UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

FRANCISCO MAESTRI TRINDADE

RenderXML – Renderizador de Interfaces de Usuário para Múltiplas Plataformas

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Marcelo Soares Pimenta
Orientador

Porto Alegre, janeiro de 2008.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Trindade, Francisco Maestri

RenderXML – Renderizador de Interfaces de Usuário para Múltiplas Plataformas / Francisco Maestri Trindade – Porto Alegre: Programa de Pós-Graduação em Computação, 2008.

71 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2008. Orientador: Marcelo Soares Pimenta.

1. Interfaces de usuário. 2. Renderização 3. UsiXML. I. Pimenta, Marcelo Soares. II. Título.

Agradecimento**S**

Gostaria de agradecer a minha família, pais e irmãos, pelo apoio dado durante toda a minha formação acadêmica e pessoal, e também durante os últimos dois anos de realização deste trabalho.

Agradeço também em especial à minha namorada, Cristina, que sempre me entendeu e ajudou durante esse período.

Ao meu orientador, Prof. Marcelo Soares Pimenta, pelo acompanhamento durante a elaboração desta dissertação, e pelas dicas, profissionais e pessoais recebidas nas reuniões realizadas.

Ao Prof. Jean Vanderdonckt, pelo apoio e auxílio nas dúvidas que surgiram, e também pelas indicações sobre caminhos a serem seguidos.

Por fim agradeço à Universidade Federal do Rio Grande do Sul, e em especial ao Instituto de Informática, seus funcionários e professores, por fornecer a melhor estrutura possível para dar apoio aos seus alunos.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AUI	*Abstract User Interface*
AUIML	*Abstract User Interface Markup Language*
CDC	*Connected Device Configuration*
CIO	*Concrete Interaction Objects*
CTT	*Concur-Task-Trees*
CUI	*Concrete User Interface*
DHTML	*Dynamic HTML*
FUI	*Final User Interface*
GoF	*Gang of Four*
HTML	*HyperText Markup Language*
IHC	Interação Homem-Computador
INESC	Instituto Nacional de Engenharia de Sistemas de Computação
IP	Interfaces Plásticas
IU	Interface de Usuário
JME	*Java Micro Edition*
JSE	*Java Standard Edition*
JSF	*Java Server Faces*
LDIU	Linguagem de Descrição de Interface de Usuário
LIFAPOR	Livros Falados em Português
MONA	*Mobile multimOdal Next-generation Applications*
OLPC	*One Laptop Per Child*
PDA	*Personal Digital Assistant*
PROCEMPA	Companhia de Processamento de Dados do Município de Porto Alegre
RIML	*Renderer-Independent Markup Language*
SMIC	Secretaria Municipal da Industria e do Comércio
SVG	*Scalable Vector Graphics*
TERESA	*Transformation Environment for Interactive Systems Representations*

TIDE	*Tranformation-based Integrated Development Environment*
UIML	*User Interface Markup Language*
UsiXML	*User Interface eXtensible Markup Language*
W3C	*World Wide Web Consortium*
WAP	*Wireless Application Protocol*
WML	*Wireless Markup Language*
XAML	*eXtensible Application Markup Language*
XHTML	*eXtensible Hypertext Markup Language*
XIML	*eXtensible Interface Markup Language*
XML	*eXtensible Markup Language*
XUL	*XML User Interface Language*
XVT	*Extended Virtual Toolkit*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O surgimento de diferentes dispositivos computacionais fez crescer a demanda pela possibilidade de utilização de um aplicativo em múltiplas plataformas, exigindo o desenvolvimento de novas técnicas que possibilitem o atendimento desse requisito de forma mais simples.

Uma das abordagens propostas para solucionar esse problema é caracterizada pelo uso de interfaces plásticas, que se adaptam ao contexto de uso, e permitem a utilização de uma mesma descrição de interface para diferentes contextos. A adoção de técnicas de desenvolvimento baseado em modelos e de linguagens de descrição de interfaces (como a UsiXML, *USer Interface eXtensible Markup Language*) permitem a descrição de interfaces de usuário em diferentes níveis de abstração, mas para o desenvolvimento de interfaces plásticas multiplataforma utilizando UsiXML, são necessárias ferramentas que permitam o mapeamento entre cada um desses níveis.

Nesse trabalho é apresentado o RenderXML, um renderizador de interfaces de usuário que atua no último nível de abstração da linguagem UsiXML, mapeando descrições de interfaces concretas para interfaces de usuário finais sendo executadas em um dispositivo específico. Além disso, o RenderXML fornece um segundo grau de liberdade ao desenvolvedor, permitindo a conexão da interface de usuário renderizada com lógicas de aplicação desenvolvidas em múltiplas linguagens de programação. Para ilustrar sua aplicação, RenderXML foi usado e testado no desenvolvimento de *software* multiplataforma, em particular para a criação de livros eletrônicos falados, tema do projeto LIFAPOR, no qual este trabalho está inserido.

Palavras-Chave: UsiXML, Interfaces Plásticas, IHC, Plasticidade.

RenderXML – Multiplatform User Interface Renderer

ABSTRACT

The existent of different computing devices has created the necessity of software development for multiple platforms, requiring new techniques that permit the fulfillment of this requirement in a simple way.

One of the proposed approaches to solve this problem is characterized by the use of plastic user interfaces, which adapt themselves to an use context, and allow the utilization of the same user interface description in multiple contexts. The adoption of model-based development techniques and user interface description languages (as UsiXML, *USer Interface eXtensible Markup Language*) enables the specification of user interfaces at different levels of abstraction, but requires tools which perform the mapping between each one of these levels.

This work presents RenderXML, a user interface rendering application which acts on the last abstraction level of UsiXML, mapping concrete user interfaces descriptions to final user interfaces running on a specific device. Moreover, RenderXML provides a second degree of freedom to the developer, allowing the connection of the rendered user interface to functional cores developed in multiple programming languages. To illustrate its application, RenderXML was used and tested in the development of multiplatform software, in particular in the creation of digital talking books, theme of the LIFAPOR project, in which this work is inserted.

Keywords: UsiXML, Plastic User Interface, HCI, Plasticity.

1 INTRODUÇÃO

O advento de diferentes dispositivos computacionais, como computadores portáteis e telefones celulares, fez surgir a necessidade de aplicações que possam ser executadas em múltiplas plataformas, proporcionando ao usuário final a possibilidade de acesso a recursos de computação em diversos ambientes. Essa situação estabelece novos desafios para a comunidade de IHC (CALVARY, 2001), como o desenvolvimento e manutenção de versões de aplicações para diferentes dispositivos, a checagem de consistência entre essas versões para garantir a interoperabilidade entre os dispositivos e a capacidade de adaptação desses aplicativos à mudanças no contexto de execução.

De fato, o problema de portabilidade do componente de diálogo de sistemas interativos não é novo e sempre foi instigante. Inicialmente, este problema visava à independência do aplicativo em relação à ferramenta (*toolkit*, *toolbox* ou biblioteca de funções de interação) utilizada em uma plataforma específica. Atualmente, o problema não envolve apenas diferentes ferramentas de desenvolvimento de interfaces, mas, sobretudo, diferentes plataformas, ou seja, o problema atual é o desenvolvimento de aplicações interativas que possam ser executadas em múltiplas plataformas e em diversos tipos de equipamentos, os quais podem possuir diferentes capacidades de recursos computacionais. Assim, pode-se ter uma aplicação executada seja em um computador *desktop*, seja na Internet via um navegador ou mesmo em um PDA (*Personal Digital Assistant*) ou celular.

Nesse contexto, a abordagem tradicional para a resolução desse problema, utilizando métodos convencionais de desenvolvimento de interfaces, onde uma especificação separada é criada para cada tipo de dispositivo, gera um esforço elevado para o desenvolvimento de aplicativos multiplataforma. Abordagens mais modernas lidam com o uso do conceito de interfaces plásticas (IP), tecnologia que visa à criação de interfaces de usuário (IU) com capacidade de adaptação a diferentes contextos. Essa abordagem tem como objetivo a reutilização de descrições de interface, eliminando a necessidade da repetição de múltiplos desenvolvimentos para múltiplas plataformas.

O termo “plasticidade” tem como inspiração a propriedade de materiais que se expandem e contraem devido às condições de ambiente sem quebrar, preservando a continuidade de uso (CALVARY, 2002). Sob a ótica da IHC, plasticidade é a capacidade de um sistema de suportar variações no contexto de uso, ainda assim preservando sua usabilidade. Especificamente, nessa dissertação, o termo multiplataforma corresponde à definição de contexto de uso, incluindo múltiplos dispositivos, sendo que alguns trabalhos também consideram múltiplas modalidades (interação gráfica e vocal, por exemplo) e múltiplas condições de ambiente existentes quando o software está sendo executado (condições de luminosidade, perfil do usuário). O principal objetivo da plasticidade é de atender a estes requisitos, preservando a usabilidade da aplicação. Assim, plasticidade não se trata apenas de condensar e

expandir informações de acordo com o contexto de uso, mas também contrair e expandir o conjunto de tarefas disponíveis para o usuário de forma a preservar a usabilidade (CALVARY, 2001).

Visando à reutilização de interfaces através de interfaces plásticas, uma descrição de mais alto nível é necessária de forma a possibilitar seu mapeamento para diferentes tipos de contexto existentes. Uma das maneiras de realizar essa especificação é a utilização de modelos de interface de usuário, os quais têm o propósito de descrever abstratamente a interface a ser criada, e podem ser definidos como uma descrição formal, declarativa e livre de implementação da interface (EISENSTEIN, 2000).

Para fornecer suporte a esse tipo de abordagem, uma das possibilidades é a utilização de linguagens de descrição de interfaces de usuário (LDIU) (LUYTEN, 2004), as quais realizam a especificação em alto nível de interfaces de usuário de forma a permitir seu desenvolvimento independentemente da aplicação. Esta descrição pode ser mapeada (na verdade, concretizada) para diferentes plataformas ou múltiplos contextos de uso, evitando o esforço duplicado de *design*. Dentre as linguagens de descrição de interfaces de usuário existentes, podem ser destacadas algumas que têm como objetivo serem utilizadas no desenvolvimento de interfaces de usuário multiplataforma: UIML (PHANOURIOU 2000), XIML (PUERTA 2002) e UsiXML (LIMBOURG 2004).

Nesse trabalho, nossa solução utiliza a técnica de desenvolvimento de interfaces de usuário baseado em modelos, através de uma linguagem de descrição de interfaces de usuário. O desenvolvimento baseado em modelos foi adotado por permitir a especificação da interface de usuário em diferentes níveis de abstração (p.ex. modelo de tarefas, interface de usuário abstrata, interface de usuário concreta), fornecendo mais opções de abordagens na criação de interfaces. Uma solução utilizando uma LDIU foi escolhida porque esse tipo de linguagem vem sendo amplamente adotado, devido a sua capacidade de abstrair a descrição da IU, fornecendo uma maneira uniforme de criar IU’s multiplataforma e até mesmo multimodais. Alem disso, como a maioria das LDIU’s possui XML como base, essas linguagens são de fácil aprendizado, tendo um grande potencial de adoção pela comunidade de desenvolvedores.

Dentre as linguagens citadas acima, a UsiXML foi escolhida para a realização desse trabalho por ser uma linguagem de descrição de interfaces de usuário baseada em modelos, e, alem de suas características técnicas, também por que seu material e informações estão disponíveis sem custo, e principalmente por possuir uma comunidade de desenvolvedores e pesquisadores aberta e ativa atualmente, que oferece um efetivo suporte e um rico ambiente de discussão entre seus usuários. Os níveis de descrição da linguagem UsiXML são baseados no *framework* de referência chamado *Cameleon*, que será descrito no capítulo 2.

A criação de interfaces de usuário multiplataforma a partir de uma descrição em UsiXML implica em uma serie de passos de mapeamento de uma descrição de mais alto nível para descrições em níveis mais concretos até o estagio final da interface de usuário, que pode ser executada nos dispositivos-alvo da aplicação. Este processo é melhor descrito no capitulo 2. De forma a possibilitar a utilização de UsiXML no desenvolvimento de software, são necessárias ferramentas capazes de realizar estes mapeamentos, de maneira automática ou assistida, culminando na reificação da interface de usuário concreta na plataforma-alvo do aplicativo.

O objetivo geral deste trabalho é, portanto, a investigação e o desenvolvimento de RenderXML, o qual é um renderizador de interfaces de usuário projetado para atuar no ultimo nível desse processo, mapeando interfaces de usuário concretas para interfaces de usuário finais em um dispositivo específico. Além disso, o RenderXML fornece um outro nível de independência, permitindo a conexão da IU renderizada com lógicas de aplicação desenvolvidas em diferentes linguagens de programação. No entanto, deve ser deixado claro que a ferramenta desenvolvida é um renderizador e não uma ferramenta de *design*, e assim não possui como objetivo detectar e/ou solucionar problemas de usabilidade e limitações existentes em cada plataforma. Essas questões devem ser atendidas por etapas anteriores do ciclo de vida da IU, tanto de forma manual, como automática, através da utilização de outras ferramentas que trabalham com UsiXML.

Como a UsiXML permite a especificação de todas as características necessárias de uma IU concreta, o RenderXML pode ser utilizado no desenvolvimento ou prototipação de interfaces de usuário finais. Nessas situações, a possibilidade de criar interfaces de usuário para múltiplas plataformas utilizando apenas uma linguagem é de extrema importância, pois faz com que o desenvolvedor precise conhecer apenas uma tecnologia, a UsiXML. Além disso, a possibilidade de conexão com lógicas de aplicação desenvolvidas em diferentes linguagens de programação amplia o escopo de utilização da ferramenta.

Visando ilustrar sua aplicação, RenderXML é utilizado e testado no desenvolvimento de *software* multiplataforma, em particular para a criação de livros eletrônicos falados, tema do projeto LIFAPOR, no qual este trabalho está inserido.

Esse trabalho está organizado da seguinte forma: o capítulo 2 apresenta os conceitos utilizados para a elaboração do trabalho, assim como os trabalhos relacionados. O capítulo 3 apresenta a proposta do RenderXML e sua implementação, sendo que os estudos de caso realizados para validar a ferramenta são mostrados no capítulo 4. As conclusões obtidas com o trabalho, assim como suas limitações e as perspectivas de trabalhos futuros são apresentadas no capítulo 5.

2 ESTADO DA ARTE

Esse capítulo tem como objetivo apresentar os fundamentos necessários para uma melhor compreensão desse trabalho. Dessa maneira, o capítulo está dividido em duas seções, sendo que primeiramente serão apresentados os conceitos existentes no desenvolvimento de interfaces de usuário multiplataforma, e então serão mostrados os trabalhos já desenvolvidos relacionados a este tema, apresentando as linguagens de descrição de interfaces de usuário propostas, assim como as ferramentas desenvolvidas para trabalhar com essas linguagens.

2.1 Conceitos

Essa seção visa à introdução dos conceitos existentes no desenvolvimento de interfaces de usuário multiplataforma. Na seção 2.1.1 será apresentado o desenvolvimento de interfaces de usuário, ressaltando porque esse tópico é de extrema importância no campo de desenvolvimento de *software*, e quais são os desafios existentes.

O conceito de plasticidade é mostrado em seguida, na subseção 2.1.2, esclarecendo os objetivos almejados pelo desenvolvimento de interfaces de usuário multiplataforma atualmente. A subseção 2.1.3 apresenta as abordagens existentes para o desenvolvimento desse tipo de tecnologia, sendo que o desenvolvimento baseado em modelos, por ser parte do tema desta dissertação, é explicado com maiores detalhes na subseção 2.1.4.

2.1.1 Desenvolvimento de Interfaces de Usuário

O desenvolvimento de interfaces de usuário é uma atividade de extrema importância para a produção de programas de computador, em parte por que consome uma grande quantidade dos recursos necessários para o desenvolvimento de *software*, e também porque a IU é a seção do aplicativo com a qual o usuário possui o maior contato, sendo que uma interface de qualidade é imprescindível para a boa aceitação de uma aplicação interativa.

Devido a essa importância, o campo de desenvolvimento de interfaces de usuário tem sido tema de pesquisa para diversos trabalhos produzidos recentemente, visando a redução da sua complexidade, além da obtenção de uma maior produtividade e qualidade nos seus resultados.

Entre os campos pesquisados pode ser ressaltada a criação de modelos de interfaces de usuário, já que nos últimos quinze anos, diversas abordagens de desenvolvimento de interfaces de usuário baseadas em modelos foram propostas.

No entanto, apesar da diversidade das abordagens e modelos propostos, na prática o desenvolvimento de interfaces de usuário permanece sendo realizado manualmente na

maioria dos casos, em grande parte porque os modelos são considerados muito teóricos e a modelagem é vista como uma complicação desnecessária para a criação das interfaces (PATERNÒ, 2005). Essa situação também pode ser atribuída ao fato de que as interfaces de usuário devem respeitar critérios de usabilidade que variam conforme o contexto, o que ainda não é atendido plenamente pelas soluções propostas em trabalhos de pesquisa.

Apesar disso, os modelos de interface de usuário têm recebido uma maior atenção ultimamente, devido ao surgimento de novas necessidades, as quais tornaram o desenvolvimento de IU uma tarefa de maior complexidade, reduzindo consequentemente a produtividade obtida por abordagens de desenvolvimento *ad hoc*. A principal delas pode ser considerada o surgimento e adoção de diversas novas plataformas computacionais por um grande número de usuários, as quais podem ser utilizadas em uma grande variedade de ambientes.

Com essa variedade de dispositivos computacionais a disposição dos usuários, as organizações devem possuir a capacidade de criar aplicativos os quais possam ser utilizados nessas diversas plataformas, sendo que esses aplicativos devem possuir uma apresentação diferenciada, não apenas tentando se adaptar com o redimensionamento de componentes de interface (PATERNÒ, 2002). Alem disso, os aplicativos devem diferir nas tarefas que podem ser realizadas pelo usuário, adaptando-se as características impostas pela plataforma utilizada e pelo ambiente onde a aplicação está sendo executada, criando a necessidade de desenvolvimento de interfaces adaptativas, as quais possam moldar-se automaticamente ao contexto de uso da aplicação.

2.1.2 Plasticidade

Plasticidade tem relação com o problema de portabilidade do componente de diálogo de sistemas interativos. Este problema não é novo e sempre foi instigante. Inicialmente, este relacionava-se à independência do aplicativo em relação à ferramenta (*toolkit*, *toolbox* ou biblioteca de funções de interação) utilizada em uma plataforma específica.

Trabalhos pioneiros baseavam-se na uniformização da comunicação entre aplicação e componente de diálogo, de forma que a aplicação usasse o mesmo protocolo de comunicação para invocar funções de diálogo em qualquer ambiente.

Dentre os trabalhos acadêmicos desenvolvidos nessa área no país podem ser citados (FRAINER, 1990; PIMENTA, 1991; PIMENTA, 1992), os quais estabeleciam uma camada intermediária entre a aplicação e cada ferramenta nativa em um ambiente (na época, Mac *Toolbox* no Apple Macintosh, *Toolkit* do Microsoft-Windows). Na verdade, esta camada intermediária oferecia uma interface abstrata comum que deveria ser mapeada para as interfaces concretas das ferramentas nativas subjacentes. Deste modo, o *look and feel* do programa era o *look and feel* da ferramenta subjacente, característica adequada aos usuários habituados à consistência de um determinado ambiente.

Atualmente, o conceito de interfaces plásticas está sendo objeto de muitos trabalhos de pesquisa, e tido como a evolução natural no processo de desenvolvimento de interfaces de usuário.

No contexto atual, as interfaces plásticas procuram solucionar muito dos problemas encontrados no desenvolvimento de interfaces de usuário, possibilitando a criação de um modelo de interface adaptável, o qual possa ser utilizado em diferentes plataformas e contextos de uso sem nenhuma necessidade de modificação do código do aplicativo.

Nessa situação, o termo *plasticidade* é inspirado na propriedade de materiais que podem expandir e contrair devido a ação de requisitos naturais, ainda assim preservando sua possibilidade de utilização (CALVARY, 2002).

No contexto de Interface Homem-Computador (IHC), *plasticidade* é a capacidade de uma interface de aplicação de suportar a variação de contextos de uso preservando sua usabilidade. Um contexto de uso para um sistema plástico pode ser interpretado em duas categorias (CALVARY, 2002; CALVARY, 2001a; CALVARY, 2001):

- Os atributos de *hardware* e *software* de uma plataforma utilizada para a interação do usuário com o sistema. Como exemplos desses atributos constam o tamanho do monitor e a largura de banda existente no dispositivo computacional onde a aplicação está sendo utilizada.
- Os atributos de ambiente que descrevem as condições existentes no local e momento em que o aplicativo está sendo executado. Essas características incluem os objetos, pessoas e eventos periféricos à tarefa atual que podem ter impacto no sistema ou no comportamento do usuário, nesse momento ou no futuro. Tipicamente podem ser consideradas as condições de luminosidade e ruído, as quais podem prejudicar a execução de determinados tipos de operação. No nível de tarefas, o contexto pode dar informações sobre a relevância de informações, tendo em vista que, por exemplo, tarefas que são principais no escritório podem ser secundárias em um trem.

De acordo com essa definição, o contexto de uso pode incorporar o mundo inteiro (CALVARY, 2001a; CALVARY, 2001). Na prática, o limite é especificado pelo desenvolvedor da aplicação, o qual determina as condições que devem ser consideradas relevantes no aplicativo em questão.

Nesta seção serão apresentados os requisitos a serem observados no desenvolvimento de Interfaces Plásticas, de forma que se possa obter um resultado de melhor qualidade, e também possuir métricas para comparação entre diferentes trabalhos.

2.1.2.1 Usabilidade e Interfaces Plásticas

Um dos pontos de avaliação para sistemas interativos é o conceito de usabilidade, o qual é o termo técnico utilizado para avaliar a qualidade de uso de uma interface de usuário.

No contexto de Interfaces Plásticas, todos os conceitos de usabilidade existentes para o desenvolvimento de interfaces de usuário devem ser mantidos. No entanto, deve ser considerada adicionalmente a manutenção da usabilidade nos diferentes contextos de uso onde o aplicativo pode ser utilizado, sendo que a preservação de usabilidade de uma interface plástica acontece se as propriedades determinadas na fase de projeto da aplicação são mantidas dentro de um escopo pré-definido de valores na medida em que ocorra a adaptação para diferentes contextos de uso (CALVARY, 2001a).

2.1.2.2 Processo de Adaptação

Para possuir a capacidade de execução em diferentes plataformas e ambientes, uma aplicação que utiliza Interfaces Plásticas deve suportar um processo de adaptação, o qual é responsável por realizar a migração de um contexto de uso para outro.

A adaptação plástica pode ser estruturado como um processo de cinco etapas (CALVARY, 2001):

- Detecção das condições para adaptação: identificação da necessidade ou oportunidade para uma mudança de contexto, de forma que o usuário possa ter a sua disposição a melhor interface para contexto atual. A detecção de condições para adaptação pode ser realizada de forma automática ou pelo usuário, de acordo com o tipo de sistema.
- Identificação das interfaces de usuário candidatas: seleção das interfaces que possam ser utilizadas no contexto atual. Essa seleção pode ser computada durante a execução, ou selecionada a partir de um conjunto de interfaces de usuário computadas previamente, ou ainda a partir de um conjunto de usuários *ad hoc* pré-definido.
- Seleção de uma interface de usuário: estratégia de resolução de problemas utilizada para a escolha da melhor interface entre as candidatas especificadas anteriormente. Essa seleção pode ser realizada de forma automática ou assistida pelo usuário.
- Transição da interface de usuário atual para a nova interface selecionada: etapa onde a interface sendo utilizada é finalizada, sendo iniciada a execução da nova interface que foi selecionada no passo anterior.
- Execução da nova interface de usuário selecionada: nessa etapa, a nova interface de usuário pode ser inicializada sem nenhuma informação de estado, ou a partir do estado no qual a interface anterior foi finalizada.

Em relação ao processo de adaptação implementado pelo aplicativo, uma das características que podem ser observadas é a necessidade de intervenção do usuário para a migração de contexto, como pode ser observado na Tabela 2.1.

Tabela 2.1: Espaço de classificação para interfaces de usuário plásticas (CALVARY, 2001).

	Plataforma	**Ambiente**	**Plataforma e Ambiente**
O Sistema	Semi-Plasticidade adaptativa	Semi-Plasticidade adaptativa	Plasticidade Adaptativa
O Usuário e o Sistema	Semi-Plasticidade Mista	Semi-Plasticidade Mista	Semi-Plasticidade Mista
O Usuário	Semi-Plasticidade Adaptável	Semi-Plasticidade Adaptável	Plasticidade Adaptável

Em sistemas com *plasticidade adaptativa*, o aplicativo é responsável por todo o processo de migração de contexto, não sendo necessária a operação do usuário em nenhum momento. Já em sistemas com *plasticidade adaptável*, é necessária a intervenção do usuário em todas as etapas do processo de adaptação, tanto em relação à plataforma utilizada quanto em relação ao ambiente onde a aplicação está sendo utilizada. Em todas as categorias existentes entre esses dois extremos, o sistema é considerado de *plasticidade mista* (CALVARY, 2001).

Com base no processo de adaptação de uma aplicação, foi proposta a métrica de *plasticidade do ponto de vista do usuário* (CALVARY, 2001), a qual deve indicar os esforços físico e cognitivo necessários ao usuário para realizar a migração entre diferentes tipos de contexto. A métrica proposta é ilustrada na Figura 2.1 onde as flechas representam as transições entre os contextos, indicados por pontos na figura, sendo que quanto mais espessa a flecha, mais difícil é para o usuário a realização da migração.

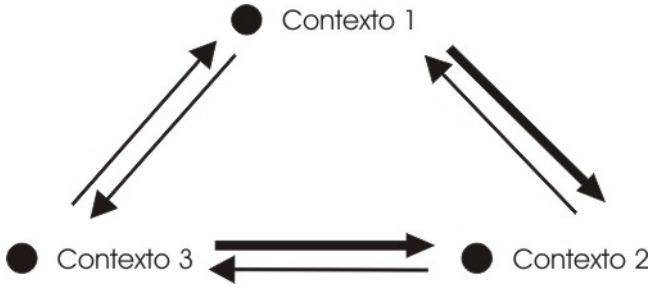

Figura 2.1: Plasticidade do ponto de vista do usuário.

Segundo o autor, apesar de não existirem meios precisos de mensurar-se a dificuldade de migração do ponto de vista do usuário, esse fator deve ser considerado, mesmo que imprecisamente, no desenvolvimento de aplicativos utilizando o conceito de Interfaces Plásticas.

Outra característica a ser considerada em relação ao processo de adaptação do aplicativo é o momento em que esse processo pode ocorrer, sendo que a transição pode ser especificada em *tempo de projeto*, fazendo com que a detecção de contexto ocorra apenas na inicialização do sistema, ou em *tempo de execução*, permitindo a realização da migração de contexto quando o programa já esta em execução.

2.1.2.3 Domínio de Plasticidade

Para se determinar a cobertura que um aplicativo utilizando interfaces plásticas possui em relação às diversas plataformas existentes, foi proposto o conceito de *Domínio de Plasticidade* (CALVARY, 2001), o qual é formado por todas as duplas Ambiente/Plataforma que são suportados pelo aplicativo em questão, e pode ser representado graficamente conforme mostrado na Figura 2.2.

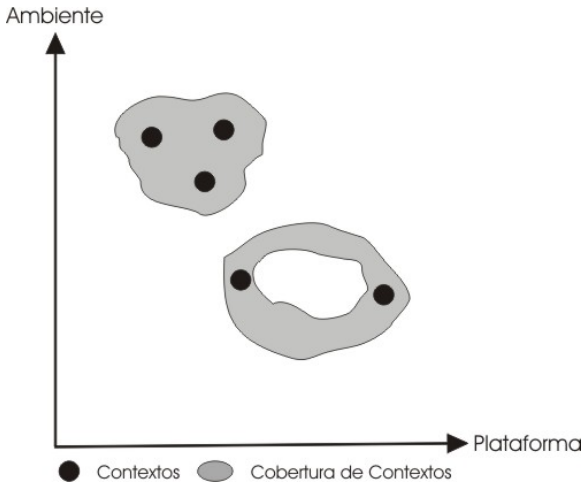

Figura 2.2: Domínio de plasticidade do sistema.

Nessa representação gráfica, a área escura representa o conjunto de contextos para os quais a aplicação está preparada. O limite da área é denominado de limite de plasticidade (*plasticity treshold*), e representa a fronteira dos contextos suportados pelo sistema, sendo que o autor também propõe uma métrica relacionada a esse conceito, denominada de *plasticidade do ponto de vista do sistema*, e pode ser utilizada para determinar o nível de plasticidade que um sistema possui pela quantidade de contextos no qual o mesmo pode ser utilizado.

2.1.3 Abordagens Existentes para o Desenvolvimento de Interface de Usuário Multiplataforma

De forma a desenvolver interfaces para múltiplas plataformas, diversas técnicas foram propostas, as quais podem ser classificadas de acordo com a *World Wide Web (W3C) Note on Authoring Techniques for Device Independence* (W3C, 2004), em três grupos: *single authoring*, *multiple authoring* e *flexible authoring*. Cada uma dessas técnicas é descrita a seguir. Como forma de exemplificar as abordagens de desenvolvimento, alguns trabalhos são citados como exemplo de determinada técnica, sendo que esses trabalhos serão melhor analisados na seção 2.2 desse documento, a qual trata sobre trabalhos relacionados.

Com a utilização da técnica de *multiple authoring*, o desenvolvedor cria um tipo específico de aplicação para cada dispositivo ou categoria. Essa situação ocasiona a (re)criação e manutenção das interfaces de usuário para cada plataforma, sendo extremamente custosa, mas também provendo o maior controle sobre os resultados obtidos. Nesse caso, o desenvolvedor projeta a aplicação para funcionar em apenas um grupo específico de dispositivos, sendo que a sua utilização em dispositivos para os quais a mesma não foi projetada pode resultar em perda de funcionalidade pela aplicação. Essa técnica é a atualmente utilizada em muitas das aplicações que podem ser executadas em múltiplas plataformas, e tem como ponto negativo um alto custo de produção e, principalmente, de manutenção, além de abrir possibilidade para a existência de inconsistências entre as diferentes implementações de uma mesma aplicação (SIMON, 2005).

Já na técnica de s*ingle authoring*, apenas uma implementação da interface de usuário é criada, a qual pode ter que ser adaptada para um dispositivo específico antes de ser apresentada para o usuário. Nesse caso, o desenvolvedor pode fornecer informações que auxiliem o processo de adaptação para cada dispositivo atendido, e normalmente o esforço envolvido na criação da implementação principal da interface de usuário é muito maior do que de cada versão de interface desenvolvida através da técnica de *multiple authoring*. No entanto, o esforço total de criação de todo o conjunto de interfaces deve ser menor utilizando-se *single authoring*.

Uma das possibilidades de implementação dessa técnica são **vocabulários ou *toolkits* independentes de plataforma**, como AUIML (AZEVEDO, 2000) ou UIML (ABRAMS, 1999). Nesse caso, o desenvolvedor descreve a interface utilizando um conjunto genérico de dispositivos. Esse conjunto normalmente é um subconjunto dos dispositivos permitidos para cada tipo de plataforma. A ferramenta de adaptação é então responsável por mapear esses dispositivos para aqueles mais apropriados na plataforma especifica sendo utilizada. Tecnologias para desenvolvimento de interfaces de usuário multiplataforma, como *Java Swing*, a linguagem de desenvolvimento de IU baseada em XML XUL (XUL), a *Microsoft's eXtensible Application Markup Language* XAML

(XAML) e todas as tecnologias de *browsers*, como HTML e WML podem ser associadas a essa técnica (SIMON, 2005).

A segunda possibilidade são técnicas que **estendem linguagens de marcação estabelecidas**, como *Renderer-Independent Markup Language* RIML, desenvolvida como parte do *Consensus Project* (CONSENSUS). Essas técnicas são baseadas no conceito de *author hints*, introduzido no *(W3C) Note on Authoring Techniques for Device Independence*, o qual permite a adição de meta-dados à descrição da interface de usuário. Esses meta-dados podem então ser utilizados na adaptação da interface a um dispositivo ou contexto específico. A linguagem RIML, citada anteriormente, é uma extensão para as linguagens XHTML 2.0 e XForms 1.0 (XFORMS, 2003) que adiciona capacidade de paginação e mecanismos de *layout* independentes de plataforma a essas linguagens.

Uma terceira abordagem são técnicas que utilizam **desenvolvimento de interfaces baseado em modelos**, como XIML (PUERTA, 2002) ou UsiXML (LIMBOURG, 2004). Esse tipo de técnica é o que pode ser utilizado para a implementação do conceito de plasticidade, abordado anteriormente nesse trabalho. As técnicas baseadas em modelo abordam o desenvolvimento de interfaces de usuário sob o ponto de vista de engenharia, ao invés do de *design*. Na prática, isto significa uma preocupação maior com o processo de construção das interfaces do que em relação às características do produto resultante. Apesar de nunca terem sido largamente adotados pelo mercado, possuem uma forte comunidade acadêmica (TRAETTEBERG, 2004).

Em um estágio intermediário, a técnica de *flexible authoring* é configurada pela situação na qual o desenvolvedor combina técnicas de *single* e *multiple authoring*. Dessa forma, o projetista pode criar versões únicas de determinadas interfaces de usuário para serem adaptadas posteriormente, e múltiplas versões de outras interfaces, quando o grau de detalhamento necessário é maior.

Na próxima subseção será apresentado em maiores detalhes o desenvolvimento de interfaces de usuário baseado em modelos, o qual se enquadra na técnica de *múltiple authoring*, e foi utilizado como base para a realização desse trabalho.

2.1.4 Desenvolvimento de Interfaces de Usuário Baseado em Modelos

A utilização de modelos no campo de desenvolvimento de *softwares* já está bastante difundida, como no caso da UML para a programação orientada a objetos, em virtude da capacidade de extração das informações essenciais de cada situação, possibilitando a resolução de problema de forma mais simples.

No desenvolvimento de sistemas interativos, o escopo de possibilidades de projeto é amplo, devendo considerar diversos aspectos, sendo que abordagens baseadas em modelos podem ajudar a gerenciar essa complexidade.

No entanto, muitos pesquisadores têm considerado os modelos nessa área muito teóricos, e um tipo de complexidade sem utilidade. Além disso, os desenvolvedores podem perder a paciência com os modelos, devido à grande quantidade de parâmetros necessários para o desenvolvimento de uma interface de usuário (VANDERDONCKT, 1999), sendo esse fator agravado com a entrada dos mesmos parâmetros para diferentes interfaces que possuem características em comum.

Em um ponto de vista mais abrangente são definidas três gerações de abordagens baseadas em modelos para o desenvolvimento de interfaces plásticas. A primeira

geração é composta por abordagens que utilizaram descrições conceituais entre os níveis abstratos e concretos da interface (PATERNÓ, 2005).

Na segunda geração, foi obtido um consenso em relação à importância do modelo de tarefas no projeto de interfaces de usuário, sendo que esse modelo foi introduzido nas abordagens existentes.

Já na terceira geração, na qual estão inseridos os trabalhos desenvolvidos atualmente, a principal questão existente é o desenvolvimento de interfaces para múltiplos dispositivos. Com o surgimento dessa necessidade, a complexidade existente no desenvolvimento de interfaces de usuário aumentou consideravelmente, fazendo crescer o interesse por abordagens baseadas em modelos.

Apesar disso ainda não foi desenvolvido um consenso nessa área sobre quais modelos e como os modelos devem ser utilizados para a criação desse tipo de interface, existindo atualmente apenas propostas a serem analisadas.

Uma das propostas existentes (BOUILLON, 2002a) abordou a reengenharia de paginas Web. Esse trabalho sugere a criação dos modelos de diálogo e apresentação a partir de paginas Web existentes. A partir dos modelos criados automaticamente, passa a ser possível a instanciação da página modelada em uma plataforma diferente.

Nessa proposta, o modelo de apresentação representa o que é apresentado ao usuário através da interface, e é formado por elementos abstratos, os quais são baseados nos elementos existentes na interface de origem, em uma forma independente de plataforma. Já o modelo de diálogo representa as interações que o usuário pode realizar com a interface, sendo formado nessa implementação pelos *links* existentes na página de origem, os quais são utilizados para a criação de uma estrutura de navegação entre as páginas, podendo ser reproduzida na plataforma de destino.

A partir dos modelos de apresentação e diálogo, segundo o autor, qualquer abordagem baseada em modelo é capaz de gerar uma nova interface concreta, não sendo essa etapa implementada no trabalho em questão.

Apesar de a pesquisa ter sido desenvolvida para possibilitar a reengenharia de paginas *Web* para múltiplas plataformas, o autor afirma não pertencer ao escopo do trabalho a realização da operação para múltiplos contextos de uso, onde além de restrições de plataforma, restrições de usuário e ambiente também são enfrentadas.

Em outro trabalho (CALVARY, 2001) foi desenvolvido um *framework* baseado em modelos de interface de usuário para a geração de interfaces plásticas, o qual foi revisado posteriormente (CALVARY, 2002). Nesse *framework*, denominado *Cameleon Reference Framework*, foi estabelecido um processo de criação, composto pelos modelos de Conceitos, Tarefas, Plataforma, Ambiente, Interadores e Evolução, o qual deve ser seguido para cada contexto de uso onde a aplicação deve ser executada.

O processo criado é separado em quatro níveis de abstração, conforme mostrado na Figura 3. No primeiro nível (*Tasks and Concepts* na Figura 2.3), o modelo de Conceitos (*Concepts* na Figura 2.3) captura as informações relativas ao domínio da aplicação, sendo complementado pelo modelo de Tarefas (*Tasks* na Figura 2.3), o qual descreve como o usuário deve realizar as tarefas oferecidas pela aplicação, e pelo modelo de Plataforma (*Platform* na Figura 2.3). O modelo de Plataforma, o qual é utilizado também no segundo nível do processo, chamado de interface de usuário abstrata, (*Abstract User Interface* na Figura 2.3), define a plataforma onde o aplicativo deverá ser

executado. No terceiro nível, chamado de interface de usuário concreta (*Concrete User Interface* na Figura 2.3), o modelo de Ambiente (*Environment* na Figura 2.3) define as restrições estabelecidas pelo ambiente de execução, enquanto o modelo de Interadores (*Interactors* na Figura 2.3) define quais componentes estão disponíveis para implementação da interface concreta.

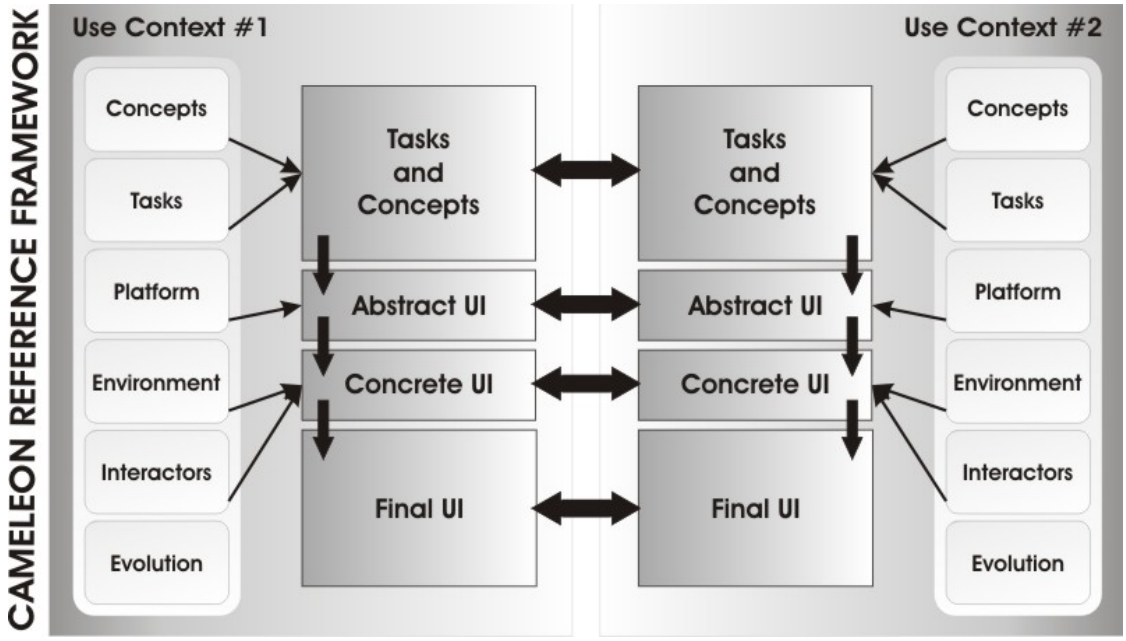

Figura 2.3: Processo de desenvolvimento de Interfaces Plásticas (CALVARY, 2001).

Nesse ponto, o processo chega ao quarto e último nível, denominado interface de usuário final (*Final User Interface* na Figura 2.3), o qual é formado pela interface final da aplicação, executando em um determinado dispositivo. Com a interface pronta para o contexto de uso especificado, o modelo de Evolução (*Evolution* na Figura 2.3) é utilizado para representar as condições de entrada e saída desse contexto, isto é, as condições que devem existir para a aplicação poder ser executada nesse determinado contexto de uso.

O *framework* de referência C*ameleon* prevê que os passos de desenvolvimento possam ser obtidos através de transformações verticais e horizontais. As transformações verticais definem os processos para a transformação de um passo de desenvolvimento - um modelo de interface de usuário - em outro mais concreto (processo de reificação), ou em outro mais abstrato (processo de abstração). As transformações horizontais definem processos para a obtenção de um modelo de IU a partir de outro de mesmo nível de reificação, mas em um contexto de uso diferente do modelo original (processo de tradução).

Dessa forma, para se realizar o desenvolvimento de interfaces de usuário multiplataforma na prática, o processo apresentado na Figura 2.4 deve ser seguido. Nesse processo, para se obter uma interface de usuário final, um modelo de tarefas genérico deve ser gerado (*Task Model* na Figura 2.4), o qual é especificado para uma categoria de dispositivos, gerando um modelo de tarefas dessa categoria (*Task Model Desktop* na Figura 2.4). A partir do modelo de tarefas específico, a interface de usuário é mapeada para uma interface de usuário abstrata (*Abstract UI Desktop* na Figura 2.4), a

qual é independente do tipo de interação sendo utilizado. Esse modelo pode então ser mapeado para uma interface de usuário concreta (*Concrete UI Desktop* na Figura 2.4), sendo independente de dispositivo. Finalmente, a interface de usuário concreta pode então ser mapeada para uma interface de usuário final a ser executada no dispositivo alvo da aplicação. Todos esses passos podem ser realizados de forma manual ou com a utilização de ferramentas, que mapeiam a interface de um nível para outro de maneira automática ou assistida.

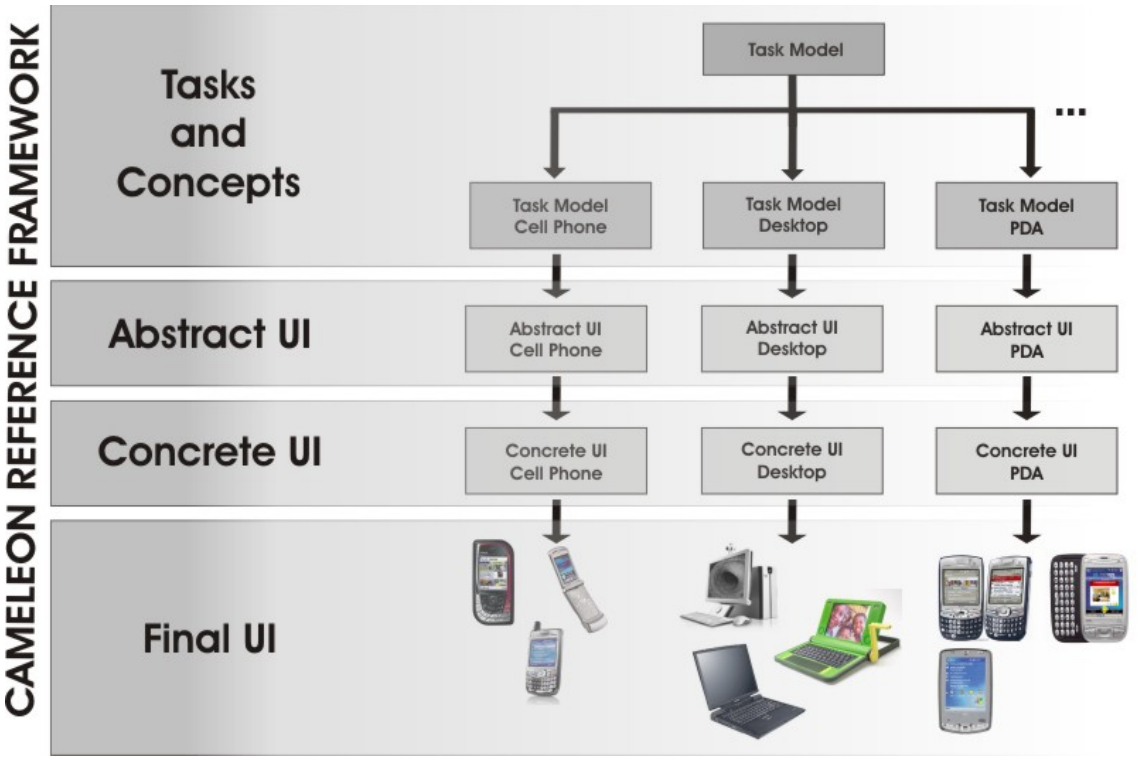

Figura 2.4: Processo para desenvolvimento de interfaces de usuário multiplataforma baseado no *framework* de referência *Cameleon*.

2.2 Trabalhos Relacionados

Com a introdução dos conceitos necessários na seção anterior, esta seção é dedicada à apresentação dos trabalhos relacionados existentes na literatura. A seção é subdividida em duas partes, sendo que a primeira mostrará as linguagens de descrição de interfaces de usuário existentes, explicando com maiores detalhes a UsiXML, linguagem utilizada para o desenvolvimento desse trabalho. A segunda será composta pelas ferramentas de desenvolvimento de interfaces de usuário multiplataforma já propostas.

2.2.1 Linguagens de Descrição de Interfaces de Usuário

As linguagens de descrição de interfaces de usuário foram propostas como uma alternativa para o desenvolvimento de interfaces para múltiplas plataformas. O conceito proposto foi o desenvolvimento de linguagens em um alto nível de abstração, passíveis de serem mapeadas para diferentes tipos de plataforma.

Nesse campo de pesquisa, as linguagens de marcação baseadas em XML tornaram-se um padrão *de facto* entre todos os projetos desenvolvidos. Isso pode ser relacionado a uma série de motivos (PHANOURIOU, 2000):

- Requerem pouca experiência em programação, e são utilizáveis por programadores inexperientes.
- Podem ser geradas automaticamente a partir de construtores visuais, dispensando a necessidade dos programadores recordarem todas as construções da linguagem.
- Não requerem compilação
- Ocupam menos espaço de armazenamento do que código compilado, o que as torna mais rápidas de obter através de uma rede.
- São armazenadas em texto puro e por isso mais resistentes a erros de bits.

Nesta seção serão apresentadas as principais propostas na área de linguagens de descrição de interfaces de usuário, e como elas podem ser utilizadas para o desenvolvimento de interfaces de usuário multiplataforma.

2.2.1.1 UIML – User Interface Markup Language

A linguagem UIML (*User Interface Markup Language*) (ALI, 2002) surgiu com o objetivo de ser uma linguagem independente de dispositivo, e que também possibilitasse a criação de interfaces de usuário por pessoas sem experiência em programação, fenômeno obtido pela linguagem HTML, a qual também é uma linguagem de marcação.

A descrição de interfaces em UIML é dividida em três seções, *interface*, *vocabularies* e *logic*, as quais são relacionadas como demonstrado na Figura 2.5.

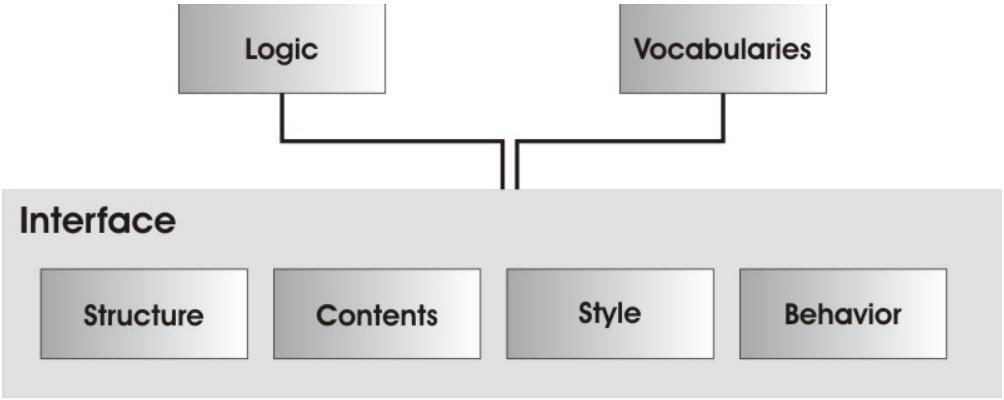

Figura 2.5: Estrutura da linguagem de descrição de interfaces de usuário UIML.

A seção *interface* contém a descrição da interface de usuário, e é subdividida em quatro seções. A seção *structure* descreve os elementos de interface, como listas e campos de texto que serão exibidos no projeto em questão. As propriedades de cada elemento de interface são definidas na seção *property*. A seção *content* especifíca a localização de conteúdo externo que deve ser adicionado à interface, e a seção *behavior* contêm ações que devem ser executadas em resposta aos eventos que ocorrem na interface.

Na seção *vocabularies* está especificado o vocabulário que será utilizado para mapear a definição da interface. Dessa forma, a tecnologia utilizada pela interface pode ser facilmente modificada, devendo-se apenas substituir o vocabulário utilizado.

A seção *logic* determina como a interface é conectada à lógica da aplicação, descrevendo o mapeamento de eventos na interface com funções do aplicativo sendo executado.

Apesar de a UIML ser uma linguagem desenvolvida para ser utilizada na criação de interfaces para múltiplos dispositivos, (LUYTEN, 2004) aponta um ponto negativo na estrutura da linguagem, a qual não permite a definição de restrições de *layout* entre componentes, como quais componentes devem permanecer próximos, prejudicando a portabilidade para diversos dispositivos.

2.2.1.2 TeresaXML

A linguagem de descrição de interface TERESAXML (BERTI, 2004) foi desenvolvida para fornecer suporte ao ambiente de desenvolvimento de interfaces TERESA (*Transformation Environment for Interactive Systems Representations).* Essa linguagem é composta por elementos de interação, elementos de composição e conexões.

Os elementos de interação são as representações abstratas dos elementos da interface, sendo que a linguagem define uma diferenciação entre elementos que possuem interação real com o usuário e elementos de saída, os quais apenas apresentam dados na interface. Os elementos de composição definem composições entre interadores, as quais podem ser do tipo agrupamento, relacionamento, ordenação e hierarquia. Já os elementos de conexão são responsáveis pela definição do comportamento dinâmico da interface de usuário.

2.2.1.3 XIML – Extensible Interface Markup Language

Outra linguagem de descrição de interfaces de usuário proposta é a XIML (*Extensible Interface Markup Language*) (PUERTA, 2002), que se propõe a representar interfaces permitindo o suporte universal à funcionalidade através de todo o ciclo de vida de uma interface (ROSA, 2005).

A linguagem XIML define uma interface através de uma coleção organizada de elementos, os quais são categorizados em um ou mais componentes. Os componentes definidos pela versão inicial da XIML são:

- Tarefa: descreve o processo de negócio ou tarefas que a interface suporta.
- Domínio: define um conjunto organizado de objetos e classes de objetos, os quais podem ser manipulados através da interface pelo usuário.
- Usuário: hierarquia de usuários da interface.
- Apresentação: hierarquia de elementos de interação que compões os objetos concretos que se comunicam com o usuário através da interface.
- Diálogo: coleção estruturada de elementos que determinam as ações de interação que estão disponíveis para o usuário na interface.

Apesar de ser uma linguagem completa, a principal dificuldade em se estudar a XIML se deve ao fato de que informações relativas à linguagem são distribuídas apenas aos membros do XIML Fórum, organização que se propõe à pesquisa, disseminação, adoção e padronização da XIML. Esse fator torna-se um complicador no momento em que, para participar do fórum, deve ser aceita uma licença que proíbe a ampla disseminação de

qualquer informação contida no fórum, e também a distribuição completa de aplicações desenvolvidas utilizando XIML, mesmo através de trabalhos científicos.

2.2.1.4 UsiXML – User Interface Extensible Markup Language

A *User Interface eXtensible Markup Language* (UsiXML) é uma linguagem de descrição de interfaces de usuário proposta com o objetivo de capturar as propriedades essenciais para a especificação, descrição, projeto e desenvolvimento de interfaces de usuário (LIMBOURG, 2004).

Como forma de atingir esse objetivo, a UsiXML é estruturada de acordo com os quatro níveis de abstração definidos no *framework* de referência *Cameleon*, o qual foi apresentado anteriormente nesse documento.

A UsiXML interpreta os quatro níveis básicos desse framework da maneira descrita a seguir (LIMBOURG, 2004):

- *Task and Concepts*: nível mais alto, onde as tarefas interativas (*tasks*) a serem realizadas são definidas sob o ponto de vista do usuário, em conjunto com os objetos (*concepts*) que são manipulados para a realização destas tarefas.
- *Abstract User Interface* (AUI): Nível de abstração em que a interface com usuário é definida de forma independente de qualquer modalidade de interação. Pode-se considerar que uma definição de AUI é a expressão da reificação de um modelo de conceitos e tarefas, mas de maneira independente de qualquer modalidade de interação. A AUI define *containers* abstratos e componentes individuais através do agrupamento de tarefas de acordo com certos critérios (por ex: padrões estruturais de modelo de tarefa, análise de carga cognitiva, identificação de relacionamentos semânticos), um esquema de navegação entre *containers*, e seleciona componentes abstratos individuais para cada conceito de forma que sejam independentes de quaisquer ambientes de interação.
- *Concrete User Interface* (CUI): Definição que se caracteriza por ser independente do tipo de plataforma computacional ou conjunto de dispositivos de determinada plataforma. Uma CUI é composta de CIOs (*Concrete Interaction Objects*), os quais são uma representação abstrata de elementos encontrados em bibliotecas gráficas (Java *AWT/Swing*, por exemplo). Portanto, uma CUI consiste de uma decomposição hierárquica de CIOs de um determinado conjunto de componentes da plataforma-alvo em um dado contexto de uso. A CUI é uma representação abstrata de uma FUI (ver adiante) de forma que seja independente de qualquer plataforma computacional ou das linguagens de programação usadas para desenvolver interfaces de usuário. Entretanto, CUIs dependem de um ambiente de interação, ou seja, uma instância de uma CUI endereça um único ambiente de interação por vez. Os artefatos de uma CUI, geralmente, são especificações em XML da hierarquia de CIOs que compõe a IU.
- *Final User Interface* (FUI): Descrição final da interface de usuário, a qual pode ser executada ou interpretada em um determinado contexto de uso (ou seja, em uma plataforma específica, com um conjunto de dispositivos específicos, utilizando objetos de interação específicos). Os artefatos de uma

> FUI são os códigos-fonte de linguagens que implementam a IU - tais como Java e HTML - e os códigos-objeto (executáveis) que renderizam a IU.

Para representar os níveis de abstração do *framework* de referência, a UsiXML define os conceitos que serão apresentados a seguir.

O modelo de tarefas (*Task Model*) descreve as diversas tarefas que podem ser realizadas pelo usuário em uma interação com o sistema computacional. Para modelar as tarefas do sistema, UsiXML utiliza uma versão estendida da notação *ConcurTaskTree* (CTT) (MORI, 2003). Nessa representação, um modelo de tarefas é composto por tarefas e relacionamento entre elas. As tarefas são descritas por um nome, um tipo e sua freqüência. Já os relacionamentos entre tarefas podem ser de dois tipos: decomposição, permitindo a representação de modelos hierárquicos, e temporal, possibilitando a especificação de uma relação temporal (seqüência, simultaneidade, etc..) entre tarefas de mesmo nível.

O modelo de domínio (*Domain Model*) descreve os conceitos do mundo real e suas associações conforme compreendido pelo usuário. A UsiXML descreve os conceitos contidos no modelo de domínio da mesma forma que um diagrama de classes UML. Entre os conceitos existentes estão classes, atributos, métodos e relações entre objetos do domínio.

De forma a permitir a definição independente do conteúdo presente na interface, o modelo de recursos (*Resource Model*) permite que se especifique o conteúdo externo, como mensagens e imagens, por exemplo, de forma separada da definição dos componentes da interface. Desse modo, a adaptação da interface de usuário para diferentes contextos de uso passa a ser facilitada, sendo necessária a modificação apenas do modelo de recursos.

O modelo de contexto (*Context Model*) define todas as entidades que podem influenciar na execução de uma tarefa interativa pelo usuário com a interface de usuário sendo especificada. O modelo de contexto deve capturar qualquer atributo relevante do contexto de uso onde a aplicação se encontra, sendo composto por: modelo de usuário, o qual classifica os usuários do aplicativo em diferentes perfis (estereótipos), modelo de plataforma, o qual captura informações relevantes da combinação de software-hardware da plataforma sendo utilizada, e modelo de ambiente, o qual define propriedades de interesse do ambiente físico onde a aplicação está sendo executada.

A interface de usuário abstrata (*Abstract User Interface*) representa uma expressão canônica da interface de usuário de forma a ser independente de qualquer modalidade de interação ou plataforma computacional. Uma AUI é composta por objetos de interação abstratos (*Abstract Interaction Objects*), os quais consistem em uma abstração de componentes presentes na maioria das plataformas, tanto gráficas, como janelas e botões, como para outros modos de interação, como vocal, por exemplo.

A interface de usuário concreta (*Concrete User Interface*) permite a especificação de uma interface de usuário de forma que seja dependente de modalidade, definindo o tipo de interação que será utilizado pela interface, e independente de plataforma, tornando possível a renderização da descrição de interface para múltiplos tipos de plataforma.

A definição de CUI em UsiXML é caracterizada por representar um conjunto de *Concrete Interaction* Objects (CIO), cada qual com a informação de suas características.

O layout de uma CUI é definido sem qualquer informação de posição absoluta, mas sim com definições de elementos do tipo gerenciadores de *layout*, os quais agrupam componentes que possuam relações de posição na interface.

Além da definição dos componentes que compõem a interface de usuário, a CUI possui um mecanismo capaz de estabelecer o comportamento dinâmico da interface, incluindo uma linguagem de definição de navegação e uma linguagem para definição da relação entre eventos e ações (*event/action*).

Um exemplo de declaração de uma interface contendo um *label* e um botão é mostrado na figura abaixo (Figura 2.6). Basicamente, esta especificação descreve uma janela (*window*), a qual tem como título default a frase *Hello World*. Além disso, é definido um gerenciador de *layout* do tipo *gridBagBox* para a janela.

```
<cuiModel id="hello_world-cui_12" name="hello world-cui">
    <window id="window_component_0" name="window_component_0"
            defaultContent="Hello world" width="470" height="255">
      <gridBagBox id="grid_bag_box_1" name="grid_bag_box_1"
            gridHeight="12" gridWidth="23"/>
    </window>
</cuiModel>
```

Figura 2.6: Especificação UsiXML de uma CUI.

Em adição aos modelos citados acima, a especificação da UsiXML também cobre aspectos dinâmicos do ciclo de vida de desenvolvimento de interface de usuário, como engenharia reversa, adaptação de contexto de uso e especificação de diálogo, mas estes aspectos não serão detalhados neste documento, tendo em vista que fogem do escopo do trabalho realizado.

2.2.2 Ferramentas para Criação de Interfaces de Usuário MultiPlataforma

Diversos trabalhos foram realizados objetivando a criação de interfaces de usuário multiplataforma, sendo que os mais relevantes serão apresentados nessa seção. Os trabalhos desenvolvidos serão separados em duas categorias, ferramentas que trabalham com a linguagem UsiXML e renderizadores de interfaces de usuário, tanto para UsiXML como para outras linguagens de descrição.

2.2.2.1 Ferramentas para a Linguagem UsiXML

Dentre as ferramentas que manipulam UsiXML, destaca-se o SketchiXML (COYETTE, 2004), capaz de gerar uma descrição concreta de interface de usuário (CUI) em UsiXML utilizando como entrada descrições de interface desenhadas à mão (*sketchs*). A SketchiXML se diferencia das demais ferramentas de interpretação de desenhos feitos a mão, já que fornece como saída uma descrição de interface de usuário independente de linguagem de programação, ao invés de uma linguagem de programação específica.

Já a ferramenta GrafiXML (LEPREUX, 2006) trabalha como uma forma diferente de entrada de dados, sendo um editor de recursos visuais, através do qual o usuário especifica a interface desejada posicionando os componentes na tela e gerando como

saída a descrição da interface projetada em UsiXML. GrafiXML é similar a qualquer editor de recursos visuais, exceto que permite a manipulação de mais propriedades dos elementos do que somente as propriedades físicas. Além disso, a ferramenta GrafiXML foi desenvolvida de forma a permitir a criação de *plug-ins* que ampliem as funcionalidades da mesma. Entre os *plug-ins* desenvolvidos estão o *ComposiXML* (LEPREUX, 2007), o qual permite a composição de interfaces de usuário a partir de elementos previamente configurados, e o *PlastiXML* (COLLIGNON, 2008), que descreve transições de interface dependentes do modelo de contexto, em particular a plataforma utilizada.

Ainda na categoria de editores visuais, VisiXML (VAN SLUYS, 2004) se posiciona como um editor de interfaces de fidelidade média, em relação a editores de fidelidade baixa, como o SketchiXML, e editores de fidelidade alta, como o GrafiXML. O VisiXML permite a descrição de interfaces de usuário utilizando a ferramenta Microsoft Visio, e gerando como saída interfaces de usuário abstratas e concretas descritas em UsiXML. A ferramenta tem como intuito permitir a descrição de interfaces por usuários leigos, sem experiência em programação.

Na área de engenharia reversa de interfaces de usuário, o aplicativo ReversiXML (BOUILLON, 2006) permite a transformação de páginas *Web* descritas em HTML para descrições de interfaces de usuário nos níveis abstrato e concreto em UsiXML. Dessa maneira, podem-se criar versões para diferentes plataformas a partir de páginas já existentes.

De forma mais abrangente, a ferramenta TransformiXML (STANCIULESCU, 2005) permite a realização de transformações de descrições UsiXML em qualquer um dos níveis existentes para outra descrição UsiXML, adaptada para um diferente contexto. Desse modo, o TransformiXML fornece suporte a engenharia avante, engenharia reversa e demais formas de adaptação de interfaces de usuário.

Pode-se observar a existência de diferentes ferramentas que utilizam a linguagem de descrição de interfaces de usuário UsiXML para oferecer suporte à criação de IU multiplataforma, principalmente na geração de interfaces de usuário concretas (CUI's) a partir de diversas formas de entrada. Dessa forma justifica-se a existência de aplicativos que trabalham exclusivamente com a renderização de IU's em UsiXML, como serão apresentados a seguir.

2.2.2.2 Renderizadores de Interfaces de Usuário

Renderização é o nome dado ao processo de mapear descrições de interfaces para interfaces concretas. Em relação ao *framework* de referência *Cameleon*, o termo renderização se refere ao processo de reificação que transforma uma definição concreta de interface (CUI) em uma interface de usuário final (FUI). A ferramenta que realiza o processo de renderização é denominada renderizador de interfaces.

Dentre os renderizadores descritos na literatura, o *QTKiXML* (DENIS, 2005) mapeia interfaces de usuário descritas em UsiXML para a linguagem Tcl-Tk. Como o ambiente de execução necessário para linguagem existe para diferentes plataformas, a interface obtida como saída do renderizador pode ser considerada multiplataforma.

Já o *FlashiXML* (BERGHE, 2004) também realiza a renderização de descrições de interface de usuário em UsiXML para interfaces finais. Neste caso, porém, a interface resultante é descrita em modo vetorial, podendo ser interpretada por qualquer plataforma equipada com *plug-ins* Flash ou *Scalable Vector Graphics* (SVG). No

FlashiXML, a interface de usuário pode ser redimensionada a qualquer momento para se adaptar a limitações impostas pela plataforma sendo utilizada.

Outro trabalho similar propõe o InterpiXML (OCAL, 2004), que realiza a renderização de descrições concretas de interface de usuário em UsiXML utilizando o *toolkit* de componentes visuais *Swing*, para a linguagem Java.

A ferramenta UsiXML2OpenLaszlo (USIXML2OPENLASZLO) também atua na renderização de interfaces de usuário descritas em UsiXML, transformando-as em descrições na linguagem *OpenLaszlo* (OPENLASZLO), a qual por sua vez pode ser transformada em uma interface descrita em *Adobe Flash* ou *Dynamic HTML*.

Já com a utilização de interfaces descritas em UIML, a ferramenta Uiml.NET (LUYTEN, 2004) permite a renderização de interfaces para a diferentes conjuntos de elementos da plataforma .NET, como *Gtk#*, *System.Windows.Forms*, *System.Windows.Forms* para o *Compact .Net Framework*, e uma pequena parte do *Wx.Net*.

Também utilizando a mesma linguagem, o aplicativo TIDE (ALI, 2002) (*Tranformation-based Integrated Development Environment*) realiza a renderização de descrições de interfaces UIML para a linguagem de programação Java.

Em relação a linguagem TeresaXML, o ambiente TERESA (*Transformation Environment for InteRactivE Systems representAtions*) (MORI, 2003) permite a criação de interfaces de usuário para múltiplas plataformas de computação a partir das descrições TeresaXML. Essa ambiente permite uma série de transformações semi-automáticas que tem como objetivo auxiliar o desenvolvedor de sistemas interativos, abrangendo todos os níveis de abstração presentes no *framework* de referência *Cameleon*, e também permitindo a geração de interfaces multimodais.

O projeto MONA (*Mobile multimOdal Next-generation Applications*) (SIMON, 2005) investigou a interação multimodal em dispositivos móveis. O objetivo do projeto foi a criação de um método mais amigável de criação de interfaces de usuário para múltiplas plataformas utilizando a técnica *single-authoring*. O projeto desenvolveu o MONA *Presentation Server*, o qual transforma descrições de interface de usuário gráficas e multimodais, adaptando-as dinamicamente para diferentes dispositivos. como telefones que utilizam a tecnologia WAP (*Wireless Application* Protocol), *smartphones* baseados na linguagem *Symbian* ou PDA's PocketPC.

Com a apresentação dos renderizadores de interface de usuário existentes, pode-se notar a existência de diversas propostas que trabalham com a linguagem UsiXML. No entanto, a maioria trabalha com linguagens que não são muito utilizadas atualmente no desenvolvimento de *software*, principalmente se tratando de *software* corporativo. No caso em que uma linguagem de maior utilização é abordada, como no *InterpiXML*, onde o renderizador gera interfaces de usuário na plataforma Java, a ferramenta trabalha apenas com uma plataforma, limitando dessa forma a sua utilização.

Além disso, todos os renderizadores apresentados têm como principal preocupação a geração das interfaces de usuário finais. Apesar de essa ser uma abordagem válida, tendo em vista que se trata da principal função do renderizador, as propostas deixam em segundo plano a conexão das interfaces geradas com a lógica da aplicação. Nas vezes em que esse problema é tratado, a ferramenta limita-se a permitir a conexão com aplicações desenvolvidas em uma linguagem de programação específica, o que mais uma vez limita a sua utilização em um ambiente de desenvolvimento real.

3 RENDERXML: RENDERIZADOR DE INTERFACES DE USUÁRIO PARA MÚLTIPLAS PLATAFORMAS BASEADO EM USIXML

Como vimos no capítulo anterior, a criação de interfaces de usuário para múltiplas plataformas envolve alguns passos. Um desses passos é a renderização de descrições de interfaces concretas para interfaces de usuário finais. Para fornecer suporte computacional a esse passo, faz-se necessária uma ferramenta que seja capaz de transformar a descrição concreta de uma interface na linguagem UsiXML em uma interface de usuário final, capaz de ser executada na plataforma-alvo da aplicação.

Essa ferramenta é denominada de *renderizador de interfaces de usuário* no contexto desse trabalho, e deve ser responsável por a) instanciar os componentes concretos necessários, b) tratar os eventos que acontecem na interface e c) conectar a mesma à lógica de aplicação. Cada uma destas ações será melhor explicada nesse capítulo.

Este capítulo tem como objetivo apresentar a ferramenta RenderXML ao leitor. Assim, a seção 3.1 mostra os requisitos e objetivos que deveriam ser atendidos pela ferramenta, sendo que a arquitetura proposta é apresentada na seção 3.2 e as decisões de projeto tomadas junto com os detalhes de implementação são explicados na seção 3.3.

3.1 Requisitos e Objetivos do RenderXML

De forma a projetar uma ferramenta de renderização que seja similar às ferramentas mais recentes existentes atualmente (estado da arte), mas que possibilite a obtenção de resultados em um curto espaço de tempo, foram estabelecidos objetivos primários e secundários para o renderizador a ser desenvolvido.

- Objetivos Primários
    a. Ser capaz de instanciar interfaces de usuário concretas descritas na linguagem UsiXML na plataforma Java.
    b. Ser capaz de instanciar interfaces de usuário tanto para plataformas *Desktop* quanto para *PDA* (*Personal Digital Assistant*) e *Web*.
    c. Poder ser utilizado no desenvolvimento de aplicativos de livros eletrônicos falados, contexto no qual este trabalho está inserido.
- Objetivos Secundários

a. Possuir uma arquitetura extensível, permitindo a adaptação do projeto desenvolvido para outros contextos de uso.

b. Possuir uma arquitetura que permita a utilização do renderizador com lógicas de aplicação não desenvolvidas em Java.

Em relação aos objetivos citados acima, a linguagem Java e as plataformas-alvo atendidas pela primeira versão da ferramenta foram selecionadas devido aos requisitos do projeto LIFAPOR, onde a mesma está inserida. Nesse projeto, o RenderXML tem como função possibilitar a criação de livros eletrônicos falados para múltiplas plataformas, e as plataformas *desktop*, PDA e *Web* foram escolhidas como as principais prioridades a serem desenvolvidas.

Dentre as linguagens de descrição de interfaces de usuário existentes, a UsiXML foi escolhida para a realização desse trabalho por ser uma linguagem de descrição de interfaces de usuário baseada no *framework* de referência *Cameleon*, possuindo diferentes níveis de abstração. Dessa forma, a linguagem se torna mais completa, permitindo a utilização de diferentes abordagens no desenvolvimento de IU's. Além de suas características técnicas, a UsiXML também possui seu material e informações disponíveis sem custo, e uma comunidade de desenvolvedores e pesquisadores aberta e ativa atualmente, organizada em torno do UsiXML *Consortium* (USIXML), que oferece um efetivo suporte e um rico ambiente de discussão entre seus usuários.

Além disso, apesar deste trabalho visar à implementação de uma ferramenta funcional, a versão inicial da ferramenta não é capaz de implementar toda a especificação da linguagem UsiXML. Considerando-se também que a UsiXML é um trabalho em andamento, sujeito à mudanças, estabeleceu-se o objetivo de desenvolver o RenderXML com uma arquitetura extensível, a qual pudesse ser adaptada à adição de novos componentes e à mudanças na especificação da linguagem.

O RenderXML possui como principal função auxiliar o desenvolvedor de interfaces de usuário, podendo ser utilizado no processo de criação de interfaces em três diferentes situações: prototipação de interfaces de usuário, desenvolvimento de interfaces para novos aplicativos e adaptação de aplicativos existentes para um ambiente de execução multiplataforma.

De forma a possibilitar sua utilização nessas situações, também foi estabelecida como foco do trabalho a conexão da interface renderizada com a lógica de aplicação, visando-se a utilização da ferramenta com lógicas de aplicação desenvolvidas em múltiplas linguagens de programação. Entendeu-se que essa conexão, embora muitas vezes não abordada por trabalhos relacionados, é de extrema importância para a real utilização da ferramenta, uma vez que, limitando-se a utilização das interfaces geradas com um tipo de linguagem de programação, limita-se também o escopo de utilização do renderizador.

Dessa maneira, o RenderXML torna-se apto a ser utilizado nas três situações descritas anteriormente. Como a UsiXML permite a descrição de todos os elementos necessários de uma interface de usuário, o RenderXML pode ser utilizado tanto na prototipação como no desenvolvimento de interfaces de usuário finais. Nessa situação, torna-se extremamente importante a possibilidade de desenvolver interfaces para múltiplas plataformas utilizando apenas uma linguagem, liberando o desenvolvedor da necessidade de dominar diversas tecnologias. Utilizando o RenderXML, o desenvolvedor necessita dominar apenas a UsiXML.

Além disso, a possibilidade de conexão com múltiplas linguagens de programação permite que aplicativos de *software* desenvolvidos em diferentes linguagens possam ter suas interfaces de usuário descritas em UsiXML. A independência em relação à linguagem de programação proporcionada pelo RenderXML torna a ferramenta também útil na adaptação de sistema existentes para um ambiente de execução multiplataforma. Isso pode ser obtido com a recriação das interfaces de usuário de um aplicativo usando UsiXML, e com a conexão das mesmas à lógica de aplicação já existente.

No entanto, apesar de o RenderXML poder ser utilizado de forma independente de outras ferramentas, mapeando interfaces de usuário concretas em UsiXML para interfaces de usuário finais, o aplicativo está inserido em um contexto maior, baseado no *framework* de referência *Cameleon*, o qual pode ser observado na Figura 3.1.

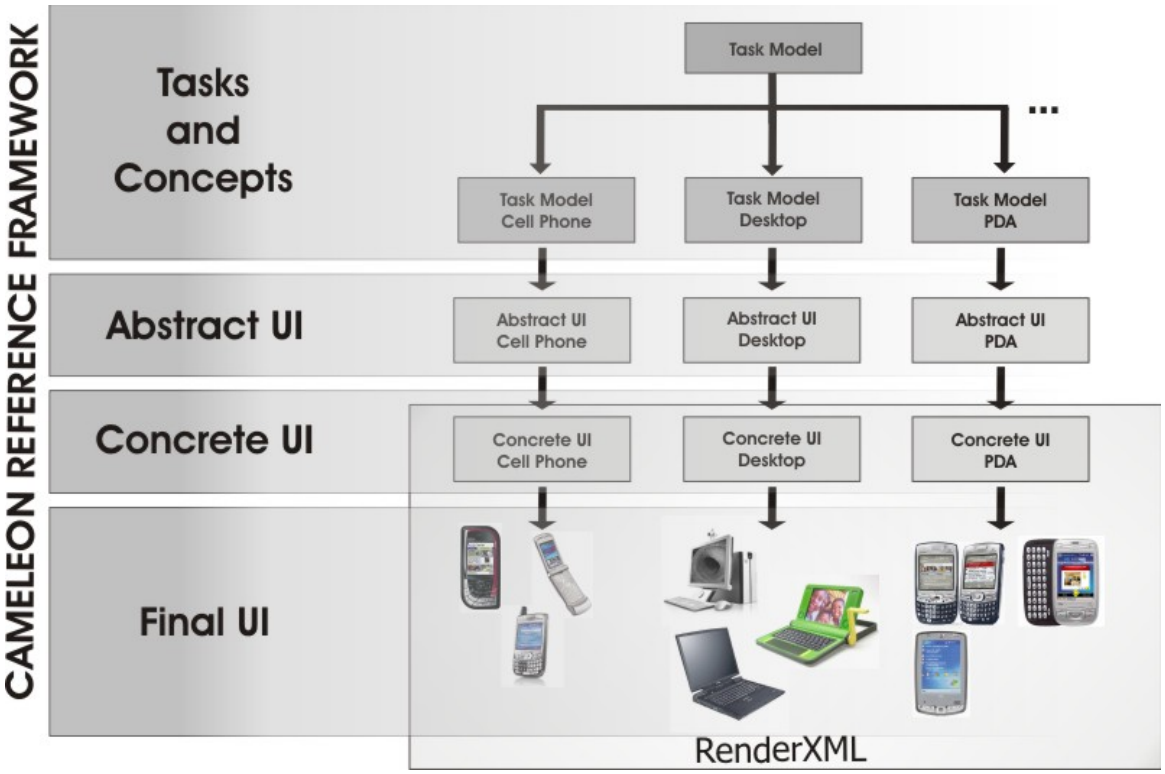

Figura 3.1: Ciclo de desenvolvimento de interfaces multiplataforma.

Conforme é mostrado na Figura 3.1, o RenderXML foi projetado para ser utilizado apenas na última etapa desse processo, mapeando interfaces de usuário concretas descritas em UsiXML para interfaces de usuário finais em um dispositivo específico.

Para que se possa realizar o processo completo, do modelo de tarefas para até a interface de usuário final, o RenderXML pode ser utilizado em conjunto com outras ferramentas desenvolvidas para trabalhar com a linguagem UsiXML, as quais foram apresentadas anteriormente neste documento.

Deve ser ressaltado que, por trabalhar especificamente com o último nível de mapeamento do processo, o RenderXML é uma ferramenta de renderização, e não um aplicativo para ser utilizado no desenho de interfaces de usuário. De fato, o RenderXML não tem como objetivo guiar as escolhas do *designer* entre alternativas de projeto.

Claramente, o RenderXML não verifica as decisões do *designer* e não avalia ou identifica aspectos de usabilidade presentes na interface de usuário. Como foi mostrado na Figura 3.1, esse tipo de problema deve ser resolvido em etapas anteriores do processo de mapeamento, de forma manual ou automática, com a utilização de outras ferramentas que trabalham com UsiXML. Também deve ser enfatizado que, no caso da adaptação de um sistema existente, o RenderXML não realiza nenhum tipo de engenharia reversa das interfaces de usuário existentes. Isso também pode ser realizado com ferramentas que visam ao processo especifico de engenharia reversa, algumas das quais foram apresentadas anteriormente nesse trabalho.

3.2 Arquitetura Proposta

De forma a atender aos requisitos apresentados na seção anterior, a seguinte arquitetura foi proposta para a ferramenta RenderXML, conforme mostrado na Figura 3.2. Essa arquitetura contempla tanto a etapa de renderização da interface de usuário, representada por linhas contínuas na Figura 3.2, como a conexão da mesma com a lógica de aplicação, representada por linhas tracejadas na mesma figura.

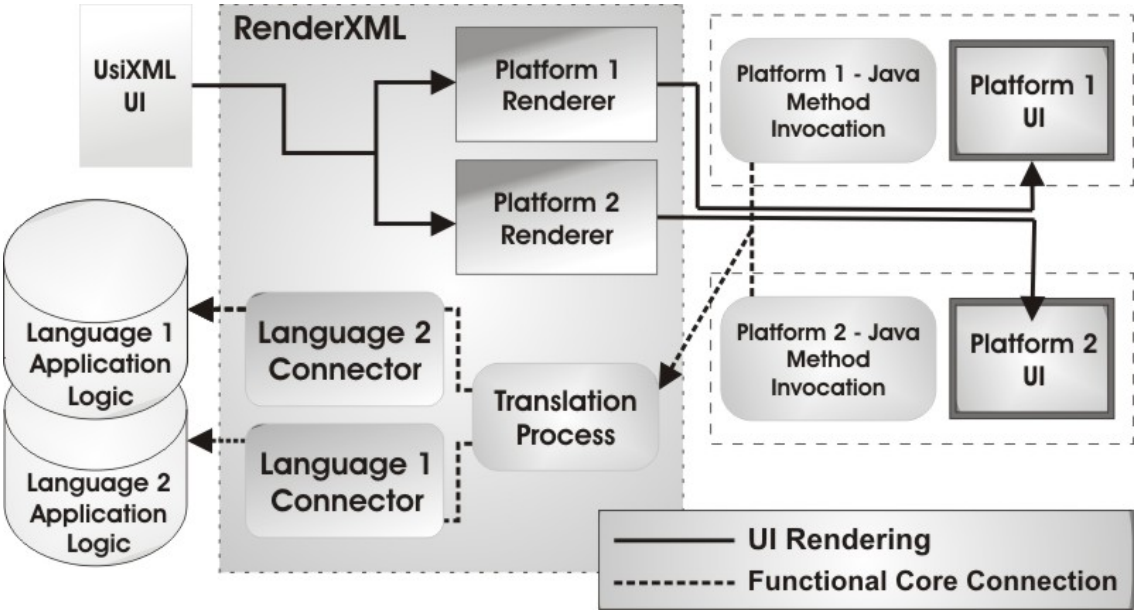

Figura 3.2: Arquitetura do RenderXML

Para realizar a renderização da interface de usuário, o RenderXML recebe como entrada uma descrição de interface de usuário concreta em UsiXML (*UsiXML UI* na Figura 3.2), a qual é tratada pelo renderizador da plataforma-alvo sendo utilizado (*Platform 1 Renderer* na Figura 3.2). Esse renderizador é responsável por, a partir da descrição dos componentes necessários da interface, realizar a instanciação desses componentes na plataforma-alvo. Após a renderização de cada componente, caso existam chamadas de métodos da lógica de aplicação, essas chamadas são direcionadas para um processo de tradução (*Translation Process* na Figura 3.2).

Na conexão da interface de usuário com a lógica de aplicação correspondente, o processo de tradução recebe as invocações de métodos realizadas e as traduz para um formato independente de linguagem de programação. Essa descrição é repassada para um conector específico da linguagem de programação sendo utilizada (*Language*

Connector na Figura 3.2), o qual traduz novamente as chamadas para uma linguagem específica, realizando a invocação do método. No caso da existência de um valor de retorno, o caminho é inverso é realizado, até o valor chegar novamente à interface de usuário.

3.3 Implementação

De maneira a possibilitar a renderização de interfaces UsiXML para as três plataformas desejadas, três renderizadores diferentes foram implementados, cada um específico para uma plataforma alvo: RenderXML4Desktop, RenderXML4CDC e RenderXML4Web, sendo que três ferramentas foram desenvolvidas utilizando a linguagem de programação Java.

O RenderXML4Desktop foi desenvolvido utilizando-se o *Java Standard Edition* (JSE) para permitir a renderização de interfaces para computadores de mesa (*desktop*), e utiliza a biblioteca de componentes *Swing* para instanciar os componentes da interface de usuário descrita.

Já o RenderXML4CDC possui como objetivo realizar a instanciação de interfaces de usuário para dispositivos móveis, como computadores de mão e telefones celulares. Para isso, a ferramenta foi desenvolvida utilizando o *Java Micro Edition* (JME), através do perfil *Connected Device Configuration* (CDC), o qual é a configuração com mais recursos existente no JME, e é implementada por *smartphones* e computadores de mão. Nessa versão do RenderXML, a biblioteca gráfica utilizada para instanciação dos componentes de interface também é a biblioteca *Swing*, porém uma versão adaptada para desenvolvimento no perfil CDC.

Em relação ao RenderXML4Web, a principal diferença em relação às duas outras versões da ferramenta é a forma de instanciação da interface. Enquanto nas versões *desktop* e móvel a interface é renderizada em tempo de execução, na versão para aplicativos *Web*, tendo em vista a necessidade de um servidor onde a aplicação possa ser executada, foi decidido que seria gerado o código-fonte da interface, o qual pode posteriormente ser executado em um servidor *Web*. Para isso, a ferramenta gera uma interface utilizando o *Java Server Faces* (JSF), o qual é um *framework* baseado em Java para o desenvolvimento de aplicações para a Internet.

Apesar de diferirem em relação à tecnologia utilizada, as três versões do RenderXML foram implementadas seguindo os mesmos princípios arquiteturais e utilizando os mesmos padrões de projeto para codificação. Dessa maneira, os detalhes sobre a implementação serão apresentados a seguir utilizando-se como exemplo o RenderXML4Desktop, porém podem ser aplicados a qualquer uma das outras duas versões existentes.

Para esclarecer a implementação da ferramenta RenderXML, serão detalhados as principais tarefas que são realizadas durante a renderização da interface de usuário, de acordo com o processo apresentado na próxima seção.

3.3.1 Processo de Renderização

O modelo de interface de usuário concreta em UsiXML é descrito através de uma estrutura em árvore, com os componentes organizados em ramos de forma a descrever a sua hierarquização na interface. Dessa maneira, para realizar a renderização da interface de usuário descrita em UsiXML, o RenderXML adota uma estratégia de percurso em profundidade da estrutura da interface. Sendo assim, após encontrar a descrição do

modelo de interface concreta, o RenderXML navega por entre os componentes descritos, instanciando cada componente e todos os seus descendentes.

Na renderização de cada componente, o RenderXML realiza o processo descrito na Figura 3.3. Nesse processo, como será detalhado nas subseções a seguir, para cada componente existente na descrição da interface de usuário em UsiXML, deve ser selecionado um renderizador, o qual é responsável pela criação do componente e definição de seus atributos. Com o componente criado, caso seja necessário, o mesmo tem o seu conteúdo obtido e suas rotinas de comportamento implementadas.

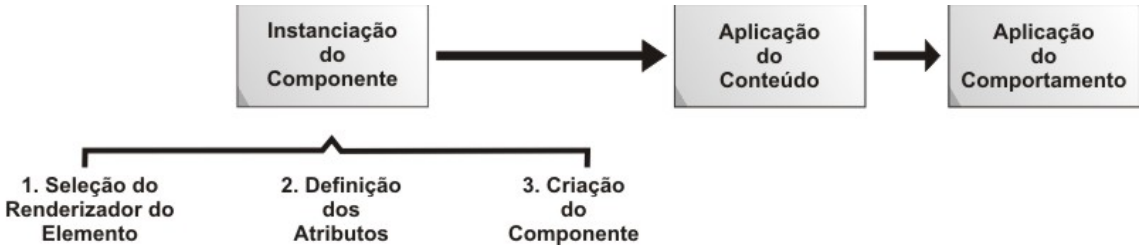

Figura 3.3: Processo de Renderização dos Componentes da Interface de Usuário Implementado pelo RenderXML.

As etapas mostradas na Figura 3.3 serão detalhadas nas subseções a seguir.

3.3.2 Instanciação do Componente

A instanciação de componentes nesse trabalho é composta pela criação dos componentes descritos na interface de usuário e a definição dos seus atributos. Dessa forma, esse processo deve partir de um modelo de interface de usuário concreta (*cuiModel*), como é mostrado na Figura 3.4, e obter como resultado final a interface executando na plataforma-alvo da aplicação, porém ainda sem o seu conteúdo e comportamento definido.

```
<cuiModel id="5-cui_20" name="5-cui">
    <window id="window_1" name="window_1" width="235" height="200">
      <flowBox id="flow_box_2" name="flow_box_2" alignment="left">
        <inputText id="input_1"
            name="input_1" isVisible="true"
          isEnabled="true" textColor="#000000"
          maxLength="50" numberOfColumns="15" isEditable="true"/>
        <button id="button_1"
                  content="/uiModel/resourceModel/cioRef[@cioId='button_1']
                           /resource/@content"
          name="button_1" isVisible="true"
          isEnabled="true" textColor="#000000">
            <behavior>

              ...

            </behavior>
        </button>

      ...

</cuiModel>
```

Figura 3.4: Exemplo de modelo de interface de usuário concreta em UsiXML.

Conforme mostra a Figura 3.4, o modelo de interface de usuário concreta possui a informação de todos os componentes que estão presentes na interface, além das informações dos seus atributos. Se analisarmos a figura acima, nesse trecho de código

UsiXML é declarada uma janela (*window*), a qual contém um campo de entrada de texto (*inputText*) e um botão (*button*).

De forma a realizar a instanciação dos componentes da interface, a arquitetura mostrada no diagrama de classes na Figura 3.5 foi implementada.

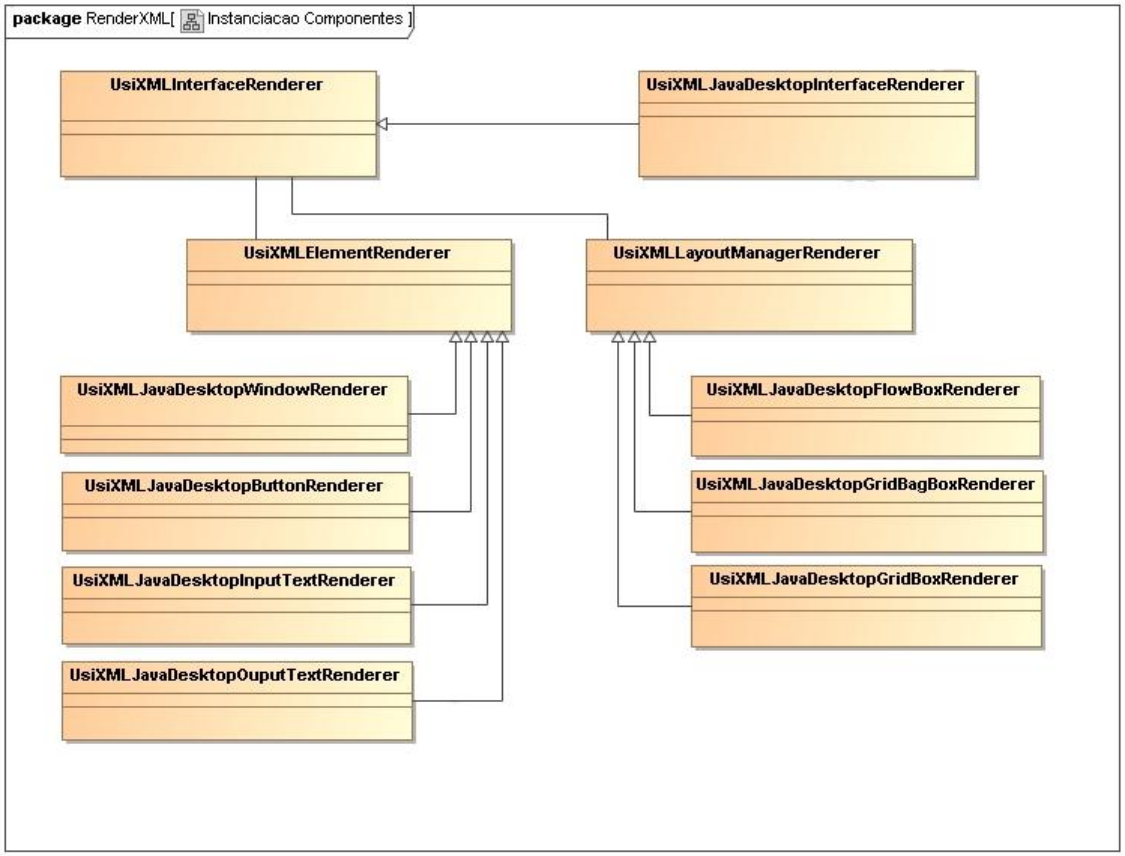

Figura 3.5: Diagrama de classes da arquitetura para instanciação de componentes do RenderXML.

Nessa arquitetura, os componentes a serem instanciados foram separados em duas categorias diferentes: elementos gráficos simples, os quais são tratados por classes descendentes da superclasse *UsiXMLElementRenderer*, e elementos representando gerenciadores de *layout*, os quais são tratados por classes descendentes da superclasse *UsiXMLLayoutManagerRenderer*. Essa diferenciação deve-se à forma pela qual interfaces de usuário são desenvolvidas utilizando a biblioteca *Swing*, exigindo uma forma específica de renderização para cada uma dessas categorias. Isso resulta do fato que os gerenciadores de *layout* devem ser adicionados ao *content pane*, o qual é o painel principal da interface de usuário, assim como os componentes que estão inseridos hierarquicamente abaixo do gerenciador de *layout*. Já no caso de elementos simples, caso existam elementos hierarquicamente abaixo, esses podem ser adicionados diretamente no elemento de nível superior.

Conforme explicado anteriormente e mostrado na Figura 3.5, para permitir o desenvolvimento evolutivo e extensível do renderizador, onde novos elementos pudessem ser adicionados de forma simples, cada tipo de componente a ser instanciado

deve possuir uma classe de renderização específica, descendente das classes *UsiXMLElementRenderer* ou *UsiXMLLayoutManagerRenderer*.

Dentro de cada um desses grupos, a classe a ser utilizada na renderização de um determinado componente existente na descrição UsiXML é selecionada através do padrão GoF *Chain of Responsability* (GAMMA, 1995). Nesse padrão, cada classe deve possuir a informação necessária para saber qual elemento deve tratar, e caso o elemento selecionado não seja de sua responsabilidade, deve redirecionar a requisição para o próximo elemento da cadeia de responsabilidade. Essa operação é mostrada no trecho de código presente na Figura 3.6.

```
@Override
public String getElementName() {
   return "button";
}

@Override
public UsiXMLElementRenderer getNextRenderer() {
   return new UsiXMLJavaDesktopInputTextRenderer(
      (UsiXMLJavaDesktopInterfaceRenderer)this.usiXMLInterfaceRenderer);
}
```

Figura 3.6: Trecho de Código do RenderXML implementando o padrão GoF *Chain of Responsability*.

Nesse trecho de código retirado da classe *UsiXMLJavaDesktopButtonRenderer*, a qual é responsável pela renderização do componente *button*, são apresentadas duas funções. A primeira, *getElementName* retorna o nome do elemento que é tratado por essa classe, no caso o elemento *button*. A segunda função retorna o próximo renderizador da cadeia (*UsiXMLJavaDesktopInputTextRenderer*), caso o elemento sendo tratado no momento não seja do tipo *button*.

Sendo assim, caso seja necessário adicionar uma classe de renderização que trate um novo elemento UsiXML na ferramenta, basta criar essa classe e modificar o atual último elemento da fila, fazendo com que o mesmo redirecione as chamadas para a classe de renderização inserida.

Após a seleção da classe de renderização correta, o processo segue com a instanciação do componente e definição de seus atributos, de acordo com a especificação da interface em UsiXML, conforme mostra o trecho de código da Figura 3.7.

```
@Override
    public Object processRendering(Element element) throws ParsingErrorException {

        String id = element.getAttributeValue("id");
        String name = element.getAttributeValue("name");
        String isEnabled = element.getAttributeValue("isEnabled");
        String isVisible = element.getAttributeValue("isVisible");

        PrimitiveElementParser parser = new PrimitiveElementParser();

        JButton button = new JButton();

        this.usiXMLInterfaceRenderer.addRenderedComponent(id,button);

        button.setPreferredSize(new Dimension(50,21));
        button.setName(name);
        button.setEnabled(parser.parseBoolean(isEnabled));
        button.setVisible(parser.parseBoolean(isVisible));
```

Figura 3.7: Trecho de código executado para instanciação de um componente no RenderXML.

No trecho de código exibido na Figura 3.7, o qual também foi retirado da classe *UsiXMLJavaDesktopButtonRenderer*, os atributos do botão são obtidos e então um componente da classe *JButton* é instanciado, tendo os seus atributos configurados após a instanciação. Com isso, o processo de instanciação do componente é finalizado, e então o elemento instanciado pode ter seu conteúdo e comportamento configurados, conforme será apresentado nas próximas subseções.

Por ser ainda um protótipo, a versão atual do RenderXML não implementa a especificação completa da linguagem UsiXML. De fato, apenas os elementos necessários para a criação dos estudos de caso desejados foram adicionados. Na Tabela 3.1 são apresentados os elementos suportados (Elemento UsiXML), assim como a classe Java utilizada em seu mapeamento (Componente Java) e os atributos considerados na instanciação do elemento (atributos).

Tabela 3.1: Componentes e atributos suportados pela versão atual do RenderXML.

Elemento UsiXML	*Componente Java*	*Atributos*	*Descrição*
		Componentes Gráficos	
window	javax.swing.JFrame	*id*	id
		name	nome
		width	largura
		heigth	altura
		bgColor	cor de fundo
button	javax.swing.JButton	*id*	id do componente
		name	nome do componente
		width	largura
		heigth	altura
		isEnabled	habilitado (sim/não)
		isVisible	visível (sim/não)
inputText	javax.swing.JTextField	*id*	id do componente
		name	nome do componente
		width	largura
		heigth	altura
		isEnabled	habilitado (sim/não)
		isVisible	visível (sim/não)
		isEditable	editável (sim/não)
outputText	javax.swing.JLabel	*id*	id do componente
		name	nome do componente
		width	largura
		heigth	altura
		isEnabled	habilitado (sim/não)
		isVisible	visível (sim/não)
		Gerenciadores de Layout	
flowBox	java.awt.FlowLayout	*id*	id do componente
		alignment	alinhamento
gridBagBox	java.awt.GridBagLayout	*id*	id do componente
		gridx	posição no eixo x
		gridy	posição no eixo y
		gridwidth	largura em número de células
		gridheigth	altura em número de células
		weightx	peso da largura do componente em relação aos demais, para distribuição de espaço extra
		weighty	peso da altura do componente em relação aos demais, para distribuição de espaço extra
		insets	margem do componente em relação à celula
		fill	forma
gridBox	java.awt.GridLayout	*id*	id do componente
		rows	número de linhas
		cols	número de colunas

3.3.3 Aplicação do Conteúdo

Conforme foi mostrado anteriormente neste trabalho, a UsiXML define um modelo de recursos, o qual contém o conteúdo que deve ser apresentado na interface de usuário, e permite a existência de uma separação entre a especificação dos componentes da interface e o conteúdo que os mesmos devem possuir. Essa separação é extremamente útil no processo de desenvolvimento de interfaces de usuário, possibilitando a criação de interfaces que possam se adaptar a diferentes contextos, como no caso da internacionalização (p.ex. mudança da língua) em que a interface deve ser apresentada.

Dessa forma, o RenderXML deve ser capaz de buscar e configurar o conteúdo de cada elemento no modelo de recursos da interface, conforme mostrado no trecho de código presente na Figura 3.8. Nesse trecho de código é realizada a declaração de um botão (*button*), o qual possui seu conteúdo definido no modelo de recursos

(*resourceModel*), sendo que a ligação entre os dois elementos é realizada através do atributo *content* da declaração do botão.

```
<uiModel>

    ...

 <cuiModel id="5-cui_20" name="5-cui">
    <window id="window_1" name="window_1" width="235" height="200">
      <flowBox id="flow_box_2" name="flow_box_2" alignment="left">
        ...
        <button id="button_1"
            content="/uiModel/resourceModel/cioRef[@cioId='button_1']
                /resource/@content"
            name="button_1" isVisible="true"
            isEnabled="true" textColor="#000000">
            <behavior>
              ...
            </behavior>
        </button>
      </flowBox>
    </window>
 </cuiModel>

    ...

 <resourceModel id="calc_resource" name="calc_resource">
      <cioRef cioId="button_1">
        <resource content="1" contextId="context-en_US"/>
      </cioRef>

  ...

 </resourceModel>
</uiModel>
```

Figura 3.8: Trecho de código UsiXML contendo a declaração de um elemento *button* e seu respectivo conteúdo.

Como a inserção de conteúdo é feita da mesma maneira em todos os componentes de interface, variando apenas o método que deve ser chamado para a inserção do conteúdo, essa funcionalidade foi implementada no RenderXML na classe *UsiXMLElementRenderer*, a qual é superclasse dos renderizadores de todos os componentes gráficos, o que é apresentado no diagrama de seqüência da Figura 3.9.

Observando a Figura 3.9, na chamada do método que realiza a instanciação do componente gráfico (Passo 1 na Figura 3.9), a única configuração que o renderizador específico de cada componente deve realizar é a definição do método que deve ser invocado para a inserção do seu conteúdo. No momento da inserção (Passo 2 na Figura 3.9) é feita a requisição do método definido utilizando técnicas de reflexão.

Com a definição do método a ser chamado, a classe *ResourceManager* é acionada (Passo 3 na Figura 3.9), tendo como função buscar, através de uma consulta XPath, o conteúdo do componente sendo instanciado. Com o conteúdo definido, o método previamente configurado é invocado e o conteúdo inserido no componente da interface.

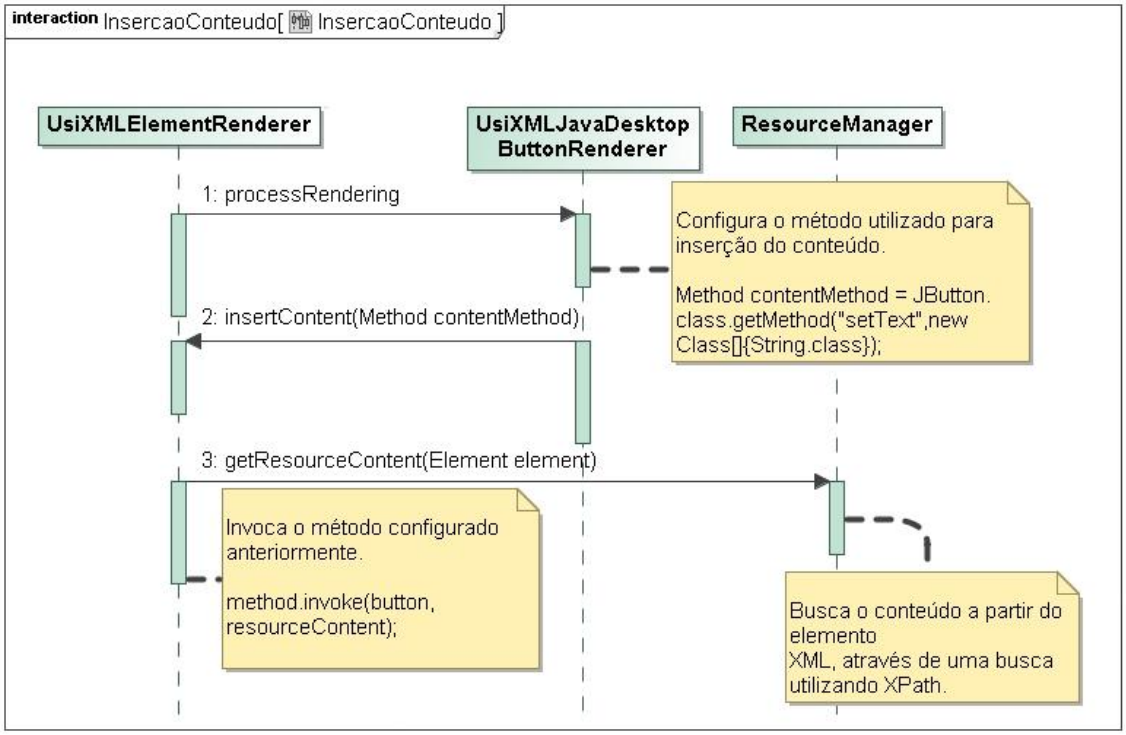

Figura 3.9: Diagrama de seqüência detalhando o processo de inserção de conteúdo.

Deve ser ressaltado que nessa versão do RenderXML, apenas conteúdos em formato texto são suportados. Para formatos multimídia, como áudio e vídeos, seria necessário adicionar na classe de renderização de cada componente o código necessário para inserir o conteúdo do formato desejado. Além disso, seria preciso inserir no renderizador as regras da linguagem UsiXML para conteúdos nesses formatos.

3.3.4 Aplicação do Comportamento

Para permitir a especificação do comportamento da interface, a UsiXML define o elemento *behavior*, o qual é mostrado no trecho de código da Figura 3.10. O elemento behavior é baseado em um modelo de evento-condição-ação, onde na ocorrência de um evento (*event* na Figura 3.10) na interface de usuário, uma condição é testada, e caso seja verdadeira, uma ou mais ações são executadas (*action* na Figura 3.10). Deve ser observado aqui que, embora a UsiXML defina a possibilidade de estabelecer-se uma condição a ser testada na execução do evento, essa funcionalidade não é suportada atualmente pelo RenderXML.

Na Figura 3.10 é mostrada a declaração de um botão (*button*), o qual possui um comportamento associado (*behavior*), composto por uma chamada de método (*method call*) e uma modificação na interface (*uiChange*).

```
<button id="button_1"
        content="/uiModel/resourceModel/cioRef[@cioId='button_1']/resource/@content"
      name="button_1" isVisible="true"
      isEnabled="true" textColor="#000000">
      <behavior>
        <event id="evt_b1" eventType="click" eventContext="button_1"/>
            <action id="act_b1" name="act_b1">
                    <methodCall methodName="buttonPressed">
                            <methodCallParam name="button" value="1"/>
                            <methodCallParam name="result" value="$valorDisplay"/>
               </methodCall>
                 <uiChange>
                    <changeElement elementId="input_1"
                                   attributeName="currentValue" value="$valorDisplay"/>
                 </uiChange>
            </action>
      </behavior>
</button>
```

Figura 3.10: Trecho de código com definição de comportamento dinâmico em UsiXML.

Conforme apresentado na figura acima (Figura 3.10), o RenderXML deve criar um observador para um evento especifico e conectá-lo a um determinado elemento gráfico sendo instanciado. Além disso, esse evento deve ser associado a uma lista de ações que serão executados em sua ocorrência.

De forma a implementar essa funcionalidade, uma abordagem em duas etapas foi utilizada: interpretação do evento e interpretação das ações.

Na interpretação dos eventos que ocorrem na interface, o RenderXML cria um tratador de eventos (*listener*) em Java específico para o evento que se deseja observar. No *framework Swing*, o *listener* é uma implementação do padrão de projeto GoF *Observer* (GAMMA, 1995), o qual permite que se adicionem dependências entre objetos de forma que, quando o estado do objeto observado mude, o objeto observador seja notificado.

Para realizar essa operação, o processo descrito no diagrama de seqüência da Figura 3.11 é implementado. Nesse processo, após a instanciação do componente gráfico, a classe de renderização (*UsiXMLElementRenderer* na Figura 3.11) solicita o tratamento do comportamento do elemento à classe *BehaviorManager* (Passo 1 na Figura 3.11), responsável por implementar o comportamento dinâmico da interface.

A partir do tipo de evento descrito, a classe *BehaviorManager* obtem três informações necessárias para a instanciação do tratador de eventos (Passos 2,3 e 4 na Figura 3.11) : o nome da interface a ser instanciada, o nome do método do tratador de eventos que deve ser executado na ocorrência do evento e o nome do método utilizado para adicionar o tratador de eventos no componente gráfico. Essas informações são obtidas da classe *EventType*, a qual tem apenas a função de definir os tipos de eventos suportados e retornar informações sobre os mesmos.

As informações são utilizadas então para criar o tratador de eventos necessário no componente gráfico sendo instanciado. Como mostra a Figura 3.11, as ações a serem executadas na ocorrência do evento, as quais são inseridas no código do tratador de eventos criado, são obtidas a partir da classe *ActionManager* (Passos 5 e 6 na Figura 3.11), em um processo que será descrito a seguir.

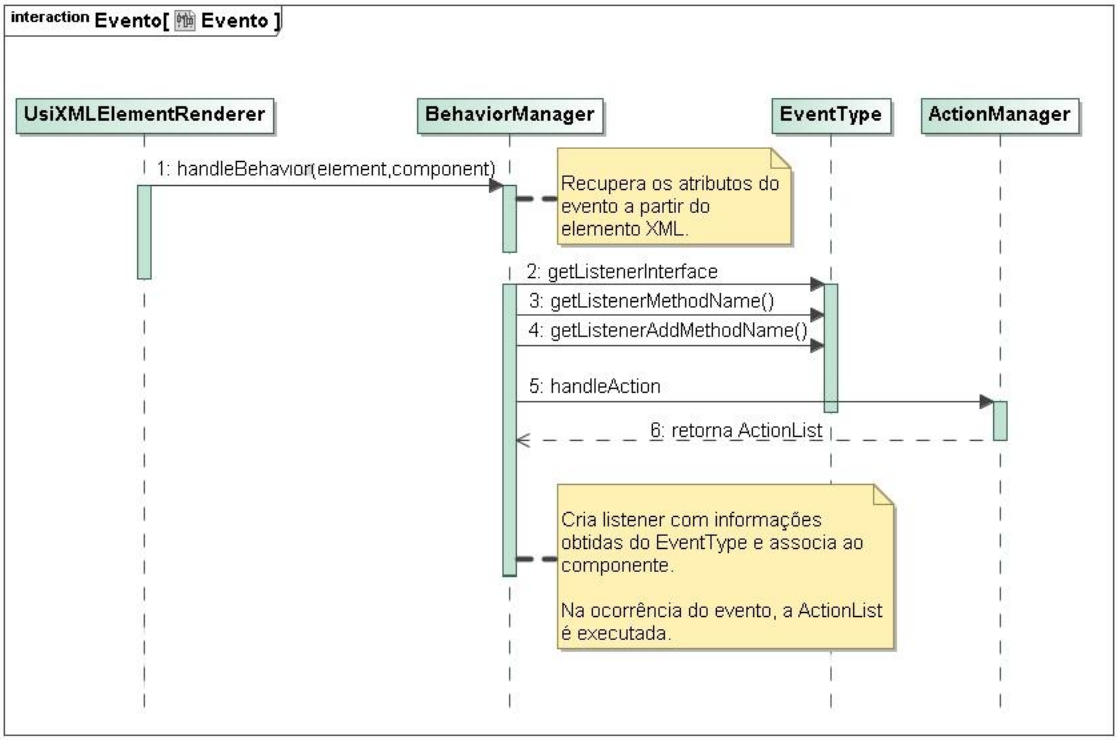

Figura 3.11: Diagrama de seqüência representando a criação dos tratadores de eventos no RenderXML.

Essa arquitetura foi escolhida porque permite uma separação clara de interesses entre a classe responsável pela instanciação do componente, descendentes da superclasse *UsiXMLElementRenderer*, a classe responsável pela interpretação do comportamento, *BehaviorManager*, e a classe responsável pela interpretação das ações da interface, *ActionManager*. Além disso, com toda a definição dos eventos suportados estando contida na classe *EventType*, a ferramenta ganha uma característica de fácil adaptação a novos tipos de eventos, já que basta modificar esta classe.

Em relação às ações que devem ser executadas na ocorrência de um evento, as mesmas podem ser de três tipos: mudança na interface (*uiChange*), transições (*triggerTransition*) e chamada de métodos externos (*methodCall*). Na instanciação do tratador de eventos, conforme apresentado anteriormente na Figura 3.11, as ações são obtidas da classe *ActionManager*, organizadas na classe *ActionList*.

Essa classe segue a arquitetura mostrada na Figura 3.12, sendo uma lista composta por elementos descendentes da classe *Action*, a qual representa uma ação a ser realizada. Dessa forma, existem três tipos de classes descendentes da classe *Action*: *UiChangeAction*, a qual representa uma ação do tipo *uiChange*, *TransitionAction* para transições na interface e *MethodCallAction*, representando uma ação do tipo *methodCall*.

A arquitetura apresentada na Figura 3.12 foi estabelecida já que permite criar uma série de ações que devem ser executadas em seqüência, e associá-las à mesma lista de ações. Dessa maneira, o tratador de eventos necessita apenas invocar o método *executeAll* da lista de ações associadas a ele, para que todas as ações necessárias sejam realizadas, o que facilita o processo de instanciação do tratador de eventos que foi abordado anteriormente nessa mesma seção.

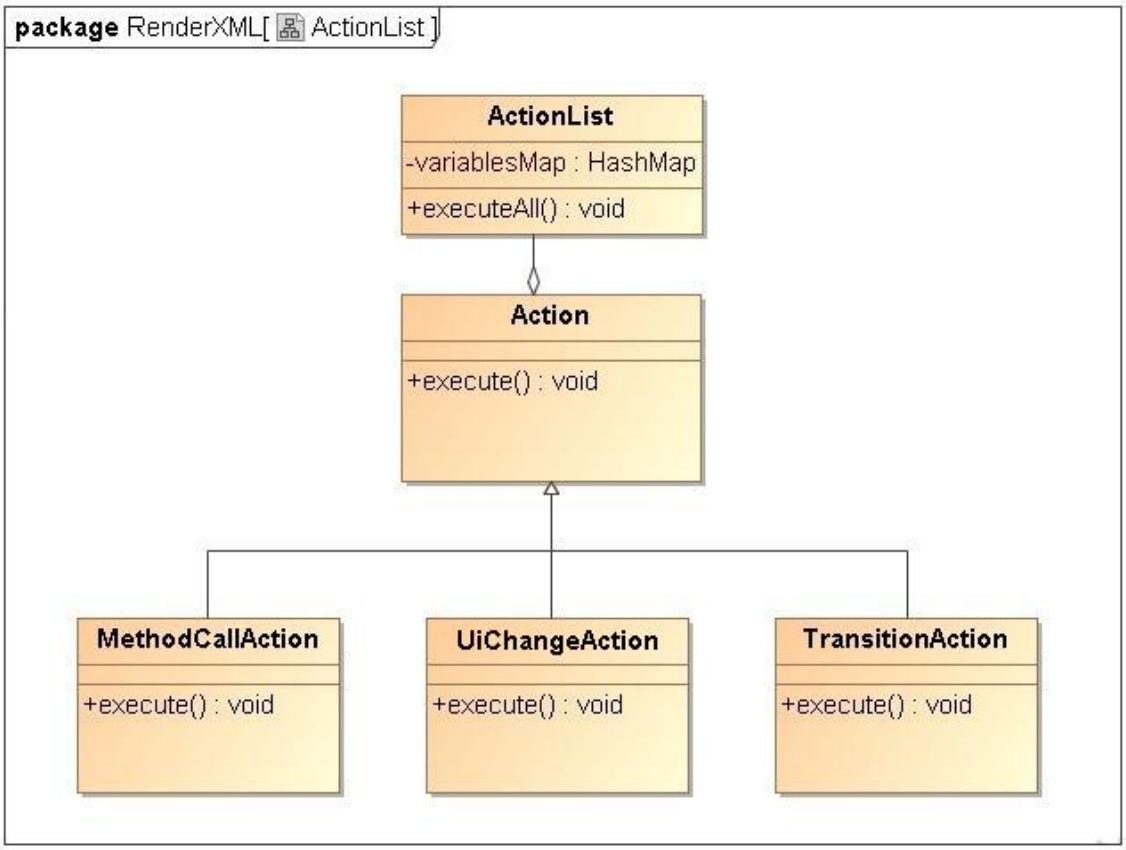

Figura 3.12: Diagrama de classes representando a arquitetura da lista de ações do RenderXML.

Em adição à possibilidade de execução das ações, a classe *ActionList* possui em sua estrutura um mapa de variáveis (*variablesMap* na Figura 3.12), o qual permite que todas as ações da lista compartilhem variáveis comuns.

Essa funcionalidade é útil no caso apresentado no trecho de código da Figura 3.13, o qual apresenta o exemplo de utilização de uma variável dentro de uma lista de ações. Esse exemplo mostra o comportamento de um botão quando pressionado pelo usuário, sendo que duas ações são executadas nesse momento: a chamada do método *buttonPressed* e a modificação do elemento com o atributo *elementId* igual a *input_1*. A existência de uma variável é necessária nesse caso, pois o valor retornado da invocação do método deve ser utilizado na modificação da interface, o que é realizado através da variável *$valorDisplay* na Figura 3.13.

```
<button id="button_1"
        content="/uiModel/resourceModel/cioRef[@cioId='button_1']/resource/@content"
        name="button_1" isVisible="true"
        isEnabled="true" textColor="#000000">
        <behavior>
           <event id="evt_b1" eventType="click" eventContext="button_1"/>
           <action id="act_b1" name="act_b1">
             <methodCall methodName="buttonPressed">
               <methodCallParam name="button" value="1"/>
               <methodCallParam name="result" value="$valorDisplay"/>
                       </methodCall>
             <uiChange>
               <changeElement elementId="input_1" attributeName="currentValue" value="$valorDisplay"/>
             </uiChange>
           </action>
      </behavior>
</button>
```

Figura 3.13: Exemplo de utilização de uma variável em UsiXML.

Dessa forma, na execução de uma lista de ações pelo RenderXML, os valores atribuídos a variáveis são armazenados no mapa de variáveis da lista de ações, podendo ser reutilizados por qualquer uma das ações seguintes.

Apesar de não estar descrito na especificação da UsiXML, foi assumido na implementação do RenderXML que as variáveis são identificadas pela existência do símbolo ‘$’ como primeiro caractere da palavra.

A forma de implementação de cada um dos tipos de ação definidos pela UsiXML é apresentada nas subseções a seguir.

3.3.4.1 Modificações na Interface

Para realizar modificações na interface de usuário renderizada, a UsiXML define o elemento *uiChange*, o qual pode ser composto por dois tipos de elementos: *triggerTransition* e *changeElement*.

O elemento *changeElement* define uma modificação de um determinado componente da interface, de forma que um atributo do mesmo possa ser alterado. O trecho de código da Figura 3.14 mostra um exemplo, onde o elemento identificado pelo *elementId* igual a *input1* tem o seu atributo com nome *currentValue* modificado para o valor 100.

```
<uiChange>
        <changeElement elementId="input_1"
                attributeName="currentValue" value="100"/>
</uiChange>
```

Figura 3.14: Exemplo de utilização do elemento *uiChange* em UsiXML.

A implementação dessa funcionalidade é realizada na forma descrita pelo diagrama de seqüência da Figura 3.15. Nesse caso, a classe *ActionManager* recebe a requisição para tratar um elemento do tipo *action* da classe *BehaviorManager* (Passo 1 na Figura 3.15), criando um novo objeto da classe *UiChangeAction* a partir desse elemento e associando-o à lista de ações que foi criada para o tratamento do evento (Passos 2 e 3 na Figura 3.15). Na criação dessa classe, o elemento é interpretado, e o componente gráfico relacionado a ele é buscado entre os componentes da interface de usuário.

Quando da execução da lista de ações, o método *execute* da classe *UiChangeAction* (Passo 5 na Figura 3.15) é invocado, e por sua vez, realiza a chamada do método *performUiChange* do respectivo componente gráfico (Passo 6 na Figura 3.15), passando

como parâmetro as informações do nome do atributo a ser modificado e o seu novo valor. Por fim, a classe responsável pela renderização do componente (*UsiXMLJavaDesktopInputTextRenderer* na Figura 3.15) realiza a atualização do mesmo.

Essa arquitetura mantém a facilidade de extensão da aplicação, uma vez que todo o código específico necessário para a atualização de cada componente gráfico está encapsulado na classe de renderização do componente, facilitando a adição de novos componentes na ferramenta.

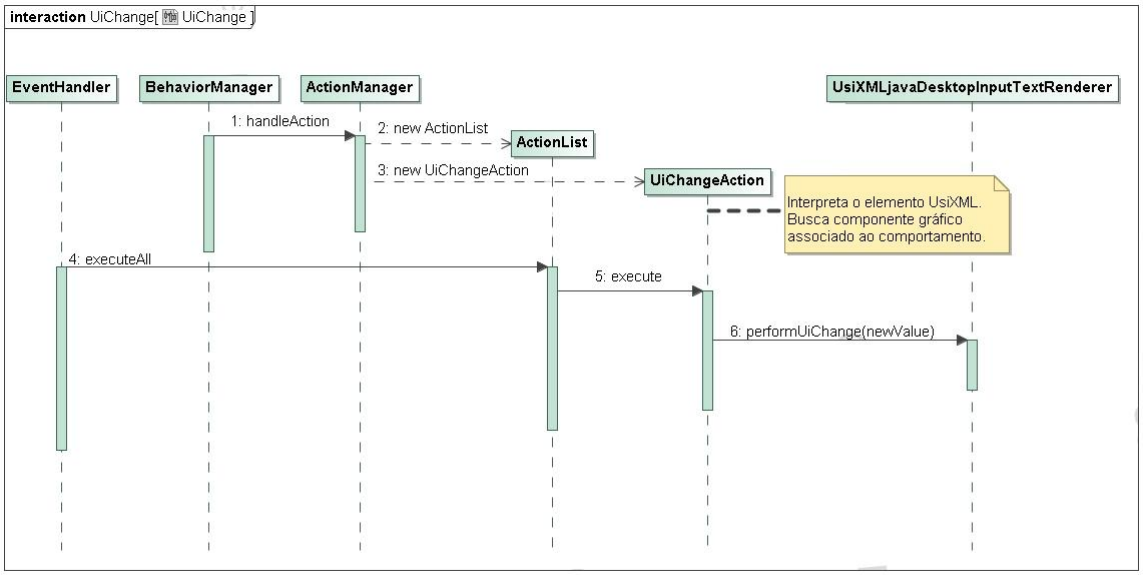

Figura 3.15: Diagrama de seqüência representando o processo de mudança de um componente da interface.

Para a implementação de transições na interface de usuário, a UsiXML define o elemento *triggerTransition*, conforme mostrado na Figura 3.16. Nesse exemplo, são definidas duas transições (*graphicalTransition*), a primeira, identificada pelo id *tr1*, define o fechamento de uma janela (*window_2*), e a segunda, identificada pelo id *tr2*, especifica a abertura de uma outra janela (*window_1*), caracterizando a transição.

```
<button id="button_voltar" content="/uiModel/resourceModel/cioRef[@cioId='button_voltar']/resource/@content" ... >
<behavior>
                <event id="evt_b2" eventType="click" eventContext="button_voltar"/>
                <action id="act_b2" name="act_b1">
                    <triggerTransition>
                        <graphicalTransition id="tr1" transitionType="close">
                            <source sourceId="button_voltar"/>
                            <target targetId="window_2"/>
                        </graphicalTransition>
                    </triggerTransition>
                    <triggerTransition>
                        <graphicalTransition id="tr2" transitionType="open">
                            <source sourceId="button_voltar"/>
                            <target targetId="window_1"/>
                        </graphicalTransition>
                    </triggerTransition>
                </action>
        </behavior>
</button>
```

Figura 3.16: Exemplo de utilização do elemento *triggerTransition* em UsiXML.

Para implementar essa funcionalidade, foi criado um processo análogo ao utilizado nas modificações da interface, descrito anteriormente. Esse processo é mostrado na Figura 3.17, e possui como diferença o fato de um objeto do tipo *TransitionAction* ser instanciado no passo 3, para interpretar o elemento que descreve a transição. No momento da execução da ação, a função *performTransition* é invocada na classe responsável pela renderização do componente gráfico, de forma a executar a transição desejada (Passo 6 na Figura 3.17).

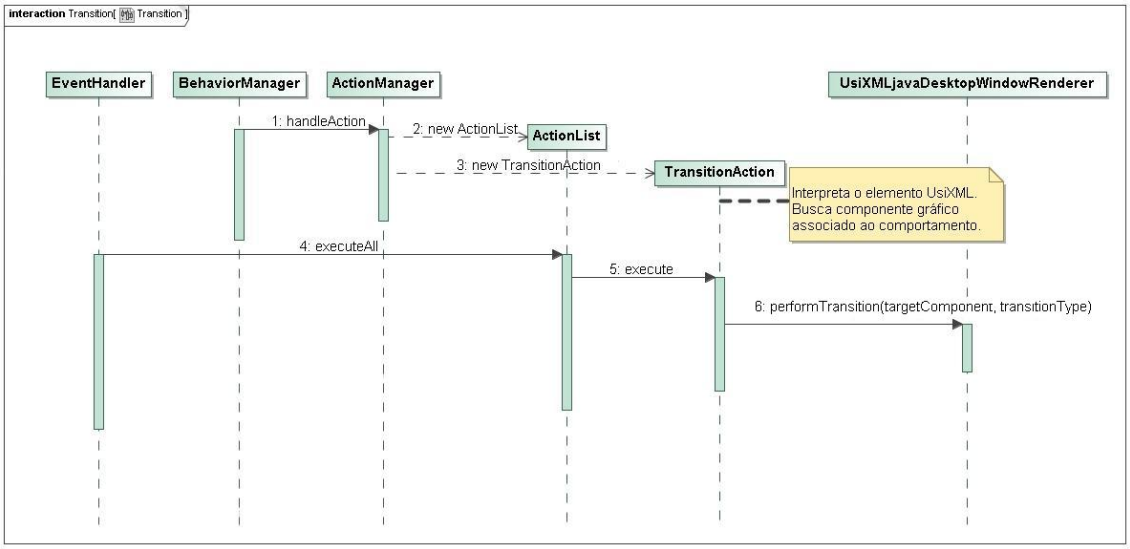

Figura 3.17: Diagrama de seqüência representando o processo de transição da interface de usuário.

3.3.4.2 Invocação de Métodos Externos

De forma a permitir a invocação de métodos externos à interface de usuário, a UsiXML define o elemento *methodCall*, o qual especifica o método a ser invocado, além dos parâmetros que devem ser utilizados na invocação do mesmo. Para que a chamada do método possa ser realizada, além de a mesma estar definida dentro do elemento *behavior* associado a algum componente, o método deve estar descrito no modelo de domínio da interface de usuário, conforme é mostrado na Figura 3.18.

```
<behavior>
        <event id="evt_b1" eventType="click" eventContext="button_1"/>
        <action id="act_b1" name="act_b1">
                <methodCall methodName="buttonPressed">
                        <methodCallParam name="button" value="1"/>
                        <methodCallParam name="result" value="$valorDisplay"/>
                </methodCall>
        </action>
</behavior>

....

<domainModel id="domain1" name="domain1">
        <domainClass id="calculator_logic" name="br.inf.ufrgs.calc.logic.CalculatorLogic">
                <method id="buttonPressed" name="buttonPressed">
                        <param dataType="integer" name="button"
                                paramType="input" passingType="byVal"/>
                        <param dataType="string" name="result"
                                paramType="output" passingType="byVal"/>
                </method>
        </domainClass>
</domainModel>
```

Figura 3.18: Exemplo de invocação de método em UsiXML.

No trecho de código mostrado na Figura 3.18, o método *buttonPressed* é invocado, passando um parâmetro com nome *button* e valor 1, e recebendo como retorno o parâmetro de nome *result*. Essas informações são complementadas pela definição do método no modelo de domínio (*domainModel*), o qual contém todas as informações necessárias para a sua invocação, como o tipo do dado de cada parâmetro (*dataType*) e se o mesmo é um parâmetro de entrada ou saída (*paramType*).

Dessa forma, o RenderXML deve poder interpretar as informações associadas a uma chamada de método externo descrita em UsiXML, e realizar a invocação no acontecimento do evento associado.

A forma de implementação dessa funcionalidade é descrita no diagrama de seqüência apresentado na Figura 3.19. Nessa situação, a classe *ActionManager* recebe a requisição para tratar um elemento do tipo *action* da classe *BehaviorManager* (Passo 1 na Figura 3.19), criando uma nova lista de ações e associando a ela uma nova instância da classe *MethodCallAction* (Passos 2 e 3 na Figura 3.19), a qual interpreta o elemento UsiXML *methodCall* , armazenando os parâmetros necessários para a invocação do método.

Na ocorrência do evento especificado, o tratador do evento executa todas as ações da lista associada a ele (Passo 5 na Figura 3.19), o que leva à execução da ação de chamada de método (Passo 6 na Figura 3.19). Nesse momento, a classe *MethodCallAction* requisita a invocação do método para a classe *DomainManager* (Passo 7 na Figura 3.19). A classe *DomainManager* recebe as informações para invocação do método externo e as associa às informações presentes no modelo de domínio da interface, realizando a chamada.

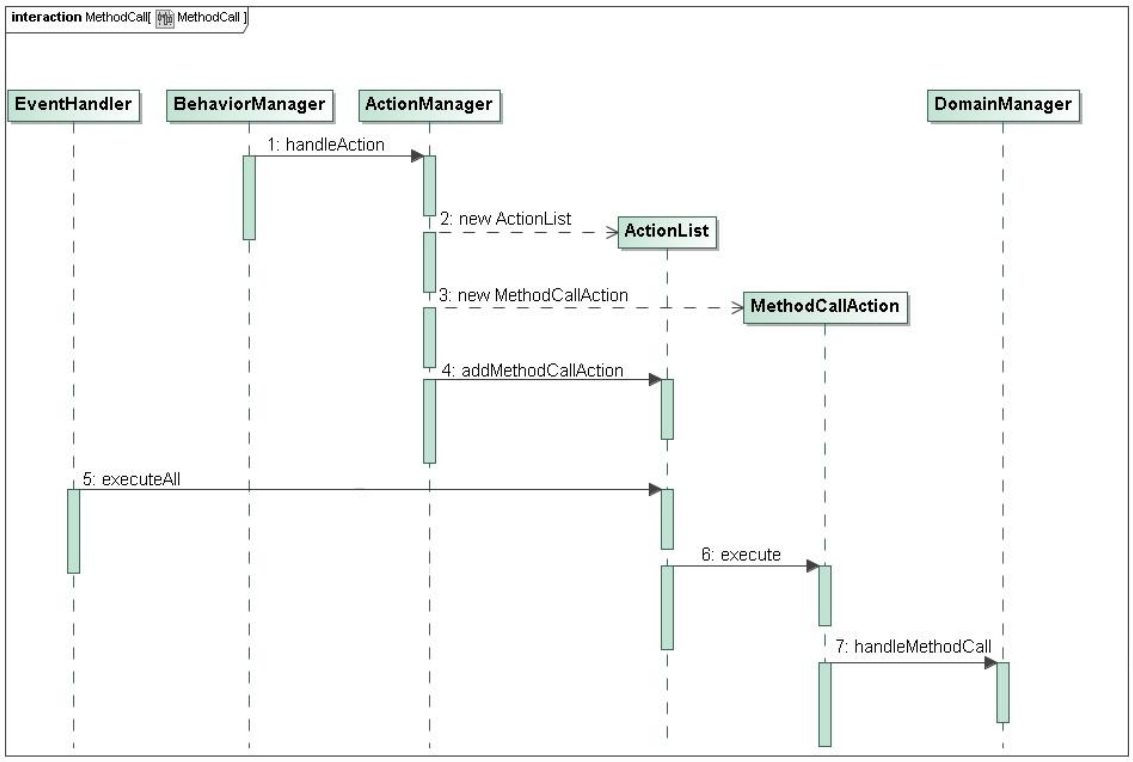

Figura 3.19: Diagrama de seqüência representando a invocação de métodos externos no RenderXML.

No entanto, conforme apresentado na seção 3.1 deste documento, o RenderXML tem como objetivo não somente permitir a invocação de métodos externos, mas também permitir que esta invocação seja realizada em lógicas de aplicação desenvolvidas em múltiplas linguagens de programação. Desse modo, foi criada a arquitetura apresentada na Figura 3.18, onde é definida a interface *LogicConector*, a qual é baseada no padrão GoF *Command* (GAMMA, 1995). Esse padrão é utilizado para encapsular uma requisição dentro de um objeto, e foi utilizado no RenderXML para criar uma maneira uniforme de realizar a invocação de métodos em lógicas de aplicação implementadas em diferentes linguagens de programação.

Na Figura 3.20 pode-se observar que para cada linguagem suportada pelo RenderXML deve ser criada uma instanciação da interface *LogicConector*, responsável pela implementação da conexão com a lógica de aplicação em uma linguagem de programação específica. A figura mostra a implementação dos conectores das duas linguagens de programação atualmente suportadas pelo RenderXML: Java e C#.

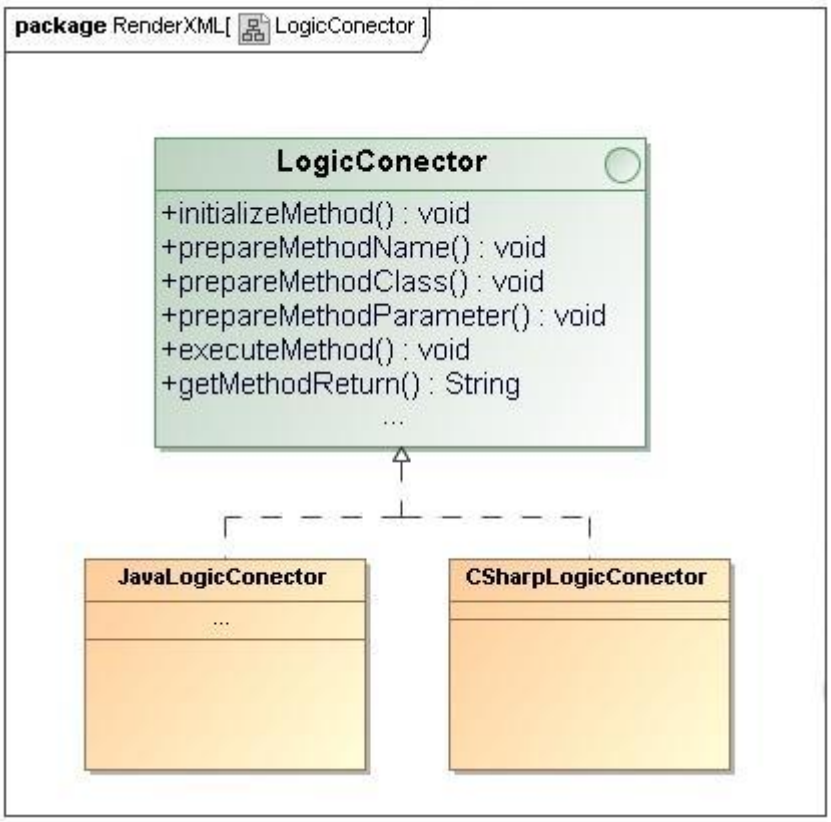

Figura 3.20: Diagrama de classes representando a arquitetura da conexão com a lógica de aplicação implementada pelo RenderXML.

Para permitir a conexão com linguagens diferentes da linguagem de implementação do RenderXML, foi utilizada a interface JNI (*Java Native Interface*) da linguagem Java, a qual permite a conexão dessa linguagem com diferentes linguagens de programação. Dessa maneira, na invocação de métodos em linguagens diferentes de Java, após o conector receber as informações do método a ser requisitado, as informações são transmitidas através de JNI para uma implementação do conector na linguagem-alvo. Essa requisição é então traduzida e o método invocado, sendo os valores de retorno transmitidos utilizando o procedimento inverso.

Esse processo é mostrado na Figura 3.21, a qual exemplifica a conexão com a lógica de aplicação nas duas linguagens suportadas pelo RenderXML.

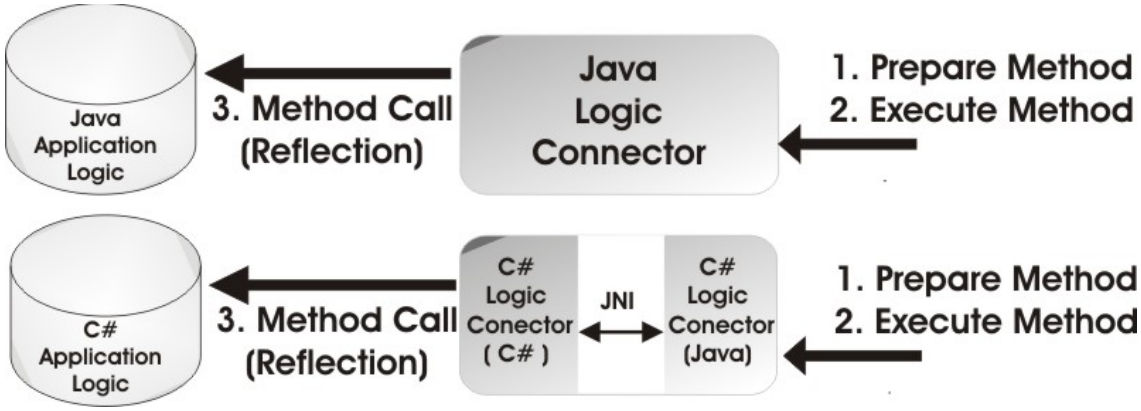

Figura 3.21: Processo de conexão com a lógica de aplicação implementado pelo RenderXML.

Conforme pode ser observado na Figura 3.21, no caso da linguagem Java, o conector recebe a requisição (Passos 1 e 2 na Figura 3.21) e realiza a chamada do método diretamente, utilizando técnicas de reflexão (Passo 3 na Figura 3.21). Já no caso da linguagem C#, o conector é dividido em duas partes, uma implementada em Java e outra em C#, conectadas através de JNI.

4 EXEMPLOS DE APLICAÇÃO DO RENDERXML

Esse capítulo mostrará os estudos de caso desenvolvidos utilizando o RenderXML. Esses tiveram como objetivo testar a utilização da ferramenta em diferentes cenários, validando o trabalho realizado. Os estudos de caso escolhidos, os quais serão apresentados nas subseções a seguir, foram selecionados de acordo com o momento em que se encontrava o desenvolvimento do RenderXML, e também pelo retorno que sua implementação traria ao trabalho.

Dessa forma, o primeiro cenário criado foi o desenvolvimento de uma calculadora multiplataforma, o qual é um exemplo simples que explorou pela primeira vez a utilização do renderizador, tendo principalmente um objetivo didático em relação ao mesmo.

Após a implementação da calculadora, um estudo de caso real foi desenvolvido em parceria com uma empresa, utilizando o RenderXML para adaptar um aplicativo existente a um cenário multiplataforma.

Como validação final, a criação do protótipo de um livro eletrônico falado multiplataforma foi realizada, o qual é o objetivo inicial de todo o trabalho, tendo em vista o contexto em que o mesmo está inserido.

4.1.1 Usando o RenderXML no Desenvolvimento de uma Calculadora Multiplataforma

Para realizar a primeira avaliação do RenderXML no desenvolvimento de uma aplicação, um exemplo de aplicação hipotética (*toy application*) foi desenvolvido: uma calculadora multiplataforma. O objetivo desse estudo de caso foi aplicar na prática a utilização do RenderXML em um pequeno projeto, podendo assim verificar o seu comportamento em uma situação real. Dessa forma, a meta estabelecida foi o desenvolvimento de uma calculadora simples que pudesse ser utilizada em duas diferentes plataformas, *Java Swing* para sua versão *desktop* e móvel, utilizando o JSE e o JME *Connected Device Configuration*, respectivamente. Além disso, as implementações deveriam poder ser utilizadas com lógicas de aplicação desenvolvidas em duas diferentes linguagens de programação, *Java* e *C#*. Um esboço da interface de usuário da calculadora a ser desenvolvida é mostrado na Figura 4.1. Essa figura apresenta uma calculadora simples, que é o objetivo desse estudo de caso.

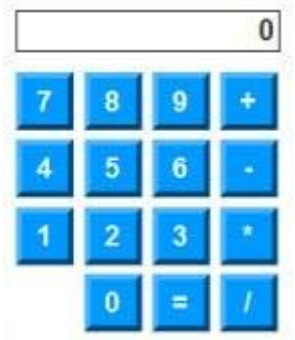

Figura 4.1:Esboço da interface de usuário da calculadora.

De forma a implementar o estudo de caso, o primeiro passo realizado foi a criação da lógica de aplicação da calculadora nas duas linguagens de programação, a qual nesse caso consiste em apenas uma classe, chamada de *CalcLogic.jar* na versão *Java* da calculadora, e *CalcLogic.cs* em sua versão para *C#*. A implementação da classe contém apenas um método, chamado de *buttonPressed*, o qual recebe o botão pressionado pelo usuário na calculadora, e retorna o valor a ser exibido no *display* da mesma.

Com a lógica de aplicação desenvolvida, o segundo passo é a criação da descrição UsiXML da interface de usuário desejada. Por motivos de clareza, o código desenvolvido será apresentado aqui em três partes, ressaltando as três principais informações necessárias no seu desenvolvimento: estrutura, conteúdo e comportamento da interface.

Para descrever a estrutura da interface, foram definidos os componentes que fazem parte da mesma, assim como a sua disposição. Conforme pode ser visto na Figura 4.1, a calculadora é composta basicamente por um campo de texto, o qual representa o seu *display*, e uma série de botões organizados lado a lado.

Dessa forma, a interface foi descrita utilizando-se um gerenciador de *layout* do tipo *flowBox*, o qual posiciona os elementos lado a lado enquanto houver espaço suficiente na interface, passando para a próxima linha quando o espaço acaba. Hierarquicamente abaixo do *flowBox* foram descritos o campo de texto e os diversos botões da interface de usuário, como é apresentado na Figura 4.2.

No trecho de código da Figura 4.2 pode-se observar a descrição de uma janela (*window*), a qual contém um elemento do tipo *flowBox* para gerenciar a disposição dos seus componentes. Dentro desse elemento encontram-se declarados o campo de texto da interface (*inputText* na Figura 4.2) e os seus diversos botões (*button* na Figura 4.2). Além da declaração dos componentes da interface, a descrição contém também os atributos que serão utilizados para instanciar cada componente.

```
<?xml version="1.0" encoding="UTF-8"?>
<uiModel xmlns="http://www.usixml.org"
  ...
  <cuiModel id="5-cui_20" name="5-cui">
    <window id="window_1" name="window_1" width="235" height="200">
      <flowBox id="flow_box_2" name="flow_box_2" alignment="left">
        <inputText id="input_1"
            name="input_1" isVisible="true"
          isEnabled="true" textColor="#000000"
          maxLength="50" numberOfColumns="15" isEditable="true"/>
        <button id="button_1"
            content="/uiModel/resourceModel/cioRef[@cioId='button_1']/resource/@content"
            name="button_1" isVisible="true"
            isEnabled="true" textColor="#000000">
          ...
        </button>
        <button id="button_2"
              content="/uiModel/resourceModel/cioRef[@cioId='button_2']/resource/@content"
          name="button_2" isVisible="true"
          isEnabled="true" textColor="#000000">
          ...
        </button>
        <button id="button_3"
            content="/uiModel/resourceModel/cioRef[@cioId='button_3']/resource/@content"
          name="button_3" isVisible="true"
          isEnabled="true" textColor="#000000">
          ...
        </button>
        ...
      </flowBox>
    </window>
  </cuiModel>
  ...
</uiModel>
```

Figura 4.2: Trecho de código da interface de usuário em UsiXML representando a definição da estrutura da calculadora.

Para permitir a instanciação correta da interface de usuário, além da sua estrutura é necessário definir o conteúdo de cada componente gráfico que será criado. No caso da calculadora, esse conteúdo se refere ao texto que será exibido em cada botão da interface. Para isso, é necessário definir o modelo de recursos da interface, o qual será utilizado para recuperar o conteúdo de cada componente gráfico.

A Figura 4.3 apresenta um trecho do modelo de recursos definido para a descrição da calculadora em UsiXML. No trecho de código presente nessa figura pode-se observar a descrição do caminho do conteúdo de um botão (*button* na Figura 4.3), o qual é informado no valor do seu atributo *content*. Esse caminho é utilizado para encontrar a descrição do conteúdo do componente no modelo de recursos da interface (*resourceModel* na Figura 4.3). No caso exemplificado na figura, está sendo definido o texto “1” para ser exibido no botão presente na interface.

```
<?xml version="1.0" encoding="UTF-8"?>
<uiModel xmlns="http://www.usixml.org"
   ...
   <cuiModel id="5-cui_20" name="5-cui">
      <window id="window_1" name="window_1" width="235" height="200">
         ...
            <button id="button_1"
                  content="/uiModel/resourceModel/cioRef[@cioId='button_1']/resource/@content"
                  name="button_1" isVisible="true"
                  isEnabled="true" textColor="#000000">
               ...
            </button>
            ...
         </flowBox>
      </window>
   </cuiModel>
   ...
   <resourceModel id="calc_resource" name="calc_resource">
      <cioRef cioId="window_1">
         <resource content="RenderXML - Calc" contextId="context-en_US"/>
      </cioRef>
      <cioRef cioId="button_1">
         <resource content="1" contextId="context-en_US"/>
      </cioRef>
      ...
   </resourceModel>
</uiModel>
```

Figura 4.3: Trecho de código da interface de usuário em UsiXML representando a definição do conteúdo da calculadora.

De maneira a finalizar a descrição da interface de usuário, e permitir a execução da mesma, o seu comportamento dinâmico deve ser descrito. No caso específico da calculadora, isso significa fazer com que o método *buttonPressed* da lógica de aplicação seja chamado quando o usuário pressionar um botão, e com que o valor de retorno desse método seja apresentado no *display* da calculadora.

Um trecho de código com a descrição do comportamento dinâmico da calculadora é apresentado na Figura 4.4. Nesse trecho pode ser observado que na declaração do botão (*button1* na Figura 4.4) é colocada a descrição de seu comportamento (*behavior* na Figura 4.4), o qual contém um evento (*event* na Figura 4.4) e duas ações, uma para chamada de método (*method call* na Figura 4.4) e outra para atualização da interface (*uiChange* na Figura 4.4). Dessa maneira, quando o usuário clica no botão, o método *buttonPressed* é invocado, sendo que o mesmo é declarado no modelo de domínio da interface (*domainModel* na Figura 4.4). Após a invocação do método, o valor de retorno é armazenado na variável *valorDisplay*, e então utilizado para atualizar o *display* da calculadora (*input_1* na Figura 4.4).

```
<?xml version="1.0" encoding="UTF-8"?>
<uiModel xmlns="http://www.usixml.org"
   ...
   <cuiModel id="5-cui_20" name="5-cui">
      <window id="window_1" name="window_1" width="235" height="200">
         <flowBox id="flow_box_2" name="flow_box_2" alignment="left">
            ...
            <button id="button_1"
                  content="/uiModel/resourceModel/cioRef[@cioId='button_1']/resource/@content"
                  name="button_1" isVisible="true"
                  isEnabled="true" textColor="#000000">
               <behavior>
                  <event id="evt_b1" eventType="click" eventContext="button_1"/>
                  <action id="act_b1" name="act_b1">
                     <methodCall methodName="buttonPressed">
                           <methodCallParam name="button" value="1"/>
                           <methodCallParam name="result" value="$valorDisplay"/>
                     </methodCall>
                     <uiChange>
                        <changeElement elementId="input_1" attributeName="currentValue"
                                                            value="$valorDisplay"/>
                     </uiChange>
                  </action>
               </behavior>
            </button>
            ...
         </flowBox>
      </window>
   </cuiModel>
   <domainModel id="domain1" name="domain1">
      <domainClass id="calculator_logic" name="br.inf.ufrgs.calc.logic.CalculatorLogic">
         <method id="buttonPressed" name="buttonPressed">
            <param dataType="integer" name="button" paramType="input" passingType="byVal"/>
            <param dataType="string" name="result" paramType="output" passingType="byVal"/>
         </method>
      </domainClass>
   </domainModel>
   ...
</uiModel>
```

Figura 4.4: Trecho de código da interface de usuário em UsiXML descrevendo o comportamento dinâmico da calculadora.

Com a interface de usuário descrita e a lógica de aplicação desenvolvida, a calculadora pode ser executada nas duas plataformas diferentes, sendo que em cada uma delas, a mesma pode ser conectada a lógicas de aplicação desenvolvidas em duas diferentes linguagens de programação. Tendo como exemplo a Figura 4.5, 4 diferentes combinações de interface de usuário com lógica de aplicação podem ser criadas (A,1; B,1; A,2; B,2;).

Na prática, para se escolher a plataforma sendo utilizada, como se modificando de A1 para A2 na Figura 4.5, a única operação necessária é a utilização do renderizador específico da plataforma desejada. No caso de se modificar a lógica de aplicação sendo utilizada, como em A1 para B1 na Figura 4.5, basta que se especifique o conector que se deseja utilizar, desde que as duas versões da lógica de aplicação implementem a mesma interface.

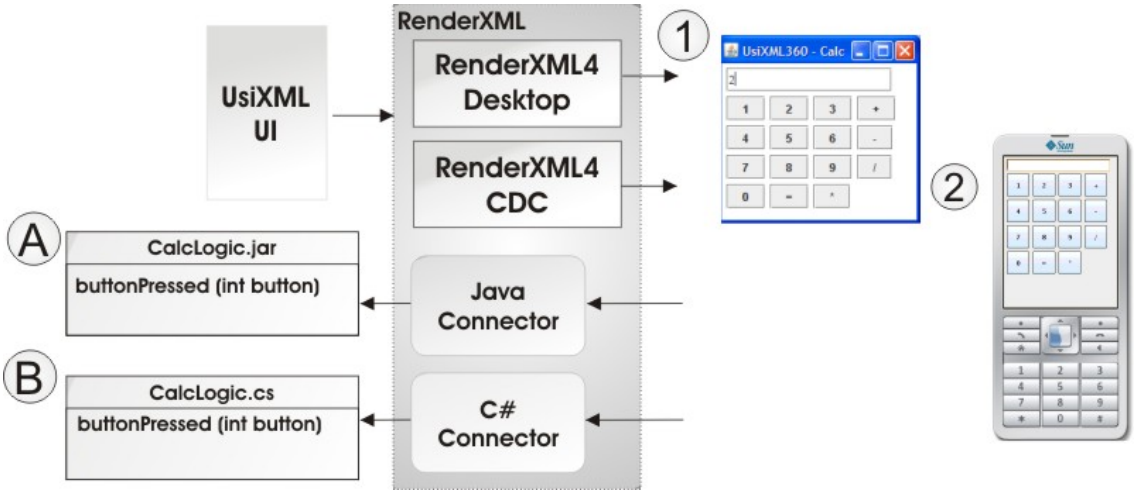

Figure 4.5: Estudo de caso com o desenvolvimento de uma calculadora multiplataforma.

Com o desenvolvimento deste estudo de caso, pode-se avaliar a utilização do RenderXML no desenvolvimento de uma aplicação, utilizando as funções de renderização da interface de usuário, definição do conteúdo da mesma e conexão com a lógica de aplicação. No entanto o estudo de caso apresenta a limitação de a aplicação desenvolvida ser de caráter meramente didático, não permitindo a avaliação do RenderXML em um caso real.

Além disso, por ter sido realizado no início do desenvolvimento do RenderXML, esse estudo de caso ainda não permite a renderização da interface de usuário para plataformas *Web*, o que também é um dos objetivos da ferramenta.

4.1.2 Adaptando um Sistema de Fiscalização Eletrônica com RenderXML

Para suprir as deficiências encontradas no estudo de caso anterior, um novo estudo de caso utilizando o RenderXML foi realizado em parceria com a Companhia de Processamento de Dados do Município de Porto Alegre (PROCEMPA), no âmbito do Projeto Fiscalização Eletrônica. Dessa maneira, pretendeu-se verificar o comportamento da ferramenta desenvolvida em um projeto comercial real.

Esse estudo de caso teve como objetivo a adaptação de um sistema existente para um ambiente multiplaforma, permitindo a sua utilização tanto em dispositivos móveis como em computadores de mesa e em um ambiente *Web*, utilizando as tecnologias *Java Swing* com JSE, JME *Connected Device Configuration* (CDC) e *Java Server Faces*.

O projeto Fiscalização Eletrônica é uma ação de modernização das tarefas de fiscalização da Secretaria Municipal de Indústria e Comércio de Porto Alegre (SMIC), seguindo as diretrizes do Programa Gestão Total da Prefeitura Municipal de Porto Alegre, em parceria com a PROCEMPA. A proposta central deste projeto é disponibilizar de forma automatizada as informações dos estabelecimentos comerciais a serem vistoriados pelos agentes de fiscalização da SMIC, através de um dispositivo móvel contendo os alvarás de funcionamento a serem vistoriados.

Com o objetivo de estudar as possibilidades de adaptação das interfaces do sistema, foram selecionados dois casos de uso do Projeto de Fiscalização Eletrônica: a) Consultar Alvará e b) Baixar Alvará. Estes casos de uso foram inicialmente implementados de fato na plataforma Microsoft .Net, sendo construídos de forma *ad*

hoc através da ferramenta de construção de interfaces do Visual Studio. A Figura 4.6 ilustra a execução da aplicação no ambiente de teste do *Microsoft Visual Studio .NET*.

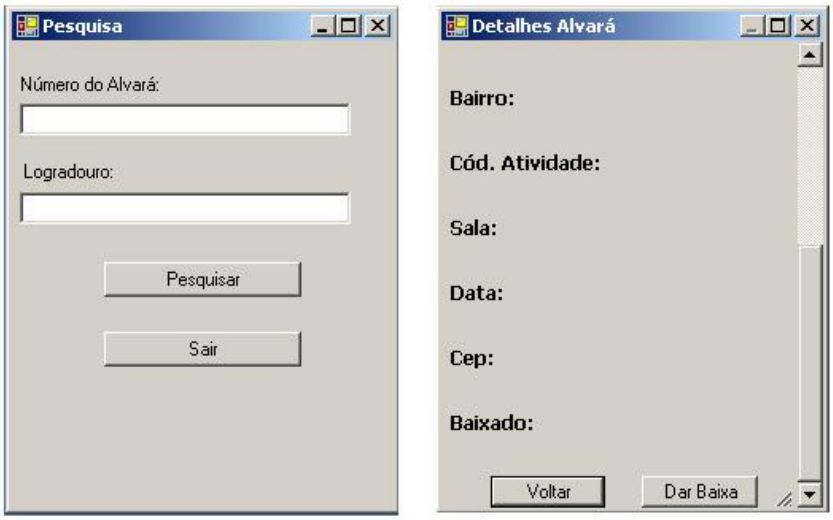

Figura 4.6: Telas dos casos de uso implementadas de maneira *ad-hoc* com o *Microsoft Visual Studio*.

O caso de uso “Consultar Alvará” permite a um agente de fiscalização encontrar um alvará ao qual pretenda exercer alguma ação fiscalizatória, consultando os dados básicos do estabelecimento e o seu status (Figura 4.6). O processo inicia com a apresentação da interface contendo duas possibilidades de filtros: por número do alvará ou por logradouro. Conforme o atributo informando, é feita uma pesquisa na base de dados, alimentando o formulário detalhado com as informações do alvará encontrado.

Já o caso de uso “Baixar Alvará” permite a um agente de fiscalização efetuar a baixa ex-oficio de um alvará encontrado após a pesquisa de alvará. Com o alvará selecionado, o agente aciona o botão e o sistema efetua sua baixa, atualizando a interface com a mudança de status.

Para adaptar o sistema para um ambiente multiplataforma, as interfaces de usuário de ambos os casos de uso foram descritas em UsiXML, de forma a poderem ser interpretadas pela ferramenta RenderXML. Da mesma forma que na seção anterior, por motivos de clareza o código desenvolvido será apresentado aqui em três partes, ressaltando as três principais informações necessárias no seu desenvolvimento: estrutura, conteúdo e comportamento da interface.

De forma a descrever a estrutura desejada, uma descrição de interface de usuário concreta em UsiXML foi criada, contendo as informações das duas janelas que fazem parte do estudo de caso, a janela de pesquisa do alvará e a janela de visualização das informações do mesmo. Em cada uma dessas janelas foi inserido um elemento do tipo *gridBagBox*, que representa um gerenciador de *layout* do tipo *GridBag* na linguagem Java, conforme mostrado no Figura 4.7.

No trecho mostrado na Figura 4.7 podem ser observadas as declarações de duas janelas (*window*), sendo que a primeira delas é apresentada mais detalhadamente,

podendo ser vista a declaração do gerenciador de *layout* (*gridBagBox*), o qual contém um rótulo (*outputText*) e um campo de entrada de dados (*inputText*).

```
<?xml version="1.0" encoding="iso-8859-1"?>
<uiModel xmlns="http://www.usixml.org"
...
   <cuiModel id="5-cui_20" name="5-cui">
      <window id="window_1" name="window_1" width="320" height="480" bgColor="#FFFFFF">
        <gridBagBox id="grid_1" name="grid_1"
          gridHeight="11" gridWidth="5">
          <constraint gridx="1" gridy="1" gridwidth="1" gridheight="1" weightx="1.0" weighty="0.0"
            fill="horizontal" insets="5,5,1,5" anchor="center">
            <outputText id="output_numero_busca" name="output_numero" isVisible="true"
              isEnabled="true" textColor="#000000"
              content="/uiModel/resourceModel/cioRef[@cioId='output_numero'/@content"/>
          </constraint>
          <constraint gridx="1" gridy="2" gridwidth="3" gridheight="1" weightx="1.0" weighty="0.0"
              fill="horizontal" insets="1,5,5,5" anchor="center">
            <inputText id="input_numero_busca" name="input_numero" isVisible="true"
              isEnabled="true" textColor="#000000"
              maxLength="50" numberOfColumns="15" isEditable="true" width="200" height="21"/>
          </constraint>

        </gridBagBox>
      </window>
      <window id="window_2" name="window_2" width="320" height="480" bgColor="#FFFFFF">
...
      </window>
   </cuiModel>
...
</uiModel>
```

Figura 4.7: Trecho de código em UsiXML descrevendo a estrutura da interface de usuário do aplicativo de fiscalização eletrônica.

Além da definição da estrutura da interface, da mesma forma que no estudo de caso anterior, o conteúdo presente em cada componente foi descrito em um modelo de recursos separado. Esse modelo contém as definições de conteúdo das duas janelas existentes na interface de usuário do projeto de fiscalização eletrônica.

A Figura 4.8 mostra parte do modelo de recursos descrito para o projeto de fiscalização eletrônica, ressaltando a definição do título da janela com id igual a *window_1*, a qual representa a janela de pesquisa do alvará, tendo seu título definido como “*Pesquisa*”.

```
<resourceModel id="alvaraEletronico_resource" name="alvaraEletronico_resource">
      <cioRef cioId="window_1">
        <resource content="Pesquisa" contextId="context-en_US"/>
      </cioRef>
      <cioRef cioId="button_pesquisar">
        <resource content="Pesquisar" contextId="context-en_US"/>
      </cioRef>
      <cioRef cioId="button_sair">
        <resource content="Sair" contextId="context-en_US"/>
      </cioRef>
      <cioRef cioId="output_numero_busca">
        <resource content="Número do Alvará" contextId="context-en_US"/>
      </cioRef>
      <cioRef cioId="output_logradouro_busca">
        <resource content="Logradouro" contextId="context-en_US"/>
    ...
</resourceModel>
```

Figura 4.8: Trecho de código em UsiXML descrevendo o modelo de recursos da interface de usuário do aplicativo de fiscalização eletrônica.

Em relação ao comportamento dinâmico da aplicação, existiam dois aspectos a serem resolvidos. O primeiro dizia respeito à conexão da interface com a lógica de aplicação, e para resolver esse problema permitindo a reutilização do código já existente, foi criada uma interface baseada no padrão GoF *Proxy* (GAMMA, 1995), a qual é descrita na Figura 4.9 mostrada abaixo.

Esse código foi desenvolvido apenas para permitir a existência de uma biblioteca única para acesso dos dados da aplicação, a qual também atendesse às restrições existentes na versão atual do RenderXML. Dessa forma, como mostra a Figura 4.9, a interface contém os métodos para pesquisa dos alvarás existentes (*ElectronicInspection.searchLicense*) e também para acesso das informações do alvará encontrado, como o seu número (*ElectronicInspection.getActiveLicenseNumber*).

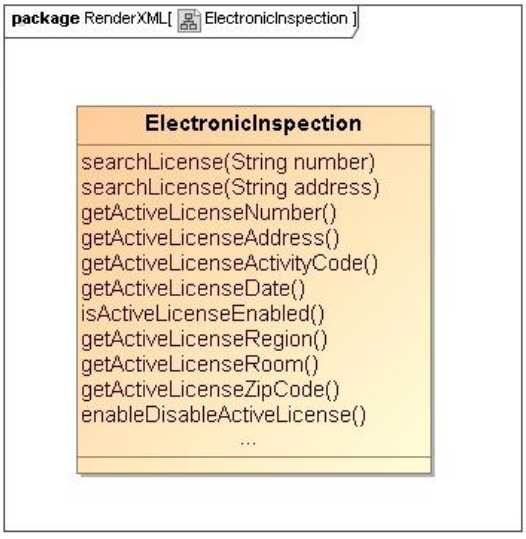

Figura 4.9: Diagrama de classes da interface criada para a conexão com a lógica de aplicação no aplicativo de fiscalização eletrônica.

O segundo aspecto a ser considerado no comportamento dinâmico da interface de usuário do aplicativo de fiscalização eletrônica diz respeito à transição entre as duas janelas existentes na aplicação. Dessa maneira, após o usuário fazer a pesquisa do alvará, a janela de pesquisa deve ser fechada, e a janela com as informações do alvará pesquisado deve ser aberta e mostrada ao usuário.

A definição de parte do comportamento dinâmico da interface de usuário é mostrada na Figura 4.10. Nessa figura está descrito o comportamento do realizado após o usuário pressionar o botão de pesquisa de alvará (*button id=”buttonPesquisar”*). No caso, é realizada uma transição gráfica para fechar a janela de pesquisa (*graphicalTransition id=”tr1”*) e então o método *searchLicense* é invocado para obter o alvará pesquisado.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<uiModel xmlns="http://www.usixml.org"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   ...
   <cuiModel id="5-cui_20" name="5-cui">
      <window id="window_1" name="window_1" width="320" height="480" bgColor="#FFFFFF">
         ...
          <constraint gridx="1" gridy="7" gridwidth="1" gridheight="1" weightx="0.0" weighty="0.0"
             fill="none" insets="5,5,5,5" anchor="center">
            <button id="button_pesquisar"
               content="/uiModel/resourceModel/cioRef[@cioId='button_pesquisar']/resource/@content"
               name="button_pesquisar" isVisible="true" isEnabled="true" textColor="#000000"
               glueHorizontal="middle" width="150" height="21">
                  <behavior>
                  <event id="evt_b1" eventType="click" eventContext="button_1"/>
                  <action id="act_b1" name="act_b1">
                        <triggerTransition>
                              <graphicalTransition id="tr1" transitionType="close">
                                    <source sourceId="button_pesquisar"/>
                                    <target targetId="window_1"/>
                              </graphicalTransition>
                        </triggerTransition>
                        <methodCall methodName="searchLicense">
                              <methodCallParam name="number" value="$valorEntrada"/>
                        </methodCall>
                        <triggerTransition>
                              <graphicalTransition id="tr2" transitionType="open">
                                    <source sourceId="button_pesquisar"/>
                                    <target targetId="window_2"/>
                              </graphicalTransition>
                        </triggerTransition>

                        <methodCall methodName="getActiveLicenseNumber">
                              <methodCallParam name="result" value="$valorNumero"/>
                        </methodCall>
                        <uiChange>
                              <changeElement elementId="input_numero" attributeName="currentValue"
                              value="$valorNumero"/>
                        </uiChange>
                  ...
                  </behavior>
```

Figura 4.10: Trecho de código em UsiXML descrevendo o comportamento dinâmico do aplicativo de fiscalização eletrônica.

Após a invocação do método, a janela de informações do alvará é aberta, através de outra transição gráfica (*graphicalTransition id=”tr2”*), e então para cada campo é

realizada a chamada de um método para recuperar seu valor, seguida da modificação do valor do elemento correspondente da interface. No trecho da Figura A1, essa operação é realizada para preencher o campo com o número do alvará (*getActiveLicenseNumber* e *changeElement*).

Dessa maneira foi possível gerar as interfaces de usuário para o aplicativo de fiscalização eletrônica em três plataformas diferentes, *desktop*, móvel e *web*. Um diagrama apresentando os resultados do estudo de caso é mostrado na Figura 4.11.

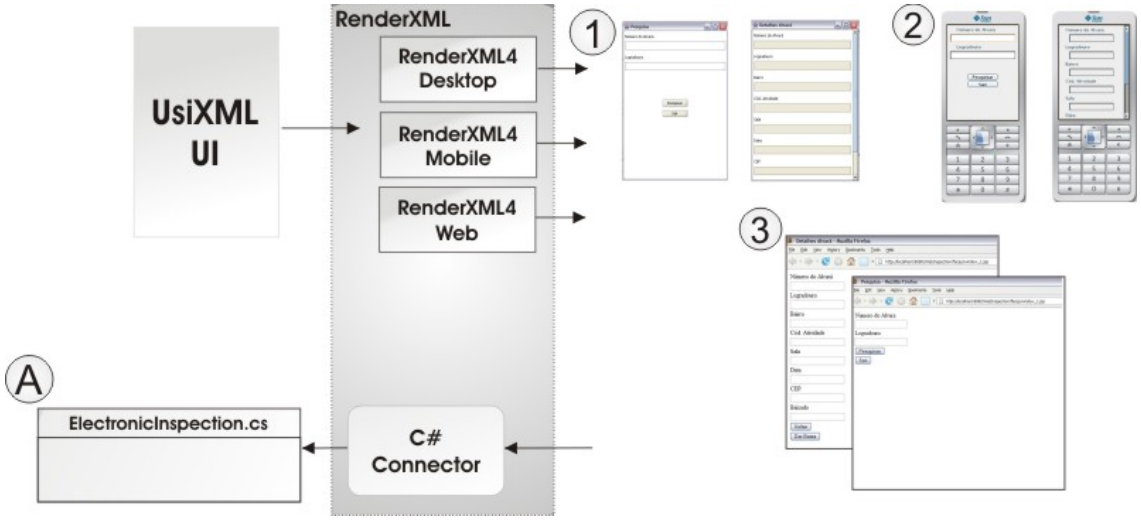

Figura 4.11: Estudo de caso com a adaptação do aplicativo de fiscalização eletrônica para múltiplas plataformas.

Com a realização desse estudo de caso foi possível comprovar a viabilidade da utilização do RenderXML na adaptação de um sistema existente para múltiplas plataformas, além de verificar a sua utilidade em um aplicativo real, proveniente de um ambiente corporativo. Além disso, a implementação do estudo de caso permitiu a verificação de alguns aspectos técnicos do RenderXML, como o funcionamento do mecanismo de transições gráficas, e sua utilização para o desenvolvimento de aplicações voltadas para a plataforma *Web*.

4.1.3 Desenvolvendo um Livro Eletrônico Falado com RenderXML

Como estudo de caso final deste trabalho, foi estabelecido o objetivo de desenvolvimento do protótipo de um livro eletrônico falado para múltiplas plataformas. Esse objetivo tem como origem o projeto LIFAPOR (LIFAPOR), dentro do qual este trabalho está inserido, o qual é um projeto de cooperação entre as universidades UFRGS (Universidade Federal do Rio Grande do Sul) de Porto Alegre/RS - Brasil e INESC-ID (Instituto Nacional de Engenharia de Sistemas de Computação - Investigação e Desenvolvimento) de Lisboa - Portugal.

O objetivo principal do projeto é desenvolver a tecnologia de livros falados, que são a sincronização de arquivos de áudio com arquivos de texto e, posteriormente, imagens. Esse desenvolvimento é muito importante pelo fato de promover a inclusão social de deficientes visuais, porém existem outras aplicações interessantes, como o ensino de português para estrangeiros, a facilitação do aprendizado de pessoas com problemas de concentração, o desenvolvimento de uma biblioteca áudio-visual de livros e a integração das línguas portuguesas (brasileiro e europeu) (LIFAPOR).

O protótipo desenvolvido foi baseado em outro trabalho desse mesmo projeto (SCHMITZ, 2007), o qual desenvolveu um leitor de livros eletrônicos falados para um dispositivo PDA da marca *Palm*, o qual é mostrado na Figura 4.12. Esse aplicativo é capaz de abrir livros eletrônicos falados, exibindo o texto de forma sincronizada com o áudio da leitura. A navegação é feita através de um *slider*, que o usuário deve posicionar de acordo com a posição desejada no livro.

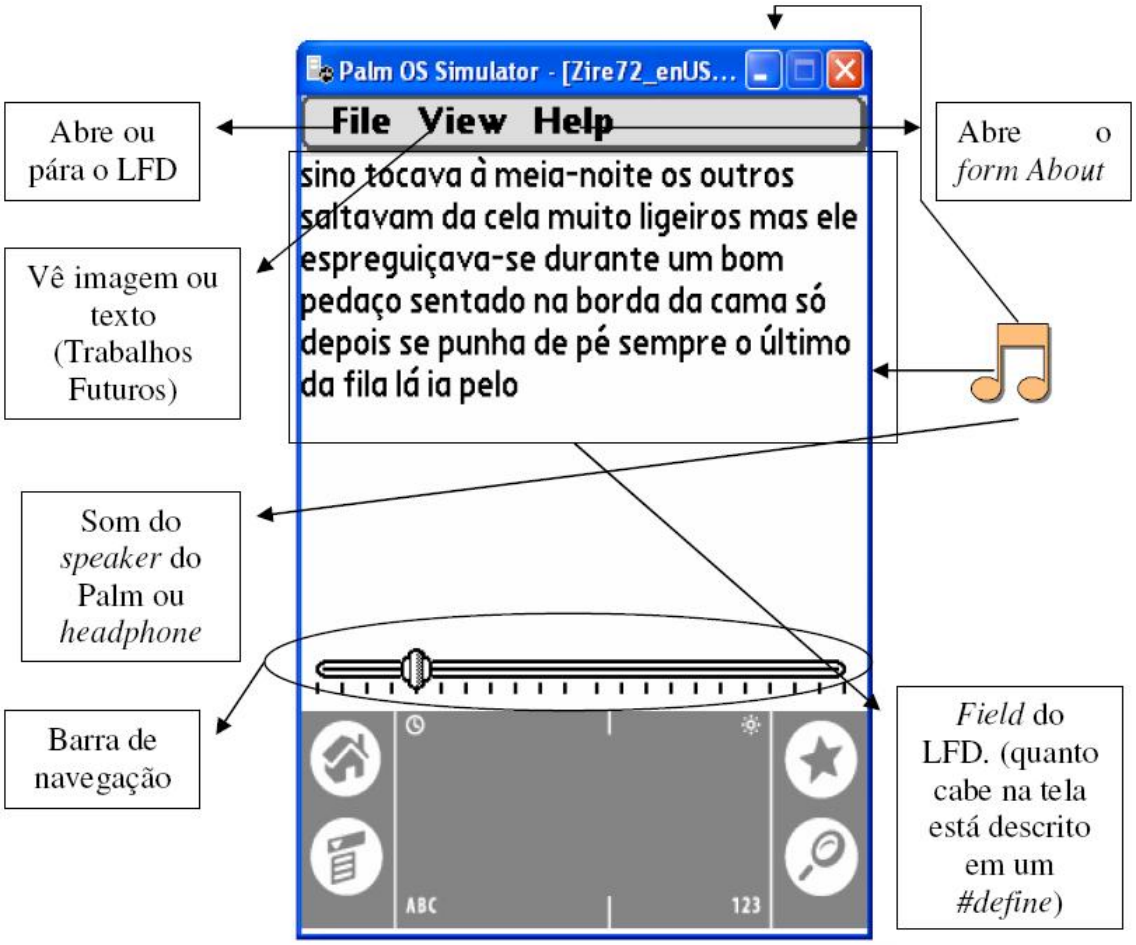

Figura 4.12: Leitor de livros eletrônicos falados para um PDA do tipo *Palm* (SCHMITZ, 2006).

Dessa maneira, o objetivo do estudo de caso foi a implementação de um protótipo de um livro eletrônico falado similar ao mostrado acima, capaz de ser executado nas três diferentes plataformas atendidas pelo RenderXML, *desktop*, móvel e *web*.

Para realizar a implementação do protótipo do leitor de livros eletrônicos falados, a aplicação teve que ser desenvolvida novamente, já que atualmente o RenderXML não suporta lógicas de aplicação desenvolvidas na linguagem C, utilizada no protótipo original. Para isso, foi criada em Java a código mostrado na Figura 4.13. O código consiste na classe *AudiobookReader*, a qual possui entre outros métodos, o método *openAudiobook(String filename)*, que realiza o carregamento de um livro eletrônico falado, e o método *readNextSentence*, o qual realiza a leitura do próximo trecho de áudio, exibindo de forma sincronizada o texto correspondente na interface.

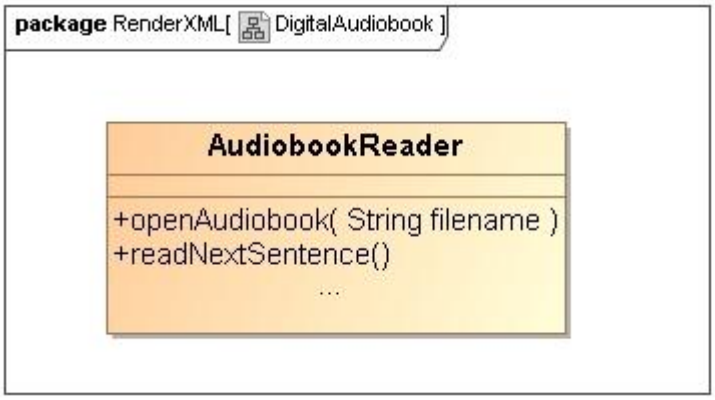

Figura 4.13: Diagrama de classes representando a lógica de aplicação do leitor de livros eletrônicos falados.

Deve ser ressaltado que a lógica de funcionamento do protótipo original teve que ser modificada. Enquanto no original bastava o usuário selecionar a posição inicial desejada para que o livro fosse lido até o fim automaticamente, no protótipo desenvolvido para o RenderXML, o usuário deve ativamente passar trecho por trecho. Essa modificação teve que ser realizada em virtude da especificação da linguagem UsiXML, a qual não possui nenhuma definição para a especificação de eventos criados na lógica de aplicação que devem ser atendidos pela interface de usuário. Dessa forma, não é possível fazer com que o texto apresentado na tela seja modificado automaticamente, sem que o usuário pressione explicitamente o botão para passar ao próximo trecho.

Com a lógica de aplicação desenvolvida, da mesma forma que nos estudos de caso anteriores, a interface de usuário foi descrita em UsiXML. Novamente por questões de clareza, a apresentação do código desenvolvido será realizada separadamente em duas partes, estrutura e comportamento.

Em relação à estrutura da interface, foram definidas duas janelas na aplicação, uma para realizar a seleção do livro eletrônico falado a ser aberto, e outra para sua exibição e navegação. A Figura 4.14 mostra um trecho de código UsiXML com a definição da estrutura das telas existentes na aplicação. Nessa figura são descritas duas janelas (*window_main* e *window_open*), sendo que a primeira representa a janela de exibição do livro eletrônico falado, contendo um botão para abrir um novo arquivo (*button_open*), um área de texto para exibição do texto do livro (*textarea_text*) e um botão para passar para a próxima sentença do livro (*button_next*). Já a segunda janela é utilizada para escolher o arquivo a ser aberto, contendo um campo de texto para a entrada do nome do arquivo a ser aberto (*input_file*) e dois botões, um para abrir o arquivo e outro para cancelar a operação (*button_ok* e *button_cancel*).

Na criação dessa interface de usuário, o elemento *textArea* foi utilizado, apesar de não constar na especificação da UsiXML. Isso teve que ser feito porque não existia nenhum elemento UsiXML que correspondesse ao elemento *javax.swing.JTextArea*, o qual deveria ser usado nessa aplicação.

```
<cuiModel id="5-cui_20" name="5-cui">
    <window id="window_main" name="window_main" width="320" height="480" bgColor="#FFFFFF">
      <gridBagBox id="grid_1" name="grid_1"
        gridHeight="3" gridWidth="3">
        <constraint gridx="0" gridy="0" gridwidth="1" gridheight="1" weightx="1.0"...>
          <button id="button_open" content="/uiModel/resourceModel/cioRef[@cioId='button_open']/resource/@content" ...>
            ...
          </button>
        </constraint>
        <constraint gridx="0" gridy="1" gridwidth="3" gridheight="1" weightx="1.0" ...>
          <textArea id="textarea_text" name="textarea_text" isVisible="true" isEnabled="true" .../>
        </constraint>
        <constraint gridx="2" gridy="2" gridwidth="1" gridheight="1" weightx="1.0" ...>
          <button id="button_next" content="/uiModel/resourceModel/cioRef[@cioId='button_next']/resource/@content" ...>
            ...
          </button>
        </constraint>
      </gridBagBox>
    </window>
    <window id="window_open" name="window_open" width="320" height="480" bgColor="#FFFFFF">
      <gridBagBox id="grid_1" name="grid_1" gridHeight="17" gridWidth="3">
        <constraint gridx="0" gridy="0" gridwidth="1" gridheight="1" ...>
          <outputText id="output_open" name="output_open" isVisible="true" isEnabled="true" textColor="#000000".../>
        </constraint>
        <constraint gridx="0" gridy="1" gridwidth="3" gridheight="1" weightx="1.0"...>
          <inputText id="input_file" name="input_file" isVisible="true" isEnabled="true" textColor="#000000".../>
        </constraint>
        <constraint gridx="1" gridy="2" gridwidth="1" gridheight="1" weightx="0.0"...>
          <button id="button_cancel" content="/uiModel/resourceModel/cioRef[@cioId='button_cancel']/resource/@content" ...>
            ...
          </button>
        </constraint>
        <constraint gridx="2" gridy="2" gridwidth="1" gridheight="1" weightx="0.0" ...>
          <button id="button_ok" content="/uiModel/resourceModel/cioRef[@cioId='button_ok']/resource/@content ...>
            ...
          </button>
        </constraint>
      </gridBagBox>
    </window>
  </cuiModel>
```

Figura 4.14: Trecho de código UsiXML mostrando a estrutura da interface de usuário do leitor de livros eletrônicos falados.

Além da estrutura das interfaces do aplicativo, o seu comportamento também foi definido no código UsiXML. Nesse estudo de caso, o comportamento consistia na chamada da função *AudiobookReader.openAudiobook* no caso da abertura de um novo livro eletrônico, e da função *AudiobookReader.readNextSentence* quando o usuário passa para o próximo trecho do livro, além das transições gráficas entre as duas telas do aplicativo.

Na Figura 4.15 é mostrado um trecho de código com o comportamento acionado pelo botão *button_next*, o qual requisita o próximo trecho do livro eletrônico falado para a lógica de aplicação. Nesse caso, o método *readNextSentence* é invocado, e o seu resultado é colocado na área de texto, através de uma modificação na interface (*uiChange*).

```
<button id="button_next"
   content="/uiModel/resourceModel/cioRef[@cioId='button_next']/resource/@content" ...>
  <behavior>
    <event id="evt_b1" eventType="click" eventContext="button_open"/>
    <action id="act_b1" name="act_b1">
      <methodCall methodName="readNextSentence">
        <methodCallParam name="result" value="$texto"/>
      </methodCall>
      <uiChange>
        <changeElement elementId="textarea_text" attributeName="currentValue" value="$texto"/>
      </uiChange>
    </action>
  </behavior>
</button>
```

Figura 4.15: Trecho de código UsiXML exemplificando o comportamento do leitor de livros eletrônicos falados.

Com a definição da interface de usuário, o protótipo do leitor de livros eletrônicos falados pode ser executado nas três diferentes plataformas suportadas pelo RenderXML, conforme mostrado na Figura 4.16.

O desenvolvimento desse estudo de caso, apesar de não ter testado nenhuma nova característica do RenderXML, mais uma vez comprovou a possibilidade de criação de aplicativos para diversas áreas utilizando essa ferramenta. Além disso, o estudo de caso teve o seu principal objetivo atingido, o qual era o desenvolvimento do protótipo multiplataforma do leitor de livros eletrônicos falados para sua utilização no contexto do projeto LIFAPOR.

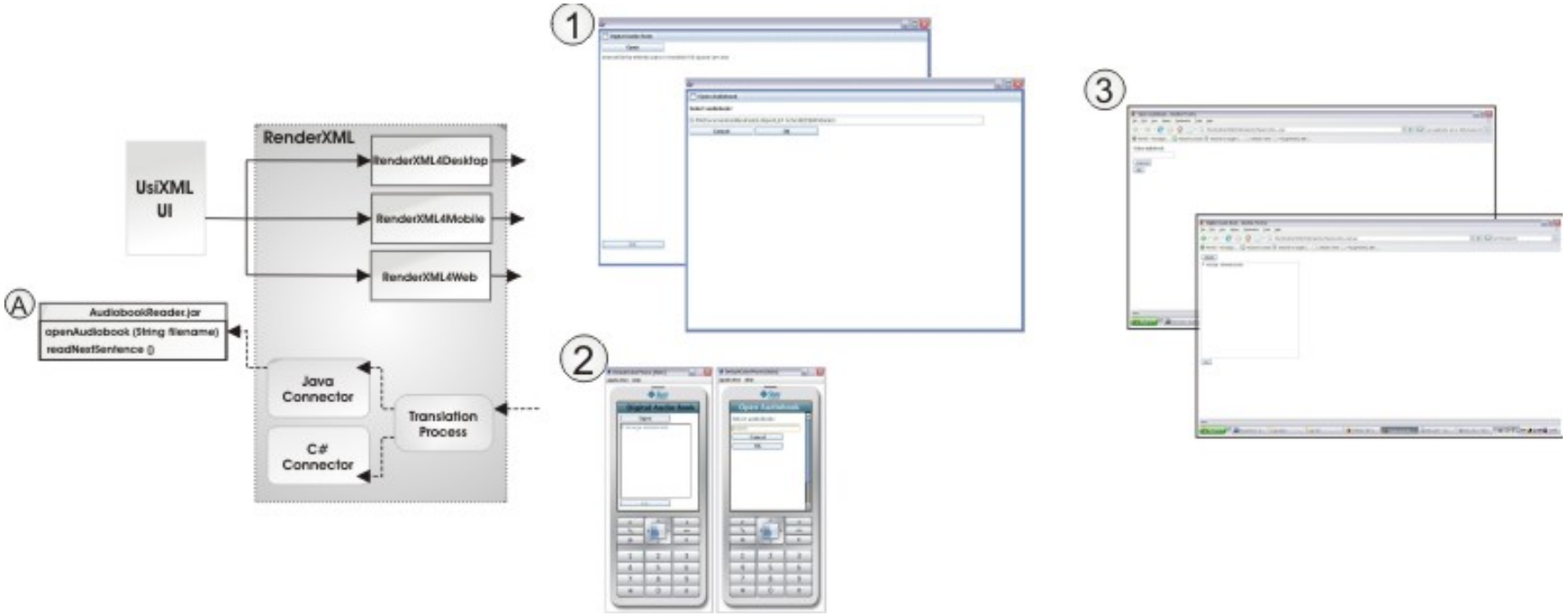

Figura 4.16: Estudo de caso com o desenvolvimento de um leitor de livros eletrônicos falados para múltiplas plataformas.

5 CONCLUSÃO

Esse capítulo apresenta as considerações finais dessa dissertação. Primeiramente são apresentadas as contribuições resultantes desta pesquisa e a ferramenta desenvolvida. A seguir são discutidas as limitações do trabalho realizado, e os motivos que as ocasionaram. Finalmente é apresentado um conjunto de possíveis rumos a serem tomados no prosseguimento desse trabalho.

5.1 Contribuições

O trabalho realizado culminou na definição e construção do RenderXML, um conjunto de ferramentas de renderização capazes de realizar a renderização de interfaces de usuário concretas em UsiXML para interfaces de usuário finais em três diferentes plataformas, computadores *desktop*, PDA's e plataformas *Web*, todas baseadas na tecnologia Java. Dessa maneira, se propôs uma abordagem prática para a prototipação e desenvolvimento de interfaces de usuário multiplataforma utilizando UsiXML, e também para a adaptação de aplicativos existentes para uma ambiente multiplataforma. A importância dessa proposta se baseia na crescente demanda por interfaces de usuário capazes de se adaptar a múltiplos dispositivos, e na tendência de utilização de uma abordagem de desenvolvimento baseada em modelos, assim como de linguagens de descrição de interfaces de usuário no atendimento desse requisito. Essa importância é comprovada pelo interesse da comunidade de usuários UsiXML na utilização do RenderXML, o qual já faz parte do acervo de trabalhos disponibilizados pelo UsiXML *Consortium*.

Em adição à contribuição dada pela ferramenta a ser utilizada individualmente, o RenderXML contribui para um esforço maior de criação de um processo de desenvolvimento de interfaces de usuário multiplataforma utilizando a linguagem UsiXML. Nesse contexto, diversas ferramentas e abordagens foram e estão sendo desenvolvidas, sendo que a etapa de renderização da interface concreta para a interface final não é atualmente atendida por nenhuma ferramenta, fazendo com que o RenderXML venha a preencher uma lacuna existente no contexto da utilização da UsiXML.

Em relação às suas funcionalidades, o RenderXML contribui de forma inovadora no enfoque dado à conexão da interface de usuário com a lógica de aplicação do *software*. Nesse aspecto, os demais trabalhos que abordaram a renderização de interfaces de usuário, tanto utilizando UsiXML como outras linguagens, não deram ênfase a essa questão, e principalmente não tentaram explicar como essa conexão pode ser realizada em um escopo maior, onde aplicativos podem ser desenvolvidos utilizando diferentes

linguagens de programação. Estamos convencidos, assim como outros membros do UsiXML *Consortium*, que esta é uma das principais contribuições do RenderXML.

Além disso, por ser a única ferramenta de renderização desenvolvida utilizando a especificação atual da UsiXML, e também por ter um enfoque diferente sobre determinados aspectos, como o da conexão com a lógica de aplicação citado acima, o desenvolvimento do RenderXML auxiliou com a utilização e avaliação da atual especificação da interface de usuário concreta existente atualmente na UsiXML. Dessa maneira, algumas limitações da UsiXML foram encontradas e expostas, inclusive dando margem à realização de outros trabalhos que visam melhorar a sua especificação atual.

5.2 Limitações

Mesmo com contribuições significativas, o trabalho realizado também possui algumas limitações, que serão detalhadas a seguir.

Apesar de ter sido avaliado nos três estudos de caso apresentados no decorrer do trabalho, o RenderXML não atende em sua versão atual à toda a especificação da linguagem UsiXML, tendo dado prioridade aos elementos necessários nos estudos de caso realizados. Isso se deve às restrições de tempo no desenvolvimento do trabalho, mas principalmente ao fato de que a UsiXML ainda é uma linguagem em evolução, e a sua documentação é precária, fazendo com que grande parte do conhecimento adquirido para realização desse trabalho tenha sido obtido através de trabalhos relacionados e contatos diretos com a equipe de desenvolvimento da linguagem, assim como outros membros do UsiXML *Consortium*, e não propriamente com sua especificação.

Em relação a sua forma de implementação, o RenderXML possui uma versão para cada tipo de plataforma a ser atendida. Embora isso seja necessário, tendo em vista que o RenderXML deve ser utilizado em tempo de execução, essa característica dificulta o atendimento de novos dispositivos, já que seria necessário o desenvolvimento de uma nova versão, ou ao menos a adaptação da uma das versões existentes para que um novo dispositivo seja atendido. No entanto, se fosse intenção realizar o mapeamento de interfaces em tempo de projeto (e não execução), o RenderXML poderia ser unificado em apenas uma ferramenta.

5.3 Trabalhos Futuros

Este trabalho possibilita diversos caminhos para sua continuidade e evolução.

A primeira possibilidade seria a evolução do RenderXML. Essa evolução poderia ser realizada em três diferentes níveis. Um deles seria a ampliação da cobertura da especificação UsiXML atendida pelo renderizador, ampliando as possibilidades de criação de interfaces de usuário com o RenderXML. Outro seria a criação de renderizadores para novos dispositivos, permitindo que a UsiXML fosse utilizada em um maior numero de casos, o que também poderia ser obtido com o aumento das possibilidades de linguagem de programação a serem utilizadas na lógica de aplicação. Em suma, pode-se expandir o trabalho realizado em três direções:

- Cobertura da especificação da linguagem UsiXML.
- Aumento do número de dispositivos/plataformas atendidas.
- Aumento do número de linguagens de programação suportadas.

Dentre os dispositivos/plataformas que poderiam ser atendidos estão outros telefones celulares, PDA's, e até mesmo computadores XO do projeto OLPC (OLPC), com o qual já realizamos alguns testes não documentados neste trabalho. O objetivo final seria permitir a criação de interfaces de usuário baseadas em UsiXML para uma grande diversidade de dispositivos e plataformas, expandindo também o número de linguagens de programação passíveis de serem utilizadas. Em particular, nosso trabalho tem como meta final a criação de uma ferramenta que possa ser utilizada no desenvolvimento de interfaces em uma grande diversidade de contextos de uso.

Além disso, o atendimento a novas modalidades de interação (multimodalidade) também seria um objetivo interessante. Para isso, seria necessária a criação de renderizadores que fossem capazes de gerar interfaces de usuário para diferentes modalidades, como a interação vocal e gestual, por exemplo.

Outra possibilidade a ser investigada é a adaptação do RenderXML para ser utilizado como um renderizador de interfaces para uma linguagem de descrição de interfaces comercial, como o caso do *JavaFX*. Dessa maneira, o RenderXML poderia ser utilizado em tempo de projeto, gerando código em *JavaFX*, o qual pode ser utilizando em diferentes dispositivos. Essa seria uma forma de associar as vantagens do desenvolvimento baseado em modelos com UsiXML à uma especificação que visa descrever interfaces de usuário para uma grande quantidade de dispositivos, com o facilidade de ser necessária apenas uma ferramenta de renderização nesse caso. Isto tem uma vantagem interessante, que é permitir a evolução do *design* e até mesmo o *redesign* da interface de usuário gerada, o que hoje não é possível já que o RenderXML atua em tempo de execução.

Em relação a outros trabalhos que utilizam a linguagem UsiXML, uma possibilidade existente é a integração do RenderXML com ferramentas de projeto de interfaces de usuário que geram como saída interfaces de usuário concretas em UsiXML, como o GrafiXML e o SketchiXML, apresentados anteriormente nesse trabalho. Essas ferramentas atualmente não possuem a possibilidade de renderização da interface de usuário projetada para uma interface de usuário final, e a sua integração com o RenderXML criaria um processo de desenvolvimento de interfaces completo com a utilização da linguagem UsiXML.

Também seria interessante a adaptação do RenderXML para sua utilização com outras linguagens de descrição de interfaces de usuário, como a UIML, apresentada no capítulo 2. Especificamente em relação a UIML, essa linguagem possui um aspecto interessante que é a definição de vocabulários, o quais são utilizados para a obtenção dos elementos necessário para renderização. Dessa maneira, múltiplas plataformas podem ser atendidas apenas por uma ferramenta de renderização, sendo necessário somente a mudança do vocabulário utilizado.

REFERÊNCIAS

ABRAMS, M. et al. UIML: An Appliance-IndependentXML User Interface Language. **Computer Networks**, Amsterdam, v. 31, n. 11, 1999. Trabalho apresentado na 8th International WWW Conference, 1999, Toronto, Canada.

ALI, M.F. et al. Building Multi-Platform User Interfaces With UIML. In: INTERNATIONAL WORKSHOP OF COMPUTER-AIDED DESIGN OF USER INTERFACES, CADUI, 2002, Valenciennes, France. **Proceedings…** Dordrecht: Kluwer Academic Publishers. 2002.

AZEVEDO, P.; MERRICK, R.; ROBERTS, D. **OVID to AUIML:** User Oriented Interface Modeling**.** [S.l.]: IBM, 2000. Technical Report.

BERGHE, Y. V. **Etude et implémentation d'un générateur d'interfaces vectorielles à partir d'un langage de description d'interfaces utilisateur**, 2004. Dissertação (Mestrado) - Université Catholique de Louvain, Louvain-la-Neuve, Bélgica.

BERTI, S. et al. The TERESA XML Language for the Description of Interactive Systems at Multiple Abstraction Levels. In: WORKSHOP ON DEVELOPING USER INTERFACES WITH XML: ADVANCES ON USER INTERFACE DESIGN, UIXML, 2004, Galipolli, Itália. **Proceedings…** Gallipoli: EDM-Luc, 2004. p. 103-110.

BOUILLON, L.; VANDERDONCKT, J.; EISENSTEIN, J. Model-Based Approaches to Reengineering Web Pages, In: INTERNATIONAL WORKSHOP ON TASK MODELS AND DIAGRAMS FOR USER INTERFACE DESIGN, TAMODIA, 2002, Bucharest, Romania. **Proceedings…** Bucharest: INFOREC Publishing House, 2002.

BOUILLON, L.; VANDERDONCKT, J.; SOUCHON, N. Recovering Alternative Presentation Models of a Web Page with VAQUITA, In: INTERNATIONAL WORKSHOP OF COMPUTER-AIDED DESIGN OF USER INTERFACES, CADUI, 2002, Valenciennes, França. **Proceedings...** Dordrecht: Kluwer Academic Publishers, 2002.

BOUILLON, L. **Reverse Engineering of Declarative User Interfaces**. 2006. Tese (Doutorado), Université Catholique de Louvain, Louvain-la-Neuve, Bélgica.

CALVARY, G.; COUTAZ, J.; THEVENIN, D. A Unifying Reference Framework for the Development of Plastic User Interfaces. In: IFIP INTERNATIONAL CONFERENCE ON ENGINEERING FOR HUMAN-COMPUTER INTERACTION, EHCI, 8., 2001, Toronto, Canada. **Proceedings…** Berlin: Springer, 2001. p. 173-192. (Lecture Notes in Computer Science, v. 2254).

CALVARY, G.; COUTAZ, J.; THEVENIN, D. Supporting Context Changes for Plastic User Interfaces: a Process and a Mechanism. In: AFIHM-BCS CONFERENCE ON HUMAN-COMPUTER INTERACTION, IHM-HCI, 2001, Lille, França. **People and**

Computers XV -- Interaction without Frontiers. Londres: Springer-Verlag, 2001. p. 349–363.

CALVARY, G. et al. Plasticity of User Interfaces: A Revised Reference Framework**.** In: INTERNATIONAL WORKSHOP ON TASK MODELS AND DIAGRAMS FOR USER INTERFACE DESIGN, TAMODIA, 2002, Bucharest, Romania. **Proceedings…** Bucharest: INFOREC Publishing House, 2002. p 127-134.

COLLIGNON, B.; VANDERDONCKT, J.; CALVARY, G. An Intelligent Editor for Multi-Presentation User Interfaces. Trabalho submetido e aprovado para o 23th ACM SYMPOSIUM ON APPLIED COMPUTING, 2008, Fortaleza, Brasil.

CONSENSUS Project. Disponível em: <http://www.consensus-online.org/>. Acesso em: nov. 2007.

COYETTE, A. et al. SketchiXML: Towards a Multi-Agent Design Tool for Sketching User Interfaces Based on UsiXML. In: INTERNATIONAL WORKSHOP ON TASK MODELS AND DIAGRAMS, TAMODIA, 2004, Prague, Czech Republic. **Proceedings…** Toulouse: ACM/Czech Technical University Publishing House, 2004. p. 75-82.

DENIS, V. **Un pas vers le poste de travail unique:** QTKiXML, un interpréteur d'interface utilisateur à partir de sa description**.** 2005. Dissertação (Mestrado) - Université Catholique de Louvain, Louvain-la-Neuve, Bélgica.

EISENSTEIN, J.; VANDERDONCKT, J.; PUERTA, A. Adapting to Mobile Contexts with User-Interface Modeling. In: WORKSHOP ON MOBILE COMPUTER SYSTEMS AND APPLICATIONS, WMCSA, 2000, Monterey, USA. **Proceedings…** Monterey, California: IEEE Press, 2000.

FRAINER, A. S.; PIMENTA, M. S.; PRICE, R. T. Como Obter Portabilidade de Programas Interativos. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 10., 1990, Vitória, ES. **Anais...** Vitória: SBC, 1990. p. 496-512

GAMMA, E. et al. **Design Patterns:** Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1995.

LIFAPOR – Livros Falados em Português. Disponível em: <www.lifapor.org>. Acesso em: nov. 2007.

LEPREUX, S.; VANDERDONCKT, J.; MICHOTTE, B. Visual Design of User Interfaces by (De)composition, In: WORKSHOP ON DESIGN, SPECIFICATION AND VERIFICATION OF INTERACTIVE SYSTEMS, DSV-IS, 13., 2006. Dublin, Ireland. **Revised Papers**. Berlin: Springer-Verlag, 2006. p. 157-170. (Lecture Notes in Computer Science, v. 4323).

LEPREUX, S. et al. Towards Multimodal User Interfaces Composition based on UsiXML and MBD principles. In: INTERNATIONAL CONFERENCE ON HUMAN-COMPUTER INTERACTION, HCII, 12., 2007, Beijing, China. **Proceedings…** Berlin: Springer-Verlag, 2007. p. 134-143. (Lecture Notes in Computer Science, v. 4552).

LIMBOURG, Q. et al. UsiXML: A User Interface Description Language for Context-Sensitive User Interfaces. In: WORKSHOP ON DEVELOPING USER INTERFACES WITH XML: ADVANCES ON USER INTERFACE DESIGN, UIXML, 2004, Galipolli, Itália. **Proceedings…** Gallipoli: EDM-Luc, 2004. p. 55–62.

LUYTEN, K.; CONINX, K. UIML.NET: an Open UIML Renderer for the .Net Framework. In: COMPUTER-AIDED DESIGN OF USER INTERFACES, CADUI, 4., 2004, Funchal, Portugal. **Proceedings…** Dordrecht, Holanda: Kluwer Academic Publishers, 2004. p. 257-268.

MORI, G.; PATERNÒ, F.; SANTORI, C. Tool Support for Designing nomadic Applications. In: INTERNATIONAL CONFERENCE ON INTELLIGENT USER INTERFACES, IUI, 9., 2004, Funchal, Portugal. **Proceedings…** New York: ACM Press, 2003. p. 141-148.

OCAL, K. **Etude et développement d'un interpréteur UsiXML en Java Swing.** Liège: Haute Ecole Rennequin, 2004.

OLPC: One Laptop Per Child. Disponível em: <http://www.laptop.org/index.en_US.html>. Acesso em: nov. 2007.

OPENLASZLO. Disponível em: <http://www.openlaszlo.org/>. Acesso em: nov. 2007.

PATERNÒ, F.; SANTORO C. One Model, Many Interfaces. In: INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN OF USER INTERFACES, CADUI, 4., 2002, Valenciennes, France. **Proceedings…**Dordrecht, Holanda: Kluwer Academics Publishers, 2002. p. 143-154.

PATERNÒ, F. Model-based Tools for Pervasive Usability. **Interacting with Computers**, Guildford, Surrey, v.17, n. 3, p. 291-315, 2005.

PHANOURIOU, C. **UIML:** A Device Independent User Interface Markup Language. 2000. Tese (Doutorado) - Virginia Polytechnic Institute and State University, Blacksburg, Virginia.

PIMENTA, M. S.; HEUSER, C. A. Canonicus: um Modelo para Portabilidade de Programas Interativos. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 5., 1991, Ouro Preto. **Anais...** Belo Horizonte: UFMG, 1991.

PIMENTA, M. S. Rumo a Portabilidade de Componentes de Dialogo. In: CONFERÊNCIA LATINO-AMERICANA DE INFORMÁTICA, CLEI, 18., 1992. **Actas**. Las Palmas de Gran Canária: CLEI, 1992.

PUERTA, A.; EISENSTEIN, J. XIML: A Common Representation for Interaction Data. In: INTERNATIONAL CONFERENCE ON INTELLIGENT USER INTERFACES, IUI, 6., 2002, San Francisco, USA. **Proceedings…** New York: ACM Press, 2002. p. 216-217.

ROSA, D. **Construindo Interfaces com o Usuário em Múltiplas Plataformas.** 2005. Trabalho de Diplomação (Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

SIMON, R.; WEGSCHEIDER, F.; TOLAR, K. Tool-supported Single Authoring for Device Independence and Multimodality. In: INTERNATIONAL CONFERENCE ON HUMAN-COMPUTER INTERACTION WITH MOBILE DEVICES & SERVICES, MOBILEHCI, 7., 2005, Salzburg, Austria. **Proceedings....** New York: ACM Press, 2005. p. 91 – 98.

SCHMITZ, L. **Livros Falados Digitais em Português para Computadores de Mão**. 2006. Trabalho de Diplomação (Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

STANCIULSESCU, A. et al. A Transformational Approach for Multimodal Web User Interfaces Based on UsiXML. In: INTERNATIONAL CONFERENCE ON MULTIMODAL USER INTERFACES, ICMI, 7., 2005, Trento, Italy. **Proceedings…** New York: ACM Press, 2005. p. 259-266.

SZCZUR, M. Transportable Applications Environment - An Integrated Design-to-Production UIMS. In: ANNUAL CONFERENCE AND EXPOSITION TO COMPUTER GRAPHICS APPLICATIONS, 1988. **Proceedings….** [S.l.: s.n.], 1988.

TRAETTEBERG, H.; MOLINA, P. J.; NUNES, N. J., Making Model-Based UI Design Practical: Usable and Open Methods and Tools. In: INTERNATIONAL CONFERENCE ON INTELLIGENT USER INTERFACES, IUI, 9., 2004. Funchal, Portugal. **Proceedings…**New York: ACM Press, 2004. p. 376-377.

USIXML: Home of the User Interface eXtensible Markup Language. Disponível em: <www.usixml.org>. Acesso em: nov. 2007.

USIXML2OPENLASZLO. Disponível em: <http://www.usixml.org/index.php?mod=pages&id=30>. Acesso em: nov. 2007.

VALDES, R. A Virtual Toolkit for Windows and the Mac**. Byte**, Peterborough, v. 14, p. 209 - 216, 1989.

VAN SLUYS, M. **Développement d'un maquetteur d'interfaces à l'aide de Microsoft Visio.** 2004. Dissertação (Mestrado) - Université Catholique de Louvain, Louvain-la-Neuve, Bélgica.

VANDERDONCKT, J.; BERQUIN, P. Towards a Very Large Model-based Approach for User Interface Development. In: INTERNATIONAL WORKSHOP ON USER INTERFACES TO DATA INTENSIVE SYSTEMS, UIDIS, 1., 1999, Edinburgh, Scotland. **Proceedings…**Los Alamitos: IEEE Computer Society Press, 1999. p. 76–85.

W3C WORKING GROUP. **Authoring Techniques for Device Independence**. 2004. Disponível em: <http://www.w3.org/TR/2004/NOTE-di-atdi-20040218/>. Acesso em: nov. 2007.

XAML.NET, a guide to XAML. Disponível em: <http://www.xaml.net/>. Acesso em: nov. 2007.

XFORMS 1.0, W3C RECOMMENDATION, 14 Oct. 2003. Disponível em: <http://www.w3.org/TR/xforms/>. Acesso em: nov. 2007.

XML User Interface Language (XUL) Project. Disponível em: <http://www.mozilla.org/projects/xul/>. Acesso em: nov. 2007.

ANEXO

Devido à extensão do material, os códigos-fonte da ferramenta RenderXML, assim como dos estudos de caso desenvolvidos estão contidos em um CD fornecido em anexo.

APÊNDICE

Informações atualizadas sobre o trabalho podem ser encontradas na pagina *web* do UsiXML *Consortium* (www.usixml.org), além da página do projeto no diretório *Sourceforge* (http://sourceforge.net/projects/renderxml/).

Além disso, informações podem ser encontradas nos artigos publicados no decorrer do trabalho, que são listados a seguir:

- Trindade, F.M., Pimenta, M.S., RenderXML – A Multi-platform Software Development Tool, Proc. of 6th Int. Workshop on TAsk MOdels and DIAgrams **TAMODIA'2007** (Toulouse, 7-9 November 2007), Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2007, pp. 292-297.
- Trindade, F., Pimenta, M., Petrillo, F., Iochpe, C. Adaptando Sistemas Existentes para um Ambiente de Execução Multiplataforma, Proc. of Workshop on Perspectives, Challenges and Opportunities for Human-Computer Interaction in Latin America **CLIHC'2007**(Rio de Janeiro, September 10-11, 2007).
- Trindade, F. Pimenta, M. UsiXML4ALL - A Multiplatform Software Development Tool, in DVD Proc. of 12th Int. Conf. on Human-Computer Interaction **HCI International'2007** (Beijing, 22-27 July 2007).