MODUS: uma metodologia de prototipagem de interfaces baseada em modelos

Marina Machado, Rui Couto, and José Creissac Campos

HASLab/INESC TEC & Departamento de Informática/Universidade do Minho
Campus de Gualtar, 4710-057 Portugal
pg25336@alunos.uminho.pt, jose.campos@di.uminho.pt, ruicouto@di.uminho.pt

Resumo A interface de utilizador é essencial para o sucesso de uma aplicação, ora a sua implementação requer tempo e esforço. Metodologias baseadas em modelos suportadas por uma geração automática têm sido propostas como solução para reduzir os custos de desenvolvimento. No entanto, um elevado nível de automação nestas abordagens requer o uso de modelos detalhados da interface, originando complicações. Por um lado, esta perspetiva é contrária a uma conceção baseada no progressivo refinamento de *mockups*, típica do desenvolvimento de interfaces. Por outro lado, obriga uma distinção entre os modelos utilizados na lógica de negócio e na interface. Este artigo propõe uma abordagem para gerar a interface de utilizador baseada diretamente em modelos estruturais da lógica de negócio. A identificação do domínio de aplicação é um factor chave da metodologia, permitindo automatizar o processo de geração. Por sua vez a separação entre conteúdo e forma favorece o refinamento iterativo das interfaces geradas.

Palavras Chave: Geração Automática de Interfaces de Utilizador, *Model-driven engineering*, *Model-Based User Interface Development*

1 Introdução

Na sociedade moderna um vasto leque de pessoas tem acesso à tecnologia, estando acostumado a usá-la no seu dia a dia. Sendo a interface de um sistema o único meio de comunicação com o utilizador, esta torna-se num fator decisivo para o sucesso das aplicações [1].

Estudos realizados na década de 1980 e 1990 mostraram que quase 50% do esforço de desenvolvimento de uma aplicação é dedicado à *Interface de Utilizador* (IU). A crescente complexidade das interfaces requer soluções de desenvolvimento inovadoras, dificultando o seu processo de criação. Tudo leva a crer [2] que o tempo gasto na sua implementação continua significativo, sendo a sua redução uma forma de otimizar o desenvolvimento das aplicações [3,4,5].

Diversos tipos de ferramentas têm sido criados para facilitar e acelerar a geração de interfaces. Na maioria dos casos necessitam da enunciação explícita de cada elemento gráfico, como por exemplo as ferramentas de especificação gráfica e as baseadas em linguagens. Um processo de criação dependente exclusivamente

destas perspetivas permanece lento e penoso [1,6,7,8]. Todavia, as ferramentas baseadas em modelos destacam-se por fornecer um certo grau de automação ao processo de criação, reutilizando esforço prévio de desenvolvimento. A utilização destas ferramentas tem sido explorada como potencial solução do problema, dando origem ao *Model-Based User Interface Development* (MBUID).

Questiona-se então o porquê das ferramentas baseadas em modelos não se terem tornado a norma no desenvolvimento automatizado de IU. Nesta categoria o nível de automação tende a ser reduzido. Um aumento no grau de automação é geralmente traduzido pela inclusão de informação adicional por parte do utilizador da ferramenta, complexificando a elaboração dos modelos [2]. As ferramentas que focam a automação falham em diversos aspetos na produção das interfaces. Em particular, tendem a ser criticadas pela fraca integração no processo de desenvolvimento de software. Por um lado, uma geração baseada em modelos tende a ter problemas na integração com o processo criativo dos *designers*, tendo um impacto negativo na qualidade da IU gerada [9]. Por outro, os modelos utilizados tendem a divergir dos modelos da camada de negócio usados em abordagens *Model-driven engineering* (MDE), levantando problemas de coordenação e produtividade [2,10].

Este artigo apresenta uma nova abordagem, suportada pela ferramenta *MODUS* (*MOdel-based Developed User Systems*), propondo a seguinte lista de contribuições:

- Tira partido da separação entre a definição do conteúdo e aspeto gráfico da interface (neste caso, para aplicações Web) de forma a gerar "esqueletos", os quais poderão ser refinados pelos *designers*.
- Centra-se no uso de um modelo estrutural do sistema, um diagrama de classes em *Unified Modelling Language* (UML) [11], promovendo a integração entre as abordagens MDE e MBUID. O diagrama de classes por definição enuncia todas as classes e respetivas relações necessárias para o desenvolvimento do sistema, sendo fundamental num contexto de MDE. Na abordagem proposta, aparenta ser uma opção apropriada pois as entidades que o compõem são decisivas no conteúdo da aplicação, sendo específicos à mesma.
- Tira partido da definição prévia do domínio da aplicação para limitar a necessidades de modelos adicionais no processo de criação, possibilitando um processo de geração substancialmente mais automatizado. Com efeito, para o mesmo domínio de aplicação, constata-se que as interfaces de utilizador tendem a ser semelhantes em navegação, estrutura e componentes visuais.
- Nesta abordagem o conceito de *Evolutionary Prototyping* [12] é indispensável. O utilizador da ferramenta tem a capacidade de manipular o resultado em qualquer uma das etapas do processo de geração, o qual será refinado até a obtenção da interface final. Graças ao seu cariz iterativo a interface elaborada satisfaz um maior número de requisitos do utilizador.
- Integra a possibilidade de seleção de detalhes de aparência da interface, tal como *front end frameworks*, *responsive design* para diferentes dispositivos, *templates*, entre outros. Deste modo, o utilizador terá um maior controlo sobre o aspeto visual do resultado final da geração.

O artigo está organizado segundo a seguinte estrutura. A secção 2 introduz os conceitos de base e a secção seguinte 3 aborda trabalhos relacionados ao contexto da abordagem. Posteriormente é apresentada a metodologia proposta e a ferramenta que a suporta na secção 4 e 5, respetivamente. Por fim, a secção 6 apresenta conclusões e o trabalho futuro a ser realizado.

2 Contextualização

2.1 *Model-Driven Engineering*

Os modelos, na engenharia de software, permitem que os programadores analisem o sistema antes de iniciar a implementação, concretizando idealmente a especificação que levará à solução final. Ao longo do tempo esta tendência tornou-se para muitos a norma, dando origem ao paradigma de MDE [13,14].

Nesta metodologia, modelos de alto nível de abstração, que representam o problema que o sistema enfrenta, são convertidos sucessivamente em modelos de nível inferior, com o objetivo de os transformar em sistemas executáveis [13,14]. Esta conversão pode ser manual ou automatizada. Em ambas abordagens a utilização de modelos permite assegurar a coerência no desenvolvimento do software, garantindo uma melhor qualidade e correção do resultado final. Acima de tudo, ao recorrer a uma transformação automática dos modelos para o código executável, assegura-se uma redução do tempo de desenvolvimento [10,15].

MDE é tipicamente aplicada à criação das camadas da lógica de negócio e de dados [16,17]. Todavia os modelos elaborados neste contexto poderiam ser reutilizados para a geração da IU, sendo uma das premissas da abordagem proposta.

2.2 *Model-Based User Interface Development*

MBUID tende a ser visto como MDE de interfaces de utilizador. É uma metodologia que procura reduzir a quantidade de tempo e esforço despendida no desenvolvimento da IU, garantindo a qualidade da interface produzida [10,15,18].

A *Cameleon Reference Framework*, amplamente aceite como a arquitetura de referência para MBUID, especifica quatro níveis principais de modelação[19,20,21]:

1. **Modelo de Domínio e Tarefas** - descrição das tarefas do utilizador e dos conceitos de domínio relacionados com a sua realização.
2. **Abstract User Interface (AUI)** - descrição da interface em termos de *Abstract Interaction Units* ou *Abstract Interaction Objects* e respetivas relações. AUI é independente da tecnologia e modalidade de interação.
3. **Concrete User Interface (CUI)** - descrição da interface em termos de *Concrete Interaction Units* ou *Concrete Interaction Objects*, definindo o *layout* e a navegação. Sendo a CUI dependente da modalidade de interação, descreve o *Look & Feel* da aplicação.
4. **Final User Interface (FUI)** - descrição da interface em termos do código fonte, quer numa linguagem de programação ou de *mark-up*. A FUI pode ser interpretada ou executada, após compilação do código.

A geração em MBUID baseia-se em modelos declarativos de alto nível. No entanto é frequente usarem-se modelos detalhados sobre diferentes aspetos da interface. Este facto limita a aceitação da abordagem, pois existe um custo acrescido de modelação, quer em número de modelos, quer em esforço de elaboração [10,15,18]. Decidiu-se, assim, centrar a abordagem proposta num processo de geração baseado num modelo estrutural do sistema complementando com a especificação do domínio da aplicação. Este diagrama é a base da criação, parcial ou completa, da IU, podendo ser refinada com apoio de ferramentas adequadas.

3 Trabalho Relacionado

Diversas abordagens baseadas em modelos têm vindo a ser desenvolvidas, tentando responder aos desafios trazidos pela evolução das interfaces de utilizador. Podem ser identificadas quatro gerações de ferramentas MBUID.

As ferramentas da primeira geração baseavam-se num modelo declarativo universal, apoiando-se numa criação totalmente automatizada. O foco era a geração de aplicações *Desktop* baseadas em operações *Create, Read, Update and Delete* (CRUD), como as apresentadas em [22,23,24]. As interfaces tendiam a ser simples, seguindo sempre o mesmo modelo visual.

A segunda geração define o modelo da IU como uma composição de modelos declarativos. A criação da interface é um processo cumulativo, procurando melhorar a qualidade da IU à custa da especificação de modelos adicionais. Nestas ferramentas o nível de automação tende a ser baixo, sendo que um aumento no grau de automação implica a inclusão de informação auxiliar sobre a interface.

A terceira geração foca-se nos desafios trazidos pelo aparecimento de novos dispositivos e plataformas. *TERESA* [25] é um exemplo típico desta geração. Para solucionar os problemas a especificação é dependente do contexto. Para cada contexto suportado, é necessário definir novas versões dos modelos. Para IUs de aplicações web, o uso de tecnologias *front end* como *Javascript* e *CSS3* permitem aumentar significativamente a compatibilidade entre diferentes dispositivos e plataformas em tempo de execução [26]. A integração dessas tecnologias no processo de geração poderá ser uma alternativa à especificação do contexto.

A quarta geração centra-se na elaboração de interfaces sensíveis ao contexto, através do desenvolvimento *multi-path* [27] e na integração com *web services*. A proposta apresentada é em grande medida ortogonal a estas preocupações, focando-se, nesta fase, em fornecer uma solução eficiente para a criação de interfaces, tendo como ponto de partida modelos estruturais da camada de negócio.

Os desenvolvimentos verificados em MBUID têm vindo a ser aplicados na indústria de software. Ferramentas foram elaboradas adotando alguns conceitos mais estabilizados deste paradigma. Exemplos são *Outsystems* [28] e *Integranova* [29]. *Outsystems* pretende aumentar a produtividade dos programadores. Embora crie uma IU a partir de um modelo de base de dados, o nível de automação é relativamente baixo, sendo a interface construída num editor *What You See Is What You Get* (WYSIWYG). Por sua vez, *Integranova* visa a criação de todas

as camadas da aplicação. Gera uma IU baseada em CRUD com um alto nível de automação, sem possibilidade de melhoria durante o processo.

De um modo geral, constata-se que as ferramentas mais automatizadas necessitam da inclusão de informação adicional, gerando interfaces mais rígidas e menos apelativas. As ferramentas mais flexíveis requerem uma maior intervenção do utilizador, não resolvendo a questão dos custos de desenvolvimento. A abordagem proposta neste artigo tenciona conciliar ambos aspetos, procurando contribuir para a solução do problema.

4 Abordagem

A Figura 1 resume a abordagem apresentada neste artigo. Em termos gerais, o processo interpreta o *input* do utilizador para gerar a IU, sob a forma de uma interface *Web*. No entanto, a abordagem contém etapas intermédias. Na Figura 1 cada etapa está etiquetada numericamente, sendo o fluxo representado por setas. Os dados de entrada estão etiquetados alfabeticamente, em maiúsculas os fornecidos pelo utilizador (específicos de uma interface em concreto) e em minúscula os disponibilizados pela ferramenta (de uma aplicação genérica).

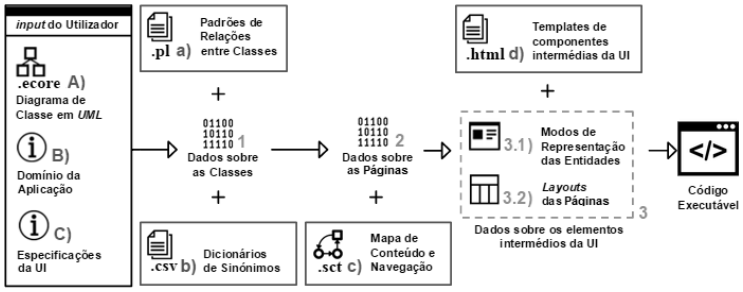


Figura 1: Visão global da abordagem

O processo é iniciado com (Figura 1, *input* do Utilizador): A) o diagrama de classes em UML no formato *ecore* [30] (ver exemplo na Figura 2); B) o domínio da aplicação, o qual é explicitado na ferramenta; C) informação adicional sobre a geração da IU, como por exemplo a *front end framework* a utilizar, a qual é definida na ferramenta. Estes dados são essenciais para todo o processo.

A geração da interface é composta por três etapas intermédias. Na primeira etapa interpretam-se os dados de entrada através de a) padrões arquitecturais e b) bases de conhecimento semântico, de forma a identificar entidades relevantes no modelo. Na segunda etapa definem-se as páginas da interface, a nível de conteúdo e navegação, a partir de c) um modelo abstrato da interface para o domínio inicialmente definido. Na terceira etapa procede-se à concretização das componentes intermédias da interface, nomeadamente modos de representação de entidades e *layouts* das páginas, através do uso de d) *templates*.

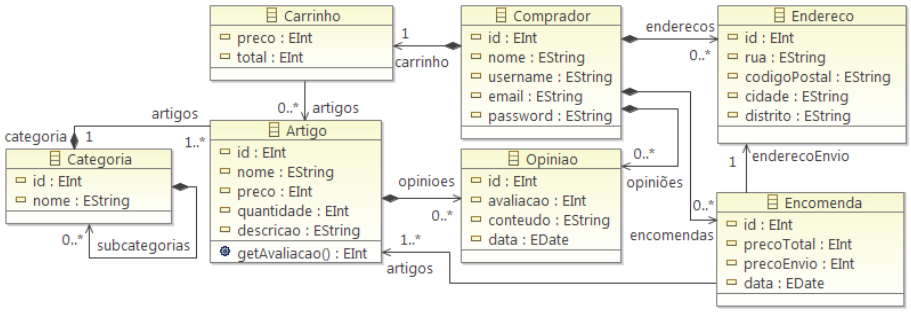


Figura 2: Diagrama de Classes em UML de uma aplicação web do tipo *eCommerce*

4.1 Reconhecimento das Classes *Standard*

Uma entidade, por outras palavras, uma classe do diagrama de classes, pode ser associada a uma classe *standard*. As classes *standard* correspondem a classes que usualmente existem na modelação de um dado domínio de aplicação. Um exemplo seria associar a entidade `Artigo` da Figura 2 à classe `Product` no domínio `eCommerce`. Esta análise permite estabelecer a base de muitas das decisões tomadas sobre o desenho da interface ao longo do processo de criação. O processo de associação é efetuado em duas etapas. A primeira etapa consiste na identificação de padrões arquiteturais em que as classes *standard* tipicamente estão envolvidas. A segunda etapa resume-se à estimativa do grau de compatibilidade entre o nome da entidade identificada e o da classe *standard*.

Na primeira etapa, transforma-se o conhecimento presente no diagrama de classes numa ontologia *Prolog*. Optou-se por usar uma linguagem de programação lógica de modo a inferir sobre a base de conhecimento da ontologia extrapolada. Desta forma, a informação das entidades é expressa em factos, pois são predicados que representam o diagrama. Por sua vez, cada padrão é descrito numa regra, a qual é consultada para obter a veracidade da correspondência com cada uma das entidades. Este processo baseia-se na abordagem apresentada em [32].

A base de conhecimento é composta por dois tipos de factos, representados na Listagem 1.1. O facto da linha 1 enuncia a existência de uma entidade com o nome `E` (`Entity`). O facto da linha 2 especifica a existência de uma relação entre entidades, sendo: `R` (`Relation`) o nome da relação; `F` (`From`) a entidade de partida; `T` (`To`) a entidade de chegada; `L` (`Lower Bound`) o limite inferior e, por fim, `U` (`Upper Bound`) o limite superior da cardinalidade de `T` na relação. Um exemplo é apresentado na Listagem 1.2, no qual existem duas entidades, Comprador e Endereço, em que um comprador pode ter múltiplos endereços.

```
1 entity(E).
2 relation(R,F,T,L,U).
```

Listagem 1.1: Factos da ontologia em *Prolog*

```
1 entity(comprador).
2 entity(endereco).
3 relation(enderecos,comprador,endereco,0,n).
```

Listagem 1.2: Exemplos de Factos da ontologia em *Prolog*

Uma interrogação corresponde à verificação da existência de um padrão de relacionamento entre classes. Neste contexto, um padrão de relacionamento retrata um conjunto de relações específicas existentes entre uma classe *standard* e as restantes, as quais tendem a ser replicadas em modelações de aplicações pertencentes ao mesmo domínio. Por exemplo, a *Listagem* 1.3 descreve o padrão da classe *standard* `Address` do domínio `eCommerce`. É definido que uma classe `U` (Utilizador) possui endereços e que uma classe `O` (Encomenda) tem pelo menos um endereço, sendo que `A` (Endereço) é uma entidade da ontologia.

```
1 ecommerce_address_pattern(A):-
2   entity(A),                    % Address is an entity
3   relation(_R,U,A,_M,_N),       % User has Address
4   has_at_least(O,A,1),          % Order has at least 1 Address
5   different(A,U), different(A,O), different(O,U).
```

Listagem 1.3: Exemplo de um padrão de relacionamento em *Prolog*

Uma vez que diferentes padrões podem coincidir entre eles, é provavél que para cada classe *standard* sejam identificadas mais do que uma entidade válida. Desta forma é necessário um mecanismo de selecção da solução mais provável de cada lista de resultados. É assim iniciada a segunda etapa.

Uma classe *standard* possui um dicionário (um documento CSV) que agrega um conjunto de sinónimos com a respetiva probabilidade de corresponderem ao nome da classe em questão. O nome de uma entidade válida é pesquisado no dicionário, sendo comparado a cada um dos registos. A percentagem de acerto é ajustada conforme o grau de semelhança com o sinónimo e a respetiva probabilidade encontrada no documento. A entidade com maior probabilidade é definida como solução da classe *standard*. Uma análise semelhante é efetuada para os atributos *standard*, tendo desta vez em conta os seus tipos respetivos.

4.2 Mapa de Conteúdo e Navegação

Nesta abordagem é utilizado um modelo auxiliar, o mapa de navegação e conteúdo. Este modelo, sob a forma de diagrama, permite representar não só as páginas, porções de páginas (parciais) e os respetivos conteúdos, bem como a navegação entre os elementos. Este diagrama unifica num modelo toda a informação necessária para a geração da IU de um dado domínio de aplicação. A representação deste mapa é baseada num diagrama de estados em UML já que este modelo permite enunciar os diferentes estados pelo qual um objeto pode transitar durante a execução de um sistema. Esta perspetiva pode ser comparada às transições entre páginas ou mudanças de conteúdo durante a execução de uma

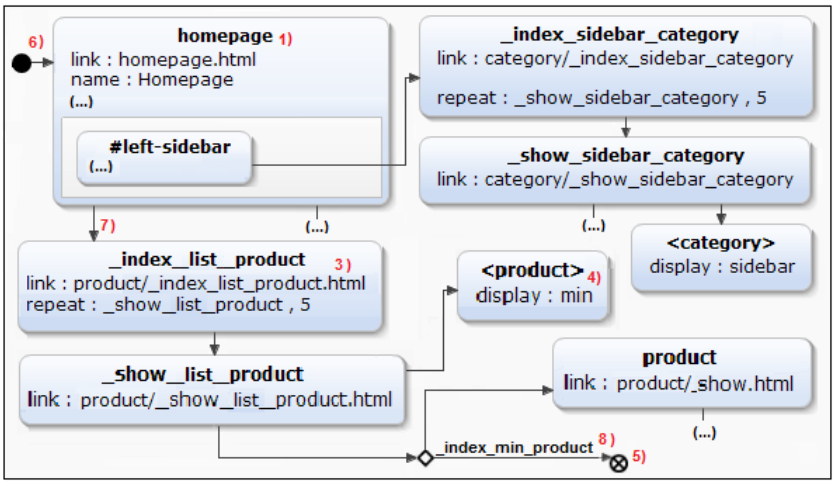


Figura 3: Exemplo demonstrativo do Mapa de navegação e conteúdo (fragmento)

aplicação baseada em *browser*. Devido à necessidade de especificar igualmente o conteúdo, foi necessário adaptar a interpretação de alguns elementos:

Estado Representa uma página, parcial, secção de página ou modo de representação de uma entidade. Os tipos associados a esta componente são distinguidos pelos identificadores ou pela sua localização no diagrama. Se o estado estiver numa região de outro estado, corresponde a uma secção da página (ver 2) da Figura 3). Os identificadores dos parciais são iniciados pelo caráter `'_'` (ver 3) da Figura 3) e os identificadores dos modos de representação de entidades são delimitados por `'<'` e `'>'` (ver 4) da Figura 3). Os restantes estados identificam páginas (ver 1) da Figura 3). Os estados possuem atributos para representar informação auxiliar.

Estado final Indica a saída de uma transição por uma condição específica (ver 5) da Figura 3).

Estado inicial Indica a página inicial da interface (ver 6) da Figura 3).

Transição Indica uma transição, substituição ou composição, dependendo do elemento de destino (ver 7) da Figura 3). Uma transição para uma página representa uma mudança de página. Uma transição para um parcial representa a substituição do conteúdo da secção de origem. Finalmente, uma transição para um modo de representação equivale a uma composição de conteúdo.

Condição Enuncia os identificadores relacionados com uma transição (ver 8) da Figura 3).

Um exemplo é apresentado na Figura 3. O mapa descreve que o sistema terá uma página inicial `homepage`. Esta página tem uma secção `#sidebar` que contém o parcial `_index_sidebar_category`, composto por uma repetição de `_show_sidebar_category`, o qual inclui o modo de representação `sidebar` da classe `Category`. Esta combinação é frequentemente usada para representar uma

lista de um parcial ou de uma entidade segundo um dado modo. A página `homepage` é composta por uma lista de `Produto` em `min`, sendo que cada elemento reencaminha para a página `product` caso o estado não seja `_index_min_product`.

4.3 Geração das componentes intermédias da IU

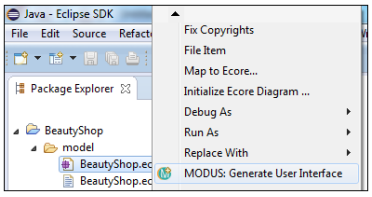
Nesta abordagem é estipulada que uma página é composta por um *layout*, que irá definir a sua estrutura, e um conjunto diferentes entidades, que irão definir o seu conteúdo. Para esse efeito os modos de representação definem, para a tecnologia da FUI, as diferentes formas de representar uma dada entidade na aplicação. Exemplos seriam uma representação completa, em miniatura, ou *hyperlink* de uma classe. Se não estiver associado a uma entidade, um modo de representação descreve um conteúdo independente da modelação, como por exemplo um *slideshow* de uma página. Por sua vez um *layout* identifica, na tecnologia da FUI, as secções que definem a página. Nomeadamente, explicita uma combinação de secções, sendo estas por exemplo *headers*, *footers*, entre outros.

O processo de criação destas componentes é executado pelo *parsing* de *templates* fornecidos pela ferramenta ou, se assim o pretender, pelo utilizador. Durante a geração, o *template* é adaptado ao *input* fornecido no programa bem como às respetivas alterações efetuadas pelo utilizador. Os *templates* são expressos em HTML, contendo atributos específicos para apoiar a criação do *output*, indicando as ações a executar, elementos da modelação, interface ou mapa de navegação. No entanto, durante a etapa final são removidos de forma a não poluírem a FUI. Os *templates* podem ser modificados pelo utilizador em *runtime*, personalizando assim as componentes intermédias que irão compor a FUI.

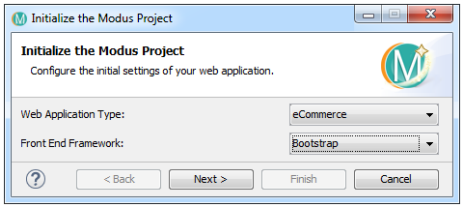
Uma criação à base de *templates* permite estabelecer um mecanismo de geração independente do resultado. A adição, remoção ou manipulação dos *templates* não influencia a implementação do processo. Esta metodologia adequa-se à área das interfaces de utilizador, em que conceitos de estética são frequentemente alterados, sofrendo uma evolução constante. A variedade de *templates* e possivéis combinações permite aumentar a flexibilidade da IU criada. Por sua vez, a importação de *templates* incrementa a diversidade de soluções geradas.

5 O *Plugin* MODUS

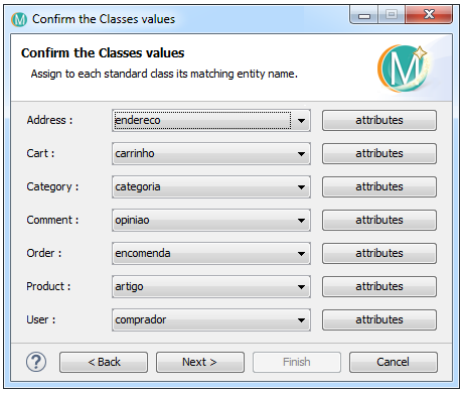
A abordagem proposta neste documento é sustentada pela ferramenta *MODUS*. Em termos de implementação, o protótipo da ferramenta é um *plugin* do *Integrated Development Environment* (IDE) *Eclipse* [31], sendo desenvolvido em *Java*. Por ser uma extensão, beneficia dos recursos disponibilizados pelo IDE, nomeadamente dos editores gráficos e da marcação de documentos para validação. Esta integração adiciona uma nova funcionalidade ao IDE, evitando que o utilizador instale um software adicional.



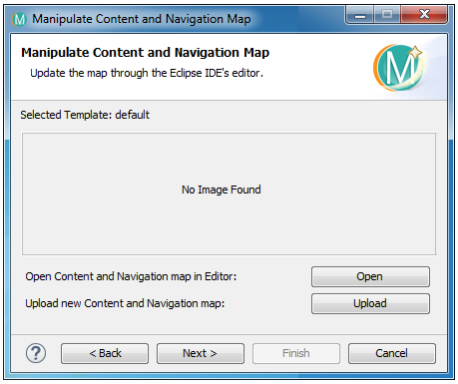
(a) Arranque do *plugin MODUS*



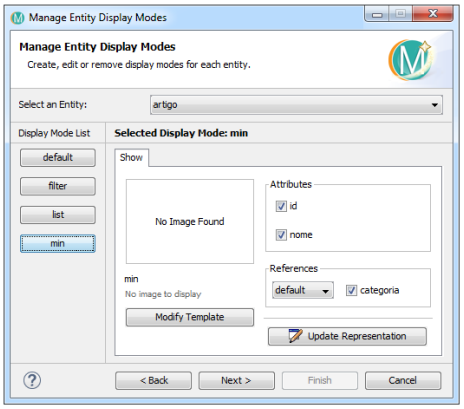
(b) *Input* do Utilizador



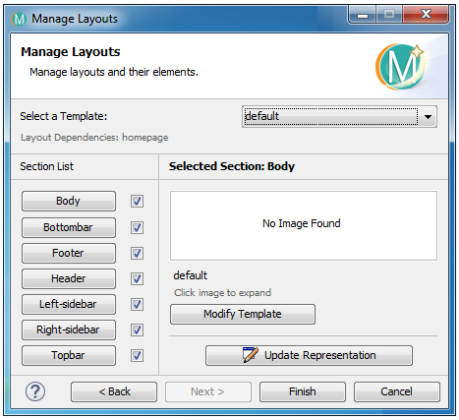
(c) Associação das classes *standard*



(d) Mapa de conteúdo e navegação



(e) Modos de representação das entidades



(f) *Layouts* das páginas

Figura 4: Interface do *plugin MODUS*

5.1 *Input* do Utilizador

Para iniciar a execução do *plugin*, é necessário importar o diagrama de classes no IDE. Uma vez carregado, pode-se selecionar, no menu de contexto associado ao formato *ecore*, a opção de gerar a IU pelo *MODUS*. Para efeitos de demonstração considera-se o modelo apresentado na Figura 2 (ver Figura 4a). Após selecionar a opção, o ficheiro é validado. Caso sejam detetados erros, é aberto, no IDE, o documento com as respetivas marcações.

A Figura 4b expõe o menu de configuração da geração, sendo o primeiro contacto com a interface do *plugin*. O menu possibilita a seleção do domínio de aplicação e das especificações da interface (as quais, no estado atual do protótipo, encontram-se incompletas). Os botões de *Next*, *Back*, *Cancel* e *Finish* permitem, respetivamente, continuar, retroceder, cancelar e terminar o processo de geração. O botão de ajuda disponibiliza alguma informação sobre a etapa atual.

5.2 Associação das Classes *standard*

A Figura 4c apresenta a interface dedicada à associação das entidades as respetivas classes *standard*. O resultado calculado é exibido, podendo ser modificado pelo utilizador. Os botões *attributes* abrem uma janela semelhante dedicada aos atributos da entidade. Pelo momento, a gama de atributos *standard* conhecidos pela ferramenta é reduzida.

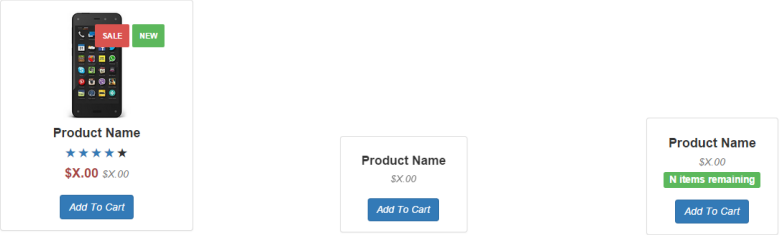
5.3 Mapa de Conteúdo e Navegação

A Figura 4d apresenta a interface dedicada ao mapa de navegação e conteúdo (ver exemplo da Figura 3). Caso pretenda, o utilizador pode alterar o mapa através do botão *Update*, o qual desencadeia a abertura do diagrama no editor gráfico disponibilizado pelo IDE. O utilizador pode igualmente importar outro modelo da lista de mapas da ferramenta, através do botão *Upload*.

5.4 Modos de Representação das Entidades

A Figura 4e expõe a interface dedicada aos modos de representação das entidades. O *drop down* situado no topo (*Select an Entity*) permite selecionar uma entidade. Esta escolha desencadeia a listagem, no lado esquerdo da interface (*Display Mode List*), dos modos de representação associados à entidade. A seleção de um dos modos, preenche o lado direito da interface com a respetiva informação. Os *tabs* permitem aceder aos tipos de apresentação disponíveis para cada modo: exibição (*Show*), que exibe a entidade na aplicação segundo um dado modo, e formulário (*Form*), que apresenta o formulário na aplicação do modo em questão. No exemplo da Figura 4e o modo `min` é apenas composto pelo tipo *Show*. Para cada alternativa é referido o *template* associado e as componentes da entidade permitidas pelo mesmo.

O botão *Modify Template* permite associar um novo *template*, quer seja selecionado a partir de uma lista pré-definida, quer seja carregado pelo utilizador. O



(a) *Template* original do modo `list` da entidade `artigo` (ver Figura 2)

(b) *Template* ajustado do modo `list` da entidade `artigo` (ver Figura 2)

(c) *Template* modificado do modo `list` da entidade `artigo` (ver Figura 2)

Figura 5: Evolução de um modo de representação na ferramenta

template é atualizado conforme a respetiva modelação antes de ser apresentado na ferramenta (ver Figura 5a). A ter em conta que no caso particular do *template* da Figura 5b há uma restrição do conteúdo a um grupo de atributos específicos. Por sua vez, O botão *Update Representation* permite editar o modo no editor HTML do IDE, tal como demonstrado na Figura 5c.

5.5 *Layouts* das *Views*

A Figura 4f apresenta a interface dedicada aos *layouts* das páginas. Esta janela aproxima-se à interface dos modos de representação de entidades, sendo a sua interação com o utilizador semelhante. O *drop down* (*Select a Layout*) permite selecionar um layout. A listagem à esquerda enumera as secções de um *layout*, sendo que cada *checkbox* define a existência da respetiva secção no *layout*. À direita apresentam-se os dados da secção selecionada. Os *templates* são igualmente ajustados conforme as alterações efetuadas no *plugin*.

5.6 Geração da interface

Após terem sido efetuados os ajustes às componentes intermédias da interface, é possível iniciar a criação da FUI. A interface final será composta por elementos utilizados no código da lógica de negócio: os parciais (ver exemplo da Figura 6) e os *layouts* (ver exemplo da Figura 7) das páginas da aplicação. Após a geração desses elementos, a FUI será acompanhada por um conjunto de páginas estáticas que irão simular a interface por completo (ver exemplo da Figura 8) . Estas páginas são elaboradas pela composição dos *layouts* com os *parciais*, conforme a sua definição no mapa de navegação e conteúdo.

Note-se que, a interface apresentada nas Figuras 7, 6 e 8, representa uma versão simples da IU gerada automaticamente. O objetivo é mostrar que é viável gerar as interfaces com conteúdos relevantes para a aplicação. No entanto, através do recurso a *front-end frameworks* apropriadas, está já a ser desenvolvido suporte à personalização avançada da interface.

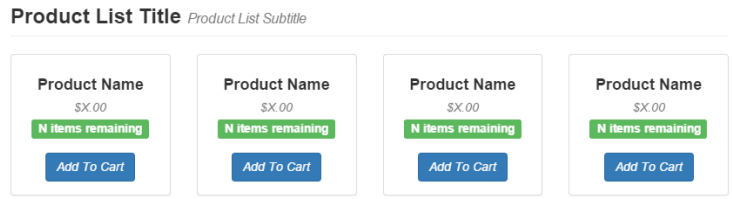


Figura 6: Parcial `_index_list_product` (ver Figura 2)

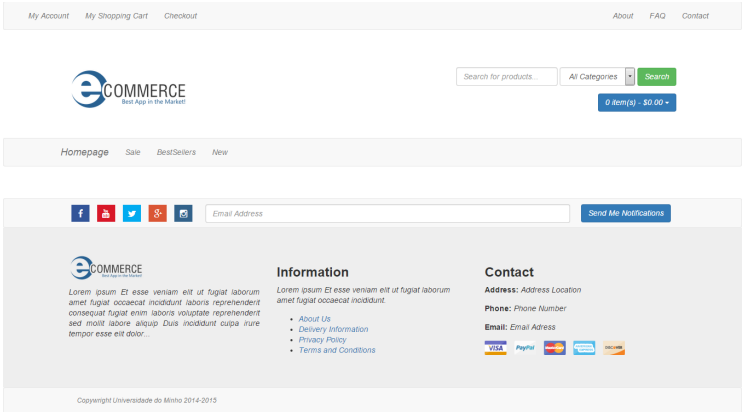


Figura 7: *Layout* `default` (ver Figura 2)

6 Conclusões e Trabalho Futuro

Este documento apresentou uma abordagem baseada em MDE e MBUID para a geração de interfaces de utilizador com um alto nível de automação. Na solução proposta o diagrama de classes e o domínio da aplicação são elementos chave. O diagrama de classes enuncia todas as entidades que irão intervir na IU. Por sua vez, o domínio é a base de diversas assunções sobre a interface, substituindo-se a necessidade de a modelar detalhadamente.

A implementação dos modos de representação de entidades e dos *layouts* é baseada na separação do conteúdo e estilo da interface de aplicações web baseadas em *browser*. Desta forma, os *templates* permitem definir detalhes estéticos da IU, contribuindo para a elaboração de interfaces complexas e apelativas. A incorporação de *front end frameworks* e, posteriormente de restantes recursos de estética, permite tirar partido dessa contribuição, possibilitando igualmente aumentar a compatibilidade entre dispositivos/plataformas e melhorar a aparência da própria interface.

Para suportar a abordagem está a ser desenvolvido um *plugin MODUS* para o *Eclipse*. Um protótipo foi elaborado, implementando por completo as três etapas intermédias para a conceção da interface do utilizador. A geração da FUI produz parciais, *layouts* das páginas e um conjunto de páginas estáticas, pos-

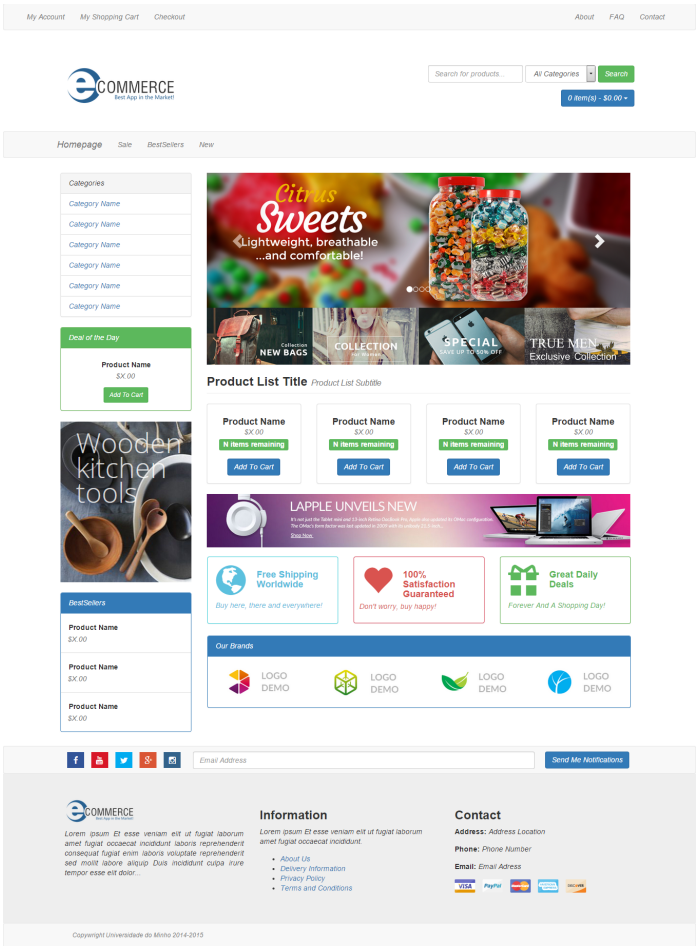


Figura 8: Página `homepage` (ver Figura 2)

suindo um alto nível de automatização. Constata-se que o grau de automação é influenciado pela intervenção do utilizador, o qual permite no entanto aperfeiçoar as componentes que irão compor a interface, garantindo-se melhores resultados.

Como trabalho futuro, tenciona-se adicionar novas possibilidades de configuração da interface, nomeadamente a seleção de temas visuais e paletas de cores. Para além disso pretende-se completar o conjunto de atributos *standard* reconhecidos pela ferramenta, de forma a suportarem um vasto leque *templates* complexos, os quais deverão igualmente ser desenvolvidos.

A validação da abordagem incluirá duas componentes distintas. Por um lado, será necessário comparar a implementação de interfaces utilizando a ferramenta com o desenvolvimento seguindo abordagens mais manuais, por forma a avaliar precisamente qual a poupança em termos de esforço de programação. Isso po-

derá ser realizado, avaliando quer o tempo gasto, quer o código produzido nos projetos em cada uma das condições. Por outro lado, será necessário comparar a qualidade das interfaces produzidas, a fim de avaliar qual o impacto do processo de automatização na qualidade final das interfaces

Referências

1. Brad A. Myers, Scott E. Hudson, and Randy F. Pausch, "Past, present, and future of user interface software tools", ACM Trans. Comput.-Hum. Interact., 7(1):3–28, 2000.
2. Gerrit Meixner, Fabio Paternò and Jean Vanderdonckt, "Past, present, and future of model-based user interface development", i-com, 10(3):2–11, 2011.
3. Brad A. Myers and Mary Beth Rosson, "Survey on user interface programming", Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '92, 195–202, ACM, 1992.
4. Sanjay Mittal, Clive L. Dym and Mahesh Morjaria, "Pride: An expert system for the design of paper handling systems", Computer, 19(7):102–114, 1986.
5. D. G. Bobrow, S. Mittal and M. J. Stefik, "Expert systems: Perils and promise", Commun. ACM, 29(9):880–894, 1986.
6. Richard Kennard and Robert Steele, "Application of software mining to automatic user interface generation", New Trends in Software Methodologies, Tools and Techniques - Proceedings of the Seventh SoMeT, 244-254, 2008.
7. Brad A. Myers, "State of the art in user interface software tools", Technical report, Carnegie Mellon University, USA, 1992.
8. Brad A. Myers, "User interface software tools", Technical report, Carnegie Mellon University, USA, 1994.
9. Pedro J. Molina, "A review to model-based user interface development technology", MBUI, Proceedings of the First International Workshop on Making model-based user interface design practical: usable and open methods and tools, Portugal, 2004.
10. Gerrit Meixner and Gaelle Calvary "Introduction to model-based user interfaces", W3C, 2014, http://www.w3.org/TR/2014/NOTE-mbui-intro-20140107/.
11. Fowler, "UML Distilled: A Brief Guide to the Standard Object Modeling Language"AddisonWesley Longman Publishing Co., Inc., 3 edition, USA, 2003.
12. A.M. Davis, "Operational Prototyping: A New Development Approach", IE Software, 9(5):70-78, 1992.
13. Thomas Stahl, Markus Voelter, and Krzysztof Czarnecki, "Model-Driven Software Development: Technology, Engineering, Management", John Wiley & Sons, 2006.
14. Jorg Rech and Christian Bunse, "Model-Driven Software Development: Integrating Quality Assurance", Information Science Reference - Imprint of: IGI Publishing, Hershey, 2008.
15. Egbert Schlungbaum, "Model-based user interface software tools current state of declarative models", Technical report, Graphics, Visualization and Usability Centre, Georgia Institute of Technology, GVU Tech Report, 1996.
16. P. Kidwell, "The mythical man-month: Essays on software engineering", IEEE Ann. Hist. Comput., 18(4):71–, 1996.
17. B. Hailpern and P. Tarr, "Model-driven development: The good, the bad, and the ugly", IBM Syst. J., 45(3):451–461, 2006.
18. Paulo Pinheiro Da Silva, "User interface declarative models and development environments: A survey", Proceedings of the 7th International Conference on Design,Specification, and Verification of Interactive Systems, 207–226, 2001.

19. Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt "A unifying reference framework for multi-target user interfaces", Interacting With Computers, 15:289–308, 2003.
20. Jean Vanderdonckt "A mda-compliant environment for developing user interfaces of information systems", Proc. of 17th Conf. on Advanced Information Systems Engineering, CAiSE'05 16–31, Springer-Verlag, 2005.
21. Jean M. Vanderdonckt and François Bodart, "Encapsulating knowledge for intelligent automatic interaction objects selection", In Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems, CHI'93, 424–429, ACM, 1993.
22. Dennis J. M. J. de Baar, James D. Foley, and Kevin E. Mullet, "Coupling application design and user interface design", In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI'92, 259–266, ACM, 1992.
23. Pedro Szekely, Ping Luo, and Robert Neches, "Beyond interface builders: Model-based interface tools", In Proceedings of the INTERCHI'93 Conference on Human Factors in Computing Systems, INTERCHI'93, 383–390, 1993.
24. Angel R. Puerta, Henrik Eriksson, John H. Gennari, and Mark A. Musen, "Beyond data models for automated user interface generation", In Proceedings of the Conference on People and Computers IX, HCI '94, 353–366, 1994.
25. Silvia Berti, Francesco Correani, Giulio Mori, Fabio Paternò, and Carmen Santoro, "Teresa: A transformation-based environment for designing and developing multi-device interfaces", In CHI '04 Extended Abstracts on Human Factors in Computing Systems, CHI EA '04, 793–794, ACM, 2004.
26. A.I. Sampaio and J.C. Campos, "Towards a Framework for Adaptive Web Applications", In C. Stephanidis, HCI International 2014 - Posters' Extended Abstracts, Part I, V.434 of Communications in Computer and Information Science, 240-245, Springer, 2014.
27. Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, Víctor López-Jaquero, "Usixml: A language supporting multi-path development of user interfaces", Engineering Human Computer Interaction and Interactive Systems, V. 3425 of Lecture Notes in Computer Science, 200–220, 2005.
28. IT Resources, Outsystems, http://www.outsystems.com/itresources/, 2011.
29. Integranova Software Solutions, Integranova, http://www.integranova.com/, 2005.
30. Frank Budinsky, Dave Steinberg, Ed Merks, Ray Ellersick, Timothy J. Grose, "Eclipse Modeling Framework : A Developer's Guide", Addison-Wesley Professional, 14–20, 2003.
31. The Eclipse Foundation, Eclipse, https://eclipse.org/, 2001.
32. Rui Couto, António N. Ribeiro, and José C. Campos, "Mapit: A model based pattern recovery tool", V.7706 of Lecture Notes in Computer Science, 19–37, 2013.