Universidade do Minho
Escola de Engenharia

Sandrine Alves Mendes

FlexiXML
Um animador de modelos de interfaces com o utilizador

Outubro de 2009

Sandrine Alves Mendes **FlexiXML Um animador de modelos de interfaces com o utilizador**

UMinho | 2009

Universidade do Minho
Escola de Engenharia

Sandrine Alves Mendes

FlexiXML
Um animador de modelos de interfaces com o utilizador

Mestrado de Informática

Trabalho efectuado sob a orientação do
Professor Doutor José Francisco Creissac Freitas de Campos

Outubro de 2009

Para ti...

Agradecimentos

Em primeiro lugar, gostaria de agradecer à minha mãe e ao meu irmão, que tão carinhosamente me apoiaram ao longo deste mestrado. Estiveram sempre presentes, não deixaram nunca de me apoiar nos momentos mais difíceis. Quero agradecer também ao meu namorado, pela imensa força, apoio e compreensão.

Este trabalho não teria sido possível sem a orientação do Professor José Creissac. Para além da orientação ao longo do mestrado, foi incasável no seu apoio, incentivo e disponibilidade.

Agradeço também todo o apoio dos meus amigos, em especial ao Nuno Guerreiro, Vítor Pinheiro e João Abrantes, os quais viveram mais de perto esta experiência.

Não poderia deixar de referir a minha entidade empregadora, a *ALERT Life Sciences Computing S.A*, por ter permitido a frequência neste Mestrado. Queria também agradecer àqueles que trabalham directamente comigo, por me terem apoiado e ajudado a garantir o cumprimento dos projectos estipuladas.

Finalmente, gostaria de agradecer a Jean Vaderdonckt, da Unidade Católica de Louvain, pelo apoio prestado relativamente à linguagem de modelação UsiXML e a ferramenta FlashiXML.

A todos, muito obrigada.

Resumo

Uma parte considerável do desenvolvimento de software é dedicada à camada de interacção com o utilizador. Face à complexidade inerente ao desenvolvimento desta camada, é importante possibilitar uma análise tão cedo quanto possível dos conceitos e ideias em desenvolvimento para uma dada interface. O desenvolvimento baseado em modelos fornece uma solução para este problema ao facilitar a prototipagem de interfaces a partir dos modelos desenvolvidos.

O desenvolvimento baseado em modelos pressupõe a existência de linguagens de modelação adequadas. Na área de desenvolvimento de interfaces com o utilizador, surgiu o conceito UIDL (*User Interface Description Language*). As UIDLs permitem descrever uma interface a um alto nível de abstracção, independente dos detalhes de implementação. Ao longo do tempo têm vindo a ser propostas diversas linguagens, surgindo a UsiXML como uma das mais promissoras. Contudo, para que as UIDLs sejam verdadeiramente úteis, é necessário suporte ao nível de ferramentas, por exemplo, editores e ferramentas de prototipagem/animação de modelos.

É com base nestas evidências que foi criada a ferramenta FlexiXML, concebida para a interpretação e animação de interfaces com o utilizador descritas em UsiXML. Neste contexto, o presente trabalho descreve uma abordagem à prototipagem de interfaces, apresentado a ferramenta FlexiXML.

Palavras-chave: Prototipagem, linguagens de modelação de interfaces (UIDLs), UsiXML.

Abstract

A considerable part of software development is dedicated to the user interaction layer. Given the complexity inherent in the development of this layer, it is important to have an analysis as soon as possible of the concepts and ideas being used in the development a given user interface. Model-based development provides a solution to this problem by facilitating the prototyping of interfaces using the developed models.

Model-based development requires the existence of adequate modeling languages. Hence, user interface development is supported by User Interface Description Language (UIDL). These languages describe an interface at a high level of abstraction, independently of implementation details. Over time several languages have been proposed, UsiXML emerging as one of the most promising. However, for UIDLs be truly useful, tool support is necessary. For example, editors and tools for prototyping/animating the models.

It was on the basis of this evidence that the FlexiXML tool was created. FlexiXML is a tool designed for the interpretation and animation of user interfaces described in UsiXML. Hence, this thesis describes an approach to user interface prototyping by presenting the FlexiXML tool.

Keywords: Prototyping, User Interface Description Languages (UIDLs), UsiXML.

Índice

Índice de Figuras

The ideal system so buries the technology that the user is not even aware of its presence. The goal is to let people get on with their activities, with the technology enhancing their productivity, their power, and their enjoyment, ever the more so because it is invisible, out of sight, out of mind. People should learn the task, not the technology. They should be able to take the tool to the task, not as today, where we must take the task to the tool. And these tools should follow three axioms of design: simplicity, versatility, and pleasurability.

[Norman, 1988]

Capítulo 1

Introdução

Porque é que são importantes as interfaces com o utilizador?
Porque é que são tão difíceis de criar e implementar as interfaces homem-computador com qualidade?

Os computadores têm-se tornado quase indispensáveis para a maioria das funções quotidianas. A popularidade do computador, potenciada pelas facilidades de acesso à internet, tem conquistado uma enorme quantidade de diferentes tipos de público, que o utilizam como ferramenta indispensável para o seu dia-a-dia. Com isso, os utilizadores têm-se tornado mais exigentes relativamente aos sistemas interactivos que utilizam, fazendo com que as interfaces de computador se tornem cada vez mais importantes.

As primeiras interfaces eram simples diálogos baseadas na troca de caracteres alfanuméricos, no conhecido formato de texto verde sobre ecrã de fundo preto. Estas eram interfaces concebidas para o uso de especialistas ou de entusiastas que investiam grande parte do tempo a investigar o funcionamento destes sistemas. A necessidade de optimização das interfaces era óbvia, por um lado para facilitar a interacção com o computador e por outro para as tornar acessíveis a outros tipos de utilizadores.

De forma a tornar as interfaces mais consistente e mais fáceis de utilizar, iniciou-se uma fase de inovação nas tecnologias/metáforas de interacção. As interfaces tornaram-se graficamente mais ricas com o uso de janelas, ícones, botões, menus, características que apelavam para a memória visual do utilizador. Várias aplicações começaram a ter um formato similar baseado nestes controlos que se tornaram normas de facto [W10].

A história das interfaces gráficas remonta ao ano de 1970, quando os investigadores da Xerox Parc criaram o computador Alto, um dos primeiros a disponibilizar uma interface gráfica com o utilizador. Embora a Xerox tenha continuado a desenvolver as ideias testadas no Alto (com o Xerox Star em 1981), foi a Apple quem melhor consegui as conseguiu explorar comercialmente com a criação do microcomputador Lisa (1983). Seguidamente a Apple lançou os computadores Macintosh (1984), que serviram de inspiração para a próxima geração de interfaces. Em 1985 a Microsoft Corp. introduziu o Windows, como interface gráfica do sistema operativo MS-DOS. Estes sistemas popularizaram o uso das interfaces gráficas com o utilizador, fazendo surgir outros sistemas (Figura 1).

Nos últimos anos tem-se assistido a uma aumento considerável de diferentes tipos de computadores e dispositivos interactivos, como telemóveis, PDA's, WebTV, entre outros. Acontecimentos recentes apontam para uma revolução na área de interfaces com o utilizador, desde melhoria no desempenho de execução de tarefas até à possibilidade de manipulação de grande quantidade de informação. Na conferência TED 2006[1] foi apresentada uma interface manipulada pelo movimento gestual sobre o ecrã, interface sensível ao toque *multi-touch*. A Apple lançou o iPhone, que concilia telemóvel, iPod e internet em apenas um dispositivo. A Microsoft anunciou também o Microsoft Surface, dispositivo que permitir interacção com os utilizadores através das mãos. Contudo, a inovação tecnológica tornou-se de tal forma aliciante que as aplicações, bem como todo o ambiente, têm-se tornado cada vez mais complexos.

[1] **Conferência TED** – Conferência anual dedicada às áreas da tecnologia, entretenimento e design. (http://www.ted.com/index.php/pages/view/id/7)

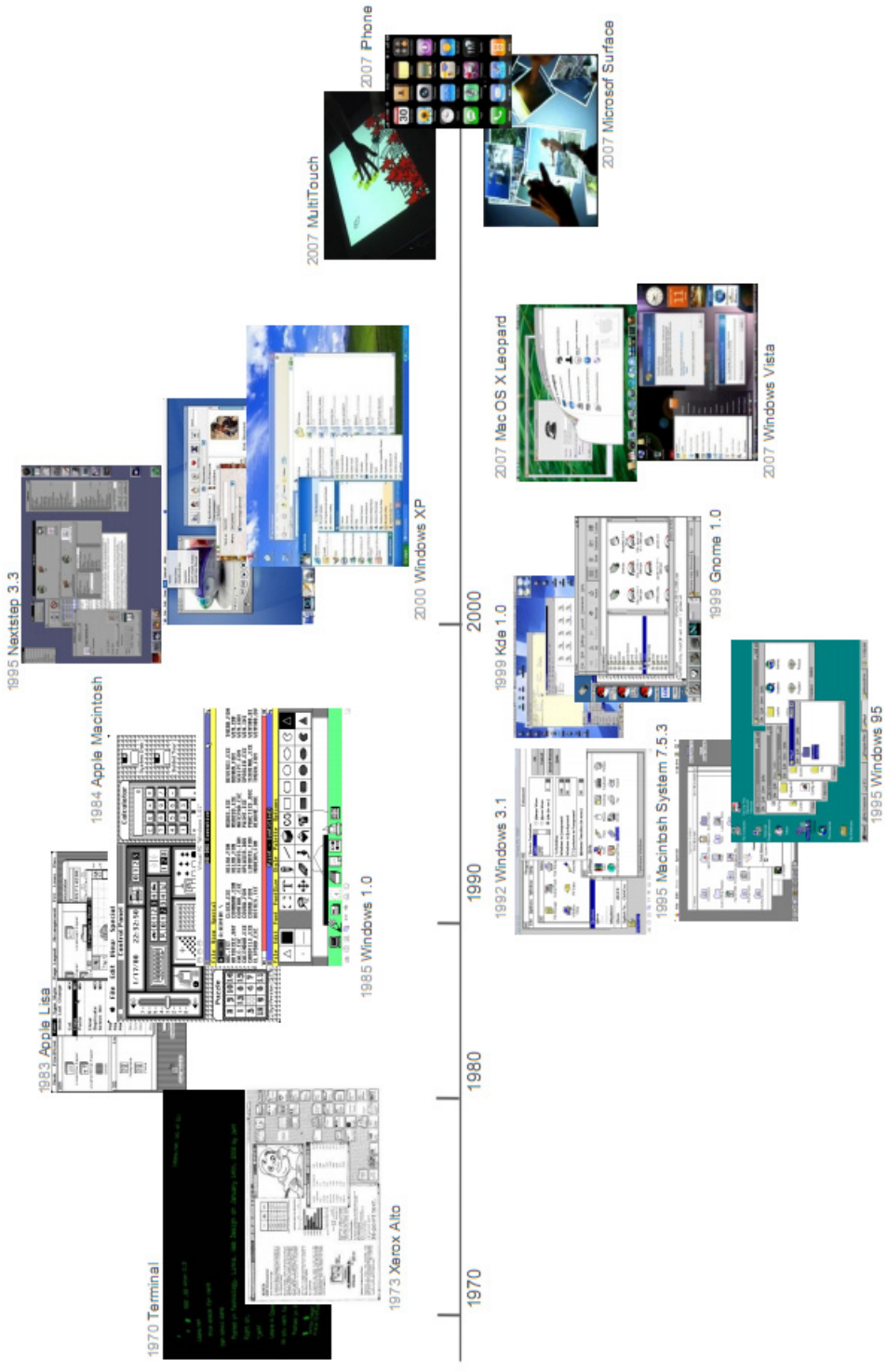


Figura 1 - Evolução das interfaces gráficas (simplificado).

1.1 A Interacção Humano-Computador

Estudos têm demonstrado que o sucesso dos sistemas interactivos depende, em grande medida, da sua usabilidade [W11]. Os utilizadores querem interagir eficientemente com o sistema a fim de tirar proveito da capacidade computacional, sem ser necessário dispensar muito tempo a aprender como o sistema funciona. Tal como afirmado por Norman [Norman, 1988] "Os utilizadores devem apreender a efectuar as tarefas e não a utilizar a tecnologia".

A inovação tecnológica, tornou-se de tal forma aliciante que as preocupações comuns dos designers de interfaces são criar tipos mais legíveis, integrar cor, som e voz, aspectos importantes, mas que são apenas uma parte do problema e não necessariamente a mais importante. A primeira preocupação deve ser a de melhorar o modo como as pessoas podem usar o computador para tirar o maior partido das suas potencialidades. Na verdade, numa fase inicial de lançamento de um produto, os utilizadores ficam fascinados pela inovação tecnológica, contudo, o que a grande maioria dos utilizadores procuram são sistemas que simplificam a execução das suas tarefas e que sejam fiáveis [Norman, 1999].

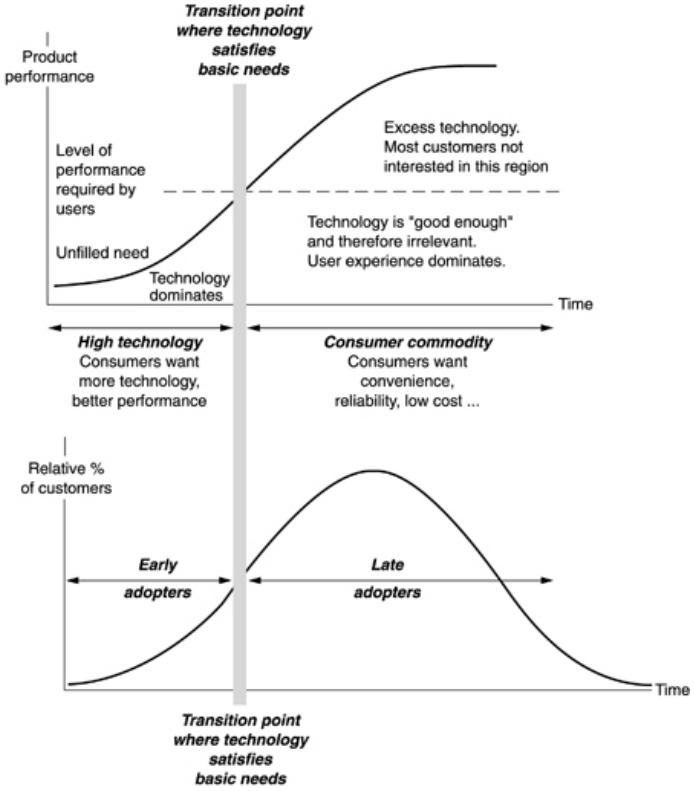


Figura 2 - Curva da procura do produto consoante a maturidade da tecnologia [Norman, 1999].

Segundo Norman (ver Figura 2) o desenvolvimento da tecnologia passa por um ciclo de vida que evolui do nascimento à maturidade. Neste ciclo as características da procura vão sendo alteradas consoante o tipo de consumidor, os quais ele denominou de "early adopters" e "late adopters". Os "early adopters" correspondem aos consumidores que ficam fascinados pela tecnologia e logo numa fase inicial querem adquirir o produto. Os "late adopters" representam a grande maioria dos consumidores, que esperam o amadurecimento da tecnologia para só depois a adquirirem.

Para que as interfaces sejam aceites e efectivamente utilizadas, o seu desenvolvimento tem de ser bem planeado. É necessário criar sistemas computacionais que justifiquem a confiança que a sociedade deposita sobre eles. A qualidade (i.e. usabilidade) dos sistemas interactivos depende, não só da concepção dos próprios sistemas, mas também dos utilizadores concretos que os vão utilizar e da tarefa que estes pretendem realizar (norma ISO 9241, [W12]). Trata-se assim de uma área multidisciplinar que abrange desde a engenharia de software até a psicologia cognitiva.

A investigação nesta área tornou-se um desafio e um foco de grande interesse, como consequência surgiu o termo **Interacção Humano-Computador** (**IHC**). Este termo surgiu na década de 80 para designar a área de conhecimento dedicada ao estudo do desenho e desenvolvimento de interfaces com o utilizador. Actualmente é consensual considerar que o objecto de estudo da IHC não se limita ao processo de desenvolvimento das interfaces com o utilizador, mas inclui a análise e concepção de todo o processo de interacção entre os utilizadores, o software e o contexto em que estes interagem [Dix, Finlay, D., & Person, 2004]. Esta área tem como principais objectivos criar sistemas usáveis, melhorar a usabilidade dos sistemas existentes e identificar problemas ou tarefas que podem ser abordados com software.

1.2 Desenvolvimento baseado em modelos

Actualmente, uma parte considerável do desenvolvimento de software é dedicada exclusivamente à camada de interacção com o utilizador. Os utilizadores têm-se tornado exigentes e descartam qualquer interface que seja difícil e desconfortável de usar pois, para eles, a interface é o próprio sistema [Silva, 2006].

Nem sempre os interfaces cumprem as reais necessidades do utilizador. A maioria das falhas detectadas em interfaces é decorrente das deficiências de comunicação entre os

profissionais de informática que as desenvolvem e os seus utilizadores finais. Este não é um problema recente, o mesmo ocorria na década de 70, quando Dervin e Nilan [Silva, 2006] afirmavam que o problema estava no planeamento dos sistemas, que obedecia à visão dos programadores, quando devia partir do conhecimento das reais necessidades, percepções, modelos mentais e estruturas de processamento da informação dos utilizadores.

Para minimizar este problema e detectar os erros o mais cedo possível no processo de desenvolvimento, o utilizador final deve participar no processo de desenvolvimento. A ideia é que o utilizador final possa acompanhar todo o processo para contribuir com as perspectivas e as funcionalidades que o software tem de incluir para dar resposta às suas necessidades.

Quando os erros são detectados numa fase avançada do projecto, mais complexa se torna a sua resolução, aumentando os custos de manutenção do mesmo. Apesar do reconhecimento de que a análise de usabilidade deve começar tão cedo quanto possível no processo de desenvolvimento, a verdade é que a separação entre IHC e Engenharia de Software não facilita esse objectivo. Nos últimos anos, tem vindo a ser desenvolvido esforços de aproximação entre estas áreas[2]. Estes esforços advêm do reconhecimento crescente do impacto negativo que falhas de usabilidade podem ter no sucesso de um sistema e da distância que ainda separa as metodologias e técnicas utilizadas em cada uma das disciplinas. Torna-se necessário estabelecer mecanismos de comunicação entre as duas áreas.

Neste contexto a utilização de modelos tem vindo a ser explorada. Os modelos são desenvolvidos de um ponto de vista da interacção entre o sistema, os seus utilizadores e o contexto que os rodeia e podem ser utilizados nas fases iniciais de desenvolvimento para ajudar a definir o sistema a ser implementado. Embora esteja a ser dado agora maior destaque aos modelos, esta abordagem teve origem na década de 80 do século passado. Nessa altura uns dos factores impulsionadores desta abordagem foram os UIMS (User Interface Management System), sistemas de gestão de interfaces. Os UIMS procuravam automatizar o processo de criação de interfaces gráficas. Dois exemplos de UIMS são: University of Alberta UIMS [Green, 1985]; DMS [Hartson, 1984]. Embora estes sistemas fossem promissores, a complexidade inerente à construção dos modelos para representação da interface não compensava comparativamente ao benefício obtido, o que fez com que a evolução destes sistemas acabasse por não acontecer [Dan R. Olsen, 1992].

[2] É de salientar os grupos de trabalho **IFIP WG 2.7/13.4** (International Federation for Information Processing Working Group, www.se-hci.org) e ***SIGCHI*** *(*Special Interest Group in Computers and Human Interaction*,* http://www.sigchi.org/).

Nos anos 80 não existia a diversidade que existe actualmente, com o aumento da diversidade de plataformas e dispositivos (telemóveis, PDA's, WebTV, etc.) surge a necessidade de disponibilizar a mesma aplicação na maior gama de dispositivos possível, para que os utilizadores possam aceder à informação, independentemente do dispositivo que usam ou mesmo quando o ambiente é alterado dinamicamente [Paternò & Santoro]. Em seguimento deste problema o termo UIMS surge novamente.

O desenvolvimento baseado em modelos fornece uma solução para os problemas identificados acima, relacionados com a comunicação entre IHC e a Engenharia de Software e com a necessidade de disponibilizar aplicações em diferentes plataformas. Neste paradigma, são criados vários modelos declarativos que descrevem as tarefas que o utilizador pode efectuar no sistema, as capacidades funcionais, o estilo/requisitos do interface, as características/preferências do utilizador e as técnicas de I/O suportadas pela plataforma utilizada. Uma abordagem baseada em modelos favorece um desenvolvimento mais sustentado. Os modelos permitem a construção de protótipos, o que ajuda os profissionais informáticos a compreenderem uma funcionalidade e facilita a participação dos utilizadores finais.

No contexto de desenvolvimento baseado em modelos e na área de interfaces gráficas com o utilizador, não pode deixar de ser focada a importância das UIDLs (User Interface Description Language). Uma UIDL consiste numa linguagem de alto nível que permite descrever as características relevantes de uma interface gráfica com o utilizador. Nesta área várias linguagens têm vindo a ser criadas, em que cada aborda diferentes aspectos de uma *interface*. Na secção 2.1 são apresentadas algumas UIDLs.

Para que as UIDLs sejam verdadeiramente úteis, é necessário suporte ao nível de ferramentas, por exemplo, editores e ferramentas de prototipagem/animação de modelos. No seguimento dos problemas referidos anteriormente este relatório descreve o FlexiXML, um animador capaz de realizar a prototipagem da interface com o utilizador a fim de permitir testes com o mesmo. Não se pretende criar a interface final, definitiva, mas sim exibir apresentações que podem ser testadas e iterativamente adaptadas aos utilizadores. A prototipagem e teste não garantem a ausência de erros, mas minimizam a propensão dos mesmos em fases avançadas do projecto e permitem avaliar a reacção do utilizador à interface desenvolvida.

1.3 Objectivos

Como referido anteriormente a construção de interfaces com o utilizador é um processo complexo. Esta complexidade deve-se ao conjunto alargado de aspectos a considerar: seja a complexidade/diversidade dos ambientes de desenvolvimento e execução, seja a quantidade de conhecimentos de programação e de engenharia de usabilidade necessários. Estas dificuldades aumentam quando a mesma interface gráfica necessita de estar disponível em múltiplos contextos. A noção de contexto caracteriza diferentes perfis de utilizador (com diferentes preferências, idiomas ou mesmo com diferentes capacidades cognitivas e físicas) e diferentes plataformas e dispositivos (telemóveis, PDA's, WebTV, etc.). Torna-se difícil construir a mesma interface para múltiplos contextos, sem criar várias versões do mesmo consoante o contexto a ser utilizado.

Paralelamente a este problema verifica-se que nem sempre as interfaces satisfazem as reais necessidades dos utilizadores. Para minimizar este problema e detectar os erros o mais cedo possível no processo de desenvolvimento, o utilizador final deve participar no processo de desenvolvimento. O objectivo é que o utilizador final possa acompanhar todo o processo contribuindo com as suas perspectivas sobre a melhor forma de o software dar resposta às necessidades dos utilizadores. É neste contexto que FlexiXML surge, como uma ferramenta de prototipagem de interfaces gráficas.

A ferramenta desenvolvida utiliza como UIDL de suporte a UsiXML [W2] . Trata-se de uma linguagem de modelação que permite especificar os aspectos essenciais de uma interface. A escolha desta linguagem deveu-se ao facto de se tratar de uma linguagem muito abrangente, que permite especificar diferentes aspectos da interface, dos quais se podem salientar: o seu aspecto gráfico, a interacção com os utilizadores e contexto de execução. Embora nesta versão do projecto apenas sejam utilizadas algumas das potencialidades da linguagem, o facto de esta abordar diferentes áreas, permite que este projecto possa evoluir para focar outros aspectos (por exemplo, interface multimodais).

A razão pela escolha deste projecto deve-se a vários factores. Por um lado, o interesse da autora pela área de interface gráficas com o utilizador, uma vez que se trata da área em que está profissionalmente alocada, e como tal pretender aprofundar os seus conhecimentos sobre a mesma. Para além do interesse estritamente profissional, tinha já sido realizado trabalho de investigação relacionado com verificação de modelos e prototipagem [Guerreiro, 2008], o que aumentou o interesse para continuar a investir nesta área.

Por outro lado, embora a UsiXML disponha de várias ferramentas especializadas em diferentes aspectos da linguagem, não dispõe de uma ferramenta actualizada que permita a animação de modelos, por forma a facilitar um processo de prototipagem e análise rápida. A ferramenta que mais se aproxima do pretendido é FlashiXML [Youri, 2004]. No entanto esta ferramenta não está adaptada à versão actual de UsiXML.

Neste projecto, pretende-se criar uma ferramenta compatível com a versão actual de UsiXML e com funcionalidades adicionais às disponibilizadas por FlashiXML. Os principais objectivos a atingir são:

- Interpretação da versão mais recente de UsiXML;
- Implementação da ferramenta numa versão mais recente de tecnologia (AIR, Flex3, ActionScript3);
- Adicionalmente, pretende-se aproveitar a oportunidade para criar uma aplicação genérica e expansível.

1.4 Estrutura da Dissertação

Além do presente capítulo, esta dissertação encontra-se organizada em cinco outros capítulos. No próximo capítulo aborda-se o tema das UIDLs e é efectuada uma descrição da linguagem UsiXML, a linguagem por omissão da ferramenta FlexiXML. No capítulo 3 são apresentadas as principais ferramentas, já existentes, que dão suporte a USIXML. O capítulo 4 consiste na apresentação da ferramenta FlexiXML, sendo exposto na secção 5 um exemplo de uma aplicação gerada por esta ferramenta. Finalmente, no capítulo 6, é efectuada a análise dos resultados do trabalho realizado, apontando-se conclusões e propostas de trabalho futuro.

"How do you get team members to start speaking the same language? Constant communication between the teams. What also works is adopting a common language to describe interactions. The use of design patterns is a powerful way to disseminate common thinking and approaches to common problems. I have been pleasantly surprised when language that design and engineering use to describe certain bad approaches (anti-patterns) gets in-grained even in our product managers vocabulary as well." [W1]

(Future Practice Interview: Bill Scott)

Capítulo 2

Estado da arte

Problemas de comunicação entres os diferentes intervenientes no desenvolvimento de interfaces, o aumento da complexidade e da diversidade dos dispositivos a que as mesmas podem ser aplicadas, são alguns dos aspectos que fazem com que este desenvolvimento seja um processo complexo e susceptível a falhas. Como Bill Scott referiu na Future Practice Interview [W1] a melhor forma de comunicação é a utilização de uma linguagem comum entre todos os intervenientes, que permita descrever correctamente o tipo de trabalho que se efectua.

As UIDL surgem como resposta à necessidade de modelação na área de desenvolvimento de sistemas interactivos. Neste capítulo apresenta-se a revisão do estado da arte ao nível de UIDLs, sendo dada particular atenção a UsiXML.

2.1 Linguagens de modelação de interfaces

Uma UIDL (User Interface Description Language) consiste numa linguagem para descrever as características relevantes de uma interface gráfica com o utilizador a um alto nível de abstracção. Este tipo de linguagem não exige conhecimentos de programação o que permite que analistas, *designers*, programadores e utilizador final a possam utilizar durante o ciclo de desenvolvimento.

Existem várias UIDLs que abordam diferentes aspectos de uma interface com o utilizador: portabilidade, independência de dispositivos, suporte a múltiplas plataformas, entre outros.

Actualmente existem várias UIDLs em utilização . Neste capítulo apenas referenciadas as linguagens mais representativas encontradas na literatura, sendo dado especial destaque a UsiXML, a linguagem utilizada neste projecto.

UIML (User Interface Markup Language) [Abrams, 1999] é uma meta-linguagem derivada do XML para descrição de interfaces com utilizador. Esta linguagem permite aos *designers* descrever a interface de forma genérica e usar *styleSheets* para transformar para outras linguagens, diferentes dispositivos e para vários sistemas operativos. Um documento UIML está divido em três partes: a descrição da interface; uma secção de mapeamento entre o documento UIML e entidades externas (plataforma, lógica aplicacional); e um *template* para reutilização de elementos já criados.

Nesta linguagem uma interface de utilizador é um conjunto de elementos com o qual o utilizador interage. Cada elemento recebe o seu conteúdo (texto, imagem, etc.), eventos que podem ser despoletados e respectivas acções a efectuar.

É necessário um *renderer* específico para cada sistema operativo e tipo de interface (HTML, WAP, Java, etc.). Relativamente aos dispositivos é necessário o desenvolvimento em separado da interface para cada dispositivo.

XIML (eXtensible Interface Markup Language) [Puerta, 2002] permite descrever a interface independentemente dos detalhes de implementação. O objectivo desta linguagem é descrever os aspectos abstractos de uma interface (tarefas, domínios e utilizador) e aspectos concretos (apresentação e diálogo) ao longo do ciclo de desenvolvimento.

XIML contém cinco componentes na definição da interface: componente de tarefas, para definir as tarefas que o utilizador pode efectuar com a interface; componente de domínio, que inclui os objectos que a compõem; componente de utilizador, que define as características do tipo de utilizador; componente de diálogo, que define a interacção com a interface; e componente de apresentação. Para além destes componentes, XIML possui também atributos para definir as propriedades de um componente e relações para criar ligações entre eles.

Teresa XML [Paternó, 2002], linguagem XML desenvolvida pelo projecto Teresa, o qual está integrado num projecto Europeu (Cameleon IST). [Souchon, 2003]

Teresa permite desenhar e gerar uma interface gráfica para uma plataforma específica. A linguagem é composta por duas partes: a notação CTT, linguagem para descrição do modelo de tarefas; linguagem e uma notação para descrição da interface.

Esta linguagem é utilizada na ferramenta TERESA, a qual permite geração no modelo de tarefas, interfaces abstractas e execução das interfaces.

WSXL (Web Service Experience) [Abrams, 1999] é um modelo para aplicações Web interactivas. Os objectivos desta linguagem são auxiliar o processo de criação de aplicações Web para uma grande variedade de canais e permitir a criação de aplicações Web a partir de outras. Esta linguagem pretende ser independente da plataforma de execução, *browser*, e linguagens de apresentação.

Para além destas UIDLs poderiam ser mencionadas outras, como: AUIML (Abstract User Interface Markup Language) [Argollo, 1997]; XUL (Extensible User Interface Language) [W7]; AAIML [Zimmermann, 2002]; Seescoa XML (Software Engineering for Embedded System using a Component Oriented Approach) [Luyten, 2002]; TADEUS-XML ([Muller, 2001]); entre outras. A maioria destas linguagens já é suportada por ferramentas para criação e interpretação das especificações, alguns exemplos são: Teresa, Uiml.NET, Liquid UI, apenas para enumerar algumas.

2.2 UsiXML

A USer Interface eXtensible Markup Language (UsiXML) consiste numa linguagem para descrição de interfaces com o utilizador (UIDL) que permite a especificação da mesma de forma independentemente da linguagem de programação e plataforma computacional em que irá ser executada[3].

Existem várias UIDLs que abordam diferentes aspectos de uma UI (Secção 2.1): portabilidade, independência de dispositivos, suporte a múltiplas plataformas, entre outros. Não existe porém muitas UIDLs que explorem simultaneamente aspectos necessários para as comuns

[3] UsiXML é já utilizada em mais de 20 organizações. [W4]

interfaces gráficas com o utilizador (GUIs) e para interfaces multimodais. A USIXML surgiu para dar respostas a este desafio.

Esta linguagem capta a essência da uma interface independentemente das características físicas. Isto significa que uma aplicação com diferentes tipos de técnicas de interacção e diferentes plataformas computacionais pode ser descrita de forma a preservar o seu *design*. É assim possível descrever uma UI em múltiplos contextos de utilização, como *Character User Interfaces* (CUIs), *Graphical User Interfaces* (GUIs), *Auditory User Interfaces* e *Multimodal User Interfaces*.

O desenvolvimento da linguagem obedeceu à intenção de suportar um processo de desenvolvimento com as seguintes características [Vanderdonckt et al.]:

- **Expressividade da UI –** qualquer interface é expressa dependendo de um contexto de uso (tipo de utilizador, plataforma e ambiente computacional) recorrendo a um conjunto de modelos que podem ser interpretados e manipulados por um software;
- **Repositório central de modelos –** Cada modelo é guardado num repositório de modelos onde todos são definidos segundo a mesma UIDL;
- **Abordagem transformacional –** Cada modelo pode ser sujeito a uma ou mais transformações que apoiam as várias etapas de desenvolvimento;
- **Múltiplos caminhos de desenvolvimento –** Etapas de desenvolvimento podem ser combinadas de forma a obter caminhos compatíveis com os requisitos de uma organização, convenções e contexto de utilização;
- **Abordagem de desenvolvimento flexível –** Abordagens de desenvolvimento (ex: *top-down*, *bottom-up*, *wide spreading* e *middle-out*) são suportadas pela flexibilidade de mudar entre caminhos alternativos o que permite aos designers seguir diferentes direcções consoante as modificações impostas pelo contexto de uso.

2.2.1 Níveis de abstracção

UsiXML permite descrever uma *interface* a vários níveis de abstracção. A organização dos diferentes níveis de abstracção é inspirada na *Framework Cameleon* (**C**ontext **A**ware **M**odelling for

Enabling and **L**everaging **E**ffective interacti**ON**) [W8], que define etapas de desenvolvimento para aplicações interactivas com vários contextos. A estrutura desta *Framework* é composta pelas seguintes camadas (Figura 3):

- **Final UI (FUI)** – Corresponde à interface operacional, ou seja, a interface que pode ser executada ou interpretada em um determinado contexto de uso (numa plataforma computacional específica e com um conjunto de dispositivos específicos, utilizando objectos de interacção específicos). Os artefactos desta camada são o código fonte que implementa a linguagem (Java, HTML, etc.) e os executáveis que efectuam o *render* da interface;
- **Concrete UI (CUI)** – Especificação da interface com o utilizador em termos de objectos gráficos de interacção e as suas relações. Efectua uma abstracção da interface que é independente da plataforma computacional;
- **Abstract UI (AUI)** – Descrição abstracta da interface que é independente do modelo de interacção (interacção gráfica, interacção vocal, …);
- **Tasks & Concepts** – Descreve as tarefas a serem efectuadas e os principais conceitos necessários para que sejam executadas. Estes objectos são considerados instâncias das classes que representam os objectos manipulados.

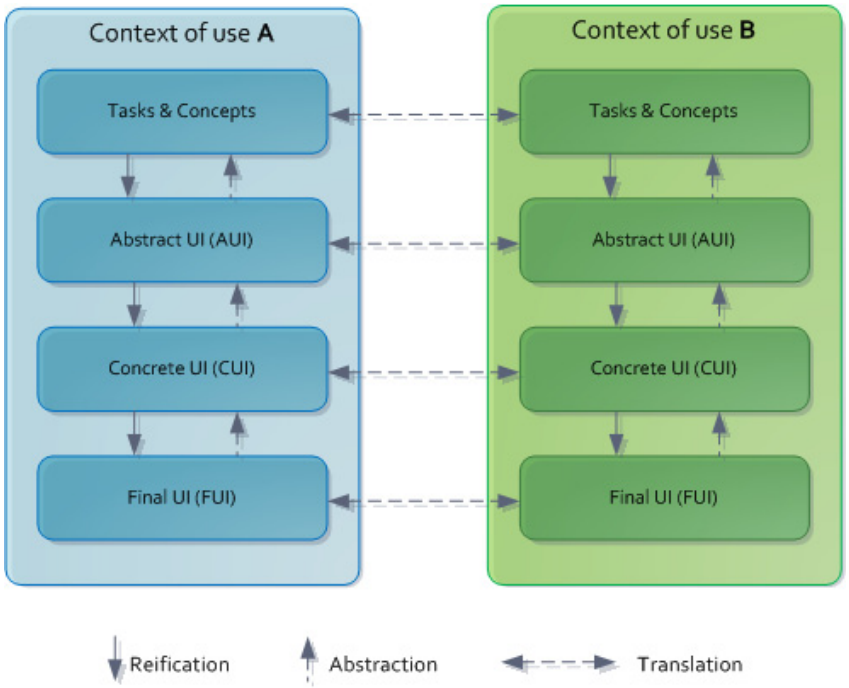


Figura 3 - *Framework* Cameleon (adaptado de [Youri, 2004]).

A *Cameleon Reference Framework* define que as etapas de desenvolvimentos podem ser realizadas através de transformações, as quais podem ter duas direcções: transformações verticais e transformações horizontais.

As transformações verticais correspondem a processos para conversão de um modelo de interface de um nível mais concreto para um mais abstracto e vice-versa, e incluem os seguintes tipos de transformações:

- **Reificação** – transformação de um nível de descrição mais abstracto para um nível mais concreto;
- **Abstracção** – transformação de um nível de descrição mais concreto para um nível mais abstracto.

As transformações horizontais permitem transformar um modelo dentro de um mesmo nível, mas entre contextos de usos diferentes, e incluem a transformação:

- **Tradução** – transformação entre etapa de desenvolvimento similar mas de contexto de uso diferente.

Com este paradigma de desenvolvimento é possível especificar uma interface em qualquer um dos níveis de abstracção e transformá-la para qualquer um dos outros níveis (recorrendo a ferramentas UsiXML, como exposto no Capítulo 3). A especificação da interface pode ser efectuada independentemente do contexto e/ou condições de interacção sendo possível obter uma ou mais interfaces gráficas para os diferentes contextos e nos diferentes níveis de abstracção. Este modelo de transformação, equiparável à *Model-Driven Architecture* (MDA) da engenharia de software, é suportado utilizando os tipos de transformações acima referidos: abstracção, reificação e tradução [Eng, 2005].

A *Model-Driven Architecture* consiste em descrever um sistema software através de modelos a partir dos quais pode ser criado o sistema final. Durante o processo de desenvolvimento de um projecto, os requisitos do utilizador podem ser alterados e os modelos associados devem ser capazes de se adaptar a essas alterações. Ou seja, a arquitectura a utilizar deve ser capaz de criar novos sistemas e adaptar e/ou melhorar sistemas existentes. [W4] Na *Model-Driven Architecture* estão definidas três fases no processo de desenvolvimento até se obter o código final [Santa et al.]:

- **CIM (Computing Independent Model)** – Modelo de especificação de alto nível, com objectivo de modelar os pré-requisitos do sistema;

- **PIM (Platform Independent Model)** – Modelo sobre a área operacional do sistema, mas sem detalhe sobre a implementação nas diversas plataformas;
- **PSM (Platform Specific Model) –** Modelo que combina os dois modelos anteriores, contém a informação para utilização do sistema numa plataforma específica.

Na Figura 4 está representado um paralelismo entre a abordagem MDA e a técnica proposta pela linguagem UsiXML (estrutura Cameleon). A ferramenta FlexiXML (descrita no Capítulo 4) situa-se na área de renderização (passagem de um modelo do tipo "*Concrete user interface*" para a apresentação/simulação da interface gráfica final).

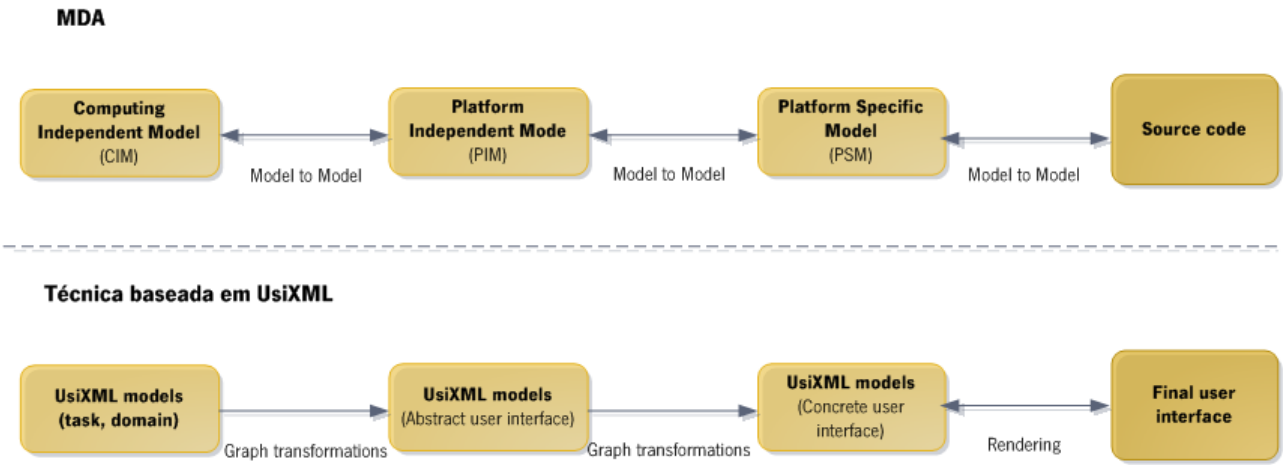


Figura 4 - MDA e UsiXML [Eng, 2005].

2.2.2 Descrição da linguagem

Uma *User Interface Description Language* (UIDL) consiste numa linguagem para descrever as características de interesse de uma interface com o utilizador. Como linguagem, tem uma sintaxe (expressões que podem ser escritas na linguagem) e semântica (significado das expressões) bem definidas.

A semântica da linguagem UsiXML é definida por um modelo descrito por um diagrama de classes UML (Figura 5). A sintaxe da linguagem UsiXML é definida por *schemas* XML que resultam da conversão do diagrama de classe UML. A sintaxe é assim definida por um conjunto de *schemas* de XML, em que cada *schema* corresponde a um dos tipos de modelo existentes na linguagem.

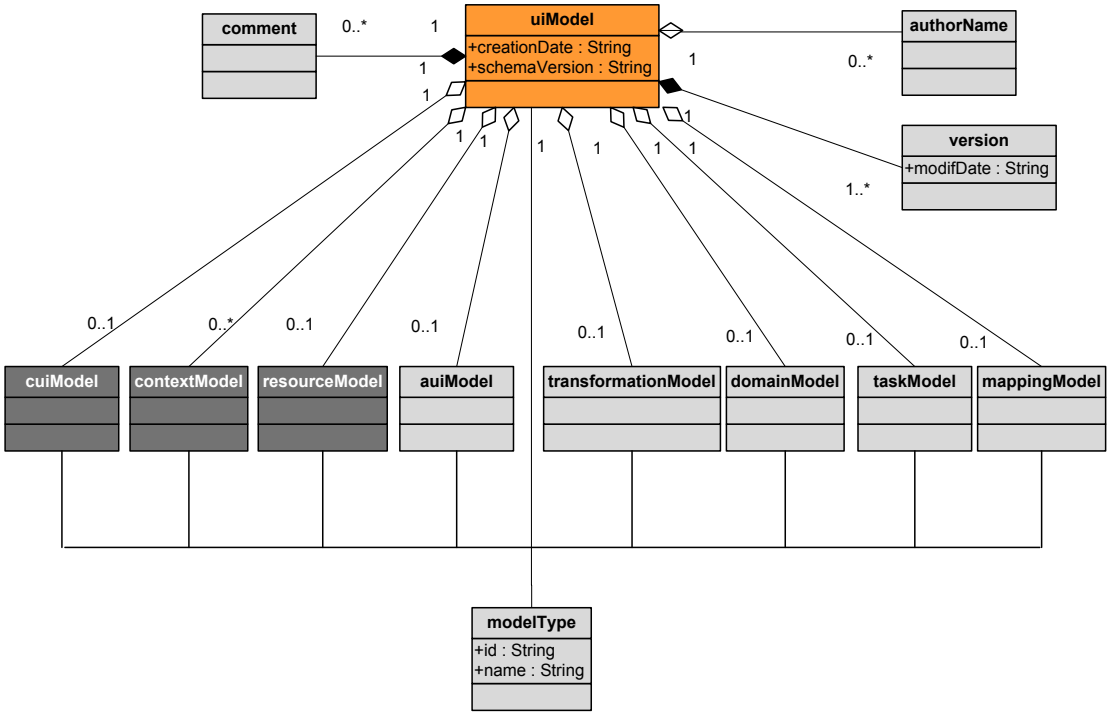


Figura 5 - Diagrama de classes da linguagem UsiXML.

Para uma melhor compreensão da semântica da linguagem a explicação da Figura 5 será acompanhada por um exemplo básico de utilização. O exemplo consiste numa aplicação para a apresentação de imagens (Figura 6). A aplicação permite a visualização de imagens e apresenta um botão que permite navegar para a imagem seguinte.

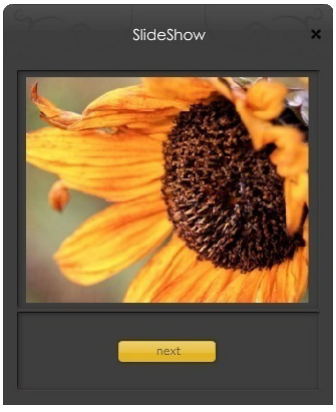


Figura 6 - SlideShow.

UiModel (User Interface Model)

O **UiModel** correspondente ao modelo central de especificação da *interface* gráfica. Este componente contém as características comuns a todos os modelos, como a versão, autor, data de criação, entre outros (Figura 7).

O modelo *uiModel* é por sua vez composto por vários modelos, que na descrição de uma interface não necessitam de ser incluídos na totalidade e quando utilizados podem ser definidos em qualquer ordem.

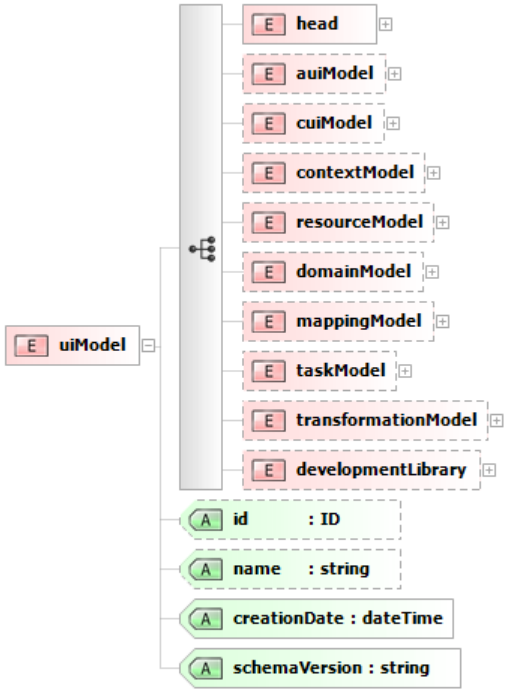


Figura 7 - *Schema* do *UIModel*.

O uiModel é composto pelas seguintes propriedades:

- **id** – Identificador do ficheiro;
- **name** – Nome do projecto;
- **creationDate** – Data de criação;
- **schemaVersion** – Versão do usiXML.

No Exemplo 1 está a apresentado o início do modelo uiModel da aplicação SlideShow.

Exemplo 1 - Especificação do modelo *uiModel*.

```
<uiModel xmlns="http://www.usixml.org"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.usixml.org/
http://www.usixml.org/spec/UsiXML.xsd"
    id="slideShow" name="SlideShow"
    creationDate="2009-05-16T13:39:23.366+02:00" schemaVersion="1.8.0">
  …

</uiModel>
```

Embora o *uiModel* contenha vários modelos, no caso específico do interpretador *FlexiXML* apenas são necessários: o *cuiModel*, o *contextModel* e o *resourceModel* (Figura 5). Estes são os 3 modelos que contêm a informação relativa ao desenho da interface gráfica e à interacção com o utilizador final.

CUIModel (Concrete User Interface Model)

O **CuiModel** especifica o conteúdo e o *layout* da interface, ou seja, define os objectos que compõe a interface gráfica (CIO - Concrete Interaction Objects) e as relações entre eles (*graphicalTransition*) (ver Figura 8). Nas próximas secções estes dois conceitos (CIO e *graphicalTransition*) são apresentados em detalhe.

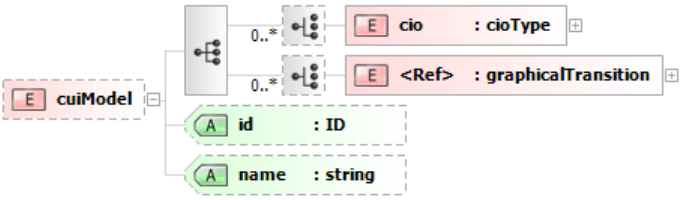


Figura 8 - *Schema* do modelo *CUIModel*.

No exemplo da aplicação de visualização de imagens, a interface gráfica pode ser decomposta em vários objectos (CIO's), como apresentado na Figura 9.

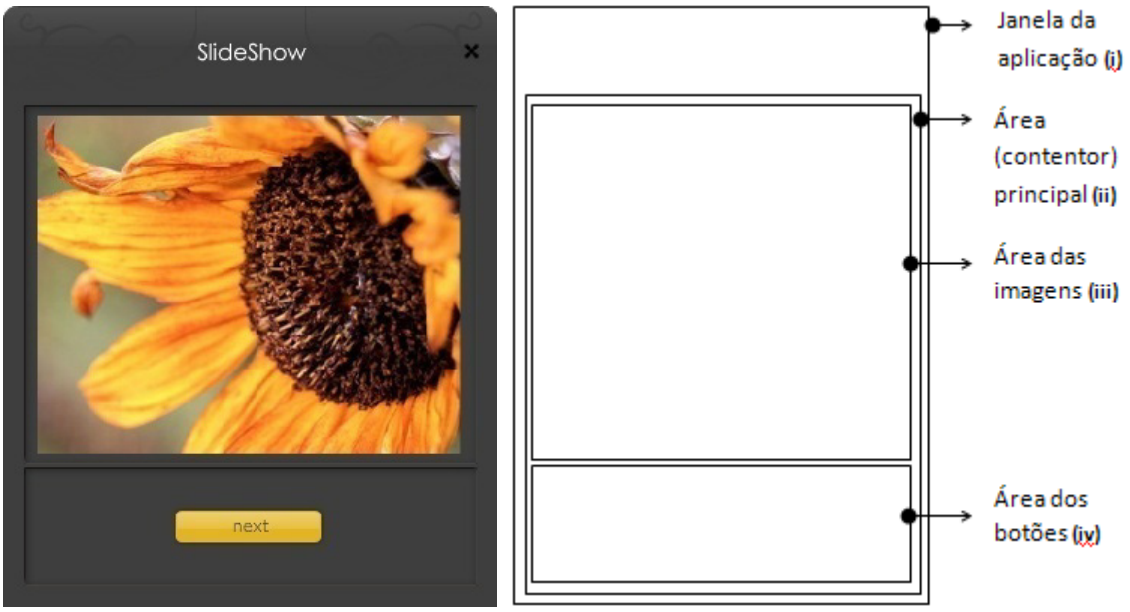


Figura 9 - Decomposição da aplicação "SlideShow" em áreas (ver correspondência com Exemplo 2).

Exemplo 2 - Especificação do modelo *cuiModel* para a aplicação "SlideShow" (contém referências à Figura 9).

```
<cuiModel id="slideShow-cui" name="slideShow-cuiModel">
      <window …> (i)
            <box …> (ii)
                  <box …> (iii)
                        <imageComponent id="I0" …/>
                        <imageComponent id="I1" …/>
                        <imageComponent id="I2" …/>
                  </box>
                  <box …> (iv)
                        <button id="B0" …/>
                        <button id="B1" …/>
                        <button id="B2" …/>
                  </box>
            </box>
      </window>
</cuiModel>
```

CIO – Concrete Interaction Objects

Os objectos **CIO** correspondem às entidades que o utilizador pode visualizar e/ou manipular, são uma abstracção dos elementos gráficos: botões, imagens, texto, etc.

Embora a linguagem UsiXML suporte vários elementos gráficos específicos, como: *DialogBox*, *DatePicker*, *ColorPicker*, *FilePicker*, nesta descrição são abordadas as características mais comuns de um elemento gráfico[4]. São elas (Figura 10):

- **id** – Identificador;
- **name –** Nome dado ao CIO. O nome pode fornecer alguma informação sobre o seu tipo, função ou conteúdo;
- **icon –** Localização do ícone associado;
- **defaultIcon –** Ícone por omissão;
- **content** – Conteúdo;
- **defaultContent** – Conteúdo por omissão;
- **help** – Ajuda textual;
- **defaultHelp** – Ajuda por omissão;
- **currentValue** – Valor actual;
- **feedBack** – Mensagem a apresentar quando um *input* é correctamente preenchido;
- **isMandatory** – Indica se o CIO é de preenchimento obrigatório.

[4] Existe mais informação sobre a linguagem em http://www.usixml.org/.

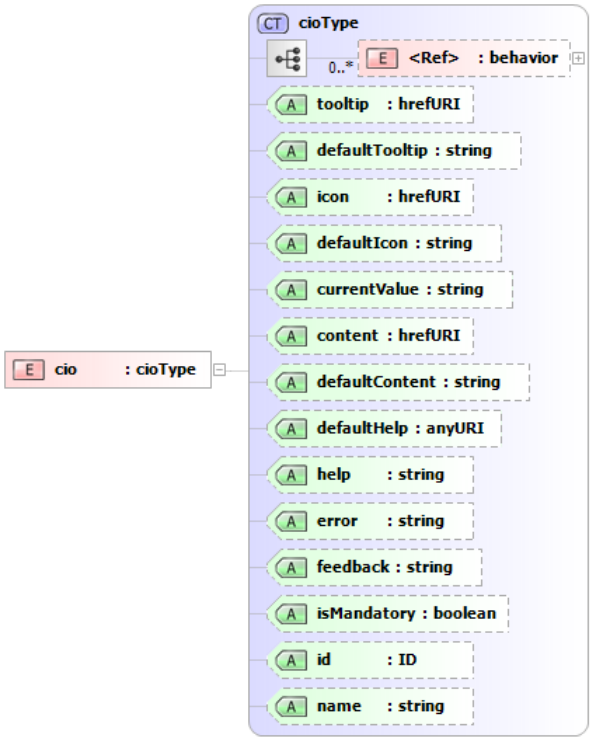


Figura 10 - *Schema* de um CIO.

Exemplo 3 - Especificação de um botão.

```
<button id="B0" name="B0"
      width="100" height="25"
      isVisible="true"
      defaultContent="next"
      defaultTooltip="Press button to see next image"/>
```

No Exemplo 3 está apresentado a especificação de um botão. O botão do exemplo tem definido um identificador ("B0"), uma dimensão, deve estar visível, tem definida uma mensagem ("next") e tem uma *tooltip* associada ("Press button to see next image"). O resultado de exemplo está apresentado na Figura 11.

Figura 11 - Visualização de um botão (Exemplo 3).

É também de salientar que um objecto CIO pode ter associado um comportamento, ***(behavior)*** (Figura 12). Um comportamento corresponde às acções que um CIO deve efectuar quando o utilizador interage com o mesmo. A interacção do utilizador com o CIO gera eventos que despoletam a execução das acções.

Um evento (***event***) é definido pelas seguintes propriedades (Figura 12):

- **id** – Identificador;
- **eventType** – Tipo de evento, por exemplo: *click* sobre um botão, *rollover* sobre um botão, mudança do conteúdo de uma caixa de texto, etc.;
- **eventContext** – Contexto do evento. Consoante o tipo de evento, esta propriedade pode conter o identificador do CIO a que o evento se aplica.

Uma acção (***action***) pode corresponder à chamada de um método (*method*) ou a uma mudança na interface gráfica (*uiChange*). Este modelo é caracterizado por (Figura 12):

- **id** – Identificador;
- **name** – Nome da acção;
- **description** – Descrição da acção.

Um método é composto apenas pelo nome do nome do método e pode ter associado uma lista de parâmetros.

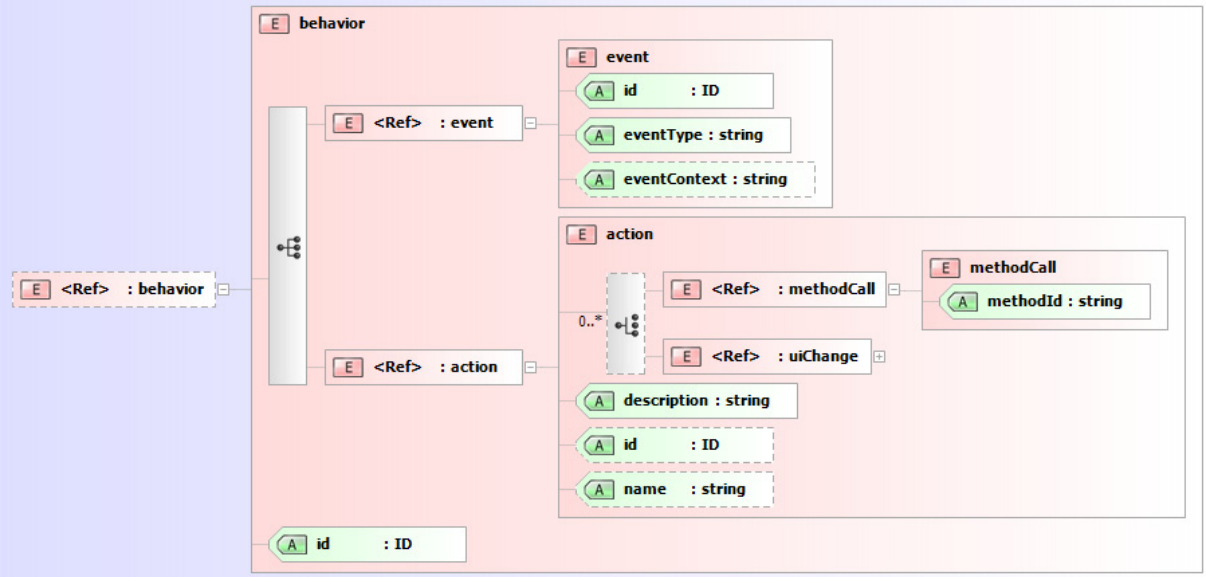


Figura 12 - *Schema* de um modelo para especificação do comportamento de um CIO (*behavior*).

Exemplo 4 - Especificação do comportamento (*behavior*) de um botão.

```
<button width="100" height="25" isVisible="true" id="B0"
defaultContent="next" name="B0">
      <behavior id="behav0">
            <event id="evt0" eventType="depress" eventContext="B0" />
            <action id="act0">
                  <transition transitionIdRef="Tr01" />
                  <transition transitionIdRef="Tr02" />
                  <transition transitionIdRef="Tr03" />
                  <transition transitionIdRef="Tr04" />
            </action>
      </behavior>
</button>
```

No Exemplo 4 está especificado o comportamento do botão "BO" quando este é pressionado (eventType do tipo "depress"). Quando este evento é despoletado é executada a acção "act0" a qual inclui quatro transições (Tr01, Tr02, Tr03, Tr04). De seguida é definido o conceito de transição.

Uma mudança gráfica pode corresponder a transições entre componentes. Por exemplo, efeitos de *fadeIn*, *fadeOut*, abrir, fechar, etc. Uma transição gráfica é definida por (Figura 13):

- **transitionType** – Tipo de transição;
- **transitionEffect** – Tipo de efeito a aplicar na transição*: fadeIn, fadeOut, wipe, box in, box out, dissolve, split, etc;*

- **Identificação dos componentes sobre a qual a transição vai ser efectuada (*CUIRelationShip*):**
  - **id** – Identificação da relação;
  - **name** – Nome da relação;
  - **sourceId –** Identificador do componente a partir do qual a transição vai ter início;
  - **targetId –** Identificador do componente onde a transição vai terminar.

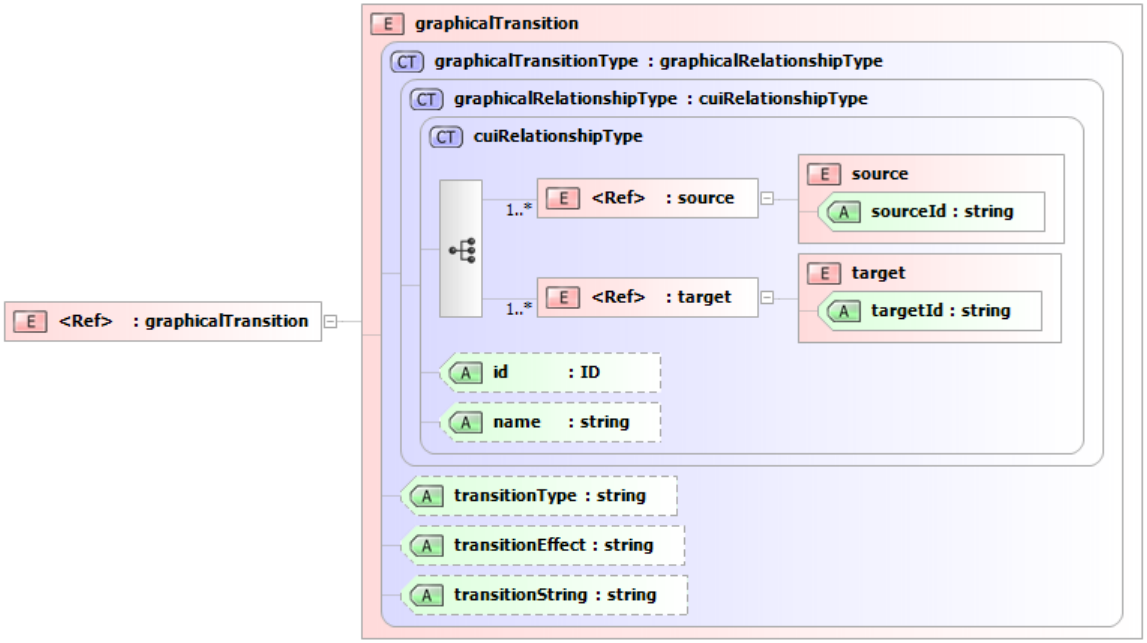


Figura 13 - *Schema* de um modelo para especificação de uma transição gráfica.

Exemplo 5 - Especificação de uma transição gráfica (*graphicalTransition*).

```
<graphicalTransition id="Tr01" transitionType="fadeOut">
      <source id="B0" />
      <target id="I0" />
</graphicalTransition>
<graphicalTransition id="Tr02" transitionType="fadeIn">
      <source id="B0" />
      <target id="I1" />
</graphicalTransition>
<graphicalTransition id="Tr03" transitionType="close">
```

```
      <source id="B0" />
      <target id="B0" />
</graphicalTransition>
<graphicalTransition id="Tr04" transitionType="open">
      <source id="B0" />
      <target id="B1" />
</graphicalTransition>
```

No Exemplo 5 estão definidas quatro transições entre componentes (este exemplo contém as definições das transições que são invocadas pelo botão do Exemplo 4):

- A transição Tr01 é uma transição do tipo *fadeOut* sobre a imagem I0 e é despoletada pelo botão B0;
- A transição Tr01 é uma transição do tipo *fadeIn* sobre a imagem I1 e é despoletada pelo botão B0;
- A transição Tr03 é uma transição do tipo *close* sobre o botão B0 e é despoletada pelo botão B0;
- A transição Tr04 é uma transição do tipo *open* sobre o botão B1 e é despoletada pelo botão B0.

No Figura 14 está apresentado resultado da especificação do Exemplo 4 e Exemplo 5.

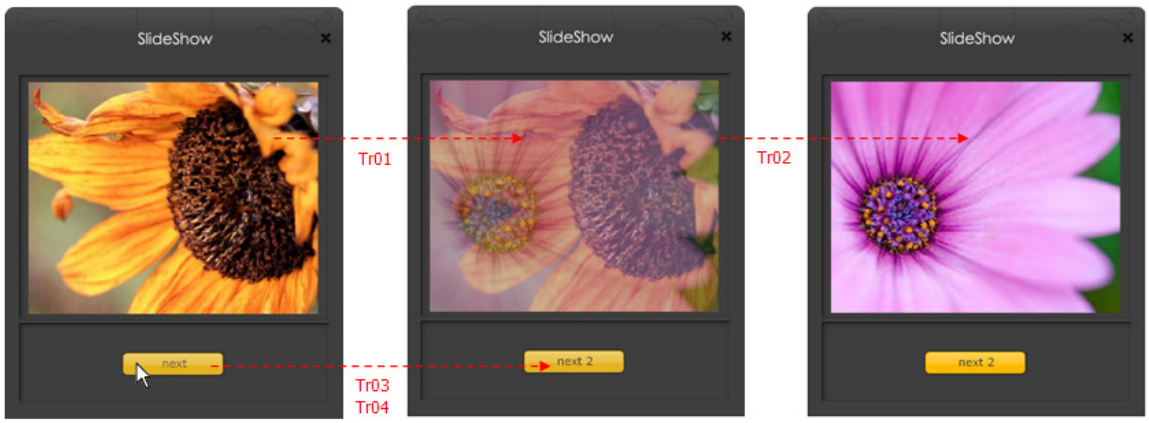


Figura 14 - Transição gráfica entre componentes (Exemplo 4 e Exemplo 5).

ContextModel

O **ContextModel** (Figura 15) é o modelo que descreve o contexto de utilização da aplicação, desde o tipo de utilizador à plataforma e ambiente computacional.

De seguida, é apresentada uma breve descrição dos diferentes tipos de contextos:

- **Plataforma (*platform*)** – Contém as características mais relevantes da plataforma de software, hardware e dispositivos anexados que influenciem a execução das tarefas do utilizador. A plataforma pode consistir numa série de dispositivos (componente *hardware platform*), numa série de componentes de software (*software platform*), características da rede a que a plataforma está conectada, capacidade de suporte de *wireless* (características WAP) e a capacidade de *browser*;
- **Ambiente (*environment*)** – Descreve as propriedades do ambiente em que o utilizador está a utilizar a interface. As propriedades podem ser físicas (por exemplo: condições de luminosidade), psicológicas (por exemplo: níveis de stress) e organizacionais (por exemplo: localização e papel definidos no organograma);
- **Tipo de utilizador (*stereoType*)** – Constitui os estereótipos de utilizador, ou seja, define categorias de utilizadores com características semelhantes (por exemplo: idioma).

Uma vez que na ferramenta FlexiXML a plataforma é fixa, só interessam as características relativas ao tipo de utilizador. No modelo para definição do tipo de utilizador a propriedade com maior interesse para a FlexiXML corresponde ao idioma (*language*), que define o idioma em que o utilizador pode visualizar a aplicação gerada. No ficheiro UsiXML é possível definir mais do que um contexto, o que permite que o utilizador possa visualizar a aplicação em diferentes idiomas.

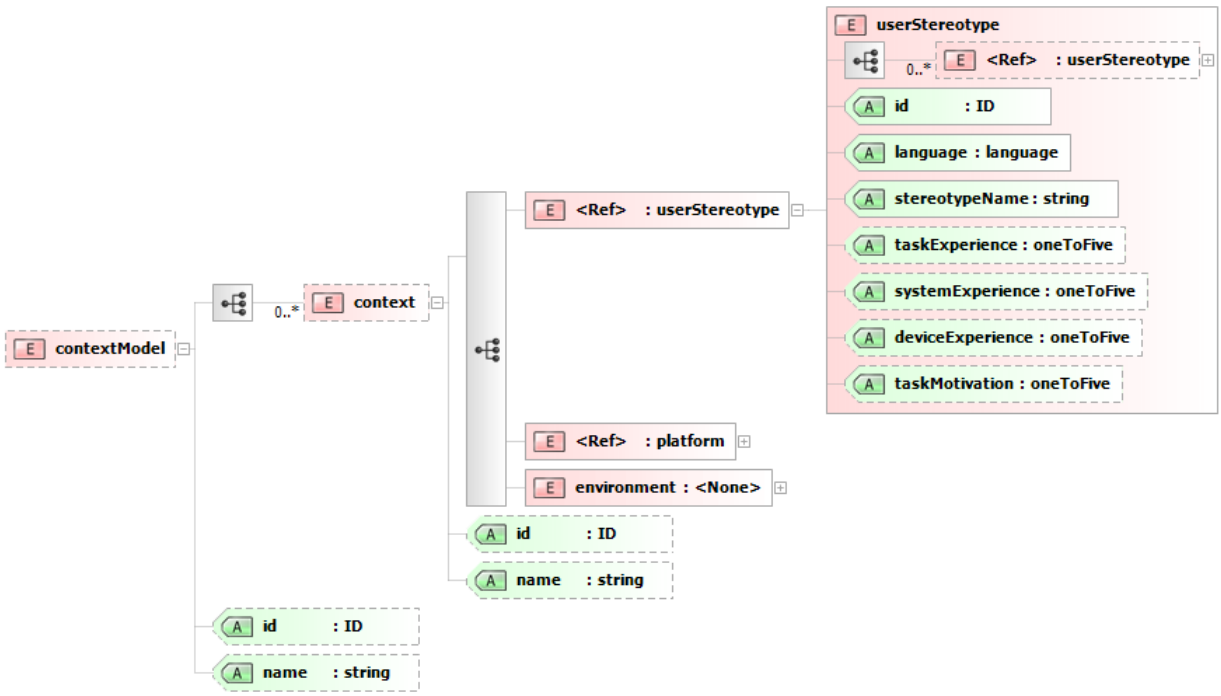


Figura 15 - *Schema* do modelo *contextModel*.

No Exemplo 6 estão criados dois contextos, "slideShowContext_En_US" e "slideShowContext_FR", para representarem o idioma inglês (language="en_US") e francês (language="FR"), respectivamente.

Exemplo 6 - Especificação do modelo *contextModel*.

```
<contextModel id="slideShowContextModel" name="slideShowContextModel">
      <context id="slideShowContext_En_US"name="slideShowContext_En_US">
            <userStereotype id="slideShowContextUser_US"language="en_US"
                  stereotypeName="slideShowContextUser_US"/>
      </context>

      <context id="slideShowContext_FR" name="slideShowContext_FR">
            <userStereotype id="slideShowContextUser_FR" language="FR"
                  stereotypeName="slideShowContextUser_FR"/>
      </context>
</contextModel>
```

Para especificação das traduções a apresentar nos componentes gráficos é necessária a utilização de um modelo adicional, *ResourceModel*.

ResourceModel

O **ResourceModel** (Figura 16), é o modelo que contém as definições dos objectos gráficos que dependem de um contexto (por exemplo, localização, idioma, cultura, etc). Este modelo contém todo o tipo de conteúdo que pode ser atribuído a um objecto de interacção (conteúdo, *tooltip*, etc).

O *ResourceModel* é composto por:

- **id** – Identificador;
- **name** – Nome;
- **cioRef** – Referência para o objecto de interacção para o qual se pretende definir as propriedades:
  - **cioId** – Identificação do objecto gráfico;
- **resource** – Define as propriedades para um contexto especifico:
  - **contextId** – Identificador do contexto a que o recurso se aplica;
  - Valores das propriedades: *tooltip*, conteúdo, *icon*, etc.

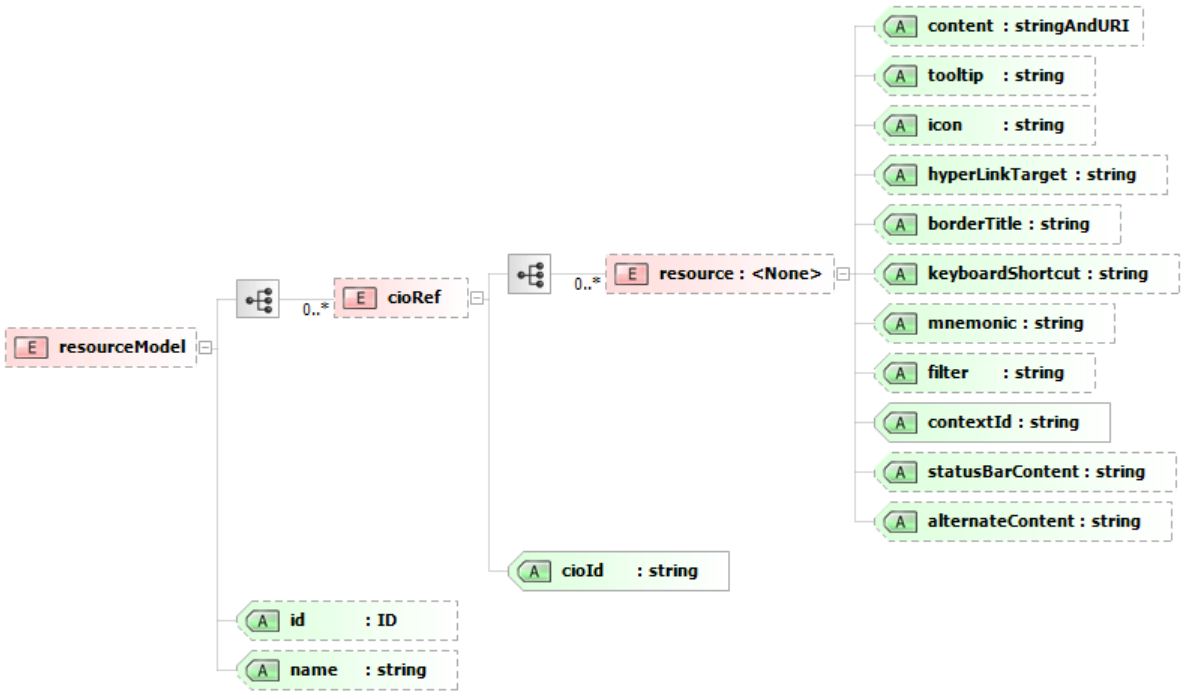


Figura 16 - *Schema* do modelo *resourceModel*.

Exemplo 7 - Especificação do modelo *resourceModel*.

```
<resourceModel id="slideShowResourceModel"
name="slideShowResourceModel">
   <cioRef cioId="slideshowWindow">
      <resource content="SlideShow" contextId="slideShowContext_En_US"/>
      <resource
content="SlideSpectacle"contextId="slideShowContext_FR"/>
   </cioRef>

   <cioRef cioId="B0">
      <resource contextId="slideShowContext_En_US" content="Next"
            tooltip="Press button to see next image"/>
      <resource contextId="slideShowContext_FR" content="Ensuite"
            tooltip="Appuyez sur le bouton pour voir ensuite l'image"/>
   </cioRef>
</resourceModel>
```

No Exemplo 7 é definido o título da janela da aplicação, texto do botão e *tooltip* do botão para o idioma Inglês e Francês (contextos criados no Exemplo 6).

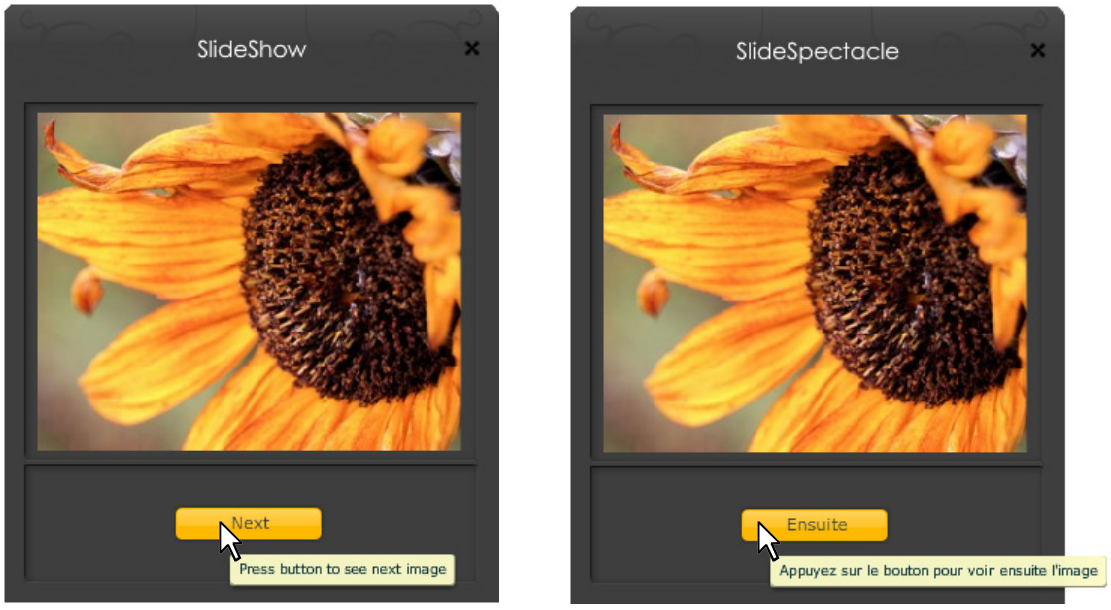


Figura 17 - Aplicação *SlideShow* em diferentes idiomas (Exemplo 6 e Exemplo 7).

No Figura 17 está apresentada a aplicação SlideShow nos idiomas inglês e francês, tal como foi definido nos Exemplo 6 e Exemplo 7. Como se pode observar o título da aplicação, o nome do botão, bem como a respectiva *tooltip* estão apresentadas nos dois idiomas.

Ao longo deste capítulo foi apresentada a linguagem UsiXML. No próximo capítulo serão apresentadas as principais ferramentas de suporte à linguagem.

Capítulo 3

UsiXML - Ferramentas

Como referido na secção 2.2.1 a linguagem UsiXML tem uma arquitectura subjacente que lhe permite que a especificação de uma interface possa ser efectuada a diferentes níveis de abstracção e dispõem de mecanismo de transformação entre eles. Para dar suporte a estas transformações, UsiXML dispõe de um conjunto de ferramentas. Na Figura 18 estão representadas as ferramentas de UsiXML que dão suporte às diferentes etapas de desenvolvimento de um projecto, desde a fase inicial de levantamento de requisitos até à obtenção da interface gráfica final. Neste capítulo é efectuada uma breve descrição das principais ferramentas desta linguagem.

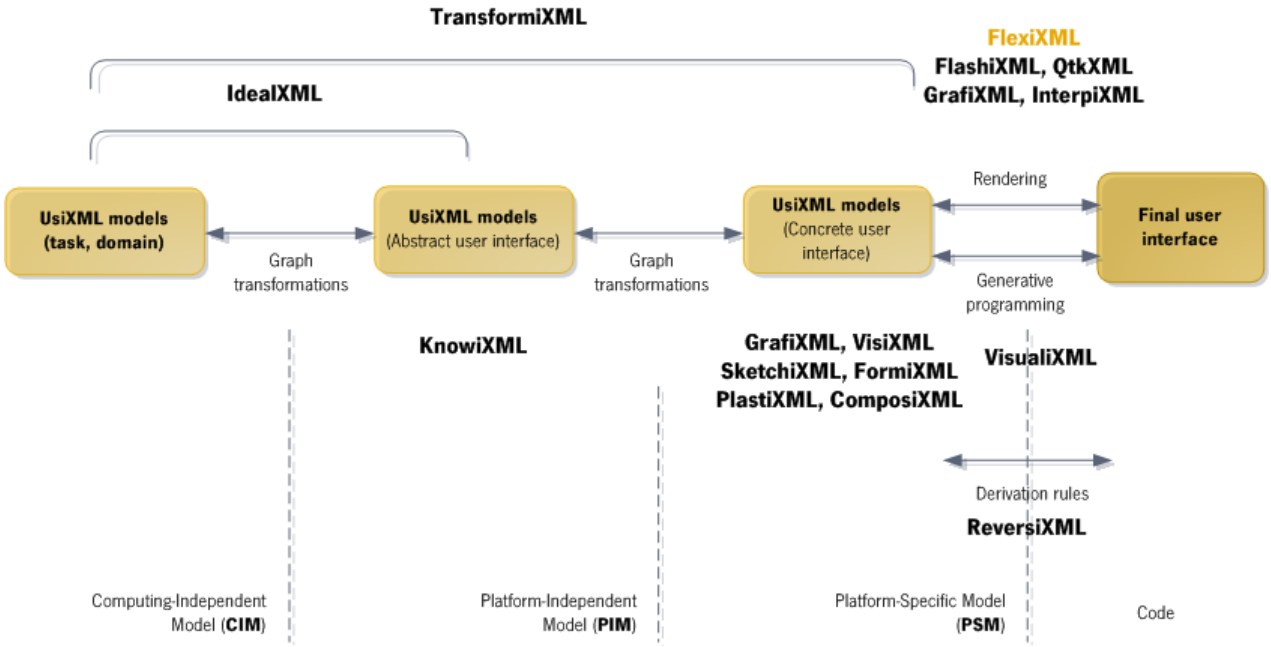


Figura 18 - Ferramentas da linguagem UsiXML [Vanderdonckt, 2005].

3.1 Editores

Nesta secção são apresentadas algumas ferramentas para criação de descrições UsiXML. A forma como as descrições são criadas varia com o tipo de ferramenta.

GrafiXML (Figura 19) é uma ferramenta que permite desenhar uma interface gráfica para múltiplos contextos de uso, isto é, vários utilizadores, plataformas e ambientes computacionais. O formato da interface gerada pode ser em Java ou XHTML, mas o principal formato é UsiXML.

Para suportar os múltiplos contextos esta ferramenta utiliza três tipos de representações: uma representação interna que consiste na especificação UsiXML, uma representação externa que corresponde à pré-visualização da interface e uma representação conceptual que corresponde ao modelo da interface.

GrafiXML é similar a qualquer ferramenta de desenho e geração de interface, mas permite manipular mais as propriedades dos *widgets*.

GrafiXML v1.2.0 é compatível com a v1.8 da linguagem *UsiXML* [Michotte, 2008] [W2].

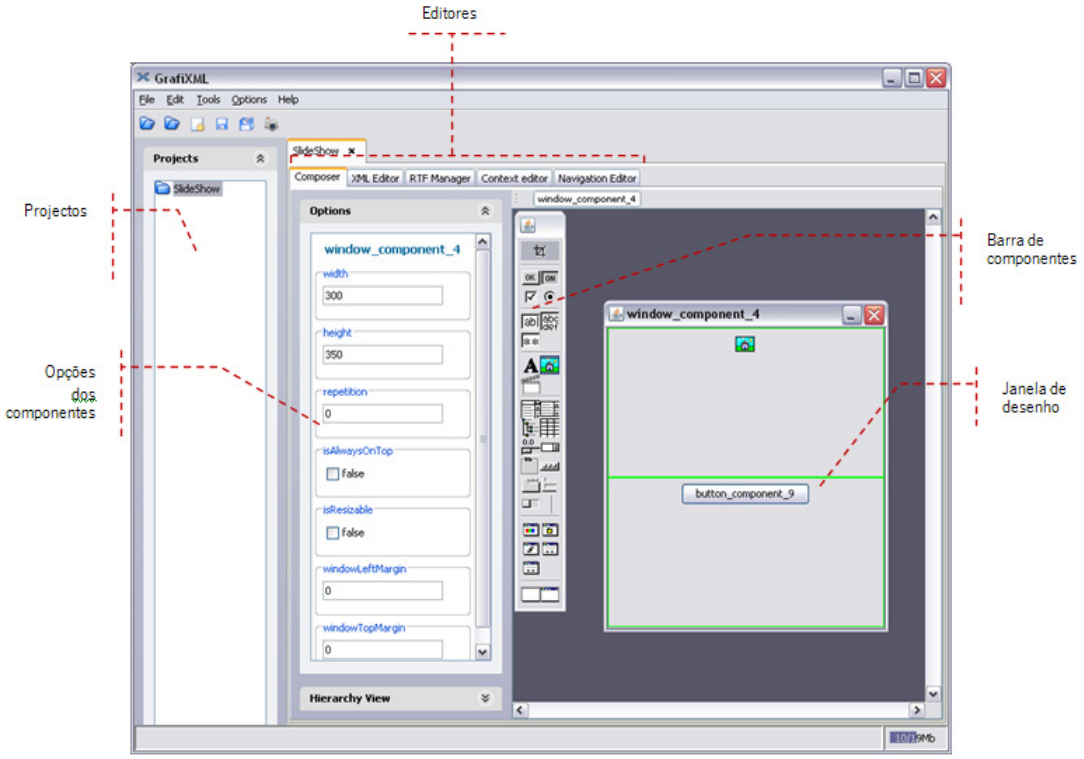


Figura 19 - GrafiXML.

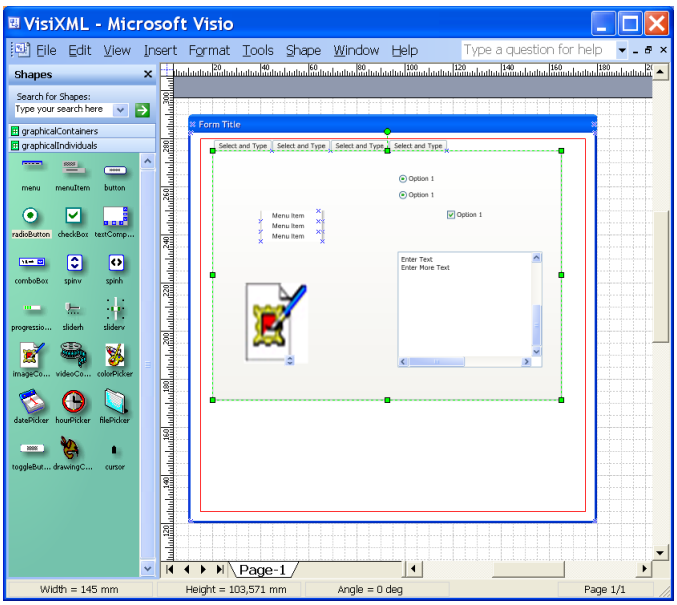


Figura 20 - VisiXML.

VisiXML [Coyette, 2007] é um editor gráfico para desenho de uma interface gráfica sobre o Microsoft Visio (Figura 20). Com este editor são definidos apenas quais os objectos gráficos que compõem a interface, ou seja, define o modelo *cuiModel* da interface.

O desenho efectuado nesta aplicação pode ser exportado para *UsiXML*, o que permite que a especificação possa depois ser editada por outra ferramenta, por exemplo o GrafiXML. [Vanderdonckt, 2005]

SketchiXML (Figura 21) é uma ferramenta interactiva que permite desenhar interfaces gráficas através de um esboço com vários níveis de detalhe e para diferentes contextos de uso.

A ferramenta permite que os *designers* ou os utilizadores finais possam esboçar a interface que pretendem. Depois de efectuados os desenhos estes passam por um processo de reconhecimento de formas e é gerada a especificação independentemente do contexto, tipo de utilizador ou plataforma computacional.

A especificação produzida pode depois ser enviada para outro editor, para definir propriedades que não são susceptíveis de manipulação nesta ferramenta. [Coyette, 2007] [W2]

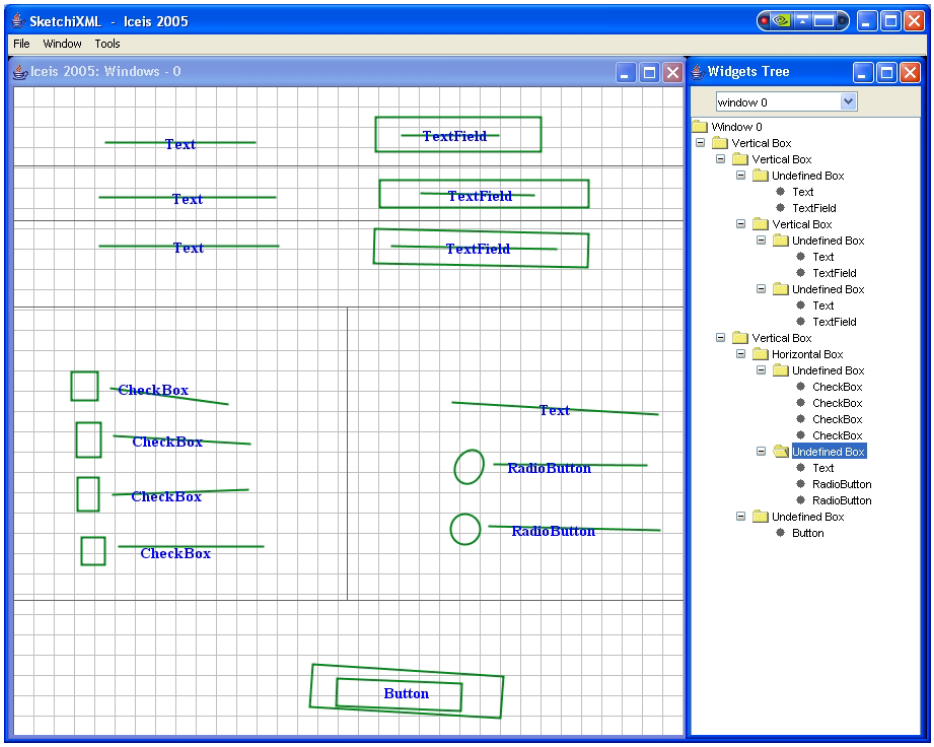


Figura 21 - SketchiXML.

IdealXML (Interface Development Environment for Applications specified in usiXML) [Montero, 2006] aplicação Java que contém um editor gráfico para modelo de tarefas, domínio e abstracto (Figura 22). É uma ferramenta orientada a modelo de padrões e é baseada em PLML (Pattern Language Markup Language) [W9]. Permite a criação e partilha de repositórios de modelos. [Montero, 2006] [W2]

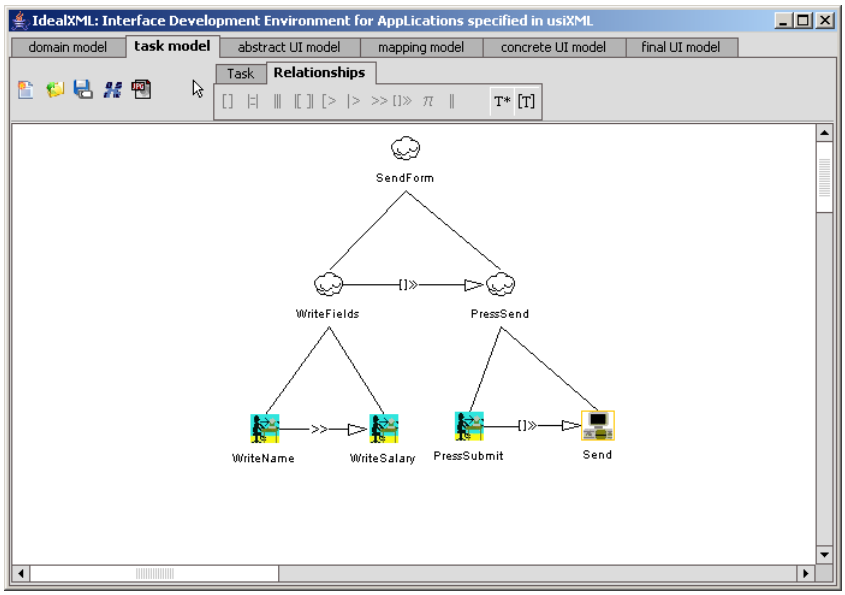


Figura 22 - IdealXML.

PlastiXML é um *plugin* desenvolvido para a ferramenta GrafiXML. Este plugin permite definir transições gráficas e operações de redimensionamento entre as janelas aplicacionais consoante o contexto. [Collignon, 2008] [W2]

ComposiXML (Composition of Applications specified in UsiXML) é um *plugin* desenvolvido para a ferramenta GraphiXML.

Este *plugin* permite utilizar qualquer componente isolado ou composto de uma interface e submete-lo a um conjunto de operações de forma a compor uma nova interface. A interface criada pode utilizar componentes existentes ou decompor um em partes mais pequenas para que possam ser reutilizadas apenas as partes pretendidas (Figura 23).

As operações utilizadas são selecção, projecção, união, intersecção, fusão, etc. (Figura 24). [Lepreux, 2007] [W2]

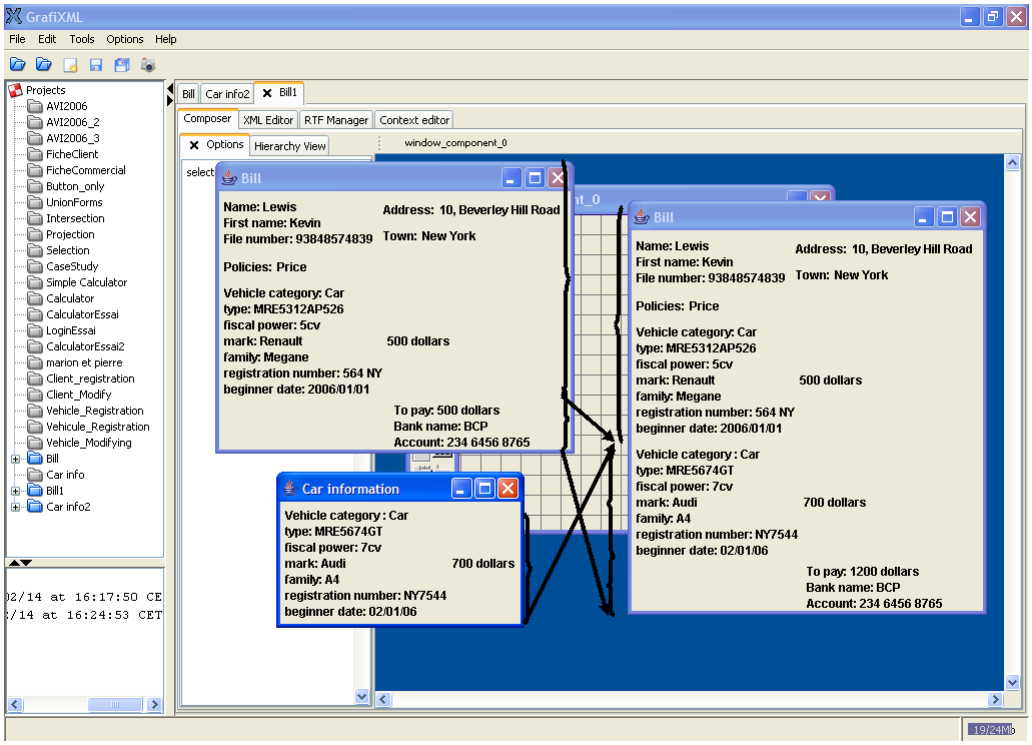


Figura 23 - ComposiXML - Exemplo de uma operação de união.

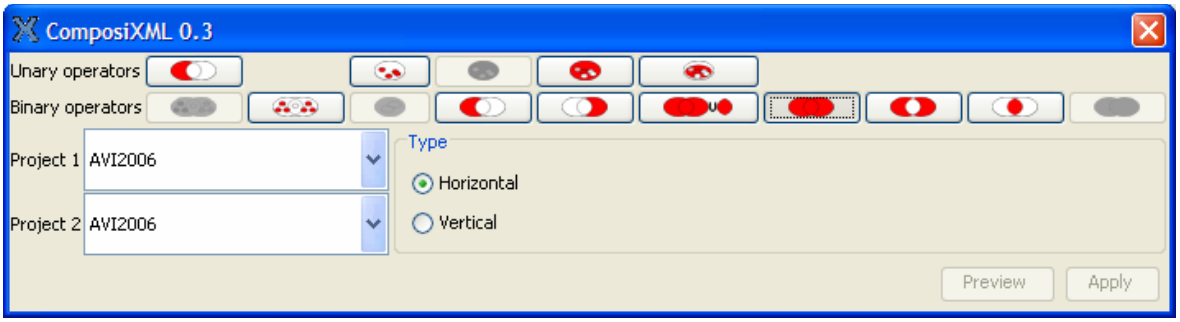


Figura 24 - ComposiXML - Operações.

3.2 Geradores

KnowUI/KnowiXML (*Knowledge-Based System)* representa o conhecimento dos designers da interface sobre a mesma. O sistema tem como objectivo guardar a justificação das

características interactivas da interface para que o utilizador possa entender a razão do desenho desenvolvido e assim melhorar a compreensão e a aceitação da interface desenhada. [Furtado, 2004] [W2]

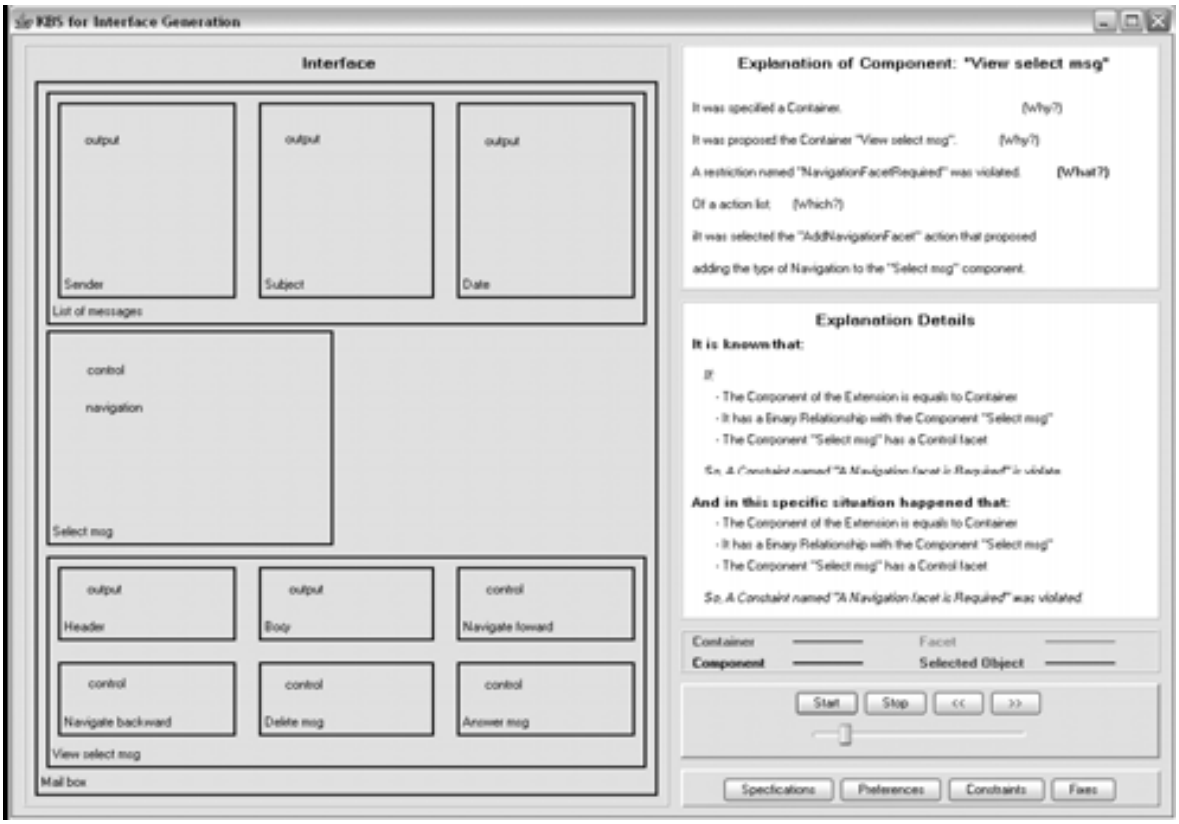


Figura 25 - KnowUI.

3.3 Interpretadores

FlashiXML [Youri, 2004] é um motor de geração de interfaces gráficas descritas em UsiXML (**Figura 26**). As interfaces são geradas em modo vectorial e SVG compatíveis. O facto de se tratar de interfaces vectoriais permite que as aplicações geradas possam ser redimensionadas e o seu conteúdo é adaptado às novas dimensões.

Esta ferramenta pode ser utilizada em qualquer plataforma computacional desde que equipada com SVG, *Flash plugin* ou *Flash player*.

FlashiXML é compatível com a v1.4.6 da linguagem *UsiXML.* [Youri, 2004] [W2]

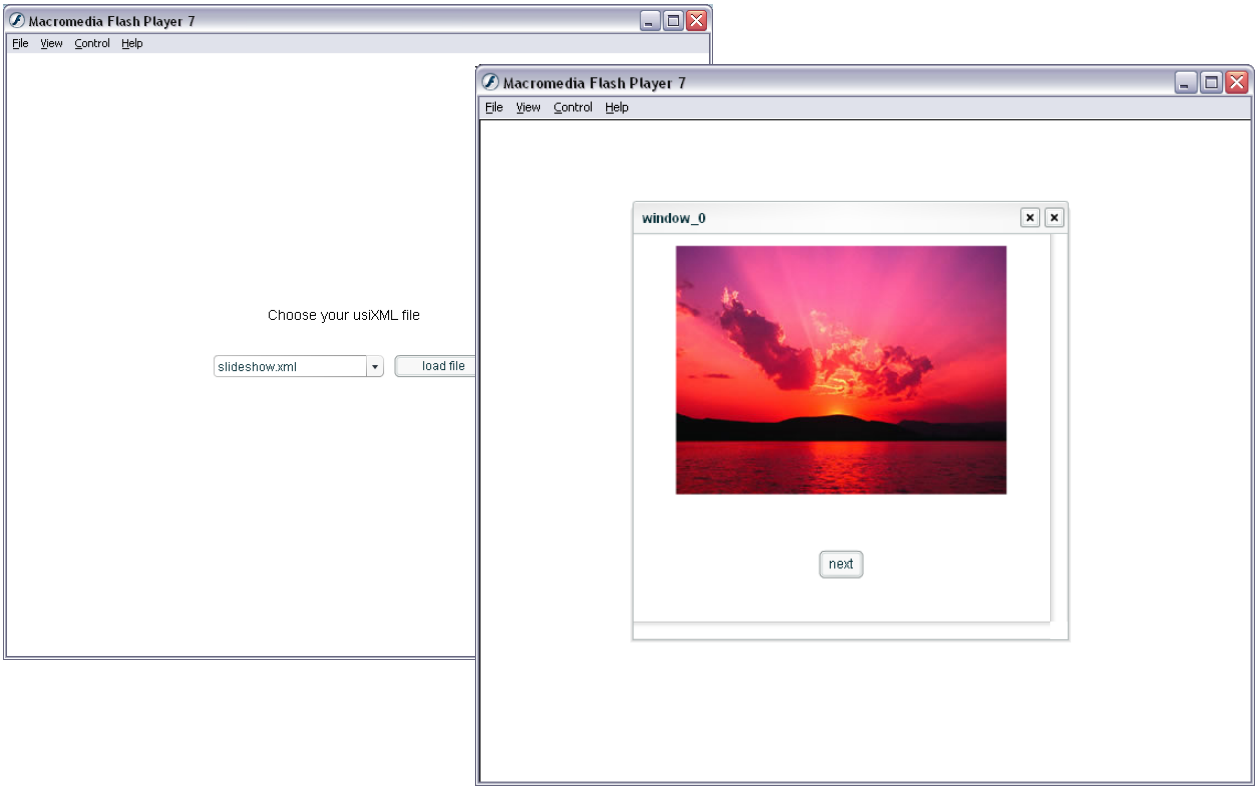


Figura 26 - FlashiXML.

QtkiXML (Figura 27) é um motor de geração de interfaces gráficas descritas em UsiXML para múltiplas plataformas sobre Tcl-Tk[5]. [Denis, 2005] [W2]

[5] TCL (Tool Command Language) é uma linguagem de programação aplicável as várias tipos de aplicações web e desktop, e também nas áreas de redes, administração e testes. É uma linguagem *open-source*, de fácil aprendizagem e extensível.

Tk é uma ferramenta gráfica para desenvolver aplicações desktop. Esta ferramenta não está desenvolvida apenas para TCL, mas também para outras linguagens dinâmicas. As aplicações produzidas podem correr em diferentes plataformas Windows, Mac OS X, Linux e outras. [W3]

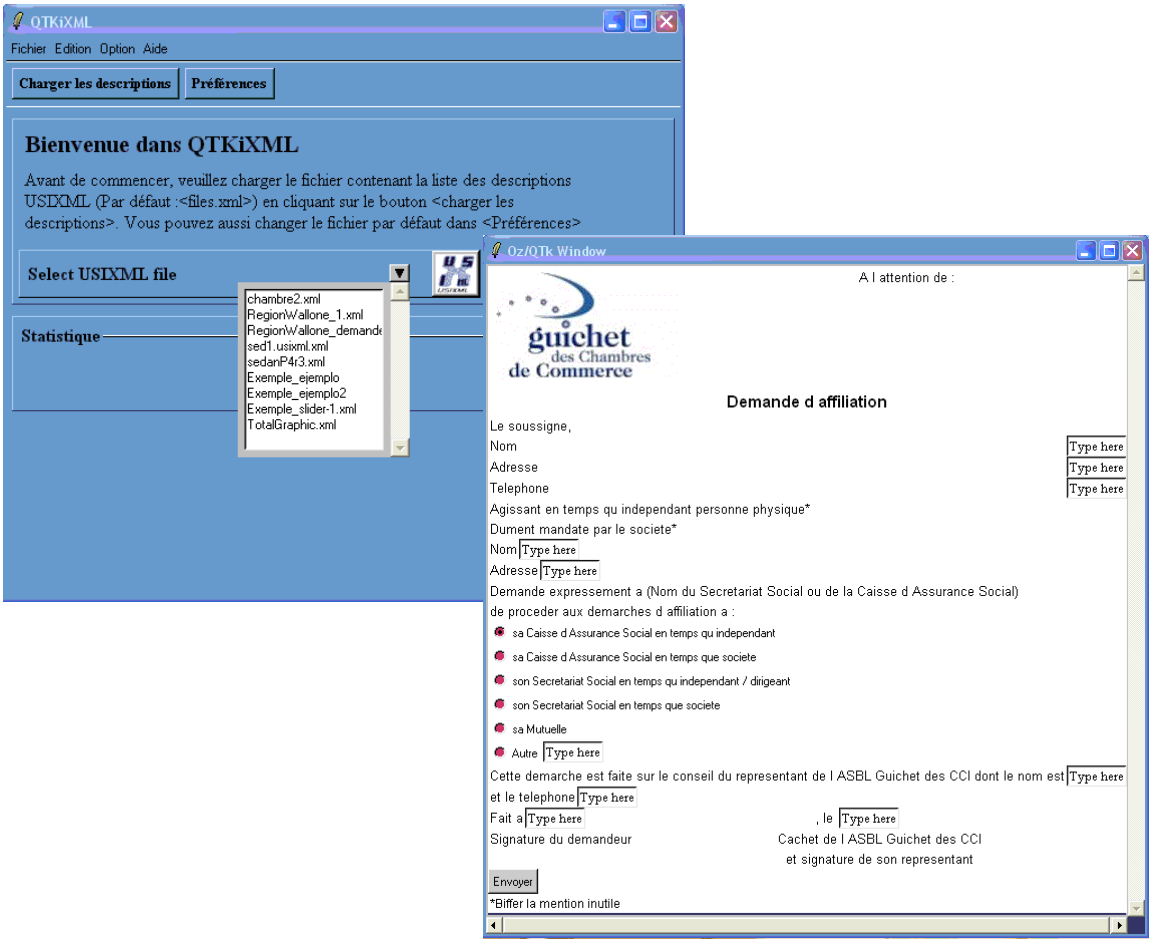


Figura 27 - QtkiXML.

InterpiXML [Goffette, 2007] é um interpretador de especificações em UsiXML. Nesta ferramenta o utilizador receber um conjunto de ficheiros UsiXML que contêm as especificações de diferentes tarefas, o objectivo é que o utilizador as possa "abrir" e executar em simultâneo para simular uma workstation única (Figura 28).

InterpiXML é compatível com a v1.8 da linguagem *UsiXML.* [Goffette, 2007] [W2]

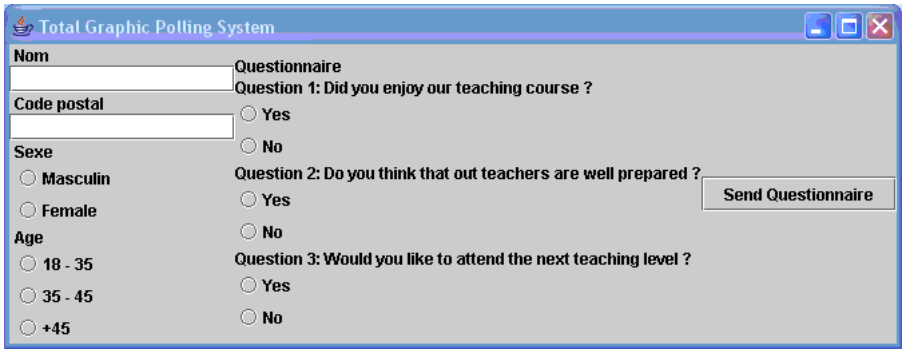
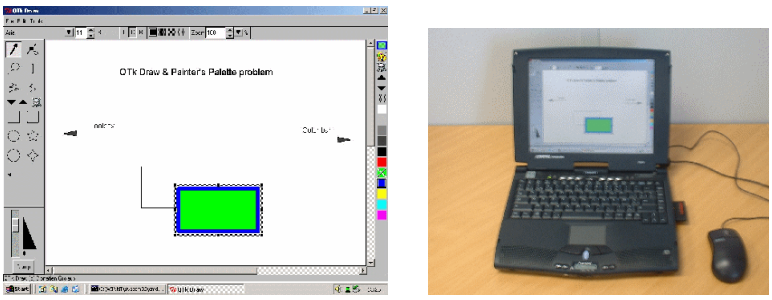


Figura 28 - InterpiXML - Exemplo de uma interface gráfica.

DistriXML [Grolaux, 2007] consiste num conjunto de modelos que dão suporte ao conceito de anexar e desanexar interfaces existentes em DUIs (Distributed User Interfaces). Com esta ferramenta uma interface gráfica pode ser descomposta em partes e cada uma das partes pode ser migrada para uma plataforma computacional diferente (Figura 29).

A próxima versão desta ferramenta designa-se por EBL (Extended Binding Layer), em que o nível de decomposição da interface é ao nível do componente gráfico, por exemplo um radio *button* poder ser decomposto num círculo e em uma *label*. [Grolaux, 2007] [W2]

Versão inicial da interface gráfica:



Após distribuição:

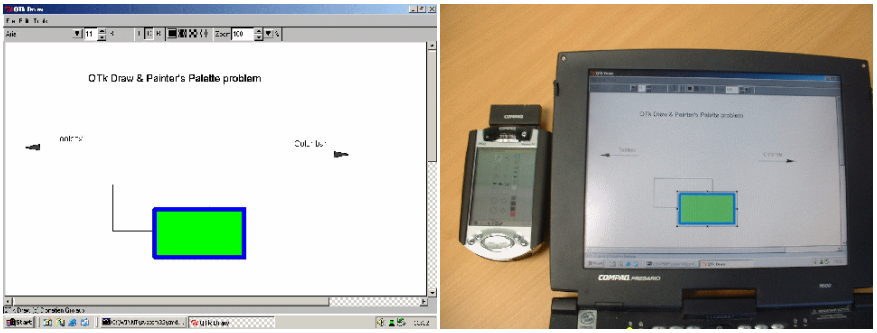


Figura 29 - DistriXML.

HaptiXML [W6] é um interpretador que gera interfaces gráficas em 3D com interacção pelo toque (haptic interaction) e baseadas em especificações UsiXML.

HapticBrowser[6] é um projecto que está a explorar diferentes abordagens das páginas Web utilizando a interacção via o toque, de forma invisuais poderem navegar na Web através do toque.

O objectivo é existir uma correspondência entre os componentes HTML e objectos 3D (**Figura 30**). [W6] [W2]

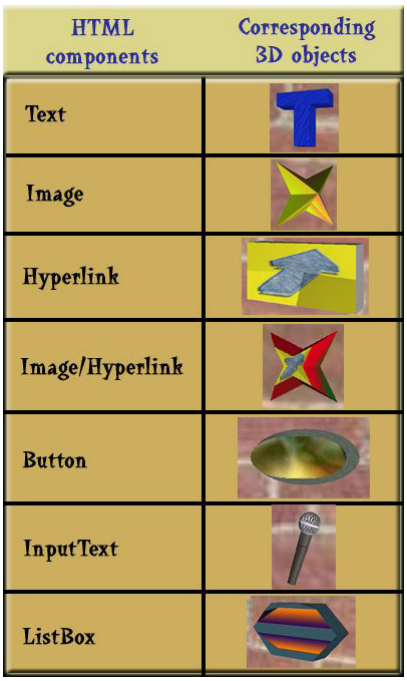


Figura 30 - HapticWebBrowser - Componentes suportados.

3.4 Análise comparativa

Nesta secção foram apresentadas várias ferramentas que dão suporte à linguagem UsiXML. Embora existam várias ferramentas, todas elas são diferentes pois cada uma aborda aspectos específicos da linguagem ou da interface gráfica a descrever.

[6] http://kaklanis.googlepages.com/nickkaklanis-3dhapticwebbrowser

Na gama dos editores podemos destacar, SketichXML [Coyette07] que utiliza como *input* interfaces desenhadas à mão (*sketchs*), GrafiXML [Michotte08] em que interface é criada por manipulação directa de componentes no ecrã, VisiXML [Coyette07] em que o desenho da interface é efectuado sobre o Microsoft Visio, entre outras ferramentas abordadas na secção 3.1.

Relativamente aos interpretadores, cada interpretador gera interfaces com um conjunto de características específicas. Das ferramentas existentes podemos salientar, FlashiXML [Youri04] que gera interfaces vectoriais, HaptiXML [Haptic09] que gera interfaces gráficas em 3D com interacção pelo toque, QtkiXML [Denis05] que cria interfaces para múltiplas plataformas, InterpiXML [Goffette07] que permite a interpretação simultânea de várias descrições UsiXML.

Como referido em epígrafe a UIDL UsiXML dispõe de várias ferramentas para edição e interpretação da linguagem. Como tal, a questão "Porquê mais uma ferramenta?" pode ser colocada. A justificação prende-se com a inexistência de uma ferramenta actualizada que permita a animação de modelos, por forma a facilitar um processo de prototipagem e análise rápida[7]. A ferramenta que mais se aproxima do pretendido é FlashiXML [Youri, 2004]. No entanto esta ferramenta não está adaptada à versão actual de UsiXML.

FlexiXML foi a ferramenta criada, a qual está adaptada à versão actual de UsiXML, implementada numa tecnologia recente e inclui funcionalidades adicionais que serão apresentadas nas próximas secções. Dado FlashiXML ser a ferramenta que mais se aproxima do pretendido, esta ferramenta foi analisada em detalhe. Contudo, a implementação de FlexiXML foi efectuada de raiz e de forma independente de FlashiXML.

[7] Jean Vaderdonckt, Unidade Católica de Louvain, 2008. Comunicação privada.

"[…] estamos a viver um momento vital e estratégico para quem desenvolve interfaces. Pode-se dizer que a tecnologia está pronta. Temos as pontes e os túneis construídos, agora as estradas precisam ser pavimentadas e as sinalizações pintadas para tornar possível o pesado tráfico da grande leva de utilizadores[…]"

[Schneiderman, 1998]

Capítulo 4

FlexiXML

FlexiXML é um motor de geração de interfaces gráficas descritas segundo uma linguagem de modelação definida. A versão actual da ferramenta suporta a linguagem UsiXML, mas está concebida para permitir a inclusão de outras linguagens declarativas e baseadas em XML. As interfaces geradas são criadas na linguagem Flex3 + AS3, contudo a ferramenta está estruturada para que o utilizador possa especificar em que linguagem pretende que a interface seja gerada. Estas duas características surgem para facilitar o uso desta aplicação, não limitando os utilizadores a uma linguagem de especificação e de geração específicas, alargando assim o número de aplicações que dela podem beneficiar.

O facto de a ferramenta FlexiXML estar desenvolvida em AIR+Flex torna-a independente da plataforma computacional em que é executada. O FlexiXML está disponível em dois formatos: desktop e Web.

Ao longo deste capítulo serão apresentadas as características da ferramenta FlexiXML, desde a tecnologia utilizada, passando pela descrição da arquitectura e principais funcionalidades, finalizando com alguns exemplos de utilização.

O conteúdo deste capítulo baseia-se no artigo [Mendes, 2009] publicado no 17º EPCG (Encontro Português de Computação Gráfica).

4.1 Tecnologia

A ferramenta FlexiXML está desenvolvida em AIR + Flex + ActionScript3 [W13].

Flex é uma *framework open-source* para criação e manutenção de aplicações interactivas, aplicações Web as quais são compatíveis com os principais *browsers*, *desktops* e sistemas operativos.

Para descrever o *layout* da interface gráfica e alguns comportamentos associados, Flex utiliza MXML, que é uma linguagem declarativa baseada em XML. Relativamente à parte lógica da aplicação esta é definida utilizando ActionScript 3 (linguagem orientada a objectos). Flex disponibiliza também uma biblioteca de componentes, os quais são extensíveis, permitindo a criação de componentes adicionais.

As aplicações criadas em Flex podem ser executadas num browser utilizando Adobe Flash® Player ou em desktop utilizando Adobe AIR.

Adobe AIR permite a criação de aplicações desktop passíveis de serem executadas em 3 sistemas operativos (Mac, Windows, Linux) com o mesmo ficheiro. AIR fornece uma API para escrita de ficheiros, *drag-and-drop*, notificações do sistema, detecção de rede, entre outros.

As razões para a escolha desta tecnologia são maioritariamente por:

- Possibilidade de ter uma aplicação Web e *deskotp* compatível com os principais *browsers* e sistemas operativos;
- Possibilidade de aceder a qualquer tipo de dados, base de dados, ficheiros XML, entre outros;
- Integração de bibliotecas para interpretação de XML (E4X);
- Facilidade na manipulação de estilos de uma aplicação;
- Existência de uma biblioteca extensível de componentes;
- Melhor performance relativamente às versões anteriores de Flash;
- Facilidade no desenho de uma aplicação.

4.2 Platafoma

A ferramenta FlexiXML está estruturada em torno do conceito de *plugins*, em que cada *plugin* implementa um conjunto de funções específicas. Este tipo de abordagem permite que a ferramenta possa estar em constante evolução com a integração de novos *plugins* com outras funcionalidades e sem impacto nos já existentes.

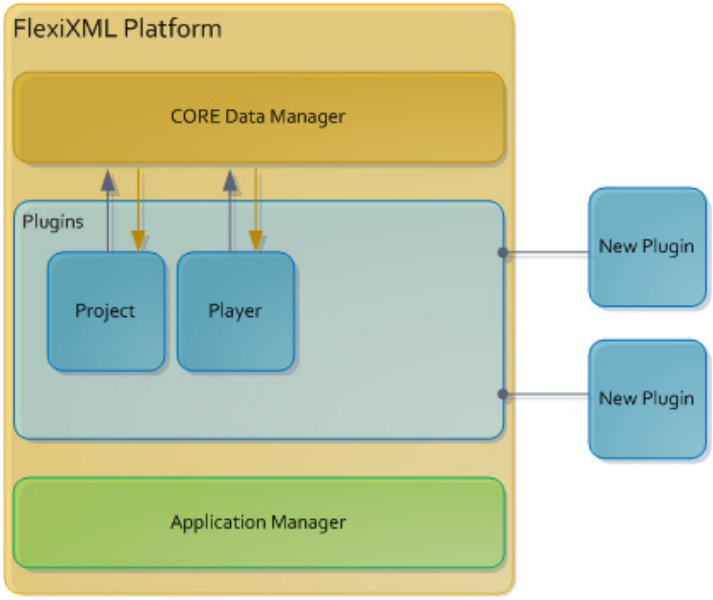


Figura 31 – Plataforma do FlexiXML.

Na Figura 31 está representada a estrutura da aplicação, a qual pode ser divida em 3 camadas:

- **Application Manager** – Camada que efectua a gestão da aplicação, sendo responsável por: controlar as mensagens a visualizar na aplicação; efectuar o pedido da estrutura de *plugins;* efectuar o pedido das UIDLs disponíveis e respectivas configurações, entre outros.
- **Plugins** – Camada onde são organizados os *plugins* a disponibilizar na aplicação. Para além dos *plugins* existentes por omissão é possível a integração de *plugins* adicionais;
- **CORE Data Manager** – Camada responsável por guardar e disponibilizar a informação que é partilhada por todos os *plugins*.

4.3 Plugins

A lista de *plugins* que a aplicação FlexiXML disponibiliza é definida num ficheiro de configuração XML cujo *schema* é o definido na Figura 32. Esta listagem é carregada no processo inicial de carregamento da aplicação.

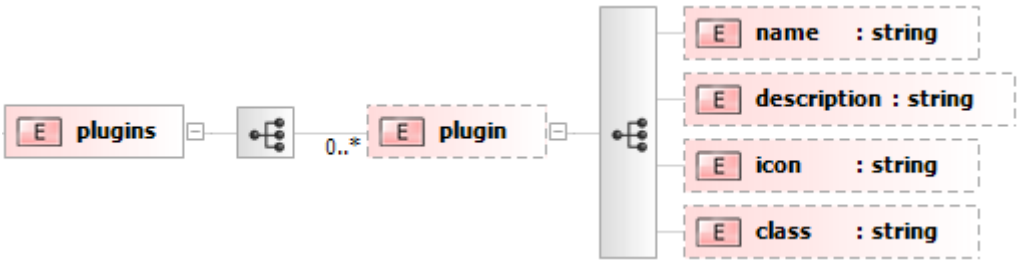


Figura 32 - *Schema* do ficheiro de configuração de *plugins*.

Cada *plugin* do ficheiro de configuração tem de definir as seguintes propriedades:

- **name** – Nome do *plugin*;
- **description** – Breve descrição do *plugin*;
- **icon** – Localização do *icon* que o representa, caso se aplique;
- **class** – Nome completo da classe responsável por controlar o *plugin*.

Suponha-se que é necessário adicionar um novo *plugin* com o nome "Editor" e cuja função é editar os ficheiros de um projecto. Para que a aplicação o disponibilize é necessário adicioná-lo neste ficheiro, fornecendo valores para as propriedades acima definidas (ver Figura 33).

```
<plugins>
	…
	<plugin>
		<name>Editor</name>
		<description>Plugin to edit the project files</description>
		<icon> </icon>
		<class>Plugins.Editor.Editor</class>
	</plugin>

	…
</plugins>
```

Figura 33 - Exemplo da definição de um *plugin* no ficheiro de configuração de *plugins*.

Cada *plugin* na aplicação é disponibilizado como um separador adicional numa barra de navegação, como representado na Figura 34.

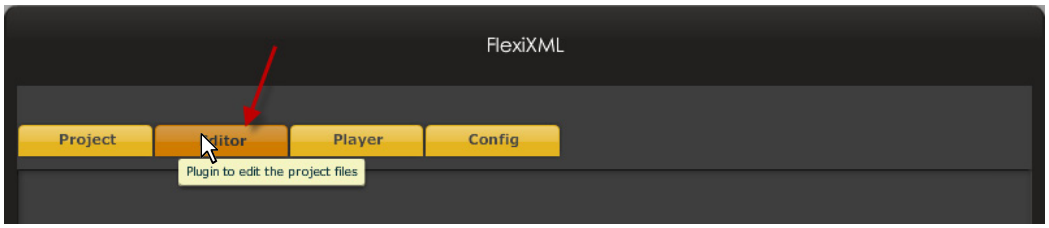


Figura 34 - Barra de *plugins* na aplicação FlexiXML.

A integração de novos *plugins* é suportada pela classe **PluginBase** (ver diagrama de classes apresentado na Figura 35).

Esta classe recebe um objecto, **PluginType**, que contém todas as propriedades necessárias para o caracterizar um plugin: nome, descrição, icon, etc… Para além das propriedades que o caracterizam a classe tem de implementar um *interface,* **IPlugin**, que define todos os métodos essências para que a aplicação FlexiXML o possa integrar.

A classe PluginBase disponibiliza a definição destes métodos por omissão. Como tal, novos *plugins* podem ser definidos por especialização desta classe base. Os métodos utilizados nesta integração são:

- **setInitialStatus()** – define o estado inicial do *plugin*: activo ou inactivo;
- **setEnabled(enabled:Boolean)** – atribui um estado específico ao *plugin*: activo ou inactivo;
- **projectLoaded(event_evt:Event)** – método executado sempre que é carregado um novo projecto na aplicação;
- **projectIsNotLoaded (event_evt:Event)** – método executado sempre que deixa de existir um projecto na aplicação.

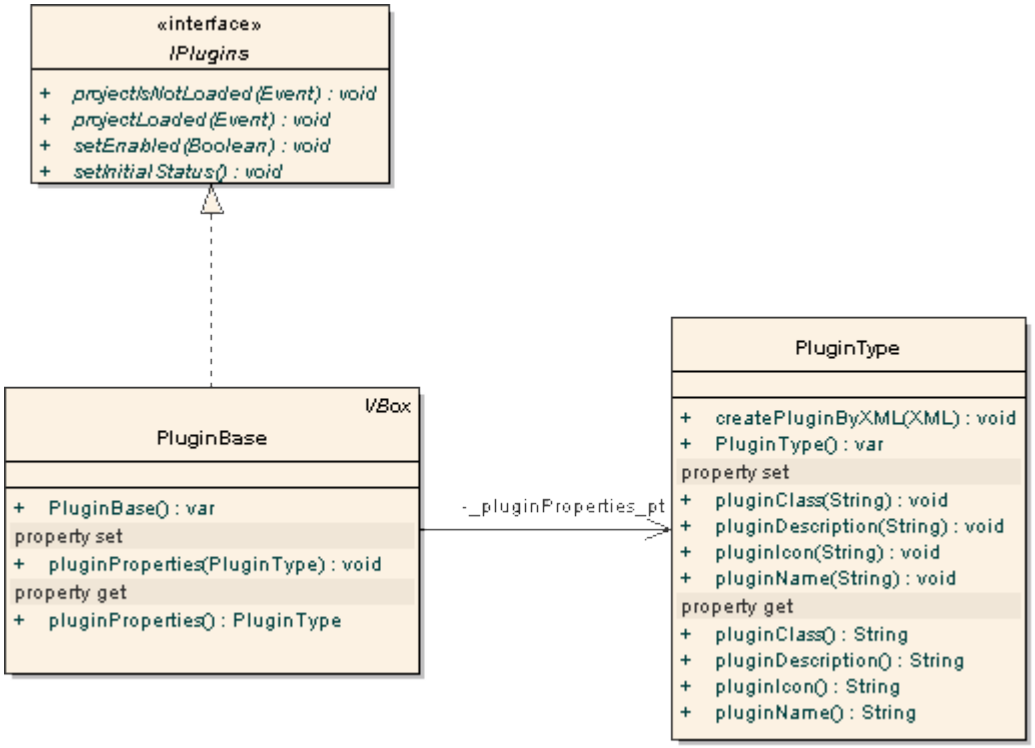


Figura 35 - Diagrama de classes de um *plugin* FlexiXML.

Para além de FlexiXML permitir a integração de novos *plugins*, a versão actual disponibiliza de raiz dois *plugins* principais: Project e Player. De seguida são apresentadas as principais funcionalidades de ambos.

4.3.1 Plugin Project

O *plugin* **Project** (Figura 36) é o responsável pelo carregamento dos projectos (Figura 36, b). É neste *plugin* que o utilizador pode especificar qual a UIDL que pretende utilizar (Figura 36, a).

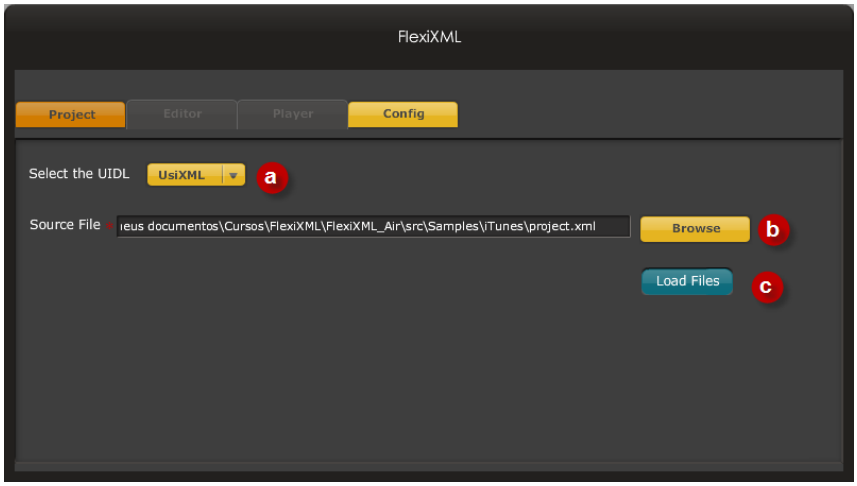


Figura 36 - Project Plugin.

Load do projecto

O *plugin* Project recebe como *input* um ficheiro com a informação necessária para carregar o projecto. Este ficheiro deverá ter com o formato definido na Figura 37. As propriedades a definir são:

- **sourceFile** – Localização do ficheiro com a especificação gráfica da interface a gerar (p.e. UsiXML);
- **externalScript** – Localização do ficheiro que contém o controlo de diálogo da interface gráfica. A linguagem UsiXML tem na definição da interface os nomes dos métodos a invocar para interagir com a mesma, contudo, a implementação destes métodos tem de ser efectuada num ficheiro externo. Na versão actual de FlexiXML esta implementação está desenvolvida em ActionScript 3, o que gera um ficheiro swf. A propriedade *externalScript* foi adicionada para permitir especificar a localização do ficheiro swf, através do qual vai ser possível efectuar a animação/interacção com a interface gráfica gerada.

  Este ficheiro só existe caso o processamento a efectuar na interacção não possa ser definido em UsiXML. Por exemplo, para situações em que a interacção consiste apenas em transições (*fadeOut*/*fadeIn*, abrir/fechar, mostrar/esconder, etc.) esta especificação pode ser definida em UsiXML, não sendo necessário criar um ficheiro com essa lógica. No caso de situações que envolvam por exemplo, cálculos,

passagem de valores entre componentes, entre outras situações, este processamento não é possível de especificar em UsiXML, como tal é necessário criar um ficheiro com um método que efectue essa processamento, e na definição em UsiXML coloca-se o nome do método com os respectivos parâmetros.

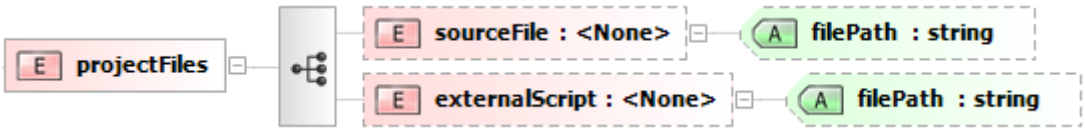


Figura 37 - *Schema* do ficheiro que define um projecto.

Uma vez recebidos este ficheiros o Project tem de interpretar a especificação da *interface* e carregar o ficheiro com a lógica da mesma. A interpretação do ficheiro com a descrição da *interface* é efectuada por um *parser* adequado à UIDL seleccionada pelo utilizador (Figura 36, a). O mapeamento entre o *parser* e a UIDL é definido num ficheiro de configuração, que é apresentado na secção seguinte. Actualmente apenas está disponível o *parser* para UsiXML, mas outros podem ser integrados. Para tal o *parser* tem que implementar o método parse().

No momento de *parsing* são criadas entidades que representam os componentes presentes na especificação e que podem posteriormente ser interpretadas por outros *plugins*, nomeadamente o *player* no momento de geração da interface. Das várias entidades criadas neste processo podem-se destacar as seguintes:

- **FUIItem** (Final User Interface item) (Figura 38) – Objecto que contém todas as propriedades que permitem definir um *widget*/controlo existente na especificação da interface gráfica (ficheiro UsiXML, p.e.). Este objecto contém propriedades como, posição x/y e dimensões do widget, conteúdo a apresentar, identificador, etc;

- **Context** e **ContextManager** (Figura 39) – Context, contêm as propriedades que permitem definir um contexto: identificador, nome, tipo (estereótipo, plataforma ou ambiente) e a referência para o objecto com as propriedades específicas do tipo de contexto (no caso da ferramenta FlexiXML apenas o tipo estereótipo é

interpretado). O ContextManager é a entidade que gere todos os contextos existente para a interface gráfica actual.

- **Resource e ResourceManager** (Figura 40) – Resource, define as propriedades que podem ser anexas a um objecto da interface gráfica. Pode ser uma imagem, um texto, uma *tooltip*, …. Esta entidade está relacionada com o contexto, pois qualquer uma destas propriedades pode ser alterada pelo contexto actual da aplicação. O ResourceManager é a entidade que gere todos os recursos existente para a interface gráfica actual.

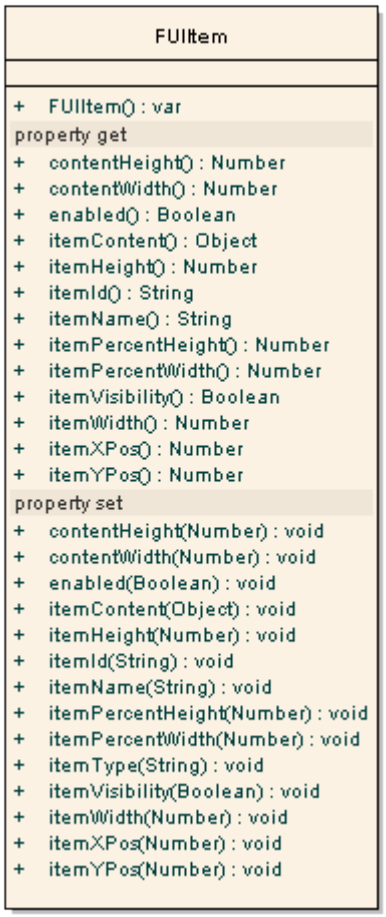


Figura 38 - API do objecto FUIItem.

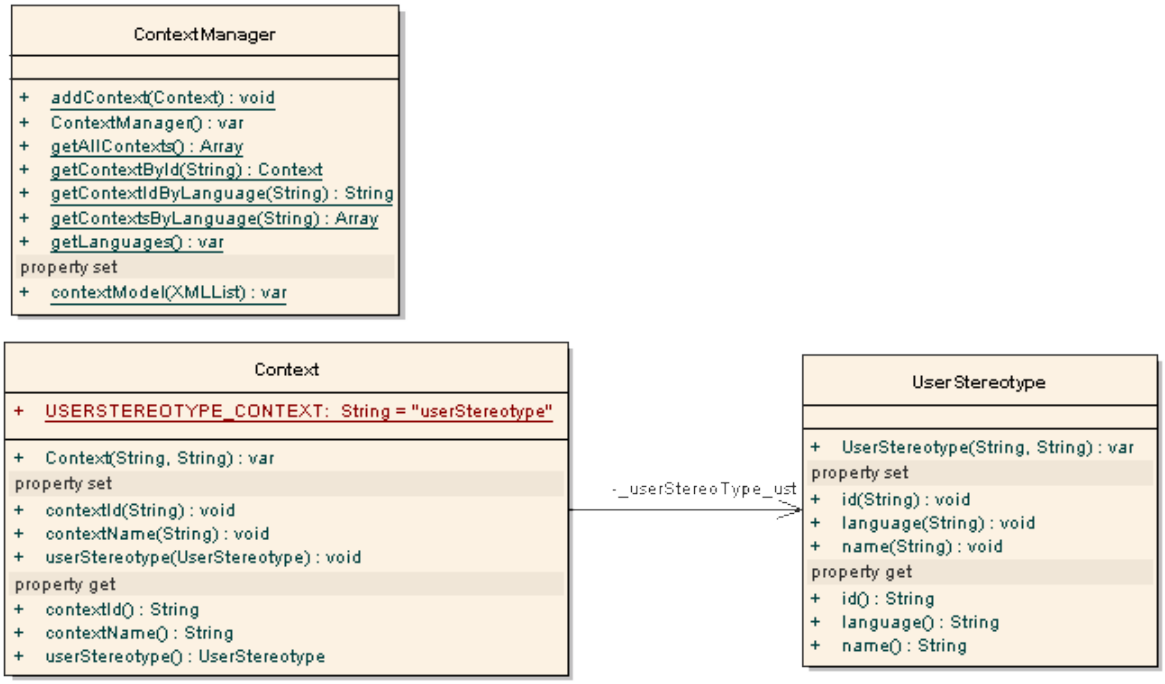


Figura 39 - Diagrama de classes das entidades que controlam os contextos da interface gráfica.

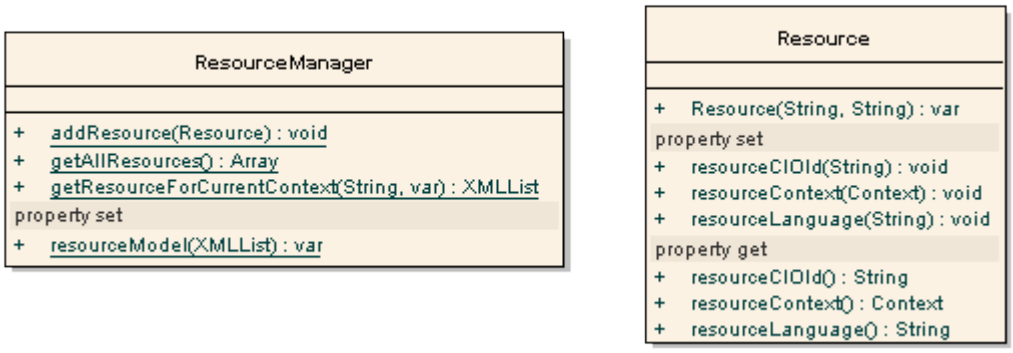


Figura 40 - Diagrama de classes do Resource e ResourceManager.

Depois de interpretados os ficheiros do projecto, a informação neles contida é enviada ao CORE Data Manager.

O CORE Data Manager (Figura 41) centraliza a informação que pode ser partilhada por todos os *plugins*. Desta forma, sempre que um *plugin* necessitar de informação sobre o projecto

actual efectua o pedido a este gestor. É através deste gestor que os *plugins* podem obter informações como:

- O identificador do contexto actual (relativo ao idioma, no caso da versão actual de FlexiXML) (contextId);
- A referência do objecto que gere a parte interactiva da interface gráfica que é criada (externalScript);
- As propriedades de todos os objectos da interface gráfica que é gerada (projectFUI);
- A referência para todos os objectos gráficos da interface gráfica que é gerada (projectGUI);
- A referência para o parser actual (projectParser).

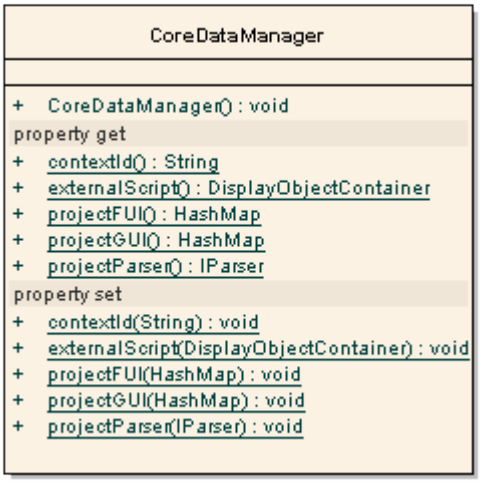


Figura 41 - API do CoreDataManager.

Especificação da lista de UIDLs

A lista de UIDLs que FlexiXML interpreta está definida num ficheiro de configuração XML (Figura 42), onde cada UIDL tem de especificar:

- **id** – Identificador da linguagem;
- **parser** – Nome completo da classe que contém o *parser* da UIDL;

- **extension** – Extensão do ficheiro de especificação;
- **default** – Indica se é a UIDL a utilizar por omissão;
- **UIGenerator** – Lista de linguagens que podem ser utilizadas para gerar a interface quando defina por esta UIDL. É neste campo que é definido o mapeamento entre os componentes da UIDL e os *widgets* da linguagem em que a interface pode ser gerada. Este campo é descrito em mais detalhe na secção 4.3.2.

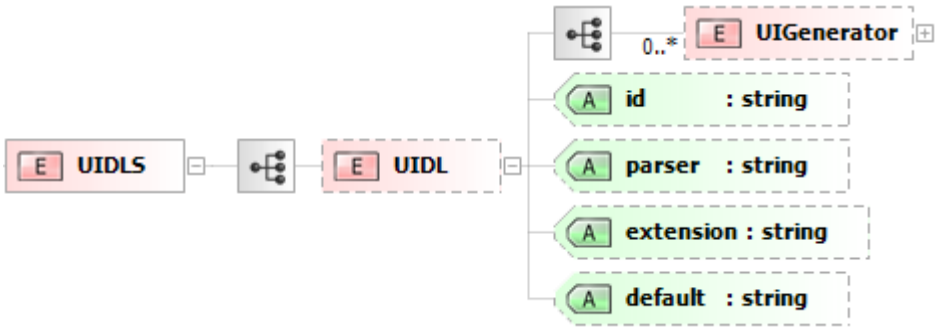


Figura 42 - *Schema* do ficheiro que define as UIDLs.

A definição da linguagem UsiXML está definida neste ficheiro, como apresentado na Figura 43.

```
<?xml version="1.0" encoding="UTF-8"?>
<UIDLS>
      <UIDL id="UsiXML"
            parser="Classes.Parsers.UsiXMLParser"
            extension="*.xml"
            default="true">
            <UIGenerator id="AS3"
      uiGeneratorClass="Classes.UIGenerators.AS3UIGenerator"
      default="true">
            …
            </UIGenerator>
      </UIDL>
<UIDLS>
```

Figura 43 - Exemplo da definição de uma UIDL no ficheiro de configuração de UIDLs.

Na Figura 43, pode-se verificar que, UsiXML surge na aplicação FlexiXML com o identificador "UsiXML", o *parser* criado para a sua interpretação foi denominado "Classes.Parsers.UsiXMLParser", os ficheiros de especificação nesta linguagem têm um formato em XML e como tal têm uma extensão "*.xml". Nesta ferramenta, UsiXML está definida como sendo a linguagem de especificação por omissão. Relativamente às linguagens que permitem gerar interfaces definidas neste formato, esta disponível por omissão ActionScript 3.

4.3.2 Plugin Player

Player Plugin (Figura 44), é responsável por gerar a interface gráfica do projecto actual com a informação relevante do utilizador (d). Para além de gerar a interface permite efectuar alterações em tempo de execução à mesma, tais como, mudança de idioma (b) e de estilos (c).

É neste *plugin* que o utilizador pode especificar qual a linguagem em que pretende que a interface seja gerada (a).

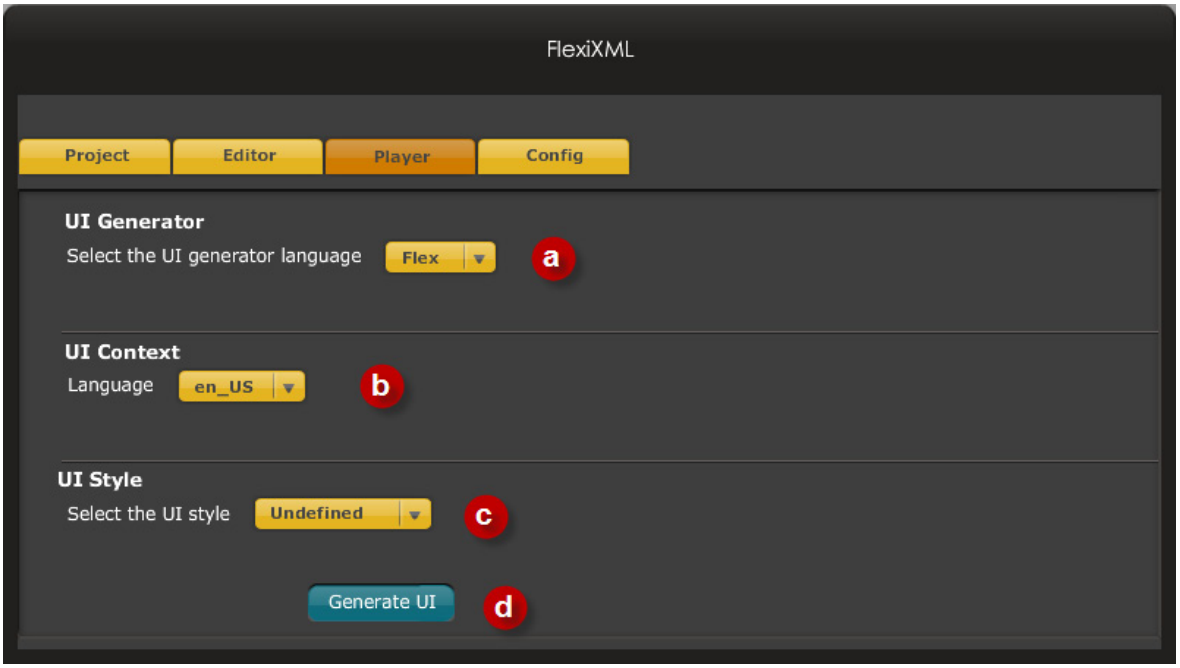


Figura 44 - Player Plugin.

Geradores

No momento de geração da interface o *plugin* Player acede ao CORE Data Manager para obter a informação acerca da descrição actual.

Esta informação é então interpretada para serem criados os componentes e/ou objectos que as representam. Para cada componente definido no modelo é criado o componente gráfico correspondente, o comportamento e conteúdo associado e as possíveis transições entre eles. O mapeamento entre cada elemento UsiXML e os *widgets*/controlos na interface a gerar está definido num ficheiro de configuração (Figura 43).

Na secção sobre o Project Plugin, referiu-se a existência de um ficheiro de configuração de UIDLs em que para cada uma era possível definir a lista de linguagens em que uma interface pode ser gerada. Para cada gerador é necessário definir um conjunto de propriedades (Figura 45):

- **id** – Identificador;
- **uiGeneratorClass** – Classe do gerador;
- **default** – Indica se é a UIDL a utilizar por omissão;
- Mapeamento entre os objectos da linguagem e os componentes gráficos que os representam, **ComponentsMapper** (Figura 46), **EventsMapper** (Figura 47) e **ActionsMapper** (Figura 48) para componentes, eventos e acções, respectivamente[8].

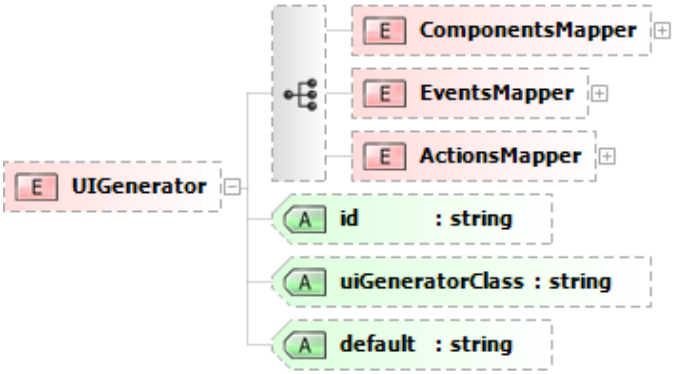


Figura 45 – *Schema* para especificação de linguagens de geração.

[8] Cada componente (*widgets*) que compõe a interface pode ter associado um comportamento. Um comportamento corresponde às acções que o componente deve efectuar quando o utilizador interage com o mesmo. A interacção do utilizador com o componente gera eventos que despoletam a execução das acções. Uma acção pode corresponder à chamada de um método ou a uma mudança na interface gráfica (p.e. transições).

```
<UIDLS>
   <UIDL id="UsiXML" ...>
     <UIGenerator id="AS3" ...>

        <ComponentsMapper>
           <window component = "Classes.Components.FlexiXMLWindow"/>
           <button component = "Classes.Components.FlexiXMLButton"/>
           <textComponent component = "Classes.Components.FlexiXMLText"/>
          <checkBox component = "mx.controls.CheckBox"/>
           …
        <ComponentsMapper>
           …
     </UIGenerator>
   </UIDL>
<UIDLS>
```

Figura 46 - Ficheiro de configuração do mapeamento de componentes UsiXML e os widgets correspondentes.

Na Figura 46 está representado um exemplo de um mapeamento entre os componentes definidos em UsiXML e os *widgets* em ActionScipt 3 que os representam, por exemplo, o componente *window* é representado na interface gerada por um componente denominado "Classes.Components.FlexiXMLWindow".

```
<UIDLS>
   <UIDL id="UsiXML" ...>
     <UIGenerator id="AS3" ...>
           …
        <EventsMapper>
           <release event = "mouseUp"/>
           <depress event = "mouseDown"/>
           <rollOver event = "mouseOver"/>
           <rollOut event = "mouseOut"/>
        </EventsMapper>
           …
     </UIGenerator>
   </UIDL>
<UIDLS>
```

Figura 47 - Ficheiro de configuração para mapeamento de eventos UsiXML e os interpretados pelo gerador.

Na Figura 46 está representado um exemplo de um mapeamento entre os tipos de eventos definidos em UsiXML e os existentes em ActionScipt 3 que os representam, por exemplo, um evento de *release* é representado na interface gerada por um evento de *mouseUp.*

```
<UIDLS>
  <UIDL id="UsiXML" ...>
    <UIGenerator id="AS3" ...>
            …
       <ActionsMapper>
          <transition>
             <boxOut effect="Classes.Animation.Zoom" direction = "OUT"/>
             <boxIn effect="Classes.Animation.Zoom" direction = "IN"/>
             <fadeOut effect="Classes.Animation.Fade" direction = "OUT"/>
             <fadeIn effect="Classes.Animation.Fade" direction = "IN"/>
             <close effect="Classes.Animation.Visibility" direction=
"OUT"/>
             <open effect="Classes.Animation.Visibility" direction =
"IN"/>
            …

          </transition>
       </ActionsMapper>
            …
     </UIGenerator>
   </UIDL>
<UIDLS>
```

Figura 48 - Ficheiro de configuração do mapeamento de transições UsiXML para transições do gerador.

Na Figura 48 está representado um exemplo de um mapeamento entre os tipos de transições definidos em UsiXML e as existentes em ActionScipt 3 que as representam, por exemplo, a transição de *boxOut* é representado na interface gerada por uma transição de *zoom* ("Classes.Animation.Zoom") com uma direcção do tipo "out".

A existência deste tipo de configurações permitem ao utilizador efectuar alterações ao aspecto visual da aplicação gerada, na medida em que aspectos como: mapeamento de componentes e transições podem ser alterados a qualquer momento. Ou seja, um efeito de *boxOut* está mapeado por omissão a uma transição de *zoom*, mas caso o utilizador pretenda outro tipo de efeito, pode alterar este mapeamento por outro.

Actualmente o FlexiXML incluí apenas um gerador em ActionScript 3, mas podem ser integrados outros, desde que implementem o *interface* definido na Figura 49. Ou seja, um gerador deve ser capaz de:

- Adicionar o comportamento a um *widget* (addBehaviourGUIItem);
- Aplicar um estilo à interface gráfica (applyStyle);
- Desenhar um componente gráfico (drawFUIItem);
- Executar transições entre componentes;
- Actualizar o conteúdo de um *widget*.

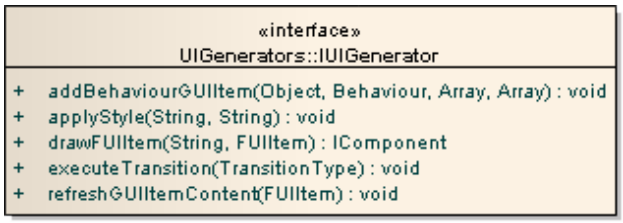


Figura 49 - Interface de um gerador de interfaces FlexiXML.

Relativamente ao desenho dos componentes gráficos, este é feito baseado no mapeamento acima referido. Quanto à definição do *layout* este é definido através de contentores que efectuam o posicionamento vertical e horizontal. Outros contentores podem ser adicionados para dar resposta a uma maior diversidade de *layouts*.

Neste momento FlexiXML permite seleccionar qual a linguagem (declarativa) a utilizar na geração da interface gráfica, embora ainda seja necessária algum trabalho futuro na integração de linguagens adicionais.

Contexto (Idioma)

Caso a especificação da interface gráfica defina vários contextos (idiomas), essa informação fica disponível neste *plugin* (Figura 44, b). Ao seleccionar o idioma o conteúdo da aplicação é actualizado em tempo de execução.

Sempre que o idioma é alterado o Player efectua o pedido ao ContextManager sobre a informação do contexto actual. Posteriormente este plugin efectua um pedido ao ResourceManager que lhe devolve a conteúdo que os componentes deverão ter para o contexto actual. Uma vez obtida esta informação o Player actualiza o conteúdo de todos os componentes presentes na interface gráfica.

Estilos

Depois de gerada a interface gráfica é possível visualizar a mesma segundo diferentes estilos em tempo de execução (Figura 44, c). Cada estilo é definido no formato CSS, o que permite que possam conter característica como imagens (*skins*), fontes, *class selectors*, entre outros.

O *plugin* Player disponibiliza uma lista de estilos pré-definidos mas outros podem ser adicionados sem ser necessário compilar o código. A lista de estilos disponibilizados é definida num ficheiro de configuração XML. Cada estilo deve definir (Figura 50):

- **id** – Identificador do estilo;
- **name** – Nome do estilo;
- **css** – Localização do ficheiro css com as propriedades do estilo;
- **swf** – Localização do ficheiro swf com as propriedades do estilo.

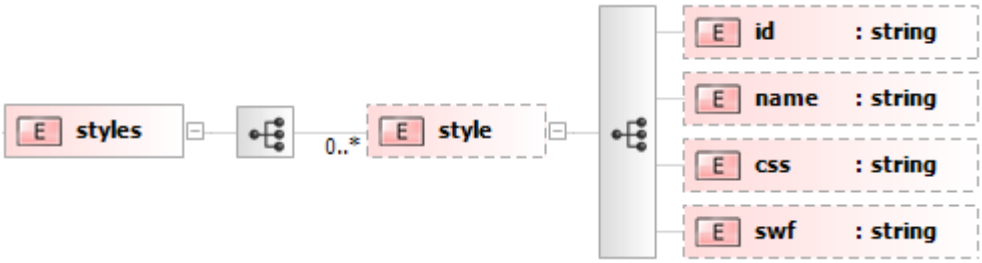


Figura 50 - *Schema* para especificação de um estilo.

Sempre que é seleccionado um novo estilo, o Player notifica o gerador da interface que o estilo foi alterado e como tal a interface deve ser actualizada. No caso do gerador definido por omissão na aplicação FlexiXML, que é o gerador em ActionScript 3, este interpreta o estilo a partir de um ficheiro externo no formato swf (o swf de um estilo surge da conversão de um ficheiro CSS). No caso específico deste gerador, o primeiro passo é validar se o estilo seleccionado já foi

carregado anteriormente, para evitar carregar novamente. Uma vez carregado o estilo, actualiza todos os componentes presentes na interface gráfica com as novas definições.

4.4 *Workflow* do processo de execução

Tendo em consideração o que foi descrito nas secções anteriores, de seguida é apresentado o processo efectuado na ferramenta FlexiXML para geração de uma interface gráfica. Este processo está representado na Figura 51.

A aplicação FlexiXML no processo inicial de carregamento recebe como *input* um conjunto de ficheiros de configuração que são interpretados por gestores adaptados à informação neles contida (Figura 51 - 1, 2 e 3). Estes gestores guardam esta informação, a qual pode depois ser acedida por qualquer *plugin*. Os principais gestores são:

- **Gestor de mensagens (System Messages Manager)** – Gestor responsável pelas mensagens utilizadas na aplicação. Todas as mensagens utilizadas na aplicação têm um código associado, o qual tem definido o respectivo valor num ficheiro de configuração XML (Figura 51 – passos a e 1). A existência deste ficheiro, permite que a aplicação FlexiXML possa ser traduzidas para diferentes linguagens, sem ser necessário recompilar o código;
- **Gestor de *plugins* (*Plugins* Manager)** – Gestor responsável por interpretar o ficheiro com a lista de *plugins* (Figura 51 – passos b e 2) e criar essa estrutura na aplicação;
- **Gestor de *UIDLs* (*Plugins* Manager)** – Gestor responsável por interpretar o ficheiro com a lista de *UIDLs* (Figura 51 **–** passos c e 3) e disponibilizar essa informação caso lhe seja pedido por algum plugins.

Uma vez efectuado este processamento inicial, a aplicação fica disponível com os *plugins* que foram configurados.

Relativamente à geração de uma interface gráfica, para se iniciar este processo FlexiXML recebe como *input* um ficheiro (Figura 51 **–** passos d e 4) onde é indicada a localização do ficheiro

com a descrição da interface (Figura 51 - e) bem como do ficheiro que contém o controlo de diálogo a mesma (Figura 51 - f), passo (4).

Estes ficheiros são interpretados pelo *plugin* Project (Figura 51 - 5) que após os interpretar envia a informação neles contida ao CORE Data Manager (Figura 51 - passos 6 e 7). O *plugin* Project cria entidades que representam os componentes presentes na especificação e que podem depois ser interpretadas pelo *Player*.

O CORE Data Manager centraliza toda a informação que pode ser partilhada por todos os *plugins*. Desta forma, sempre que um *plugin* necessitar de informação sobre o projecto actual efectua o pedido a este gestor.

Só depois de o projecto ser carregado pelo Project é que o Player pode gerar a interface gráfica (Figura 51 - 9) baseado na informação disponibilizada pelo CORE Data Manager (Figura 51 - 8).

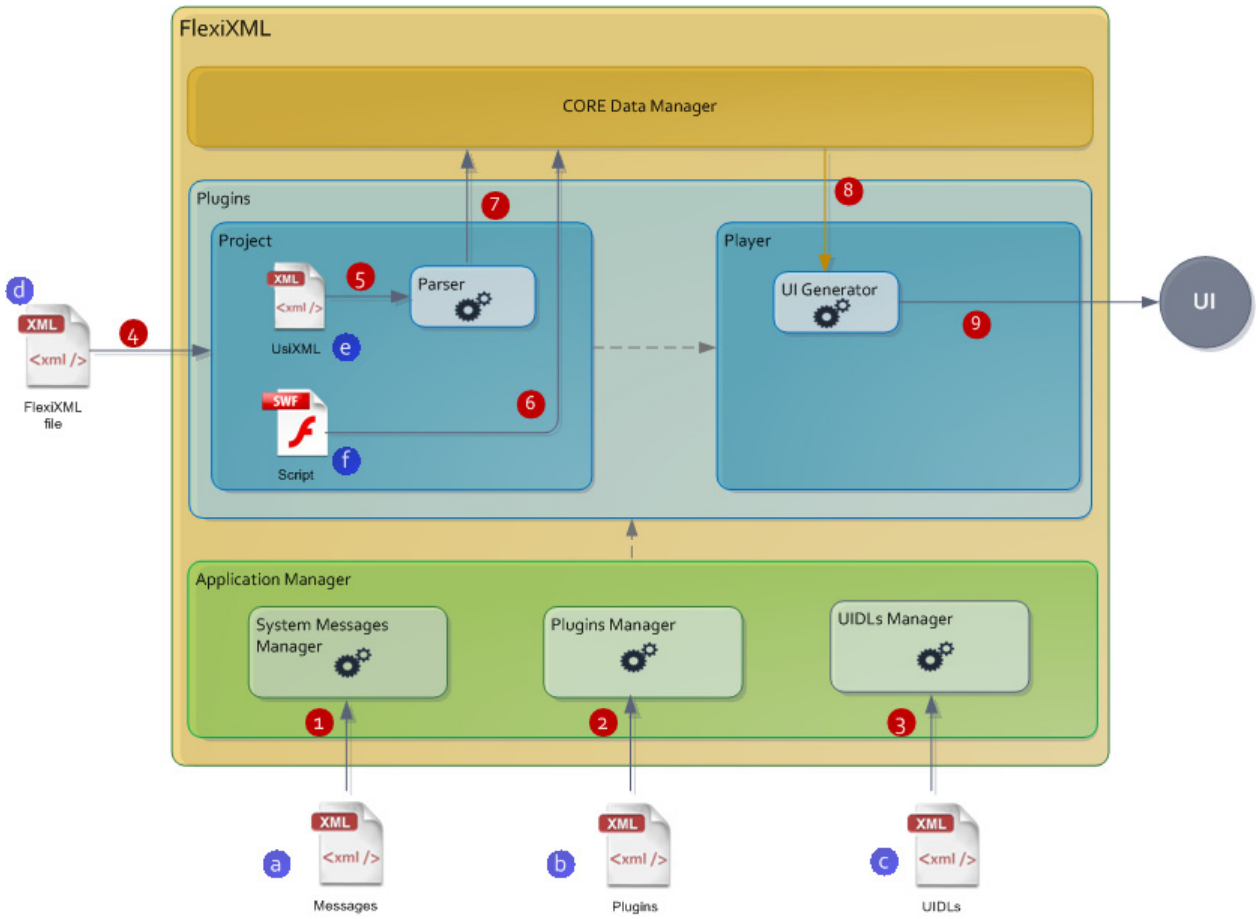


Figura 51 - *Workflow* do FlexiXML.

Capítulo 5

Exemplo Ilustrativo

Nesta secção é apresentado um exemplo de uma interface gráfica gerada utilizando a ferramenta FlexiXML. O exemplo consiste numa aplicação para apresentar e ouvir álbuns de música. Dada a complexidade do modelo, nesta demonstração são apresentados apenas alguns extractos do mesmo.

5.1 Desenho

A aplicação "Music Player" pode ser divida em 4 áreas:

- **Player** – Área onde são apresentados os botões para efectuar o controlo sobre a música que se pretende ouvir.
- **CurrentMusic** – Corresponde à área onde é apresentada a informação da música que está actualmente a tocar;
- **View** – Composta pelos botões que definem os formatos de visualização da lista de álbuns;
- **CurrentView –** Área onde é apresentada a vista actual.

A área View contém 3 botões que permitem alternar entre os 3 formatos de visualização que aplicação apresenta: **ListView** (permite visualizar os álbuns e as respectivas músicas em lista); **GridView** (Visualização de todos os álbuns em grelha); **CoverView** (Visualização de um

álbum a cada momento). O utilizador pode, a qualquer momento, alterar a vista que pretende visualizar.

Tendo em consideração esta estrutura a aplicação pode ser decomposta nas áreas apresentas na Figura 52.

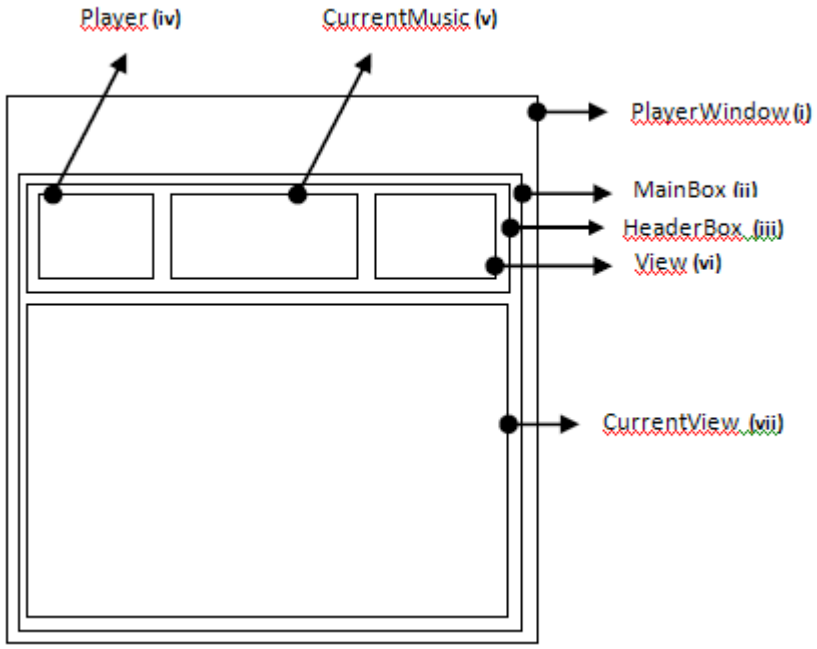


Figura 52 - Decomposição da aplicação em áreas (ver correspondência com Figura 53).

```
<cuiModel id="musicPlayer-cui" name=" musicPlayer-cuiModel">
   <window id="playerWindow" …> (i)
      <box id="mainBox" …>(ii)
         <box id="headerBox" …> (iii)
            <box id="playerBox" …/> (iv)
            <box id="currentMusicBox" …/> (v)
            <box id="viewsBox" …/> (vi)
         </box>
         <box id="currentView" …> (vii)
                        […]
         </box>
      </box>
   </window>
</cuiModel>
```

Figura 53 - CuiModel da aplicação MusicPlayer.

A Figura 53 apresenta a estrutura base do modelo. Nele estão identificados os principais componentes estruturais (contentores principais que dividem cada área). Por questões de espaço os detalhes de cada um deles estão omitidos.

Da especificação iniciada na Figura 53, FlexiXML gera a aplicação apresentada na Figura 56. O resultado final com a especificação completa gera a interface representada na Figura 56.

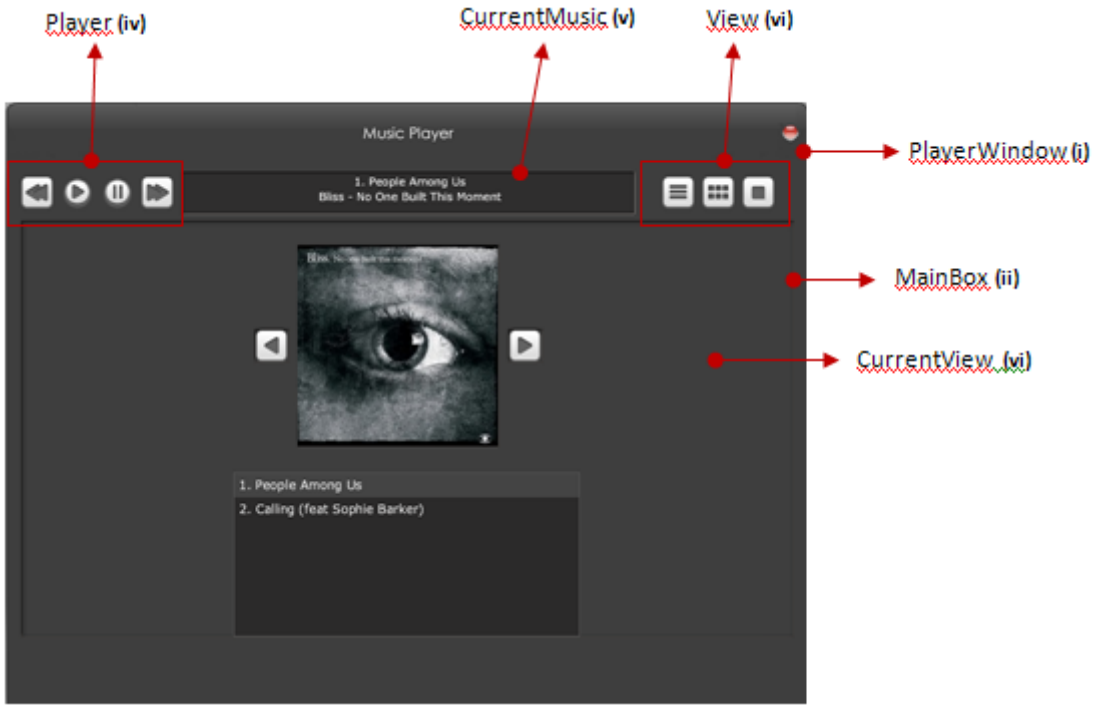


Figura 54 - Music Player (CoverView) (ver correspondência com Figura 52 e Figura 53).

A Figura 54 apresenta a vista do Music Player com as imagens dos álbuns. Para ser possível apresentar mais detalhe sobre o modelo, na Figura 55, está a apresentado como foi especificado as imagens dos álbuns a visualizar.

```
<box borderWidth="0" isBalanced="true" width="202" height="202"
type="stack" id="cover" isVisible = "true">
   <imageComponent id="cover0"
      defaultContent= "Samples/iTunes/playList/Bliss/blissCoverView.jpg"
      width="200" height="200" isVisible="true" />

   <imageComponent id="cover1"
defaultContent="Samples/iTunes/playList/MayraAndrade/mayraCoverView.jpg"
      width="200" height="200" isVisible="false" />
```

```
   <imageComponent id="cover2"
defaultContent="Samples/iTunes/playList/Mentaloscope/mentaloscopeCoverVie
w.jpg" width="200" height="200" isVisible="false" />

    …

</box>
```

Figura 55 - Modelo de para especificação das imagens na vista CoverView.

Na Figura 55 pode-se verificar que o contentor das imagens é uma *stack* (*box* definida com o tipo *stack*), pois só pode ser visualizada uma imagem a cada momento. Cada imagem está definida com o componente *imageComponent,* o qual recebe algumas propriedades, nomeadamente, identificador, localização da imagem, dimensões e a indicação de se a imagem está visível.

Figura 56 - Music player (coverView e gridView) gerado pelo FlexiXML.

5.2 Comportamento

Na aplicação "Music Player" é possível alternar entre as diferentes vistas, pressionando o botão correspondente. Por exemplo, se o botão "GridView" for pressionado, deve ser visualizada esta vista e escondidas as restantes. Para além desta transição, deve ser invocado o método "updateGridView" que é responsável por atribui os dados necessários a esta vista (p.e lista de álbuns a visualizar). A especificação deste exemplo está apresentada na Figura 57 e na Figura 58, o resultado final está representado na Figura 59.

```
<button id="gridButton”>
  <behavior id="gridView">
     <event id="gridViewEvt" eventType="depress" eventContext="gridButton"/>
     <action id="gridViewAct">
        <transition transitionIdRef = "GridViewTr1" />
        <transition transitionIdRef = "GridViewTr2" />
        <transition transitionIdRef = "GridViewTr3" />

        <methodCall methodName = "updateGridView"/>

     </action>
  </behavior>
 </button>
```

Figura 57 - Especificação do comportamento do botão da "*GridView*" .

Na Figura 57 está especificado o comportamento do botão "gridButton" quando este é pressionado (eventType do tipo "depress"). Quando este evento é despoletado é executada a acção "gridViewAct" a qual inclui três transições gráficas e a invocação de um método denominado "updateGridView".

```
<graphicalTransition id="GridViewTr1" transitionType="fadeOut">
   <source id="gridButton" />
   <target id="listView" />
</graphicalTransition>

<graphicalTransition id="GridViewTr2" transitionType="fadeOut">
   <source id="gridButton" />
   <target id="coverView" />
</graphicalTransition>

<graphicalTransition id="GridViewTr3" transitionType="fadeIn">
   <source id="gridButton" />
   <target id="gridView" />
</graphicalTransition>
```

Figura 58 - Especificação das transições gráficas despoletadas pelo botão da "*GridView*".

A definição das transições gráficas indicadas na acção do botão está definida na Figura 58. As transições despoletadas pelo botão "gridButton" consistem em efectuar um *fadeOut* sobre a vista em lista (transição GridViewTr1) e sobre a vista com imagens dos álbuns (transição GridViewTr2) para que estas não fiquem visíveis. De forma a apresentar a vista em grelha é efectuado um *fadeIn* (transição GridViewTr3).

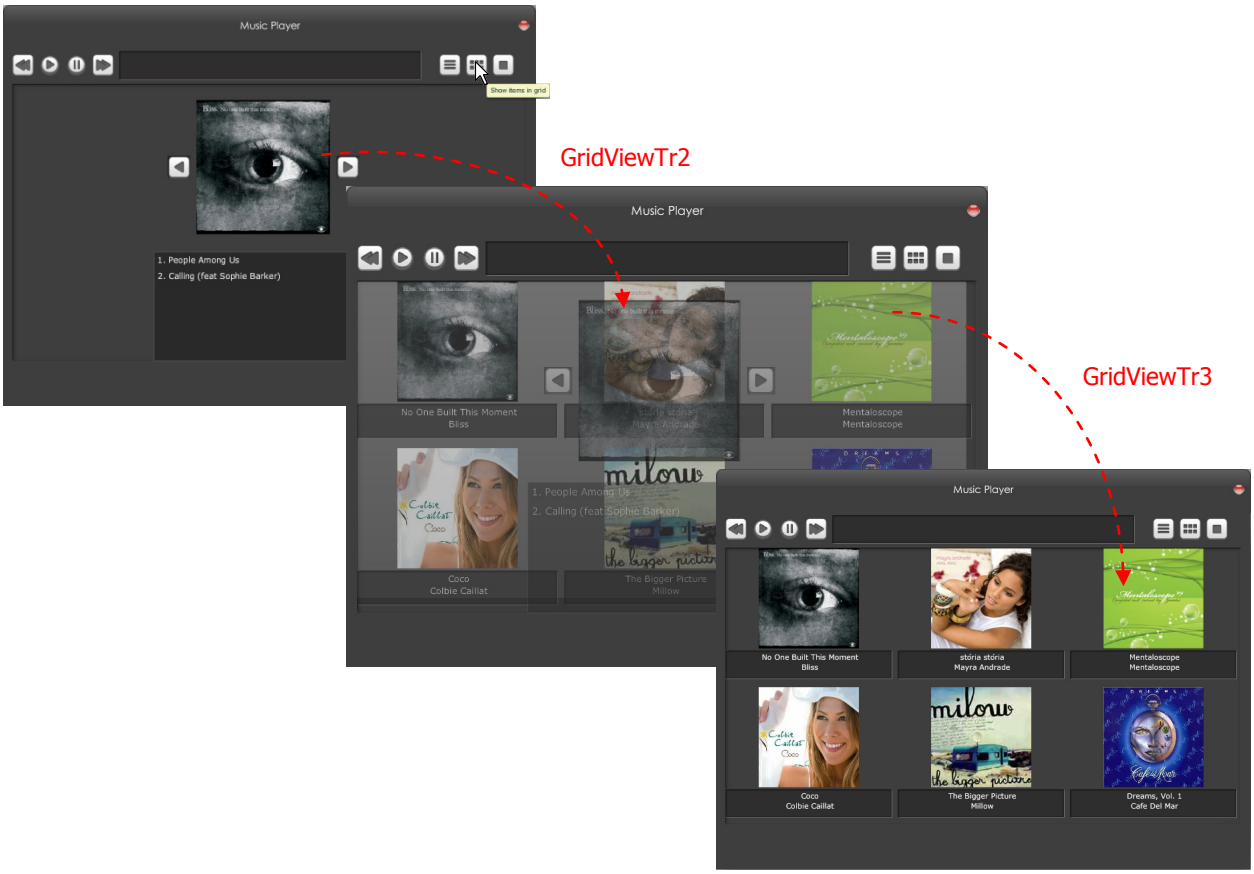


Figura 59 - Transições gráficas para visualização da "GridView".

5.3 Contexto (Idioma)

Para ilustrar a utilização do contexto, descreve-se agora os passos necessários para a disponibilização da interface em dois idiomas.

Para a criação da aplicação em diferentes idiomas é necessário criar dois modelos: o ContextModel e o ResourceModel (conteúdo dos objectos nos diferentes idiomas).

Na aplicação "Music Player" foram criados dois idiomas: inglês e francês (Figura 60).

```
<contextModel id="playerContextModel" name="playerContextModel">
   <context id="playerContext_En_US" name="playerContext_En_US">
      <userStereotype
            id="playerContextUser_US"
            language="en_US"
            stereotypeName="playerContextUser_US"/>
   </context>

   <context id="playerContext_FR" name="playerContext_FR">
      <userStereotype
            id="playerContextUser_FR"
            language="FR"
            stereotypeName ="playerContextUser_FR"/>
   </context>
</contextModel>
```

Figura 60 - Especificação dos idiomas.

Na Figura 61 é apresentada a definição do título da aplicação nos diferentes idiomas.

```
<resourceModel id="playerResourceModel" name="playerResourceModel">
 <cioRef cioId="playerWindow">
   <resource content="Music Player" contextId="playerContext_En_US"/>
   <resource content="Lecteur de Musique" contextId="playerContext_FR"/>
 </cioRef>
</resourceModel>
```

Figura 61 - Especificação do título da aplicação nos diferentes contextos.

Na Figura 62 é apresentada a janela da aplicação com os títulos nos dois idiomas.

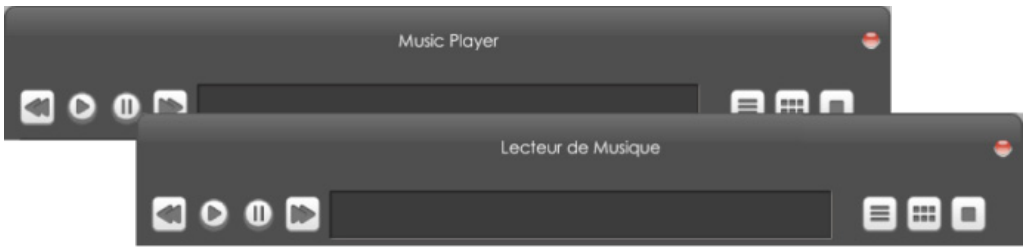


Figura 62 - Aplicação "Music Player" em inglês e francês.

5.4 Estilos

FlexiXML permite ainda a alteração em tempo de execução do estilo da interface gerada. A lista de estilos que podem ser aplicados está definida num ficheiro de configuração (secção 4.3.2), onde é indicado qual o nome e localização do ficheiro com as definições do mesmo (formato CSS e swf).

Cada estilo é definido no formato CSS, o que permite que possam conter característica como imagens (skins), fontes, class selectors, entre outros. FlexiXML interpreta este estilos a partir de um ficheiro externo no formato swf (o swf surge da conversão do ficheiro CSS).

Na Figura 63 é apresentada a *interface* de "Music Player" com diferentes estilos[9].

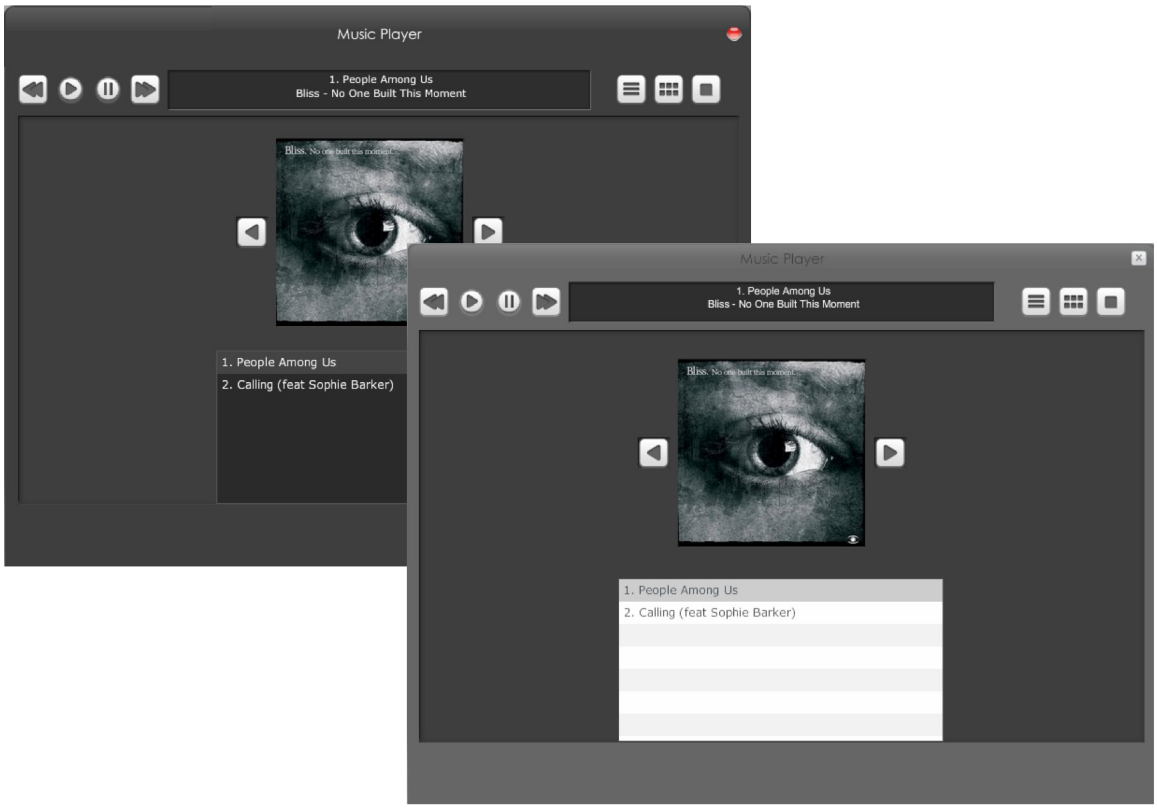


Figura 63 – "Music Player" com diferentes estilos.

[9] Os estilos utilizados na aplicação tiveram por base alguns estilos disponíveis em http://www.scalenine.com/.

Capítulo 6

Conclusões e Trabalho Futuro

A aceitação de uma aplicação depende em grande medida da qualidade da sua interface gráfica. A prototipagem ajuda a que a qualidade da solução possa ser avaliada numa fase inicial [Bäumer, 1996], permitindo que os problemas detectados possam ser analisados atempadamente.

O objectivo deste projecto de mestrado consistiu na criação de uma ferramenta de suporte à linguagem UsiXML, que fosse capaz de interpretar e animar (visualizar) os modelos definidos nesse formato. Não se pretendia criar a interface final, definitiva, mas sim exibir apresentações que pudessem ser testadas e iterativamente adaptadas aos utilizadores.

A UsiXML, a linguagem utilizada neste projecto, permite a especificação da interface com o utilizador a um alto nível de abstracção, independentemente da linguagem de programação e plataforma computacional em que irá ser executada. A ferramenta desenvolvida, FlexiXML, representa uma abordagem à prototipagem de interfaces, permitindo a utilizadores finais interagirem com uma representação da interface modelada.

6.1 Contributos

O principal contributo deste trabalho é, tal como referido acima, a FlexiXML, uma ferramenta de prototipagem de modelos UsiXML. A ferramenta suporta a versão mais recente da

linguagem e foi desenvolvida por forma a ser expansível e configurável. Mesmo não tendo por objectivo a geração da interface final de um dado software, implementa ainda assim, de forma automatizada, uma possibilidade de último passo de reificação tal como definido na *framework* Chameleon (ver Figura 3 na página 28). Os capítulos 4 e 5 apresentaram a ferramenta e um exemplo de aplicação. A secção 6.2, que se segue, apresenta uma análise comparativa das características da ferramenta desenvolvida.

São ainda contributos deste trabalho uma revisão do estado da arte. Esta revisão foi efectuada a dois níveis. No capítulo 2 são identificadas as principais propostas de linguagens de modelação de interfaces, sendo dado especial relevo à UsiXML. No capítulo 3, são identificadas as principais ferramentas de suporte à UsiXML.

Do trabalho realizado resultou ainda uma publicação nas actas do 17º Encontro Português de Computação Gráfica.

6.2 Discussão de resultados

Como referido em capítulos anteriores, FlexiXML surge no seguimento de uma ferramenta já existente: FlashiXML. De seguida, é apresentada uma tabela comparativa entre as duas ferramentas (Tabela 1). Os critérios de comparação foram seleccionados de forma a serem analisados aspectos tecnológicos, funcionais e aspectos relacionados com objectivos definidos para o projecto.

A opção por considerar apenas o FlashiXML para efeitos de comparação deveu-se ao facto de a FlashiXML ser a ferramenta que em termos conceptuais mais se assemelha à FlexiXML. Isto, quer em termos de objectivos, quer em termos de tecnologias de implementação. Com efeito, nos restantes interpretadores de UsiXML identificados (ver secção 3.3) é dada relevância a aspectos como: geração da *interface* para múltiplas plataformas (QtkiXML); simulação de uma *workstation* única (InterpiXML); *interfaces* 3D (HaptiXML), entre outros. Ou seja, trata-se de aspectos que não são relevantes na versão actual de FlexiXML.

Tabela 1 - Tabela comparativa entre FlexiXML e FlashiXML.

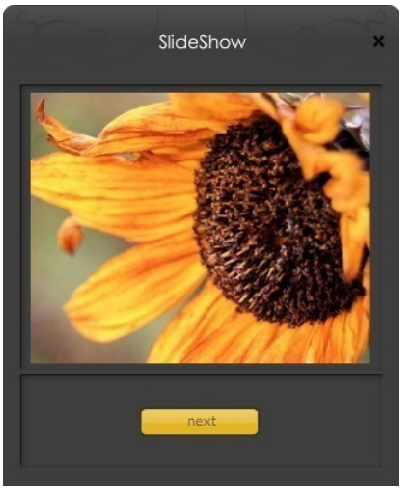
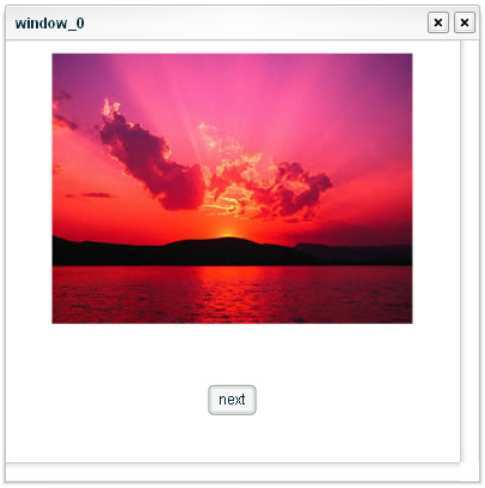
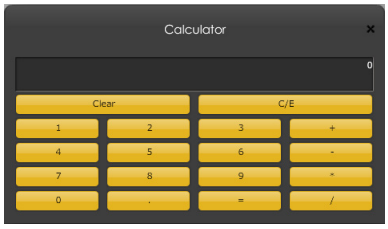
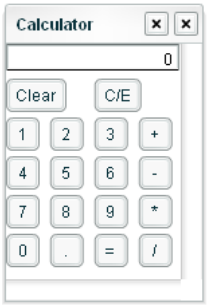
	FlexiXML	FlashiXML
Tecnologia	AIR + Flex 3 (ActionScript 3)	Flash (ActionScript 2)
UILDs suportadas	UsiXML v1.8 (mas com suporte para integração de outras linguagens)	UsiXML v1.4.6
Plugins	Permite a integração de novos plugins.	Não foi criada sob o conceito de plugins.
Componentes	Box (Horizontal, vertical, stack), ComboBox, Button, CheckBox, RadioButton, Text, Image, List, Window. Com a utilização dos ficheiros de configuração (secção 4.3.2) é possível mapear os tipo de componentes UsiXML com qualquer um dos componentes que o FlexiXML suporta.	Box (Horizontal, vertical, stack), ComboBox, Button, CheckBox, RadioButton, Text, Image
Transições gráficas	Blinds, fade, fly, iris, photo, pixel dissolve, rotation, squeeze, tween, visibility, wipe, zoom. Com a utilização dos ficheiros de configuração (secção 4.3.2) é possível mapear os tipo de transições UsiXML com qualquer um das transições que o FlexiXML suporta.	Fade (in/out), visibility (open, close), box (in/out).
Idiomas	Suporta a aplicação dos diferentes idiomas em execução.	Suporta a aplicação dos diferentes idiomas em que a interface é gerada, mas obriga a que a interface

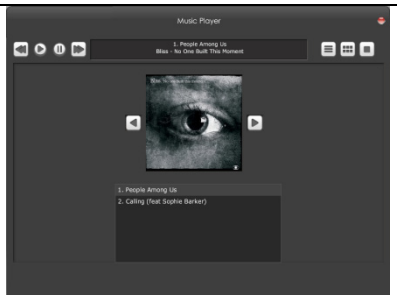
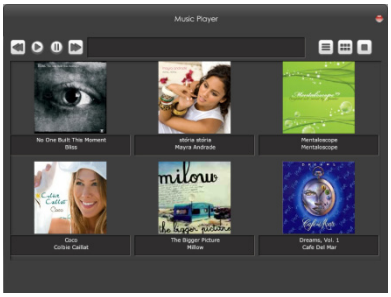
		seja gerada novamente.
Estilos	Suporta a aplicação de diferentes estilos em tempo de execução.	Os estilos aplicados são os existentes por omissão (definidos nos componentes do Flash).
Flexibilidade	Trata-se de uma ferramenta flexível, pois permite que a vários aspectos da interface possam ser configurados, por exemplo: mapeamento de componentes e transições.	Permite a definição de algumas configurações, nomeadamente: comprimento por omissão de uma imagem, a indicação de se a opção de mudança de idioma na aplicação gerada deve ficar disponível, entre outros.

FlexiXML foi sujeita a um conjunto de modelos de interfaces descritas em UsiXML, de forma a testar as diferentes funcionalidades por ela disponibilizadas. Para além de modelos que existiam para o FlashiXML foram criados modelos adicionais. De seguida, são apresentadas algumas interfaces gráficas obtidas em ambas as ferramentas.

Para possibilitar a animação das interfaces em ambos os ambientes foram desenvolvidas versões alternativas dos controladores e diálogo. ActionScript 3 para FlexiXML e ActionScript 2 para FlashiXML.

Tabela 2 Interfaces obtidas em FlexiXML e FlashiXML.

	FlexiXML	FlashiXML
SlideShow		
Calculadora	  Duas versões da calculadora. A especificação UsiXML é igual, mas depois de gerada a interface, foram aplicados estilos diferentes.	
Player de música		Não foi possível visualizar este modelo no FlashiXML.

	 	A parte interactiva desta interface está desenvolvida em ActionScript 3, a qual não é compatível com a versão de FlashiXML.

Tendo em consideração os objectivos inicialmente propostos, e para concluir a apresentação do trabalho realizado, podemos destacar as seguintes características de FlexiXML:

- Implementação numa versão mais recente de tecnologia (AIR, Flex3, ActionScript3). O facto de se tratar de uma tecnologia mais recente por si já é uma vantagem, contudo pode-se salientar também: a utilização de AIR permite que FlexiXML seja independente da plataforma computacional em que é executada e ActionScript 3 tem um melhor desempenho relativamente a ActionScript 2;
- A interpretação de modelos UsiXML na versão mais recente da linguagem: desenho da interface e interacção com a mesma (transições gráficas, invocação de métodos);
- A visualização da interface gerada nos diferentes idiomas em que é definida;
- A possibilidade de utilização de estilos para alterar o aspecto da interface, possibilitando a alteração do estilo das interfaces geradas em tempo de execução;
- A arquitectura da ferramenta está ainda pensada para suportar a integração de novos *plugins*, bem como de novas linguagens de modelação e de programação das interfaces gráficas.

6.3 Trabalho futuro

Um dos objectivos iniciais na estruturação desta ferramenta foi permitir que esta pudesse ser genérica e expansível. Tendo em consideração o trabalho descrito, bem como questões que foram sendo levantadas ao longo da sua execução, existe ainda um conjunto de funcionalidades que podem ser adicionadas de forma a optimizar a ferramenta FlexiXML, dais quais podemos salientar:

- Criação de novos plugins, nomeadamente um editor de modelos;
- Alargar a biblioteca de componentes (*widgets* e contentores com definição de *layouts*), de modo a acomodar um maior conjunto de representações possíveis;
- Geração de protótipos para diferentes dispositivos e plataformas;
- Implementação de *parsers* para novas UIDLs e geração em diferentes linguagens de programação.

Apesar da execução da lista de tarefas identificar acima possibilitar a obtenção de uma ferramenta ainda mais completa e útil, a versão actual está já funcional.

Siglas e acrónimos

Cameleon - Context Aware Modelling for Enabling and Leveraging Effective interactiON

CIO - Concrete Interaction Objects

CUIR - Concrete User Interface Relationships

IHC - Interacção Humano-Computador

MDA - Model-Driven Architecture

RIA - Rich Internet Application

UIDL – User Interface Description Language

UIMS - User Interface Management System

UsiXML - USer Interface eXtensible Markup Language

Bibliografia

[Abrams, 1999] Abrams M., C. Phanouriou, A.L. Batongbacal, S.Williams, and J. Shuster. UIML: *An Appliance-Independent XML User Interface Language*. In A. Mendelzon, editor, Proc. of 8th Inter. World-Wide Web Conference WWW'8 (Toronto, May 11-14, 1999), Amsterdam, 1999. Elsevier.

[Argollo, 1997] Argollo M. Jr. and Olguin *C.. Graphical user interface portability*. CrossTalk: The Journal of Defense Software Engineering, 10(2):14–17, 1997.

[Arsanjani, 2002] Arsanjani A. et al. *(WSXL) Web Service Experience Language Version 2*. IBM Note 10 April 2002. IBM, 2002.

[Bäumer, 1996] Bäumer, Dirk . et tal. User interface prototyping - concepts, tools, and experience. ICSE '96: Proceedings of the 18th international conference on Software engineering. IEEE Computer Society. Berlin, German.1996.

[Campos, 2004] Campos, J. C. (2004). Análise de usabilidade baseada em modelos. *In Interacção 2004 - 1ª Conferência Nacional em Interacção Pessoa-Máquina*, (pp. 171-176).

[Campos, 2006] Campos, J. C., & Harrison, M. (2006). *Encyclopedia of Human-Computer Interaction*. (I. G. Reference, Ed.) In Claude Ghaoui.

[Collignon, 2008] Collignon, B., Vanderdonckt, J., Calvary, G., *Model-Driven Engineering of Multi-Target Plastic User Interfaces*, Proc. of 4th IARIA International Conference on Autonomic and

Autonomous Systems ICAS'2008 (Gosier, 16-21 March 2008), IEEE Computer Society Press, Los Alamitos, 2008, to appear.

[Coyette, 2007] Coyette, Adrien, *A Methodological Framework for Multi-Fidelity Sketching of User Interfaces*, Ph.D. thesis, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, 22 October 2007.

[Dan R. Olsen, 1992] Dan R. Olsen, J. (1992). *User Interface Management Systems: Models and Algorithms.* Morgan Kaufmann Publishers.

[Denis, 2005] Denis, Vincent, *Un pas vers le poste de travail unique : QTKiXML, un interpréteur d'interface utilisateur à partir de sa description*, M.Sc. thesis, Université catholique de Louvain, Louvain-la-Neuve, Belgium, September 2005.

[Dix, Finlay, D., & Person, 2004] Dix, A., Finlay, J., D., G., & Person, R. (2004). *Human-Computer Interaction* (3rd Edição ed.).

[Grolaux, 2007] Grolaux, Donatien, *Transparent Migration and Adaptation in a Graphical User Interface toolkit*, Ph.D. thesis, Université catholique de Louvain, Louvain-la-Neuve, Belgium, 4 September 2007.

[Drakos, 1996] Drakos, N. (1996). *Computer Based Learning Unit*. University of Leeds.

[Eng, 2005] *Engineering Human Computer Interaction and Interactive Systems* (Vol. Volume 3425/2005). (2005). Springer Berlin / Heidelberg.

[Furtado, 2004] Furtado, E., Furtado, V., Soares Sousa, K., Vanderdonckt, J., Limbourg, Q., *KnowiXML: A Knowledge-Based System Generating Multiple Abstract User Interfaces in UsiXML*, Proc. of 3rd Int. Workshop on Task Models and Diagrams for user interface design TAMODIA'2004 (Prague, November 15-16, 2004), Ph. Palanque, P. Slavik, M. Winckler (eds.), ACM Press, New York, 2004, pp. 121-128.

[Goffette, 2007] Y. Goffette, H.-N. Louvigny, *Development of multimodal user interfaces by interpretation and by compiled components : a comparative analysis between InterpiXml and OpenInterface*, M.Sc. thesis, UCL, Louvain-la-Neuve, 28 August 2007

[Green, 1985] Green, M. (1985). The University of Alberta user interface management system. *Proceedings of SIGGRAPH'85, 12th Annual Conference (San Francisco, Calif. July 22-26)* (pp. 205-213). New York: ACM.

[Guerreiro, 2008] Guerreiro, N., Mendes, S., Pinheiro V. and Campos, J. C. *AniMAL - a user interface prototyper and animator for MAL interactor models*. In Actas da 3a. Conferência Nacional em Interacção Pessoa-Máquina (Interacção 2008). pp 93-102, GPCG, 2008.

[Hartson, 1984] Hartson, H. E. (1984). A human-computer dialogue management system. *Proceedings of INTERACT'84, First IFIP Conference on Human-Computer Interaction (London, Sept)* (pp. 57-61). International Federation for Information Processing.

[Lepreux, 2007] Lepreux, S., Hariri, A., Rouillard, J., Tabary, D., Tarby, J.-C., Kolski, Ch., *Towards Multimodal User Interfaces Composition based on UsiXML and MBD principles*, Proc. of 12th Int. Conf. on Human-Computer Interaction HCI International'2007 (Beijing, 22-27 July 2007), Part III, J. Jacko (ed.), Lecture Notes in Computer Science, Vol. 4552, Springer-Verlag, Berlin, 2007, pp. 134-143

[Luyten, 2002] Luyten K., Vandervelpen C., and Coninx K.. *Adaptable user interfaces in compponent based development for embedded systems.* In Proceedings of the 9th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2002, (Rostock, June 12-14, 2002). Springer Verlag, 2002.

[Mendes, 2009] Mendes, S. and Campos, J. C. FlexiXML – Um animador de modelos UsiXML. 17º Encontro Português de Computação Gráfica, 2009.

[Michotte, 2008] Michotte, B., & Vanderdonckt, J. (2008). GrafiXML, A Multi-Target User Interface Builder based on UsiXML. *Proc. of 4th International Conference on Autonomic and Autonomous Systems ICAS'2008 (Gosier, 16-21 March 2008), IEEE Computer Society Press.* Los Alamitos.

[Montero, 2006] Montero, F., López-Jaquero, V., *IdealXML: An Interaction Design Tool-A Task-Based Approach to User Interfaces Design*, Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2006 (Bucharest, 6-8 June 2006), Chapter 20, Springer-Verlag, Berlin, 2006, pp. 245-252.

[Muller, 2001] Muller A., Forbrig P., and Cap C. H.. *Model-based user interface design using markup concepts*. In Ch. Johnson (Eds.), editor, In Proc. Of 8th International Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'2001 (Glasgow, 13-15 Juin 2001), pages 16–27, Berlin, 2001. Springer-Verlag.

[Norman, 1999] Norman, D. A. (1999). *The Invisible Computer: Why Good Products Can Fail, the Personal Computer Is So Complex, and Information Appliances Are the Solution*. The MIT Press.

[Norman, 1988] Norman, D. A. (1988). *The Psychology of Everyday Things.* (B. Books, Ed.) New York.

[Paternò & Santoro] Paternò, F., & Santoro, C. (s.d.). *A unified method for designing interactive systems adaptable to mobile and stationary plataforms*, Interacting with Computers, Volume 15, Issue 3, June 2003, Pages 349-366.

[Paternó, 2002] Paternó. F and Santoro. C. *One model, many interfaces*. In Ch Kolski and J. Vanderdonckt (Eds.), editors, Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces CADUI'2002 (Valenciennes, 15-17 May 2002), pages 143–154, Dordrecht, 2002. Kluwer Academics Publishers.

[Preece, 1994] Preece, J. (1994). *Human-Computer Interaction*. New York, NY: Addison-Wesley Publishing Company.

[Puerta, 2002] A. Puerta and J. Eisenstein. *XIML: A common representation for interaction data*. In Proc. Of the 7th International Conference on Intelligent User Interfaces (Santa Fe, United States, January 2002), pages 69 – 76., New York, 2002. ACM Press.

[Santa et al.] Santa, L., Pereira, R. and Silvestrin, G. *Aplicação de Model-Driven Architecture no Deployment de Aplicações Distribuídas*. Instituto de Informática, Universidade Federal do Rio Grande do Sul, Brazil.

[Schneiderman, 1998] Schneiderman, B. (1998) *Design the User Interface* - 3rd Edition. Reading, Mass.: Addison-Wesley

[Silva, 2006] Silva, E. (2006). *Sistemas interactivos*. Departamento de Computação, Universidade Federal de Ouro Preto.

[Souchon, 2003] Souchon, N., Vanderdonckt, J., *A Review of XML-Compliant User Interface Description Languages*, Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003 (Madeira, 4-6 June 2003), Jorge, J., Nunes, N.J., Falcao e Cunha, J. (Eds.), Lecture Notes in Computer Science, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 377-391.

[Vanderdonckt, 2005] Vanderdonckt, J. (2005). A MDA-Compliant Environment for Developing User Interfaces of Information Systems. *Proc. of 17th Conf. on Advanced Information Systems. Vol. 3520.* Springer-Verlag, Berlin: O. Pastor & J. Falcão e Cunha (eds.), Lecture Notes in Computer Science.

[Vanderdonckt et al.] Vanderdonckt, J., Limbourg, Q., Michotte, B., Bouillon, L., Trevisan, D., Florins, M., *UsiXML: a User Interface Description Language for Specifying Multimodal User Interfaces*, in Proc. of W3C Workshop on Multimodal Interaction WMI'2004 (Sophia Antipolis, 19-20 July 2004).

[Youri, 2004] Youri, Vanden B. *Etude et implémentation d'un générateur d'interfaces vectorielles à partir d'un langage de description d'interfaces utilisateur*, Universite Catholique de Louvain.2004.

[Zimmermann, 2002] Zimmermann G., Vanderheiden G., and Gilman A.. *Universal remote console-prototyping for the alternate interface access standard*. In N. Carbonell and C. Stephanidis, editors, Universal Access: Theoretical Perspectives, Practice and Experience - 7th ERCIM UI4ALL Workshop (Oct. 2002, Paris, France). Springer-Verlag, 2002.

Referências WWW

[W1]http://www.rosenfeldmedia.com/announcements/2009/02/future_practice_interview_bill.php

[2009/02/24]

Rosenfeld Media publica informação sobre área de *design* de interface com o utilizador.

[W2] http://www.usixml.org/ [2009/06/26]

Université catholique de Louvain. *UsiXML – USer Interface eXtensible Markup Language*. http://www.usixml.org/. Acedido em 26 de Junho de 2009.

[W3] http://www.tcl.tk/ [2009/05/29]

[W4] http://www.informatik.uni-rostock.de/koll_v_ss07_2.html [2009/05/29]

[W5] http://forge.morfeo-project.org/wikis/sfe-public/index.php/UsiXML [2009/05/10]

[W6] http://kaklanis.googlepages.com/nickkaklanis-3dhapticwebbrowser [2009/05/29]

Nick Kaklanis. 3D HapticWebBrowser.

[W7] https://developer.mozilla.org/en/XUL_Tutorial [2009/06/26]

Mozilla foundation. *XUL Tutorial*,. Acedido em 26 de Junho de 2009.

[W8] http://giove.cnuce.cnr.it/cameleon.html [2009/04/19]

The Cameleon Project – plasticity of user interfaces.

[W9] http://www.cs.kent.ac.uk/people/staff/saf/patterns/plml.html [2009/08/30]

Site da Universidade de Ciências de Computação de Kent.

[W10] https://www-306.ibm.com/software/ucd/

Esta área do *site* da IBM fornece informação sobre desenho de interfaces gráficas: conceitos, princípios e *guidelines*.

[W11] http://www.usabilityprofessionals.org/usability_resources/usability_in_the_real_world/benefits_of_usability.html [2009/09/29]

[W12] http://www.iso.org/iso/catalogue_detail.htm?csnumber=16883 [2009/09/29]

Site da ISO (International Organization for Standardization)

[W13] http://www.adobe.com/ [2009/10/25]

Site da Adobe.