4
Geração Automatizada de Interfaces

A Ontologia de Descrição de Interfaces Ricas, discutida no capítulo 3, lança a base para a especificação de uma completa solução de software para auxiliar o projeto de interfaces (*Computer-Aided User Interfaces Design*), abrangendo desde a etapa de modelagem até a geração dirigida por modelos, além de prover também um ambiente de execução para as interfaces.

Entende-se por interface abstrata a representação, em alto nível, das interações entre o usuário e a aplicação. A interface concreta é a representação dos elementos de interface dependente da tecnologia utilizada. Na solução particular proposta por este trabalho, a plataforma-alvo de geração das interfaces é a tecnologia DHTML (a combinação de XHTML, CSS, DOM e Javascript). Isto funciona como uma restrição tecnológica para a especificação do ambiente de execução das interfaces concretas. Entretanto, o ambiente de modelagem, conforme descrito neste capítulo, permanece agnóstico quanto à plataforma tecnológica. No capítulo seguinte, documentaremos a sua implementação integrado à plataforma do HyperDE.

4.1.
Arquitetura SWUI de Interfaces Ricas

A **Arquitetura SWUI de Interfaces Ricas** descreve os componentes de software e as tecnologias empregadas para a modelagem, geração e execução de interfaces.

Uma vez definida a Ontologia de Descrição de Interfaces Ricas, bem como mecanismos de mapeamento entre a descrição abstrata dos elementos de interface e sua representação concreta, é possível especificar um compilador para a linguagem, capaz de traduzir descrições abstratas em interfaces concretas. A solução completa inclui também uma máquina que interpreta e executa a descrição concreta das interfaces.

A descrição de uma interface abstrata pode conter, entretanto, alguns tipos de referência que não podem ser resolvidos em tempo de compilação. Esta é a situação, por exemplo, quando o projetista especifica que a interface de destino indicada em um *SimpleActivator* não deverá ser carregada como uma nova página HTML, mas deverá compor a interface corrente. As interfaces que incluem outras devem poder ser compiladas de forma independente de suas interfaces componentes, adiando assim, a

resolução de referências do tipo: “o código da interface A deve conter o código da interface B”.

Além disso, visto que a interface concreta é representada por um trecho de código XHTML, se faz necessário formatar este código como um documento HTML válido antes de enviá-lo para o navegador Web do cliente da aplicação. Resolver as composições de interface e formatá-las como documentos HTML são funções realizadas em tempo de execução pelo renderizador de interfaces.

Em resumo, a arquitetura para geração e execução de interfaces RIA possui como componentes principais:

- **Ontologia de Descrição de Interfaces Ricas**: para representação abstrata das interfaces;
- **Gerador de Interfaces Ricas**: compilador de descrição abstrata para código concreto;
- **As linguagens XHTML + RDFa + JSON**: para representação concreta das interfaces;
- **Renderizador de Interfaces Ricas**: responsável por formatar documentos HTML a partir do código das interfaces concretas;
- **Interpretador de Interfaces Ricas**: responsável por executar o código das interfaces concretas;
- **Gerenciador de Transações**: responsável por mediar a comunicação assíncrona entre visão e modelo.

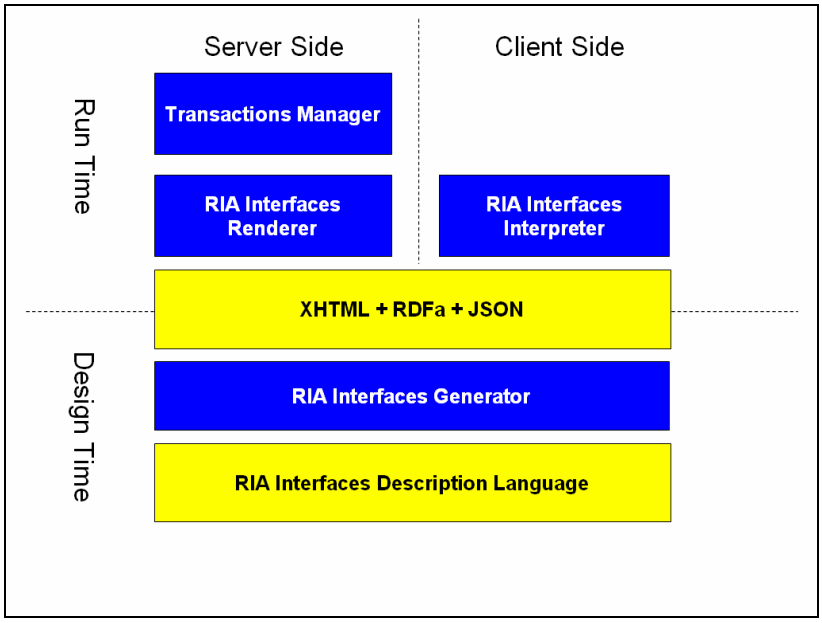

Figura 29 – Arquitetura SWUI de Interfaces Ricas

O Gerenciador de Transações provê o ambiente de execução que dá suporte às interfaces de transição, mencionadas no capítulo 3. As interfaces de transição

comunicam ao usuário o progresso do processamento de uma operação no modelo. A Arquitetura SWUI de Interfaces Ricas implementa esta funcionalidade através do conceito de Transação, que substitui o ciclo tradicional de requisição/resposta entre cliente e servidor por um protocolo de comunicação assíncrona baseado em fila de mensagens.

4.2. Geração de Interfaces Ricas

O gerador de interfaces aceita como entrada a especificação de uma ou mais interfaces abstratas e produz como saída a representação da(s) interface(s) concreta(s) correspondente(s), nas linguagens XHTML + RDFa + JSON. A ferramenta de geração de interfaces é a composição de duas outras ferramentas:

- **Compilador de *Widgets* Abstratos em Concretos**: traduz a descrição da estrutura e do layout da interface em um trecho de código XHTML+RDFa;
- **Compilador de Descrições Retóricas**: traduz a descrição das transições, eventos e decorações em estruturas de dados JSON (*JavaScript Object Notation*)[45].

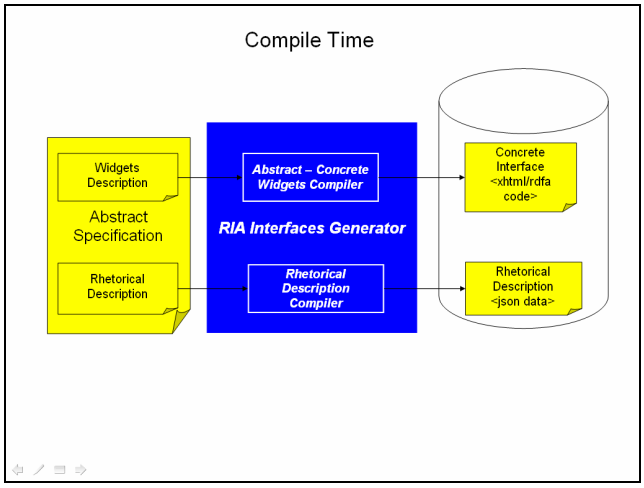

Figura 30 – Geração de Interfaces

4.2.1. Compilação de *Widgets* Abstratos em Concretos

O processo de compilação utiliza as regras de mapeamento abstrato-concreto na especificação dos *widgets* que compõem a interface. Estas regras são indicadas pelas propriedades *mapsTo*, *tagAttributes* e *cssClasses*.

[45] Formato para intercâmbio de dados. (http://www.json.org/)

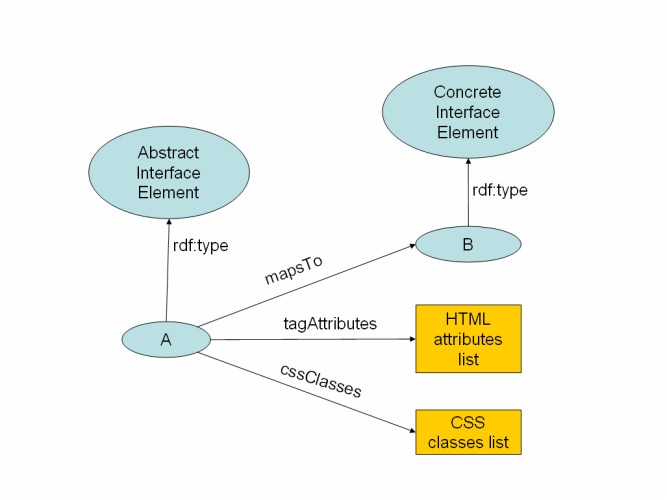

Figura 31 – Propriedades de mapeamento abstrato-concreto

A propriedade *mapsTo* é obrigatória nas instâncias de *AbstractInterfaceElement* e *AbstractInterface*. Quando um *widget* é mapeado em uma instância de *HTMLTag*, o valor da propriedade *legalTag* nesta última é usado para traduzi-lo. Opcionalmente, as propriedades *cssClasses* e *tagAttributes* em *AbstractInterfaceElement* conferem ao projetista maior controle sobre a tradução em código HTML. A primeira permite especificar quais classes CSS definirão o estilo do elemento, e a última, outros atributos para a *tag*.

A tabela a seguir mostra os valores da propriedade *legalTag* nos elementos concretos do tipo *HTMLTag*.

Widgets Concretos	*Tradução HTML*
Button	<input type='button'>
CheckBox	<input type='checkbox'>
ComboBox	<select>
ComboItem	<option>
Composition	
Form	<form>
Header1	<h1>
…	...
Header6	<h6>
Image	<img>
Label	<label>
Link	<a>
ListItem	
OrderedList	
Paragraph	<p>
RadioButton	<input type='radio'>
Table	<table>

Widgets Concretos	*Tradução HTML*
TableBody	<tbody>
TableCell	<td>
TableFooter	<tfoot>
TableHeadCell	<th>
TableHeader	<thead>
TableRow	<tr>
Text	
TextArea	<textarea>
TextBox	<input type='text'>
UnorderedList	

Tabela 3 – Regras de tradução para código HTML

Se o *widget* for mapeado em uma instância de *DHTMLRichControl*, o código utilizado para traduzi-lo será fornecido pela propriedade *htmlCode* de *DHTMLRichControl*. A lógica que define o comportamento do *widget* (propriedade *jsCode*) e as referências aos arquivos externos necessários à sua formatação – folhas de estilo – e funcionamento – scripts (propriedade *dependencies*) não serão incluídas no código da interface concreta, somente no documento HTML renderizado para o cliente.

4.2.2. Anotação Semântica do código XHTML

Existem propriedades dos *widgets* abstratos que não possuem correspondência na linguagem XHTML, mas precisam ser preservadas através do processo de tradução. Este é o caso das propriedades que estabelecem a correspondência entre os objetos de interface e os objetos navegacionais (*Model Binding*): *fromClass* e *fromAttribute*. Quando um *widget* representa um nó navegacional, a informação que ele deverá exibir não pode ser conhecida em tempo de compilação da interface, somente em tempo de execução. Para representar no código da interface concreta o mapeamento com a camada de modelo, a solução adotada é a utilização de RDFa (*Resource Description Framework in attributes*)[46].

Os atributos `@typeof` e `@property` são introduzidos pela especificação RDFa com a finalidade de anotar semanticamente o conteúdo de uma página XHTML. O atributo `@typeof` especifica o tipo de um recurso RDF, ao passo que o atributo `@property` especifica uma propriedade do recurso. [43]

[46] Conjunto de extensões à linguagem XHTML, propostas pelo consórcio W3C, para expressar dados estruturados (http://rdfa.info/) [42].

Para ilustrar o emprego dos atributos RDFa no código das interfaces concretas, considere a seguinte especificação abstrata simplificada (somente mapeamento visão-modelo está indicado):

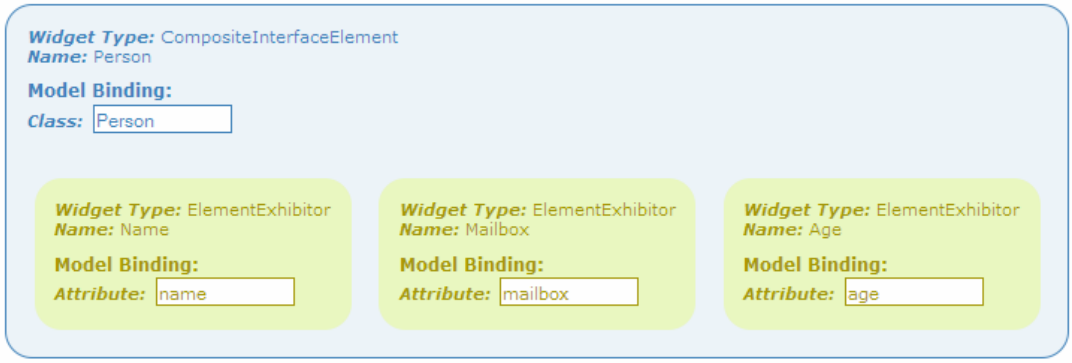

Figura 32 – Especificação abstrata da visualização de um objeto, incluindo o mapeamento visão-modelo

A seguir, o código da interface concreta, exibindo uma instância da classe *Person*, e o respectivo modelo de dados em RDF. As anotações RDFa estão em destaque:

```
<div typeof="Person">
   <p property="name">John Smith</p>
   <p property="mailbox">jsmith@example.com</p>
   <p property="age">27</p>
</div>
```

Figura 33 – Código HTML+RDFa correspondente

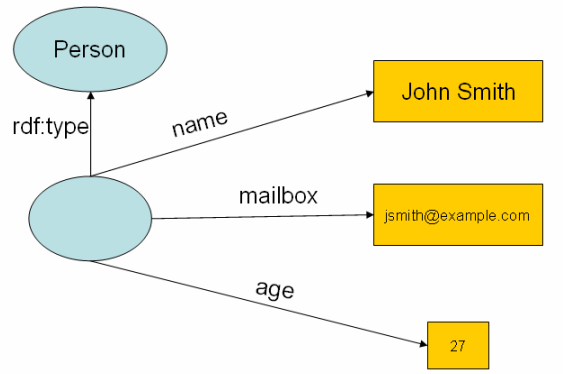

Figura 34 – Modelo de dados RDF correspondente.

Os *widgets* concretos funcionam, na verdade, como *templates* para a exibição dos objetos navegacionais. Eles descrevem o tipo e a formatação do elemento HTML que deverá ser criado para exibir cada atributo de um objeto navegacional. A instanciação de objetos de interface é realizada no navegador do cliente, pelo Interpretador de Interfaces Ricas, a partir do casamento entre os atributos do objeto navegacional e o modelo de dados RDF descrito no documento HTML. Desta forma, o funcionamento das interfaces concretas depende da estrita correspondência dos

nomes dos atributos navegacionais em `@property` com os nomes das chaves na tabela *hash* (JSON) que representa o objeto navegacional em tempo de execução.

Se uma instância de *CompositeInterfaceElement* tiver sido especificada com a propriedade *ordered* igual a `true`, este fato também precisa ser indicado no código HTML traduzido. Em vez de anotações RDFa, porém, podemos fazê-lo por meio das classes CSS associadas à *tag*.

4.2.3. Compilação de Descrições Retóricas

A compilação de descrições retóricas traduz a descrição RDF das transições, eventos e decorações em estruturas de dados JSON. As estruturas JSON serão enviadas para o cliente junto com o código das interfaces concretas, como parte do conteúdo do documento HTML renderizado. Estas informações instruirão o Interpretador de Interfaces Ricas a respeito de quais efeitos de decoração deverão ser aplicados nos elementos de interface, como forma de realçar uma transição ou como resposta à ocorrência de um evento.

4.3. Renderização de Interfaces Ricas

Antes de ser enviado para o navegador Web do cliente, o código XHTML+RDFa da interface concreta precisa ser inserido como conteúdo do elemento `<body>` de um documento HTML válido. O processo de formatação de uma interface concreta como um documento HTML é chamado de renderização da interface.

Se a descrição abstrata especifica uma composição de interfaces, as referências do tipo “o código da interface A deve conter o código da interface B”, salvas durante o processo de compilação, serão finalmente resolvidas quando da sua renderização.

O documento HTML que renderiza uma ou mais interfaces também inclui diversas funções Javascript, geradas dinamicamente com o propósito de:

a) Retornar a representação JSON do(s) objeto(s) navegacional(is) a serem exibidos;
b) Retornar a representação JSON das transições, eventos e decorações especificadas para a(s) interface(s);
c) Prover a lógica de funcionamento para controles ricos incluídos na interface;
d) Prover a lógica que implementa os efeitos de decoração especificados para os elementos da interface.

O código das duas primeiras classes de funções é gerado dinamicamente a partir da consulta ao estado da aplicação ou a partir do resultado da compilação das descrições retóricas, enquanto que os códigos dos controles ricos e dos efeitos ficam

armazenados como propriedades destes objetos, no repositório de instâncias da ontologia.

A renderização do documento HTML também incluirá em seu elemento `<head>` a declaração de folhas de estilo necessárias à formatação dos *widgets* concretos, assim como vínculos às dependências externas dos controles ricos e dos efeitos de decoração especificados na descrição retórica, tais como folhas de estilo e scripts. As dependências externas são indicadas na propriedade *dependencies* dos controles ricos e dos efeitos. As instâncias de *DHTMLRichControl* possuem ainda a propriedade *cssCode*, que armazena folhas de estilo para declaração interna.

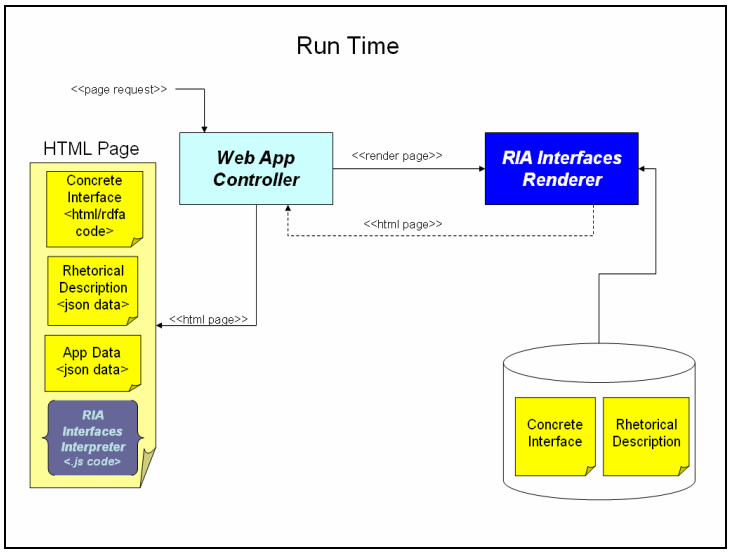

Figura 35 – Renderização de Interfaces

4.4. Execução de Interfaces Ricas

Em geral, o código da interface concreta não deve ser exibido, mas sim, interpretado como um gabarito ou modelo para a criação de objetos de interface. É necessário dizer que se o elemento abstrato foi especificado para representar um atributo ou classe navegacional (através da propriedade *fromAttribute* e *fromClass*), o elemento concreto que o apresentará será criado em tempo de execução da interface. Esta é a função do Interpretador de Interfaces Ricas.

O documento HTML renderizado contém tanto a representação JSON dos objetos navegacionais a serem exibidos, como também o código da interface concreta que provê os *templates* para a criação dos objetos de interface. O Interpretador de Interfaces Ricas, ao representar visualmente cada um dos atributos do nó navegacional, usará a anotação semântica no atributo `@property` para decidir qual *widget* usar como *template*.

O documento HTML também contém as especificações de transições e eventos. O Interpretador de Interfaces irá consultar as descrições retóricas para decidir: (a) a ordem em que deverá ocultar ou exibir objetos, (b) qual tipo de animação ou decoração utilizará para isso e (c) quais *listeners* de eventos deverá registrar em cada *widget*.

Além disso, o Interpretador de Interfaces é a entidade que irá desempenhar o papel do cliente durante a comunicação assíncrona entre Visão e Modelo, descrita em detalhes na seção seguinte.

4.5. Transações

Transação, no âmbito do nosso discurso, designa as interações entre Visão e Modelo mediadas por um sistema de fila de mensagens, que tem por finalidade reduzir o grau de acoplamento entre as duas camadas. Uma transação é representada por um identificador único e possui uma estrutura de dados FIFO associada a ela, onde são enfileiradas as respostas produzidas pela execução de uma operação no modelo.

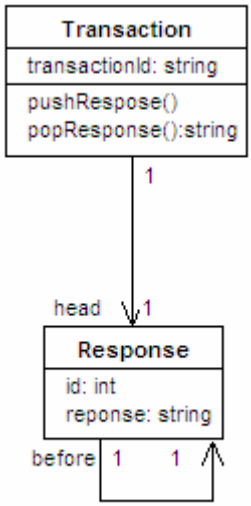

Figura 36 – Modelo de dados das Transações

O projetista especifica que uma operação deverá ser executada sob o contexto de uma transação quando indica uma interface de transição na propriedade *transitionInterface* do ativador ou evento que acionam a operação. Neste caso, quando o *SimpleActivator* for ativado, ou o evento ocorrer, o Interpretador de Interfaces Ricas irá solicitar o início de uma transação ao Gerenciador de Transações e, em seguida, enviará a requisição ao modelo. Quando uma requisição é feita no escopo de uma transação, o método executado em resposta a esta requisição tem a opção de armazenar os resultados parciais do processamento na fila de respostas da transação. O final do processamento será simbolizado por meio de um marcador de fim de

transação, que também será salvo na fila. Em paralelo, após o cliente enviar a requisição, ele iniciará um *loop* de chamadas para consumir o conteúdo desta fila (técnica de *polling*). O *loop* será encerrado quando o cliente retirar da fila o marcador de fim de transação. Somente então será processada qualquer resposta que o método tenha retornado diretamente para o cliente.

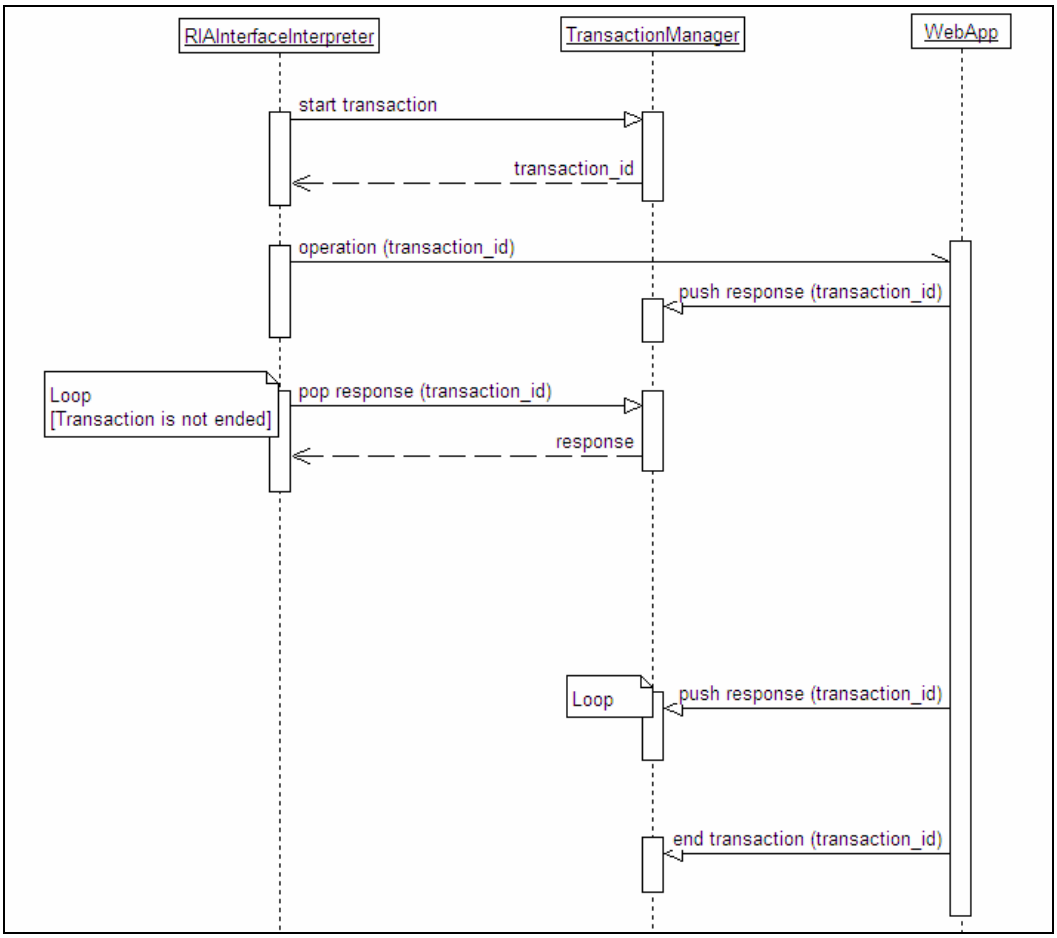

Figura 37 – Diagrama de Seqüência de uma Transação

O Gerenciador de Transações é um artefato de *software* especializado em gerar os identificadores de transação e inicializar as filas de respostas. Ele também provê a interface para os clientes consumirem os dados armazenados na fila. O formato JSON é utilizado para intercâmbio de dados via fila de transação.

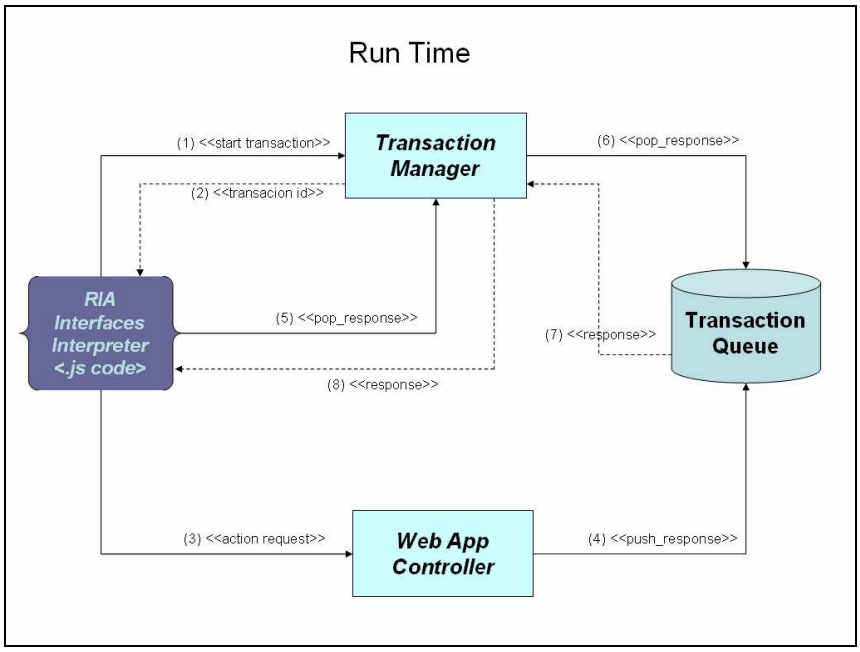

Figura 38 – Execução de uma operação sob o contexto de uma transação

As transações fornecem, por meio das filas de respostas, um mecanismo para comunicar, à camada de visão, a ocorrência de eventos na camada de modelo. Nessa configuração, o Modelo é capaz não apenas de retornar o *resultado final* do processamento da requisição, mas também de atualizar a Visão com qualquer *informação* que se faça *disponível durante* o processamento. Assim, antes mesmo que a execução de uma operação esteja completa, o Modelo pode salvar na fila de respostas dados que informem o progresso da transação, os quais serão recuperados e utilizados para atualizar a Visão. Os objetos JSON, recuperados da fila de respostas em tempo de execução, são processados e exibidos pelo Interpretador de Interfaces Ricas, da mesma forma como ocorre com os objetos navegacionais carregados junto com a interface concreta após a sua renderização.