6
Exemplos

Este capítulo descreve a elaboração do modelo de interface para uma aplicação de demonstração que pode ser obtida junto com o código do próprio *framework* HyperDE.[73]

Suponha uma aplicação cuja finalidade seja exibir os relacionamentos entre os docentes e alunos no Departamento de Informática da PUC-Rio. O diagrama a seguir retrata o esquema navegacional da aplicação. As classes navegacionais e os elos (*links*) entre elas foram definidos no ambiente HyperDE.

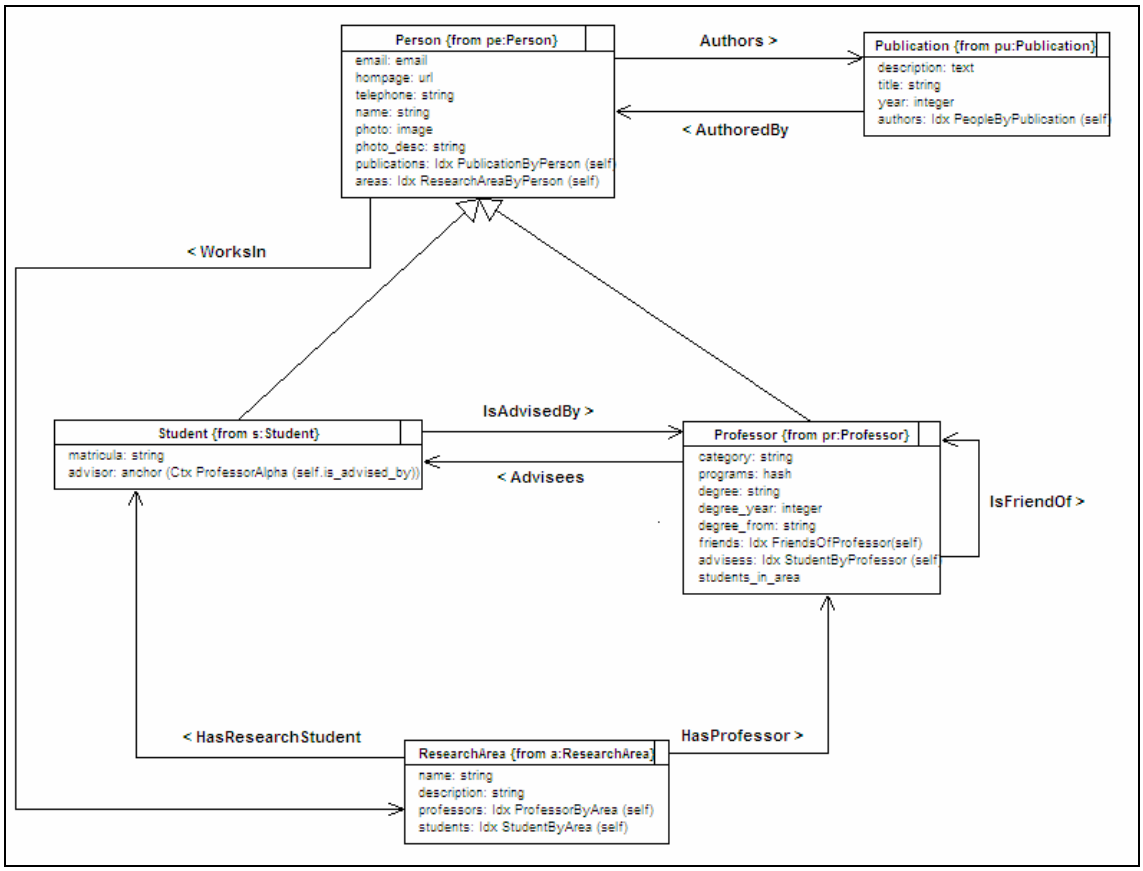

Figura 86 – Esquema navegacional do site do Departamento de Informática (Site DI)

Seguem agora as definições de alguns dos contextos e índices que figuram como tipos de atributos das classes navegacionais:

[73] A versão do HyperDE com as funcionalidades descritas neste trabalho, acompanhada da aplicação de demonstração, pode ser obtida em http://www.tecweb.inf.puc-rio.br/hyperde/browser/branches/hyperde_amluna

- Contexto **ProfessorAlpha** – Conjunto dos nós pertencentes à classe **Professor**, ordenados de forma crescente pelo atributo **name**;
- Índice **StudentsByProfessor** – Estrutura de acesso associada ao contexto de mesmo nome. As entradas do índice são os elementos do contexto derivado de elo **StudentsByProfessor**: o conjunto dos nós da classe **Student** que estão relacionados ao nó **Professor** pelo elo **IsAdvisedBy**.

O objetivo do modelo de interfaces documentado neste capítulo é demonstrar os principais recursos da Arquitetura SWUI de Interfaces Ricas, por exemplo:

- Modelagem de interfaces de contexto e de índice navegacional, o que inclui a ligação de dados (*databinding*) entre os objetos de percepção e os objetos de navegação;
- Modelagem de outros componentes tipicamente presentes nas aplicações HyperDE tais como: *landmarks* e estruturas de acesso (índices de contexto e âncoras para anterior e próximo elementos dentro de um contexto);
- Integração dos modelos abstratos com a linguagem de estilo CSS;
- Modelagem de controles de interface sensíveis a eventos de usuário;
- Modelagem de transições animadas;
- Modelagem de ativadores para as operações do Modelo;
- Comunicação assíncrona entre as camadas de Visão e Modelo, mediada pela fila de respostas da transação.

6.1.1. Interface ProfessorView

A interface **ProfessorView** tem por finalidade renderizar os nós da classe navegacional *Professor*. Os atributos que serão renderizados nesta interface estão resumidos na tabela a seguir. Optamos por agrupar os atributos semanticamente em quatro conjuntos: dados pessoais, dados acadêmicos, informações para contato, lista de orientandos, lista de áreas de pesquisa, lista de amigos e lista de publicações:

Classe do Atributo	*Atributo*	*Tipo*
AcademicData	category	string
	degree	string
	degree_from	string
	degree_year	integer
AdviseesList	advisees	indexnavattribute
AreasList	areas	indexnavattribute
ContactInfo	email	email
	homepage	url

Classe do Atributo	*Atributo*	*Tipo*
	telephone	string
FriendsList	friends	indexnavattribute
PersonalData	name	string
	photo	image
PublicationsList	publications	indexnavattribute

Tabela 4 – Atributos da classe navegacional *Professor*

A interface apresenta informação somente de leitura; por isso, utilizaremos para mapear cada atributo da classe navegacional um *widget* do tipo *ElementExhibitor*. A exceção a esta regra são os atributos âncoras de navegação, que serão mapeados por instâncias de *SimpleActivator*. Cada classe de atributo na tabela acima dará origem a uma componente da interface, cuja modelagem descreveremos a partir de agora.

6.1.2. Componente PersonalData

A componente reutilizável **PersonalData** agrupa os *widgets* que renderizam os atributos **name** e **photo**. No caso do atributo **name**, também desejamos informar aos usuários a identificação do atributo, além do seu valor. Tal requisito é satisfeito através da inclusão de um terceiro *widget*, **ProfNameLabel**, que não mapeia nenhum atributo navegacional, mas tem em sua propriedade *DefaultContent* o texto a ser exibido no rótulo: `"Name: "`.

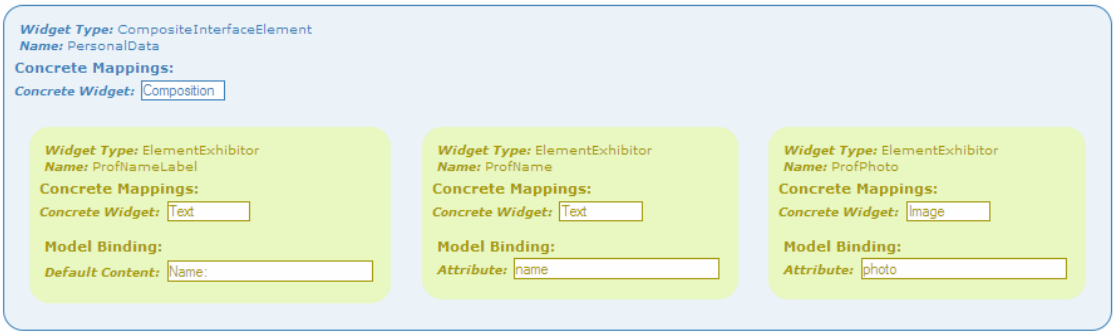

Figura 87 – Especificação abstrata da componente *PersonalData*

6.1.3. Componente AcademicData

A componente **AcademicData** exibe rótulos identificando todos os atributos que apresenta. É composta de duas instâncias de *CompositeInterfaceElement*, que separam **category** dos atributos relativos à titulação.

Figura 88 – Especificação abstrata da componente *AcademicData*

6.1.4. Componente ContactInfo

Em **ContactInto** agrupamos os pares rótulos/valor de cada atributo em diferentes elementos do tipo *Composite*. Observamos, também, a primeira ocorrência de um *SimpleActivator*, dando ao atributo **homepage** a semântica de *hyperlink*, através do mapeamento para o *widget* concreto **Link**. O *ElementExhibitor* **ProfHPText** também mapeia o atributo **homepage**, mas para exibir a URL como o texto visível da âncora. Este é a única situação em que um elemento abstrato diferente de *CompositeInterfaceElement* pode conter outros elementos[74].

[74] Na Ontologia de Descrição de Interfaces Ricas, a propriedade *hasElementExhibitor* tem apenas a classe *SimpleActivator* em seu domínio.

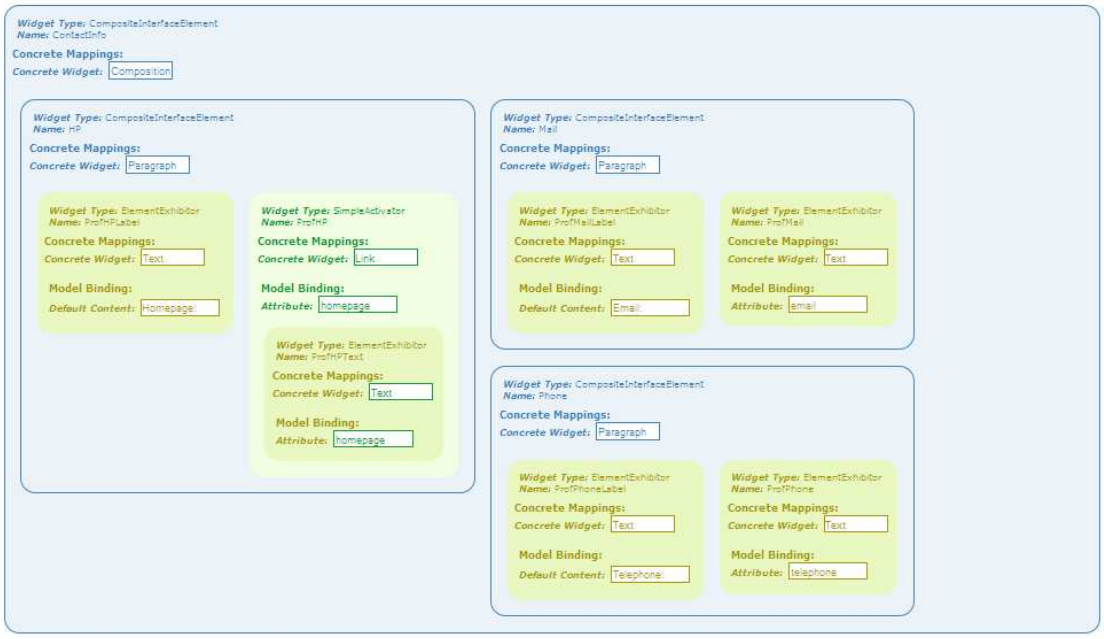

Figura 89 – Especificação abstrata da componente *ContactInfo*

6.1.5. Componente AdviseesList

Para modelar o componente de interface que irá renderizar a lista de alunos orientados pelo professor, é preciso ter em mente que o atributo **advisees** é do tipo índice. Os atributos índices são serializados no objeto JSON que representa o nó navegacional como *arrays*, e as entradas do índice, como elementos do *array*.

O primeiro mapeamento é feito entre a composição **AdviseesList** e o atributo **advisees**. Visto que um índice é uma lista, é compreensível que atributos do tipo índice sejam mapeados em composições cujos elementos são objetos do mesmo tipo. Os itens da lista, por sua vez, são representados pela composição **Advisee**, que mapeia a classe navegacional **Student**, o tipo das entradas do índice.

O próximo passo especificar quais *widgets* irão apresentar os atributos das entradas do índice. Para isto, precisamos conhecer o modelo de dados do índice **StudentByProfessor**, conteúdo do atributo **advisees** em **Professor**.

Índice	*Atributo*	*Tipo*
StudentByProfessor	name	ContextAnchorIndexAttribute

Tabela 5 – Índice *StudentByProfessor*

Conforme foi explicado no capítulo 5, quando o atributo representa uma âncora de navegação (caso do atributo **name** em **StudentByProfessor**), a URL alvo estará associada à chave de nome “`<attribute_name>|swui:target`”. Além disso, os atributos das entradas de um índice são referenciados no objeto JSON pelas chaves “`<node_attribute_name>|<index_entry_attribute_name>`”. Portanto,

precisamos renderizar os seguintes dois atributos para cada elemento da lista de orientandos: **advisees|name** e **advisees|name|swui:target**. Isto é feito através dos *widgets* **AdviseeLink** e **AdviseeName**.

O elemento **AdviseesListLabel** em **AdviseesList** tem por finalidade apenas apresentar um título para a lista: “Advisees:”.

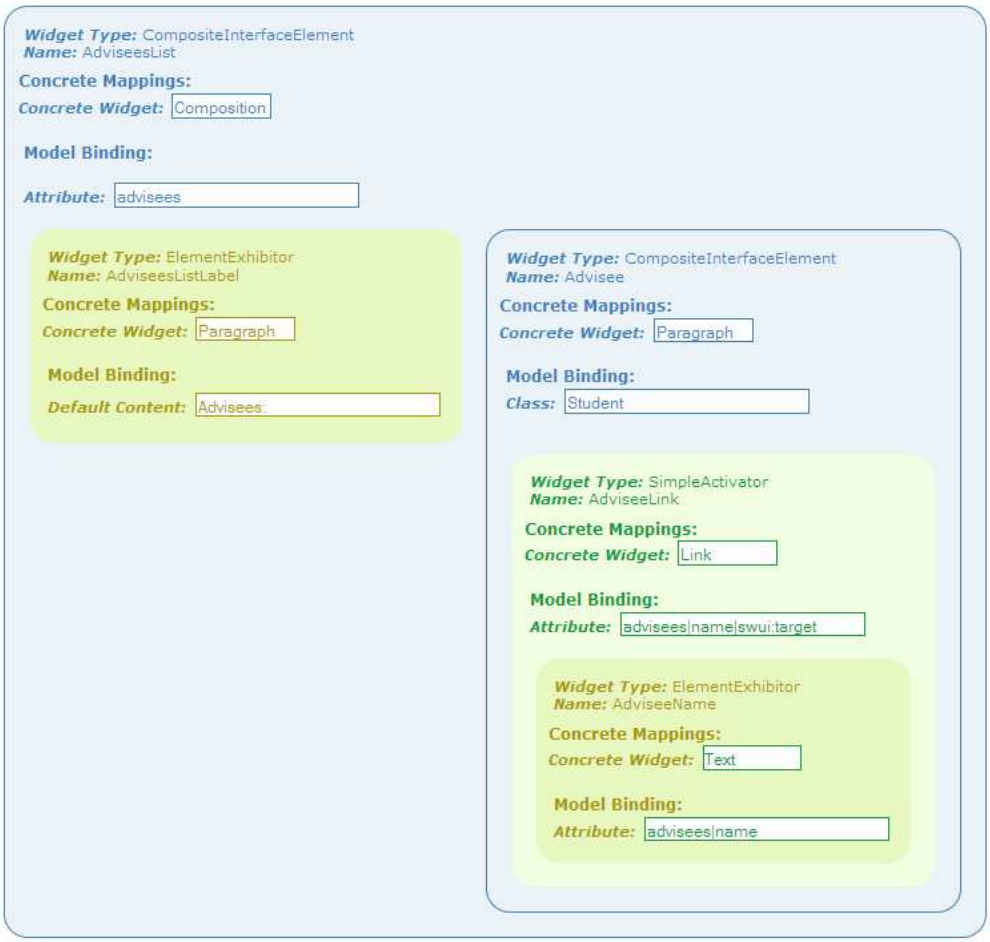

Figura 90 – Especificação abstrata da componente *AdviseesList*

Não detalharemos as demais componentes listas, *AreasList*, *FriendsList* e *PublicationsList*, visto que sua modelagem é totalmente análoga à *AdvisessList*.

Finalmente, o *widget* que representa o nó **Professor**, na especificação XML abaixo. O mapeamento entre **ProfessorNode** e a classe navegacional **Professor** é feito através da propriedade *fromClass*.

```
<CompositeInterfaceElement name="ProfessorNode" mapsTo="Composition" fromClass="Professor" >

    <%= include PersonalData %>

    <%= include AcademicData %>

    <%= include ContactInfo %>
```

```
            <%= include AdviseesList %>

            <%= include AreasList %>

            <%= include FriendsList %>

            <%= include PublicationsList %>

        </CompositeInterfaceElement>
```

Figura 91 – *Widget ProfessorNode*

Criaremos, então, **ProfessorView** como uma instância de *ContextInterface*, associada à classe **Professor**.

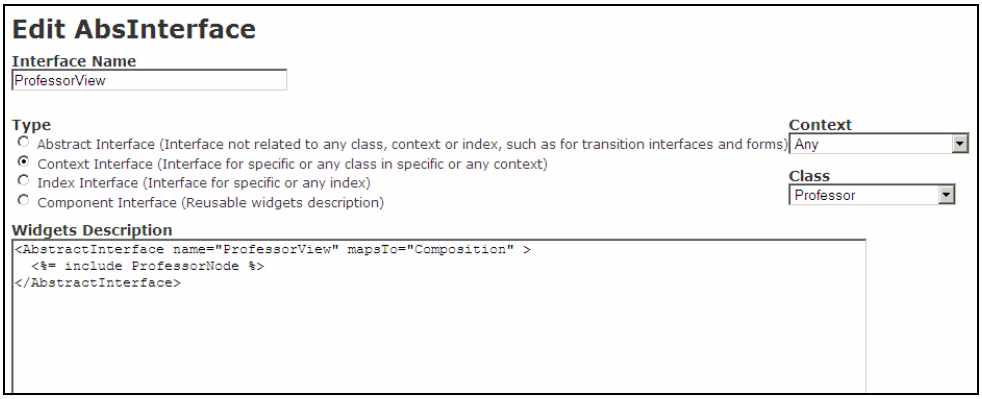

Figura 92 – Edição da interface *ProfessorView*

6.1.6. Integração com *Cascading Style Sheets*

O uso de CSS permite o controle sobre a aparência dos elementos de interface. A ligação dos *widgets* abstratos com as folhas de estilo se dá por meio da propriedade *cssClasses*.

A figura abaixo mostra como a interface **ProfessorView** sem qualquer mapeamento com a linguagem CSS seria percebida.

As seguintes classes CSS nos ajudarão a modificar a disposição visual dos elementos na interface, melhorando a comunicação das informações apresentadas:

- **label**: aplicada aos *widgets* que têm a função de exibir apenas um rótulo ou título. Aplica o estilo negrito à fonte;
- **swui_floatRight**: aplicada ao widget que exibe o atributo **photo**. Alinha a imagem à direita da tela;
- **list**: aplicada às composições que representam listas. Modifica a cor da fonte e desenha uma borda completa no elemento;
- **swui_composite**: aplicada à composição final dos atributos. Define margens e espaçamentos;
- **narrow**: aplicada à composição final dos atributos. Centraliza o elemento na tela, desenha uma borda à sua esquerda e define a sua largura como 40% da janela.

Figura 93 – Interface *ProfessorView*, sem a aplicação de CSS

As classes **swui_composite** e **swui_floatRight** estão definidas para todas as aplicações desenvolvidas no HyperDE porque fazem são internas ao Módulo de Interfaces Ricas. As demais classes foram definidas na interface concreta **MainCSS**, que, por sua vez, foi vinculada à interface **ProfessorView**. O resultado está a seguir:

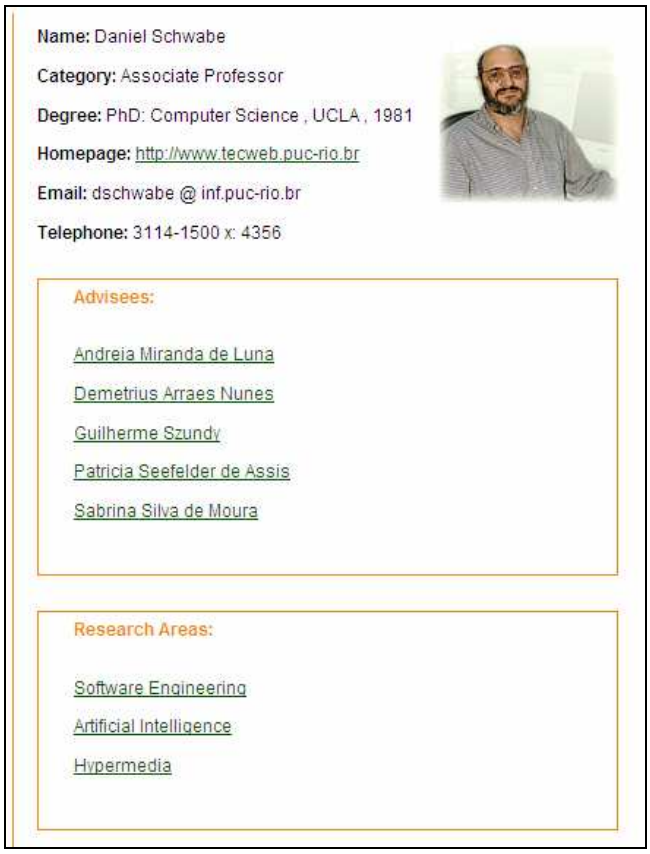

Figura 94 – Interface *ProfessorView*, com CSS aplicado

6.1.7. Interface Concreta

Na listagem abaixo, apresentamos o código concreto gerado pelo processo de compilação da descrição abstrata dos elementos de interface. O Interpretador de Interfaces Ricas, em tempo de carregamento da página, usará os elementos HTML neste código como *templates* para instanciar os *widgets* que de fato serão exibidos para o usuário.

```
      <div id='ProfessorView' typeof='shdm:AbstractInterface' class='swui_invisible'>
            <div id='ProfessorNode' typeof='Professor' class='swui_invisible
swui_composite narrow'>
                  <div id='PersonalData'>
                        <span id='ProfNameLabel' property='shdm:literal'
class='label'> Name:</span>
                        <span id='ProfName' property='name'> </span>
                        <img id='ProfPhoto' property='photo' alt='photo'
class='swui_floatRight'/>
                  </div>
                  <div id='AcademicData'>
                        <p id='AcademicDataCat'>
                              <span id='ProfCatLabel' property='shdm:literal'
class='label'> Category:</span>
                              <span id='ProfCat' property='category'> </span>
                        </p>
```

```
                        <p id='AcademicDataDeg'>
                              <span id='ProfDegreeLabel' property='shdm:literal'
class='label'> Degree:</span>
                              <span id='ProfDegree' property='degree'> </span>
                              <span id='ProfDegreeFromLabel' property='shdm:literal'>
, </span>
                              <span id='ProfDegreeFrom' property='degree_from'>
</span>
                              <span id='ProfDegreeYearLabel' property='shdm:literal'>
, </span>
                              <span id='ProfDegreeYear' property='degree_year'>
</span>
                        </p>
                  </div>
                  <div id='ContactInfo'>
                        <p id='HP'>
                              <span id='ProfHPLabel' property='shdm:literal'
class='label'> Homepage:</span>
                              <a id='ProfHP' property='homepage' onclick="javascript:
navTarget(this.href, 'ProfessorView', '');return false;">
                                    homepage<span id='ProfHPText' property='homepage'>
</span>
                              </a>
                        </p>
                        <p id='Mail'>
                              <span id='ProfMailLabel' property='shdm:literal'
class='label'> Email:</span>
                              <span id='ProfMail' property='email'> </span>
                        </p>
                        <p id='Phone'>
                              <span id='ProfPhoneLabel' property='shdm:literal'
class='label'> Telephone:</span>
                              <span id='ProfPhone' property='telephone'> </span>
                        </p>
                  </div>
                  <div id='AdviseesList' property='advisees' class='list'>
                        advisees<p id='AdviseesListLabel' property='shdm:literal'
class='label'> Advisees:</p>
                        <p id='Advisee' typeof='Student' class='swui_invisible'>
                              <a id='AdviseeLink' property='advisees|name|shdm:target'
onclick="javascript: navTarget(this.href, 'ProfessorView', '');return false;">
                                    advisees|name|shdm:target<span id='AdviseeName'
property='advisees|name'> advisees|name</span>
                              </a>
                        </p>
                  </div>
                  <div id='AreasList' property='areas' class='list'>
                        areas<p id='AreasListLabel' property='shdm:literal'
class='label'> Research Areas:</p>
                        <p id='Area' typeof='ResearchArea' class='swui_invisible'>
```

```
                              <a id='AreaLink' property='areas|name|shdm:target'
onclick="javascript: navTarget(this.href, 'ProfessorView', '');return false;">
                                    areas|name|shdm:target<span id='AreaName'
property='areas|name'> areas|name</span>
                              </a>
                        </p>
                  </div>
                  <div id='FriendsList' property='friends' class='list'>
                        friends<p id='FriendsListLabel' property='shdm:literal'
class='label'> Friends:</p>
                        <p id='Friend' typeof='Professor' class='swui_invisible'>
                              <a id='FriendLink'
property='friends|Colleagues|shdm:target' onclick="javascript: navTarget(this.href,
'ProfessorView', '');return false;">
                                    friends|Colleagues|shdm:target<span
id='FriendName' property='friends|Colleagues'> friends|Colleagues</span>
                              </a>
                        </p>
                  </div>
                  <div id='PublicationsList' property='publications' class='list'>
                        publications<p id='PublicationsListLabel'
property='shdm:literal' class='label'> Publications:</p>
                        <p id='Publication' typeof='Publication'
class='swui_invisible'>
                              <a id='PublicationLink'
property='publications|name|shdm:target' onclick="javascript: navTarget(this.href,
'ProfessorView', '');return false;">
                                    publications|name|shdm:target<span
id='PublicationTitle' property='publications|name'> publications|name</span>
                              </a>
                        </p>
                  </div>
            </div>
      </div>
```

Figura 95 – Código concreto da interface *ProfessorView*

6.1.8. Navegação Intra e Intercontextual

Nas aplicações HyperDE, as interfaces que renderizam algum nó em contexto tipicamente contém índices que permitem ao usuário navegar para outros elementos dentro do mesmo contexto. Outras estruturas de acesso, os *landmarks* da aplicação, por sua própria definição, devem estar sempre acessíveis, não importando o contexto corrente; por isso, a interface **ProfessorView** ainda carece de alguns componentes. São eles: **DefaultLandmarks** (os *landmarks* da aplicação), **DefaultNavBar** (a barra de navegação do contexto: âncoras para anterior e próximo elementos) e **DefaultContextIndex** (índice de elementos de contexto). Estas componentes também

são parte integrante do Módulo de Interfaces Ricas e estão presentes em qualquer aplicação HyperDE.[75]

A nova versão de **ProfessorNode** será então:

```
<CompositeInterfaceElement name="ProfessorNode" mapsTo="Composition" fromClass="Professor"
cssClasses="swui_composite">

    <%= include DefaultLandmarks %>

    <%= include DefaultNavBar %>

    <%= include DefaultContextIndex %>

    <CompositeInterfaceElement name="Node" mapsTo="Composition" cssClasses="swui_composite
swui_node swui_floatRight narrow">

        <%= include PersonalData %>

        <%= include AcademicData %>

        <%= include ContactInfo %>

        <%= include AdviseesList %>

        <%= include AreasList %>

        <%= include FriendsList %>

        <%= include PublicationsList %>

    </CompositeInterfaceElement>

</CompositeInterfaceElement>
```

Figura 96 – *Widget ProfessorNode*

E o resultado final:

[75] As descrições das componentes *Default** estão disponíveis no Apêndice IV.

Figura 97 – Versão final da interface *ProfessorView*

6.1.9. Pesquisa Twitter

A tela de consulta às informações relativas a uma Área de Pesquisa dá acesso também a API de busca do Twitter[76], caso o usuário deseja pesquisar os últimos *tweets* citando a área. Este caso de uso foi especialmente idealizado para demonstrar a aplicação do protocolo de fila de mensagens na comunicação entre Visão e Modelo.

O componente principal deste exemplo é a operação **searchTwitter**, acessível através da URL `/operations/searchTwitter/:id?n=<n>&t=<t>`, onde `:id` é o *placeholder* para o identificador do objeto **ResearchArea**, cujo nome será usado como termo da pesquisa no serviço Twitter. A operação **searchTwitter** realiza n chamadas a API do Twitter, com o intervalo de `t` segundos entre elas. À proporção em que `n` e `t` aumentam, também aumenta a probabilidade das novas consultas atualizarem o conjunto de *tweets* retornado pelas consultas anteriores.

Aproveitaremos o tempo de processamento potencialmente alto da operação **searchTwitter** para invocá-la sob o contexto de uma Transação, conceito introduzido no capítulo 4. Para que obtenhamos alguma vantagem em fazê-lo, a operação searchTwitter foi codificada para armazenar o resultado de cada chamada à API do Twitter na fila de respostas. Ao final do laço de consultas ao serviço externo, **searchTwitter** responderá com um comando de redirecionamento para a página de

[76] Serviço popular de *microblog*. A API de busca no Twitter está acessível em http://search.twitter.com/search.json?phrase=<phrase>

resultados da busca, hospedada no site do Twitter. O código é listado abaixo. Os trechos de manipulação da fila estão em destaque:

```
area = ResearchArea.find(params["id"])
n = params["n"]
t = params["t"]

n.times do
  # Twitter API call is encapsulated in a ResearchArea object's operation
  response = area.searchForTweets
  # Saves each response in the queue
  push_response(response)
  sleep t
end

# Finishes the transaction
push_response("EOT")

# Sends a redirect to the client's browser
query = area.name
query = query.gsub(" ","+")
url = "http://search.twitter.com/search?q=#{query}"
redirect_to_full_url(url,200)
```

Figura 98 – Código da operação *searchTwitter*

A operação **searchForTweets**, da classe **ResearchArea**, implementa as chamadas à busca do Twitter e retorna um array de resultados, ou *tweets*[77], os quais são enfileirados. Os objetos JSON que representam os *tweets* possuem os seguintes atributos:

- **swui:id**: identificador do *tweet*;
- **swui:order**: posição que o *tweet* deverá ocupar se inserido em uma lista;
- **profile_image_url**: endereço da foto do usuário autor do *tweet*;
- **profile_url**: endereço do perfil do usuário autor do *tweet*;
- **text**: texto do *tweet*.

Em **AreasView**, a interface que renderiza os nós da classe **ResearchArea**, o elemento de interface que irá ativar a operação é modelado como um *SimpleActivator* e tem, em sua propriedade *fromAction*, o endereço de chamada à operação. Para que a operação seja executada no escopo de uma transação, é necessário especificar também a propriedade *transitionInterface* do ativador.

[77] As mensagens postadas no serviço do Twitter são conhecidas como *tweets*.

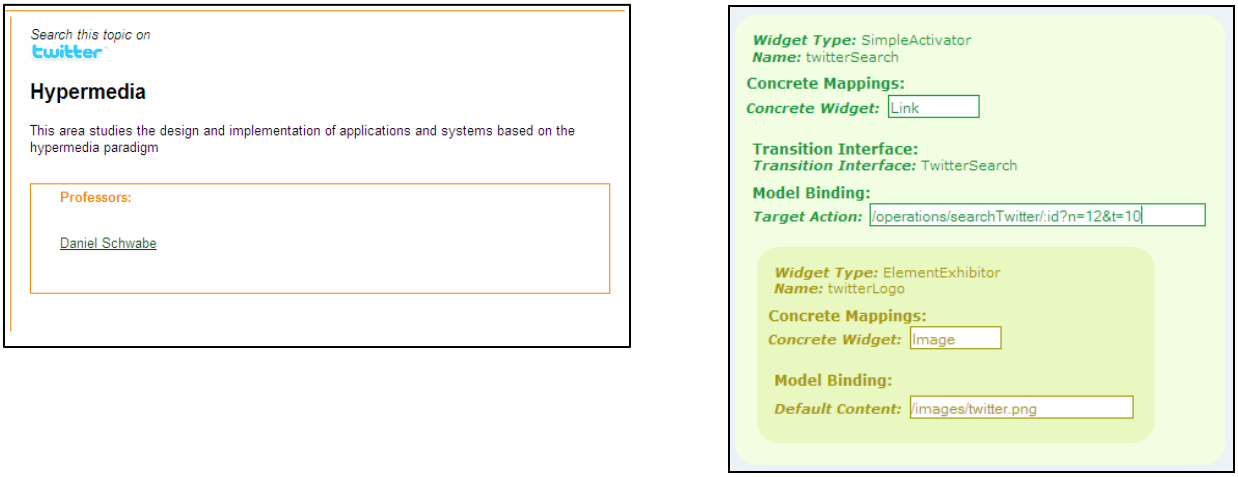

Figura 99 – Detalhe da interface *AreasView*, junto com a especificação do ativador *twitterSearch*

6.1.10. A Interface de Transição

A interface **TwitterSearch** é a interface de transição associada a operação **searchTwitter**. Visto que a interface de transição precisa estar aninhada na interface principal, o projetista incluirá em **AreasView** uma instância de *CompositeInterfaceElement* com a propriedade **loadInterface** igual a "TwitterSearch".

A interface de transição em nosso exemplo possui dois elementos principais: (a) título da página; (b) lista de resultados da pesquisa.

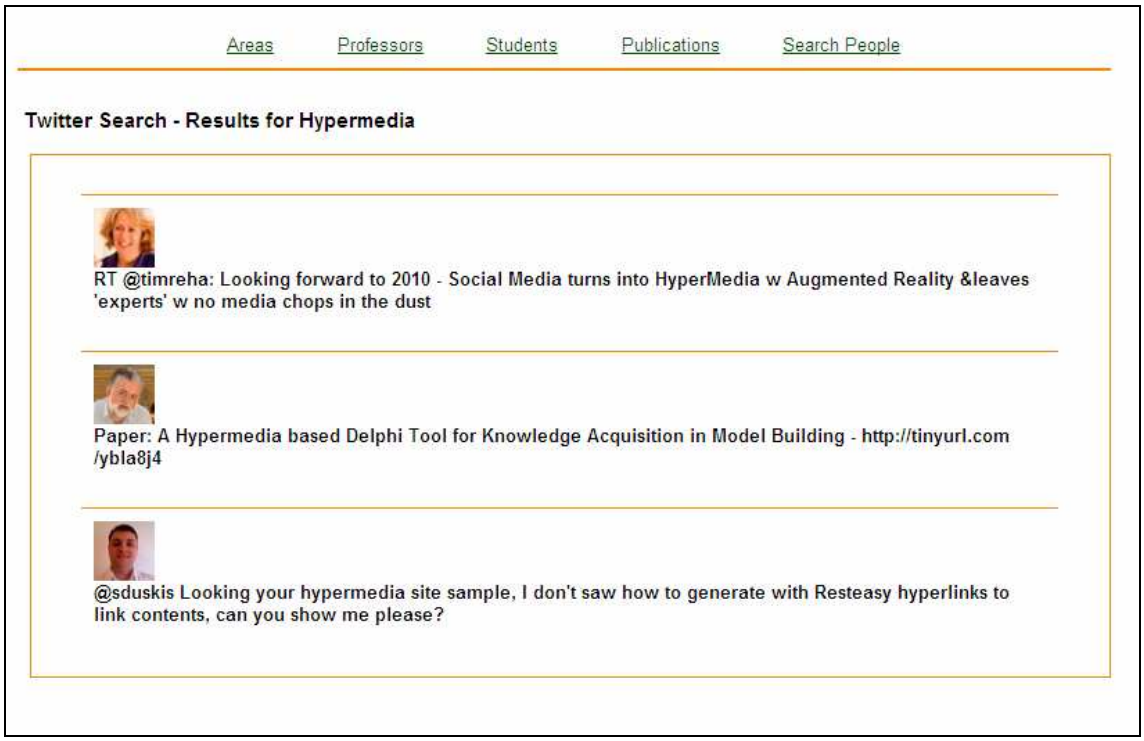

Figura 100 – Interface de transição *TwitterSearch*

Visto que o título da página é função da área de pesquisa consultada, a especificação deste componente da interface não pode ser determinada em tempo de projeto. Portanto, código Ruby foi embutido na descrição dos *widgets* de TwitterSearch, para gerar em tempo de execução um elemento do tipo *ElementExhibitor*, cuja propriedade *defaultContent* contém o nome da área de pesquisa.

```
      <%
      name = jsonData["name"]
      buffer << "<ElementExhibitor name=\"title\" mapsTo=\"Header3\"
defaultContent=\"Twitter Search - Results for #{name}\" cssClasses=\"\"
tagAttributes=\"\"/>"
      %>
```

Figura 101 – Código Ruby embutido na especificação da interface *TwitterSearch*

A lista de resultados é especificada como segue:

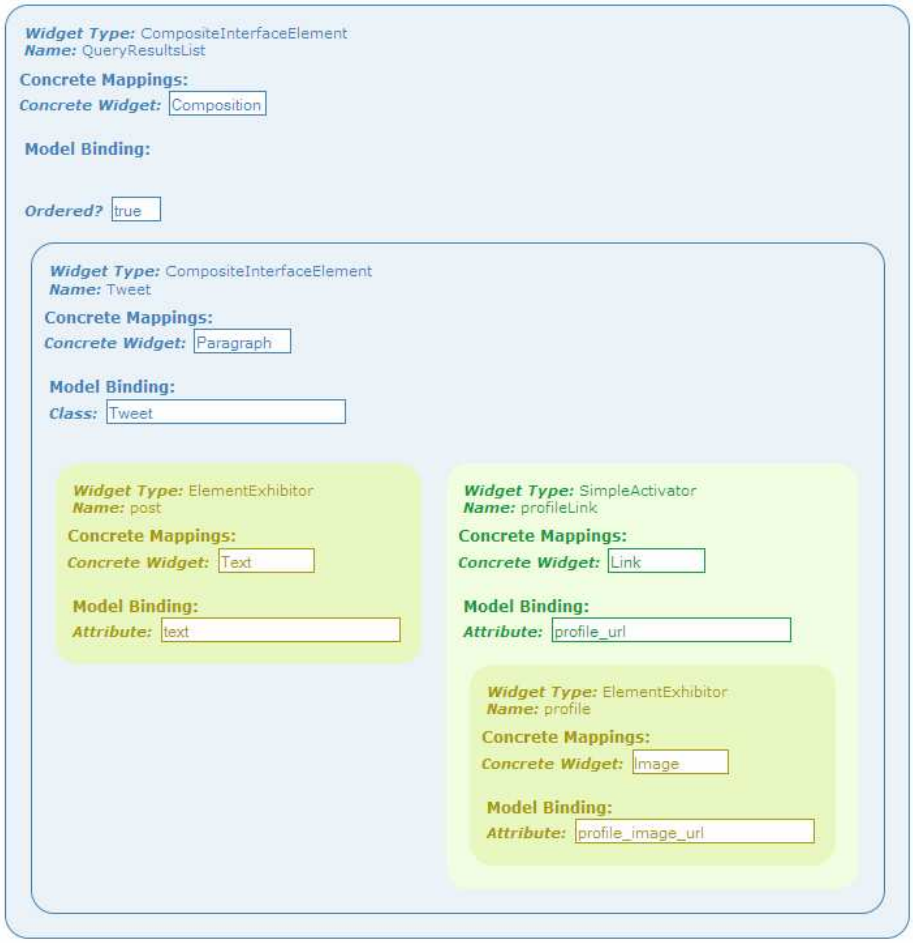

Figura 102 – Especificação abstrata do *widget QueryResultsList*

É desejável que o Modelo controle a ordenação dos elementos na lista, posicionando os *tweets* mais recentes sempre no topo. Tornamos a composição **QueryResultsList** sensível à ordenação definida pelo Modelo através da propriedade **ordered** igual a `true`.

Os itens da lista são representados pela composição **Tweet**, que mapeia a classe de mesmo nome. O *widget* **post** renderiza o texto do *tweet*, enquanto que **profileLink** e **profile** fazem da foto do autor uma âncora para o seu perfil no site do Twitter.

6.1.11. Retórica em Interfaces

Como conclusão, ilustraremos o uso do modelo de retórica, por meio de seqüências de animações que irão modificar a composição da interface principal quando a operação **searchTwitter** for iniciada e, em seguida, realçar a chegada de novos elementos na interface de transição **TwitterSearch**.

Nossos objetivos são: (a) ocultar os dados da área de pesquisa antes de exibir a lista de resultados da busca; (b) animar a inserção de elementos na lista de resultados da busca. A cada um destes objetivos, corresponderá uma diferente estrutura retórica.

A primeira estrutura retórica, chamada **HideContextData**, executa uma decoração do tipo saída (*RemoveElement*) para remover (ocultar) a composição de *widgets* que descrevem o nó visualizado. Em **AreasView**, esta composição é representada pelo *CompositeInterfaceElement* de nome **AreaData**. Na listagem abaixo, a especificação da estrutura retórica:

```
rhetoricalStructure "name"=>"HideContextData", "hasDecorations"=>"decSeq01"

decorationSeq "name"=>"decSeq01", "firstOfSeq"=>"HideContextDataAnim"

decoration "name"=>"HideContextDataAnim", "type"=>"RemoveElement", "hasElement"=>"AreaData",
"effect"=>"blindUp", "parameters"=>"duration:  1.0"
```

Figura 103 – Especificação da estrutura retórica *HideContextData*

A remoção do elemento **AreasData** será animada com o efeito concreto **blindUp** (*wrapper* para o efeito Script.aculo.us de mesmo nome).

A próxima estrutura retórica, **DisplayTweet**, executa duas decorações em seqüência: **DisplayTweetListAnim** e **DisplayTweetAnim**, ambas decorações de entrada sobre o elemento *Tweet* da interface **TwitterSearch**. A primeira executa o efeito concreto **showList**, o qual anima a exibição da lista a que o elemento pertence, usando o efeito Appear da bilbioteca Script.aculo.us. Em seguida, a decoração **DisplayTweetAnim** aplica o efeito **blindDown**[78] ao elemento.

```
rhetoricalStructure "name"=>"DisplayTweet", "hasDecorations"=>"decSeq02"
```

[78] *Wrapper* para o efeito Script.aculo.us de mesmo nome.

```
        decorationSeq "name"=>"decSeq02", "firstOfSeq"=>"DisplayTweetListAnim",
"lastOfSeq"=>"DisplayTweetAnim"

        decoration "name"=>"DisplayTweetListAnim", "type"=>"InsertElement", "hasElement"=>"Tweet" ,
"effect"=>"showList", "parameters"=>"duration:  1.0"

        decoration "name"=>"DisplayTweetAnim", "type"=>"InsertElement", "hasElement"=>"Tweet" ,
"effect"=>"blindDown", "parameters"=>"duration:  1.0"
```

Figura 104 – Especificação da estrutura retórica *DisplayTweet*

O próximo passo é determinar os eventos que irão disparar a execução das estruturas retóricas. O primeiro deles é o evento de clique do mouse sobre o ativador da operação. Assim que a operação for acionada, os dados relativos ao contexto de navegação deverão ser removidos da tela, para ceder lugar aos resultados da busca no Twitter. Assim, a descrição de retórica da interface **AreasView** seria:

```
        event "name"=>"TwitterLogoClick", "concreteEvent"=>"click", "onInterface"=>"AreasView",
"onElement"=>"twitterLogo", "hasRhetoricalStructures"=>"HideContextData"

        rhetoricalStructure "name"=>"HideContextData", "hasDecorations"=>"decSeq01"

        decorationSeq "name"=>"decSeq01", "firstOfSeq"=>"HideContextDataAnim"

        decoration "name"=>"HideContextDataAnim", "type"=>"RemoveElement", "hasElement"=>"AreaData",
"effect"=>"blindUp", "parameters"=>"duration:  1.0"
```

Figura 105 – Especificação do evento *TwitterLogoClick*

Finalmente, a estrutura retórica **DisplayTweet** deve ser executada sempre que ocorrer a entrada de elementos em **TwitterSearch**. Por definição, o evento de inserção de *widgets* em uma interface ocorre durante a transição na qual a interface de referência é o estado de destino. Por este motivo, a segunda estrutura retórica faz parte da descrição de retórica da interface **TwitterSearch**.

```
        transition "name"=>"TwitterSearch", "toInterface"=>"TwitterSearch", "hasRhetoricalStructures"=>"
DisplayTweet "

        rhetoricalStructure "name"=>"DisplayTweet", "hasDecorations"=>"decSeq02"

        decorationSeq "name"=>"decSeq02", "firstOfSeq"=>"DisplayTweetListAnim",
"lastOfSeq"=>"DisplayTweetAnim"

        decoration "name"=>"DisplayTweetListAnim", "type"=>"InsertElement", "hasElement"=>"Tweet" ,
"effect"=>"showList", "parameters"=>"duration:  1.0"
```

```
decoration "name"=>"DisplayTweetAnim", "type"=>"InsertElement", "hasElement"=>"Tweet" ,
"effect"=>"blindDown", "parameters"=>"duration:  1.0"
```

Figura 106 – Especificação da transição *TwitterSearch*