4
Ambiente

Neste capítulo descrevemos a implementação da abordagem para inserção de transições suaves em aplicações hipermídia. O ambiente definido dá suporte às etapas de Modelagem de Interfaces, Modelagem de Transições e Interpretação das Especificações de Transições, da abordagem proposta.

Como identificado pela abordagem, existem diversas propriedades relativas à forma de apresentação das interfaces e animações que são definidas de acordo com a tecnologia utilizada e o ambiente implementado. Sendo assim, definimos que o ambiente será apresentado na forma de documentos HTML, especificando assim como serão representadas as interfaces concretas do sistema. Definimos ainda que as animações serão implementadas utilizando a tecnologia *Javascript*, em conjunto com o DOM (*Document Object Model*) e o CSS (*Cascading Style Sheets*), um conjunto de tecnologias que define o DHTML (Dynamic HTML). O ambiente de animação exigirá, então, uma implementação das primitivas de animações sobre esta tecnologia para a definição dos efeitos de animação disponíveis para especificação.

Nosso ambiente estende um ambiente de desenvolvimento de aplicações hipermídia denominado HyperDE (Nunes, 2005), que combina um *framework* MNVC (*Model-Navigational-View-Controller*) e um ambiente de desenvolvimento rápido, possibilitando a geração de protótipos de aplicação em um curto espaço de tempo. As aplicações são elaboradas utilizando a abordagem OOHDM/SHDM, gerando uma aplicação conforme a sua modelagem especificada. Este sistema utiliza a linguagem de programação Ruby e é desenvolvido sobre um framework para web, denominado Rails.

Uma visão geral do ambiente e do framework MNVC é apresentada no diagrama apresentado pela Figura 26 a seguir:

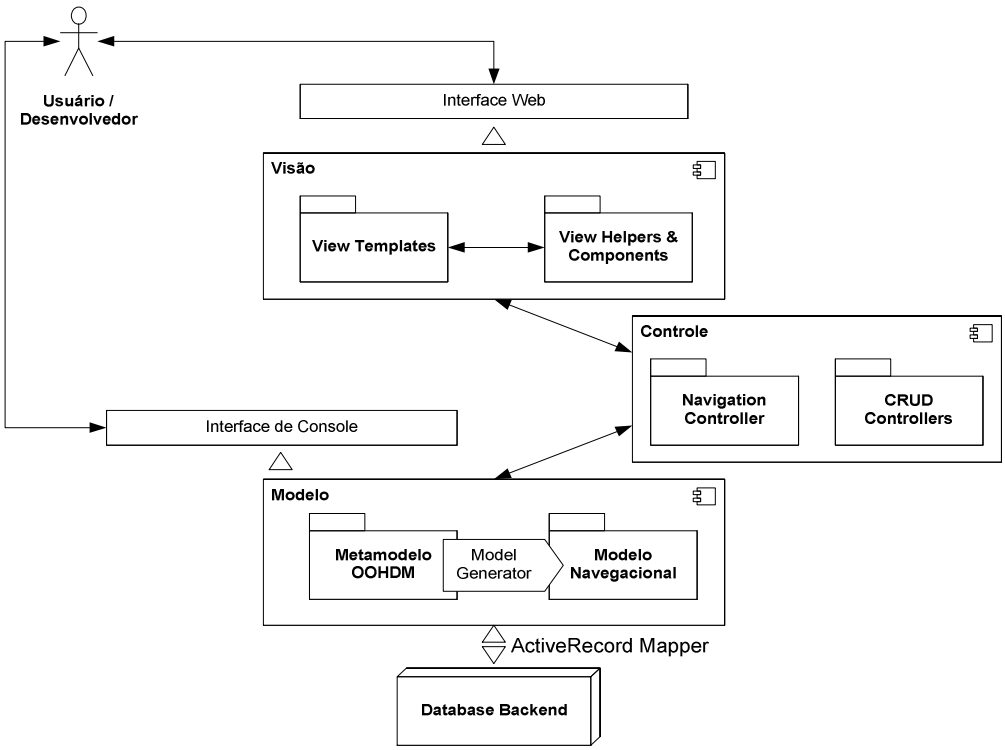

Figura 26 – Componentes da arquitetura do HyperDE (Nunes, 2005)

O framework referido como MNVC do HyperDE é baseado no padrão MVC (Model-View-Controller), que separa a camada responsável pela representação do modelo de dados da aplicação da camada responsável pela visualização desse modelo pelo usuário. É elaborada ainda uma terceira camada, denominada Controlador, entre as camadas de Modelo e de Visão, com o objetivo de coordenar os eventos disparados pela interação do usuário com a interface da aplicação (Nunes, 2005). A adição da letra "N" no acrônimo MVC significa "navegacional", indicando que estamos tratando também do modelo navegacional, construído sobre as primitivas dos métodos OOHDM/SHDM.

Para possibilitar a inserção e execução de transições suaves aos protótipos gerados pela aplicação, realizamos a extensão do ambiente, possibilitando as seguintes funcionalidades:

- Especificação de Interfaces Abstratas;
- Geração de Interfaces Concretas a partir da especificação de Interfaces Abstratas;
- Especificação das Transições suaves;

- Interpretação das Animações que compõem as transições definidas, durante o tempo de execução da aplicação.

É importante ressaltar que o ambiente gerará apenas protótipos de aplicações hipermídia, uma vez que a dinamicidade da navegação durante a interação é limitada, devido à constante introspecção sobre a base de dados do ambiente durante a execução. Isto inviabiliza o uso das aplicações desenvolvidas com a ferramenta de forma produtiva.

4.1. Arquitetura

A arquitetura da abordagem implementada pode ser dividida em dois módulos, que permitem a realização das etapas definidas no capítulo anterior, conforme descrito:

- Módulo de Interface: Dá suporte à etapa de Modelagem de Interfaces, permitindo a especificação das interfaces abstratas e concretas e realizando o mapeamento entre elas, gerando assim, a interface final do sistema.
- Módulo de Transições: Suporta a etapa de Modelagem de Transições, possibilitando a realização da especificação das transições suaves. Suporta ainda a etapa de Interpretação das Especificações de Transições, interpretando as animações durante a execução do sistema elaborado.

A partir das especificações da interface será gerada, pelo primeiro módulo, a interface concreta, apresentada ao usuário durante a execução do sistema. Essa interface será composta por diversos *widgets*, definidos conforme descrito pela seção anterior. Caso este *widget* seja do tipo *SimpleActivator* e esteja indicando uma navegação para o qual está definida uma transição suave for seja ativado durante a interação, ele acionará a interpretação das transições especificadas apresentando um conjunto de animações para o usuário, a partir do módulo de Transições.

Na versão original do HyperDE, sobre o qual o ambiente foi extendido, não há suporte para elaboração de interfaces abstratas, nele as interfaces concretas

são diretamente definidas através das views, que indicam instâncias da classe *View* do sistema. Nela, as interfaces concretas customizadas podem ser definidas pelo usuário para cada combinação de classe em contexto, índices ou visões genéricas reutilizáveis. Tais visões são apresentadas pelas ações *context* e *show_index* do controlador navegacional, que buscam a instância definida para contextos ou índices, respectivamente. Já as visões genéricas são apresentadas como layout desses tipos de visões (Nunes, 2005).

Realizamos em nossa implementação uma extensão do sistema HyperDE, possibilitando adicionar as funcionalidades presentes nos dois módulos descritos, conforme representado pela Figura 27 a seguir.

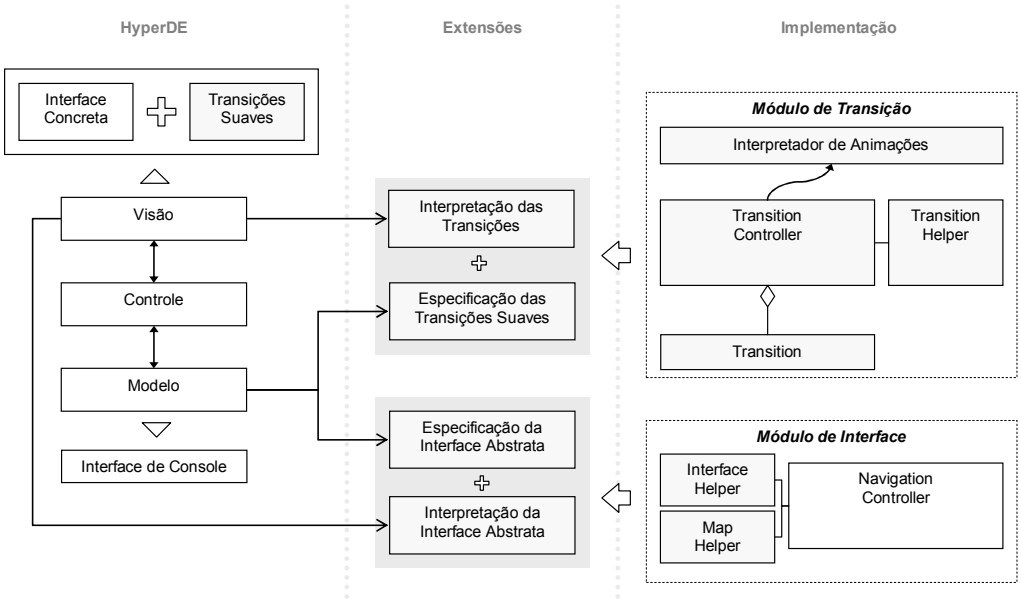

Figura 27 – Arquitetura do ambiente desenvolvido

Como se pode observar pelo esquema, no módulo de interfaces, nós estendemos a camada de modelo e de visão do sistema, possibilitando a especificação da interface abstrata e a interpretação desta, definindo uma interface concreta. Esta extensão é realizada através dos módulos denominados *Helpers*, que permitem definir métodos específicos para apresentação de informações, simplificando a especificação de uma visão. Criamos então, um módulo auxiliar de interfaces denominado *Interface Helper*, que funciona como uma camada de abstração para definição da interface, assim como *Map Helper,* que permite a definição da ontologia de *widgets* concretos.

No módulo de transições, nós estendemos o modelo permitindo a definição de instâncias de transições. Estendemos também as visões do sistema, possibilitando a interpretação das transições suaves. Essa extensão é implementada através da adição de uma classe *Transition* e de um controlador de transições, associado a um módulo auxiliar de transições, denominado *Transition Helper*.

4.2. Módulo de Interfaces

Como descrito na abordagem, antes de realizar a definição das transições precisamos identificar quais interfaces irão participar da transição e quais *widgets* serão apresentados em cada das interfaces. Neste módulo possibilitamos então a realização da especificação e interpretação das interfaces estendendo a ferramenta HyperDE para dar suporte à definição de interfaces abstratas e seu mapeamento para interfaces concretas.

No sistema, a customização das visões é realizada através da definição do atributo *template* da *View*, descrevendo a interface concreta através de uma mistura de códigos HTML e trechos de códigos Ruby (delimitados pelos símbolos "<%=" e "%>") que determinam funções de auxílio de interface. Estendemos tais funções auxiliares, reunindo-as em uma biblioteca na camada de interface do *framework*. Através dessas funções, possibilitamos a construção das interfaces baseadas na etapa de interfaces abstratas da abordagem OOHDM/SHDM.

4.2.1.Especificação de Interfaces Abstratas

Para a realização da especificação conforme apresentado pela abordagem, possibilitamos a definição dos *widgets* através da ontologia de *widgets* abstratos e da ontologia de interfaces concretas. Esse processo aproxima o processo de modelagem da interface na ferramenta ao processo de modelagem de interfaces abstratas, proposta na metodologia. Essa modificação permite ainda facilitar a identificação dos *widgets* para posterior especificação das transições uma vez que cada elemento deverá ser definido relacionando-os com os elementos do modelo navegacional.

A alteração se dará no processo de especificação do atributo *template*, estendendo a forma de descrição da interface. Nesta especificação definiremos a

interface abstrata, ao invés da interface concreta, como é realizada na ferramenta HyperDE. Para tal, foram criados uma biblioteca de funções de interface que, como na biblioteca de funções auxiliares, permite a construção das interfaces baseadas nas primitivas do modelo navegacional.

Conforme descrito no capítulo 2, os widgets que compõem a interface abstrata poderão ser de quatro tipos distintos: os ativadores (*SimpleActivator*), exibidores (*ElementExhibitor*), capturadores (*ValueCapturer*) e compostos (*CompositeInterfaceElement*). Definimos então na biblioteca auxiliar, representada pelo módulo *InterfaceHelper*, um conjunto de quatro funções que representam esses tipos de elementos.

Descrevemos adiante cada uma dessas funções, porém, é interessante definir antes alguns parâmetros que são comuns a todos os tipos de funções:

- *name*: Indica uma *string* para identificação do *widget*;
- *maps_to*: Definido como uma string, esse parâmetro representa a propriedade *MapsTo* na abordagem, indicando o *widget* concreto da ontologia de interfaces concretos sobre o qual o *widget* abstrato será mapeado. Caso este parâmetro não seja especificado, o *widget* será mapeado para um objeto *default*, especificado na ontologia.
- *value*: Representa as propriedades *fromElement*, *fromAttribute* e *Value* na modelagem de transições, indicando o objeto e o atributo do modelo navegacional representado e o item de informação que será exibido pelo *widget*. A função identifica as propriedades em um único parâmetro através da análise do tipo de objeto passado como parâmetro, conforme indicado pela Figura 28.

Tipo de Dado	Elemento Navegacional	Atributo Navegacional	Valor exibido
String	Indefinido ou herdado pelo último elemento definido.	Indefinido	String passada no parâmetro
AuxInput	AuxInput.element	AuxInput.attribute	AuxInput.eval
Outros	Indefinido	Indefinido	Indefinido

Figura 28 – Propriedades identificadas para os widgets de acordo com as informações definidas

Como pode ser observado, utilizamos um objeto auxiliar definido como *AuxInput* para identificar os três possíveis atributos. Quando definimos este objeto

um método verifica o tipo de dado, que poderá ser do tipo *Index* (Representando um índice), *Node* (Representando um Nó) ou *IndexEntry* (Representando uma entrada do índice) buscando suas propriedades e definindo os atributos: *Element*, *Attribute* e *Eval*.

É importante ressaltar que, ao navegarmos para um contexto ou índice, o controlador de navegação irá definir um novo elemento *ThisIndex* ou *ThisNode*, do tipo *AuxInput,* indicando as instâncias acessadas e disponibilizando a estrutura para o uso na especificação da interface. Esses tipos de objetos poderão ser referenciados como um objeto (ex.: *ThisNode*) ou um objeto e um atributo (ex.: *ThisNode.name*).

Descrevemos a seguir as funções que representam os *widgets* abstratos, definidos pela biblioteca auxiliar.

*Composite***:** Representa elementos do tipo *CompositeInterfaceElement*. Nela definimos um *widget* abstrato que é composto por outros *widgets*. Esta função define um elemento de bloco, devendo ser terminada com "<% end %>", após a definição dos seus elementos internos. A função apresenta os seguintes parâmetros adicionais:

- *repeat*: Corresponde a propriedade *isRepeated* na abordagem e indica se as informações que compõem o elemento serão repetidas, realizando então uma iteração sobre o elemento do parâmetro *Value* da função. Para cada iteração encontrada será definida uma instância *ThisEntry* do objeto *AuxInput*, referenciando o objeto para possibilitar a definição de elementos relacionados.
- *limit*: Indica o número de vezes que as informações contidas no elemento se repetirão, limitando assim o número de iterações realizadas sobre o conjunto de elementos definido.
- *block*: Este parâmetro representa a propriedade *HasInterfaceElement* na abordagem, sendo especificado implicitamente, através da estrutura hierárquica apresentada. Ele define uma estrutura de bloco, permitindo identificar qual o conjunto de elementos que compõem a função (Definidos entre o início e o fim da função).

Exhibitor: Esta função representa elementos do tipo *ElementExhibitor*. Nela definimos um *widget* abstrato que apresenta um dado de informação na forma de um *string*, imagem, url ou email. Esta função não possui qualquer parâmetro adicional.

Capturer: Esta função representa elementos do tipo *ValueCapturer*. Nela definimos um *widget* abstrato que captura alguma informação do usuário. Esta função não foi inteiramente implementada no ambiente, devendo então ser especificada através de códigos HTML.

Activator: Representa elementos do tipo *SimpleActivator*. Estes elementos irão definir uma navegação específica no momento de sua ativação. Sendo assim, este tipo de elemento é o único que poderá acionar uma transição, estabelecendo a relação com o módulo de transições, conforme descrito posteriormente. Para esta função temos o seguinte parâmetro adicional:

- *target_url:* Este parâmetro permite especificar um endereço *url*, fazendo com que ocorra redirecionamento para um endereço específico durante a sua ativação. Quando este parâmetro não é especificado, a função encontrará o estado de navegação destino associado ao elemento do parâmetro *Value.*

Exemplificamos a especificação de uma interface na Figura 29, que representa um contexto do tipo Filmes por ordem Alfabética.

```
<%composite "MoviesAlpha", :repeat => false, :value => @ThisNode do%>
	<%composite "MovieInfo", :repeat => false, :value => @ThisNode do%>
		<%composite "movie_picturecomposite", :repeat => false, :value => @ThisNode.picture do%>
			<%=exhibitor "movie_picture", :value=> @ThisNode.picture%>
		<%end%>
		<%composite "movie_namecomposite", :repeat => false, :value => @ThisNode.name do%>
		  <%=exhibitor "movie_namelabel", :value=> "Name:"%>
		  <%=exhibitor "movie_name", :value=> @ThisNode.name%>
		<%end%>
		<%composite "movie_descriptioncomposite", :repeat => false, :value => @ThisNode.description do%>
			<%=exhibitor "movie_descriptionlabel", :value=> "Description:" %>
			<%=exhibitor "movie_description", :value=> @ThisNode.description%>
		<%end%>
	<%end%>
	<%composite "movie_websitecomposite", :repeat => false, :value => @ThisNode.website do%>
		<%=exhibitor "movie_websitelabel", :value=> "Website:" %>
		<%=exhibitor "movie_website", :value=> @ThisNode.website%>
	<%end%>
	<%composite "movie_directorscomposite", :repeat => false, :value => @ThisNode.directors do%>
		<%=exhibitor "movie_directorslabel", :value=> "Directors:" %>
		<%composite "movie_directors", :repeat => true, :value => @ThisNode.directors do%>
			<%=activator"movie_directorname", :value=> @ThisEntry.director%>
		<%end%>
	<%end%>
	<%composite "movie_genrescomposite", :repeat => false, :value => @ThisNode.genres do%>
		<%=exhibitor "movie_genreslabel", :value=> "Genres:"%>
		<%composite "movie_genres", :repeat => true, :value => @ThisNode.genres do%>
			<%=activator"movie_genrename", :value=> @ThisEntry.genre%>
		<%end%>
	<%end%>
	<%composite "movie_studioscomposite", :repeat => false, :value => @ThisNode.studios do %>
		<%=exhibitor "movie_studioslabel", :value=> "Studio:"%>
		<%composite "movie_studios", :repeat => true, :value => @ThisNode.studios do%>
			<%=activator "movie_studioname", :value=> @ThisEntry.studio%>
		<%end%>
	<%end%>
	<%composite "movie_actorscomposite", :repeat => false, :value => @ThisNode.actors do%>
		<%=exhibitor "movie_actorslabel", :value=> "Actors:"%>
		<%composite "movie_actors", :repeat => true, :value => @ThisNode.actors do%>
			<%composite "movie_actorrolecomposite", :repeat => false, :value => @ThisEntry.actor do%>
				<%=activator "movie_actorname", :value=> @ThisEntry.actor%>
				<%=exhibitor "movie_actorrole", :value=> @ThisEntry.role%>
			<%end%>
		<%end%>
	<%end%>
	<%composite "ContextIndex", :repeat => false, :value => @context.index do%>
		<%= exhibitor "index_title", :value => @content_for_title %>
		<%composite "index_composite", :repeat => true, :limit => 20, :value => @context.index do%>
			<%= activator "index_node", :value=> @ThisEntry.item %>
		<%end%>
	<%end%>
<%end%>
```

Figura 29 – Especificação de interface abstrata para o contexto filmes por ordem alfabética

Como pode ser observado na definição acima, podemos definir na interface elementos do tipo *ExhibitorElement*, associando uma string para o seu parâmetro *Value*, sem referenciar uma instância do modelo navegacional. Nestes casos, o widget não será associado ao modelo, sendo utilizada para complementar o layout ou exibir algum dado adicional na *View*. Na especificação apresentada (Figura 29) definimos, por exemplo, um elemento do tipo *ExhibitorElement* denominado *MovieWebsiteLabel* que tem como valor a string “Website:”. Embora este widget não apresente uma associação definida, ele representa um rótulo para o widget denominado *MovieWebsite*. Supondo que a instância navegada não possui esta informação, o rótulo seria exibido da mesma forma, quando o designer poderia desejar que os rótulos só fossem apresentados quando associados às informações disponíveis.

Para possibilitar essa opção, definimos no objeto *AuxInput*, o método *Exists*, que verifica se o elemento indicado possui um valor associado durante a

navegação. Este método pode ser empregado de acordo com o exemplo da Figura 30 a seguir:

```
...
<%composite "movie_websitecomposite", :repeat => false, :value => @ThisNode.website do%>
       <%=exhibitor "movie_websitelabel", :value=> "Website:" if @ThisNode.website.exists%>
       <%=exhibitor "movie_website", :value=> @ThisNode.website%>
<%end%>
...
```

Figura 30 – Exemplo de utilização do método *Exists*

4.2.2. Identificação dos *Widgets*

Como já foi descrito, precisamos saber quais instâncias ou nós os *widgets* representarão durante a navegação, para assim, possibilitar a análise das condições apresentadas durante a execução e então determinar qual função de animação será executada. Esta identificação só poderá ser realizada na interpretação das especificações da modelagem de interface realizada, durante a execução do sistema.

Para identificar os elementos propomos uma notação composta pelos itens a seguir:

- Nome: Nome especificado pelo parâmetro correspondente na especificação;
- Classe: Nome da classe que o elemento representa no modelo navegacional;
- Instância: Identificação (Id) da Instância ou nó representado durante a navegação;
- Atributo: Nome do atributo que o elemento representa no modelo navegacional.
- Identificador único (UID): Uma identificação numérica única, para assegurar representações únicas para cada elemento.

Note que os dados Nome e Atributo são obtidos pela especificação da interface abstrata, para o UID geramos um inteiro, que é incrementado a cada *widget* apresentado. Para obtenção dos outros dados, precisamos identificar os elementos do modelo referenciados durante a execução bem como as instâncias representadas, obtendo então as propriedades referentes aos dados solicitados.

Para possibilitar acesso a esses dados durante a execução, bem como apresentar os elementos com uma identificação própria, optamos por associar as informações aos *widgets* concretos na interface descrita pelo documento HTML. Esta identificação pode ser realizada através dos atributivos dos objetos HTML, *id* e *class*, que são identificadores de elementos. Associamos então os dados aos *widgets* concretos conforme apresentado a seguir (Figura 31):

```
ID = UID + Classe + Instância + Atributo
CLASS = Nome do Widget
```

Figura 31 – Propriedade de identificação dos *widgets*

Exemplificamos na Figura 32 a seguir uma identificação obtida a partir da especificação apresentada.

```
<%=activator "movie_actorname", :value=> @ThisEntry.actor%>
<%=exhibitor "movie_actorrole", :value=> @ThisEntry.role%>
```

```
<div class='movie_actorname' id='00037-Actor-o_BrunaDiTullio_f7f3-actor'>
       <a href="#">Bruna Di Tullio</a>
</div>

<div class='movie_actorrole' id='00038-Role-o_CountessLusaValadas_6188-role'>
       Countess Luísa Valadas
</div>
```

Figura 32 – Especificação e identificação obtida no *widget* concreto

4.2.3. Mapeamento para Interfaces Concretas

Com a especificação da interface abstrata realizada, podemos mapear os elementos para *widgets* concretos definidos na ontologia de interfaces concretas. Em nossa implementação representamos esta ontologia através de uma biblioteca auxiliar, definida pelo módulo *InterfaceHelper*.

Para o mapeamento, definimos uma função *Maps*, que associa ao widget abstrato um widget concreto, conforme definido pela especificação da interface abstrata. Esse mapeamento é realizado durante a interpretação da interface, sendo definido como a saída de cada função que especifica um elemento abstrato. Como parâmetro dessa função, temos os dados que identificam o nome do widget abstrato escolhido e o parâmetro *Meta*, que representa os metadados que descrevem as informações de identificação (*id* e *class*) do elemento, conforme especificado no item anterior.

Para a representação do conjunto de *widgets* concretos, definimos no módulo uma especificação dos possíveis *widgets* concretos, identificando quais *tags* deverão ser definidos no início e no final do elemento, representando assim um objeto HTML. Devemos definir também qual será o objeto padrão, denominado *default_mapping*, que representa o *widget* para qual o objeto será mapeado caso a propriedade *MapsTo* do elemento abstrato não tenha sido definida.

Descrevemos na Figura 33 a seguir, um conjunto de *widgets* concretos definidos no módulo para especificação de *tags* que correspondem a uma tabela. Perceba que as definições apresentam, dentre as *tags*, uma informação descrita como *meta*, que indica onde será inserido a descrição de identificação do elemento.

```
def table
    {"start" => "<div metha><table border='1'><tbody>",
    "end" => "</tbody></table></div>"}
  end

  def first_column
    {"start" => "<tr><td><div metha>",
     "end" => "</div></td>"}
  end

  def column
    {"start" => "<td><div metha>",
     "end" => "</div></td>"}
  end

  def last_column
    {"start" => "<td><div metha>",
     "end" => "</div></td><tr>"}
  end

  def row
    {"start" => "<tr><td><div metha>",
     "end" => "</div></td><tr>"}
  end
```

Figura 33 – Ontologia de *widgets* concretos implementada

Em nosso ambiente, observamos que os elementos podem ser comumente representados por estruturas de blocos <DIV> utilizando então propriedades CSS para definição de estilos. Para facilitar esta opção, definimos como *default_mapping* a estrutura abaixo (Figura 34), onde temos estruturas de bloco para o widget concreto padrão.

```
def default_mapping
   {"start" => "<div metha>",
   "end" => "</div>"}
end
```

Figura 34 – *Widget* concreto para mapeamento padrão

O conjunto de *widgets* concretos especificado pode sempre ser estendido de acordo com a necessidade de representação, podendo até mesmo definir *widgets* concretos específicos para outras tecnologias, tal como XAML ou Flash.

É importante ressaltar que com a identificação dos *widgets*, podemos fazer uso extensivo de CSS, podendo representar assim, diversas interfaces concretas a partir de uma mesma especificação e mapeamento.

4.2.4. Ativação das Transições

Na função *Activator,* responsável pela definição dos elementos ativadores, ativamos as transições. Durante a definição da ação do ativador identificamos se irá ocorrer uma transição de forma suave ou de forma estática. Desta forma, podemos dizer que, quando o *widget* representado por esse tipo de função, for ativado pelo usuário na interação, ocorrerá uma comunicação com o módulo de transições, estabelecendo a ligação entre os dois módulos da arquitetura. O modo como ocorrerá esta ligação estará definido adiante no documento.

No momento da interpretação da interface, as funções do tipo Activator determinarão se as transições identificadas terão uma especificação definida na especificação de transições suaves. Essa identificação é realizada pela função auxiliar *ReviewAnchor*, que compara a transição identificada pelo *widget* com as transições especificadas na especificação de transições. Tal comparação é feita sobre as propriedades *SourceInterface* e *TargetInterface*, que identificam uma transição. Após a realização desta comparação, temos os seguintes comportamentos definidos:

- Se o elemento não apresenta especificação, definimos a âncora apontando para o controlador de navegação, associando-o às ações dessa classe, determinadas pelo elemento, sem qualquer alteração sobre o método.
- Se o elemento apresenta uma especificação, definimos uma âncora remota que aponta para o controlador de transições, associando-o à ação *Animate* deste controlador e indicando que ocorrerá uma transição suave especificada, após sua ativação.

4.3. Módulo de Transições

Este módulo é responsável pelas transições suaves, possibilitando a sua especificação pelo desenvolvedor, assim como a interpretação das animações que compõe as transições durante a interação do usuário com o sistema. Para possibilitar essas duas funcionalidades estendemos a ferramenta HyperDE, assim como realizado para o módulo de interfaces.

Para representação das transições foi definida, no sistema, uma nova classe denominada *Transition*, onde as instâncias de transições suaves poderão ser criadas e especificadas. Definimos mais adiante quais os atributos e características desses objetos.

A execução das transições suaves se dará durante o processo de interação, após a ativação de um *widget* que define uma ação de transição e que possui uma especificação associada. Essa interpretação será realizada pelo controlador de transições (*TransitionControler*), através da ação *Animate*, que por sua vez, faz uma comunicação com um interpretador *Javascript*, executando o conjunto de animações definidos para os elementos de interface. Este interpretador é descrito em detalhes na próxima seção.

4.3.1. Especificação das Transições Suaves

Para a especificação das transições o desenvolvedor deverá criar as instâncias da classe *Transitions*, inserindo as informações que definem a transição. A classe de transição possui os atributos descritos a seguir, que são definidas para cada instância existente:

- *source_view_id*: Este atributo corresponde à propriedade *SourceInterface* na modelagem das transições. Nele indicamos qual a *view* que define a interface origem da transição, através do identificador id, correspondente.
- *target_view_id*: Corresponde à propriedade *TargetInterface* na modelagem das transições. Nele indicamos qual a *view* que define a interface destino da transição, através do identificador *id*, correspondente. Este atributo, junto ao atributo, *source_view_id* identifica a transição.
- *animation*: Este atributo corresponde à propriedade *AnimationFunctions* da modelagem. Nele definimos, através de um texto, a especificação das

animações que irão compor a transição. Ao ativarmos a transição este texto será interpretado, executando as animações definidas.

- *rhetoric*: Corresponde à propriedade *RhetoricalStructure* do modelo, onde definimos, através de um texto, a especificação da estrutura retórica da transição. Essa definição é interpretada durante a ativação da animação, criando estruturas associada às animações definidas.

Assim como as demais entidades do sistema, a classe de transições herda as ações ("list", "new", "edit",etc) da classe abstrata *CrudController*, que realiza o gerenciamento da entidade no sistema, com *templates* de visão customizáveis. Sendo assim, o gerenciamento das transições pela interface Web do ambiente, se assemelha aos demais gerenciamentos definidos para as demais classes presentes. Ele será realizado inicialmente através da listagem em ordem alfabética das transições já definidas, indicando com ancoras as opções para editá-los ou excluí-los. Ao final da lista é apresentada a âncora "*Add New Transition*", que possibilita a criação de uma nova instância de transição. Ilustramos a interface que apresenta essa lista, na Figura 35

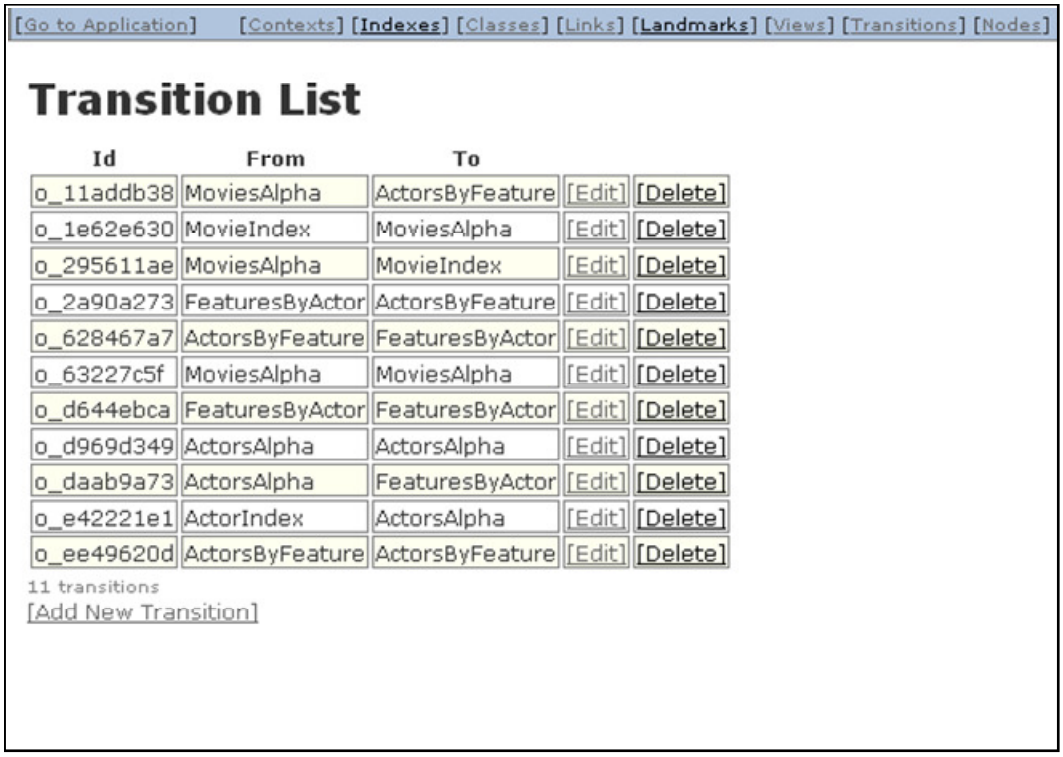

Figura 35 – Tela que representa a listagem das transições existentes

Durante a edição ou criação da transição, devemos informar os dados que correspondem aos atributos da classe. A interface que apresenta essa ação esta ilustrada pela Figura 36 abaixo.

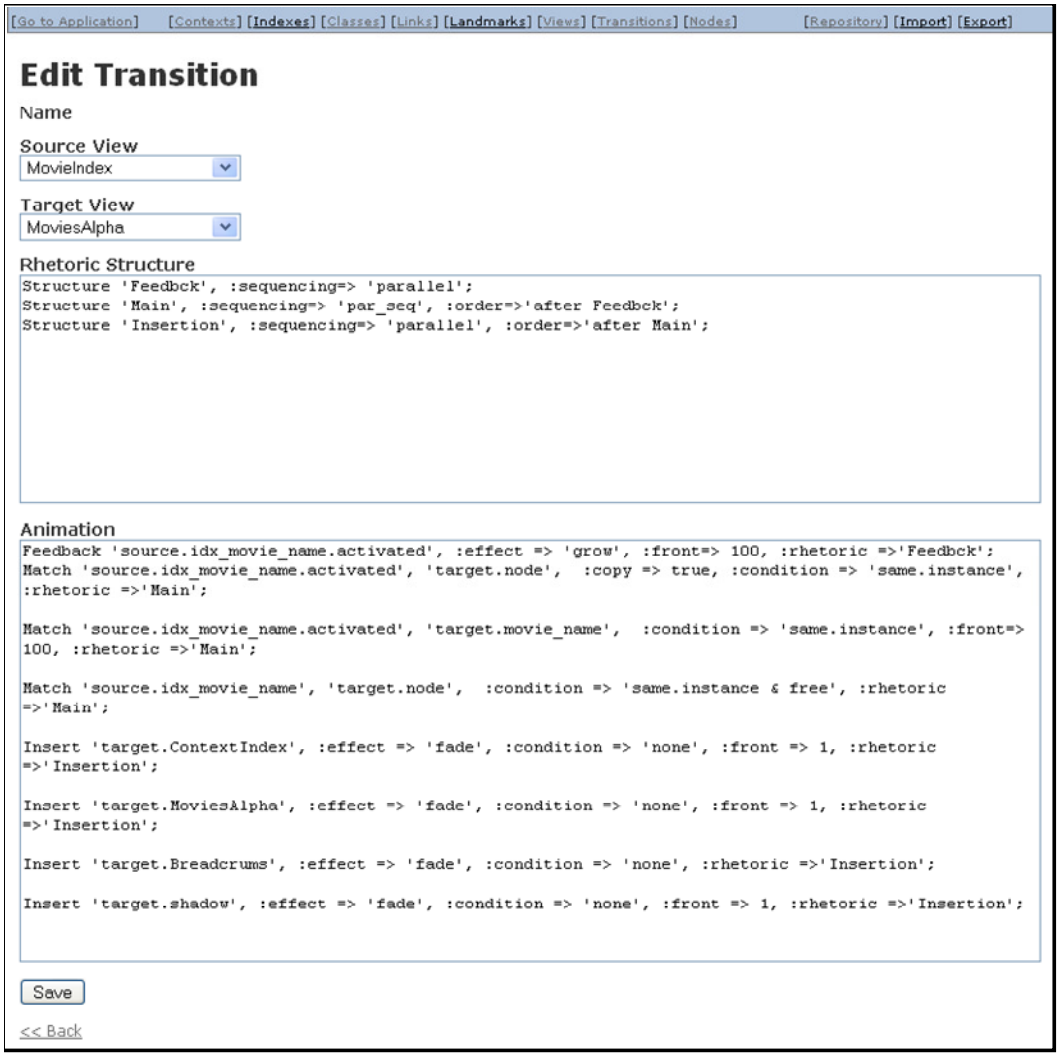

Figura 36 – Tela que representa a edição de uma transição suave

4.3.1.1. Especificação das Animações

A especificação das animações é realizada através da elaboração de um texto que define a propriedade relacionada para a transição suave. Esse texto deverá conter uma descrição das animações que ocorrerão durante a transição, sendo interpretado durante a execução desta.

Como forma de representar as possíveis animações definimos uma biblioteca de funções auxiliares para o controlador de transições, implementada no módulo auxiliar *TransitionHelper*. Neste módulo estarão definidas as funções de

animação**: *Insert, Remove, Match, Trade* e *Emphasize***. Representando todos os tipos de animação, conforme definido pela abordagem.

Durante a interpretação, o texto onde está descrito o conjunto de funções de animação será avaliado, executando cada função encontrada. Na execução destas, será comunicada ao interpretador cada animação que deverá ser executada.

Para cada uma dessas funções descritas, precisamos ainda identificar quais parâmetros deverão ser especificados. Descrevemos a seguir quais são eles:

source: Corresponde ao elemento origem na propriedade *Elements* da modelagem. Nesse parâmetro referenciamos o *widget* de origem que irá participar da animação. Para indicar o *widget*, indicamos uma string que obedece a uma notação específica. Nela devemos definir o estado a que o *widget* pertence (*source* ou *target*), seguido por um ponto (".") e depois pelo nome do *widget* definido na especificação de interfaces (ex. source.name). Podemos ainda indicar o *widget* ativado na interação, adicionando o texto ".activating" ao fim da referência (ex. source.nome.activating).

target: Corresponde ao elemento destino na propriedade *Elements* da modelagem. Nesse parâmetro referenciamos o *widget* destino que irá participar da animação. A referência do *widget* é realizada da mesma forma do parâmetro anterior (*source*).

condition: Corresponde às três condições de animação descritas na especificação de animações de transição. Esse parâmetro é definido por um string que pode ser escrito conforme as opções a seguir, ou pela combinação dessas opções, utilizando o caractere "&".

- *same*: Indica que os *widgets* de origem e destino deverão ter os mesmos dados definidos pela sua identificação. Como a identificação dos *widgets* considera mais de uma informação, definimos uma notação onde podemos representar as opções a seguir, estendendo o texto com um ponto "." e com a opção desejada (ex. same.instance).
    - *Class*: Indica que os widgets deverão ser da mesma classe no modelo;
    - *Instance*: Indica que os *widgets* deverão representar uma mesma instância.

- *Attribute*: Indica que os *widgets* deverão representar o mesmo no modelo;

- *free*: Indica que o *widget* referenciado na animação não foi utilizado anteriormente na especificação para qualquer outra animação.
- *noCorrespondent*: Utilizado em *widgets* que compõe um widget composto, indica se o elemento não possui um widget correspondente na composição que apresenta o *widget* destino.

rhetoric: Corresponde à propriedade *RhetoricalStructure* da modelagem, indicando qual será a estrutura retórica de transição a qual a animação pertencerá. Indicamos a estrutura através de uma string que define o nome de uma estrutura definida.

front: Não corresponde a qualquer propriedade do modelo. É um parâmetro auxiliar elaborado como forma de possibilitar a especificação da ordem de sobreposição dos elementos, redefinindo o valor do atributo de estilo denominado z-index, do objeto HTML. Esse parâmetro poderá ser definido como um número inteiro, ou referenciando outro item da interface, obedecendo a notação composta pelas strings a seguir:

- *above*: Define o parâmetro como o valor encontrado pelo atributo z-index do elemento referenciado mais 1. A notação deverá apresentar essa string seguida de um ponto e do elemento referenciado (above source.name).
- *below*: Define o parâmetro como o valor encontrado pelo atributo z-index do elemento referenciado menos 1. A notação para definição é a mesma da opção anterior. (below source.name).

copy: É um parâmetro específico às funções de animação *Trade* e *Match*, que indica a duplicação do *widget*, antes da realização da transformação, permitindo assim, que o *widget* permaneça inalterado enquanto sua cópia participa de uma animação. Esse parâmetro é definido pelos valores booleanos *true* ou *false*.

interval: Específico à função de animação *Trade*, especifica o intervalo de tempo em segundos em que ocorrerá a alteração do elemento destino, com relação ao destino origem, especificando um intervalo para substituição dos elementos. Esse parâmetro é especificado com de um inteiro, podendo ser positivo ou negativo.

***children*:** Específico à função de animação *Insert*. Em uma animação de inserção de um elemento, indica se seus filhos, indicados pela árvore DOM do documento, também serão inseridos. Esse parâmetro é definido pelos valores booleanos *true* ou *false*.

Temos ainda uma extensão dos parâmetros das animações que permitem a definição da estrutura retórica de estilos da animação, são elas: *duration, effect, source_effect, target_effect*. Descrevemos melhor estes parâmetros no próximo item.

Exemplificamos na Figura 38 uma especificação de animações realizada. Nela apresentamos uma transição do contexto Filmes por ordem Alfabética para o Contexto Atores por Filme (Figura 37).

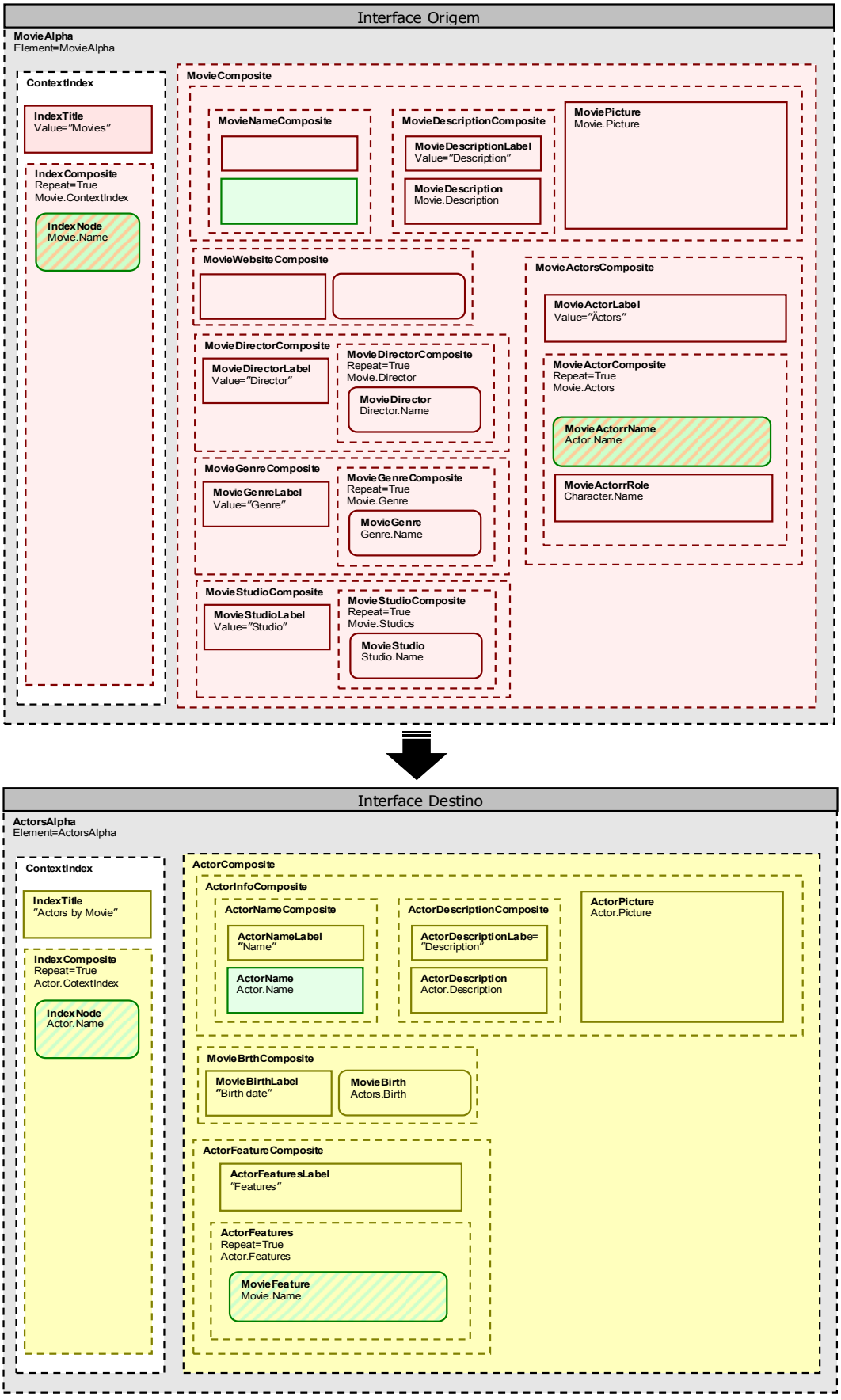

Figura 37 – Interfaces de uma transição entre um índice de atores para o contexto de atores por ordem alfabética

```
Feedback 'source.movie_actorname.activated', :effect =>'grow', :duration => 2, :rhetoric =>'Feedback';

Match 'source.movie_actorname.activated', 'target.actor_name', :condition =>'same.instance & free', :rhetoric =>'Main';
Match 'source.movie_name', 'target.actor_feature', :condition => 'same.instance & free', :rhetoric =>'Main';
Match 'source.index_node', 'target.actor_feature', :condition =>'same.instance & free', :rhetoric =>'Secondary';
Match 'source.movie_actorname', 'target.index_node', :condition =>'same.instance & free', :rhetoric =>'Secondary';

Remove 'source.MoviesAlpha', :condition =>'free', :effect => 'fade',:rhetoric =>'Removal');
Remove 'source.index_composite', :condition =>'free', :effect => 'fade', :rhetoric =>'Removal');
Remove 'source.index_title', :condition =>'free', :effect => 'fade', :rhetoric =>'Removal');

Insert 'target.ActorsByFeature', :condition =>'free', :effect =>'fade', :rhetoric =>'Insertion';
Insert 'target.ContextIndex', :condition =>'free', :effect =>'fade', :rhetoric =>'Insertion';
```

Figura 38 – Especificação de transição de um índice de atores para o contexto atores por ordem alfabética

4.3.1.2. Especificação da Retórica de Transição

Na especificação da retórica de transição, definimos inicialmente a estrutura retórica da transição para identificação da seqüência de execução das animações. Descrevemos então uma especificação de estrutura retórica na propriedade *rhetoric* da instância de transição criada. Esta descrição é realizada na forma de um texto que será interpretado pelo controlador e *helper* de transições, durante a execução da aplicação.

Assim como na especificação de animações implementada, a representação das estruturas retóricas será realizada através de uma função presente no módulo auxiliar *TransitionHelper* do controlador de transições. A função denominada *Structure*, permite a representação de uma estrutura que será comunicada ao interpretador de animações, durante a interpretação da transição. Definimos as propriedades da função que representa a estrutura retórica a seguir, indicando como a especificação deverá ser definida no texto.

- *name*: Parâmetro correspondente à propriedade *StructureName* na modelagem da transição. Define o nome para identificação da estrutura criada. Este parâmetro deverá ser especificado como um *string* desejado.
    - *sequencing*: Parâmetro correspondente à propriedade de mesmo nome na modelagem da transição. Define a ordenação interna das animações que compõem a estrutura. Este parâmetro poderá ser especificado de acordo com as *strings* a seguir:
        - *parallel* – execução das animações em paralelo.
        - *sequential* – execução das animações seqüencial.

- *par_seq* – execução das animações em paralelo para elementos que representam um mesmo tipo de *widget* e seqüencial entre os *widgets* distintos.

- *order*: Parâmetro correspondente à propriedade de mesmo nome na modelagem da transição. Define a ordem em que será executada a estrutura em questão, com relação às outras estruturas retóricas definidas. Como identificado na especificação na abordagem, este parâmetro poderá ser representado pelas *strings* abaixo, seguidas por um espaço e pelo nome da estrutura desejado.
  - *with* – especificado na forma "with nomedaestrutura", identifica que a estrutura ocorrerá após outra estrutura definida,
  - *after* – especificado na forma "after nomedaestrutura", identifica que a estrutura ocorrerá junto a outra estrutura definida,

Exemplificamos uma especificação de estrutura retórica na Figura 39, abaixo, identificando também a ordem das estruturas.

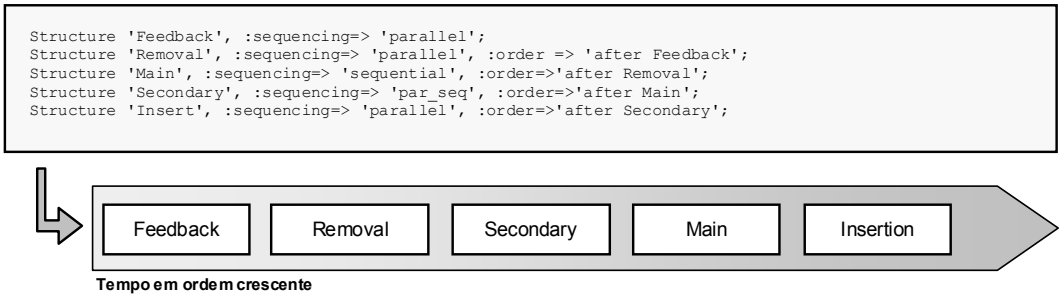

Figura 39 – Especificação de uma estrutura retórica de transição e representação correspondente

A identificação das animações que pertencem a cada estrutura definida será realizada durante a interpretação da transição, quando serão criados os objetos de estrutura no interpretador de animação e executadas as animações definidas, identificando através do parâmetro *rhetoric* a estrutura a qual cada uma pertence.

Para definição dos estilos retóricos da transição, indicando como as animações serão apresentadas para o usuário, estendemos as propriedades de

animação definidas pelo modelo. Com esta implementação a definição dos estilos é feita da forma *inline*, exigindo que a especificação dos estilos retóricos seja realizada na própria especificação da animação. Descrevemos abaixo as duas propriedades adicionais definidas:

- **duration**: Corresponde a propriedade *Duration* dos estilos retóricos definidos pela abordagem. Nela especificamos qual será o tempo de duração da animação em segundos. Esta propriedade deve ser descrita por um inteiro positivo.
- **effect**: Corresponde a propriedade *Effects* dos estilos retóricos definidos pela abordagem. Os efeitos deverão ser descritos na forma de strings tendo um conjunto de efeitos específicos para cada tipo de animação, conforme apresentado pela Figura 40.
- **source_effect**: Específico à função de animação *Trade*. É similar ao parâmetro *effect*, descrito acima, porem, identifica o efeito para o *widget* de origem especificado. As possíveis descrições de efeitos para este parâmetro é idêntica aos efeitos para inserção e remoção.
- **target_effect**: Similar ao parâmetro *source_effect*, descrito acima, porem, identifica o efeito para o widget destino.

Animações	***Insert***	***Remove***	***Emphasize***	***Trade***
Efeitos	*fade* *grow* *blind* *slide_down* *slide_left* *slide_right*	*fade* *puff* *drop* *switch* *shrink* *slideup* *blind* *squish* *fold* *slide_right* *slide_left*	*grow* *shake* *pulsate* *highlight*	Efeitos de ***Insert*** **+** Efeitos de ***Remove***

Figura 40 – Efeitos de animação

É importante ressaltar que a escolha dos efeitos é restrita às capacidades de representação do interpretador de animações. Os efeitos definidos são compatíveis com os recursos oferecidos pelo uso da biblioteca de animações Scirpt.aculo.us (http://script.aculo.us/) no interpretador.

Destacamos no trecho de especificação representada pela Figura 41, a seguir uma especificação *inline* de estilos retóricos.

```
Feedback 'source.movie_actorname.activated', :effect =>'grow', :duration => 2, :rhetoric =>'Feedback';
...
Remove 'source.MoviesAlpha', :condition =>'free', :effect => 'fade', :rhetoric =>'Removal');
...
Insert 'target.ActorsByFeature', :condition =>'free', :effect =>'fade', :rhetoric =>'Insertion';
...
```

Figura 41 – Especificação *inline* dos estilos retóricos

4.3.2. Interpretação das Transições Suaves

Definimos agora como é realizada a interpretação das transições suaves após sua especificação. Essa interpretação se dará durante a execução do sistema, quando uma transição que possui uma especificação definida, ocorre. O diagrama abaixo, representado na Figura 42, apresenta como ocorrerá o processo de execução das animações, relacionando o controlador de transições ao interpretador de animações *javascript* e à interface.

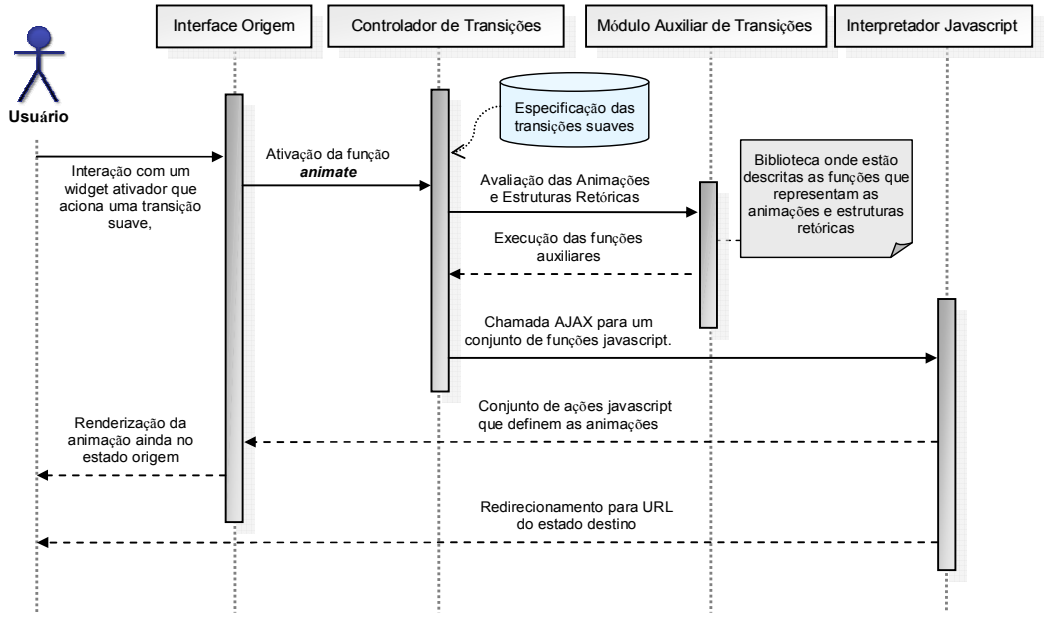

Figura 42 – Diagrama de seqüências representando a execução de uma transição suave

Como pode ser visto no diagrama, a transição suave é ativada a partir de um *widget* ativador específico que executa a ação *animate* no controlador de transições. Durante a ativação desta função são passados alguns parâmetros definidos na interface origem. Essas informações são geradas para cada *widget*

ativador que aciona uma transição suave, durante a interpretação da especificação da interface. Descrevemos os parâmetros a seguir:

- *target_url*: Especifica a *url* do estado final da transição para possibilitar o redirecionamento para o estado destino ao final da execução da animação. Esta informação será comunicada ao interpretador *javascript*.
- *transition_id*: Identifica qual a transição ativada através do seu ID, sendo utilizada para referenciar a transição suave, indicando sua especificação de animações e estruturas retóricas.
- *activated_id*: Identifica qual o *widget* concreto ativado através de seu ID. Esta informação será comunicada ao interpretador *javascript*.

A comunicação do controlador de transições com o interpretador *javascript* é possibilitada na função *animate*, através da ativação de uma nova ação de renderização tipo *update* na página. Renderizações deste tipo permitem a geração de código HTML ou XML através da comunicação assíncrona com o servidor, caracterizando uma chamada AJAX. Associamos essa possibilidade de renderização com a funcionalidade do *Rails*, denominada RJS *Templates*, que permite a geração de código *javascript*, no lugar de HTML ou XML, sendo executado quando retornado ao navegador. Podemos então definir chamadas a funções específicas *javascript* do interpretador de animação.

Além das informações dos parâmetros repassadas para o interpretador, realizamos as seguintes comunicações, definidas por chamadas de código *javascript*:

- Referenciação da página destino: Chamada que permitirá importar o documento HTML, do estado final da interface, permitindo a referenciação dos elementos de interface que o compõem.
- Definição da Estrutura Retórica: Chamada que definirá no interpretador *javascript* um objeto com as possíveis estruturas retóricas, definidos através das funções de estrutura executadas no módulo auxiliar de transição.

- Definição das Animações: Chamada que definirá no interpretador *javascript* um *array* de Animações através das funções de animação executadas no módulo auxiliar de transição.
- Leitura das Transições: Realiza a execução das animações definidas, definindo também a URL para o redirecionamento ao final da transição.

4.4. Interpretação das Animações

A interpretação das animações caracteriza a última etapa para realização das transições suaves. E nessa etapa que ocorrerá a execução das animações de acordo com as especificações realizadas durante a modelagem da transição.

Como já mencionado, o Interpretador foi implementado sobre a tecnologia *javascript* e permite a geração de animações DHTML através da utilização de um conjunto de bibliotecas de interfaces *javascript,* denominado Script.aculo.us. Essa biblioteca estende o framework *Prototype* (Prototype Javascript framework) para elaboração de *web* sites dinâmicos, adicionando dentre outras funcionalidades um framework de animação que permite a realização de diversos efeitos através do uso de primitivas de animação.

Para o funcionamento das animações definimos uma biblioteca de transições *javascript*, que definem todas as funções necessárias para execução de uma transição suave. Essa biblioteca, assim como as bibliotecas do Script.aculo.us, *Prototype* são referenciados em cada pagina HTML gerada, que compõe o protótipo elaborado pelo sistema.

A interpretação é acionada a partir do controlador de transições, que realiza as chamadas *javascript* que serão executadas pelo navegador. A partir das chamadas feitas, executamos então as funções, agrupadas em uma seqüência de cinco etapas, como ilustrado a seguir (Figura 43).

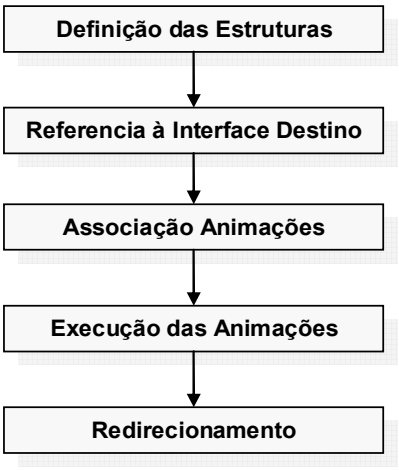

Figura 43 – Seqüência de etapas realizadas pelo interpretador de animações

Descrevemos a seguir as etapas bem como algumas das funções que as compõe.

4.4.1. Definição das Estruturas

Nesta etapa definimos as estruturas que indicam quais animações e quais estruturas retóricas foram definidas. Estas informações são passadas durante a chamada de funções, realizada pelo controlador de transições.

Para definição das estruturas retóricas, é definido um objeto javascript do tipo *Structure,* a partir da função *Structure* do módulo auxiliar de transições. Para as animações será chamada a função *javascript, populate_transitions*, para cada função de animação executada durante a avaliação das especificações no módulo auxiliar de transições. A função indicada irá adicionar ao array *transition_array*, um objeto *javascript* do tipo *Func*, com as animações definidas.

4.4.2.Referência à Interface Destino

Esta etapa corresponde ao processo de importação da página destino, permitindo assim acesso aos *widgets* concretos que a compõem. Este processo é realizado através da função *ImportHTML*, que cria um *iframe* escondido, com o documento do estado destino da transição, no documento da interface origem. Essa etapa é indispensável para permitir a referenciação dos elementos da interface destino.

4.4.3. Identificação das Animações

Nesta etapa definimos algumas propriedades das animações definidas e identificamos através das condições passadas, qual o tipo de animação que irá ocorrer para cada elemento.

Referenciamos os *widgets* que irão compor as interfaces através da execução da função *load_widgets*, para cada animação que compõe a estrutura *transition_array*. Os *widgets* são procurados no DOM, através da identificação do atributo de identificação "*class*", sendo referenciados caso as condições (*same*, *free* e *no_correspondent*) passadas sejam satisfeitas. Quando um elemento for atribuído a uma animação, ele será adicionado à lista *source_list* ou *target_list*, que indica quais elementos foram utilizados em uma animação. Essa adição é necessária para possibilitar verificação da condição *free*.

4.4.4. Execução das Animações

Como o nome indica, nesta etapa é realizada a execução das animações que foram definidas na etapa anterior. A execução é feita a partir da função, *run_actions*, chamada para cada elemento definido na estrutura *transition_array*. Nessa função identificamos também qual a estrutura retórica que cada animação pertencerá, associando-o a estrutura de lista de animações, de acordo com os parâmetros especificados pela estrutura retórica.

As funções de animação serão então chamadas para cada tipo de animação encontrado.

4.4.4.1. Funções de Animações

Descrevemos a seguir o conjunto de funções de animação implementados, identificando alguma de suas características. Estas funções traduzem as funções definidas anteriormente no modelo.

- *Show*: Define uma animação auxiliar, usada por outras funções de animação. Essa primitiva de animação define o aparecimento de um elemento através de um dos possíveis efeitos de entrada. Estendemos ainda um novo tipo de efeito definido como slide_left e slide_right.

- *Hide*: Define uma animação auxiliar, usada por outras funções de animação. Esta animação define o desaparecimento de um elemento através de um dos possíveis efeitos de saída. Como na função definida anteriormente, estendemos as possibilidades de efeito com os efeitos slide_left e slide_right.
- *Enphasize*: Define a animação para ênfase do elemento, ativando um dos possíveis efeitos de ênfase.
- *Match*: Define a animação sobre um elemento, igualando seus parâmetros ao elemento destino. Esta função é realizada identificando o posicionamento absoluto de ambos os elementos e calculando a diferença para utilização na animação *Moveby*, que igualará suas posições. Em seguida são obtidos os estilos do elemento destino, para a animação *Morph,* no elemento origem, igualando assim, os demais atributos de estilo.
- Trade: Define uma animação de troca, onde ocorrerá a substituição de um elemento origem por um novo elemento destino. Nessa função ocorrerá inicialmente a obtenção do posicionamento absoluto (função *AbsolutePos*) dos elementos, calculando então o caminho que separa os elementos e realizando uma animação *MoveBy* no elemento origem, movendo-o até metade do caminho de onde está posicionado o elemento destino. Em seguida o elemento origem que desaparece através da função *Hide* e o novo elemento destino é inserido e aparece com o uso da função *Show*. Por fim, o novo elemento faz uma animação de translação (*MoveBy*) pelo restante do caminho até sua posição final.
- *Remove*: Define a animação para remoção de um elemento através da utilização da função de animação *hide*, especificado acima.
- *Insert*: Define a animação para inserção de um elemento do documento do estado destino da transição. Esta função é executada em dois passos. Primeiro se anexa o elemento ao *Body* do documento origem, localizando e importando suas definições de estilo CSS. Em seguida, caso o parâmetro *deepbool*, seja verdadeiro, indicando que os filhos do elemento também serão importados, far-

se-á uma busca pelos seus elementos filho, verificando se os elementos participam de outra animação. Caso participem, o elemento filho terá o atributo de estilo *visibility* atribuído como "*hidden",* caso não participem seus estilos CSS serão importados. Por fim, o atributo será exibido através da animação *Show*, especificado acima.

Para todas as funções não auxiliares definidas fazemos duas transformações específicas:

- Definimos o atributo de estilo *z-index* do elemento através da função *bringToFront* atribuindo o valor ao parâmetro passado, ou incrementando em um o valor do atributo global front, que é sempre incrementado a cada animação realizada.
- Retiramos o elemento da estrutura DOM original, anexando-o ao elemento *Body* do documento, permitindo assim que as alterações realizadas sobre os elementos pais deste elemento não interfiram sobre os elementos que o compõe e que já participam de outra transformação. Para esta redefinição do elemento na estrutura DOM, mantendo seu posicionamento localizamos seu posicionamento absoluto através da função *AbsolutePos*, atribuindo tais valores depois da redefinição.
- Para assegurar que a estrutura de alinhamento do documento não será afetada com as transformações do elemento, criamos uma cópia de todo elemento que será animado. Definimos então o atributo de estilo *visibility* da cópia como "*hidden*".

4.4.5. Redirecionamento

Esta é a etapa final executada pelo interpretador, quando as animações já foram apresentadas ao usuário. Ela é definida pela função *redirect_to_target*, tendo o parâmetro *page* passado pelo controlador de transições. A etapa consiste no redirecionamento para a *url* do estado destino da transição. Como esse redirecionamento é feito após o término das animações, é esperado que o usuário

já esteja vendo uma interface que corresponde à interface final da transição, tornando a mudança imperceptível.

4.5. Desempenho e Compatibilidade

O desempenho para na interpretação das transições não é ideal devido ao elevado intervalo de tempo exigido para se carregar a interface origem e iniciar a transição, identificando assim, um atraso elevado entre a interação do usuário e a resposta do sistema. A performance durante a execução das funções de animação se mostrou satisfatória, sem apresentar atrasos durante a exibição da animação, defendendo assim, a utilização da biblioteca Script.aculo.us.

Quanto à compatibilidade, a implementação desenvolvida é compatível apenas com Navegadores elaborados pelo grupo Mozilla. A incompatibilidade é resultado das diferentes interpretações, funcionalidades e definições de código javascript.

Por fim, cabe ressaltar que o sistema demonstrou, durante o desenvolvimento, vários *bugs* relacionados à apresentação de elementos sobre estilos CSS, necessitando de diversos ajustes e funções específicas para tratar alguns casos individualmente.