# 5
# Especificação e Execução de Interfaces RIA no HyperDE

Para acompanhar a especificação da Arquitetura SWUI de Interfaces Ricas, descrita no capítulo anterior, desenvolvemos também uma implementação de referência, integrada ao ambiente de modelagem e execução de aplicações hipermídia conhecido como HyperDE. A escolha do HyperDE como plataforma de implementação da Arquitetura SWUI justifica-se pelo fato de que ele foi desenvolvido para dar suporte ao projeto de aplicações com base no método SHDM. Isto significa que o HyperDE é capaz de entender as primitivas do método SHDM, tais como: classes navegacionais, contextos e índices e, a partir desses modelos, gerar uma aplicação totalmente funcional. No entanto, ainda faltava ao HyperDE um módulo tal que o projetista pudesse especificar abstratamente as interfaces da aplicação e confiar ao ambiente a geração das interfaces executáveis.

O presente capítulo documenta como o HyperDE foi estendido com as funcionalidades de interpretação de modelos abstratos de interfaces e geração automática de código[47]. Em sua versão anterior, as interfaces no HyperDE eram chamadas de *Views*, e o projetista precisava codificá-las na linguagem de *templates* RHTML (código Ruby embutido em documentos HTML). Contudo, substituímos a tela de edição de *Views* pelo que chamaremos de **Módulo de Interfaces Ricas**, compreendendo:

a) Editor de instâncias dos conceitos da Ontologia de Descrição de Interfaces Ricas;
b) Gerador de Interfaces Ricas;
c) Renderizador de Interfaces Ricas;
d) Interpretador de Interfaces Ricas;
e) Gerenciador de Transações.

## 5.1.
## Modelo de Dados

O diagrama de classes a seguir retrata o modelo de dados subjacente ao Módulo de Interfaces Ricas do HyperDE.

[47] As alterações realizadas no ambiente HyperDE foram testadas sobre a seguinte plataforma: linguagem Ruby 1.8.6, interpretador JRuby 1.3.1 e *framework* Ruby on Rails 2.3.4.

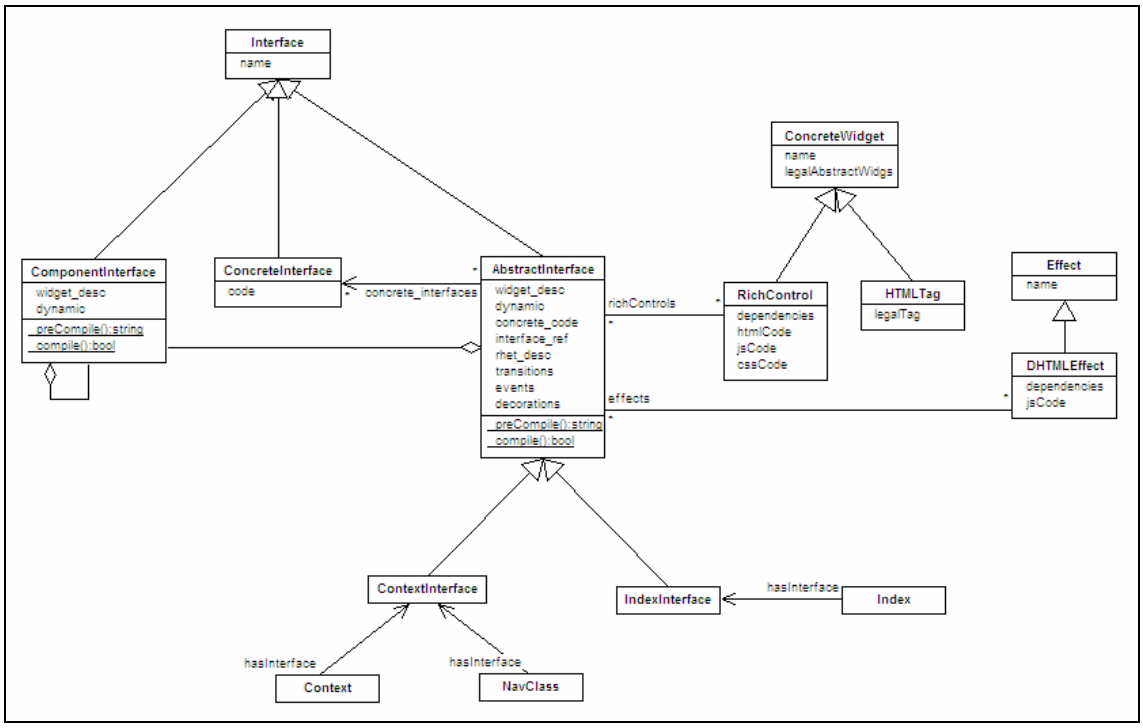



Figura 39 – Modelo de dados do Módulo de Interfaces Ricas do HyperDE

As instâncias das classes *HTMLTag* e *DHTMLEffect* correspondem às instâncias dos conceitos de mesmo nome na Ontologia de Descrição de Interfaces Ricas. As classes *ConcreteWidget, RichControl* e *Effect*, por sua vez, correspondem aos conceitos *ConcreteInterfaceElement*, *DHTMLRichControl* e *ConcreteEffect*, da mesma ontologia.

*Interface* é uma generalização das seguintes subclasses: *AbstractInterface*, *ComponentInterface* e *ConcreteInterface*. A classe *AbstractInterface* que pode ser especializada em *ContextInterface e IndexInterface,* é a abstração para as interfaces de uma aplicação HyperDE. Seus atributos **widget_desc** e **rhet_desc** armazenam, respectivamente, a descrição de instâncias dos conceitos (a) *AbstractInterface* e *AbstractInterfaceElement* e (b) *Transition*, *RhetoricalStructure*, *Decoration* e *Event*, da Ontologia de Descrição de Interfaces Ricas.

As demais classes, embora não representem conceitos da ontologia do capítulo 3, ou (a) fazem parte do metamodelo do HyperDE, como é o caso das classes *Context*, *Index* e *NavClass*, que representam os conceitos de **contexto de navegação**, **índices** e **classes navegacionais**, do método SHDM, ou (b) agregam novos recursos ao ambiente de modelagem, por exemplo, criação de componentes reutilizáveis (*ComponentInterface*) e a declaração de folhas de estilo CSS (*ConcreteInterface*).

## 5.2. Ambiente de Modelagem

O ambiente de modelagem do Módulo de Interfaces Ricas, acessível pela aba *Interfaces* do HyperDE, é composto por quatro módulos:

a) Editor de instâncias das classes *AbstractInterface* e *ComponentInterface* (link *Abstract Interfaces*);
b) Editor de instâncias da classe *ConcreteInterface* (link *CSS Style Sheets*)
c) Editor de instâncias da classe *ConcreteWidget* (link *Concrete Widgets*);
d) Editor de instâncias da classe *DHTMLEffect* (link *Effects*)

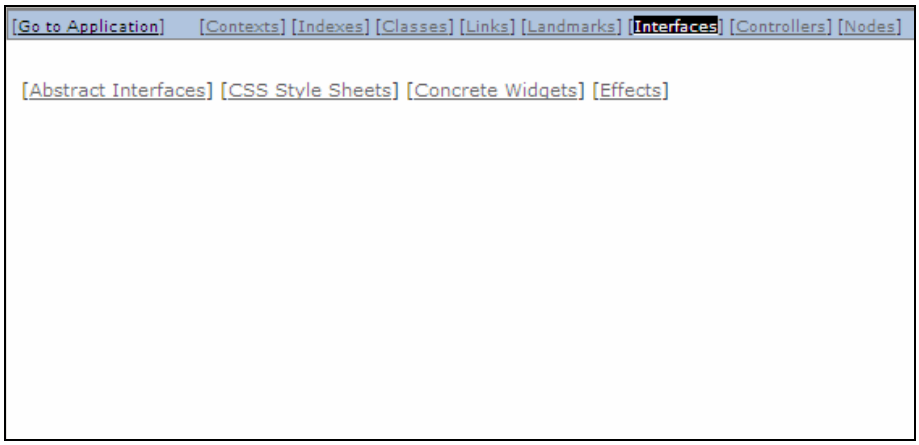



Figura 40 – Menu do Módulo de Interfaces Ricas

### 5.2.1. Especificação de Interfaces Abstratas

A classe *AbstractInterface* descreve a composição e o comportamento de uma interface RIA. No HyperDE, ela pode ser especializada como uma *ContextInterface* ou uma *IndexInterface*. Ao definir uma *ContextInterface*, o desenvolvedor poderá relacioná-la a um contexto e/ou classe navegacional específicos. Semelhantemente, uma *IndexInterface* pode ser associada a um índice navegacional. A semântica desses relacionamentos pode ser descrita da seguinte maneira:

a) Quando o usuário requisita os dados de um determinado contexto sem especificar a interface a ser exibida:
   - Se houver interface associada ao contexto, ela será exibida;
   - Se houver interface associada à classe navegacional, independentemente do contexto, ela será exibida.
b) Quando o usuário requisita os dados de um determinado índice sem especificar a interface a ser exibida:
   - Se houver interface associada ao índice, ela será exibida.

Uma *ContextInterface* também pode ser relacionada a “qualquer” contexto. Da mesma forma, uma *IndexInterface*, a “qualquer” índice. Um relacionamento deste tipo indica que a tal interface será a escolha *default* para exibir os dados de um contexto ou de um índice, caso não exista uma outra interface associada especificamente com eles.

As interfaces **DefaultContextInterface** e **DefaultIndexInterface** são fornecidas como componentes do Módulo de Interfaces Ricas. Como sugerido pelos seus próprios nomes, a primeira é uma *ContextInterface* relacionada a “qualquer” contexto e a “qualquer” classe navegacional. A última é uma *IndexInterface* relacionada a “qualquer” índice. Essas duas interfaces servem como visualizações *default* para todos os contextos e índices navegacionais de uma aplicação hipermídia. As interfaces *DefaultContextInterface* e *DefaultIndexInterface* compõem um modelo de interfaces disponível para qualquer aplicação desenvolvida no ambiente HyperDE. Desta forma, a aplicação já estará funcional assim que seu modelo navegacional tenha sido definido.

As instâncias da classe genérica *AbstractInterface*, entretanto, não estão relacionadas a nenhuma classe, contexto ou índice. Interfaces deste tipo não renderizam necessariamente nós em contexto ou estruturas de acessos. Este é o caso de formulários de entrada de dados e interfaces de transição, por exemplo.

A figura a seguir mostra a tela de edição de interfaces abstratas.

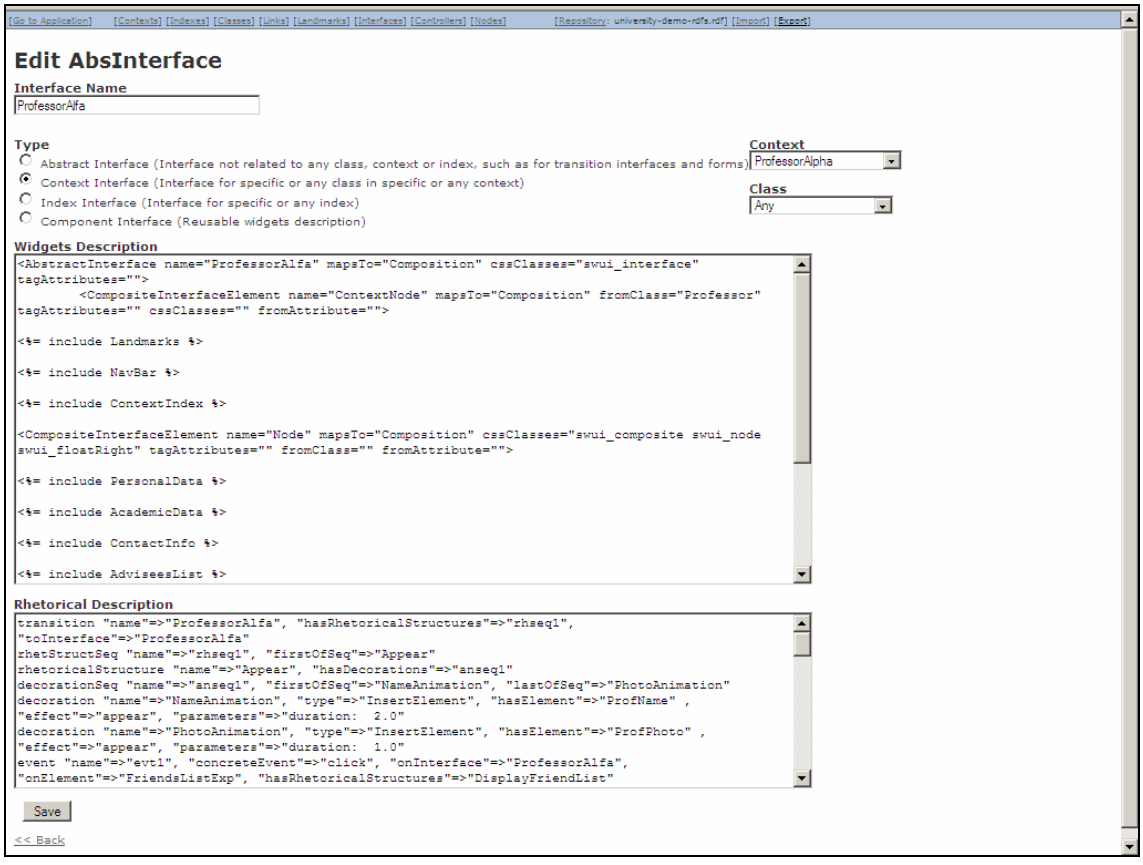


Figura 41 – Tela de edição de interfaces abstratas

Os campos ***Interface Name***, ***Widgets Decription*** e ***Rhetorical Decription*** correspondem aos seguintes atributos da classe *AbstractInterface*:

- **name**: nome da interface;
- **widget_desc**: a descrição dos *widgets* que compõem a interface;
- **rhet_desc**: a descrição de retórica da interface;

A mesma tela permite também a criação de instâncias da classe *ComponentInterface*, que são descrições reutilizáveis de *widgets* abstratos. As instâncias dessa classe possuem apenas os atributos **name** e **widget_desc**; portanto, quando o projetista seleciona a opção “Component Interface” no campo ***Type*****,** a tela de edição não possui o campo ***Rhetorical Decription***.

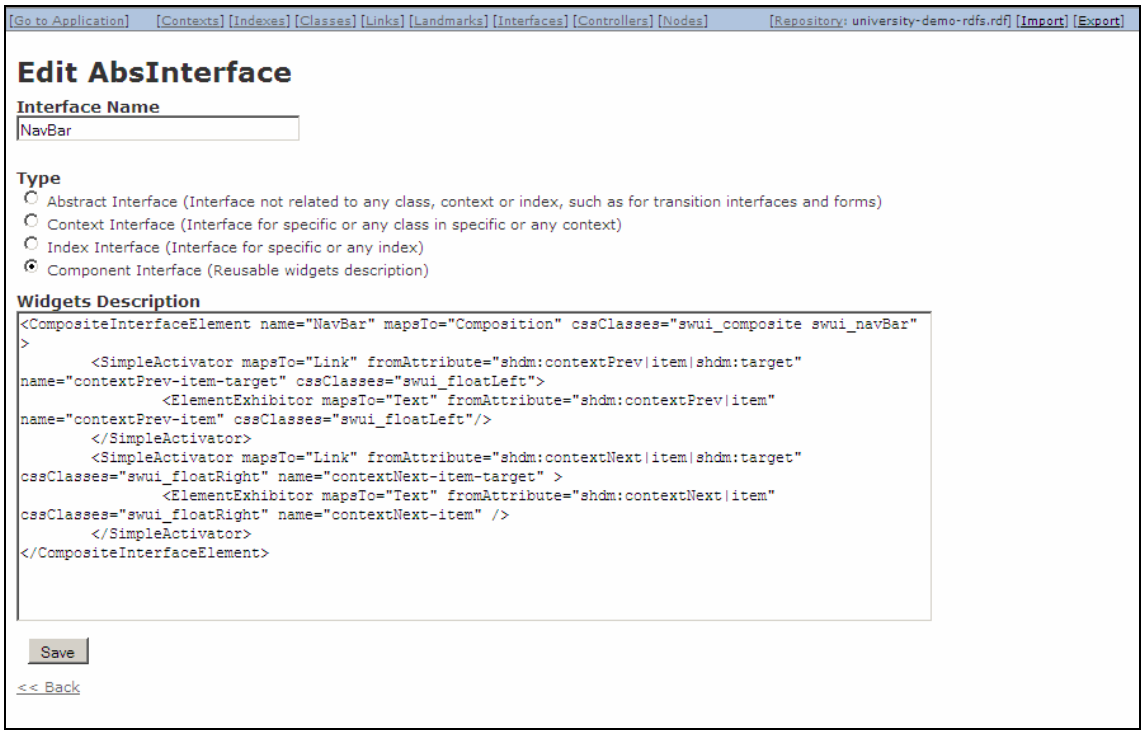



Figura 42 – Edição de componentes de interface

A Ontologia de Descrição de Interfaces Ricas fornece o vocabulário usado na especificação dos campos *Widgets Decription* e *Rhetorical Decription*.Todavia, diferentes sintaxes são empregadas em cada caso.

A descrição da estrutura da interface abstrata segue uma sintaxe XML, onde o nome da classe ou tipo do *widget* é o nome da *tag* XML e as propriedades do *widget* aparecem como atributos da *tag*. A exceção a esta regra são as propriedades *hasInterfaceElements* e *hasElementExhibitor* da ontologia, que especificam a composição de *widgets*. Na descrição XML da interface, esta composição é indicada simplesmente pelo aninhamento dos elementos XML que representam os *widgets*.

A seguir, ilustramos o uso da sintaxe XML para representar uma instância do conceito *AbstractInterface*. Consideremos novamente a interface "ProfessorView". Primeiramente, a representação RDF (modelo triplas sujeito-predicado-objeto) da especificação de *widgets*, usando o vocabulário da Ontologia de Descrição de Interfaces Ricas e a sintaxe N3:

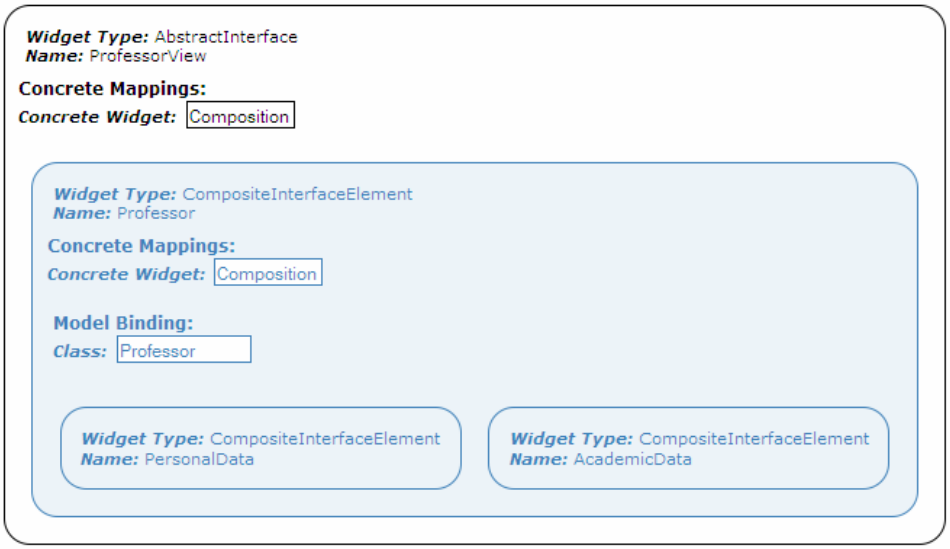



Figura 43 – Especificação abstrata da interface *ProfessorView* (Representação gráfica)

**Notation 3 Form**

```
@prefix swui: <http://swui.awardspace.com/rdf/swui.rdfs#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

swui:instance1 rdf:type swui:AbstractInterface ;
               swui:name "ProfessorView" ;
               swui:mapsTo "Composition" ;
               swui:hasInterfaceElement swui:instance2 .

swui:instance2 rdf:type swui:CompositeInterfaceElement;
               swui:name "Professor" ;
               swui:mapsTo "Composition" ;
               swui:fromClass "Professor" ;
               swui:hasInterfaceElement swui:instance3 ;
               swui:hasInterfaceElement swui:instance4 .
```

Figura 44 – Especificação abstrata da interface *ProfessorView* (Notação N3)

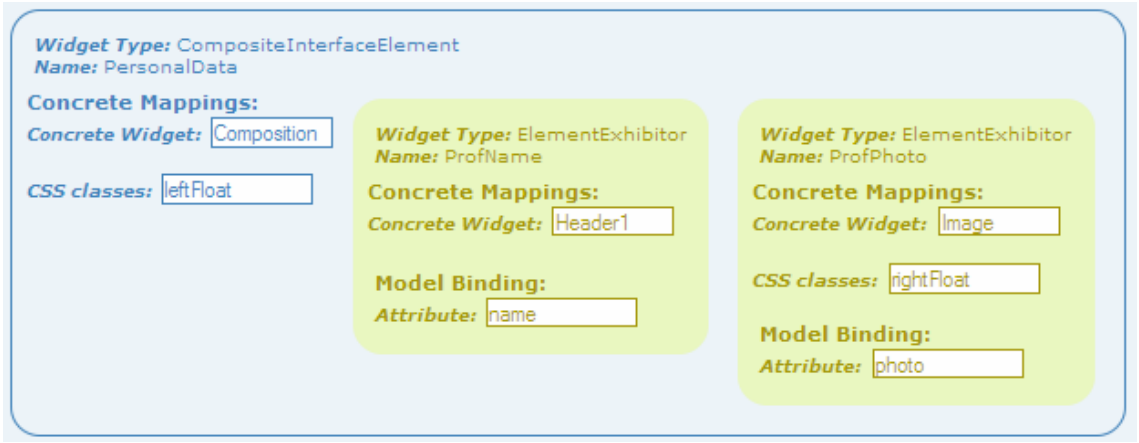



Figura 45 – Especificação abstrata da componente *PersonalData* (Representação gráfica)

**Notation 3 Form**

```
swui:instance3 rdf:type swui:CompositeInterfaceElement ;
               swui:name "PersonalData" ;
               swui:mapsTo "Composition" ;
               swui:cssClasses "leftFloat" ;
               swui:hasInterfaceElement swui:instance5 ;
               swui:hasInterfaceElement swui:instance6 .

swui:instance5 rdf:type swui:ElementExhibitor ;
               swui:name "ProfName" ;
               swui:mapsTo "Header1" ;
               swui:fromAttribute "name" .

swui:instance6 rdf:type swui:ElementExhibitor ;
               swui:name "ProfPhoto" ;
               swui:mapsTo "Image" ;
               swui:cssClasses "rightFloat" ;
               swui:fromAttribute "photo" .
```

Figura 46 – Especificação abstrata da componente *PersonalData* (Notação N3)


Figura 47 – Especificação abstrata da componente *AcademicData* (Representação gráfica)

**Notation 3 Form**

```
swui:instance4 rdf:type swui:CompositeInterfaceElement ;
               swui:name "AcademicData" ;
               swui:mapsTo "Composition" ;
               swui:cssClasses "rightColumn" ;
               swui:hasInterfaceElement swui:instance7 ;
               swui:hasInterfaceElement swui:instance8 .

swui:instance7 rdf:type swui:ElementExhibitor ;
               swui:name "ProfCategory" ;
               swui:mapsTo "Header3" ;
               swui:fromAttribute "category" .

swui:instance8 rdf:type swui:CompositeInterfaceElement ;
               swui:name "DegreeData" ;
               swui:mapsTo "Paragraph" ;
               swui:hasInterfaceElement swui:instance9 ;
```

```
              swui:hasInterfaceElement swui:instance10 ;
              swui:hasInterfaceElement swui:instance11 ;
              swui:hasInterfaceElement swui:instance12 .

swui:instance9 rdf:type swui:ElementExhibitor ;
              swui:name "DegreeDataLabel" ;
              swui:mapsTo "Label" ;
              swui:defaultContent "Degree:" .

swui:instance10 rdf:type swui:ElementExhibitor ;
               swui:name "ProfDegree" ;
               swui:mapsTo "Paragraph" ;
               swui:fromAttribute "degree" .

swui:instance11 rdf:type swui:ElementExhibitor ;
               swui:name "ProfDegreeFrom" ;
               swui:mapsTo "Paragraph" ;
               swui:fromAttribute "degree_from" .

swui:instance12 rdf:type swui:ElementExhibitor ;
               swui:name "ProfDegreeYear" ;
               swui:mapsTo "Paragraph" ;
               swui:fromAttribute "degree_year" .
```

Figura 48 – Especificação abstrata da componente *AcademicData* (Notação N3)

Finalmente, a mesma especificação usando a sintaxe XML proposta para representar as instâncias de *AbstractInterface*[48], bem mais concisa:

```
<AbstractInterface name="ProfessorView" mapsTo="Composition">
  <CompositeInterfaceElement name="Professor" mapsTo="Composition" fromClass="Professor"
cssClasses="node">
    <CompositeInterfaceElement name="PersonalData" mapsTo="Composition"
cssClasses="leftFloat">
      <ElementExhibitor name="ProfName" mapsTo="Header1" fromAttribute="name"/>
      <ElementExhibitor name="ProfPhoto" mapsTo="Image" fromAttribute="photo"
cssClasses="rightFloat"/>
    </CompositeInterfaceElement>
    <CompositeInterfaceElement name="AcademicData" mapsTo="Composition">
      <ElementExhibitor name="ProfCategory" mapsTo="Header3" fromAttribute="category"
cssClasses="rightColumn" />
      <CompositeInterfaceElement name="DegreeData" mapsTo="Paragraph">
        <ElementExhibitor name="DegreeDataLabel" mapsTo="Label"
defaultContent="Degree:"/>
        <ElementExhibitor name="ProfDegree" mapsTo="Paragraph" fromAttribute="degree"/>
        <ElementExhibitor name="ProfDegreeFrom" mapsTo="Paragraph"
fromAttribute="degree_from"/>
        <ElementExhibitor name="ProfDegreeYear" mapsTo="Paragraph"
fromAttribute="degree_year"/>
      </CompositeInterfaceElement>
    </CompositeInterfaceElement>
  </CompositeInterfaceElement>
</AbstractInterface>
```

Figura 49 – Especificação abstrata da interface *ProfessorView* (sintaxe XML)

[48] O arquivo `<HyperDE_root>/lib/ai_data/swui.dtd`, disponível no Apêndice II, contém a gramática da linguagem, utilizada para a validação das descrições XML.

Para reutilizar a especificação de uma componente no campo ***Widget Description*** de uma interface abstrata, o projetista empregará a diretiva <%= `include [component_interface_name]` %>. A pré-compilação irá incluir na descrição XML da interface abstrata a especificação das componentes referenciadas pelas diretivas `include`. Para ilustrar a inclusão de componentes, considere o seguinte trecho de código na descrição de uma interface abstrata:

```
<CompositeInterfaceElement name="Menu">
  <%= include Landmarks %>
</CompositeInterfaceElement>
```

Figura 50 – Trecho de código exemplificando o uso da diretiva `include`

A especificação XML correspondente, após a resolução das diretivas `include`, será:

```
<CompositeInterfaceElement name="Menu">
  <CompositeInterfaceElement name="Landmarks">
      <CompositeInterfaceElement name="Landmark">
          <SimpleActivator name="landmark-target">
              <ElementExhibitor name="landmark-label"/>
          </SimpleActivator>
      </CompositeInterfaceElement>
  </CompositeInterfaceElement>
</CompositeInterfaceElement>
```

Figura 51 – Código correspondente após a resolução da diretiva `include`

O trecho em destaque na especificação acima é o conteúdo do atributo **widget_desc** da instância de *ComponentInterface* identificada pelo nome “Landmarks”. O código XML acima servirá como fonte na compilação da interface concreta.

### 5.2.2. Descrição Retórica

Embora a linguagem XML tenha se mostrado uma escolha adequada para representar as descrições dos *widgets*, optamos por desenvolver uma DSL Ruby (*Domain Specific Language*), para representar as descrições retóricas. A sintaxe desta linguagem descreve: (a) o tipo de objeto (*transition*, *event* ou *decoration*, por exemplo); (b) as propriedades do objeto em uma lista de pares chave-valor.

Para ilustrar a especificação retórica e a sintaxe DSL usada no HyperDE, consideremos uma segunda versão para a interface “ProfessorView”, que exibe as instâncias da classe **Professor**.

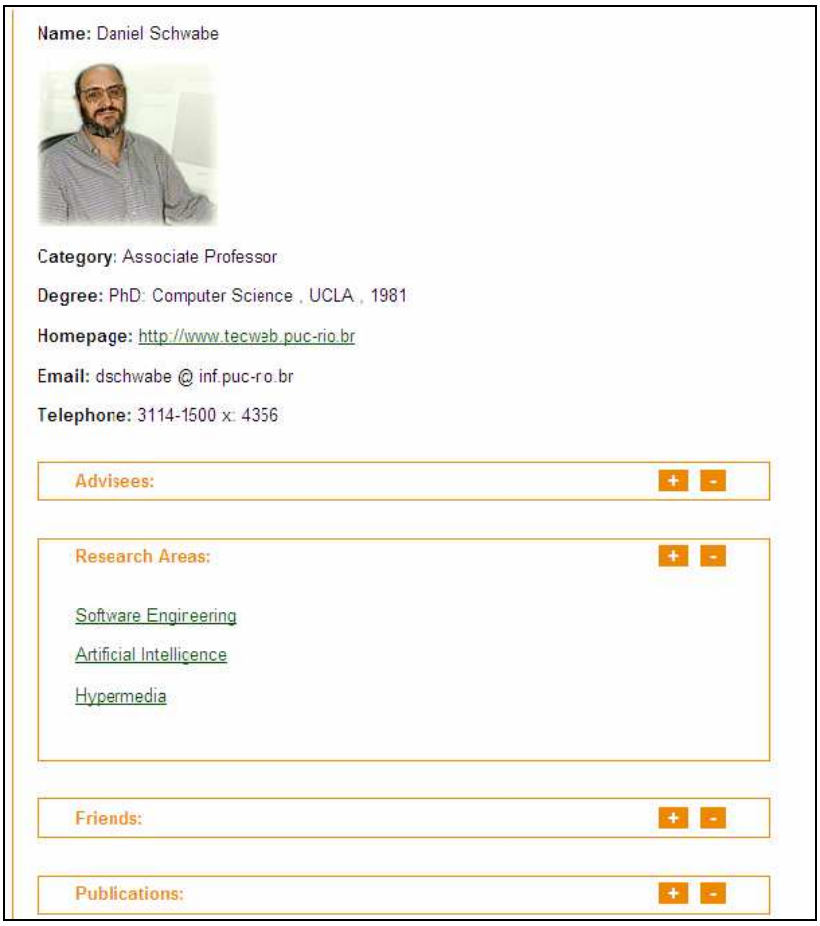



Figura 52 – Segunda versão da interface *ProfessorView*

Definimos o seguinte comportamento para a interface:

a) Os campos nome e fotografia do professor devem ser exibidos, nesta seqüência, com uma animação do tipo *fade-in*;

b) Os sinais "+" e "–" nos cabeçalhos das listas **Advisees**, **Research Areas**, **Friends** e **Publications** são sensíveis ao clique do *mouse* e expandem ou contraem as listas.

O código DSL abaixo declara a seqüência em que os campos nome (**ProfName**) e fotografia (**ProfPhoto**) serão animados na entrada da interface.

```
transition "name"=>"InterfaceEntry", "hasRhetoricalStructures"=>"rhseq1",
"toInterface"=>"ProfessorView"

rhetStructSeq "name"=>"rhseq1", "firstOfSeq"=>"PersonalData"

rhetoricalStructure "name"=>" PersonalData", "hasDecorations"=>"anseq1"

decorationSeq "name"=>"anseq1", "firstOfSeq"=>"NameAnimation", "lastOfSeq"=>"PhotoAnimation"

decoration "name"=>"NameAnimation", "type"=>"InsertElement", "hasElement"=>"ProfName" ,
"effect"=>"appear", "parameters"=>"duration: 2.0"
```

```
decoration "name"=>"PhotoAnimation", "type"=>"InsertElement", "hasElement"=>"ProfPhoto" ,
"effect"=>"appear", "parameters"=>"duration: 1.0"
```

Figura 53 – Especificação da retórica da interface (código DSL)

A primeira linha define uma instância do conceito *Transition*, que será executada sempre que o estado final da transição for a interface de visualização dos professores (**ProfessorView**). A representação RDF desta instância é mostrada abaixo:

**Notation 3 Form**

```
@prefix swui: <http://swui.awardspace.com/rdf/swui.rdfs#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

swui:instance1 rdf:type swui:Transition ;
               swui:name "InterfaceEntry" ;
               swui:toInterface "ProfessorView" ;
               swui:hasRhetoricalStructures swui:instance2 .
```

Figura 54 – Especificação de uma transição (Notação N3)

Conforme explicado no capítulo 3, o conceito *RhetStructSeq*, da Ontologia de Descrição de Interfaces Ricas, representa a seqüência em que as estruturas retóricas ocorrem durante uma transição. As estruturas retóricas (*RhetoricalStructure*), por sua vez, agrupam semanticamente as decorações nos elementos da interface. A segunda linha do código DSL define uma instância de *RhetStructSeq*, que contém apenas uma estrutura retórica, definida na linha seguinte. A representação RDF destas instâncias é mostrada abaixo:

**Notation 3 Form**

```
swui:instance2 rdf:type swui:RhetStructSeq ;
               swui:name "rhseq1" ;
               swui:firstOfSeq swui:instance3 .

swui:instance3 rdf:type swui:RhetoricalStructure ;
               swui:name "PersonalData" ;
               swui:hasDecorations swui:instance4 .
```

Figura 55 – Especificação de uma seqüência de estruturas retóricas (Notação N3)

Na seqüência, declaramos as animações que serão aplicadas aos campos nome e fotografia do professor, e também a ordem relativa entre elas. O conceito *DecorationSeq* é definido como a seqüência em que as decorações (alterações animadas ou não na interface) ocorrem dentro de uma estrutura retórica. A representação RDF da instância de *DecorationSeq* é mostrada abaixo:

**Notation 3 Form**

```
swui:instance4 rdf:type swui:DecorationSeq ;
               swui:name "anseq1" ;
               swui:firstOfSeq swui:instance5 ;
               swui:lastOfSeq swui:instance6 .
```

Figura 56 - Especificação de uma seqüência de decorações (Notação N3)

E, a seguir, as instâncias de *Decoration*, cujas propriedades definem os *widgets* a serem animados e efeito de animação.

**Notation 3 Form**

```
swui:instance5 rdf:type swui:InsertElement ;
               swui:name "NameAnimation" ;
               swui:hasElement "ProfName" ;
               swui:hasEffect "appear" ;
               swui:parameters "duration: 2.0" .

swui:instance6 rdf:type swui:InsertElement ;
               swui:name "PhotoAnimation" ;
               swui:hasElement "ProfPhoto" ;
               swui:hasEffect "appear" ;
               swui:parameters "duration: 1.0" .
```

Figura 57 – Especificação de decorações (Notação N3)

Finalmente, o código DSL que declara as animações que deverão ocorrer em resposta aos eventos de clique nos botões de expandir e contrair a lista **Research Areas**.

```
        event "name"=>"evt1", "concreteEvent"=>"click", "onInterface"=>"ProfessorView",
"onElement"=>"AreasListExp", "hasRhetoricalStructures"=>"DisplayAreaList"

        rhetoricalStructure "name"=>"DisplayAreaList", "hasDecorations"=>"an1"

        decorationSeq "name"=>"an1", "firstOfSeq"=>"AreasListAnimationDisp"

        decoration "name"=>"AreasListAnimation", "type"=>"InsertElement", "hasElement"=>"Area",
"effect"=>"appear", "parameters"=>"duration: 1.0"

        event "name"=>"evt2", "concreteEvent"=>"click", "onInterface"=>"ProfessorView",
"onElement"=>"AreasListCol", "hasRhetoricalStructures"=>"HideAreaList"

        rhetoricalStructure "name"=>"HideAreaList", "hasDecorations"=>"an2"

        decorationSeq "name"=>"an2", "firstOfSeq"=>"AreasListAnimationHide"

        decoration "name"=>"AreasListAnimationHide", "type"=>"RemoveElement", "hasElement"=>"Area",
"effect"=>"fade", "parameters"=>"duration: 1.0"
```

Figura 58 – Especificação da retórica da interface (código DSL)

Definimos o conceito *Event* como a ocorrência de uma ação sobre os elementos de interface. Uma instância de *Event* referencia, obrigatoriamente, o *widget* que “escuta” o evento e a estrutura retórica que será executada como resposta, além do nome do evento no ambiente de implementação. A representação RDF dos eventos de clique nos botões “+” (**AreasListExp**) e “–” (**AreasListCol**) é mostrada abaixo:

**Notation 3 Form**

```
swui:instance1 rdf:type swui:Event ;
               swui:name "evt1" ;
               swui:onElement "AreasListExp" ;
               swui:onInterface "ProfessorView" ;
               swui:concreteEvent "click" ;
               swui:hasRhetoricalStructures swui:instance3 .

swui:instance2 rdf:type swui:Event ;
```

```
                swui:name "evt2" ;
                swui:onElement "AreasListCol" ;
                swui:onInterface "ProfessorView" ;
                swui:concreteEvent "click" ;
                swui:hasRhetoricalStructures swui:instance4 .
```

Figura 59 – Especificação de eventos (Notação N3)

Onde **swui:instance3** e **swui:instance4** são os identificadores das estruturas retóricas chamadas **DisplayAreaList** e **HideAreaList**, respectivamente.

### 5.2.3. Mapeamentos Abstrato-Concreto e entre Visão e Modelo

Alternativamente à descrição textual da interface abstrata, o HyperDE provê uma representação gráfica dos atributos e da composição dos seus *widgets*, onde também é possível editar as propriedades que representam os dois tipos de mapeamento descritos no capítulo 3: primeiramente, o mapeamento entre *widgets* abstratos e sua representação concreta, indicado pelas propriedades *mapsTo*, *tagAttributes* e *cssClasses* e, em seguida, o mapeamento entre as camadas de visão e modelo, denotado pelas propriedades *fromAction*, *fromAttribute* e *fromClass*. A tela de mapeamentos é acessível pelo link *Mappings* na listagem de interfaces abstratas.

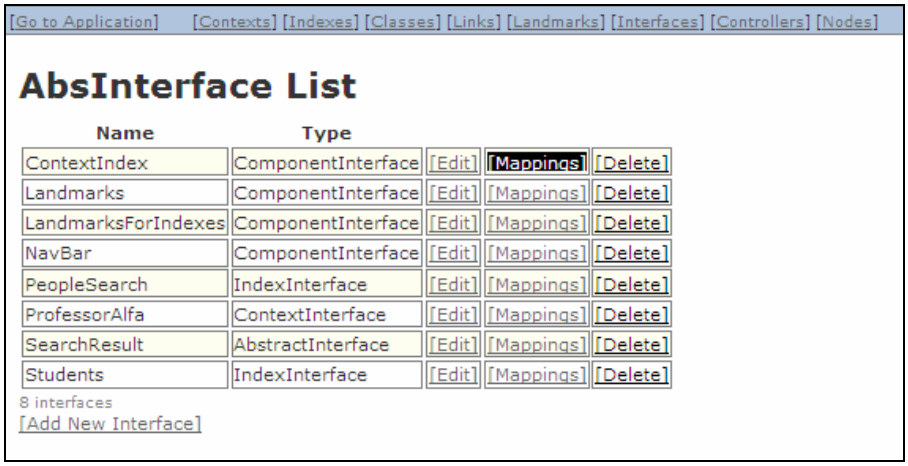



Figura 60 – Listagem de interfaces abstratas

A figura a seguir mostra a tela de mapeamentos carregada com a especificação abstrata da componente denominada "**Person**"[49].

[49] Os documentos XML que descrevem interfaces são formatados em documentos HTML com o auxílio da linguagem XSLT (*eXtensible Stylesheet Language Transformations*).

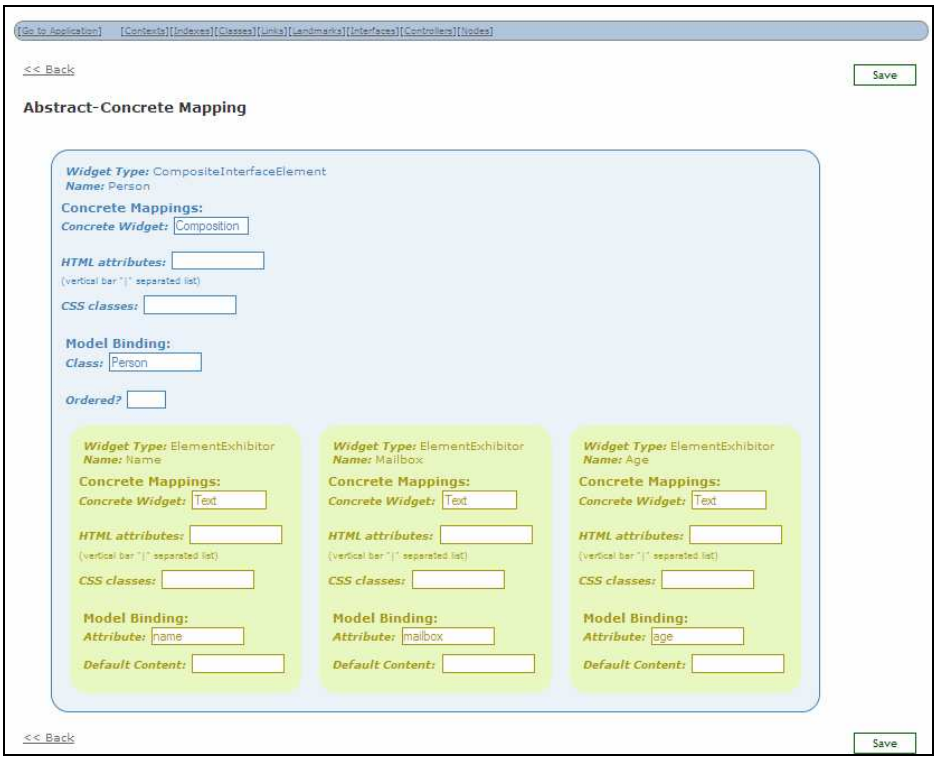



Figura 61 – Tela de mapeamentos da interface

Os campos **Concrete Widget**, **HTML Attributes**[50] e **CSS classes** correspondem, respectivamente, às propriedades *mapsTo*, *tagAttributes* e *cssClasses* dos *widgets*. As propriedades relativas ao mapeamento entre visão e modelo são apresentadas sob o título **Model Binding**: **Attribute** (propriedade *fromAttribute*), **Class** (propriedade *fromClass*) e **Target Action** (propriedade *fromAction*). E, finalmente, a edição das propriedades *defaultContent* (nas instâncias de *ElementExhibitor* e *IndefiniteVariable*) e *ordered* (*widgets* do tipo *CompositeInterfaceElement*) também está acessível através dos campos **Default Content** e **Ordered?**

### 5.2.4. Declaração de Folhas de Estilo

A declaração das classes CSS referenciadas no mapeamento abstrato-concreto precisa ser fornecida pelo projetista. No HyperDE, folhas de estilo CSS são representadas pela classe *ConcreteInterface*. A figura a seguir mostra a tela do editor de interfaces concretas carregada com a declaração de uma folha de estilo.

[50] O campo **HTML attributes** aceita uma lista de atributos HTML separados por "|".

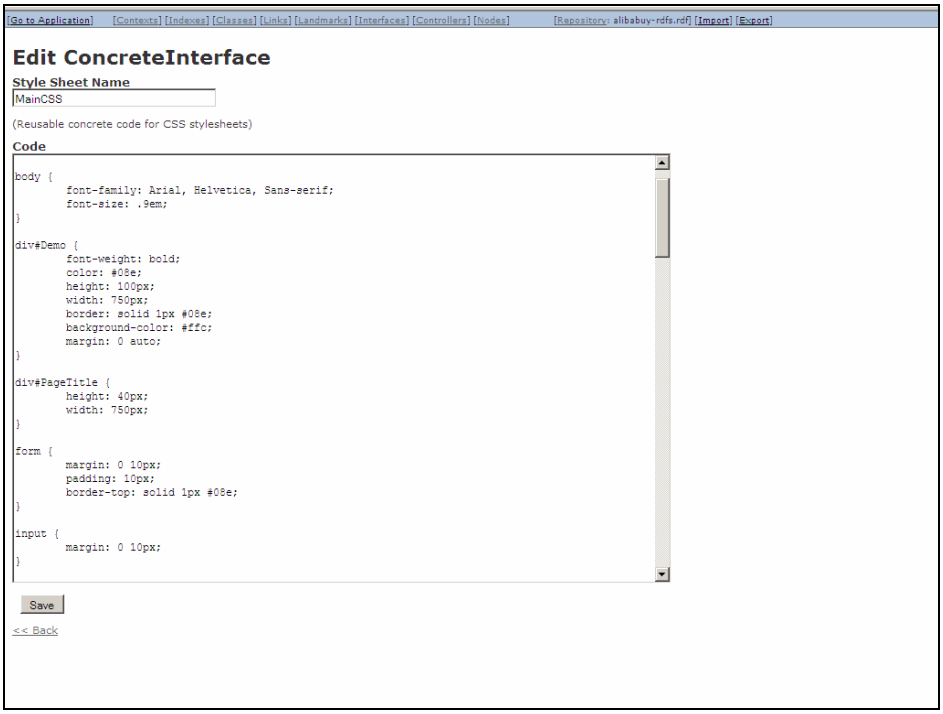



Figura 62 – Tela de edição de folhas de estilo CSS

Além de especificar quais classes serão utilizadas para formatar os *widgets* concretos, é preciso também vincular a interface abstrata à folha de estilo que contém a declaração das classes. Isto é realizado na tela de mapeamentos por meio do campo **CSS Style Sheets**. Este campo somente está disponível para instâncias da classe *AbstractInterface* (não está disponível para componentes reutilizáveis) e deve ser preenchido com uma lista de interfaces concretas separadas por ponto-e-vírgula.


Figura 63 – Detalhe da tela de mapeamentos da interface

O Módulo de Interfaces Ricas provê, também, uma folha de estilo disponível no escopo de todas as interfaces de qualquer aplicação HyperDE. Os nomes das classes CSS declaradas nesta folha de estilo iniciam com o prefixo `swui_`.[51] Entre as classes CSS definidas pelo ambiente, destacamos as seguintes:

- **swui_ordered**: não aplica estilo; introduzida no atributo **class** da *tag* durante a compilação como forma de indicar que o elemento HTML traduz um *widget* abstrato definido com a propriedade *ordered* igual a `true`;

[51] O arquivo swui.css, disponível no Apêndice III, contém a declaração da folha de estilo vinculada a todos os documentos HTML renderizados pelo Módulo de Interfaces RIA.

- **swui_invisible**: torna o elemento invisível; introduzida, durante a compilação, no atributo **class** da *tags* que representam instâncias de uma classe navegacional ou de *AbstractInterface*;
- **swui_sortable**: torna uma tabela ou uma lista ordenáveis, usando código Javascript de terceiros[52];
- **swui_draggable**: torna o elemento arrastável, usando código Javascript de terceiros.[53]

### 5.2.5. Especificação de *Widgets* Concretos

As instâncias de *HTMLTag* representam os controles de interface fornecidos pela linguagem HTML, enquanto que a classe *RichControl* é utilizada para persistir controles de interface mais sofisticados que os disponíveis na linguagem HTML, por exemplo, os fornecidos por bibliotecas *open source* tais como **Yahoo! User Interface Library (YUI)[54]** e **LivePipe UI[55]**. O Módulo de Interfaces Ricas permite ao projetista não somente mapear elementos abstratos em concretos, mas também, estender o próprio conjunto de *widgets* concretos disponíveis pela ontologia.

O link *Concrete Widgets* no Módulo de Interfaces, dá acesso ao editor de instâncias da classe *ConcreteWidget*, que generaliza *HTMLTag* e *RichControl*. Os atributos da superclasse são:

- **name**: nome do controle;
- **legalAbstractWidgs**: referências aos tipos de elementos abstratos que podem ser mapeados para o *widget* concreto.



[52] O código que torna as tabelas ordenáveis é a biblioteca SortTable (http://www.kryogenix.org/code/browser/sorttable/sorttable.js). A bilbioteca Script.aculo.us (http://script.aculo.us/) é utilizada para criar com listas ordenáveis a partir dos seguintes elementos HTML: ,<ul>,<ol>,<li>.
[53] O código é fornecido pela classe Draggable, da biblioteca Script.aculo.us.
[54] http://developer.yahoo.com/yui/
[55] http://livepipe.net/

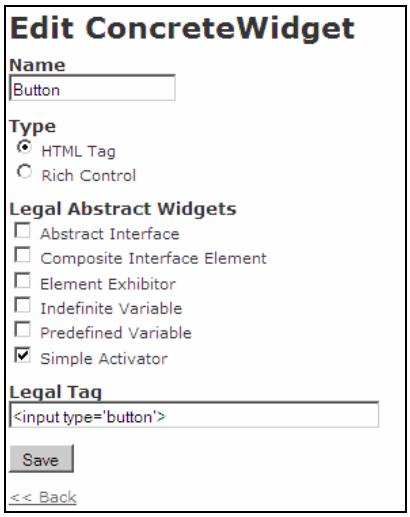



Figura 64 – Editor de instâmcias da classe *HTMLTag*

As instâncias de *HTMLTag* indicam, no atributo **legalTag**, o elemento HTML a ser usado na tradução de um elemento concreto em código. Entretanto, o tradutor de controles ricos em código executável necessita conhecer: (a) o bloco de código HTML que representa o controle; (b) o código Javascript que contém a lógica necessária para o funcionamento do controle e, opcionalmente, (c) a folha de estilo CSS interna que especifica a aparência do elemento assim como (d) as dependências externas necessárias ao funcionamento do controle (scripts e folhas de estilo). Estas informações são fornecidas, respectivamente, pelos atributos **dependencies**, **htmlCode**, **jsCode** e **cssCode** da classe *RichControl*.

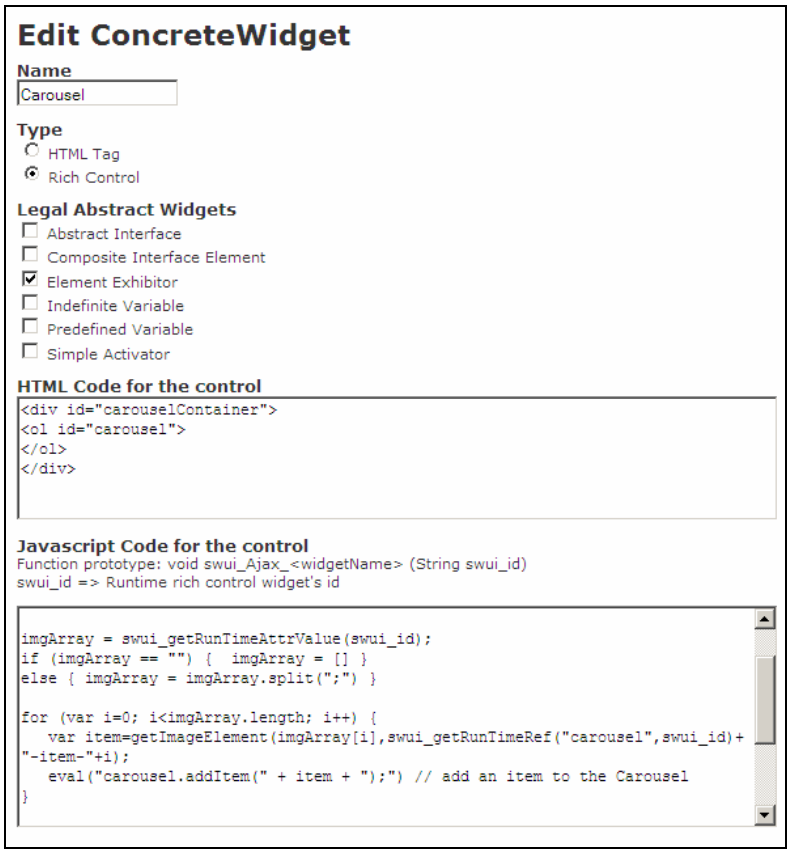



Figura 65 – Editor de instâncias da classe *RichControl*

Ao criar um novo controle rico, o projetista deverá observar algumas regras, a saber:

- A porção de código HTML que representa o controle deve possuir um único elemento raiz (uma tag ** **, por exemplo);
- Todos os elementos HTML que compõem o controle devem possuir o atributo **id**.

Há também convenções aplicáveis à função Javascript que define o comportamento do controle. O projetista fornece apenas o corpo da função associada ao controle rico; a assinatura desta função será: void swui_Rich_<widgetName> (String swui_id), onde swui_id é a variável que armazena o identificador do controle (valor do atributo id do elemento raiz) em tempo de execução. Esta variável, portanto, está disponível no escopo da função e será utilizada como argumento nas chamadas a duas funções Javascript especialmente definidas para:

- Referenciar objetos DOM: `swui_getRunTimeRef(String template_id, String swui_id)`, onde `template_id` é o valor do atributo **id** em tempo de definição do *widget*;

- Recuperar o valor do atributo navegacional mapeado pelo controle rico: `swui_getRunTimeAttrValue(String swui_id)` [56].

Finalmente, o projetista deverá ter em mente que a função Javascript somente será executada quando da criação do controle. Portanto, ela deverá ser programada para registrar *listeners* para os eventos do controle rico.

Para ilustrar, suponha um controle rico caracterizado pelo seguinte funcionamento: o controle possui: (a) um campo de entrada tipo texto, onde o usuário digita valores numéricos inteiros, e (b) um elemento **div** cujo tamanho aumenta proporcionalmente ao valor digitado pelo usuário.

O código HTML de tal controle poderia ser:

```
      <div id="MyRichControl">
        <input id="input-value" type="text" />
        <div  id="output-graph"  style="border:  1px  solid  #000;  background-color:
#0f0;">
      </div>
```

Figura 66 – Exemplo de controle rico (código HTML)

E a lógica de funcionamento definida como segue:

```
      inputWidgetRef = swui_getRunTimeRef("input-value",swui_id);
      inputWidget = document.getElementById(inputWidgetRef);
      outputWidgetRef= swui_getRunTimeRef("output-graph",swui_id);

      inputWidget.addEventListener("onchange", function (event) {
         value=parseInt(inputWidget.value);
         new Effect.Scale(outputWidgetRef, value); // Scriptaculous Effect
         }
       ,false);
```

Figura 67 – Exemplo de controle rico (código Javascript)

O código acima registra um *listener* para o evento **“onchange”** da caixa de texto. A função que será executada quando ocorrer este evento utiliza o valor inteiro digitado pelo usuário como parâmetro para aplicar o efeito **Scale** da biblioteca Script.aculo.us no elemento **div**. Observe, nos trechos em destaque, que as referências aos elementos **input-value** e **output-graph** são obtidas por meio de chamadas à função `swui_getRunTimeRef`. Isto se faz necessário porque se este controle rico estiver sendo usado para representar um *widget* abstrato mapeado em algum objeto do Modelo, então, em tempo de execução, o atributo **id** do controle e dos elementos que o compõem terá o seguinte formato: **<template_id>$id<nav_node_id>**, onde **template_id** é o **id** que foi definido para o *widget* no código HTML do controle e **nav_node_id** é o identificador do nó

[56] As funções swui_getRunTimeRef(String element_id, String swui_id) e swui_getRunTimeAttrValue(String swui_id) são definidas na biblioteca de funções Javascript que compõem o Interpretador de Interfaces RIA.

navegacional. A geração de *widgets* a partir de *templates* será detalhada ainda neste capítulo, na seção Execução das Interfaces[57].

### 5.2.6. Especificação de Efeitos de Decoração

Assim como é possível ao projetista estender o conjunto de *widgets* concretos disponíveis na Ontologia de Descrição de Interfaces Ricas, o Módulo de Interfaces do HyperDE também oferece, acessível pelo link *Effects*, um editor de instâncias da classe *DHTMLEffect*, que representa os efeitos de decoração aplicáveis a elementos de interface na plataforma DHTML (HTML + CSS + DOM + Javascript).

As instâncias de *DHTMLEffect* podem ser usadas para encapsular códigos de efeitos visuais dinâmicos fornecidos por bilbliotecas *open source*, por exemplo, o **Script.aculo.us**. Os atributos da classe são:

- **name**: nome do efeito;
- **jsCode**: o bloco de código Javascript que implementa o efeito;
- **dependencies**: trecho de código que deve ser incluído no cabeçalho do documento HTML para carregar as dependências externas necessárias ao funcionamento do efeito (scripts externos).



[57] No endereço http://www.tecweb.inf.puc-rio.br/hyperde/browser/branches/hyperde_amluna, pode-se obter o código do HyperDE acompanhado de uma aplicação que ilustra o uso de dois controles ricos: carrossel de imagens (*carousel*) e *widget* de avaliação (*rating widget*).

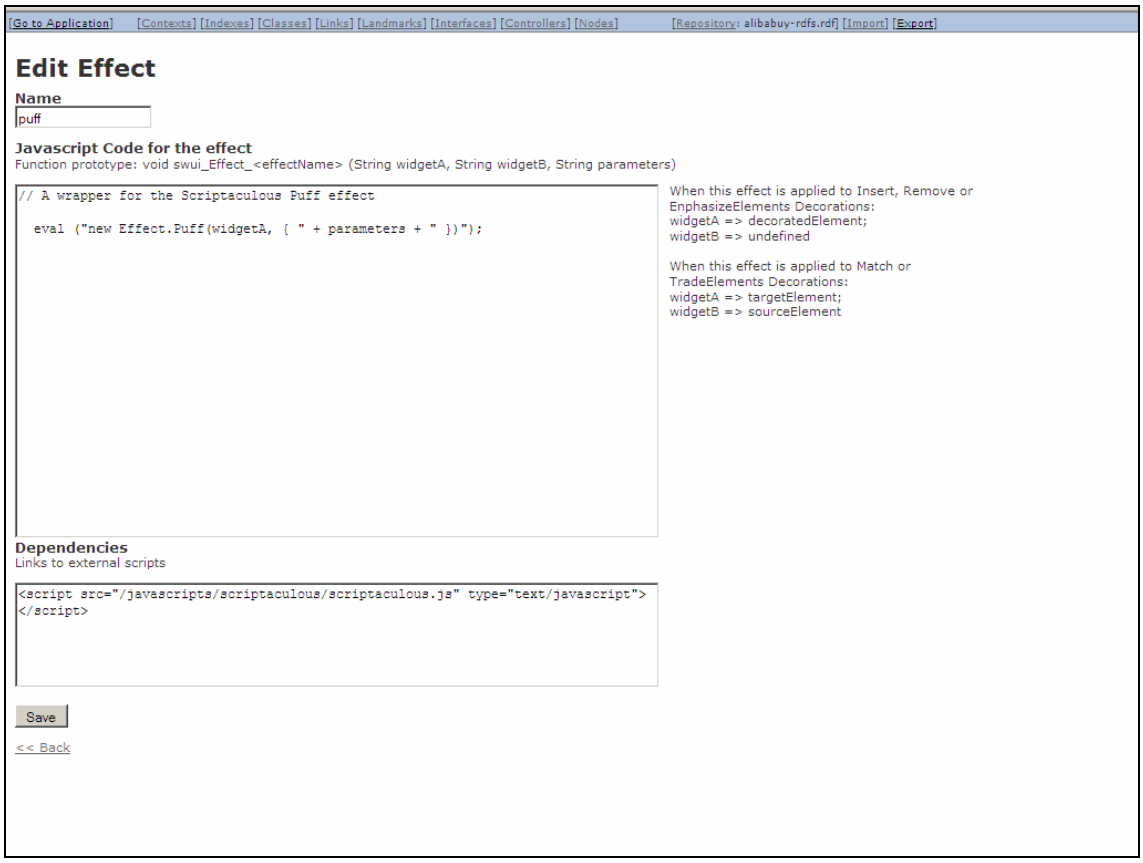



Figura 68 – Editor de instâncias da classe *DHTMLEffect*

O conteúdo do campo **Javascript Code for the effect**, na tela de edição de efeitos, serve como entrada para o corpo da função Javascript cuja assinatura será: `void swui_Effect_<effectName> (String widgetA, String widgetB, string parameters)`, onde **widgetA** e **widgetB** são as referências (atributo **id**) dos elementos que participarão da decoração, e **parameters** indica os parâmetros para o efeito aplicado. Em decorações do tipo *Insert*, *Remove* ou *Emphasize*, há somente um *widget* alvo do efeito, cujo identificador substituirá o parâmetro formal **widgetA** na chamada a função. Quando um efeito está associado a uma decoração do tipo *Match* ou *Trade*, o parâmetro **widgetA** se refere ao elemento a ser inserido, enquanto que **widgetB**, ao elemento a ser removido.

A seguir, ilustramos a codificação de um efeito capaz de alterar qualquer propriedade de estilo em um elemento HTML. O parâmetro informado para o efeito deve indicar qual propriedade será modificada e para qual valor. A lógica de funcionamento poderia ser definida como segue[58]:

```
    // It changes widget style's properties
```

[58] A função `swui_piece(String string, Char delimiter, Integer start, Integer end)` é definida na biblioteca de funções Javascript que compõem o Interpretador de Interfaces RIA e retorna uma sub-cadeia da `string` recebida, usando o caracter `delimiter` como separador e os inteiros `start` e `end` para indicar a posição da sub-cadeia procurada.

```
      //  parameters  argument  must  be  supplied  in  the  following  format:
"property:value"
        var propertyName = swui_piece(parameters, ":", 0, 1);
        var propertyValue = swui_piece(parameters, ":", 1);
        eval("document.getElementById('" + widgetA + "').style." + propertyName +
"='" + propertyValue + "'")
```

Figura 69 – Exemplo de efeito de decoração (código Javascript)

### 5.2.7. Compilação de Interfaces Abstratas

O Módulo de Interfaces Ricas também compreende o compilador de *widgets* abstratos em concretos e o compilador de descrições retóricas, descritos no capítulo anterior. O resultado do processo de compilação será armazenado nos seguintes atributos e relacionamentos do objeto *AbstractInterface*:

a) Caso o mapeamento abstrato-concreto tenha sido especificado:
   - **concrete_code**: o código da interface concreta;
   - **concrete_interfaces**: as referências às folhas de estilo CSS vinculadas à interface;
   - **interface_ref**: as referências de composição não resolvidas;
   - **richControls**: as referências aos controles ricos usados para mapear os *widgets* da interface.

b) Caso a descrição retórica da interface tenha sido especificada:
   - **transitions**: descrição das transições em formato JSON;
   - **events**: descrição dos eventos em formato JSON;
   - **decorations**: descrição das decorações em formato JSON;
   - **effects**: as referências aos efeitos de decoração especificados para as transições e eventos.

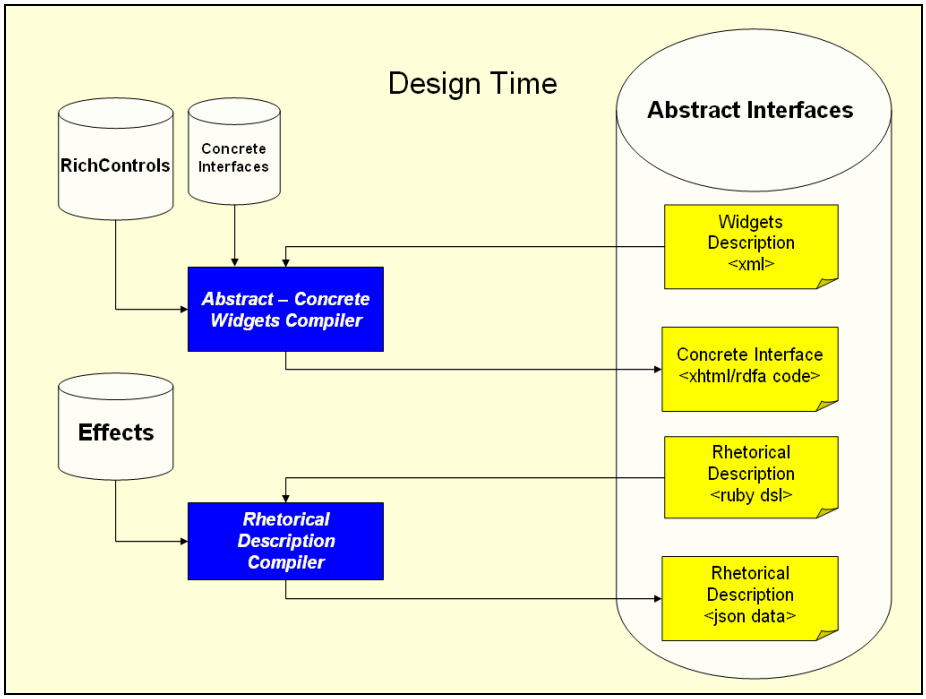



Figura 70 – Compilação de interfaces abstratas

A compilação de *widgets* abstratos em concretos se dá em duas etapas: a primeira, chamada de pré-compilação, resolve as diretivas `<%= include [component_interface_name] %>`, para inserir o código da componente reutilizável no código da interface que a inclui. A pré-compilação também executa o código Ruby especificado no corpo da diretiva `<% %>`, das chamadas **interfaces abstratas dinâmicas**, que serão explicadas em detalhes na seção de mesmo nome, ainda neste capítulo. A segunda etapa realiza a compilação propriamente dita, isto é, a validação da especificação abstrata e a tradução em código concreto. Embora as instâncias de *ComponentInterface* não sejam compiladas em código concreto, são realizados a pré-compilação e o *parsing* da descrição abstrata, para verificar se o documento XML correspondente é bem-formado e válido.

## 5.3. Ambiente de Execução

O HyperDE implementa o ambiente de execução de interfaces Ricas através dos seguintes componentes:

a) O módulo *ConcreteInterfaces*, desempenhando o papel do Renderizador de Interfaces Ricas;
b) As bibliotecas Javascript `swui_*.js`, executando as funções do Interpretador de Interfaces Ricas;
c) O controlador *Transactions*, para gerenciar as transações;

d) O modelo *SwuiController*, provendo o método `push_response` da fila de transação.

### 5.3.1. Renderização de Interfaces

O processo de renderização de uma interface inicia-se com a requisição da visualização de um nó em contexto[59] ou de um índice[60]. O componente do HyperDE responsável por atender este tipo de requisições é o controlador *Navigation*. Suas responsabilidades incluem recuperar os dados do contexto ou índice. Ele estende a classe *SwuiController*, que provê métodos para serializar os dados solicitados em objetos JSON e, em seguida, decidir qual interface abstrata será utilizada para exibir estes dados. Os critérios aplicados à seleção da interface e a sua precedência estão descritos a seguir:

**a) Independentemente se foi requisitado um contexto ou um índice:**

- Selecionar a interface cujo nome é indicado pelo usuário (no parâmetro `swuiInterface` da URL de requisição[61]).

**b) Quando é feita a requisição de um contexto:**

- Selecionar a *ContextInterface* associada ao contexto;
- Selecionar a *ContextInterface* associada à classe a que pertencem os elementos do contexto;
- Selecionar a *ContextInterface* associada a qualquer contexto (seleção *default*).

**c) Quando é feita a requisição de um índice:**

- Selecionar a *IndexInterface* associada ao índice;
- Selecionar a *IndexInterface* associada a qualquer índice (seleção *default*).

Uma vez selecionada a interface abstrata, o módulo *ConcreteInterfaces* será invocado para formatar um documento HTML contendo o código da interface concreta e o código das seguintes funções Javascript:

- `readJSONData()`, cujo valor de retorno são os dados a serem exibidos;
- `readTransitions()`, cujo valor de retorno é a especificação das transições;

[59] Formato de uma URL de requisição de contexto: `http://<server-name>:<server-port>/navigation/context/<context-id>`

[60] Formato de uma URL de requisição de índice: `http://<server-name>:<server-port>/navigation/show_index/<index-id>`

[61] Formato: `?swuiInterface=<interface-name>`. Existem alguns "curingas" que podem ser passados no lugar do nome de uma interface, e que possuem uma semântica diferente. Por exemplo, (a) o valor "0" retorna o *hash JSON* contendo os dados do nó ou índice, recurso usado internamente pelo Interpretador de Interfaces RIA; (b) o valor "1" retorna a descrição abstrata (código não compilado) da interface selecionada para exibir o contexto ou índice, cuja finalidade é o *debugging* da aplicação.

- `readEvents()`, cujo valor de retorno é a especificação dos eventos da interface;
- `readDecorations()`, cujo valor de retorno é a especificação das decorações.

A listagem a seguir mostra o código das funções `readEvents()`, `readTransitions()` e `readDecorations()`, para a instância da interface “ProfessorView” apresentada na seção Descrição Retórica.

```
<script type='text/javascript'>

// Retrieves the transitions specification embedded in the page code
function readTransitions(interfaceName) {
if (interfaceName == 'ProfessorView') {
return '{
    "nodes": [
        {
            "transition": "ProfessorAlfa",
            "from": "",
            "to": "ProfessorAlfa",
            "rhetStructsOrd": [
                {
                    "rhetStructId": "Appear",
                    "animationsSeq": [
                        "NameAnimation",
                        "PhotoAnimation"
                    ]
                }
            ]
        }
    ]
}'
}
 return ''
}

// Retrieves the events specification embedded in the page code
 function readEvents(interfaceName) {
if (interfaceName == 'ProfessorView') {
return '{
    "nodes": [
        {
            "event": "click",
            "widget": [
                "AreasListExp"
            ],
            "rhetStruct": {
                "rhetStructId": "DisplayAreaList",
                "animationsSeq": [
                    "AreasListAnimationDisp"
```

```
                    ]
                }
            },
            {
                "event": "click",
                "widget": [
                    "AreasListCol"
                ],
                "rhetStruct": {
                    "rhetStructId": "HideAreaList",
                    "animationsSeq": [
                        "AreasListAnimationHide"
                    ]
                }
            }
        ]
    }'
    }
     return ''
    }

    // Retrieves the decorations specification embedded in the page code
     function readDecorations(interfaceName) {
    if (interfaceName == 'ProfessorView') {
    return '{
        "nodes": [
            {
                "animation": "NameAnimation",
                "type": "insert",
                "effect": {
                    "name": "appear",
                    "params": "duration:  2.0"
                },
                "widget": [
                    "ProfName"
                ]
            },
            {
                "animation": "PhotoAnimation",
                "type": "insert",
                "effect": {
                    "name": "appear",
                    "params": "duration:  1.0"
                },
                "widget": [
                    "ProfPhoto"
                ]
            },
            {
                "animation": "AreasListAnimationDisp",
                "type": "insert",
```

```
                "effect": {
                    "name": "appear",
                    "params": "duration:  1.0"
                },
                "widget": [
                    "Area"
                ]
            },
            {
                "animation": "AreasListAnimationHide",
                "type": "remove",
                "effect": {
                    "name": "fade",
                    "params": "duration:  1.0"
                },
                "widget": [
                    "Area"
                ]
            }
        ]
    }'
    }
     return ''
    }
    </script>
```

Figura 71 – Funções readEvents(), readTransitions() e readDecorations(), para a interface *ProfessorView*

O documento renderizado também incluirá: (a) as funções Javascript que definem o comportamento dos controles ricos presentes na interface e as respectivas folhas de estilo internas e externas; (b) as funções Javascript que implementam os efeitos de decoração que deverão ser aplicados aos elementos da interface.

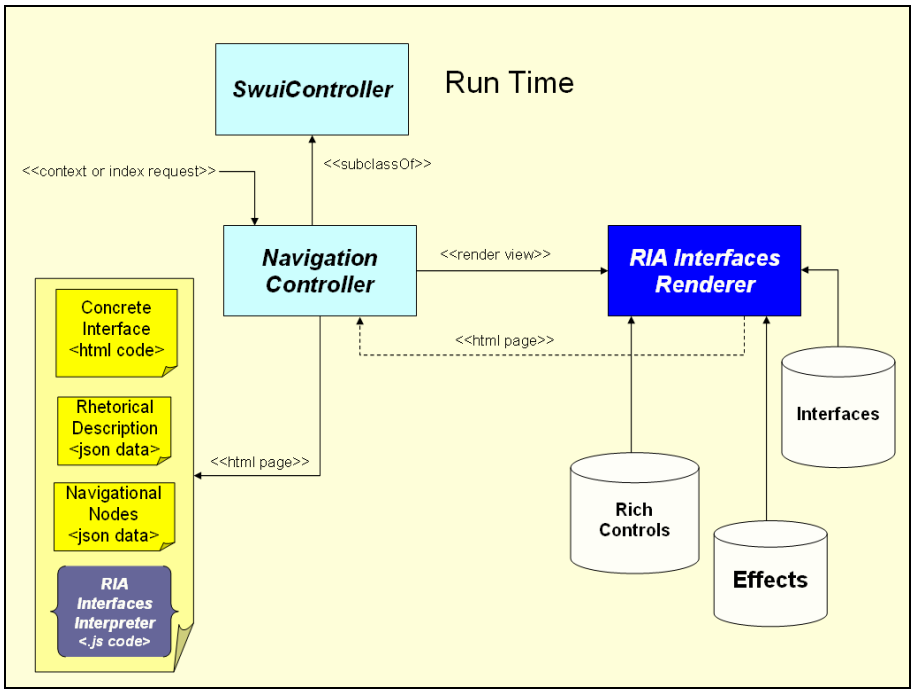



Figura 72 – Renderização de Interfaces

Alternativamente, o usuário poderá solicitar diretamente a exibição de uma interface não vinculada a um contexto ou índice[62]. Nesta hipótese, o documento HTML renderizado não contém os dados de nenhum nó navegacional. Um formulário de entrada de dados é um exemplo típico de interface que não precisa exibir objetos do modelo.

### 5.3.2. Representação dos Objetos do Modelo na Interface Renderizada

Quando o usuário solicita a visualização de um nó navegacional, o Módulo de Interfaces Ricas o envia para o cliente de forma serializada, como o valor de retorno da função Javascript `readJSONData()`. Quando é feita a requisição de um índice navegacional, as entradas do índice são serializadas em um *array* de objetos JSON.

A próxima listagem mostra um trecho do documento HTML que renderiza a instância da interface “ProfessorView” apresentada na seção Descrição Retórica. No código da função `readJSONData()`, o *hash* JSON com os dados exibidos.

```
// Retrieves the JSON Context or Index data embedded in the page code
function readJSONData(interfaceName) {
if (interfaceName == 'ProfessorView') {
return '{
    "nodes": [
        {
            "swui:id": "o_989c8ef9",
            "degree_from": "UCLA",
```

```
                    "photo": "http://localhost:3000/images/daniel.gif",
                    "name": "Daniel Schwabe",
                    "degree": "PhD: Computer Science",
                    "category": "Associate Professor",
                    "degree_year": "1981",
                    "homepage": "http://www.tecweb.puc-rio.br",
                    "telephone": "3114-1500 x: 4356",
                    "areas": [
                        {
                            "areas|name": "Software Engineering",
                            "areas|name|swui:target":
"/navigation/context/o_c68c0e56@0?p=o_989c8ef9"
                        },
                        {
                            "areas|name": "Artificial Intelligence",
                            "areas|name|swui:target":
"/navigation/context/o_c68c0e56@1?p=o_989c8ef9"
                        },
                        {
                            "areas|name": "Hypermedia",
                            "areas|name|swui:target":
"/navigation/context/o_c68c0e56@2?p=o_989c8ef9"
                        }
                    ],
                    "email": "dschwabe @ inf.puc-rio.br"
                }
            ]
        }'
        }
        return ''
        }
```

Figura 73 – Função readJSONData() para uma instância da interface *ProfessorView*

Em termos simples, um objeto JSON é uma coleção não-ordenada de pares chave-valor. No caso geral, os nomes dos atributos de um nó navegacional ou entrada de índice figuram como os nomes das chaves e os valores dos atributos, como os respectivos valores associados às chaves. No entanto, algumas convenções são aplicadas a certos casos particulares, a saber:

- O identificador do nó é informado na chave `swui:id`;
- Se o atributo é do tipo índice (*IndexAttribute*), o seu valor será representado no *hash* como um *array* de objetos, onde os objetos são as entradas do índice. Este é o caso do campo **Research Areas** na interface “ProfessorView”, que exibe o conteúdo do atributo **areas** do objeto **Professor**. Os atributos das entradas de um índice são referenciados por chaves nomeadas como

[62] Formato de uma URL de requisição de interface abstrata: `http://<server-name>:<server-port>/navigation/ show_interface/<interface-name>`

"<node_attribute_name>| <index_entry_attribute_name>". No exemplo acima, o índice de áreas de pesquisa possui apenas um atributo chamado **name**, daí a chave de nome composto `areas|name`;

- Quando o atributo do nó representa uma âncora de navegação (*ContextAnchorAttribute*, *IndexAnchorAttribute*), a URL alvo estará associada à chave de nome "`<attribute_name>|swui:target`". Considerando novamente o exemplo do índice **Research Areas**, o atributo **name** é uma âncora para o objeto **Area**, portanto, a chave `areas|name|swui:target`.

O *hash* JSON transporta não somente os atributos de um nó navegacional. Primitivas do método SHDM – *landmarks*[63] e índices dos elementos do contexto – também estão acessíveis como atributos do nó, por meio das seguintes chaves:

- `swui:landmarks`: *array* de objetos do tipo *landmark;*
- `swui:landmarks|swui:landmark`: *label* do objeto *landmark*;
- `swui:landmarks|swui:landmark|swui:target`: URL alvo da âncora que o objeto *landmark* representa;
- `swui:contextIdx`: *array* de entradas do índice do contexto;
- `swui:idxEntry|item`: *label* da entrada do índice;
- `swui:idxEntry|item|swui:target`: URL de destino da entrada do índice;
- `swui:contextPrev|item`: *label* do elemento anterior na ordenação do contexto;
- `swui:contextPrev|item|swui:target`: URL para navegar para o elemento anterior na ordenação do contexto;
- `swui:contextNext|item`: *label* do próximo elemento na ordenação do contexto;
- `swui:contextNext|item|swui:target`: URL para navegar para o próximo elemento na ordenação do contexto;

Na figura abaixo, apresentamos uma terceira versão da interface "ProfessorView". Aqui, além de renderizar o objeto **Professor**, a interface contém os *landmarks* da aplicação, o índice do contexto, e as âncoras para os elementos anterior e próximo.

[63] Landmarks designam estruturas de acesso ou contextos que podem ser acessados a partir de qualquer local da aplicação


Figura 74 – Terceira versão da interface *ProfessorView*

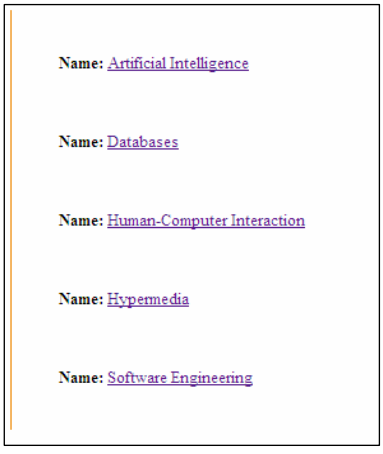
A listagem a seguir mostra como as informações relativas ao contexto de navegação são incluídas como atributos do objeto navegacional no *hash* JSON:

```
    {
        "nodes": [
            {
                "swui:landmarks": [
                    {
                        "swui:landmarks|swui:landmark": "Areas",
                        "swui:landmarks|swui:landmark|swui:target":
"/navigation/show_index/o_927d0c9c?p="
                    },
                    {
                        "swui:landmarks|swui:landmark": "Professors",
                        "swui:landmarks|swui:landmark|swui:target":
"/navigation/show_index/o_3ead323d?p="
                    },
                    {
                        "swui:landmarks|swui:landmark": "Students",
                        "swui:landmarks|swui:landmark|swui:target":
"/navigation/show_index/o_acc25a43?p="
                    },
                    {
                        "swui:landmarks|swui:landmark": "Publications",
                        "swui:landmarks|swui:landmark|swui:target":
"/navigation/context/o_b12e03f2?p="
                    },
                    {
                        "swui:landmarks|swui:landmark": "Search People",
                        "swui:landmarks|swui:landmark|swui:target":
"/navigation/show_index/o_29a686da?p="
                    }
                ],
                "swui:contextIdx": [
                    {
```

```
                        "swui:idxEntry|item": "Arndt Von Staa",
                        "swui:idxEntry|item|swui:target":
"/navigation/context/o_881e9e9d@0?p="
                    },
                    {
                        "swui:idxEntry|item": "Clarisse Sieckenius ...",
                        "swui:idxEntry|item|swui:target":
"/navigation/context/o_881e9e9d@1?p="
                    },
                    {
                        "swui:idxEntry|item": "Daniel Schwabe"
                    },
                    {
                        "swui:idxEntry|item": "Marco Antonio Casano...",
                        "swui:idxEntry|item|swui:target":
"/navigation/context/o_881e9e9d@3?p="
                    },
                    {
                        "swui:idxEntry|item": "Rubens Mello",
                        "swui:idxEntry|item|swui:target":
"/navigation/context/o_881e9e9d@4?p="
                    }
                ],
                "swui:contextPrev|item": "Clarisse Siecke...",
                "swui:contextPrev|item|swui:target":
"/navigation/context/o_881e9e9d@1?p=",
                "swui:contextNext|item": "Marco Antonio C...",
                "swui:contextNext|item|swui:target":
"/navigation/context/o_881e9e9d@3?p="
            }
        ]
    }
```

Figura 75 – Landmarks, e índices de contexto, serializados em um *hash* JSON

Todavia, quando a estrutura de dados JSON representa um índice navegacional, os *landmarks* da aplicação estão presentes, não como atributos, mas como entradas adicionais do índice. As chaves que dão acesso aos atributos de um objeto do tipo *landmark* são: "`swui:landmark`": (atributo *label*) e "`swui:landmark|swui:target`" (URL alvo da âncora).




A figura ao lado mostra uma instância do índice **Research Areas**, e a listagem abaixo, o *hash* JSON correspondente, incluindo as entradas relativas aos *landmarks* da aplicação.

Figura 76 – Índice “ReseachAreas”

```
{
    "nodes": [
        {
            "swui:id": "32673000",
            "name": "Artificial Intelligence",
            "name|swui:target": "/navigation/context/o_e41b816b@0?p="
        },
        {
            "swui:id": "32330170",
            "name": "Databases",
            "name|swui:target": "/navigation/context/o_e41b816b@1?p="
        },
        {
            "swui:id": "32275370",
            "name": "Human-Computer Interaction",
            "name|swui:target": "/navigation/context/o_e41b816b@2?p="
        },
        {
            "swui:id": "32226800",
            "name": "Hypermedia",
            "name|swui:target": "/navigation/context/o_e41b816b@3?p="
        },
        {
            "swui:id": "32179250",
            "name": "Software Engineering",
            "name|swui:target": "/navigation/context/o_e41b816b@4?p="
        },
        {
            "swui:landmark": "Areas",
            "swui:id": "o_20be79a3",
            "swui:landmark|swui:target": "/navigation/show_index/o_927d0c9c?p="
        },
        {
            "swui:landmark": "Professors",
            "swui:id": "o_7fb79c4e",
            "swui:landmark|swui:target": "/navigation/show_index/o_3ead323d?p="
        },
        {
            "swui:landmark": "Students",
            "swui:id": "o_05ce83a3",
            "swui:landmark|swui:target": "/navigation/show_index/o_acc25a43?p="
        },
        {
            "swui:landmark": "Publications",
            "swui:id": "o_4a7722e5",
            "swui:landmark|swui:target": "/navigation/context/o_b12e03f2?p="
        },
        {
            "swui:landmark": "Search People",
            "swui:id": "o_PersonTypes_f798",
            "swui:landmark|swui:target": "/navigation/show_index/o_29a686da?p="
```

```
            }
        ]
    }
```

Figura 77 – Índice navegacional serializado em um *hash* JSON

Índices são listas e, como mencionado no capítulo 3, o Modelo pode informar ao Módulo de Interfaces Ricas qual a posição de cada elemento em uma lista. Para este fim, deve-se utilizar a chave `swui:order` na representação JSON do objeto, como no exemplo abaixo:

```
    {
        "nodes": [
            {
                "swui:id": "32226800",
                "swui:order": "2",
                "name": "Hypermedia",
                "name|swui:target": "/navigation/context/o_e41b816b@3?p="
            },
            {
                "swui:id": "32179250",
                "swui:order": "1",
                "name": "Software Engineering",
                "name|swui:target": "/navigation/context/o_e41b816b@4?p="
            }
        ]
    }
```

Figura 78 – Exemplo de utilização da chave swui:order

No *array* acima, embora a área de pesquisa **Hypermedia** preceda **Software Engineering**, a primeira será inserida após a segunda na lista, por causa da ordem especificada na chave `swui:order`. Para tanto, basta que a composição **Research Areas** esteja associada à classe CSS **swui_ordered**.

### 5.3.3. Interfaces Abstratas Dinâmicas

Por interfaces abstratas dinâmicas, nos referimos às interfaces cuja especificação de estrutura e mapeamento é gerada em tempo de execução da aplicação (*on-the-fly*). Em outras palavras, a descrição dos *widgets* que compõem a interface (atributo **widget_desc** do objeto) contém código Ruby a ser avaliado. Diferentemente das interfaces clássicas, o seu código concreto não foi pré-compilado, visto que a própria definição abstrata dependerá do contexto da aplicação quando a interface for requisitada. Por este motivo, a cada solicitação de renderização, o Módulo de Interfaces Ricas executará os seguintes passos previamente:

- Executar o código Ruby armazenado em **widget_desc**. A execução deste código deve retornar uma especificação abstrata válida;
- Compilar a especificação abstrata resultante.

A descrição dos *widgets* da interface pode combinar código Ruby e código XML. De fato, o código Ruby em uma interface abstrata dinâmica aparece tipicamente embutido na especificação XML dos *widgets*. O símbolo <% é utilizado para introduzir um trecho de código Ruby na descrição XML, enquanto que %>, para fechá-lo.

As instâncias de *AbstractInterface* e também quaisquer componentes reutilizáveis podem conter código Ruby em seu campo ***Widgets Decription*** como ilustrado na figura a seguir:

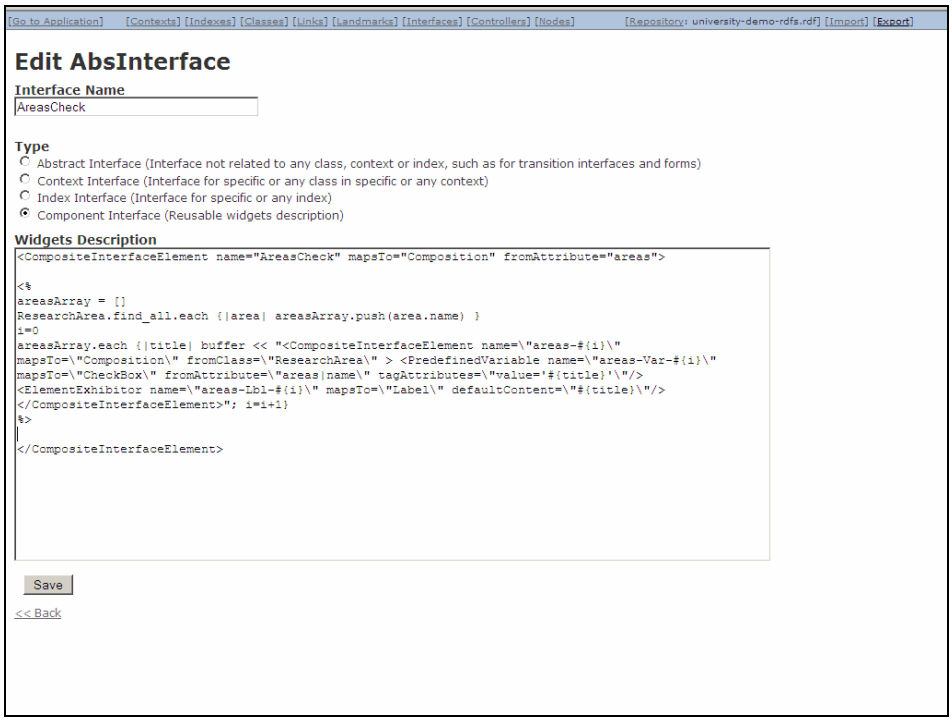



Figura 79 – Código Ruby na descrição abstrata da interface

Novamente, há um conjunto de variáveis e funções que foram convencionadas para uso no código das interfaces dinâmicas, a saber:

- **buffer**: armazena o resultado da computação realizada: seu valor deve ser atribuído dentro do código Ruby da interface dinâmica e deve conter um trecho de especificação abstrata na sintaxe XML;
- **jsonData** e **list**: variáveis inicializadas no pré-compilador da interface (responsável por interpretar o código Ruby); são os parâmetros do método `ModelUtils.traverseJSON(Hash jsonData, boolean list)`. As chaves do *hash* **jsonData** correspondem aos nomes dos atributos do(s) objeto(s) que a interface irá renderizar. A variável **list** é inicializada como **true** se os dados em **jsonData** representam um índice navegacional.

O código da interface **DefaultJSONDisplay** utiliza uma chamada ao método `ModelUtils.traverseJSON`, um iterador Ruby[64]. O bloco de código passado ao método percorre a estrutura de dados disponível em `jsonData` e seleciona as classes de *widgets* para representar cada atributo do objeto, com base no nome do atributo, o seu tipo e até mesmo o seu conteúdo. Por este motivo, todo bloco passado ao método `ModelUtils.traverseJSON` deve receber cinco argumentos, nesta ordem:

- **type**: indica o tipo do atributo. Se for um *Array*, o valor de **type** será definido pelo iterador como "OpenList" antes de percorrer o *Array* e como "CloseList" ao terminar de percorrê-lo. Se o atributo não for um *Array*, o valor de type será definido como "Item";
- **key**: chave no *hash* que representa o objeto;
- **value**: valor no *hash* que representa o objeto;
- **hash**: *hash* que representa o objeto;
- **list**: booleano que sinaliza se um atributo é do tipo *Array* ou não.

O bloco em **DefaultJSONDisplay** armazena na variável **buffer** a especificação abstrata que utiliza instâncias de CompositeInterfaceElement para representar atributos do tipo índice, instâncias de SimpleActivator para representar âncoras, e instâncias de *ElementExhibitor* para representar os demais tipos de atributos. Na figura a seguir, uma instância de **DefaultContextInterface** em detalhe, exibindo os dados de um nó da classe **Student**. A **DefaultJSONDisplay** é a componente principal das interfaces **DefaultContextInterface** e **DefaultIndexInterface**[65].

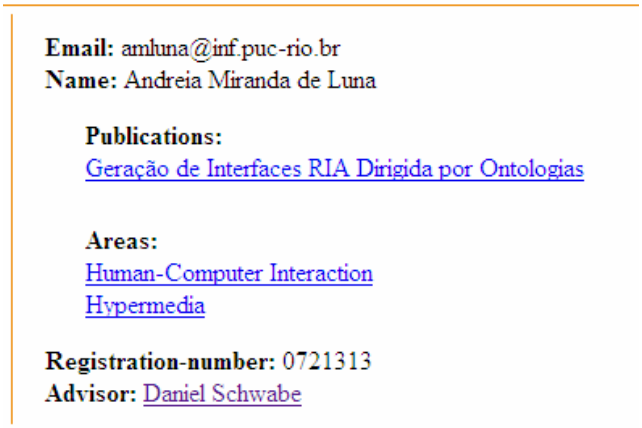



Figura 80 – Detalhe de uma instância de *DefaultContextInterface*

Para demonstrar o poder das especificações dinâmicas, o código abaixo é capaz de gerar um conjunto de controles *PredefinedVariable*, mapeados para botões tipo Radio, onde cada opção representa uma instância de **Research Areas**.

[64] Iteradores, em programação, são métodos que podem invocar um bloco de código repetidamente. [44]

```
<CompositeInterfaceElement name="AreasRadio" mapsTo="Composition" >

<%
areasArray = []
ResearchArea.find_all.each {|area| areasArray.push(area.name) }

i=0

areasArray.each  do |name|
   buffer << "<CompositeInterfaceElement name=\"area-#{i}\" mapsTo=\"Composition\">"
   buffer << "<PredefinedVariable name=\"area-#{i}-Var\" mapsTo=\"RadioButton\"
tagAttributes=\"value='#{name}'|name='area-Var'\"/>"
   buffer << "<ElementExhibitor name=\"area-#{i}-Lbl\" mapsTo=\"Label\" defaultContent=\"#{name}\"/>"
   buffer << "</CompositeInterfaceElement>"
   i=i+1
end
%>

</CompositeInterfaceElement>
```

Figura 81 – Exemplo de código fonte de uma interface abstrata dinâmica

A especificação abstrata resultante da avaliação do código Ruby seria:

```
<CompositeInterfaceElement name="AreasRadio" mapsTo="Composition" >

   <CompositeInterfaceElement name="area-0" mapsTo="Composition">
      <PredefinedVariable name="area-0-Var" mapsTo="RadioButton" tagAttributes=
"value='Databases'|name='area-Var'"/>
      <ElementExhibitor name="area-0-Lbl" mapsTo="Label" defaultContent="Databases"/>
   </CompositeInterfaceElement>

   <CompositeInterfaceElement name="area-1" mapsTo="Composition">
      <PredefinedVariable name="area-1-Var" mapsTo="RadioButton" tagAttributes= "value='Human-
Computer Interaction'|name='area-Var'"/>
      <ElementExhibitor name="area-1-Lbl" mapsTo="Label" defaultContent="Human-Computer
Interaction"/>
   </CompositeInterfaceElement>

   <CompositeInterfaceElement name="area-2" mapsTo="Composition">
      <PredefinedVariable name="area-2-Var" mapsTo="RadioButton" tagAttributes= "value='Human-
Computer Interaction'|name='area-Var'"/>
      <ElementExhibitor name="area-2-Lbl" mapsTo="Label" defaultContent="Human-Computer
Interaction"/>
   </CompositeInterfaceElement>

   <CompositeInterfaceElement name="area-3" mapsTo="Composition">
```

[65] Os códigos das interfaces *default* e de suas componentes estão disponíveis no Apêndice IV.

```
            <PredefinedVariable name="area-3-Var" mapsTo="RadioButton" tagAttributes= "value='Human-
Computer Interaction'|name='area-Var'"/>
            <ElementExhibitor name="area-3-Lbl" mapsTo="Label" defaultContent="Human-Computer
Interaction"/>
        </CompositeInterfaceElement>

        <CompositeInterfaceElement name="area-4" mapsTo="Composition">
            <PredefinedVariable name="area-4-Var" mapsTo="RadioButton" tagAttributes= "value='Human-
Computer Interaction'|name='area-Var'"/>
            <ElementExhibitor name="area-4-Lbl" mapsTo="Label" defaultContent="Human-Computer
Interaction"/>
        </CompositeInterfaceElement>

    </CompositeInterfaceElement>
```

Figura 82 – Exemplo de código compilado de uma interface abstrata dinâmica

### 5.3.4. Execução das Interfaces

A representação da estrutura e do funcionamento da interface concreta combina as linguagens XHTML+RDFa+JSON; junto com a interface renderizada, o navegador Web do cliente também precisa carregar o código do Interpretador de Interfaces Ricas, a biblioteca Javascript que tem a função de "executar" a interface[66].

O código do Interpretador de Interfaces Ricas está dividido em quatro bibliotecas:

- `swui_dom.js`: suas funções manipulam a árvore DOM;
- `swui_rhet.js`: suas funções dão comportamento à interface;
- `swui_model.js`: suas funções implementam a comunicação entre visão e modelo;
- `swui_common`: suas funções dão suporte às demais bibliotecas.

A execução da interface inicia com travessia da árvore DOM, em busca das anotações RDFa. Em seguida, a biblioteca **swui_rhet.js** irá recuperar e interpretar as descrições dos eventos, transições e decorações especificados para a interface. Estas descrições foram compiladas em objetos JSON e incluídas no código do documento HTML renderizado como os valores de retorno das funções readEvents(), readTransitions() e readDecorations().

[66] Os testes com o código Javascript do Interpretador de Interfaces Ricas foram realizados no navegador Mozilla Firefox 3.5.5 com o complemento Firebug 1.4.5.

Inicialmente, as interfaces concretas estão invisíveis na página. As bibliotecas **swui_dom.js** e **swui_rhet.js** dividem as funções de exibir e ocultar as interfaces, executando as respectivas transições de entrada e saída, se houver.

Quando uma interface é exibida, é preciso também instanciar os objetos de interação que representam objetos de dados. Quando um elemento da interface é mapeado em um elemento do modelo (classe ou atributo), ele também é traduzido em um *wiget* concreto invisível. No entanto, diferentemente das interfaces, as composições que representam objetos navegacionais não serão tornadas visíveis em tempo de carregamento da página. Antes, elas irão prover os *templates* que a biblioteca **swui_dom.js** usará para criar dinamicamente os *widgets* que efetivamente serão exibidos na interface.

Os *templates* fornecem o tipo de *widget* a ser criado, bem como suas propriedades de estilo. O conteúdo a ser exibido é provido pelo(s) objeto(s) navegacionais recebidos junto com a interface. A biblioteca **swui_dom.js** consegue acessar tais objetos através da chamada à função readJSONData().

Para decidir qual elemento HTML usar como *template*, a biblioteca **swui_dom.js** fará o casamento entre as chaves do *hash* JSON e os atributos `@property` dos elementos HTML. Por este motivo, o conteúdo dinâmico da interface somente poderá ser exibido caso os nomes dos atributos utilizados no mapeamento entre visão e modelo tenha estrita correspondência com os nomes das chaves na representação *hash* dos objetos da aplicação.

O procedimento de instanciar *widgets* em tempo de carregamento da página, para exibir os dados recebidos em um objeto JSON, é chamado de renderização do objeto. Nos novos *widgets*, o atributo **id** será a composição do **id** do elemento usado como *template* e o identificador do objeto navegacional sendo renderizado, ou seja, terá o formato: **<template_id>$id<nav_node_id>**.

Caso o *template* seja elemento de uma composição associada à classe CSS **swui_ordered**, a biblioteca **swui_dom.js** posicionará o item dentro da lista conforme indicado na chave `swui:order`, do *hash* JSON.

### 5.3.5. Retórica na Interface

Tipicamente, após o cliente receber uma nova interface, o Interpretador de Interfaces Ricas determinará se foi especificada alguma estrutura retórica para ser executada durante a sua entrada (transição que declara apenas o estado de destino).

Em outras situações, talvez seja necessário verificar se há a necessidade de executar estruturas retóricas durante a saída de uma interface (transição que declara

apenas o estado de origem), ou durante uma transição em que se conhecem ambos os estados. O primeiro cenário ocorre quando o usuário aciona um elemento que ativa uma mudança de estado na interface (em conseqüência de uma navegação ou do retorno de uma resposta à execução de uma operação no Modelo), enquanto que o segundo cenário é possível quando a interface de destino está aninhada na interface de origem.

Em qualquer caso, porém, a semântica das estruturas retóricas está sujeita a algumas restrições:

- Os elementos que não figuram em nenhuma especificação retórica serão exibidos primeiro (entrada da interface) e escondidos por último (saída da interface);
- Na entrada de interfaces de transição, a retórica em seu conteúdo estático (*widgets* que não mapeiam objetos do modelo) ocorrerá em paralelo com a retórica em seu conteúdo dinâmico;
- Os elementos que participam de alguma decoração cujo efeito pretendido seja o de torná-lo visível (por exemplo, o efeito `appear`), devem ser especificados com a propriedade de estilo `display` igual a '`none`';
- Não é possível executar uma seqüência de duas ou mais decorações de entrada (ou saída) sobre um mesmo elemento de interface – a menos que a série de decorações intercale o surgimento e a remoção do elemento.

### 5.3.6. Comunicação de Eventos entre Visão e Modelo

A comunicação de eventos entre as camadas de Visão e Modelo é implementada, em um sentido, pelo Interpretador de Interfaces Ricas e, em outro, pela fila de respostas associadas a uma transação, conceito discutido no capítulo anterior.

A biblioteca **swui_dom.js** é responsável por invocar o Modelo em resposta a eventos de interface, por exemplo, a ativação de um *link* ou submissão de um formulário. O projetista encapsula as funcionalidades da aplicação em ações de controladores (instâncias da classe *Controller*, definida no metamodelo do HyperDE).

As operações são referenciadas pela propriedade *fromAction* de um ativador ou evento, uma URL cuja sintaxe é: `/<controller>/<action>/<id>`. O parâmetro **id** representa o identificador de um nó navegacional e somente pode ser conhecido em tempo de execução. Por este motivo, caso seja necessário suprir tal parâmetro na chamada a uma operação, o projetista informará o valor "**/<controller>/<action>/:id**" na propriedade *fromAction* do elemento abstrato, onde **:id** é um símbolo (*placeholder*) que será substituído pelo valor real do identificador no momento da chamada à URL.

Caso o projetista da interface tenha especificado uma interface de transição associada ao elemento ativador, a operação será executada sob o contexto de uma transação[67]. O controlador *Transactions* provê a interface para iniciar as transações e para consumir a fila que armazena os resultados das operações:

- `void start()`[68]: retorna para o Interpretador de Interfaces Ricas o identificador de uma nova transação;
- `void pop_response()`: remove o primeiro elemento da fila de respostas da transação e o retorna para o Interpretador de Interfaces Ricas[69].

As interfaces de transição, por definição, são exibidas apenas entre o início e o fim da transação e renderizam os objetos armazenados na fila de respostas durante o processamento da ação no Modelo. As interfaces de transição devem ser carregadas dentro de um componente da interface que disparou a ação (a composição que referencia a interface de transição na propriedade *loadInterface*). Ao mesmo tempo, a ação deve ser codificada de tal forma a utilizar a fila de respostas da transação. A classe *SwuiController*, superclasse de todos os controladores definidos pelo usuário, provê os seguintes métodos:

- `void push_response(Object data)`: recebe como argumento o elemento a ser armazenado na fila de respostas da transação. Pode ser um objeto, um *array* de objetos navegacionais, ou mesmo o string marcador de fim de transação (`"EOT"`). O método também serializa os objetos navegacionais em os objetos JSON antes de armazená-los na fila;
- `void render_context(Object navContext, Integer contextNodePosition)`: retorna a interface que renderiza um nó em contexto[70];
- `void render_index(Object navIndex)`: retorna a interface que renderiza um índice[71];
- `void render_response(Object data, String interfaceName)`: retorna a interface[72] que renderiza o resultado de uma operação. Se a

[67] Formato das chamadas às operações do Modelo:
(a) chamada clássica: `http://<server-name>:<server-port>/<controller>/<action>[/<id>]`
(b) chamada sob o escopo de uma transação: `http://<server-name>:<server-port>/<transaction-id>/<controller>/<action>[/<id>]`

[68] Formato das chamadas à `start()`: `http://<server-name>:<server-port>/transactions/start`

[69] Formato das chamadas à `pop_response()`: `http://<server-name>:<server-port>/transactions/pop_response/<transaction-id>`

[70] O parâmetro **contextNodePosition** representa a ordem do nó navegacional dentro do contexto referenciado pelo parâmetro **navContext**.

operação que invocou este método estiver sendo executada sob o contexto de uma transação, a chamada à `render_response` gravará o marcador `"EOT"` na fila de respostas antes de renderizar a interface com o resultado.

Após o início de uma transação, o Interpretador de Interfaces Ricas iniciará um *loop* para consumir a fila respostas, retirando a cada iteração um dos elementos e o renderizando.

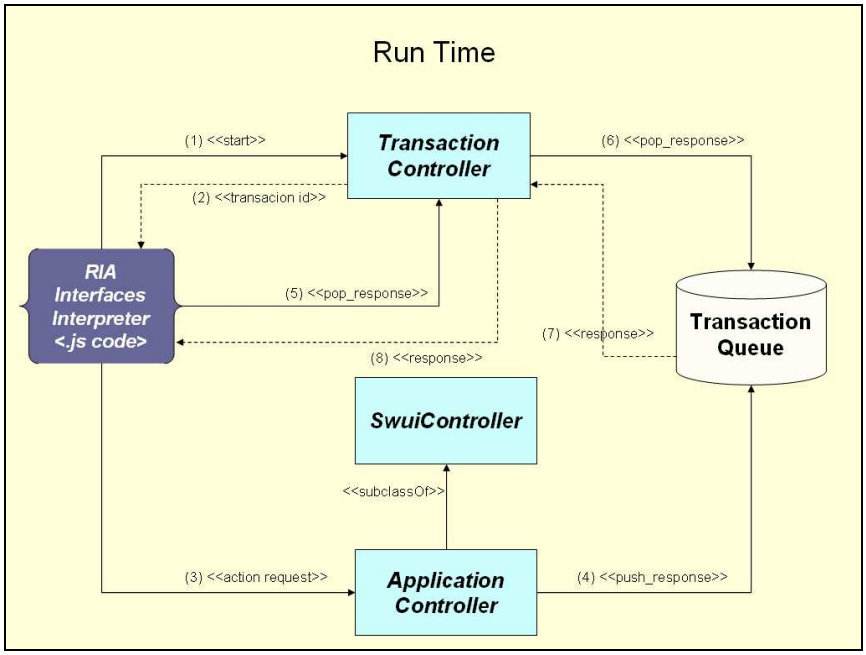



Figura 83 – Execução de uma operação sob o contexto de uma transação

Abaixo, ilustramos o código de uma operação que realiza a busca de amigos de um professor. O índice navegacional **FriendsOfProfessor** representa o resultado desta computação.

```
# Professor's id is held in params['id']
professor = @params["id"]

# Retrieves the index
index = Index.find_by_name('FriendsOfProfessor')

# Sets the parameter Professor (params['p'])for the index
@params["p"]= professor

# Renders the index
render :text => render_index(index)
```

Figura 84 – Exemplo de operação definida no Modelo

[71] Em `render_context` e `render_index`, a interface será selecionada segundo os critérios detalhados na Seção Renderização de Interfaces.

[72] A interface identificada no parâmetro **swuiInterface** da requisição HTTP tem precedência sobre a interface selecionada no argumento **intefaceName** da função `render_response`. Se os dois valores forem omitidos, a seleção *default* será **DefaultAbstractInterface**.

Observe, agora, como a operação poderia ser reescrita para exibir incrementalmente os amigos do professor na interface de transição, antes de finalmente renderizar a lista completa. No código anterior, a busca foi realizada de forma implícita (utilizando o índice **FriendsOfProfessor**). Desta vez, a busca é feita de forma explícita, salvando os resultados parciais na fila de respostas e renderizando a interface **DefaultIndexInterface** ao final da pesquisa. Este exemplo ilustra como a codificação de uma operação precisaria ser modificada para tirar proveito do protocolo de comunicação baseado em fila de mensagens entre as camadas de Visão e Modelo. Caso o cenário descrito envolvesse uma busca com alto custo em tempo de processamento, o recurso de informar a camada de Visão sobre quaisquer resultados que se tenham tornado disponíveis certamente agregaria interatividade à aplicação.

```
      # Professor's id is held in params['id']
      professor = @params["id"]

      # Finds the professor's friends and stores each of them in the transaction
response queue
      Data = Array.new
      Professor.find(professor).is_friend_of.each { |friend| push_response(friend);
data << friend }

      render :text => render_response(data, "DefaultIndexInterface")
```

Figura 85 – Operação reescrita para suportar a execução sob o contexto de uma transação

A fila da transação pode ser usada para mais do que armazenar os resultados de uma operação. O conteúdo da fila também pode comunicar a ocorrência de eventos na camada de Modelo durante o progresso de uma transação. Por causa do *loop* de leitura da fila no lado cliente, a Visão será atualizada a cada informação feita disponível. Os objetos recuperados da fila podem representar instâncias de qualquer tipo de evento. Imagine, por exemplo, que numa comunidade virtual, durante a execução de uma operação qualquer, em vez de atualizar a interface com os resultados parciais, desejemos comunicar ao usuário que um dos seus amigos acaba de fazer o *login* na aplicação.