Universidade do Minho
Escola de Engenharia

Paulo Jorge Paradela Camacho

Geração Dinâmica de Interfaces

Julho de 2014

Universidade do Minho
Escola de Engenharia

Paulo Jorge Paradela Camacho

Geração Dinâmica de Interfaces

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de

Professor José Creissac Campos

Julho de 2014

Abstract

The goal of this work is to support the dynamic generation of user interfaces, based on a generic model which can quickly adjust the interface of an application due to minor adjustments in the values of the model itself. The work is set in the context of optometric exams' management. The objective is to not have to modify the base application's code whenever new requirements are added: exams, assignments or parameters, since the interface will be able to dynamically respond to these new requirements. It also an objective to support the dynamic selection of tests, tailored to customer needs, and to link them to optometric exams.

Resumo

Pretende-se suportar a geração dinâmica de interfaces de utilizador baseada num modelo genérico que permita ajustar rapidamente a interface das janelas de uma aplicação, em função de pequenos ajustes nos valores do próprio modelo. Este trabalho foi desenvolvido no contexto da gestão de exames de optometria. O objetivo é não ser necessário modificar o código base da aplicação sempre que são acrescentados novos requisitos: exames, parâmetros ou designações, já que a interface será capaz de responder dinamicamente a estes novos requisitos. Pretende-se ainda possibilitar a seleção dinâmica de exames, adaptada às necessidades dos clientes, e associá-los a testes de optometria.

Agradecimentos

A realização da dissertação de mestrado só foi possível graças à disponibilidade e ao apoio, de várias pessoas e instituições, às quais gostaria de manifestar palavras de agradecimento, em especial:

Ao Professor José Creissac Campos, pela disponibilidade em orientar este trabalho, pela revisão crítica do texto, esclarecimentos e sugestões, pela indicação da bibliografia relevante para a temática em análise, pela amizade e pela confiança que sempre me concedeu.

À empresa F3M, Information System, SA, pela possibilidade de realização do presente trabalho e por todos os meios colocados à disposição, que foram úteis para esta dissertação, esperando que esta dignifique, esta instituição.

Por último, à minha família, pelo amor, compreensão e pelo encorajamento a fim de prosseguir a elaboração deste trabalho.

A todos reitero a minha gratidão.

Índice de ilustrações

Índice

Capítulo I – Introdução

Atualmente, num mercado tecnológico em constante mudança, a pesquisa de soluções que optimizem o processo de desenvolvimento de software é um fator da maior relevância já que, para além de inúmeras alterações dos requisitos, obrigações legais e funcionalidades, as aplicações devem estar preparadas para as constantes evoluções no próprio ambiente tecnológico. Surge assim a necessidade do Software se adaptar de forma rápida e escalável a novos requisitos e à alteração dos existentes.

Esta é a principal motivação para o desenvolvimento de uma ferramenta que possibilitasse a geração dinâmica de interfaces gráficas de utilizador para a aplicação WinOPT, com base no preenchimento de um modelo.

1.1. Aplicação WinOPT

A aplicação WinOPT – Gestão de Ópticas (F3M - Information Systems, S.A.) é uma aplicação, que tem como destinatários uma gama de clientes, da área da Óptica, com diferentes necessidades a incidir sobre o mesmo produto. Nesta aplicação podemos ter uma configuração de cliente único com uma loja só, ou um conjunto de lojas do mesmo cliente, ou até um conjunto de empresas com várias lojas. Deste modo, a mesma aplicação deve apresentar diferentes interfaces de utilizador e comportamentos, sobre o mesmo ambiente, para responder a distintos tipos de clientes ou configurações. Por exemplo, de cliente para cliente, o conjunto de exames disponíveis é muito variado o que implica uma apresentação da interface de forma dinâmica, em função da especificidade de cada cliente, que pretende selecionar no momento os exames oftalmológicos pretendidos.

Este requisito implica que a aplicação tem lidar com configurações diferentes, modificar a interface do utilizador e o seu comportamento em função das parametrizações definidas, sem

modificar propriamente o código base da aplicação. Em consequência, são necessárias formas de agilizar e automatizar a criação das interfaces adaptadas a cada cliente.

Outra dificuldade é adaptar a aplicação de forma rápida quando se pretende inserir novos exames oftalmológicos na interface ou alterar os existentes, ou seja, é preciso alterar a interface, de uma forma rápida e eficaz, sem grande impato no desenvolvimento.

Figura 1 - Aplicação WinOPT – Gestão de Ópticas.

1.2. Conceito de Engenharia de Linha de Produto

O WinOPT (ver Figura 1) é um exemplo de uma aplicação que pode ser analisada na perspetiva Software *Product Line*, uma vez que permite múltiplas configurações e parametrizações.

A abordagem Software Product Line (SPL), de acordo com Kruger (2006), é um conceito que visa ajudar a definir e desenvolver uma linha de produtos similares, com variações nas suas caraterísticas e funcionalidades. Este processo decorre ao longo de cada etapa do ciclo de desenvolvimento de software, ou seja, na definição de requisitos, na concepção, no desenvolvimento e nos testes, conforme é mostrado no esquema da Figura 2.

Uma das caraterísticas que distingue as linhas de produtos de software de outros processos de desenvolvimento é que, a reutilização de software é preditiva, isto é, ao invés de colocar os componentes de software numa biblioteca à espera que venham a ser reutilizados, as

linhas de produtos de software só criam artefatos de software quando a reutilização é previsível em um ou mais produtos ou numa linha de produtos bem identificados.

Das inúmeras vantagens da abordagem SPL, segundo Kruger (2006), destacam-se as seguintes:

- Permitir um aumento na diversidade de produtos oferecida, sem implicar o correspondente aumento nos recursos.
- Incrementar a eficiência e a produtividade através da redução do custo de desenvolvimento por produto.
- Diminuir o tempo de lançamento de produtos novos e/ou atualizações de produtos existentes, reduzindo assim o tempo de resposta a novas oportunidades e alterações de mercado.
- Incrementar a qualidade do produto e melhorar a gestão do risco associado.
- Facilmente optar pelos produtos ou caraterísticas mais rentáveis sem ter que estar à espera da rentabilização de produtos a longo prazo e permitir relançar facilmente produtos inativos.

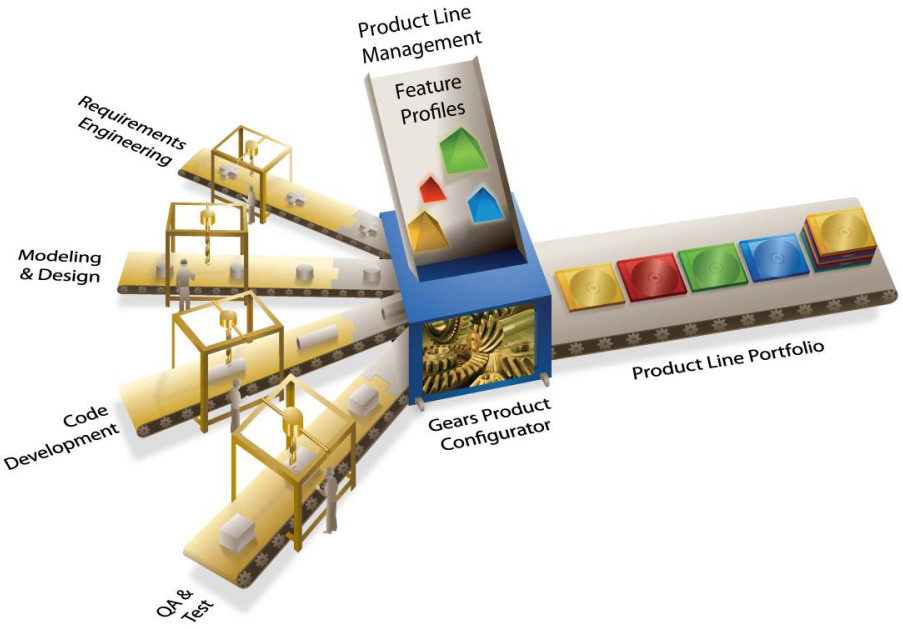

Figura 2 - Product Line Engineering Framework (BigLever software).

O SPL é atualmente uma abordagem em crescimento que está a auxiliar as empresas a atingir a multidiversidade de produtos com a velocidade e eficiência, necessárias, para satisfazer os consumidores e as empresas. Um exemplo muito conhecido é o da indústria automóvel, onde o ponto fulcral está na implementação de uma única linha de produção, na qual um único produto, é produzido com muitas variações, reduzindo assim os custos de criação e manutenção da linha. O WinOPT pode assim beneficiar da abordagem SPL, em particular na componente de interface, que é bastante importante na aplicação e sofre constantes modificações. Em particular no módulo de exames de optometria, onde existe um vasto conjunto de exames disponíveis para serem realizados.

O que é pretendido neste projeto, é que este seja um ponto de partida para a inclusão do conceito de linha de produto de software (SPL) nos produtos da empresa. Ou seja, para agilizar de uma forma sistemática o desenvolvimento de diferentes produtos sobre a mesma plataforma, ao contrário de disponibilizar diferentes produtos totalmente independentes. Uma das formas de atingir este objetivo é ter componentes (da interface) que se adaptam em tempo de execução.

1.3. Geração Automática de Interfaces

Atualmente as empresas de desenvolvimento de Software despendem uma parte significativa de seu tempo construindo e ajustando a interface de utilizador. Com base num inquérito de efetuado sobre o tempo gasto no desenvolvimento da interface de utilizador, Myers e Rosson (1992), estimam que nas aplicações, em média 48% do código pertence à definição da interface de utilizador, e acrescentam que o tempo gasto no desenvolvimento da interface do utilizador é em média cerca de 45% durante a fase de desenho, 50% durante a fase de implementação, e cerca 37% durante a fase de manutenção. Por conseguinte qualquer nível de automação nesta área terá um claro benefício no desenvolvimento de aplicações de software.

De acordo com, Nichols e Faulring (2005), a geração automática da interface do utilizador é um tema largamente discutido na área de desenvolvimento de aplicações de software. Um enorme conjunto de ferramentas de software de geração da interfaces foram criadas para ajudar o programador a criar a interface de utilizador. A geração mais recente

destas ferramentas é denominada como a das ferramentas de nível superior (*high lever tools*), segundo referem Schlungbaum e Elwert (1996).

Segundo Kennard e Steele (2008) existem bastantes abordagens que procuram dar resposta à automatização da interface de utilizador, nomeadamente projetos como, COUSIN (Hayes et al., 1985), TRIDENT (Bodart et al., 1995), Naked Objects (Pawson, 2002), UsiXML (Vanderdonckt et al., 2004) e AUI (Lu e Wan, 2007), só para citar alguns.

Ainda de acordo com Kennard e Steele (2008), os projetos de desenvolvimento usam geralmente uma de três abordagens para o desenvolvimento da interface de utilizador:

1. Ferramentas baseadas em linguagens – neste caso, a interface é programada de forma mais direta, utilizando linguagens específicas para a interface.
2. Ferramentas interativas de especificação gráfica – por exemplo, os editores gráficos presentes em ambientes de desenvolvimento de software,
3. Ferramentas de geração automática da interface de utilizador baseadas em modelos – ver (Meixner et al., 2011),

As ferramentas baseadas em linguagens, bem como as ferramentas de especificação gráficas interativas, estão comercialmente disponíveis e têm tido bastante sucesso nas empresas das Tecnologias de Informação e Comunicação (TIC). Estas ferramentas são frequentemente usadas, mas ainda assim o desenvolvimento da interface de utilizador, usando estas ferramentas, é uma atividade difícil, suscetível a erros e que consome muito tempo de desenvolvimento. As ferramentas baseadas na linguagem permitem a especificação do controlo da interface do utilizador de uma forma relativamente simples. No entanto o problema é que o programador deve especificar para cada *layout*, a posição, o formato e a ordem de cada componente da interface de utilizador através de codificação, o que constitui um trabalho moroso.

Num lado diametralmente oposto, com auxílio de ferramentas de especificação gráficas interativas, o *designer* cria o *layout* da interface de utilizador de forma interativa, e a criação do layout pode até ser construída por não-programadores. Esta até parece ser a forma mais natural para desenvolver a interface de utilizador, no entanto o principal problema desta abordagem é que a especificação de controlo de diálogo tem que ser acrescentada através de uma linguagem

de programação ou através da utilização de uma linguagem específica para a definição do controlo numa interface de utilizador.

Uma outra falha importante nas duas abordagens anteriores é que os resultados da análise de requisitos, na maioria das atuais aplicações de software, não podem ser inseridos diretamente no desenvolvimento do *layout* sem intervenção do programador/designer. Um exemplo é quando se pretende acrescentar novas opções nas aplicações ou mudar radicalmente todo a interface deste, o custo de desenvolvimento e teste é muito elevado e é um obstáculo à revitalização das aplicações. Daí, a sugestão da aplicação da geração automática de interfaces de utilizador como parte essencial do desenvolvimento de ambientes de interface de utilizador no futuro próximo. Nesse sentido foram construídas várias aplicações de desenvolvimento da interface de utilizador com base em modelos para resolver os problemas assinalados nas ferramentas anteriores (Meixner et al., 2011).

A abordagem baseada em modelos apresenta muitas vantagens em potência em relação aos métodos tradicionais de construção de interfaces de utilizador, como por exemplo, o *design* eficiente, as ferramentas de geração em *runtime*, o auxílio na definição conceitual, na reutilização e consistência do software, e no próprio desenvolvimento iterativo e integrado das aplicações de software. No entanto, esta abordagem ainda necessita de mais análise, já que as interfaces de utilizador que são criados, não são ainda suficientemente boas ou práticas, limitando assim a sua aplicabilidade nos sistemas atuais. Para além disso, as linguagens de especificação associadas a estes modelos são muito complexas e difíceis de aprender e utilizar.

1.4. Objetivos e Resultados Esperados

O objetivo deste trabalho é desenvolver uma solução que permita gerar o *layout* da interface de gestão de exames de optometria, para que esta se possa adaptar à mudança contínua das regras/requisitos do negócio. Para facilitar esse trabalho, pretende-se ainda criar e tipificar *layouts* tipo para as janelas padrão da aplicação, para que esta gere *layouts* gráficos distintos em função da tipologia da janela. A mais longo prazo o objetivo é contribuir para tornar o processo de desenvolvimento de aplicações mais produtivo, mais simples e com melhor qualidade, diminuindo o custo de manutenção e de evolução do software e aumentando assim o ciclo de vida da aplicação.

Este trabalho propõe a geração da interface no momento de execução, reduzindo a necessidade de utilizar ferramentas interativas de especificação gráfica e ferramentas de geração de Interface de Utilizador (UI – do inglês *User Interface*) baseadas em linguagens. A ideia proposta é seguir as metodologias associadas à geração de Interface de Utilizador baseadas em modelos, simples e com pouco controlo de diálogo, contudo gerando a interface em tempo de execução e não de compilação. Espera-se, assim, que a geração de interfaces de utilizador possa ser feita de forma mais prática, desde que estabelecendo limites úteis para o processo de geração. O trabalho será finalizado com a elaboração de um protótipo que irá implementar esta ideia.

Na construção do protótipo, a implementar neste trabalho, almeja-se atingir as seguintes metas:

1. Não escrever o código fonte, no que diz respeito à interface, mas sim gerar a interface do sistema em tempo de execução.
2. Mudar facilmente todo a interface de uma aplicação.
3. Mudar rapidamente um componente obsoleto da aplicação por outro mais recente.
4. Padronizar a interface da aplicação.
5. Diminuir o custo de desenvolvimento e o custo de correção de erros relativo à interface de utilizador.

Como resultado deste trabalho, pretende-se obter os seguintes produtos resultantes da aplicação da abordagem seguida:

1. Definição de um modelo de estruturação da interface que possa ser escalável para outros módulos da aplicação WinOPT e se possível para outros ambientes onde esta seja implementada.
2. Elaboração de um protótipo ou mecanismo de criação dinâmica de exames no módulo de testes de optometria, no ambiente de produção da aplicação WinOPT, que seja facilmente expandido para novos exames.

Na concretização deste trabalho, podem ser identificadas um conjunto de vantagens que seguidamente se identificam:

1. Agilização na manutenção das aplicações, já que não existe a necessidade de alterar a interface sempre que esta sofra modificações nos requisitos.
2. Eliminação da tarefa de criação e configuração de controlos nas janelas, quando estes existem em grande número, sendo esta uma tarefa repetitiva e entediante, aumentando assim a probabilidade de erro, e aumentando a produtividade e eficácia e no desenvolvimento de aplicações.
3. Diminuição substancial do tamanho da própria aplicação e do seu executável, sendo independente do número de janelas já que são todas com base no mesmo código fonte.
4. Elevado nível de padronização nas interfaces da aplicação, já que são geradas da mesma forma, evitando assim interfaces distintas desenvolvidas com base nas preferências de diferentes programadores.

1.5. Estrutura do Documento

Este documento acompanha a ordem cronológica do desenvolvimento do trabalho proposto. Para além da introdução, este documento está estruturado nos seguintes capítulos:

- Geração Automática de Interfaces – Este capítulo contém o resultado da análise de algumas das abordagens existentes atualmente na área da geração automática de interfaces. Neste capítulo são detalhadas as arquiteturas existentes para a construção da interface de utilizador de forma mais eficiente. São avaliadas as suas vantagens ou desvantagens de forma a ter em conta aquando da elaboração o sistema pretendido. Aqui são também apresentadas as lacunas das metodologias dos sistemas anteriores, em relação às necessidades da aplicação que se pretende melhorar.
- Metodologia RunUI (Runtime UI) - Neste capítulo é apresentada a abordagem RunUI e como esta combina as vantagens dos modelos de referencia estudados, especialmente os modelos *Camaelon framework* e o *Model-Driven Asset Generation*. É apresentado um esquema da aplicação e a sua relação com o modelo idealizado. Com base neste modelo é apresentado um esboço do protótipo e da sua estrutura, onde são apresentados os seus elementos, o exame, os grupos de testes e os testes

propriamente ditos. Ainda neste capítulo são elencados os principais desafios que surgiram na realização deste trabalho.

- Modelação RunUI - Neste capítulo são descritos os componentes que suportam a abordagem RunUI, sendo feita a apresentação do modelo de classes e o conceito de metadados que foi utilizado na elaboração do protótipo de geração da interface. Neste capítulo é descrita a base de dados que suporta a informação dos metadados e a linguagem RunUI que permite definir o comportamento da interface de utilizador.
- Ferramenta Desenvolvida – Este capítulo contém a parte central do trabalho realizado, já que apresenta de modo pormenorizado a ferramenta desenvolvida. É apresentado o esquema da implementação da ferramenta, sendo detalhadas as suas principais funcionalidades e descrito o mecanismo de geração da interface de utilizador, em tempo de execução com recurso aos metadados. Neste capítulo é feita igualmente uma avaliação da ferramenta de geração da interface, quer ao nível do impato no desenvolvimento, quer ao nível da própria implementação da ferramenta em produção, em clientes com uma versão da aplicação WinOPT onde a ferramenta desenvolvida foi integrada. Ainda neste capítulo são descritas as principais limitações desta abordagem e o *feedback* dos utilizadores que a utilizam diariamente.
- Conclusão e Trabalho Futuro – Neste capítulo são apresentadas as conclusões deste trabalho e as perspetivas de evolução da ferramenta na aplicação WinOPT e noutras aplicações da própria empresa, sendo feita uma avaliação da viabilidade da sua reutilização. Como ponto de trabalho futuro é destacado a portabilidade dos metadados, de forma a não ficarem restritos à base de dados e são apresentadas alternativas futuras para ultrapassar esta limitação. Um outro ponto de evolução discutido neste capítulo, é a evolução do conceito de metadados e do mecanismo de geração da interface em tempo de execução, para aplicações Web à custa da tecnologia JSON, o que alargaria o âmbito de utilização da abordagem RunUI definida na realização deste trabalho.

Capítulo II – Geração automática de interfaces

O desenvolvimento de interfaces gráficas de utilizador, é ainda uma atividade demorada, embora nos últimos anos, tenham sido feitos avanços significativos relativamente a ferramentas de desenvolvimento e a sistemas de gestão da interface de utilizador.

Nos ambientes integrados de desenvolvimento comercialmente disponíveis hoje, cada objeto da interface deve ser criado e desenhado explicitamente. Em comparação com o uso de *toolkits*, estes sistemas ajudam muito na concepção e implementação de interfaces gráficas. No entanto, as especificações de controlo de diálogo têm ainda de ser adicionadas por programação, o que exige um grande esforço e requer um conhecimento especializado. Além disso, pouco ou quase nenhum suporte é disponibilizado para a seleção de objetos de interação adequados para determinada tarefa.

Esta situação conduz em geral a um esforço extra e a potenciais inconsistências, por exemplo, quando na análise de desenho de interface de utilizador, se modificam os requisitos. A geração automática de interfaces procura responder a estes desafios.

2.1. Desenvolvimento de Interfaces de Utilizador baseado em modelos (MBUID)

O desenvolvimento da interface de utilizador baseado em modelos (MBUID – do inglês *Model Based User Interface Development*), tenta resolver os desafios atuais de geração da Interface de Utilizador, nomeadamente:

- Heterogeneidade de utilizadores,
- Heterogeneidade de plataformas,
- Heterogeneidade de linguagens de programação,
- Heterogeneidade de ambientes de trabalho,

Ao longo do tempo podem ser caraterizadas 4 gerações de MBUID:

1. Geração automática e direta da interface do Utilizador, a partir de um único modelo, através de Sistemas de Gestão de Interfaces com o Utilizador (UIMS - *UI Management Systems*) que permitiam auxiliar a gestão da interface com o utilizador de forma a aumentar a eficiência do programador. No entanto o programador necessitava ter conhecimentos da especificação, o que colocava algumas limitações à utilização destas abordagens. Para além disso os UIMS não eram facilmente portáveis.
2. O modelo da UI é estruturado noutros modelos, modelo de tarefas, modelo de diálogo e modelo de apresentação, e permite aos programadores gerarem a interface de utilizador. O modelo de tarefas representa uma visão dos requisitos do sistema, o modelo de apresentação representa o aspeto visual do sistema e o modelo de diálogo o conjunto de ações que o utilizador pode executar em cada estado do sistema.
3. Especificação de UI independente da plataforma, isto é, os programadores têm que modelar a interface tendo em conta varias restrições, como por exemplo o caso do tamanho, posição dos controlos.
4. Atualmente está em curso o desenvolvimento de UI sensíveis ao contexto e independentes de plataforma e com diferentes configurações, usualmente chamadas *model-driven* em vez de *model-base*, ou seja, são semi-automatizadas, sendo transformadas num executável, ou então o código é renderizado por um interpretador.

A *Cameleon Framework* (Calvary, 2003) é o modelo de referência para a quarta geração do MBUID, definindo um processo de desenvolvimento que se adapta a múltiplos contextos de utilização das aplicações, sendo constituída pelos seguintes níveis de modelação:

1. *Tasks e Concepts* – este nível apresenta as várias tarefas a serem realizadas e os conceitos de domínio requeridos para realizar essas tarefas. Os conceitos e entidades manipuladas são representadas por instâncias de classes.
2. *Abstract UI* (AUI) – este nível descreve uma interface de utilizador apenas através da semântica da interação, sem detalhar nenhuma caraterística do dispositivo, método de interação ou tecnologia de implementação. Ele define *containers* abstratos e componentes individuais.

3. *Concrete UI* (CUI) – este nível descreve uma interface de utilizador em potencial após uma modalidade de interação particular ter sido selecionada (por exemplo, gráfica, vocal, multimodal). Esta etapa é apoiada por diversas ferramentas que ajudam os *designers* e programadores a editar, criar ou esboçar uma interface de usuário

4. *Final UI* (FUI) – Interface do final do utilizador - este nível é corresponde ao código de um interface de utilizador produzida a partir dos níveis anteriores. Este código pode ser interpretado ou compilado.

Na Figura 3 é apresentado um esquema da *Cameleon Framework*, de concretização de um modelo de domínio em diferentes modelos abstratos de interface, de acordo com o contexto até ao modelo final de interface.

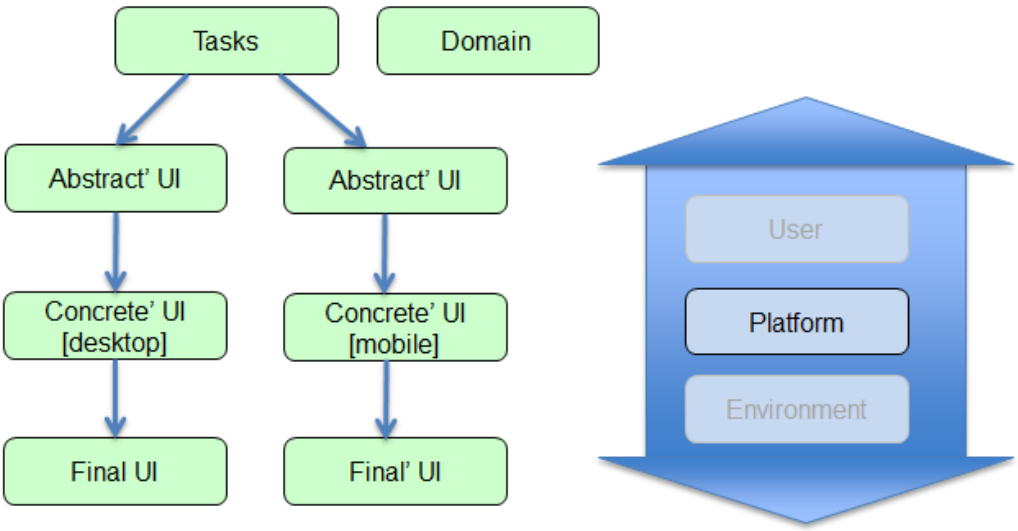

Figura 3 - Camaleon Framework.

A Cameleon Framework fornece um enquadramento teórico ao desenvolvimento de aplicações interativas baseado em modelos. Diversas abordagens têm sido propostas para concretizar esse desenvolvimento. A abordagem baseada em *ConcurTaskTrees* (Paternò, 1999), em conjunto com a linguagem Maria, permite definir tanto AUI, como CUI. Esta abordagem não só permite editar os modelos abstratos e derivá-los em modelos concretos, como também possibilita a inclusão de eventos. Esta *framework* permite especificar a interface de utilizador, em diferentes níveis de abstração, em termos de funcionalidades (análise de tarefas) e as entidades

manipuladas pela própria interface. Para além disso suporta diversas plataformas, nomeadamente: *desktop*, móvel, vocal e multimodal.

A linguagem UsiXML (Calvary et al., 2011) permite descrever uma interface a diversos níveis de abstracção. A organização destes níveis permitem definir etapas de desenvolvimento para aplicações interativas, com vários contextos, o que ajuda na exploração de soluções diferentes para a representação final da interface.

Segundo, Limbourg et. al (2005), os *designers* podem definir a interface de utilizador de uma nova aplicação sem ser necessário ter conhecimentos de programação, apenas especificando na linguagem de descrição da Interface de utilizador (UIDL - User Interface Description Language)..

Uma outra abordagem usa três linguagens base para cada uma das camadas de UI: UseML, DISL e UIML (Meixner, Breiner and Seissler, 2011). A linguagem UseML transforma as tarefas do modelo em *use objects* (UO) para definir uma hierarquia, sendo as folhas da árvore, os objetos elementares. Esta informação é usada para definir o Dialog Model. A linguagem de Especificação do Modelo de Diálogo de Interface (DISL- Dialog Interface Specification Model) é uma linguagem independente de plataforma e modal. Está focada na escalabilidade, reatividade e fácil aprendizagem dos programadores na criação de interfaces. Existem ainda vários desafios nesta abordagem, nem sempre fácil de aplicar, dos quais destacamos:

1. Risco de existência de muitos modelos e pouca normalização.
2. Necessidade de definir uma linguagem simples e padrão para a descrição da interface de utilizador.

As abordagens referidas acima, embora tenham permitido explorar os conceitos, não estão num estado de desenvolvimento que permita a sua aplicação mais generalizada. Por exemplo, existem áreas da indústria que estão já a desenvolver MBUID específico para a sua área, como é o caso da indústria automóvel, onde existem processos que são bastante flexíveis. Assim de modo similar, será explorada a aplicação destes conceitos na resolução do problema concreto deste trabalho.

2.2. Desenvolvimento de Interface de Utilizador em tempo de execução

Os sistemas apresentados anteriormente permitem descrever a interface através da geração de código estático, o que dificulta uma maior padronização da interface e reduz o seu dinamismo.

As ferramentas que descrevemos a seguir permitem a partir de um modelo de entidade-relação ou através de um modelo de domínio definir os componentes básicos da interface e consequentemente gerar o respetivo *layout* e descrever o controlo de diálogo.

2.2.1. Genius

Na abordagem Genius (Janssen, 1993) o *designer* usa o modelo de dados da aplicação existente para desenhar a interface do Utilizador. No modelo de dados o programador deve definir as *views* que são utilizadas na criação do *layout* e na definição do controlo de diálogo.

Nesta abordagem são usados os modelos de Entidade Relação (ER) como ponto de partida da especificação da interface, sendo esta caraterizada pelos seguintes pontos:

1. As *views* são definidas com base no modelo de dados.
2. Um componente base gera as interfaces estáticas a partir da definição das *views*.
3. As transições entre as *views* são definidas através de redes de diálogo.
4. As redes de diálogo são transformadas em código para o UIMS, já que o sistema de destino desta é na realidade um sistema UIMS.

Ao nível de constituição a ferramenta Genius é composta pelos seguintes componentes.

1. *View* - consiste num *subset* de entidades, relações e atributos do modelo de dados. Existem dois tipos de funções associadas às *views*: funções de manipulação de dados e funções de navegação.
2. Componente para Geração de Interface de Utilizador – Consiste num conjunto de rotinas e funções usadas para gerar a interface tomando como base os seguintes elementos:
    a. Tipos abstratos – podem conter atributos para ser preenchidos com os dados específicos dos aplicativos, como por exemplo o nome e gama de um campo

de dados, assim como atributos de *layout* como o alinhamento, cor e tamanho.

b. Os objetos de interação padrão - são por exemplo janelas, caixas de diálogo, menus, campos de entrada, constante campos, as opções exclusivas, opções não exclusivas, listas, entre outros.

c. Regras para a seleção e disposição dos objetos de interação.

O processo de geração da Interface de Utilizador, na abordagem da ferramenta Genius, é constituído pelas seguintes etapas:

1. Em primeiro lugar são escolhidos os controlos segundo os dados associados.
2. Em segundo lugar os valores dos atributos dos controlos e os seus valores por defeito são atributos.
3. E por fim, em terceiro lugar é definida a disposição dos *containers* de controlos e a definida a sequência de *focus* dos próprios controlos.

Em resumo a técnica utilizada na abordagem Genius, as redes de diálogo, pode ser utilizada para uma especificação simplista da interface, no que diz respeito à definição da visibilidade dos controlos.

A geração automática da interface do utilizador é realizada a partir das *views* por um sistema baseado em regras derivadas de diretivas existentes.

2.2.2. Mecano

A abordagem Mecano (Puerta e tal., 1994) apresenta um ambiente de desenvolvimento da interface de utilizador baseada em modelos de domínio e usando as entidades deste para impulsionar a geração da referida interface. A partir dos modelos de domínio é possível gerar as interfaces de utilizador, de forma automática, mas em *design time*. Esta abordagem é aplicável em áreas mais restritas, como por exemplo, aplicações baseadas em formulários, onde as tarefas são essencialmente relacionadas com a manutenção de dados, como adicionar, consultar, atualizar e apagar.

Os fatores mais relevantes desta ferramenta são as seguintes:

1. Faz uso de modelos de domínio para especificar a geração de interface de utilizador.
2. Os modelos de domínio podem ser reutilizados em diferentes aplicações e apresentam de forma explícita dados de domínio e relacionamentos que não estão descritas nos modelos de dados
3. Permite gerar não só o *layout* estático da interface como o comportamento dinâmico, específico do domínio, e disponibiliza interfaces base para domínios bastante amplos e complexos.
4. Disponibiliza um ambiente de *design* altamente automatizado que suporta o ciclo de desenvolvimento completo de uma interface.
5. Permite definir a interface através de um modelo textual, permitindo assim gerar muitas instâncias da mesma interface por diferentes sistemas de tempo de execução, proporcionando assim um grau de portabilidade para as interfaces geradas.

A natureza automática do processo de desenvolvimento de *Mecano* restringe o espaço de *design* das interfaces geradas, com pouca intervenção humana, sendo mais eficaz nas interfaces com estruturas de diálogo relativamente fixas, como é caso dos formulários de inserção de dados.

2.2.3. Model-Driven Asset Generation at Runtime

A Geração da Interface de Utilizador, com base em metadados da aplicação, em tempo de execução (Rockford, 2010) assenta na construção da aplicação de negócio para que esta vá ler diretamente os seus metadados e dinamicamente crie os componentes de interface da aplicação a partir daqueles metadados. O resultado desta operação é um modelo executável, já que em tempo de execução é construída diretamente a interface de utilizador com base nos metadados que na prática constituem o modelo de interface.

Um bom exemplo desta abordagem é um *browser* Web, que consome HTML e usa essas informações para fazer o *rendering* dos componentes de interface relativamente complexos em tempo de execução. Neste caso as *tags* de HTML funcionam como metadados, e a página Web exibida ao utilizador, com todos os seus controlos e gráficos, é o resultado da

interpretação dos metadados. O navegador é assim capaz de consumir uma grande variedade de metadados HTML a partir de fontes diferentes e gerar as páginas correspondentes.

A geração da interface em tempo de execução resulta na mesma produtividade, qualidade e vantagem de consistência como a criação de código em tempo de *design*. No entanto, a geração em tempo de execução oferece benefícios adicionais, como por exemplo, a flexibilidade e a adaptabilidade. Quando novos metadados são inseridos nas ferramentas de *design*, o gerador deve ser executado novamente, o aplicativo deve ser reconstruído e redistribuído, o que não é tão eficiente. Em contrapartida nesta abordagem, quando novos metadados são criados no modelo de geração em tempo de execução, o mecanismo de geração da UI simplesmente consome os novos metadados e dinamicamente redefine completamente a interface da aplicação, na maioria dos casos, sem ser necessário reconstruir ou redistribuir a aplicação. Tal como no caso das páginas Web os utilizadores não precisam de instalar um novo *browser* cada vez que acedem um novo website, também aqui a interface se atualiza sem ser necessário reinstalar a aplicação.

Este último comportamento é o pretendido na realização deste trabalho através do protótipo que se pretende construir e onde são apresentados dois modelos concretos distintos (CUI), ou seja, duas formas de representar os dados no formulário. O motor de geração da UI fará uso dos metadados e com base na lista de tipos possíveis de dados, bem como uma lista de controlos disponíveis para cada modelo, em tempo de execução, gerará os elementos de interface do utilizador para apresentar o modelo final pretendido (FUI).

Muitos produtos de software são sustentados por este género de abordagem para que não seja necessário, em função de diferentes necessidades que variam de cliente para cliente, reescrever a interface de utilizador e a sua lógica de comportamento. Uma grande quantidade de aplicações, essencialmente de comércio eletrónico, tem este padrão de Interface de utilizador dinâmico. Por exemplo, quando é necessário inserir informações sobre o produto, como os clientes podem estar a vender/comprar qualquer tipo de produto, ao invés de estar a reescrever o código para cada tipo de negócio diferente, basta simplesmente permitir aos clientes introduzir os metadados para os atributos do produto, através de um janela de gestão dos metadados, e a interface é apresentada.

No caso da ferramenta implementada, os metadados não são inseridos diretamente pelo cliente, já que os exames precisam de um suporte com conhecimento técnico para que não sejam inseridas informações incorretas ou inválidas.

2.3. Conclusões

Em resumo, as abordagens acima descritas possuem algumas caraterísticas que vão de encontro ao que é pretendido com a realização deste projeto. No entanto, nenhuma das abordagens apresentadas satisfaz na totalidade os objetivos pretendidos, pois a primeira contêm uma complexidade acrescida na criação inicial, enquanto a segunda e terceira implicam a definição de uma linguagem para criação da interface de utilizador. Após análise das abordagens anteriores, encontramos algumas lacunas, que são relevantes no âmbito deste projeto:

1. Alguma complexidade na especificação da interface de utilizador, isto é, o desenvolvimento daquela não é suportada por uma estrutura conhecida ou usada diariamente pelos programadores.
2. O processo de criação da interface de utilizador muitas das vezes não é simples, já que recorre a uma linguagem específica de definição da interface.
3. Elevada complexidade na definição do controlo de diálogo, embora não sendo crucial na geração da interface de utilizador para o sistema WinOPT foi já incluído lógica de comportamento na definição do modelo com base no qual foi construído o protótipo.

Assim, das abordagens analisadas, aquelas que mais se aproximavam dos objetivos pretendidos são as seguintes:

1. O conceito base AUI, CUI, e FUI da *Camaleon Framework* mas sem a definição de uma linguagem de modelação.
2. O mecanismo de geração de UI em tempo de execução semelhante ao da ferramenta *Genius*.

3. A leitura de metadados definidos na estrutura e posterior *rendering* da interface de utilizador de forma semelhante ao descrito na abordagem Geração da Interface Utilizador em tempo de execução (2.2.3).

Capítulo III – Metodologia RunUI (Runtime UI)

Conforme foi descrito nos capítulos anteriores deste documento, atualmente nas aplicações de software é despendido imenso tempo no desenvolvimento da interface de utilizador. Tempo este que poderia ser utilizado eficientemente noutras tarefas de análise e conceção de software.

Tomando como base as abordagens estudadas, seria útil dispor de uma ferramenta, se possível integrada na aplicação, que agilize e padronize a interface de utilizador. Assim será interessante desenvolver um protótipo que responda aos desafios apresentados anteriormente.

3.1. A abordagem RunUI (Runtime UI)

O principal objetivo deste projeto é a geração da interface de utilizador em tempo de execução de uma forma dinâmica, ou seja, em função de um modelo definindo, neste caso, a informação a apresentar na Interface de Utilizador. Na Figura 4 é apresentada a abordagem proposta. A partir de um modelo abstrato composto pelo conjunto de exames possíveis de apresentar e um modelo concreto que contém os componentes seleccionados, é gerada a interface final pelo protótipo desenvolvido na ferramenta Visual Studio 6™.

Como no projeto em curso a interface possui relativamente pouco controlo de diálogo, foram conjugados os níveis de modelação da interface da *framework Camaleon* com o modelo de geração da interface em tempo execução baseado nos metadados, para a definição daquela que será a nossa aplicação de construção da interface de utilizador. Com base nesta composição de abordagens será definido um protótipo como suporte à geração da interface do utilizador.

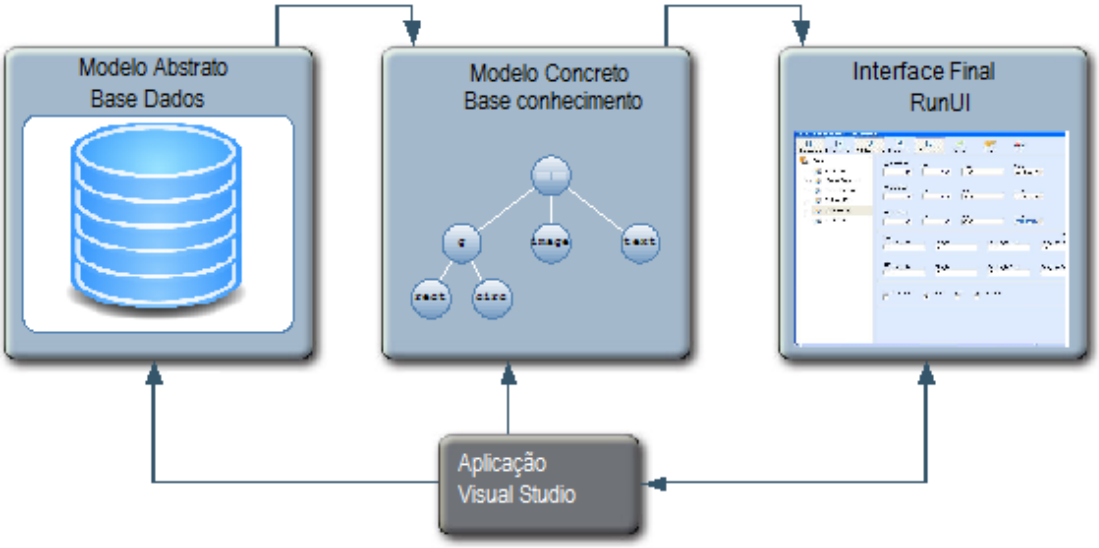

Figura 4 - Abordagem Runtime UI (RunUI).

3.2. Esquema de aplicação da metodologia

Como referido acima, o trabalho desenvolvido aproveita um pouco dos vários conceitos descritos nas abordagens estudadas, como é o caso da plataforma *Cameleon* (Calvary, 2003). Traçando um paralelo com esta plataforma temos como domínio os exames de optometria e as tarefas são a adição, manutenção e consulta de exames. O modelo abstrato da UI é definido pela lista de exames que constituem, de forma dinâmica, o teste de optometria. O modelo concreto da UI define qual o esquema e quais os componentes de interface de utilizador que se pretende que sejam utilizados na definição da interface final. Para o mesmo modelo abstrato, isto é, para o mesmo conjunto de exames, é possível escolher diferentes componentes que estão disponíveis no momento de geração, de acordo com o tipo de dados. Por exemplo, para um determinado tipo de dados enumerado, podemos escolher se queremos apresentá-lo num *combo box* ou numa lista, o mesmo sucedendo com os restantes tipos de dados.

A interface de utilizador final corresponde à janela de formulário de exames, que permite registar os exames pretendidos pelo utilizador da aplicação, no entanto a interface final de utilizador não é gerada em *design*, ou seja, não é gerado código da UI de forma automática, como no caso de plataforma *Camaleon* (Calvary, 2003). Na abordagem utilizada no trabalho

desenvolvido, o *layout* é gerido pelo algoritmo de geração de interface, ou seja, é feito o *rendering* dos componentes em tempo de execução, tal como acontece na metodologia *Model-Driven Asset Generation at Runtime* (Rockford, 2010), com base nos metadados que constituem o modelo abstrato de UI a interface é gerada em tempo de execução com base na selecção do modelo concreto final.

Neste trabalho a geração da UI não abrange aspetos da interface como por exemplo a transição entre janelas e outras funcionalidades ao nível do comportamento de controlo de utilizador. O mecanismo de geração desenvolvido assenta essencialmente sobre aspetos de definição da estrutura do formulário.

Finalmente, os dados inseridos pelo utilizador são guardados de forma persistente, numa tabela constituída por colunas genéricas que guardam a informação tal como foi preenchida pelo utilizador e que a voltam a apresentar na interface, seguindo o modelo abstrato de dados, respeitando o formato definido para cada um dos campos do exame.

Na Figura 5 é apresentado de forma esquemática o modelo que suporta o trabalho desenvolvido na implementação da metodologia escolhida.

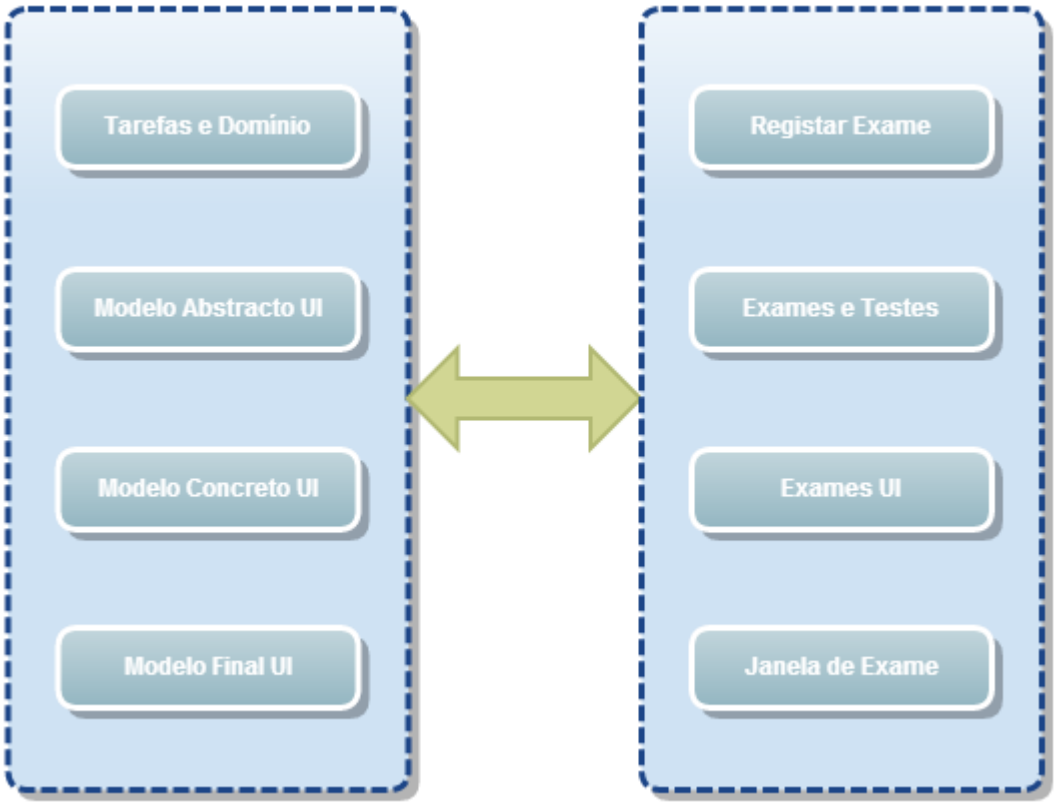

Figura 5 - Metodologia aplicada.

A ideia desta abordagem será incluir a funcionalidade embebida na aplicação de gestão de óptica WinOPT, de forma a recolher melhor o *feedback* quer dos programadores, quer dos utilizadores que diariamente trabalham com o módulo de testes de optometria.

3.3. Esboço do protótipo

É apresentado na Figura 6, um primeiro esboço da interface de utilizador gerado pelo protótipo a implementar neste trabalho, com aplicação da abordagem idealizada.

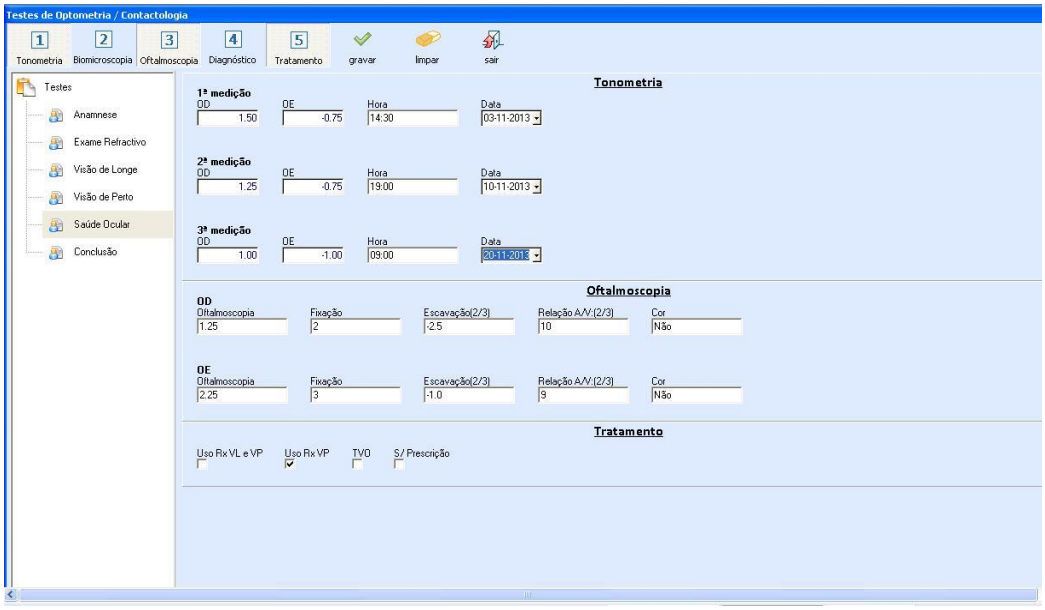

Figura 6 - Esboço do protótipo.

O esboço apresentado na imagem já foi gerado através do protótipo. No modelo são definidos os metadados, ou seja, a lista de exames possíveis que podem ser apresentados no formulário e os seus testes, a lista dos testes obrigatórios e a validação de valores inseridos. A seguir o protótipo lê esta informação dos metadados, que contêm toda a informação necessária para gerar a interface, e dispõem os componentes da interface no formulário em tempo de execução. No caso da Figura 6 são apresentados os grupos dos exames à esquerda e para o grupo seleccionado em cima são apresentados os testes possíveis que podem ser feitos na barra de tarefas em cima.

3.4. Principais desafios na criação do protótipo

Com base na realidade dos interfaces atuais são assinaladas as principais dificuldades que se pretenderam ultrapassar na definição do protótipo. Alguns destes desafios tiveram impato no processo de geração, outros implicaram incluir também informação no próprio modelo (os metadados).

A seguir são descritos os desafios e como cada um destes foi resolvido:

1. Posicionar os controlos, garantindo uma boa usabilidade da interface e clareza na leitura dos dados.
    - Solução: Aqui foram definidos os tipos de dados possíveis (booleano, texto, numero, data, moeda, lista de valores, imagem) e quais os componentes de interface que seriam mais adequados para a apresentar a informação, por exemplo, calendário para o tipo data.
2. Possibilitar a redefinição da ordem de navegação dos controlos em função do seu posicionamento na janela.
    - Solução: Um exame é composto por vários grupos de determinam uma análise específica do exame. Por sua vez cada grupo é constituido por testes singulares que medem as patologias do paciente. Aqui a solução passou por indicar no modelo qual a ordem ou posição do teste dentro do seu grupo.
3. Mostrar apenas os grupos de testes preenchidos, apresentando um resumo dos valores introduzidos.
    - Solução: Permitir a inclusão no modelo do grupo de testes ao qual o teste pertence e indicar no modelo se o teste está ou não seleccionado.
4. Permitir validar os valores inseridos, ou seja, caso os valores sejam inválidos, assinalar no próprio controlo que este contém um valor definido como não válido.
    - Solução:
        - Indicar no modelo o intervalo de valores válidos permitidos pelo teste.
5. Permitir validar os campos obrigatórios, i.e., assinalar no controlo que é obrigatório case este esteja selecionado.
    - Solução: Indicar no modelo se o teste é ou não de preenchimento obrigatório.
6. Ao atualizar a informação dos testes apresentar uma mensagem com a lista de campos obrigatórios cujos valores são inválidos.
    - Solução: Indicar no modelo qual o tipo de dados de cada teste que constitui um exame e incluir validação no mecanismo de geração da interface de utilizador.

7. Permitir a consulta dos dados relacionados associados ao controlo, caso este esteja definido como sendo de consulta. Por exemplo, apresentar a lista de médicos existentes.
    - Solução: Indicar no modelo qual a origem dos dados que podem ser inseridos em cada teste.

Capítulo IV – Modelação RunUI

Na ferramenta desenvolvida as tarefas a realizar e o domínio de interface são fixos, ou seja, não mudam ao longo do tempo. O que se pretende fazer é registar e consultar os exames de optometria de uma forma dinâmica.

Na Figura 7 é apresentado o modelo aplicado na implementação do protótipo. O conteúdo da interface que deve ser apresentado é lido em tempo de execução a partir de uma base de dados (ver seção 4.4). Isto possibilita dinamismo na criação do interface, no sentido que permite ao utilizador que selecione o grupo de exames que pretende realizar no momento, já que os exames necessários são diferentes dependendo do paciente e da altura de realização do exame.

Figura 7 - Modelo de implementação.

4.1. Modelo de Classes

No trabalho realizado a geração da interface de utilizador é construída com ajuda de um mecanismo de criação da interface em tempo de execução, que assenta na estrutura representada no diagrama de classes apresentado na Figura 8.

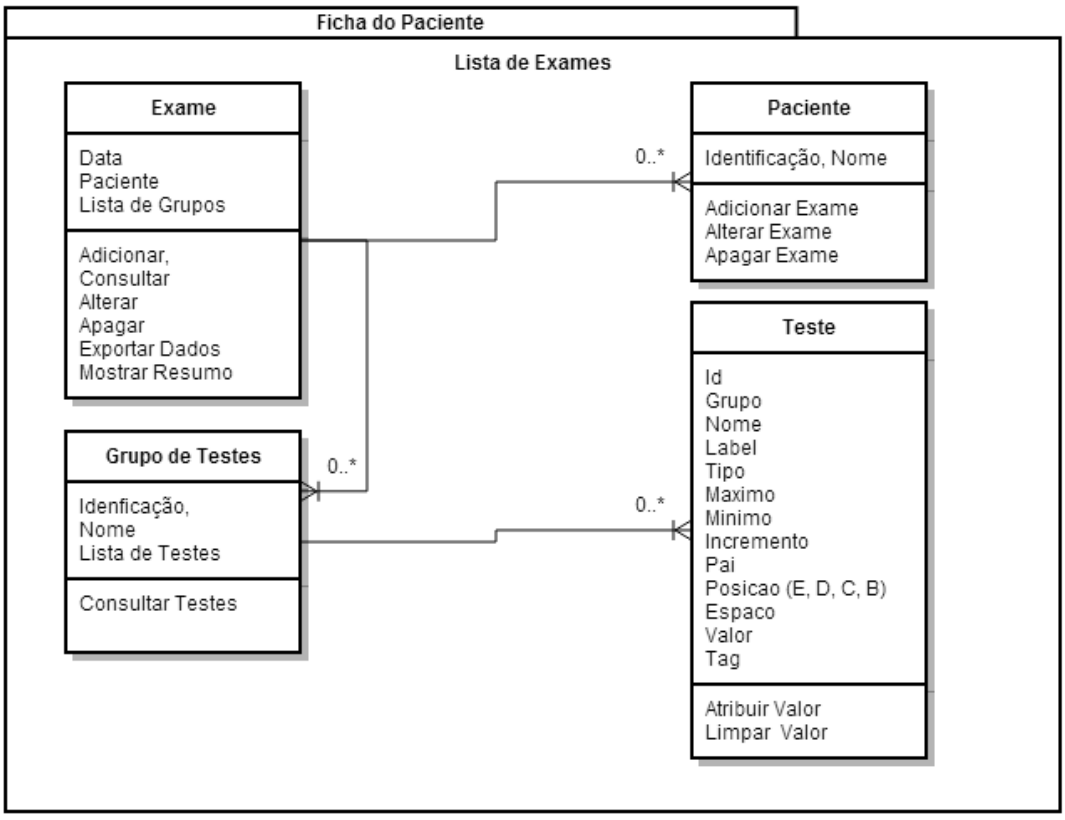

Figura 8 - Diagrama de classes do modelo de geração.

O modelo de dados que suporta a informação necessária para criação da interface de utilizador é composto pelos seguintes componentes que são lidos diretamente do modelo, que está guardado na base de dados.

Exame

O exame é constituído por grupos de testes que podem ser selecionados de forma dinâmica em função do paciente.

Grupo de Testes

Os testes que pertencem à mesma tipologia de testes oftalmológicos e estão relacionados entre si, são agregados em grupos independentes de testes.

Teste

Um teste é o componente atómico da interface que possibilita ao utilizador registar cada um dos valores da inspecção feita ao paciente. Para cada teste podemos definir o seu grupo, o seu nome e o seu tipo de dados, e por exemplo se é do tipo texto, valor, data ou booleano.

Podemos também definir a sua disposição no formulário, se está ativo/inativo, e no caso de ser um valor, podemos definir o intervalo de valores, ou seja, qual o valor máximo, o mínimo e incremento.

O modelo, além de guardar a informação dos metadados e outras informações da interface, regista também a informação e valores dos exames introduzidos pelos utilizadores.

Paciente

O paciente é a entidade sobre a qual são registados os exames.

4.2. Conceito de Metadados

Este sistema é composto por uma base de dados composta por tabelas que contêm nos seus registos os metadados e as regras de criação de interfaces concretos. Esta base de dados contém também as tabelas onde os dados inseridos na aplicação serão guardados. Esta ideia é relativamente simples e adaptável e suporta as necessidades do trabalho que consiste em gerar um interface dinâmico com a informação de testes de optometria.

Programadores e engenheiros de software usam os metadados como uma função abstrata da lógica, ou seja, os metadados geram uma estrutura lógica de auto descrição para permitir

que os dados caraterizem os recursos de aplicativos e as suas funcionalidades. Geralmente encontram-se exemplos de metadados definidos em esquemas de bases de dados, ficheiros XML[1], SOAP, Webservices, páginas web HTML, ou inclusive definidas no código fonte das aplicações. Existe uma ampla gama de utilizações dos metadados, desde uma implementação tradicional, onde alguns elementos da aplicação são representados através de metadados (cor, tamanho do texto), até uma utilização maior, onde toda a aplicação é configurada através de dados auto descritivos.

Os programadores podem assim definir a UI da forma que melhor entenderem, usando simples instruções Transact-SQL®, aplicadas sobre a base de dados, ou disponibilizando um interface para inserirem estas instruções e modificarem os atributos dos metadados, alterando de forma imediata a interface de utilizador.

A informação da interface de utilizador, ou seja, os metadados, bem como a configuração da interface concreto, são guardadas em tabelas numa base de dados relacional MS SQL Server®.

4.3. Base de dados do Modelo

O modelo contém os metadados que que definem a UI e no caso do trabalho desenvolvido estes metadados foram guardados em tabelas que estão integradas na base de dados da aplicação por uma questão de facilidade de integração, mas poderiam estar isoladas noutra base de dados, noutro motor de base de dados ou inclusive noutro formato de dados como por exemplo, um ficheiro XML, um ficheiro de texto, etc. (ver Capítulo 6). Neste caso foi aproveitada a estrutura de base de dados da aplicação para guardar a informação dos metadados da UI, ou seja, a interface é definida em função dos metadados que são lidos da base dados e do modelo concreto escolhido. Na Figura 9 são apresentadas as tabelas da base de dados MS SQL Server®, que suportam a implementação do conceito dos metadados.

[1] XML http://www.w3.org/TR/REC-xml/ Extensible Markup Language (XML) 1.0 (Fifth Edition), (acedido em 27-07-2014)

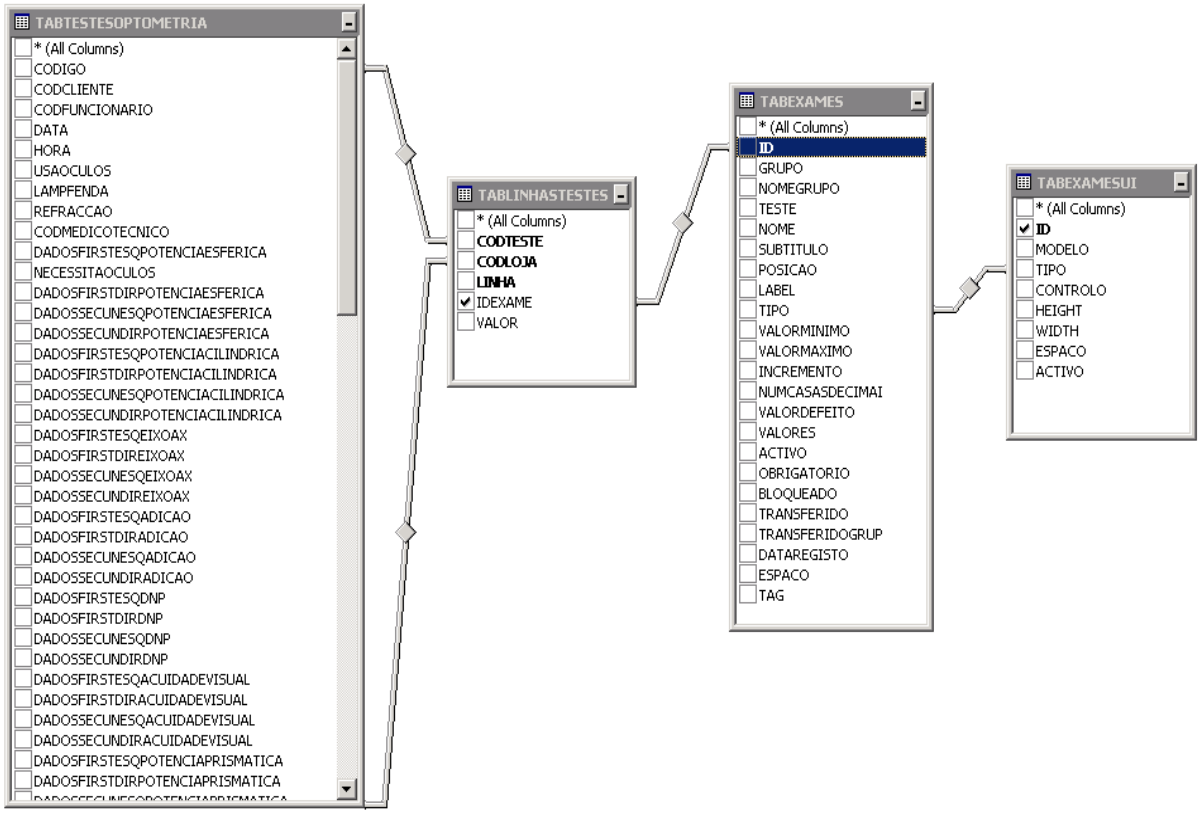

Figura 9 - Base Dados do Modelo UI.

A informação dos metadados é composta por duas tabelas que são independentes da aplicação onde estão integradas, ou seja, poderiam ser inseridas noutra estrutura que não a base de dados da aplicação.

Exames

Nesta tabela são guardados os exames que irão funcionar como metadados para a geração da interface em tempo de execução.

Os exames são constituídos por grupos de testes de forma a permitir a selecção dinâmica quer dos grupos, quer dos testes possíveis. Este modelo de agrupamento dos testes possibilita ao utilizador a escolha e visualização de uma parte da bateria total de testes a fazer ao paciente. Deste modo e como só alguns dos testes são aplicáveis a um determinado paciente, a interface mostrará apenas a selecção particular de conteúdo, podendo ser acrescentados, ou removidos, testes do exame, de uma forma dinâmica e intuitiva, através da selecção numa lista de grupos de testes.

Para cada campo pertencente a um teste que se pretende apresentar na UI foram definidos os seguintes atributos:

- ID (inteiro), Identifica o teste de forma unívoca.
- GRUPO (inteiro), Identifica o grupo ao qual o teste pertence.
- NOMEGRUPO (texto), nome do grupo.
- TESTE (inteiro), número do teste dentro do grupo.
- NOME (texto), nome do teste apresentado no cabeçalho.
- SUBTITULO (texto), subtítulo do teste, já que este pode ser constituído por vários testes.
- POSICAO (inteiro), posição do teste no grupo.
- LABEL (texto), texto a apresentar antes de cada teste individual que o identifica.
- TIPO (inteiro), Tipo de dados
    - 1 booleano
    - 2 número
    - 3 Lista
    - 4 data
    - 5 texto
    - 6 texto longo
    - 7 imagem
    - 8 hiperligação
- VALORMINIMO (decimal), valor mínimo aceite.
- VALORMAXIMO (decimal), valor máximo aceite.
- INCREMENTO (decimal), valor do incremento.
- NUMCASASDECIMAIS (inteiro), número de casas decimais.
- VALORDEFEITO (texto), valor por defeito.
- VALORES (texto), lista de valores possíveis, no caso de ser uma lista.
- ACTIVO (booleano), indicação se o teste está ativo.
- OBRIGATORIO (booleano), indicação se o campo é obrigatório ou não.
- BLOQUEADO (booleano), indicação se o teste está bloqueado inicialmente.
- DATAREGISTO (data), data de criação do teste.
- ESPACO (inteiro), indicação se deve existir um avanço a seguir a este teste

- TAG (texto), identificador único do controlo RunUI associado ao teste que contém o campo de persistência na base dados.
- ORIGEM (texto), *query* que disponibiliza os dados para consulta e cria uma lista de pesquisa que depois passa o valor selecionado para o controlo relacionado.
    - Esta *query* é usada para apresentar os dados quando o utilizador está posicionado no controlo e permite pesquisar pelo campo associado ao controlo.
- PROPRIEDADE (texto), lista de propriedades e *query* que define o estado ou valor atribuído ao teste.
    - Neste campo são definidas as diversas *queries* que permitem definir cada uma das propriedades do controlo, como por exemplo: valor, cor, se está ativo/inativo.
    - Este processo é feito com base nos valores inseridos nos outros controlos e que estão guardados numa tabela temporária da base de dados.
    - Podemos definir mais do que uma propriedade para cada componente de interface.
    - Permite um grau razoável de flexibilidade em necessidade de muito conhecimento do mecanismo de geração.
    - No entanto não devem ser definidas *queries* muito complexas, já que depois serão mais difíceis de entender ou modificar pelos programadores.

Exames UI – Modelo concreto

Nesta tabela são registados os controlos concretos a utilizar na interface, possibilitando assim que para o mesmo tipo de dados possam ser utilizados componentes diferentes ao nível da interface da janela de exames.

Esta tabela contém as seguintes colunas:

- ID (inteiro), identificação única do registo.
- MODELO (inteiro), identifica a interface concreta.
- TIPO (inteiro), tipo de dados de cada teste inserido num exame:
    - 1:: Booleano
    - 2:: Numero

    - 3:: Lista
    - 4:: Data
    - 5:: Texto Longo
    - 6:: Texto
    - 7:: Imagem
    - 8:: Hiperligação
- CONTROLO (texto), nome do componente de UI.
- HEIGHT (inteiro), altura definida para o componente de UI.
- WIDTH (inteiro), largura definida para o componente de UI.
- ESPACO (inteiro), indica se existe avanço, ou seja, se existe uma intervalo de espaço maior que o normal, entre este controlo e o seguinte.
- ACTIVO (inteiro), indica se o controlo está ativo ou não.

Dados dos Exames - Persistência

Os valores inseridos pelo utilizador são guardados numa tabela relacionada com os testes e integrada na própria aplicação. Nesta tabela são guardados os valores de cada um dos testes que compõem os exames, que foram escolhidos dinamicamente pelos utilizadores. Os valores ficam guardados linha a linha por cada teste preenchido. Se não forem preenchidos os testes, não são guardados, poupando assim recursos do sistema e espaço na base de dados.

Esta tabela não tem a ver com a Geração de UI, no entanto ela é importante já que é responsável pela persistência dos dados. A tabela é constituída pelos seguintes campos:

- CODTESTE (inteiro), identificação do teste.
- CODLOJA (inteiro), identificação da loja onde foi registado.
- IDEXAME (inteiro), guarda a identificação do exame.
- VALOR (texto), guarda o valor registado para cada teste do exame.

4.4. Linguagem Run UI

Para a geração de código dinâmico em tempo de execução foi preciso criar uma linguagem de modelação para auxiliar na própria construção do programa que vai implementar a interface.

Esta linguagem, com base nas variáveis do sistema e nos valores inseridos pelo utilizador permite definir o comportamento da interface a gerar.

No trabalho desenvolvido esta linguagem é utilizada essencialmente nos campos Origem e Propriedade da tabela Exames:

- Campo ORIGEM:
    - A *query* definida é utilizada para definir a consulta de dados associados ao controlo que se pretende mostrar ao utilizador quando este está posicionado naquele, permitindo pesquisas rápidas.
    - Aqui a linguagem usada é Transact SQL®, na sua forma corrente, para efetuar consulta de dados à base de dados como por exemplo a lista de pacientes: *<select nome from tabclientes>*
    - É possível modificar facilmente os dados que são visualizados no componente do interface, bastando para isso modificar a *query* de consulta.
    - A lista de dados devolvida pela *query* SQL disponibiliza a funcionalidade de *intellisence,* ou seja, à medida que o utilizador insere letras vai filtrando o resultado.

- Campo PROPRIEDADE:
    - É utilizada uma linguagem definida basicamente à custa da linguagem Transact SQL® e de um número reduzido de *Tags* de forma a auxiliar a definir o comportamento de cada componente da interface, caso seja necessário.
    - Este processo é feito com base nos valores inseridos nos outros controlos. Os valores inseridos pelo utilizador são guardados numa tabela temporária e com eles, e com outras variáveis da aplicação, é definido o comportamento dos controlos.
    - Permite consultar variáveis de sistema como por exemplo [@codlojaactual], [@tabelatemp], através das *tags* @.
    - A linguagem tem a seguinte sintaxe:
        - Regra = Propriedade ("|" Propriedade)*
        - Propriedade = Atributo ";" Query "$"
        - Atributo = Enabled | Value | Readonly | Backcolor | Locked | Text
        - Query = "Select" Valor "From" Tabela "Where" Condicao

- Valor = Campo da tabela temporária
- Tabela = [@ <nome da tabela temporária>]
- Condicao = [@ <nome do campo >] and | or
- [@Variável] = Variável da aplicação

- A seguir são apresentados exemplos de *queries* utilizadas na definição do comportamento da interface do WinOPT:
  - "ReadOnly; select * from [@tabelatemp] where campo1=1 and campo2=2 and codloja=[@glb_codlojaactual]"
    - Neste caso a propriedade *Readonly* do componente é verdadeira se o campo1 tiver o valor 1, o campo2 tiver o valor 2 e a loja escolhida for igual à loja atual.
  - "value; select sum (cast(isnull(valor,0) as money)) from [@tabelatemp] where idexame in (247,250)'"
    - Neste caso a propriedade *Value* do componente é igual à soma dos testes 247 e 250.
  - "Backcolor; select (case when campo1='' then 255 else 127 end) as from [@tabelatemp] where idexame=255"
    - Neste caso a propriedade *Backcolor* do componente, é branca se o campo1 do exame 255, não está preenchido e é cinzenta caso contrário.
  - "Enabled; select valor from [@tabelatemp] where idexame=274 and selecionado=1 and codloja=[@glb_codlojaactual]"
    - Neste caso a propriedade *Enabled* do componente, é igual ao valor do exame 274, ou seja, ativo se for maior que 0.

4.5. Conclusão

Neste capítulo foi descrita a forma como se pretende implementar a ideia da geração automática, em tempo de execução, da interface para a inserção/edição de exames da aplicação WinOPT. Descreveram-se as tabelas e as classes onde são guardados os modelos que

suportarão a geração da interface, bem com a linguagem desenvolvida para permitir definir o comportamento da interface.

No próximo capítulo é descrita a ferramenta desenvolvida como resultado do presente trabalho.

Capítulo V – Ferramenta Desenvolvida

Tal como descrito, a ferramenta desenvolvida consiste na elaboração de um protótipo que possibilite gerar de forma dinâmica a interface em função de metadados inseridos na estrutura de dados.

5.1. Esquema de Implementação

Na Figura 10, apresenta-se a ideia proposta para a criação do protótipo de geração da UI em tempo de execução.

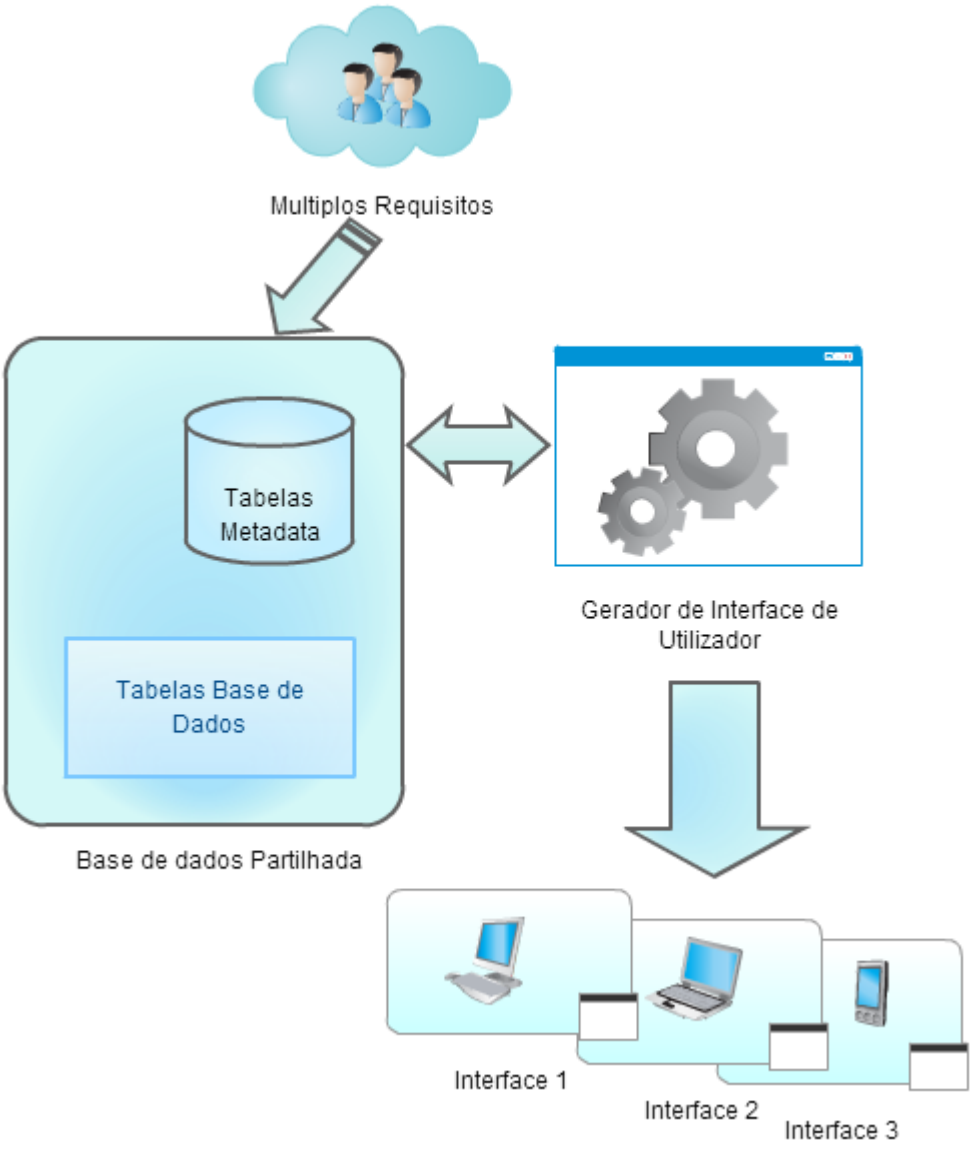

Figura 10 - Modelo de implementação.

Tal como descrito no capítulo anterior o modelo abstrato está contido numa base de dados, que pode ser partilhada, sendo definido pelo conjunto de exames que constituem os metadados. O modelo concreto, também definido na base de dados, é responsável pela definição dos componentes concretos de interface do utilizador a serem incluídos na interface em função do tipo de dados. Finalmente, a interface final é gerada em tempo de execução com base nos metadados e no modelo concreto escolhido.

O gerador da interface de utilizador faz a geração da interface do utilizador de forma dinâmica em função dos metadados inseridos. Este gerador funciona como um motor de *rendering* que, para além de desenhar a UI, permite adicionar de forma automática mensagens de erro, validar os valores inseridos ou adicionar outros elementos de forma padronizada.

No caso do trabalho desenvolvido o *rendering* dos controlos é feito num procedimento centralizado, o que permite mais facilmente criar mecanismos de processamento rápidos e centralizados com regras reutilizáveis.

5.2. Mecanismo de Geração UI (RunUI)

O mecanismo de geração da Interface de Utilizador RunUI foi desenvolvido na ferramenta Microsoft Visual Studio 6® e é responsável pela geração da interface de utilizador em tempo de execução. Este mecanismo faz uso de tabelas temporárias do MS SQL Server® para ler e guardar os valores necessários à criação da Interface de utilizador bem como para guardar os valores inseridos pelo utilizador.

O mecanismo de geração em tempo de execução é responsável por gerar e apresentar os componentes da interface, de forma dinâmica, realizando as seguintes tarefas:

a. A aplicação lê a informação de testes da base de dados, na tabela exames. Lê apenas os que estão ativos e vai-os inserido pela ordem definida na mesma tabela.

b. O mecanismo consulta qual o modelo concreto que está selecionado na tabela examesui, e em função do tipo de dados define qual o tipo do componente que deve carregar para a interface.

Em função do modelo concreto selecionado o formulário apresenta diferentes interfaces finais de utilizador, permitindo assim adaptar facilmente a interface sem alterar o programa. A Figura 11 apresenta um exemplo concreto de preenchimento da base de dados. Os valores inseridos indicam que o modelo concreto 1 foi escolhido (ver coluna MODELO). Ou seja, os controlos do modelo 1 estão ativos e serão esses a serem utilizados em função do tipo de dados do teste. Por exemplo para o tipo 3, lista de valores enumerados, será utilizado o controlo *"combo"* (*combo box normal*).

ID	MODELO	TIPO	CONTROLO	HEIGHT	WIDTH	ESPACO	ACTIVO
1	1	1	check	200	200	0	True
2	1	2	numero	300	800	0	True
3	1	3	combo	0	1600	0	True
4	1	4	data	0	1100	0	True
5	1	5	textolongo	600	3000	0	True
6	1	6	texto	300	1600	0	True

Figura 11 - Valores para ativar controlos do modelo concreto 1.

O mecanismo de geração produz interface de utilizador apresentada na Figura 12, ou seja aplica as regras do modelo 1 à interface.

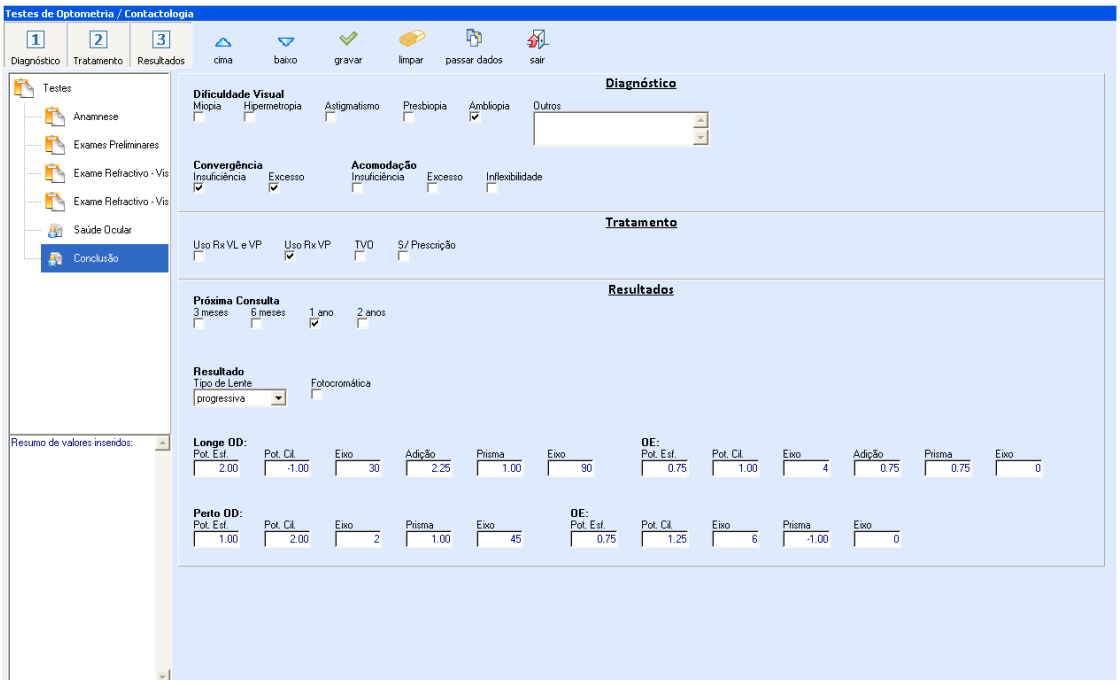

Figura 12 - Janela de exames gerada no modelo concreto 1.

Da mesma forma, inserindo os valores na base de dados conforme mostra a Figura 13, ou seja, com os controlos do modelo concreto 2 ativos, para a lista de valores enumerados (controlo de tipo 3), é utilizado o controlo com o nome *"combo2"* (*listbox usual*).

	ID	MODELO	TIPO	CONTROLO	HEIGHT	WIDTH	ESPACO	ACTIVO
	7	2	1	check2	200	200	0	False
	8	2	2	numero2	300	800	0	False
	9	2	3	combo2	0	1600	0	False
	10	2	4	data2	0	1100	0	False
	11	2	5	textolongo2	600	3000	0	False
	12	2	6	texto2	300	1600	0	False
	13	2	7	imagem	2200	4500	0	True
	14	2	8	link	300	1200	0	True

Figura 13 - Valores para ativar controlos do modelo concreto 2.

O resultado da interface de utilizador gerada é a apresentada na Figura 14, onde podemos ver que os controlos numéricos e booleanos foram substituídos por outros controlos e que por exemplo, as listas de valores enumerados são apresentadas em controlos do tipo *listbox* em vez

combobox. Para alternar entre as diversas interfaces, é necessário ativar os controlos do modelo concreto pretendido, neste caso diretamente na base de dados.

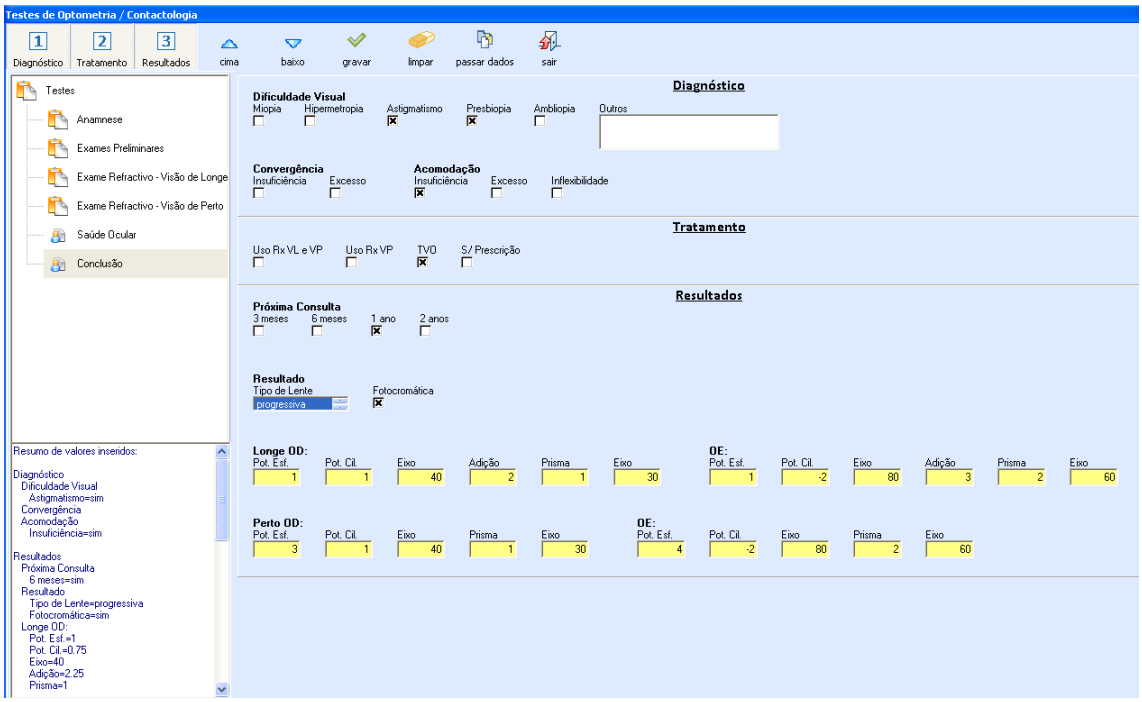

Figura 14 - Janela de exames gerada no modelo concreto 2.

5.3. Principais funcionalidades do mecanismo de geração

O mecanismo de geração da UI permite desenhar os componentes da interface de forma a gerir os diversos tipos de dados (texto, valores enumerados, número, data, imagem, *link*, etc.). Esta informação é definida no campo tipo.

Considere-se por exemplo, para um campo do tipo texto com uma lista de valores enumerados bem definidos, tal como definido nos seguintes metadados:

LABEL	Tipo de Lente
TIPO	3
VALORES	unifocal;progressiva;bif.janela;bif.seg. invisível

O resultado da interface gerada é apresentado na Figura 15.

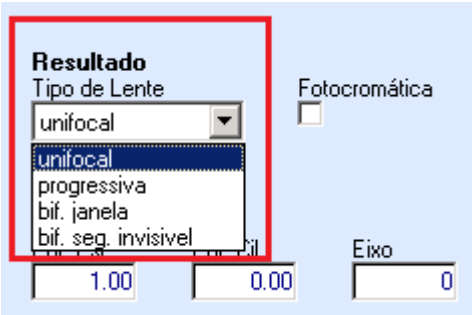

Figura 15 - Campo texto lista de valores.

Os campos do tipo data são apresentados com auxílio do controlo de calendário, permitindo a selecção fácil da data pretendida. Para os seguintes metadados:

ID	245
NOME	TESTE1
TIPO	4 (DATE)

O resultado da interface gerada é apresentado na Figura 16.

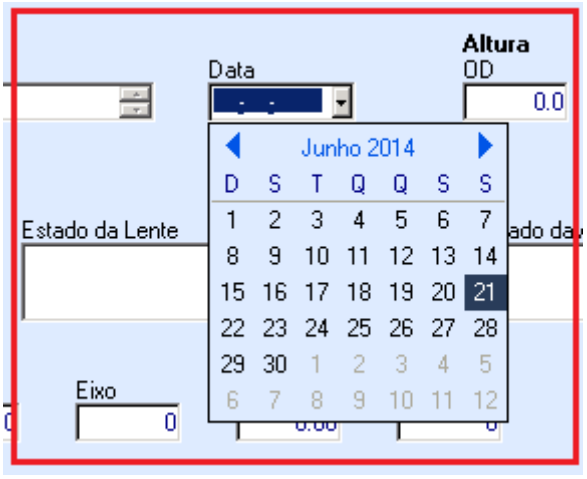

Figura 16 - Campo data com calendário.

Os valores numéricos podem conter formatações associadas ao tipo de dados, valores decimais ou do tipo moeda e no caso de serem graduações podemos definir qual o valor de incremento pré-definido. No campo Valordefeito podemos igualmente indicar qual o valor por omissão para o campo. Com base nos metadados inseridos:

NOME	Pot. Esf.
TIPO	2
VALORMINIMO	-50
VALORMAXIMO	50
INCREMENTO	0.25
NUMCASASDECIMAIS	2
VALORDEFEITO	0

O resultado da interface gerada é apresentado na Figura 17.

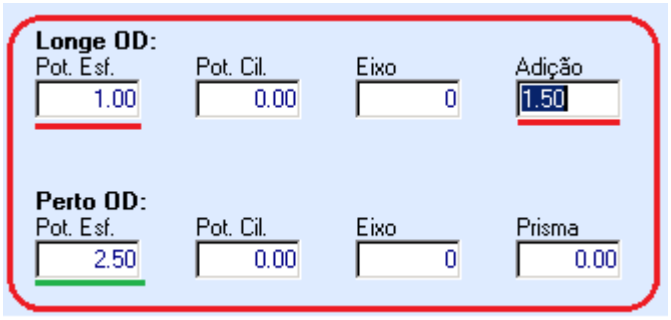

Figura 17 - Campo numérico.

Outros tipos de dados podem ter comportamentos diferentes, em função do seu tipo, como por exemplo o *hyperlink* para aceder ao endereço web, ou permitir associar uma imagem ao teste. Para os metadados:

ID	100
NOME	TESTE2
TIPO	7 (IMAGEM)

O resultado da interface gerada é apresentado na Figura 18.

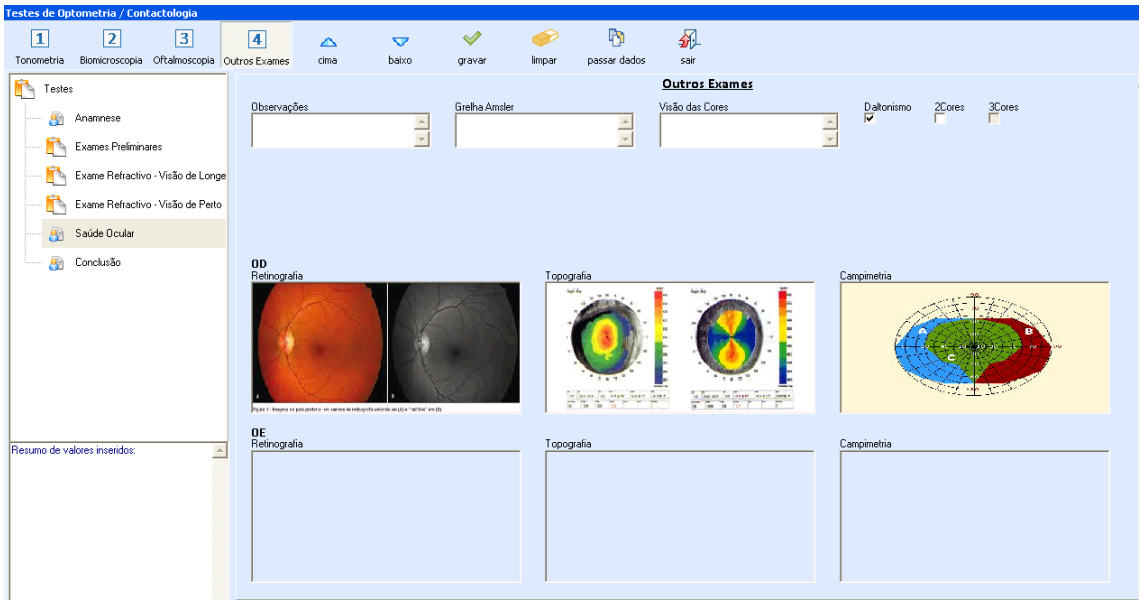

Figura 18 - Janela de exames com testes do tipo imagem/link.

O mecanismo de geração permite igualmente definir a ordem de sequência de controlos da interface. Esta sequência ordenada de transição entre os componentes é feita de forma automática em função do valor colocado no campo posição definido na tabela exames. Exemplo dos metadados para definir a posição:

ID	100
NOME	TESTE NA POSICAO 4
POSICAO	4

Na sequência do descrito na Seção 4.4, o mecanismo de geração possibilita a consulta de dados relacionada com o campo onde se está posicionado. Por exemplo, a consulta de lista de clientes no campo cliente, disponibilizando nesse caso a funcionalidade de *intellisense*, pode ser definida com base nos metadados:

ORIGEM	select * from (select nome as value from tabclientes) tab

O resultado da interface gerada é apresentado na Figura 19.

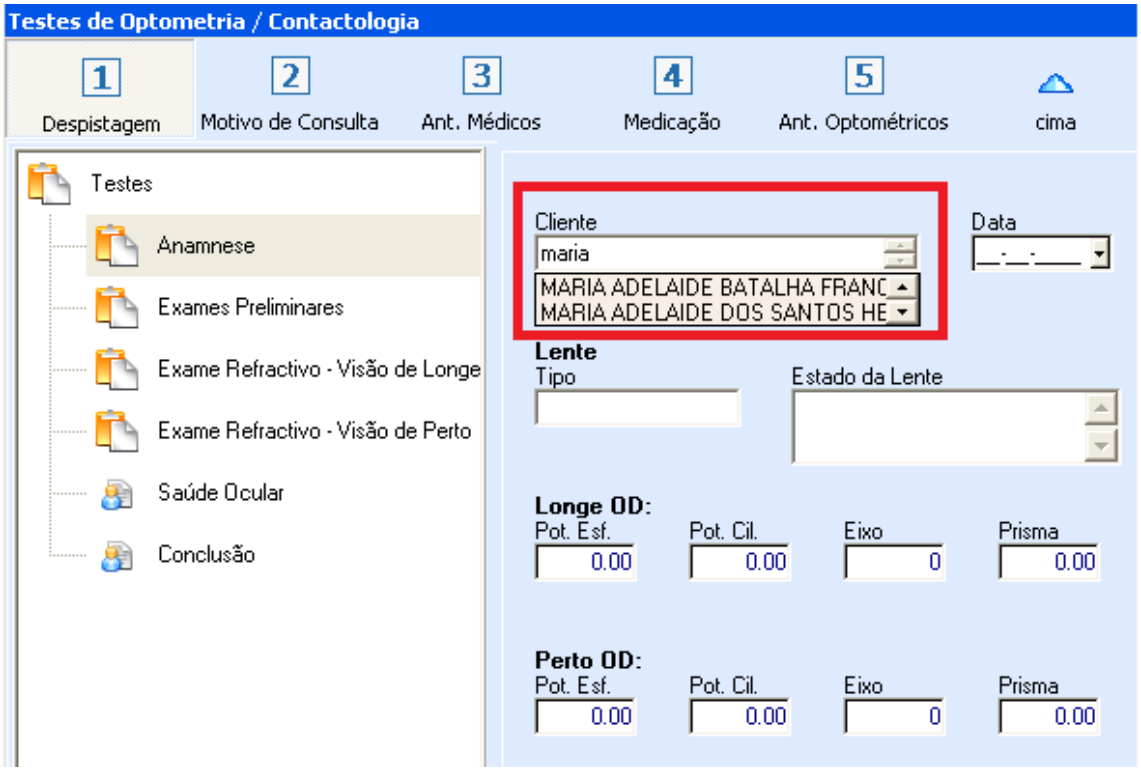

Figura 19 - Consulta de uma tabela relacionada com pesquisa.

Ao longo da análise do funcionamento da interface verificou-se que existem componentes cuja propriedade ou valor depende dos valores inseridos noutros controlos e/ou variáveis externas da própria aplicação. Por exemplo, propriedades como Ativo, Editável, Cor de Fundo, Valor (cálculo de valores segundo fórmulas), etc. Por isso, foi incluído o campo Propriedade, que permite que o valor de determinada propriedade de um campo do teste seja obtida em função dos valores inseridos num ou mais campos do mesmo exame ou de outros e/ou de variáveis globais à aplicação. Por exemplo para o campo potência esférica de perto, e com os metadados:

PROPRIEDADE	value;select sum(cast(isnull(valor,0) as money)) from [@tabelatemp] where idexame in (247,250)

O resultado da interface gerada é apresentado na Figura 20. Note-se que a potência esférica de perto é igual à soma da potência esférica e da adição de longe.

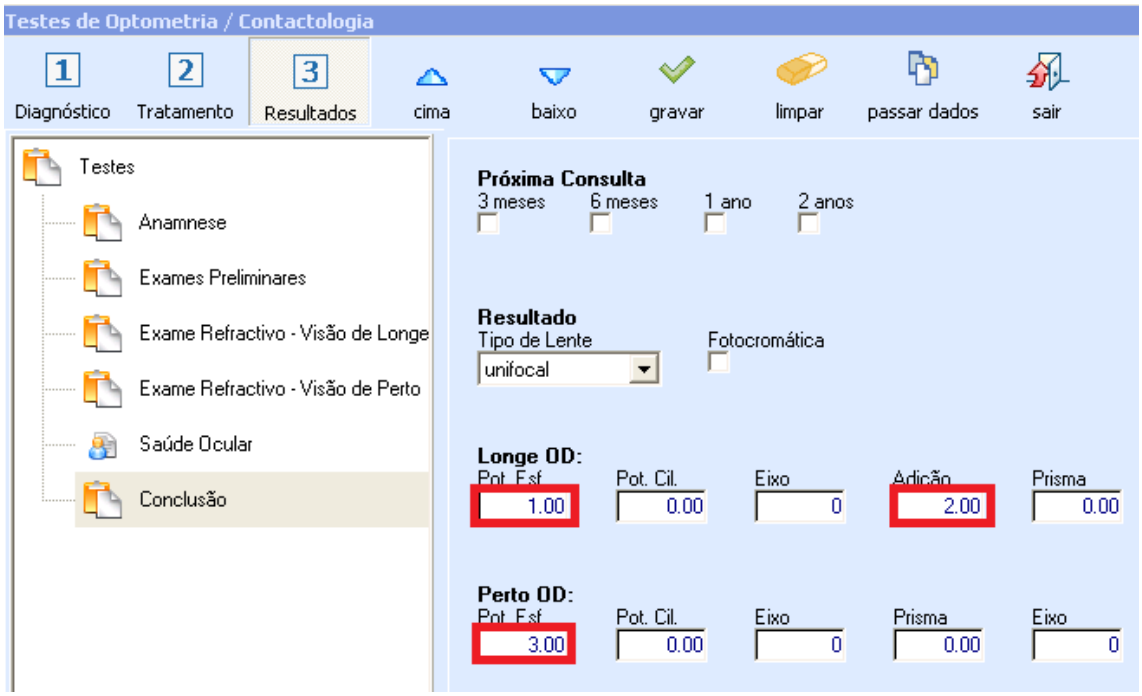

Figura 20 - Propriedade Dependente dos valores inseridos pelo utilizador.

Esta funcionalidade de dependência entre componentes é importante já que permitiu definir o comportamento do controlo em função dos valores inseridos pelo utilizador no momento em que está a usar a interface. Um exemplo é o caso do campo booleano "2 cores", cujo controlo se pretende ativar caso o utilizador seleccione a opção daltonismo. Assim com para os metadados:

SUBTITULO	Daltonismo
POSICAO	5
LABEL	2Cores
TIPO	1
PROPRIEDADE	Enabled;select valor from [@tabelatemp] where idexame=274 and selecionado=1 and codloja=[@codlojaactual]

O resultado da interface gerada é apresentado na Figura 21.

Figura 21 - Campo booleano dependente.

Para além disso, é possível definir no modelo validações dos valores inseridos. Ou seja, com base nos metadados, o protótipo percorre cada um dos testes e valida se os valores estão entre os limites definidos para cada teste. No caso de alguns não estarem dentro dos limites é apresentada uma mensagem com a lista de testes inválidos.

Para isso basta definir, por exemplo, os seguintes metadados:

VALORMINIMO	-50
VALORMAXIMO	50

O mesmo sucede para a validação do preenchimento de campos obrigatórios, isto é, o protótipo percorre de forma automática a lista de testes e se algum dos testes estiver definido como obrigatório, verifica e avisa o utilizador quais os campos que necessita preencher. Para isso basta definir por exemplo os seguintes metadados:

OBRIGATORIO	1 (0 FACULTATIVO)

O aviso apresentado pela interface gerada é apresentado na Figura 22.

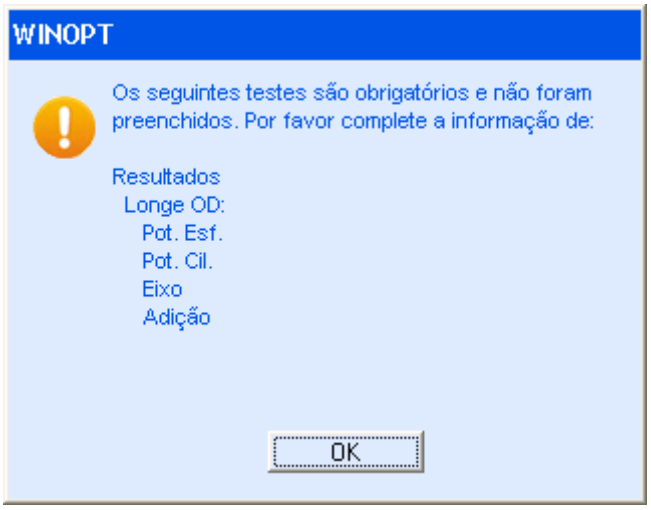

Figura 22 - Exemplo de validação de inserção de campos obrigatórios.

O protótipo possibilita ainda guardar a informação inserida pelo utilizador de forma persistente na base de dados, independentemente do número de testes seleccionados. O protótipo adapta-se mesmo que sejam acrescentados novos testes em futuras evoluções do módulo de exames.

O protótipo permite a integração dos dados inseridos com outras janelas da aplicação com base em *tags* definidas para cada teste. Ou seja, em função da *tag* única atribuída nos metadados posso consultar o valor inserido em determinado teste, permitindo a exportação dos dados inseridos de uma forma genérica.

Finalmente o protótipo permite apresentar um resumo dos valores inseridos no exame, isto é, para os testes selecionados no exame é mostrada uma listagem com os valores inseridos para cada um dos valores do teste. Estes valores são agrupados por grupo de testes.

O resumo de valores gerado pela interface é apresentado na Figura 23.

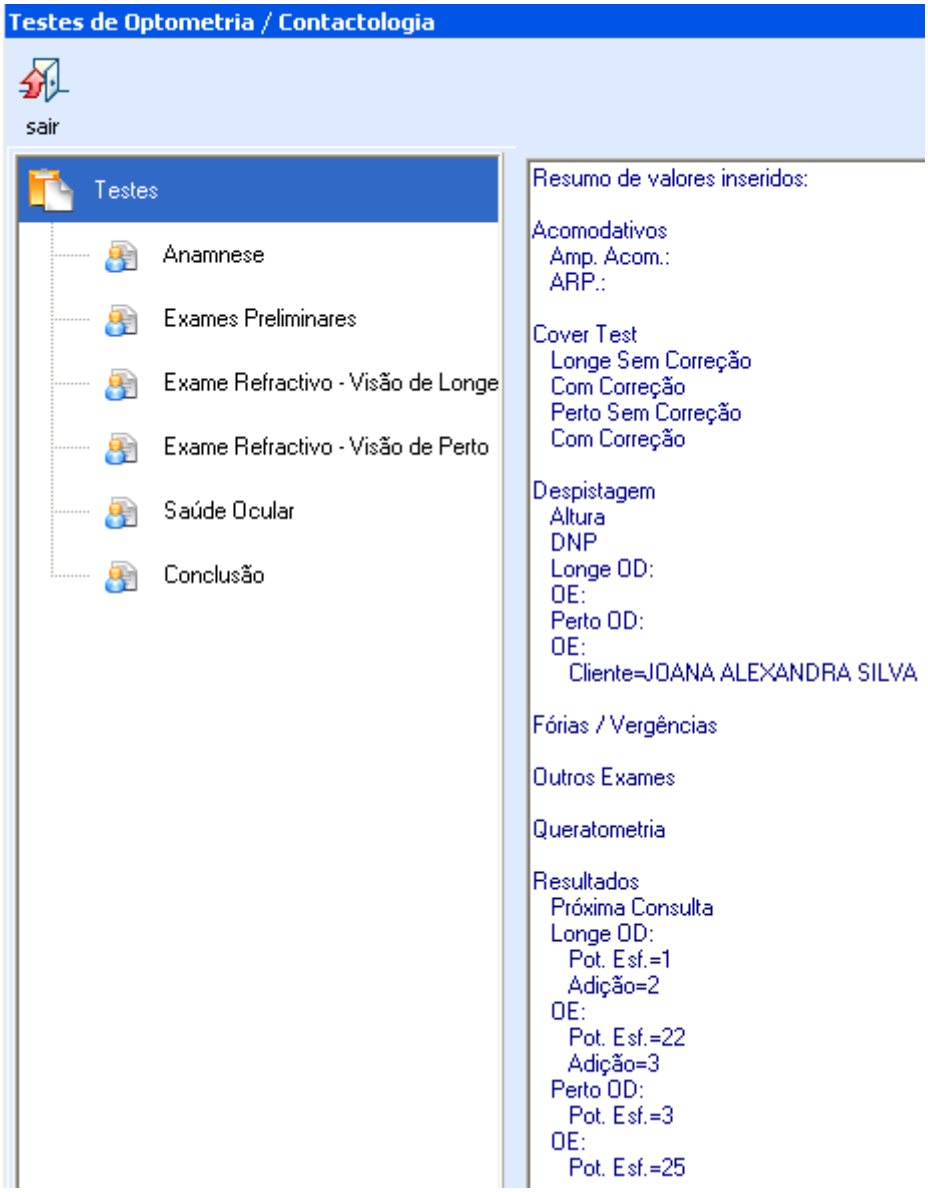

Figura 23 - Resumo de valores inseridos.

5.4. Avaliação da abordagem no desenvolvimento de UI

5.4.1. Impato no desenvolvimento

O protótipo desenvolvido neste trabalho foi inserido numa aplicação já em produção para avaliar a sua aplicabilidade, vantagens e desafios para futura utilização noutras aplicações e tipologias de desenvolvimento. Os programadores envolvidos no desenvolvimento da aplicação base, concordaram que o protótipo como mecanismo de geração da interface de utilizador terá uma boa viabilidade, bem como um bom potencial de evolução, tendo bastante aceitação.

Foi feita uma apresentação aos colegas da empresa, da área das ópticas, onde a aplicação *core* é o WinOPT, e do ponto de vista da sua reutilização, os elementos da área de gestão e de apoio ao cliente, acharam este trabalho bastante positivo, sugerindo inclusive que esta abordagem poderia ser estendida a outras aplicações, na plataforma atual, aplicações *desktop,* e também noutras plataformas, como é o caso das aplicações Web.

Resumidamente, foram apontados como pontos fortes:

- O software de geração de UI pode ser produzido apenas uma única vez, com evoluções contínuas de melhoria.
- O mecanismo de geração da interface poder ser facilmente reutilizado/adaptado noutras aplicações.
- Possibilitar uma maior facilidade e rapidez de inclusão de novas funcionalidades nas aplicações existentes.
- Permitir uma economia de recursos, libertando-os para outras tarefas mais produtivas.
- Resultar em menos tempo para codificação e desenho do Interface e em mais tempo para análise de requisitos, análise de soluções e testes às aplicações.
- Permitir uma diminuição de erros da interface, já que é gasto muito menos tempo na resolução de problemas relacionados e possibilita uma maior normalização das aplicações.
- Resumindo, garantir uma maior qualidade dos produtos de software e aumento de produtividade.

5.4.2. Implementação do protótipo em clientes

O protótipo foi disponibilizado em conjunto com a aplicação base aos clientes para recolha de *feedback* do protótipo. Os clientes adaptaram-se facilmente ao *layout* criado dinamicamente e usaram o novo módulo de exames, sem recorrer a formação, experimentando as novas funcionalidades sem impato na sua rotina de utilização do programa. O fato do interface ser gerado de forma automática, em tempo de execução, não teve impato, nem na usabilidade, nem no desempenho da inserção dos exames.

Assim a inclusão do protótipo não afetou a funcionalidade já usada pelos utilizadores finais e mais tarde foram solicitados novos requisitos pelos utilizadores. O custo de satisfazer, quer ao nível de tempo, quer a nível de complexidade, foi mais reduzido que se fosse feito o desenvolvimento usual, já que, mediante uns quantos *inputs* nos metadados, na base de dados, se obtiveram as alterações pretendidas no interface. Assim os utilizadores finais ficaram satisfeitos porque mais rapidamente foram disponibilizados os novos requisitos e com custos mais reduzidos. Inclusive foi solicitada a possibilidade de permitir a impressão dos dados, funcionalidade que não estava definida na execução deste trabalho.

5.4.3. Limitações da abordagem

Relativamente a desafios a enfrentar para a conclusão deste trabalho, podem ser enumerados os seguintes desafios:

1. Complexidade da construção de janelas partilháveis por várias opções da aplicação, já que nem todas seguem o mesmo padrão.
2. Dificuldade de aumentar o tempo de maturidade de uma janela, ou seja, estender a sua utilização ao resto da aplicação, já que esta pode conter inúmeras configurações/extensões e sofrer várias adaptações ao longo da sua utilização.
3. Maior dificuldade na organização dos controlos e gestão da aparência da interface gerada dinamicamente.

Capítulo VI – Conclusão e Trabalho Futuro

A interface do utilizador é essencial para todas as aplicações de software, quer seja uma aplicação web quer seja uma aplicação *desktop*, e desenvolvê-la consome bastante tempo e energia. O custo elevado no desenvolvimento de interfaces de utilizador, as dificuldades na implementação de novas funcionalidades, nas aplicações, ao nível da interface do utilizador, a falta de padronização das aplicações entre outros, justificam que neste projeto se estude uma maior automação na geração da interface do utilizador.

Neste trabalho procurou-se mostrar que uma das soluções para este problema é a utilização de metadados para gerar a interface do utilizador em tempo de execução para domínios relativamente restritos. Para a geração da interface de utilizador ser mais facilmente automatizada, ao desenvolver este trabalho foram aplicados dois princípios: limitar o âmbito do domínio do problema e utilizar o conhecimento de domínio para a definição dos metadados.

Este trabalho explorou soluções atuais de geração automática da UI e foram analisadas duas abordagens principais: ferramentas de geração baseadas em modelos, e ferramentas baseadas em metadados e com geração dinâmica de UI em tempo de execução.

Por fim, o trabalho apresentou um protótipo que incorpora estas ideias e as disponibiliza para evoluções futuras. De momento estas irão centrar-se em incorporar o protótipo num número diversificado de aplicações para testar a sua eficácia na prática. Em particular, o seu sucesso, adaptabilidade e implicações de desempenho no desenvolvimento de software, uma vez que se aplica a geração automática da UI, e a eficácia de colocar limites úteis na geração de interface do utilizador, terá de ser avaliada.

No entanto, o que torna esta abordagem apelativa é que em projetos grandes este método permite economizar imenso tempo já que as alterações que tem de ocorrer em cada janela, quando por exemplo existe um erro na aplicação, apenas têm que ser feitas no mecanismo de geração (motor de *rendering*). Por outro lado, mesmo com muitos elementos ou atributos o desempenho da aplicação é pouco afetado, já que os elementos são carregados inicialmente para memória.

O objetivo futuro pretendido é que diversas aplicações, com objetivos diferentes, podem ser desenvolvidas utilizando os mesmos mecanismos de processamento da interface e dentro da

mesma aplicação. O impato a mais longo prazo a que se destina esta análise é aumentar a flexibilidade e utilidade de geração automática da UI de tal forma que ela se torne uma parte integrante no desenvolvimento de software já em produção.

A capacidade de modificar os metadados através de uma interface de utilizador poderá ser um avanço significativo desta ferramenta, sendo um dos objetivos para a versão final. Ou seja, pretende-se construir um motor de *rendering* completamente dinâmico, de forma a permitir fazer a manutenção dos próprios metadados e suas operações básicas. Uma vez atingido este patamar, a eficiência no desenvolvimento de novos produtos, bem como modificações personalizadas por cliente sairão muito beneficiadas com esta solução.

6.1. Portabilidade dos Metadados

Uma das propostas para o futuro será a portabilidade do modelo atual para fora da aplicação atual, para outras plataformas/e ou ambientes. Conforme referido acima, o modelo está guardado numa base de dados relacional, mas tal não é de todo obrigatório, já que pode estar guardado noutro formato que permita uma maior portabilidade da definição da interface de utilizador. Por exemplo, um ficheiro XML[2] conforme apresentado abaixo.

Com uma linguagem, baseada em XML, pretende-se a definição de uma linguagem para descrever uma interface, tanto para os níveis de abstracção AUI como para a CUI, de modo a possibilitar a geração de forma dinâmica da interface de utilizador. A seguir é apresentada uma proposta da estrutura XML para a definição da interface:

```
<EXAMES>
<EXAME>
<ID>275</ID>
<GRUPO>5</GRUPO>
<NOMEGRUPO>Saude Ocular</NOMEGRUPO>
<TESTE>21</TESTE>
<NOME>Outros Exames</NOME>
<SUBTITULO>OD</SUBTITULO>
```

[2] XML http://www.w3.org/TR/REC-xml/ Extensible Markup Language (XML) 1.0 (Fifth Edition), (acedido em 27-07-2014)

<POSICAO>6</POSICAO>
<LABEL>3Cores</LABEL>
<TIPO>1</TIPO>
<VALORMINIMO>2</VALORMINIMO>
<VALORMAXIMO>4</VALORMAXIMO>
<INCREMENTO>0.5</INCREMENTO>
<NUMCASASDECIMAIS>1</NUMCASASDECIMAIS>
<VALORDEFEITO>2</VALORDEFEITO>
<VALORES>20;2.5;3.0;3.5;4.0</VALORES>
<ACTIVO>1</ACTIVO>
<OBRIGATORIO>0</OBRIGATORIO>
<BLOQUEADO>0</BLOQUEADO>
<DATAREGISTO>2014-05-22 00:08:42.360</DATAREGISTO>
<ESPACO>0</ESPACO>
<TAG>PE_BAS</TAG>
<ORIGEM>
select * from (select nome as value from tabclientes) tab
</ORIGEM>
<PROPRIEDADE>value;select sum(cast(isnull(valor,0) as money)) from [@tabelatemp] where idexame in (247,250)
| Enabled;select valor from [@tabelatemp] where idexame=274 and selecionado=1 and codloja=[@codlojaactual]
</PROPRIEDADE>
</EXAME>
</EXAMES>

Figura 24 - Exemplo de XML com dados UI.

O sistema a implementar teria neste caso de ir ler a informação ao ficheiro XML para poder construir a interface de utilizador, em tempo de execução, interpretá-la e aplicar as mesmas regras que existem atualmente no protótipo.

A informação do modelo concreto de dados também estaria representada no formato XML conforme apresentado abaixo.

<EXAMESUI>
<EXAMEUI>
<ID>8</ID>
<MODELO>2</MODELO>
<TIPO>2</TIPO>
<CONTROLO>numero2</CONTROLO>
<HEIGHT>300</HEIGHT>
<WIDTH>800</WIDTH>
<ESPACO>1</ESPACO>
<ACTIVO>1</ACTIVO>
</EXAMEUI>
</EXAMESUI>

Ter a informação necessária para a construção da interface num conjunto de ficheiros portáveis seria mais prático, se pretendêssemos aplicar o mesmo modelo noutras aplicações. No entanto, neste caso o objetivo como era aplicar as alterações de interface de utilizador sempre sobre a mesma aplicação e que os dados dos exames fossem de certa forma persistentes, não foi utilizada uma estrutura externa à aplicação. As alterações efetuadas durante o desenvolvimento e alterações realizadas *à posteriori* são feitas facilmente bastando para isso ajustar a informação do modelo de interface que está incorporado na base de dados.

6.2. Geração dinâmica da UI para aplicações Web

Ainda como trabalho futuro, esta metodologia poderia ser aplicada de forma semelhante numa aplicação Web, por exemplo com recurso à tecnologia JSON[3] Schema (JSON).

Um JSON Schema define um formato baseado em JSON para construir a estrutura de dados JSON para validação, documentação e controlo da interacção da interface de utilizador. O JSON Schema disponibiliza um contrato para os dados JSON necessário para a aplicação Web, e como os dados da aplicação podem ser modificados. O Schema de dados JSON é auto descritivo e pode ser usado tanto para o próprio esquema como para os dados.

[3] JSON http://json.org/json-pt.html JSON - Introdução, (acedido em 27-07-2014)

A abordagem JSON Schema orientada a dados, para gerar UI e definir a aparência dos controlos tem, indiscutivelmente, vantagens sobre outras tecnologias, já que não permite truques de codificação extravagantes ou expressões idiomáticas. A metodologia de controlo JSON, requer a criação de uma interface de utilizador de forma declarativa, com dados estáticos, o que tende a produzir código UI mais fácil de ler e de manter. O controlo JSON reflete diretamente a estrutura do DOM que irá criar e assim é fácil de visualizar a estrutura do DOM. Deste modo é mais fácil a visualização da interface do utilizador resultante e por outro lado como o JSON é mais simples de codificar, permite o desenvolvimento de ferramentas que podem facilmente gerar ou modificar o código UI em tempo de execução.

A seguir é apresentado um pequeno exemplo do que poderia ser um ficheiro JSON com informação necessária para gerar de uma forma dinâmica a interface de utilizador. Esta poderia ser gerada com recurso a um mecanismo de geração UI implementado do lado do cliente que iria interpretar o código JSON recebido e faria o *rendering* da interface no momento de apresentação ao utilizador Web.

```
{
   "$schema": "http://json-schema.org/draft-03/schema#",
   "name": "Teste",
   "type": "object",
   "properties": {
      "id": {
         "type": "number",
         "description": "Id Teste",
         "required": true
      },
      "nome": {
         "type": "string",
         "description": "Nome do Teste",
         "required": true
      },
      "posicao": {
         "type": "number",
         "minimum": 0,
```

```
            "maximum": 100,
            "required": true
        },
        "tags": {
            "type": "array",
            "items": {
                "type": "string"
            }
        },
        "graduacao": {
            "type": "object",
            "properties": {
                "esfera": {
                    "type": "number"
                },
                "cilindro": {
                    "type": "number"
                }
            }
        }
    }
}
```

Figura 25 - Exemplo de Código JSON UI.

Referências:

Kennard, R. and Steele, R. (2008). Application of software mining to automatic user interface generation. In Fujita, H. and Zualkernan, I. A., editors, 7th International Conference on Software Methodologies, Tools and Techniques, volume 182 of Frontiers in Artificial Intelligence and Applications, pages 244–254. IOS Press.

Janssen, C., Weisbecker, A., and Ziegler, J. (1993). Generating user interfaces from data models and dialogue net specifications. In Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems, CHI '93, pages 418–423, New York, NY, USA. ACM.

Meixner, G., Paternò, F., and Vanderdonckt, J. (2011). Past, present, and future of model-based user interface development. i-com, 10(3):2–11.

Myers, B. A. and Rosson, M. B. (1992). Survey on user interface programming. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '92, pages 195–202, New York, NY, USA. ACM.

Kruger, C. (2006). Software Mass Customization, Technical Report, BigLever Software, March.

Nichols, J. and Faulring, A. (2005). Automatic interface generation and future user interface tools. In Tools ACM CHI 2005 Workshop on The Future of User Interface Design Tools.

Schlungbaum, E. and Elwert, T. (1996). Automatic user interface generation from declarative models. In Proceedings of the Second International Workshop on Computer-Aided Design of User Interfaces (CADUI 96), pages 3–18. Presses Universitaires de Namur.

Paternò, F. (1999) Model-based Design and Evaluation of Interactive Applications, Spring Verlag.

Lu, X. and Wan, J. (2007). User interface design model. In Feng, W. and Gao, F., editors, Proceedings of the 8th ACIS International Conference on Software Engineering, Artificial Intel- ligence, Networking and Parallel/Distributed Computing, SNPD 2007, volume 3, pages 538–543. IEEE Computer Society.

Hayes, P. J., Szekely, P. A., and Lerner, R. A. (1985). Design alternatives for user interface management systems based on experience with COUSIN. In Proceedings of the SIGCHI

Conference on Human Factors in Computing Systems, CHI '85, pages 169–175, New York, NY, USA. ACM.

Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Provot, I., Sacré, B., e Vanderdonckt, J. (1995). Towards a systematic building of software architecture: the trident methodological guide. In Palanque, P. and Bastide, R., editors, Design, Specification and Verification of Interactive Systems (DSV-IS '95), Eurographics, pages 262–278. Springer Vienna.

Pawson, R. (2002). Naked Objects, IEEE Software, 19(4):81–83.

Vanderdonckt, J., Limbourg, Q., Michotte, B., Bouillon, L., Trevisan, D., and Florins, M. (2004). USIXML: a User Interface Description Language for Specifying Multimodal User Interfaces. In W3C Workshop on Multimodal Interaction, pages 19–20.

Lhotka, R. (2010). Design Time Code Generation and Runtime Model-Driven Generation. MSDN Data Developer Center. Microsoft. (http://msdn.microsoft.com/en-us/data/ff621668.aspx)

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., e Vanderdonckt, J. (2003). A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers, 15(3):289–308.

Calvary, G., de Wasseige, O., Faure, D., and Vanderdonckt, J. (2011). User interface extensible markup language SIG. In Campos, P., Graham, N., Jorge, J., Nunes, N., Palanque, P., and Winckler, M., editors, Human-Computer Interaction – INTERACT 2011, volume 6949 of Lecture Notes in Computer Science, pages 693–695. Springer Berlin Heidelberg.

Puerta, A. R., Eriksson, H., Gennari, J. H., and Musen, M. A. (1994). Beyond data models for automated user interface generation. In Cockton, G., Draper, S., and Weir, G., editors, People and Computers IX. Proceedings of British HCI '94, pages 353–366. Cambridge University Press.