Dado da instância do modelo ligado ao elemento `output` correspondente ao editor. O atributo `value` pode ser utilizado para o caso de texto formatado por uma função XPath.
VisualizadorAgenda	eventos	Dado da instância do modelo ligado ao elemento `output` correspondente ao editor.
Evento	data	Tipo complexo da instância do modelo liagada ao elemento `output` correspondente ao editor.
	descricao	Tipo complexo da instância do modelo liagada ao elemento `output` correspondente ao editor.
	alarme	Tipo complexo da instância do modelo liagada ao elemento `output` correspondente ao editor.

Tabela 4.8 - Mapeamento para VisualizadorTexto e Evento

Por exemplo, no caso do visualizador de mensagens de texto da Figura 3.28:

```
<model id="visual-msg-args" schema="visual_msg_args.xsd">
  <instance id="visual-msg-args" src="task://input>
</model>

...

<output model="visual-msg-args" value="i18n('msg_detalhes',
/msg/texto, /msg/remetente, /msg/telefone, /msg/horario)"
appearance="ext:visual-msg-texto"/>
```

Neste exemplo, os detalhes da mensagem de texto foram fornecidos como argumento de entrada da tarefa de visualização, disponíveis no modelo `visual-msg-args`. O elemento `output` corresponde ao visualizador de mensagens de texto, conforme indicado pelo atributo `appearance`. Os dados são obtidos do modelo referenciado por expressões XPath e convertidos para texto formatado.

A especificação atual do W3C para XForms apenas suporta tipos simples associados ao elemento `output`. Para que tipos complexos sejam referenciados, como o conteúdo do

corpo de um email , é necessário fazer uma extensão à linguagem.

Extensão 5: Elemento `output` com suporte a tipos complexos.

4.4.10 Diálogos de Notificação e Confirmação

Diálogos de notificação correspondem ao elemento `message` de XForms, contudo, este elemento suporta apenas notificações temporárias (`ephemeral`) e diálogos de confirmação sem eventos de retorno para o documento (`modal`), tornando necessário fazer extensões à especificação do W3C.

O mapeamento dos atributos e relacionamentos das classes identificadas na Tabela 3.9 está indicado na Tabela 4.9, a seguir:

Classe	**Atributo/Relac.**	**Descrição**
Notificacao	iconeTipo	Característica gráfica definida por uma propriedade CSS associada ao elemento `message` correspodente à notificação.
	mensagem	Elemento `output` contido no elemento `message` correspondente à notificação.
Confirmacao	acaoEsquerda	Evento enviado ao elemento cujo atributo `id` está especificado em um atributo não padrão do elemento `message`.
	acaoDireita	Evento enviado ao elemento cujo atributo `id` está especificado em um atributo não padrão do elemento `message`.

Tabela 4.9 - Mapeamento para Notificacao e Confirmacao

A especificação atual do W3C para XForms não define eventos gerados pelo elemento `message`, visto que este elemento é uma ação e não um controle. Para o caso de diálogos de confirmação, é necessário estender a linguagem para permitir que a resposta do usuário seja retornada por meio de eventos, por exemplo `confirmation-ok` e `confirmation-cancel`. Isto pode ser feito permitindo que o identificador do elemento alvo do evento seja especificado como um atributo do elemento `message`, de forma similar ao atributo `target` do elemento `dispatch`, que também é uma ação.

Extensão 6: Eventos gerados pelo elemento `message`.

Também é necessário estender a linguagem para permitir definir uma mensagem como sendo transiente, de confirmação, etc. Na especificação atual do W3C para XForms, o atributo `level` suporta três valores, `ephemeral`, `modal` e `modeless`, além de nomes qualificados. O valor `ephemeral` pode ser utilizado para o caso de diálogos transientes. O valor `modal`, corresponde aos diálogos de confirmação, porém um único valor não permite distinguir entre os diversos casos para as ações da esquerda e direita. Uma solução é adicionar nomes qualificados para cada tipo possível, como `ext:modal-yes-no` e `ext:modal-ok-cancel`.

Extensão 7: Nomes qualificados para o atributo `level` do elemento `message`.

Por exemplo, para o diálogo de confirmação da Figura 3.33:

```
<action ev:event="DOMActivate">
  <message level="modal-yes-no" target="resposta">
    <output value="i18n('apagar_msg', /data/msg/indice)"/>
  </message>
</action>

...

<group id="resposta">
  <action ev:event="confirmation-ok">
  ...
  </action>
</group>
```

O elemento `output` contém uma expressão XPath que define o texto da mensagem. O tipo de mensagem é definido pelo atributo `level`, neste caso um diáologo de confirmação, com ações "Sim" e "Não". O evento de resposta da confirmação é enviado ao elemento `group` e tratado pela ação correspondente.

É importante notar que algumas notificações, como a da Figura 3.32 associada ao editor de um formulário, são um comportamento padrão do tipo de controle e, portanto, não requerem uma declaração no documento XForms.

4.4.11 Diálogo Inicial

O diálogo inicial é formado por uma série de campos de apresentação de dados e atalhos para iniciar tarefas, podendo ser representado por meio de um conjunto de elementos `output` e `trigger`, respectivamente, contidos em um elemento `group`.

O mapeamento dos atributos e relacionamentos das classes identificadas na Tabela 3.10 está indicado na Tabela 4.10, a seguir:

Classe	**Atributo/Relac.**	**Descrição**
DialogoInicial	relogio	Elemento `output` contido no elemento `group` correspondente ao diálogo inicial, contendo um atributo `appearance` adequado ou associado a uma propriedade CSS.
	operadora	Elemento `output` contido no elemento `group` correspondente ao diálogo inicial, contendo um atributo `appearance` adequado ou associado a uma propriedade CSS.
	informacao	Elemento `output` contido no elemento `group` correspondente ao diálogo inicial, contendo um atributo `appearance` adequado ou associado a uma propriedade CSS.
	atalhoCima	Elemento `trigger` contido no elemento `group` correspondente ao diálogo inicial, contendo um atributo `appearance` adequado ou associado a uma propriedade CSS.
	atalhoBaixo	Elemento `trigger` contido no elemento `group` correspondente ao diálogo inicial, contendo um atributo `appearance` adequado ou associado a uma propriedade CSS.
	atalhoEsquerda	Elemento `trigger` contido no elemento `group` correspondente ao diálogo inicial, contendo um atributo `appearance` adequado ou associado a uma propriedade CSS.

	atalhoDireita	Elemento `trigger` contido no elemento `group` correspondente ao diálogo inicial, contendo um atributo `appearance` adequado ou associado a uma propriedade CSS.
Atalho	evento	Evento `DOMActivate` enviado ao elemento `trigger` correspondente ao atalho quando ele é ativado.
	icone	Característica gráfica definida por uma propriedade CSS associada ao elemento `trigger` correspondente ao atalho.

Tabela 4.10 - Mapeamento para DialogoInicial

Por exemplo, para o diálogo inicial indicado na Figura 3.36:

```
<group appearance="dialogo-inicial">
  <output id="campo-relogio" bind="dados-relogio"/>
  <output id="campo-informacao" bind="dados-informacao"/>
  <output id="campo-operadora" bind="dados-operadora"/>
  <trigger id="atalho-esquerda">
    <label/>
    <action ev:event="DOMActivate">
    ...
    </action>
  </trigger>
  ...
</group>
```

Neste exemplo, o diálogo inicial corresponde ao elemento `group`. O atributo `id` é usado para identificar cada campo, permitindo que propriedades CSS sejam associadas a cada um deles, conforme necessário.

4.5 Temas e Estilos

Como descrito na sessão 3.6 , o estilo é caracterizado pelas propriedades de cada controle da interface de usuário, as quais podem ser alteradas dinamicamente sem afetar o conteúdo dos mesmos, mas apenas sua apresentação. Estas propriedades correspondem diretamente às folhas de estilo de CSS, que podem ser especificadas para cada documento XForms, como já visto nos diversos exemplos descritos na sessão anterior.

É importante observar que a especificação W3C para CSS não define todas as

propriedades necessárias para a interface de um dispositivo como o analisado neste trabalho, sendo portanto necessário adicionar novas propriedades não padrão.

Extensão 8: Novas propriedades não padrão para CSS.

4.6 Considerações Finais

As seguintes extensões são minimamente necessárias para que XForms possa ser utilizada como uma linguagem de definição da interface de usuário dos dispositivos considerados neste trabalho:

1. Novos eventos enviados aos elementos `submission` e `model`.
2. Função XPath para permitir selecionar nós de uma instância de um modelo diferente do utilizado pelo contexto atual.
3. Novos eventos para indicar que teclas especiais foram ativadas.
4. Elemento `item` como alvo do evento `DOMActivate`.
5. Elemento `output` com suporte a tipos complexos.
6. Eventos gerados pelo elemento `message`.
7. Nomes qualificados para o atributo `level` do elemento `message`.
8. Novas propriedades não padrão para CSS.

As extensões 1 e 3 não alteram a especificação atual, apenas adicionam novos tipos de eventos.

A extensão 2 é um relaxamento do comportamento atual descrito na especificação de XForms, que retorna um conjunto de nós vazios nesta situação. Esta restrição foi provavelmente imposta para simplificar a implementação da linguagem.

A extensão 4 altera o alvo do evento para um nível abaixo do definido pela especificação de XForms. Assim, o alvo atual, correspondente ao elemento `select` ou `select1`, continua no caminho do evento. É importante observar que o elemento `item` já é alvo de eventos, como `xforms-select`.

A extensão 5 apenas adiciona um suporte inexistente atualmente, visto que a especificação de XForms é restrita a tipos básicos.

As extensões 6 e 7 também apenas adicionam funcionalidades inexistentes atualmente em XForms. A extensão 6, em particular, é prevista na especificação do W3C.

A extensão 8 não altera a especificação CSS, apenas adiciona novas propriedades não padrão, prática já observada em outras soluções, como navegador *Mozilla*.

Conforme mostrado, o número reduzido de extensões necessárias à especificação do W3C, todas relativamente simples no sentido de não alterarem a estrutura da linguagem, sugere a possibilidade de uso de XForms como linguagem de definição para interfaces de usuário de dispositivos semelhantes ao modelo utilizado nesta análise.

Capítulo 5

Conclusões e Trabalhos Futuros

Este trabalho foi motivado pela constatação prática de que a constante evolução dos dispositivos portáteis, como telefones celulares, leva a uma série de problemas relacionados à implementação das mudanças necessárias no *software* responsável pela interface de usuário. As alterações no código das aplicações são difíceis de serem feitas, exigindo programadores especializados e grande tempo para serem desenvolvidas e testadas. A especificação das mudanças é freqüentemente ambígua, levando à necessidade de re-trabalhos no código. As soluções exploradas deveriam permitir que mudanças no fluxo e composição das telas e na apresentação gráfica dos componentes da interface de usuário fossem feitas sem a necessidade de alterações no código. Uma das principais vantagens desta abordagem seria permitir que especialistas em usabilidade realizassem diretamente estas mudanças, de forma ágil e sem introduzir defeitos no *software* dos dispositivos.

A solução proposta neste trabalho foi baseada na utilização de linguagens declarativas na especificação da interface de usuário de dispositivos portáteis. As principais considerações que guiaram a busca da solução foram: uso de tecnologias e *softwares* da Internet existentes atualmente; maior flexibilidade na realização de mudanças, inclusive de forma dinâmica; facilidade na criação de ferramentas para especificação da interface de usuário; redução do acoplamento entre o código da interface de usuário do responsável pelas funcionalidades do dispositivo, restringindo as mudanças de código à adição de novas funcionalidades.

Este trabalho envolveu o estudo de diversas linguagens declarativas baseadas em XML para definição de interfaces de usuário. Os critérios de comparação estabelecidos foram: ser um padrão aberto, viabilizando o uso de *softwares* livres; ser totalmente declarativa, para evitar a necessidade de programadores; ser independente de plataforma, de forma a poder ser utilizada em dispositivos portáteis; conter um modelo de dados, permitindo o acesso fácil às informações e estados do dispositivo; poder ser utilizada por navegadores, permitindo o reuso de código dos mesmos para a interface de usuário. Com base nestes critérios, XForms, um padrão para formulários na *Web*, foi considerado como uma alternativa para a definição de interfaces de dispositivos portáteis.

Para confirmar a possibilidade de uso de XForms como linguagem de definição de interfaces de usuário de dispositivos portáteis, foi realizada uma analise estática da interface de usuário de um telefone celular comercial. Desta análise resultou um modelo composto de um conjunto de diagramas de classe representando os diversos elementos da interface de usuário, além de uma descrição do modelo de tarefas e da arquitetura de *software* vigente.

Para viabilizar a solução, foi apresentada uma proposta de arquitetura baseada em uma adaptação do modelo cliente-servidor utilizado pelos navegadores para os quais XForms foi originalmente concebida. Esta arquitetura define alguns componentes: o gerenciador de tarefas interpreta os documentos XForms; as aplicações fazem o mapeamento dos dados e executam operações de forma independente da interface de usuário; a apresentação cria os controles da interface de usuário e aplica as propriedades gráficas indicadas pelas folhas de estilo CSS.

Em seguida foi realizado um mapeamento entre o modelo gerado e a especificação de XForms, identificando quais extensões seriam necessárias ao padrão definido atualmente pelo W3C para que ele possa ser usado na definição da interface de usuário dos dispositivos compatíveis com o modelo.

Concluímos que, de uma forma geral, XForms requer poucas extensões para poder ser usado como uma linguagem de definição de interface de usuário para os dispositivos considerados neste trabalho. Contudo, este resultado não pode ser generalizado para outros dispositivos sem que estudos maiores sejam conduzidos.

Os principais problemas previstos com a solução proposta seriam um maior uso de memória e de processamento. Isto poderia ser em parte solucionado dividindo os dados da aplicação em pequenos subconjuntos, ao invés de instanciar todos ao mesmo tempo. Notamos que diversos destes dispositivos já suportam navegadores com várias das tecnologias necessárias para a solução proposta, porém apenas a construção de um protótipo permitiria avaliar este impacto com mais exatidão.

O fato de XForms ser uma linguagem declarativa não a torna facil de ser utilizada no desenvolvimento de aplicações complexas, como pode ser observado no exemplo do Apêndice B. Isso reforça a importância de desenvolver ferramentas de auxílio à geração e manutenção dos documentos XForms.

Como trabalhos futuros podem ser consideradas as seguintes propostas:

- Criar protótipos da solução proposta neste trabalho, objetivando realizar uma análise de desempenho e produtividade.
- Criar ferramentas de auxílio ao desenvolvimento de interfaces de usuário que utilizem XForms como linguagem de definição.
- Realizar o mapeamento para a interface de outros dispositivos portáteis, na tentativa de generalizar as extensões propostas.
- Definir um vocabulário adequado para UIML e compara-lo à solução XForms proposta neste trabalho.
- Explorar o uso de ligações bidirecionais entre as instâncias do modelo XForms e os dados do sistema.

Além disso, seria possível utilizar o modelo de dispositivo portátil como base para outros trabalhos relacionados a interfaces de usuário.

A expectativa é que soluções semelhantes à apresentada sejam no futuro introduzidas em dispositivos comerciais, em particular telefones celulares. Um indicativo é a crescente demanda das operadoras de telefonia celular por interfaces de usuário que atendam os requisitos impostos por elas aos fabricantes destes equipamentos. Uma outra aplicação futura para este tipo de solução seria utilizar a capacidade de alteração dinâmica da interface de usuário como forma de adapta-la às características do usuário final, provendo diferentes fluxos e controles, por exemplo mais simples para usuários menos experientes e mais completos para usuários mais experientes (*power users*), aumentando assim a usabilidade do equipamento.

Referências Bibliográficas

[1] Ali, M. F. Pérez-Quiñones, M. A. Abrams, M., 2001. *A Multi-Step Process for Generating Multi-Platform User Interfaces using UIML*. Virginia Tech / Harmonia, pg. 3.

[2] Ali, M. F., 2004. *A Transformation-based Approach to Building Multi-Platform User Interfaces Using a Task Model and the User Interface Markup Language*. Tese (Doutorado). Virginia Polytechnic Institute and State University.

[3] Colton, P., 2005. *XAML: Microsoft's New Markup Language* [Web]. Disponível em http://www.byte.com/documents/s=9553/byt1110214556054/0307_colton.html [Consultado em setembro 2005].

[4] Dubinko, M., 2003. *XForms Essentials*. O'Reilly.

[5] Goderis, S., De Meuter, W., Brichau, J., 2002. *A case in Multiparadigm Programming: User Interfaces by means of Declarative Meta Programming*. Vrije Universiteit.

[6] Grolaux, D., Van Roy, P., Vanderdonckt, J., 2001. *QTk: An Integrated Model-Based Approach to Designing Executable User Interfaces*. DSV-IS 2001.

[7] Hackos, J., Redish, J., 1998. *User and Task Analysis for Interface Design*. Chichester: Wiley.

[8] Harmonia, 2002. *User Interface Markup Language (UIML) Specification v.3.0 First draft* [Web]. Disponível em: http://www.uiml.org/specs/ [Consultado em setembro 2005].

[9] IBM, 2004. *Abstract User Interface Markup Language Toolkit* [Web]. Disponível em: http://www.alphaworks.ibm.com/tech/auiml [Consultado em setembro 2005].

[10] Kost, S., 2004. *Dynamically generated multimodal application interfaces*.

[11] Luyten, K., Coninx, K., 2001. *An XML-based runtime user interface description language for mobile computing devices*. Limburgs Universitair Centrum. Springer-Verlag.

[12] Merrick, R. A., 2001. *Device Independent User Interfaces in XML* [Web]. Disponível em: http://xml.coverpages.org/MerrickBelCHI.pdf [Consultado em setembro 2005].

[13] Microsoft, 2005. *Windows Vista* [Web] Disponível em: http://msdn.microsoft.com/windowsvista/about/ [Consultado em setembro 2005].

[14] Mozilla.org, 2003. *XUL Project* [Web] Disponível em: http://www.mozilla.org/projects/xul/ [Consultado em setembro 2005].

[15] Myers, B. A., Rosson, M. B., 1992. *Survey on User Interface Programming*. ACM.

[16] Myers, B. A., 1994. *Challenges of HCI Design and Implementation*. ACM Interactions.

[17] Myers, B. Hudson, S. E. Pausch R., 2000. *Past, Present and Future of User Interface Software Tools*. Carnegie Mellon University, ACM Transactions on Computer-Human Interaction, pg. 4, 15, 20.

[18] O'Reilly, 2004. *Inside XAML* [Web] Disponível em: http://www.ondotnet.com/pub/a/dotnet/2004/01/19/longhorn.html [Consultado em setembro 2005].

[19] Phanouriou, C., 2000. *UIML: A Device-Independent User Interface Markup Language*. Tese (Doutorado).Virginia Polytechnic Institute and State University, pg. 3, 7, 9, 10, 16, 17, 19, 50.

[20] Pinheiro da Silva, P., 2000. *User Interface Declarative Models and Development Environments: A Survey*. Berlin: Springer-Verlag.

[21] Puerta, A., Eisenstein, J., *1999. Towards a general computational framework for model-based interface development systems (MOBI-D)*. ACM Press.

[22] Puerta, A., Eisenstein, J., *XIML: A Universal Language for User Interfaces*. RedWhale Software.

[23] Raman, T. V., 2003. *XML Powered Web Forms*. Addison-Wesley.

[24] Schlungbaum, E., 1996. *Model-based User Interface Software Tools - Current state of declarative models*. Georgia Institute of Technology.

[25] Uotila, A., 2000. *A User Interface Toolkit for a Small Screen Device*. Tese (Mestrado). University of Tampere.

[26] Usability Net, 2003. *Task Analysis Methods* [Web]. Disponível em: http://www.usabilitynet.org/tools/taskanalysis.htm [Consultado em setembro 2005].

[27] Van Roy, P., Haridi, S., 2003. *Concepts, Techniques, and Models of Computer Programming*. Swedish Institute of Computer Science.

[28] Zelkowitz, M. V., Wallace, D. R., 1998. *Experimental Models for Validating Technology*, IEEE.

[29] W3C, 2004. *Extensible Markup Language (XML) v1.1* [Web]. Disponível em: http://www.w3.org/TR/2004/REC-xml-20040204/ [Consultado em setembro 2005].

[30] W3C, 2004. *XML Schema* [Web]. Disponível em: http://www.w3.org/XML/Schema [Consultado em setembro 2005].

[31] W3C, 1999. *XPath v1.0* [Web]. Disponível em: http://www.w3.org/TR/xpath [Consultado em setembro 2005].

[32] W3C, 2000. *Document Object Model (DOM) Level 2 Core Specification v 1.0* [Web]. Disponível em: http://www.w3.org/TR/DOM-Level-2-Core/ [Consultado em setembro 2005].

[33] W3C, 2003. *XML Events* [Web]. Diponível em: http://www.w3.org/TR/xml-events/ [Consultado em setembro 2005].

[34] W3C, 2003. *XForms v1.0* [Web]. Disponível em: http://www.w3.org/TR/xforms/ [Consultado em setembro 2005].

[35] W3C , 2005. *XForms - The Next Generation of Web Forms* [Web]. Disponível em: http://www.w3.org/MarkUp/Forms/ [Consultado em setembro 2005].

[36] W3C, 1998. *Cascading Style Sheets (CSS) Level 2* [Web]. Disponível em: http://www.w3.org/TR/1998/REC-CSS2-19980512/ [Consultado em setembro 2005].

[37] W3C. *CSS3 UI Module*. Disponível em: http://www.w3.org/TR/css3-ui/ [Consultado em setembro 2005].

[38] WAP Forum. *Wireless Mark-up Language (WML) v2.0 Specification* [Web]. Disponível em: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html [Consultado em setembro 2005].

Apêndice A

Tecnologias da Web Relacionadas a Interfaces de Usuário

A Internet motivou o desenvolvimento de diversas tecnologias necessárias à transferência, armazenamento e apresentação de dados. O W3C (*World Wide Web Consortium*) é uma organização que procura estabelecer padrões (recomendações), garantindo a interoperabilidade entre os diversos sistemas utilizados na Internet. Além disso, estes padrões permitem o desenvolvimento de *software* aberto e o uso dos mesmos como alternativa aos sistemas proprietários. A seguir estão descritas, de forma resumida, algumas destas tecnologias relacionadas a interfaces de usuário.

A.1 XML

XML (*Extensible Markup Language*) é uma recomendação W3C, definida em [29], de uma meta-linguagem para descrever a estrutura e conteúdo dos chamados "documentos XML". Os documentos XML podem ser utilizados para representar dados estruturados, contendo caracteres de dados e de marcação (*markup)*. O termo *markup* se refere aos indicadores de inicio e fim, separadores, comentários, instruções de processamento, entre outros, os quais definem a estrutura lógica e atributos do documento.

Um documento XML é composto por elementos. O tipo de um elemento é indicado pelo seu nome, sendo cada ocorrência no documento XML uma instância daquele elemento. Os elementos podem ser vazios ou conter outros elementos, inclusive do mesmo tipo, ou ainda seqüências de caracteres (texto). O nível mais alto do documento XML deve conter obrigatoriamente um único elemento. O conteúdo de um elemento é delimitado por meio de marcadores de início e fim, formados pelos caracteres '<', '>' e '/', como exemplificado abaixo:

```
<forma>
   <descricao>Uma roda</descricao>
   <circulo/>
</forma>
```

O elemento `forma` contém os elementos `descricao` e `circulo`. O elemento `descricao` contém o texto "Uma roda". É importante notar que uma sintaxe abreviada

pode ser usada no caso de elementos vazios, dispensando o marcador de fim, como no caso do elemento `circulo`.

É posssível associar atributos a um elemento. Os atributos são formados por pares nome e valor, declarados dentro do marcador de início do elemento. Os pares são separados entre si por espaços. Cada par é formado pelo nome do atributo, seguido pelo caractere '=' e pelo respectivo valor delimitado por aspas duplas. Por exemplo:

```
<forma id=”roda”>
   <descricao>Uma roda</descricao>
   <circulo x=”10” y=”10” raio=”5”/>
</forma>
```

No exemplo acima, o elemento `forma` tem um atributo `id` cujo valor é "roda". O elemento `circulo` tem os atributos `x`, `y` e `raio`, com valores "10", "10" e "5" respectivamente.

O documento XML pode conter comentários, delimitados pelas seqüências de caracteres `<!--` e `-->` . Por exemplo:

```
<!-- Isto é um comentário -->
```

Os documentos XML devem obrigatoriamente começar com uma declaração que indica a versão da recomendação W3C em uso:

```
<?xml version="1.0"?>
```

Podemos dizer que um determinado conjunto de elementos e atributos define um vocabulário, ou linguagem, baseado em XML. Para permitir que um mesmo documento XML utilize elementos de mais de uma linguagem sem que ocorra colisão entre os nomes dos elementos e atributos, cada linguagem deve estar associada a um espaço de nomes (*namespace*) diferente. Cada espaço de nomes é definido por um prefixo, o qual deve estar associado a uma URI (*Uniform Resource Indicator*). O atributo `xmlns` é utilizado para estabelecer esta associação. Por exemplo:

```
<mapa xmlns=”urn:geo” xmlns:frm=”urn:formas”>
  <cidade>
    <nome>Campinas</nome>
```

```
    <frm:forma>
      ...
    </frm:forma>
  </cidade>
  ...
</mapa>
```

Neste exemplo, o espaço de nomes *default* está associado aos elementos e atributos `mapa`, `cidade` e `nome`, que descrevem informações de um mapa, e o espaço de nomes correspondente ao prefixo `frm` está associado ao elemento `forma`, que descreve uma forma gráfica.

A.2 DOM

DOM (*Document Object Model*) é uma recomendação W3C, definida em [32], para uma interface independente de plataforma que permite a programas e *scripts* acessarem dinamicamente e atualizarem o conteúdo e estrutura de documentos XML. Na interface DOM, um documento XML é representado como uma árvore de objetos, cada qual contendo um conjunto de atributos e métodos, de acordo com o modelo de classes simplificado abaixo baseado na especificação *DOM Level 2*:

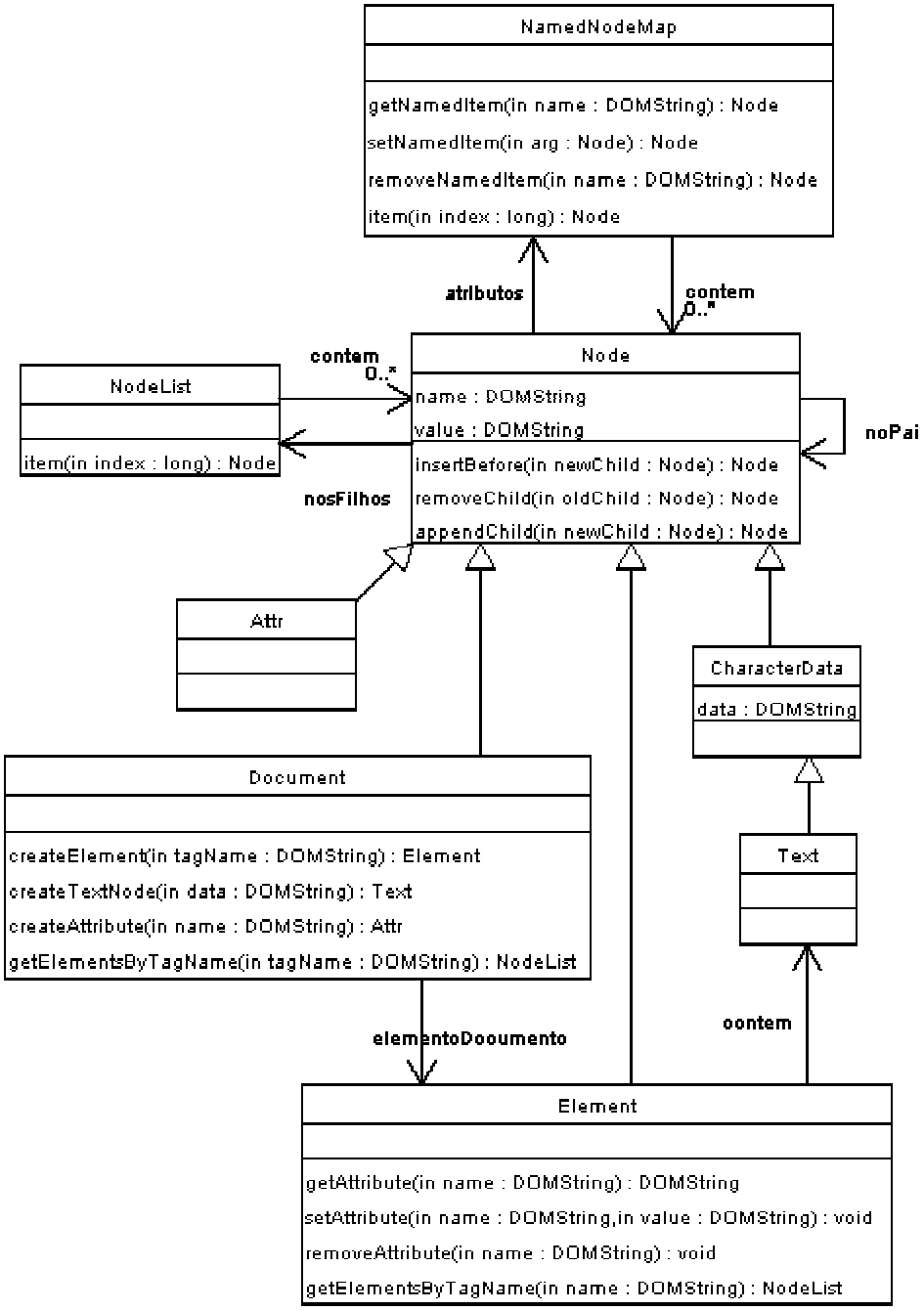

Figura A.1 - Modelo DOM (simplificado)

Neste diagrama, é possível ver que os nós da árvore DOM são derivados de uma classe comum, `Node`. Esta classe suporta operações de inserção e remoção de nós na árvore, tendo uma relação para o nó "pai". Os objetos da classe `NodeList` contêm referências para um conjunto de nós, sendo usada pela classe `Node` para referenciar os nós "filhos" de um elemento. Os nós podem ser de várias classes especializadas, como `Document`, que representa o documento em si, e `Element`, que representa cada um dos

elementos contidos no documento. A classe `Attr` corresponde aos atributos de um elemento, sendo seus objetos referenciados através da classe `NamedNodeMap`.

A árvore DOM é normalmente construída por um programa (também chamado de processador XML) através da leitura do documento XML de entrada. Esta árvore pode ser então consultada e modificada por outros programas através da interface DOM (representada na Figura .1 através dos métodos de cada classe). É importante notar que a árvore pode ser construída totalmente de forma programática. Após ser construída ou modificada, a árvore pode ser transformada em um documento XML de saída, o qual pode ser gravado como um arquivo de texto.

A.3 XML Events

XML Events é uma recomendação W3C, definida em [33], para um mecanismo de tratamento de eventos integrado à interface DOM, permitindo associar comportamentos a documentos XML. XML Events é uma linguagem baseada em XML. O espaço de nomes associado ao vocabulário XML Events é http://www.w3.org/2001/xml-events.

XML Events considera como um evento qualquer acontecimento associado a um elemento, chamado de "alvo", de um documento XML representado por uma árvore DOM. Ações podem ser associadas a elementos do documento, chamados de "observadores", de forma a serem executadas por meio de rotinas de tratamento (*handlers*), em resposta à ocorrência de um determinado evento. O modelo de propagação de eventos e execução de ações está representado a seguir na Figura A.2:

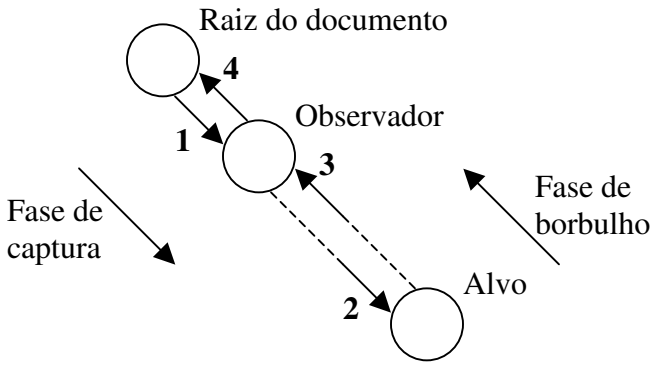

Figura A.2 - Modelo de propagação XML Events

Inicialmente, na chamada fase de captura (*capture*), o evento parte da raiz do documento em direção ao elemento alvo. Assim que ele é atingido, a fase de borbulho (*bubbling*) é iniciada, na qual o evento retorna em direção à raiz do documento. Em ambas

as fases, quaisquer ações que estejam porventura associadas àquele evento nos nós do percurso serão executadas.

A associação entre evento e ação é feita pelo elemento `listener`. O atributo mandatório `event` especifica o nome do evento a ser tratado. Os atributos opcionais `target` e `observer` contêm o identificador dos elementos alvo e observador, respectivamente. O atributo opcional `handler` contém uma URI correspondente a rotina de tratamento. O atributo opcional `phase` permite especificar se a execução da rotina de tratamento deve ocorrer na fase de captura (`capture`) ou borbulho (`default`). O atributo opcional `propagate` permite determinar se o evento deve continuar a se propagar (`continue`) ou não (`stop`). O opcional atributo `defaultAction` permite especificar se a ação *default* deve ser executada ao final (`perform`) ou não (`cancel`). Por exemplo:

```
<listener event=”DOMActivate” observer=”botao” handler=”#ac”>
```

Neste exemplo, a rotina de tratamento `#ac` vai ser executada quando o evento `DOMActivate` atingir o elemento cujo atributo `id` tem o valor `botao`.

Também é possível estabelecer a associação entre evento e ação diretamente no elemento observador, utilizando os mesmos atributos descritos anteriormente. Por exemplo:

```
<trigger>
  <label>Ok</label>
  <action ev:event="DOMActivate">
    <send submission=”processar"/>
  </action>
</trigger>
```

Se o atributo `observer` for omitido, então o observador será o elemento “pai” daquele no qual foi especificado o atributo `event` (no exemplo acima o observador será o elemento `trigger`). Se o atributo `handler` for omitido, o tratamento será feito pelo próprio elemento no qual o atributo `event` foi especificado (no exemplo acima a ação será executada pelo elemento `action`).

A.4 XML Schema

XML Schema é uma recomendação W3C, definida em [30], de uma linguagem XML que permite expressar restrições de sintaxe, estrutura e valores para instâncias de documentos XML. O espaço de nomes associado ao vocabulário XML Schema é http://www.w3.org/2001/XMLSchema.

Cada elemento e atributo de um documento XML pertence a um determinado tipo. Os tipos podem ser simples ou complexos. Tipos simples são definidos por XML Schema e não podem ser alterados, apenas estendidos. Alguns tipos simples mais comuns estão indicados na Tabela A.1, a seguir:

Tipo Simples	**Descrição**
string	Seqüência de caracteres
normalizedString	Seqüência de caracteres, excluindo retornos de linhas e tabulações
token	Usado para representar uma sequencia de uma ou mais palavras. Restrição ao tipo `normalizedString`, não permitindo quebras de linha/tabulações, espaços no inicio/fim, e dois ou mais espaços entre palavras
base64Binary	Dados binários codificados em base 64
hexBinary	Dados binários codificados em hexadecimal
integer	Números inteiros
positiveInteger	Números inteiros positivos maiores que zero
negativeInteger	Números inteiros negativos menores que zero
nonNegativeInteger	Números inteiros positivos, incluindo zero
nonPositiveInteger	Números inteiros negativos, incluindo zero
long	Números inteiros positivos e negativos de 64 bits
unsignedLong	Números inteiros positivos de 64 bits
int	Números inteiros positivos e negativos de 32 bits
unsignedInt	Números inteiros positivos de 32 bits
short	Números inteiros positivos e negativos de 16 bits
unsignedShort	Números inteiros positivos de 16 bits
byte	Números inteiros positivos e negativos de 8 bits
unsignedByte	Números inteiros positivos de 8 bits
decimal	Números decimais
float	Números de ponto flutuante (precisão simples)
double	Números de ponto flutuante (precisão dupla)
boolean	Booleanos (true, false, 1, 0)
duration	Duração de tempo
dateTime	Data e hora
date	Data
time	Hora

gYear	Ano
gYearMonth	Mês e ano
gMonth	Mês
gMonthDay	Dia e mês
gDay	Dia
anyURI	Endereço no formato URI
language	Língua

Tabela A.1 - Tipos definidos por XML Schema

A definição de elementos é feita através do elemento `element`. O atributo `name` indica o nome do elemento e o atributo `type` indica o seu tipo. Por exemplo:

```
<element name="data" type="date"/>
```

Um fragmento de um documento XML correspondente a este exemplo seria:

```
<data>2005-12-31</data>
```

Os tipos dos elementos podem ser simples ou complexos. O atributo `type` pode ser omitido, e o tipo definido imediatamente após a definição do elemento. É possível ainda utilizar os atributos `maxOccurs` e `minOccurs` para restringir o número de vezes que o elemento pode ser declarado, em relação ao elemento imediatamente mais externo do documento.

De forma semelhante aos elementos, a definição de atributos é feita através do elemento `attribute`. O atributo `name` indica o nome do atributo e o atributo `type` seu tipo. Por exemplo:

```
<element name="diretorio">
  <complexType>
    <attribute name="caminho" type="string" />
  </complexType>
</element>
```

É importante notar que os atributos são sempre de tipo simples. É possível ainda utilizar os atributos `default` para indicar o valor inicial de um atributo, e o atributo `use` para indicar se ele é mandatário (`required`).

A definição de um tipo complexo pode ser feita através do elemento `complexType`.

O atributo `name` indica o nome do tipo que esta sendo definido. O elemento `sequence` permite definir o conjunto de elementos que podem ser inseridos imediatamente abaixo do elemento que esta sendo definido. Por exemplo:

```
<complexType name=”TipoEntrada”>
  <sequence>
    <element name="nome" type=”string”>
    <element name="data" type=”date”>
  </sequence>
</complexType>
```

Um elemento do tipo `TipoEntrada`, definido acima, pode conter os elementos `nome` e `data`, cujos tipos são `string` e `date`, respectivamente.

Novos tipos podem ser criados a partir de outros, por extensão ou restrição. No primeiro caso é usado o elemento `extension`, o qual permite adicionar novos elementos e atributos ao tipo base. No segundo caso é usado o elemento `restriction`, o qual permite excluir elementos e atributos do tipo base ou restringir os valores permitidos. Em ambos os casos o atributo `base` indica o nome do tipo base.

Os tipos complexos podem sempre ser definidos pela restrição de um outro tipo complexo ou pela extensão de um outro tipo simples ou complexo. É possível definir novos tipos simples através da restrição de outros tipos simples. Por exemplo:

```
<simpleType name=”TemperaturaAgua”>
 <restriction base=”number”>
  <fractionDigits value=”2”/>
  <minExclusive value=”0.00”/>
  <maxExclusive value=”100.00”/>
 </restriction>
</simpleType>
```

Neste exemplo, o tipo básico `number` teve seus valores restritos a faixa 0 a 100.

Os seguintes elementos podem ser usados na criação de novos tipos por restrição:

Elemento	**Descrição**
minExclusive	Limitante inferior exclusivo
minInclusive	Limitante inferior inclusivo
maxExclusive	Limitante superior exclusivo

maxInclusive	Limitante superior inclusivo
totalDigits	Número máximo total de dígitos
fractionDigits	Número máximo de dígitos fracionários
length	Tamanho exato
minLength	Tamanho mínimo
maxLength	Tamanho máximo
enumeration	Enumerado de valores
whiteSpace	Substituir, preservar ou remover espaços em branco
pattern	Expressão regular

Tabela A.2 - Elementos de Derivação por Restrição

Para ilustrar melhor o uso da linguagem XML Schema, considere o conteúdo de um determinado diretório do sistema de arquivos representado por meio de um documento XML. Uma instância deste documento poderia ser definida pelo seguinte documento XML Schema:

```
<schema xmlns=”http://www.w3.org/2001/XMLSchema”>
  <element name=”diretório” type”TipoDiretorio” />

  <complexType name=”TipoDiretorio”>
    <element name=”entrada" type=”Entrada” />
    <attribute name=”caminho” type=”string”/>
  </complexType>

  <complexType name=”TipoEntrada”>
    <sequence>
      <element name=”nome” type=”string”>
      <element name=”data” type=”date”>
      <element name=”tipo”>
        <simpleType>
          <restriction base="string">
            <enumeration value="imagem"/>
            <enumeration value="som"/>
            <enumeration value="diretorio"/>
          </restriction>
       </simpleType">
    </sequence>
  </complexType>

</schema>
```

Uma instância de documento XML compatível com este esquema seria:

```
<diretorio caminho=”/imagens”>
  <entrada>
```

```
    <nome>a.jpg</nome>
    <tipo>imagem</tipo>
  </entrada>
  <entrada>
    <nome>b.jpg</nome>
    <tipo>imagem</tipo>
  </entrada>
</diretorio>
```

É importante notar que o esquema acima representa um único diretório, e não a estrutura completa do sistema de arquivos. A vantagem desta representação é que ela requer bem menos nós e, portanto, memória.

A.5 XPath

XPath (*XML Path Language*) é uma recomendação W3C, definida em [31], de uma linguagem declarativa utilizada para referenciar partes de documentos XML e realizar operações sobre elas. Esta linguagem não é basseada em XML, sendo formada pelas chamadas expressões XPath e o modelo de dados sobre a qual elas operam, correspondente ao documento XML que esta sendo processado. As expressões XPath são normalmente usadas como atributos de elementos de um documento XML.

A.5.1 Modelo de Dados

O modelo de dados XPath é uma árvore de nós, correspondente ao documento XML que esta sendo processado. Existem sete tipos possíveis de nós possíveis:

- Elemento
- Atributo
- Texto
- Espaço de nomes (“namespace”)
- Instrução de processamento
- Comentário
- Raiz

A relação entre os diversos tipos de nós da árvore XPath pode ser vista através do diagrama de classes mostrado na Figura A.3, a seguir:

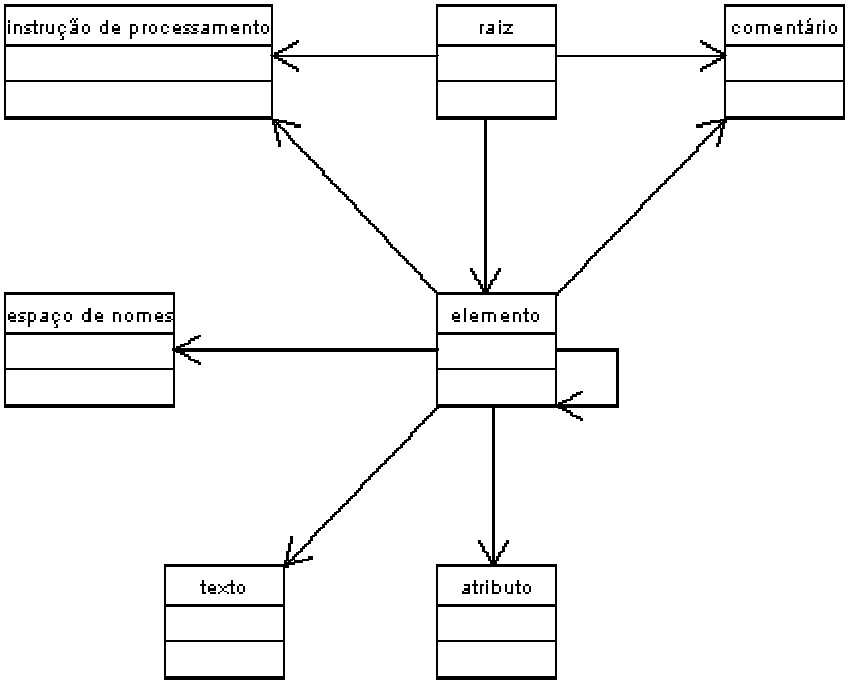

Figura A.3 - Relação entre Nós XPath

A árvore XPath é semelhante mas não identica a árvore DOM. A árvore XPath pode conter apenas um nó do tipo `raiz`, o qual contém um nó do tipo `elemento`, correspondente ao elemento mais externo do documento XML. O nó raiz também pode conter comentários e instruções de processamento que aparecem fora do documento XML. Nós do tipo `elemento` podem conter quaisquer tipos de nós, exceto o nó `raiz`. Todos os nós são obrigatoriamente folhas, exceto pelos nós do tipo `raiz` e `elemento`.

Como exemplo, consideremos o seguinte fragmento de documento XML:

```
<relogio id=”1”>
  <data>15</data>
  <hora>12</hora>
</relogio>
```

A árvore XPath correspondente a este documento é mostrada na Figura A.4, a seguir:

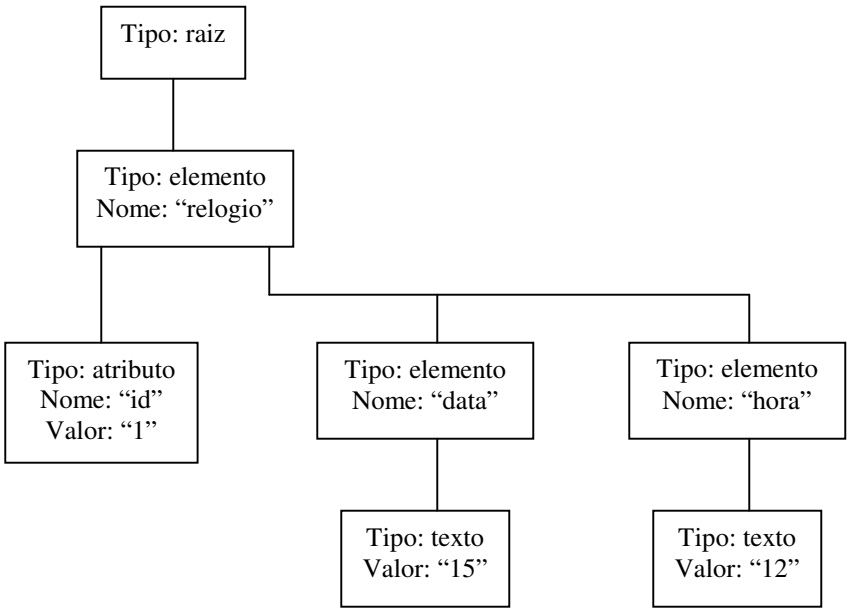

Figura A.4 - Exemplo Árvore XPath

Por este exemplo é possível notar que as árvores XPath podem conter um número bastante grande de nós em relação ao documento XML que está sendo processado.

A.5.2 Expressões

As expressões XPath são formadas pela composição de diferentes elementos:

- Funções
- Caminhos de localização
- Operadores
- Tipos básicos

Estas expressões, ao serem avaliadas, retornam um objeto de um dos tipos definidos pela linguagem. As expressões são avaliadas em relação a um contexto, formado, entre outros atributos, por um nó denominado “nó de contexto”, o qual representa um ponto de referência na árvore XPath. A sintaxe completa das expressões XPath pode ser encontrada nas especificações do W3C e não será vista aqui em detalhes.

A.5.3 Tipos de Objetos

XPath suporta objetos do tipo numérico com ponto flutuante, cadeias de caracteres, booleano (`true` e `false`) e conjunto de nós ordenado e sem duplicatas. Os tipos são implicitamente convertidos quando necessário.

A.5.4 Caminhos de Localização

Um caminho de localização (*Location Path*) permite selecionar um conjunto de zero ou mais nós da árvore XPath a partir do nó de contexto. Caminhos podem ser relativos ou absolutos. Caminhos relativos são compostos de uma seqüência de um ou mais passos de localização separados por "/", interpretados da esquerda para a direita. Cada passo utiliza o conjunto de nós selecionados no passo anterior. Caminhos absolutos são iniciados por "/", que seleciona a raiz do documento que contém o nó de contexto, seguido opcionalmente por um caminho relativo. Um passo de localização tem a seguinte forma:

Eixo::*Teste*[*Predicado*]

XPath define diversos eixos partindo de um nó. Cada eixo determina a "direção" de seleção de nós a partir do contexto atual, retornando um conjunto de nós (potencialmente vazio). Os possíveis eixos são mostrados na Tabela A.3, a seguir:

Eixo	**Conjunto de nós**
self	Apenas o nó de contexto.
child	Nós filhos do nó de contexto. Pode ser omitido.
attribute	Atributos do nó de contexto.
parent	Pai do nó de contexto.
descendant	Descendentes do nó de contexto.
descendant-or-self	Descendentes do nó de contexto ou ele próprio.
ancestor	Ancestrais do nó de contexto.
ancestor-or-self	Ancestrais do nó de contexto ou ele próprio.
following	Nós seguintes ao de contexto, na ordem do documento.
following-sibling	Nós irmãos seguintes ao nó de contexto.
preceding	Nós anteriores ao de contexto, em ordem reversa do documento.
preceding-sibling	Nós irmãos anteriores ao nó de contexto.
namespace	Nomes de espaço do nó de contexto.

Tabela A.3 - Eixos XPath

Cada eixo tem um tipo principal de nó. Se um eixo pode conter elementos então seu tipo principal é elemento, caso contrário:

- Eixo atributo – tipo principal é atributo.
- Eixo espaço de nomes – tipo principal é espaço de nomes.
- Demais tipos de eixo – tipo principal é elemento.

Os possíveis testes são descritos na Tabela A.4, a seguir:

Teste	Descrição
nome	Verdadeiro para o nó do tipo principal que tem este nome.
*	Verdadeiro para todos os nós to tipo principal.
text()	Verdadeiro para qualquer nó do tipo texto.
comment()	Verdadeiro para qualquer nó do tipo comentário.
processing-instruction(string)	Verdadeiro para qualquer nó do tipo instrução de processamento com este nome.
node()	Verdadeiro para qualquer nó de qualquer tipo.

Tabela A.4 - Testes XPath

Predicados permitem filtrar um conjunto de nós, produzindo um sub-conjunto contendo apenas os nós que satisfazem uma determinada condição. Um predicado é formado por uma expressão composta de funções e operadores booleanos, a qual é avaliada para cada um dos nós do conjunto original. Apenas os nós para os quais esta expressão retornar verdadeiro serão incluídos no conjunto resultante do caminho de localização.

Uma sintaxe abreviada foi definida para os caminhos de localização mais comuns, indicada na Tabela A.5, a seguir:

Forma abreviada	Forma completa
vazio	child::
//	/descentent-or-self::node()/
.	self::node()
..	Parent::
@	attribute::

Tabela A.5 - Sintaxe Abreviada para Caminhos de Localização

A.5.5 Funções

Funções XPath possuem zero ou mais argumentos. Os argumentos e o resultado da função são de um dos quatro tipos básicos descritos anteriormente. Além das funções da biblioteca de XPath, outras podem ser adicionadas. Algumas das funções da biblioteca XPath estão indicadas na Tabela A.6, a seguir:

Retorno	Função	Descrição
Numérico	last()	Tamanho do contexto
	position()	Posição dentro do contexto
	count(node-set)	Número de nós no conjunto
	number(object)	Transforma um objeto em um número

	sum(number, number)	Soma os dois argumentos
	floor(number)	Faz o arredondamento para baixo do argumento
	ceiling(number)	Faz o arredondamento para cima do argumento
	string-length(string)	Calcula o comprimento da cadeia de caracteres
Booleano	contains(string, string)	Retorna verdadeiro se a primeira cadeia de caracteres contem a segunda
	boolean(object)	Converte um objeto para o tipo booleano
	false()	Retorna o tipo falso
	not(boolean)	Valor negado do argumento
	true()	Retorna o tipo verdadeiro
	starts-with(string, string)	Verdadeiro se a primeira cadeia de caracteres começa com a segunda
String	string(object)	Converte um objeto emuma cadeia de caracteres
	concat(string, string)	Concatena duas cadeias de caracteres
	name(node-set)	Retorna o nome do nó
Conjunto de nós	id(object)	Retorna o conjunto de nós que tem o atributo "id" igual ao argumento

Tabela A.6 - Funções Definidas por XPath

A.5.6 Operadores

As expressões XPath podem utilizar os seguintes operadores, aplicáveis de acordo com o tipo do(s) operando(s):

Tipo do(s) Operando(s)	Operador	Descrição
Booleano	>	Maior
	>=	Maior ou igual
	<	Menor
	<=	Menor ou igual
	=	Igual
	!=	Diferente
	and	E lógico
	or	OU lógico
Numérico	+	Soma
	-	Subtração
	mod	Resto da divisão inteira
	div	Divisão
	*	Multiplicação
Conjunto de nós	|	União de conjuntos de nós

Tabela A.7 - Operadores XPath

A.5.7 Exemplos de expressões

Para melhor entender como os diversos elementos descritos são utilizados na formação de expressões XPath, iremos utilizar o seguinte documento XML:

```
<a>
  <b x=”1”>
    <c>T1</c>
  </b>
  <b x=”2”>
    <c>T2</c>
  </b>
</a>
```

A Tabela A.8, a seguir, mostra alguns exemplos de expressões XPath e o resultado correspondente para este documento:

Expressão	**Resultado**
/a/b	Conjunto contendo ambos os dois nós correspondentes aos elementos `b`.
/a/b[@x=’2’]	Conjunto contendo apenas o nó correspondente ao elemento `b` cujo atributo `x` tem o valor `x2`.
/a/b[2]	Conjunto contendo apenas o nó correspondente ao segundo elemento `b`.
/a/b[1] | /a/b[2]	Conjunto contendo os dois nós correspondentes ao primeiro e segundo elemento `b`.
/a/b[2]/parent::*	Conjunto contendo o nó correspondente ao elemento `a`.
/a/descendant::*	Conjunto contendo os quatro nós correspondentes aos elementos `b` e `c`.
/a/b/@*	Conjunto contendo os dois nós correspondentes aos atributos “1” e “2”.
/a/b/c/text()	Conjunto contendo os nós correspondentes aos textos “T1” e “T2”

Tabela A.8 - Exemplos de Expressões XPath

A.6 XForms

XForms (*XML Forms*) é uma recomendação do W3C, definida em [34], para a nova geração de formulários na Internet. Esta recomendação, que se encontra atualmente na versão 1.0, de outubro de 2003, surgiu para substituir o suporte a formulários da linguagem HTML, que apresenta diversas limitações. O termo “XForms” é sempre usado no plural, referindo-se ao conjunto de elementos que compõem este padrão. O nome de espaço

associado ao vocabulário XForms é http://www.w3.org/2002/xforms.

A especificação XForms é formada por diversos módulos, cada qual contendo um conjunto de elementos XML. A composição de determinados módulos constitui um perfil da linguagem. Atualmente existe apenas um perfil recomendado, composto de todos os módulos definidos pela linguagem. É possível definir um perfil formado por um subconjunto de módulos, com o objetivo, por exemplo, de permitir o uso de XForms em dispositivos de menor capacidade (atualmente existe uma versão candidata para este perfil, denominada *XForms Basic*).

XForms é baseado na separação entre o modelo de dados e os controles utilizados da interface de usuário. Indo mais além, os controles de XForms expressam apenas sua intenção, e não a forma como devem ser apresentados para o usuário. O processamento de XForms é baseado em eventos, como especificado pela recomendação XML Events do W3C.

A.6.1 Expressões de Ligação

É comum o uso de expressões XPath em atributos dos elementos de XForms para referenciar dados de instâncias do modelo. Estas expressões XPath são denominadas expressões de ligação, e são normalmente utilizadas para realizar a associação entre os dados do modelo e os controles. Estas expressões são avaliadas dentro de um contexto, como descrito na especificação de XForms. O atributo `model` permite indicar qual o identificador do modelo que deve ser usado como contexto. O atributo `ref` contém uma expressão XPath, do tipo caminho de localização, que retorna um nó (*single-node binding*) ou conjunto de nós (*node-set binding*). A função `instance()` pode ser usada como parte da expressão de ligação para indicar qual instância dentro daquele modelo deve ser usada como contexto (por *default* é usada a primeira instância).

Como descrito em [1], XPath também é usado em XForms como uma linguagem de scripting leve, por meio de expressões computadas (*computed expressions*), que são as funções XPath, permitindo atribuir propriedades aos dados do modelo ou ainda realizar ações sobre os dados do modelo.

A seguir estão descritos os elementos que compõem cada um dos módulos de XForms.

A.6.2 Módulo “Core”

Este módulo define os principais elementos estruturais de XForms relacionados aos modelos de dados. Estes elementos não são apresentados diretamente para o usuário. O módulo *Core* é composto dos seguintes elementos:

- `model`
- `instance`
- `submission`
- `bind`

O elemento `model` contém instâncias de modelos de dados formados por elementos XML. Através do atributo `schema` é possível fazer referencia a um documento externo no formato XML Schema contendo a definição dos tipos de dados utilizados pelos elementos do modelo. Por exemplo:

```
<model schema=file://modelos/sistema/relogio>
...
</model>
```

As instâncias do modelo de dados são indicadas pelo elemento `instance`. Os elementos de cada instância podem ser obtidos de uma fonte externa, indicada pelo atributo `src`, ou declarados diretamente. Por exemplo:

```
<model>
  <instance>
    <relogio>
      <dia>2005-05-26</dia>
      <hora>21:00</hora>
    </relogio>
  </instance>
</model>
```

Ou no caso de uma fonte externa:

```
<model>
  <instance id=”relogio” src=”/sistema/relogio”/>
</model>
```

O elemento `bind` permite atribuir propriedades a um conjunto de nós do modelo. O conjunto de nós é definido pelo atributo `nodeset`, que contém uma expressão XPath do

tipo caminho de localização. As seguintes propriedades podem ser atribuídas aos nós do conjunto:

- `type` – define o tipo XML Shema dos dados, para o caso de um arquivo externo neste formato não ter sido utilizado.
- `readonly` – expressão booleana XPath que indica se os dados podem ser alterados ou são apenas de leitura.
- `required` – expressão booleana XPath que indica se os dados são obrigatórios para submeter o formulário.
- `relevant` – expressão booleana XPath que indica se os dados são relevantes para submeter o formulário.
- `calculate` – expressão XPath que permite calcular dinâmicamente o valor dos dados.
- `constraint` – expressão booleana XPath que permite validar os dados antes do formulário poder ser submetido.

Também é possível utilizar o atributo `id` para estabelecer referências ao conjunto de nós, permitindo que os controles XForms sejam "desacoplados" do modelo, ou seja, associados de forma indireta e independente da estrutura do modelo. Por exemplo:

```
<model>
  ...
  <bind nodeset="/relogio/data" id="relogio-data" />
  <bind nodeset="/relogio/hora" id="relogio-hora" />
</model>
```

Neste exemplo, os controles podem fazer referência aos identificadores `relogio-data` e `relogio-hora`, ao invés de usar caminhos de localização XPath que dependem da estrutura do modelo.

As instâncias são utilizadas para receber o valor inicial e enviar os resultados de um formulário. O elemento `submission` define, através dos atributos `bind` e `ref`, quais dados da instância devem ser submetidos. Os atributos `method` e `encoding` definem como os dados devem ser submetidos. O atributo `action` contém a URI que indica qual o destino dos dados. Por exemplo:

```
<model>
  ...
  <submission id="ajustar-relogio" method="post"
action=http://relogio.cgi />
```

```
</model>
```

É importante notar que as instâncias dos modelos de dados XPath podem ser utilizadas também para manter dados intermediários durante o tempo de vida do formulário, funcionando como variáveis.

A.6.3 Módulo "Action"

Ações são utilizadas em XForms para tratar eventos de forma totalmente declarativa, com uma semântica de alto nível de abstração, evitando o uso de *scripts*.

Este módulo define os elementos relacionados às ações que podem ser invocadas por eventos gerados pelo documento XForms. O módulo *Action* é composto dos seguintes elementos:

- `message`
- `send`
- `setvalue`
- `dispatch`
- `action`
- `rebuild`
- `recalculate`
- `revalidate`
- `refresh`
- `reset`
- `load`

O elemento `message` permite gerar uma mensagem de notificação para o usuário. O atributo `level` pode ser usado para especificar o tipo de mensagem (por exemplo, modal, transiente, etc). É importante notar que XForms não especifica como estes tipos devem ser apresentados para o usuário, apenas a intenção associada a cada um deles. Por exemplo:

```
<message level=”modal”>Relógio ajustado!</message>
```

Neste caso, para uma interface baseada em janelas, um diálogo modal seria apresentado.

O elemento `send` permite submeter uma instância do modelo. O atributo `submission` deve conter uma referência para o elemento `submission` correspondente

em um modelo do documento. Por exemplo:

```
<send submission="ajustar-relogio" />
```

O elemento `setvalue` permite modificar o valor de uma instância do modelo de dados. O atributo `bind` deve conter uma referência para o elemento do modelo que deve ser alterado. O atributo `value`, quando presente, contém uma expressão XPath cujo resultado será atribuído ao elemento. Por exemplo:

```
<setvalue bind="relogio-hora" value="var/hora" />
<setvalue bind="relogio-data">2005-06-05</setvalue>
```

O elemento `dispatch` permite enviar o evento indicado pelo atributo `name` ao elemento referenciado pelo atributo `target`. Por exemplo:

```
<dispatch name="evento" target="destino">
```

O elemento `setfocus` permite forçar que o foco seja transferido para o controle indicado pelo atributo `control`. Por exemplo:

```
<setfocus control="campo-2"/>
```

O elemento `action` é usado para agrupar diversas ações a serem executadas quando um determinado evento for gerado. Por exemplo:

```
<action ev:event="DOMActivate">
  <setvalue bind="relogio-hora">12:00</setvalue>
  <send submission="ajustar-relogio"/>
</action>
```

Em geral a atualização do estado das instâncias dos modelos de dados somente ocorre após o processamento da ação mais externa. Os elementos `rebuild`, `recalculate`, `revalidate` e `refresh` permitem forçar uma atualização da representação interna do modelo indicado pelo atributo `model`. O elemento `reset` permite reinicializar o modelo referenciado pelo atributo `model`.

Finalmente, o elemento `load` permite carregar um recurso externo indicado pelo

atributo `resource`. Através dos atributos `show`, `new` e `replace` é possível controlar o comportamento desta ação, por exemplo, se o formulário atual deve ser substituido ou não.

A.6.4 Módulo "Form Controls"

Este módulo define os principais elementos relacionados com a interface de usuário. XForms define uma série de controles para interação com o usuário. Os controles permitem definir a intenção, ou seja, aspectos funcionais, e não a forma de apresentação e comportamento, os quais dependem da modalidade da interface de usuário do equipamento.

Este módulo é composto dos seguintes elementos:

- `input`
- `secret`
- `textarea`
- `output`
- `range`
- `trigger`
- `submit`
- `label`
- `help`
- `hint`
- `select1`
- `select`
- `item`
- `value`
- `alert`

Alguns atributos são comuns a maioria dos tipos de controles. A ligação entre um controle e os dados do modelo é feita pelos atributos `bind`, `ref`, `nodeset` e `model`. O atributo `bind` contém o identificador do elemento `bind` correspondente em um modelo do documento. Os atributos `ref` e `nodeset` contêm uma expressão XPath, do tipo caminho de localização, que seleciona, respectivamente, um ou múltiplos nós da instância do modelo. O atributo `model` especifica o identificador de um modelo, para o caso de haver mais de um no documento XForms. Esta ligação permite ao controle ter acesso aos valores, propriedades e tipos dos dados de instância do modelo. É importante notar que um mesmo tipo de controle pode ser apresentado na interface de usuário de forma completamente diferente dependendo do tipo dos dados aos quais ele está ligado. Os atributos `navindex` e `accesskey` estão associados à navegação na interface de usuário entre os controles do

formulário. O atributo `appearance` (`full`, `compact`, `minimum`) permite fornecer uma sugestão para a apresentação do controle. Isso também pode ser feito por meio da linguagem CSS, usando o atributo `class` ou outros.

O elemento `input` define um controle de entrada de dados. Por exemplo:

```
<input ref="relogio-data">
  <label>Data</label>
</input>
```

O elemento `textarea` é usado para a entrada de texto livre com campos de múltiplas linhas. O elemento `secret` é usado para entrada de texto sem realimentação para o usuário, como no caso de campos de senha.

Os elementos `input`, `textarea` e `secret` são controles de entrada de dados que possuem em comum o atributo `inputmode`, que fornece uma sugestão para as características do dispositivo de entrada, como conjunto de caracteres permitidos, modo numérico, modo de símbolos, maiúsculas, minúsculas, etc.

O elemento `output` define um controle para apresentação de dados. No caso deste controle não estar ligado a uma instância do modelo, o atributo `value` pode ser utilizado para apresentar o resultado de uma expressão XPath. A seguir é mostrado um exemplo para cada caso:

```
<output bind="relogio-data"/>
<output value="date()"/>
```

O elemento `range` define um controle para seleção de um valor dentro de uma determinada faixa. Os atributos `start` e `end` definem os limites da faixa, e o atributo `step` define o tamanho dos passos. Por exemplo:

```
<range bind="volume" start="0" end="5" step="1">
  <label>Volume</label>
</range>
```

Os elementos `trigger` e `submit` definem controles que permitem iniciar uma ação ou submeter uma instância do modelo, respectivamente. No caso do elemento

submit, é necessário especificar o atributo submission com uma referência para o respectivo elemento submission do modelo. Se estes controles estiverem ligados a um nó da instância do modelo de dados, as propriedades daquele nó irão influenciar o controle. Por exemplo:

```
<trigger bind="acao-1">
  <label>Ajustar relógio</label>
  ...
</trigger>

<submit submission="ajustar-relogio">
  <label>Ajustar relógio</label>
</submit>
```

Neste exemplo, se a instancia do modelo corrspondente a ligação acao-1 tiver um valor false para a propriedade relevant, o controle correspondente ao elemento trigger estará indisponível para o usuário.

Os elementos label, help, hint e alert, ao serem incluídos dentro dos demais controles, permitem associar a eles descrições, textos de ajuda e mensagens de erro. Os elementos help e hint definem textos de ajuda completo e abreviado, respectivamente. O elemento alert define uma mensagem de erro. O exemplo a seguir ilustra o uso destes elementos:

```
<input ref="relogio-data">
  <label>Data</label>
  <hint>Data</hint>
  <help>Entre com a data</help>
  <alert>Data inválida</alert>
</input>
```

Os elementos select1 e select definem controles de seleção simples e múltipla, respectivamente. O elemento item define um item para seleção. O elemento value define o valor associado ao item, e o elemento label contém uma descrição do item. O valor correspondente ao item selecionado é atribuído à instância do modelo ligado ao elemento select ou select1, conforme indicado pelos atributos model, ref e bind. O elemento choices permite agrupar itens relacionados. Estes elementos são usados como indicado no exemplo a seguir:

```
<select1 bind=”relogio-despertar”>
  <label>Despertador</label>
  <choices>
    <label>Ligar/Desligar</label>
    <item>
      <label>Ligado</label>
      <value>1</value>
    </item>
    <item>
      <label>Desligado</label>
      <value>0</value>
    </item>
  </choices>
  ...
</select1>
```

Neste exemplo, a lista de seleção simples contém duas opções, `Ligado` e `Desligado`. Conforme uma destas opções é selecionada, a instância do modelo indicada pelo atributo `bind` irá ter seu valor alterado para “1” ou “0”, respectivamente.

A.6.5 Módulo “Group”

Este módulo contém um único elemento `group`, o qual permite agrupar controles relacionados entre si. Os atributos `bind` ou `ref` permitem ao mesmo tempo definir um contexto para as expressões XPath usadas nos controles dentro do grupo, como também aplicar propriedades a todos os controles dentro do grupo, através da instância do modelo referenciada. Por exemplo:

```
<group ref=”/sistema/relogio”>
  <label>Ajuste do relógio</label>
  <input ref=”data”>
    <label>Data</label>
  </input>
  <input ref=”hora”>
    <label>Hora</label>
  </input>
</group>
```

Neste exemplo, as expressões XPath utilizadas dentro do elemento `group` serão relativas ao caminho “/sistema/relogio”. Se o nó da instancia correspondente a este caminho tiver o valor `false` para a propriedade `relevant`, todos os controles dentro do grupo estarão indisponíveis. O mesmo se aplica a outras propriedades, como `readonly`.

A.6.6 Módulo “Switch”

Os elementos deste módulo são usados para determinar quais controles estão ativados ou podem ser visualizados, no caso de um equipamento com modalidade gráfica, num certo momento. Os elementos deste módulo são:

- `switch`
- `case`
- `toggle`

O elemento `switch` é composto de uma série de elementos `case`. Cada elemento `case` contém um conjunto de controles que devem ser apresentados (não necessariamente no sentido visual da palavra) simultaneamente. O atributo `selected` do elemento `case` contém um valor booleano que indica se ele esta inicialmente ativo. Apenas um dos elementos `case` pode estar ativo num determinado instante. O elemento `toggle` é uma ação que permite ativar um determinado elemento `case`. O exemplo a seguir ilustra o uso destes elementos:

```
<switch>
  <case id=”tela-data” selected=”true”>
    <input bind="relogio-data">
      <label>Data</label>
      <toggle ev:event="DOMActivate" case="tela-hora"/>
    </input>
  </case>
  <case id="tela-hora" selected="false">
    <input bind="relogio-hora">
      <label>Hora</label>
      <send ev:event="DOMActivate" submission=”ajustar-
relogio” />
    </input>
  </case>
</switch>
```

Neste exemplo, supondo uma interface gráfica onde cada `case` corresponde a uma tela, uma tela contendo o campo de entrada de data do relógio seria apresentada inicialmente. Ao entrar com a data, uma tela com o campo de entrada de hora do relógio seria apresentada, em conseqüência do elemento `toggle`. Ao entrar com a hora, os dados seriam submetidos para processamento, como indicado pelo elemento `send`.

A.6.7 Módulo “Repeat”

Este módulo especifica diversos elementos relacionados com repetições de dados e controles em XForms. Os elementos deste módulo são:

- `itemset`
- `copy`
- `insert`
- `delete`

O elemento `itemset` pode ser utilizado dentro dos elementos `select1` e `select` para criar os itens da lista de forma dinâmica. Os atributos `nodeset` e `model`, ou alternativamente o atributo `bind`, devem especificar um conjunto homogêneo de nós da instância do modelo de dados. O elemento `itemset` deve conter os elementos `label` e `value`. Estes elementos referenciam os dados correspondentes a cada elemento do conjunto homogêneo, sendo que o atributo `ref` normalmete contém um caminho de localização relativo. Por exemplo:

```
<select1 model=”operacao” ref=”arquivo”>
  <label>Selecione uma imagem</label>
  <itemset model=”arquivos” nodeset=”/diretorio/entrada”>
    <label ref=”nome” />
    <value ref=”nome” />
  </itemset>
</select>
```

Neste exemplo, o modelo corresponde ao exemplo de XML Schema visto ao final da sessão A.4 . A expressão XPath no atributo `nodeset` seleciona um conjunto homogêneo de elementos `entrada`. Para cada um destes elementos, o atributo `ref` obtem o rótulo e valor de cada item a partir do conteúdo do elemento `nome` da instância do modelo.

O elemento `copy` pode ser utilizado no lugar do elemento `value` para indicar que toda a subárvore indicada pela expressão XPath contida no atributo `ref` deve ser copiada, ao invés de um simples valor.

As ações `insert` e `delete` permitem, respectivamente, inserir e remover elementos de uma instância do modelo de dados. Em ambos os casos, o atributo `nodeset` indica por meio de uma expressão XPath, do tipo caminho de localização, um conjunto

homogêneo de nós do modelo indicado pelo atributo `model`. Já o atributo `at` contém uma expressão XPath que retorna o índice do nó em que a operação deve ser realizada. No caso especifico do elemento `insert`, o atributo `position` indica se a inserção deve ocorrer antes (`before`) ou depois (`after`) deste nó. A operação cria um novo conjunto de nós do mesmo tipo que os demais do conjunto homogêneo selecionado. Por exemplo:

```
<insert nodeset="opcoes" at="last()" position="after" />
```

Este módulo também contém os elementos `repeat` e `setindex`, porém eles estão além do escopo deste trabalho, pois são mais úteis quando o documento contém outras linguagens de apresentação, como XHTML.

A.6.8 Funções XPath

XForms define as seguintes funções em adição à biblioteca XPath:

Retorno	**Função**	**Descrição**
Booleano	boolean-from-string(string)	Converte a string "1" ou "true" no valor booleano verdadeiro.
	if(boolean, string, string)	Avalia a condição booleana e retorna o a primeira string caso ela seja verdadeira ou a segunda string caso ela seja falsa.
Numérico	avg(node-set)	Retorna a media aritmética dos valores de cada nó do conjunto.
	min(node-set)	Retorna o menor valor dentre os nós do conjunto.
	max(node-set)	Retorna o maior valor dentre os nós do conjunto.
	count-non-empty(node-set)	Retorna o número de nós não vazios (ou seja, que tem tamanho maior que zero quando convertido para string).
	index(string)	Índice corrente do elemento "repeat" que possui o identificador especificado.
	days-from-date(string)	Retorna o número de dias entre a data especificada no formato xsd:dateTime e o dia 01/01/1970.
	seconds-from-dateTime(string)	Retorna o número de segundos entre a data especificada no formato xsd:dateTime e as zero horas do dia 01/01/1970.
	seconds(string)	Retorna o número de segundos correspondente a duração especificada no formato xsd:duration.

	months(string)	Retorna o número de segundos correspondente a duração especificada no formato xsd:duration.
String	now()	Retorna o horário corrente no formato xsd:dateTime.
	property(string)	Retorna o valor da propriedade especificada (“version” ou “conformance-level”).
Conjunto de nós	instance()	Retorna a instância do modelo de dados correspondente ao identificador especificado.

Tabela A.9 - Funções XPath definidas por XForms

É importante notar que as expressões XPath utilizadas na linguagem XForms são avaliadas em função de um nó e um modelo de contexto, como descrito em [34] (“Evaluation Context”).

A.7 CSS

CSS (*Cascading Style Sheets*) é uma recomendação W3C, definida em [36], para uma linguagem que permite definir propriedades de apresentação para classes de elementos de um documento XML. CSS permite uma maior separação entre conteúdo e estilo de interfaces de usuário. Isto permite representar documentos por meio de diferentes modalidades de interface, como visual e auditiva, e até mesmo para formatar dados a serem impressos. É importante notar que CSS não é uma linguagem baseada em XML.

As instruções da linguagem CSS são geralmente declaradas em separado do documento XML ao qual estão associadas, em arquivos conhecidos como “folhas de estilo”. A forma padrão de associar um arquivo CSS, chamado, por exemplo, “style.css”, a um documento XML, é incluindo neste último a seguinte instrução de processamento:

```
<?xml-stylesheet type="text/css" href="file://style.css" ?>
```

A sintaxe utilizada na linguagem CSS é composta por regras. Cada regra é formada por um ou mais “seletores”, separados por virgulas, seguidos de uma declaração delimitada pelos caracteres ‘{‘ e ‘}’. A declaração é formada por pares nome e propriedade, separados pelo caractere ‘:’. Os pares são separados entre si por caracteres ‘;’. Os seletores indicam a quais elementos do documento XML devem ser aplicados os valores de propriedades indicados na declaração. Por exemplo:

```
select, select1
{
  font-size: 12pt;
  color: blue;
  background-image: url(bg.jpg)
}
```

No exemplo acima, todos os elementos `select` e `select1` do documento XML associado terão o valor da propriedade `font-size` alterado para `12pt` e o da propriedade `background-image` alterado para "bg.jpg". É importante notar que as propriedades definidas por CSS são voltadas para a visualização de documentos HTML e XML em navegadores da Internet, porém é possível para determinadas implementações suportarem propriedades próprias e com outros objetivos, como por exempo a formatação de documentos para impressão.

O seletor '*' permite selecionar todos os elementos do documento. Um nome de um elemento seleciona apenas aquele elemento. É possivel restringir os elementos selecionados pelos seletores. Uma das formas de fazer isso é especificando o valor dos atributos `class` e `id` dos elementos no documento XML logo em seguida aos seletores, através dos caracteres '.' e '#', respectivamente. Por exemplo:

```
.principal, select#001 { font-size: 8pt }
```

Neste exemplo, a propriedade `font-size` será alterada para `8pt` em todos os elementos que tiverem um atributo `class` com o valor "principal" e também para o elemento `select` cujo atributo `id` tiver o valor "001". Da mesma forma, é possível utilizar também quaisquer atributos de um elemento, como indicado no exemplo abaixo:

```
message[level="modal"] { font-size: 10pt }
```

Neste exemplo, os elementos `message` que possuirem o atributo `level` com valor "modal" terão o valor da propriedade `font-size` modificada para `10pt`.

Uma outra forma de restringir os elementos selecionados é utilizando seletores contextuais, que são formados por uma combinação de dois ou mais seletores. Neste caso os seletores são processados da esquerda para a direita como um padrão de busca que reduz cada vez mais o escopo dos elementos selecionados. Apenas os elementos selecionados

pelo seletor mais à direita terão suas propriedades alteradas. Por exemplo:

```
group select { color: green }
```

Neste caso foi utilizado o caractere "espaço" para combinar os seletores, indicando que os elementos `select` que forem descendentes de um elemento `group` terão o valor da propriedade `color` alterada para `green`. Também é possível utilizar os caracteres '>' e '+' para combinar os seletores, indicando respectivamente que o elemento a direita deve ser "pai" do elemento à esquerda ou precede-lo imediatamente. É importante notar que, no caso de mais de uma regra selecionar o mesmo elemento, prevalecerá apenas uma, de acordo com o especificado pela linguagem CSS em [36] (*Cascading Order*).

Uma característica importante de CSS é o suporte a pseudoclasses e pseudoelementos, que permitem associar informação aos elementos além do que existe no documento XML, ou seja, como se fossem elementos e atributos do tipo `class` fictícios, gerados automaticamente pelo processador CSS. Os valores de pseudoclasses e pseudoelementos são especificados nos seletores precedidos do caractere ':' e da seqüência de caracteres "::", respectivamente, como indicado no exemplo a seguir:

```
input:invalid { color: red }
label::first-letter { font-size: 12pt }
```

No exemplo acima, os elementos do tipo `input` que estiverem em uma condição inválida (indicado pela pseudoclasse `invalid`) terão a propriedade `color` alterada para `red`. Já no caso dos elementos `label`, o primeiro caractere (indicado pelo pseudoelemento `first-letter`) terá a propriedade `font-size` modificada para `12pt`.

A linguagem CSS possui um conceito de herança. Ao atribuir um valor de propriedade para um determinado elemento do documento XML, os demais elementos que estiverem contidos nele (descendentes) também utilizarão o mesmo valor para aquela propriedade.

A versão 3 de CSS, ainda não aprovada pelo W3C, introduz diversas pseudoclasses especialmente voltadas para elementos usados em interfaces de usuário, como no caso de XForms. A Tabela a seguir indica as pseudoclasses propostas, como definido em [37]:

Pseudoclasse	Descrição
:hover	Cursor sobre o elemento (se aplicável à modalidade da interface).
:active	O elemento esta sendo ativado.
:focus	O elemento esta em foco.
:enabled	O elemento está habilitado.
:disabled	O elemento está desabilitado.
:checked	O elemento esta selecionado
:indeterminate	O elemento não está selecionado.
:valid	O elemento contém um valor válido em relação ao tipo de dado ao qual está associado.
:invalid	O elemento contém um valor inválido em relação ao tipo de dado ao qual está associado.
:in-range	O elemento contém um valor dentro do range permitido.
:out-of-range	O elemento contém um valor fora do range permitido.
:required	O elemento contém um valor obrigatório.
:optional	O elemento contém um valor opicional.
:read-only	O elemento contém um valor apenas de leitura.
:read-write	O elemento contém um valor que pode ser alterado.
:contains(string)	O elemento ou seus descendentes contém a cadeia de caracteres passada como parâmetro.

Tabela A.10 - Pseudoclasses de CSS

Também fazem parte de CSS3 os pseudoelementos `::selection` e `::value`, que correspondem às regiões selecionadas e ao valor do elemento, respectivamente.

Uma propriedade importante introduzido por CSS3 é `appearance`, para melhor suporte a linguagens que tentam abstrair a representação gráfica dos controles da interface de usuário, como XForms. Esta propriedade permite escolher uma dentre diversas apresentações para um mesmo controle.

Apêndice B

Exemplo de uma Aplicação em XForms

Para melhor ilustrar o uso de XForms na definição de interfaces de usuário de dispositivos portáteis como apresentado neste trabalho, é descrito a seguir um exemplo de uma aplicação, aqui denominada "Notas", para gerenciamento de notas de texto armazenadas no dispositivo. As seguintes funcionalidades são providas pela aplicação:

- Visualizar o conjunto de notas existentes.
- Visualizar e editar o texto de uma determinada nota.
- Apagar uma determinada nota.
- Criar uma nova nota.

A aplicação é formada pelo conjunto de tarefas correspondentes a estas funcionalidades. Os documentos XForms definem os modelos de dados, operações e conteúdo das telas que compõem cada tarefa.

B.1 Tarefas da Aplicação

As tarefas da aplicação e as respectivas telas estão descritas a seguir.

B.1.1 Tarefa de Visualizar Conjunto de Notas

A tarefa principal da aplicação permite visualizar as notas armazenadas no dispositivo através de uma lista, conforme representado na figura a seguir:

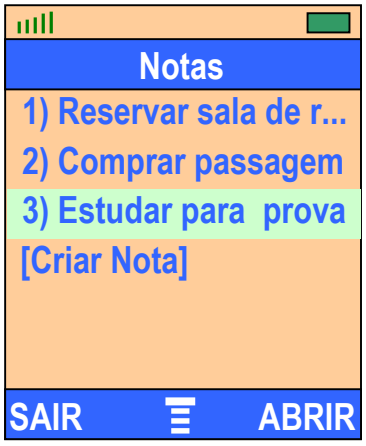

Figura B.1 - Visualizar e editar notas

Esta tela é mostrada assim que a aplicação é iniciada. A lista permite visualizar o índice e o início do texto de cada uma das notas existentes. O item em destaque permite executar operações sobre a respectiva nota. A ação “Sair” termina a aplicação. A ação “Abrir” dá início à tarefa de visualização do texto completo da nota em destaque, permitindo também alterar o conteúdo da mesma. A tela do respectivo menu de opções está representada na figura a seguir:

Figura B.2 - Menu de opções

Nesta tela, a operação “Apagar” remove a nota em destaque e a operação “Criar Nota” inicia a tarefa de criação de uma nova nota.

B.1.2 Tarefa de Visualizar e Editar uma Nota

A tarefa de visualização e edição de notas é composta pelo editor de texto representado na figura a seguir:

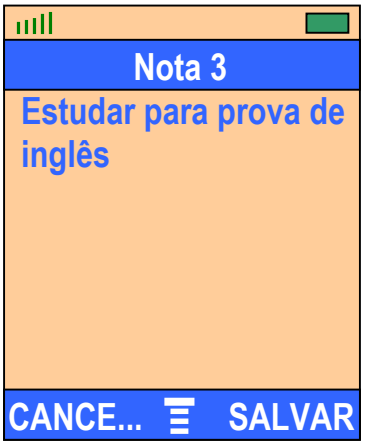

Figura B.3 - Tela de visualização e edição de nota

Nesta tela, a ação “Salvar” permite gravar o conteúdo da nota modificada. A ação “Cancelar” encerra a tarefa e retorna a lista de notas.

B.1.3 Tarefa de Criar uma Nota

A tarefa de criação de uma nota é composta pelo editor de texto representado na figura a seguir:

Figura B.4 - Tela de criação de nota

Esta tela é semelhante a de visualização e edição de notas, porém é sempre inicialmente vazia. A ação “Salvar” permite gravar o conteúdo da nova nota. A ação “Cancelar” encerra a tarefa e retorna a lista de notas.

B.1.4 Tarefa de Remover uma Nota

A tarefa de remoção de uma determinada nota é composta apenas por uma notificação, conforme representado na figura a seguir:

Figura B.5 - Notificação de remoção de nota

B.2 Documentos da Aplicação

A seguir estão descritos exemplos dos documentos XForms correspondentes ao modelo de dados e às tarefas da aplicação.

B.2.1 Modelo do Repositorio de Notas

O exemplo a seguir define o documento XML Schema para o repositório de notas no dispositivo. Cada nota posui como atributos um índice, a data e hora de criação ou última atualização, além do conteúdo completo do texto:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- XML Schema para repositorio de notas -->
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="notas">
    <complexType>
      <element name="nota">
        <complexType>
          <sequence>
            <element name="indice" type="positiveInteger"/>
            <element name="dataHora" type="dateTime"/>
            <element name="texto" type="string"/>
          </sequence>
        </complexType>
      </element>
    </complexType>
  </element>
</schema>
```

B.2.2 Tarefa de Visualizar Conjunto de Notas e Remover Nota

O exemplo a seguir define o documento XForms responsável pela tarefa de visualização do conjunto de notas armazenadas no dispositivo. A tarefa de remoção de uma nota não possui uma documento próprio, sendo tratada por este mesmo documento:

```
<!-- Tarefa principal para bloco de anotacoes -->
<?xml version="1.0" encoding="UTF-8" ?>
<html xmlns:xf="http://www.w3.org/2002/xf"
      xmlns:ev="http://www.w3.org/2001/xml-events"
      xmlns="http://www.w3.org/2002/06/xhtml2">

  <!-- Modelo ccorrespondente ao repositorio de notas -->
  <xf:model id="rep" schema="notas.xsd">
    <xf:instance src="model://notas"/>
    <xf:bind id="notas" nodeset="/notas"/>
  </xf:model>
```

```
<!-- Modelo contendo variaveis temporarias -->
<xf:model>
  <xf:instance id="temp">
    <vars xmlns="">
      <indice/>
      <criar/>
      <abrir/>
    </vars>
  </xf:instance>
  <!-- Indice da nota selecionada -->
  <xf:bind id="nota-sel" nodeset="/vars/indice"/>
  <!-- Acao "Abrir" disponivel se indice eh uma nota -->
  <xf:bind id="acao-abrir" nodeset="/vars/abrir"
           relevant="instance('temp')/vars/indice != 0"/>
  <!-- Acao "Criar" disponivel se indice igual a zero -->
  <xf:bind id="acao-criar" nodeset="/vars/criar"
           relevant="instance('temp')/vars/indice = 0"/>
</xf:model>

<!-- Modelo para parametros da tarefa visualizar nota -->
<xf:model>
  <xf:instance>
    <params xmlns="">
      <indice/>
    </params>
  </xf:instance>
  <!-- Indice da nota -->
  <xf:bind id="abrir-indice" nodeset="/params/indice"/>
  <!-- Chamada da tarefa visualizar/editar nota -->
  <xf:submission id="abrir-nota"
                 ref="/params"
                 action="task://abrir-nota"
                 method="ext:call" replace="none"/>
  </xforms:submission>
</xf:model>

<!-- Modelo para executar procedimento de apagar nota -->
<xf:model>
  <xf:instance>
    <params xmlns="">
      <indice/>
    </params>
  </xf:instance>
  <xf:bind id="apagar-indice" nodeset="/params/indice"/>
  <!-- Chamada do procedimento para apagar uma nota -->
  <xf:submission id="apagar-nota"
```

```
                ref="/params"
                action="proc://apagar-nota"
                method="ext:call" replace="none">
  <!-- Notificacao com tecla "OK" indicando que
       a nota foi apagada -->
  <xf:message level="modal-ok" ev:event="ext:note-deleted">
    <!-- "Nota <n> apagada!" -->
    <xf:output value="i18n('formato_nota_apagada',
                           /params/indice)"/>
  </xf:message>
</xf:model>

<!-- Modelo para iniciar tarefa de criar nota -->
<xf:model>
  <xf:instance>
    <params/>
  </xf:instance>
  <!-- Chamada da tarefa para criar uma nota -->
  <xf:submission id="criar-nota"
                ref="/params"
                action="task://criar-nota"
                method="ext:call" replace="none">
</xf:model>

<!-- Modelo para finalizar tarefa -->
<xf:model>
  <xf:instance/>
  <!-- Chamada para terminar tarefa -->
  <xf:submission id="terminar-tarefa"
                action="task://return"
                method="ext:call">
</xf:model>

<!-- Definicoes de telas -->
<xf:switch>
  <!-- Tela de sumario de notas -->
  <xf:case id="tela-notas">
    <!-- Titulo da tela “Notas” -->
    <xf:label>
      <xf:output value="i18n('notas')"/>
    </xf:label>

    <!-- Lista de selecao simples -->
    <xf:select1 bind="nota-selecionada"
                appearance="ext:simple-list">
      <xf:label/>
      <!-- Entradas da lista contendo o texto
```

```
      de cada nota -->
  <xf:itemset bind="notas">
    <xf:label>
      <xf:output value="i18n('formato_sumario_notas',
                             indice, texto)"/>
    </xf:label>
    <xf:value ref="indice"/>
    <!-- Tecla central de selecao -->
    <xf:action ev:event="DOMActivate">
      <!-- Inicializa parametros -->
      <xf:setvalue bind="abrir-indice"
                   ref="instance('temp')/vars/indice"/>
      <!-- Inicia tarefa de visualizacao de nota -->
      <xf:send submission="abrir-nota"/>
    </xf:action>
  </xf:itemset>
  <!-- Ultima entrada da lista "[Criar Nota]" -->
  <xf:item>
    <xf:label>
      <xf:output value="i18n('ac_criar_nota_fc')"/>
    </xf:label>
    <!-- Indice correspondente a esta acao -->
    <xf:value>0</value>
    <!-- Tecla central de selecao -->
    <xf:send ev:event="DOMActivate"
           submission="criar-nota"/>
  </xf:item>
</xf:select1>

<!-- Tecla de acao "Sair" -->
<xf:trigger appearance="ext:left-softkey">
  <xf:label>
    <xf:output value="i18n('sair')"/>
  </xf:label>
  <!-- Finaliza tarefa -->
  <xf:send ev:event="DOMActivate"
           submission="terminar-tarefa"/>
</xf:trigger>

<!-- Tecla de acao "Selecionar" -->
<xf:trigger bind="acao-criar"
            appearance="ext:right-softkey">
  <xf:label>
    <xf:output value="i18n('selecionar')"/>
  </xf:label>
  <!-- Inicia tarefa de criacao de nota -->
  <xf:send ev:event="DOMActivate"
```

```
            submission="criar-nota"/>
      </xf:trigger>

      <!-- Tecla de acao "Abrir" -->
      <xf:trigger bind="acao-abrir"
                  appearance="ext:right-softkey">
        <xf:label>
          <xf:output value="i18n('abrir')"/>
        </xf:label>
        <xf:action ev:event="DOMActivate">
          <!-- Inicializa parametros -->
          <xf:setvalue bind="abrir-indice"
                       ref="instance('temp')/vars/indice"/>
          <!-- Inicia tarefa de visualizacao de nota -->
          <xf:send submission="abrir-nota"/>
        </xf:action>
      </xf:trigger>

      <!-- Menu de opcoes -->
      <xf:group appearance="ext:menu">

        <!-- Opcao "Apagar" -->
        <xf:trigger>
          <xf:label>
            <xf:output value="i18n('apagar')"/>
          </xf:label>
          <xf:action ev:event="DOMActivate">
            <xf:setvalue bind="apagar-indice"
                         value="instance('temp')
                                /vars/indice"/>
            <xf:send submission="apagar-nota"/>
          </xf:action>
        </xf:trigger>

        <!-- Opcao "Criar Nota" -->
        <xf:trigger>
          <xf:label>
            <xf:output value="i18n('criar_nota')"/>
          </xf:label>
          <xf:send ev:event="DOMActivate"
                   submission="criar-nota"/>
        </xf:trigger>
      </xf:group>
    </xf:case>
  </xf:switch>
</html>
```

B.2.3 Tarefa de Visualizar e Editar Nota

O exemplo a seguir define o documento XForms responsável pela tarefa de visualização e edição de uma determinada nota armazenadas no dispositivo:

```
<!-- Tarefa para visualizar/editar nota -->
<?xml version="1.0" encoding="UTF-8" ?>
<html xmlns:xf="http://www.w3.org/2002/xf"
      xmlns:ev="http://www.w3.org/2001/xml-events"
      xmlns="http://www.w3.org/2002/06/xhtml2">

  <!-- Modelo para parametros de entrada da tarefa -->
  <xf:model>
    <xf:instance id="params-entrada" src="task://input"/>
    <xf:bind id="nota-indice" nodeset="/params/indice"/>
  </xf:model>


  <!-- Modelo ccorrespondente ao repositorio de notas -->
  <xf:model id="rep" schema="notas.xsd">
    <xf:instance src="model://notas"/>
    <xf:bind id="nota-texto" nodeset="/notas/
        nota[instance('params-entrada')/params/indice]/
        texto"/>
  </xf:model>

  <!-- Modelo para parametros da operacao salvar nota -->
  <xf:model>
    <xf:instance id="input">
      <dados xmlns="">
        <params>
          <indice/>
          <texto/>
          <dataHora/>
        </params>
      </dados>
    </xf:instance>
    <xf:bind id="salvar-indice" nodeset="/params/indice"/>
    <xf:bind id="salvar-texto" nodeset="/params/texto"/>
    <xf:bind id="salvar-dataHora"
             nodeset="/params/dataHora"/>
    <!-- Chamada do procedimento salvar nota -->
    <xf:submission id="salvar-nota" method="ext:call"
                   action="proc://salvar-nota">
      <xf:send ev:event="xf-submit-done"
               submission="terminar-tarefa"/>
    </xf:submission>
```

```
</xf:model>

<!-- Modelo para finalizar tarefa -->
<xf:model>
  <xf:instance/>
  <!-- Chamada para finalizar tarefa -->
  <xf:submission id="terminar-tarefa" method="ext:call"
                 action="task://return">
</xf:model>

<!-- Definicoes de telas -->
<xf:switch>
  <!-- Tela de edicao de uma nota -->
  <xf:case id="tela-editar-nota">
    <!-- Titulo da tela "Nota <n>" -->
    <xf:label>
      <xf:output value="i18n('formato_titulo_abrir_nota',
                      instance('params-entrada')
                      /params/indice )"/>
    </xf:label>

    <!-- Editor de texto -->
    <xf:input bind="nota-texto"
              appearance="ext:editor-texto"/>

    <!-- Tecla de acao "Cancelar" -->
    <xf:trigger appearance="ext:left-softkey">
      <xf:label>
        <xf:output value="i18n('cancelar')"/>
      </xf:label>
      <xf:send ev:event="DOMActivate"
               submission="terminar-tarefa"/>
    </xf:trigger>

    <!-- Tecla de acao "Salvar" -->
    <xf:trigger appearance="ext:right-softkey">
      <xf:label>
        <xf:output value="i18n('salvar')"/>
      </xf:label>
      <xf:action ev:event="DOMActivate">
        <!-- Inicializa parametros -->
        <xf:setvalue bind="salvar-indice"
                     value="instance('params-entrada')
                            /params/indice"/>
        <xf:setvalue bind="salvar-texto"
                     value="instance('rep')/notas/
            nota[instance('params-entrada')/params/indice]/
```

```
            texto"/>
          <xf:setvalue bind="salvar-dataHora"
                       value="now()"/>
          <!—Inicia operacao salvar nota -->
          <xf:send submission="salvar-nota"/>
        </xf:action>
      </xf:trigger>

    </xf:case>
  </xf:switch>
</html>
```

B.2.4 Tarefa de Criar uma Nota

O exemplo a seguir define o documento XForms responsável pela tarefa de criação de uma nova nota:

```
<!-- Tarefa para criar nota -->
<?xml version="1.0" encoding="UTF-8" ?>
<html xmlns:xf="http://www.w3.org/2002/xf"
      xmlns:ev="http://www.w3.org/2001/xml-events"
      xmlns="http://www.w3.org/2002/06/xhtml2">

  <!-- Modelo para parametros de entrada da tarefa -->
  <xf:model>
    <xf:instance id="params-entrada" src="task://input"/>
  </xf:model>

  <!-- Modelo para parametros da operacao criar nota -->
  <xf:model>
    <xf:instance>
      <params xmlns="">
        <texto/>
        <dataHora/>
      </params>
    </xf:instance>
    <xf:bind id="criar-texto" nodeset="/params/texto"/>
    <xf:bind id="criar-dataHora" nodeset="/params/dataHora"/>
    <!-- Chamada da operacao criar nota -->
    <xf:submission id="criar-nota"
                   action="proc://salvar-nota"
                   method="ext:call"/>
      <xf:send ev:event="xf-submit-done"
               submission="terminar-tarefa"/>
    </xf:submission>
  </xf:model>
```

```
<!-- Modelo para operacao finalizar tarefa -->
<xf:model>
  <xf:instance/>
  <!-- Chamada de operacao terminar tarefa -->
  <xf:submission id="terminar-tarefa" method="ext:call"
                 action="task://return">
</xf:model>

<!-- Definicoes de telas -->
<xf:switch>
  <!-- Tela de edicao de uma nota -->
  <xf:case id="tela-criar-nota">
    <!-- Titulo da tela "Criar Nota" -->
    <xf:label>
      <xf:output value="i18n('titulo-criar-nota')"/>
    </xf:label>

    <!-- Editor de texto -->
    <xf:input bind="texto" appearance="ext:editor-texto"/>

    <!-- Tecla de acao "Cancelar" -->
    <xf:trigger appearance="ext:left-softkey">
      <xf:label>
        <xf:output value="i18n('cancelar')"/>
      </xf:label>
      <xf:send ev:event="DOMActivate"
               submission="terminar-tarefa"/>
    </xf:trigger>

    <!-- Tecla de acao "Salvar" -->
    <xf:trigger appearance="ext:right-softkey">
      <xf:label>
        <xf:output value="i18n('salvar')"/>
      </xf:label>
      <xf:action ev:event="DOMActivate">
        <!-- Inicializa parametros -->
        <xf:setvalue bind="criar-texto"
                     value="instance('params-entrada')
                            /params/texto"/>
        <xf:setvalue bind="criar-dataHora" value="now()"/>
        <!-- Inicia operacao criar nota -->
        <xf:send submission="criar-nota"/>
      </xf:action>
    </xf:trigger>

  </xf:case>
```

```
  </xf:switch>
</html>
```