UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

FERRAMENTA DE MAPEAMENTO DE UIDs PARA JSF

FILIPE BIANCHI DAMIANI

Florianópolis - SC
2011 / 1

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

FERRAMENTA DE MAPEAMENTO DE UIDs PARA JSF

Filipe Bianchi Damiani

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Orientadora:
Profa. Dra. Patrícia Vilain

Membros da Banca:
Prof. Dr. Leandro José Komosinski
Prof. Dr. Ricardo Pereira e Silva

Florianópolis - SC
2011 / 1

Filipe Bianchi Damiani

FERRAMENTA DE MAPEAMENTO DE UIDs PARA JSF

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Orientadora: _____
Profa. Dra. Patrícia Vilain

Banca Examinadora

_____
Prof. Dr. Leandro José Komosinski

_____
Prof. Dr. Ricardo Pereira e Silva

À minha família.

Agradecimentos

Agradeço aos meus pais, Mário Antônio Damiani Filho e Itajane Maria Bianchi Damiani, por absolutamente tudo. Ao meu irmão Maurício Bianchi Damiani, que juntamente com meus pais, fez com que estes últimos anos acontecessem na minha vida. Se venço hoje, é graças também a vocês.

Aos meus amigos, quero agradecer os grandes momentos de alegria e também os de tristeza que compartilhamos. E pelas valiosas histórias que, certamente, contaremos aos nossos netos.

A minha namorada que amo tanto, pelo carinho, companheirismo e compreensão durante este ultimo ano.

A minha orientadora, professora Patrícia Vilain, por todo apoio, incentivo, ensinamentos e dedicação, fundamentais para a realização deste trabalho.

Sumário

Lista de figuras

Lista de Tabelas

Resumo

Este trabalho apresenta o desenvolvimento e validação de uma ferramenta de mapeamento dos UIDs (Diagrama de Interação do Usuário) para JSF (Java Server Faces). O UID representa toda troca de informação entre o usuário e a aplicação, e seus elementos apresentam características semelhantes com as dos componentes utilizados nas interfaces das aplicações, O JSF é um framework para desenvolvimento de aplicações WEB que utilizam linguagem Java, que possui uma biblioteca padrão de componentes visuais para elaboração das páginas.

A partir das características dos elementos do UID e dos componentes das páginas JSF, e com base em um conjunto de regras de mapeamento dos elementos do UID para conceitos de uma ontologia de Widgets Abstratos, foi possível desenvolver um conjunto de regras que automatiza o mapeamento dos elementos do UID para componentes de páginas JSF.

A validação da ferramenta desenvolvida foi realizada com a geração de páginas para um conjunto de UIDs elaborados para o projeto de uma aplicação para venda de CDs musicais através de uma loja virtual. E, ainda, com a comparação de páginas geradas pela ferramenta com páginas de sites comerciais.

Palavras-Chave: UID, JSF, WIDGETS ABSTRATOS, MVC, APLICAÇÃO WEB

1. Introdução

O Diagrama de Interação do Usuário (UID) é uma ferramenta diagramática que representa a interação do usuário com o sistema. Os UIDs são elaborados pelos analistas durante a tarefa de levantamento de requisitos, juntamente com os casos de uso. Além de contribuírem com a tarefa de levantamento de requisitos, ainda podem ser utilizados nas etapas de projeto conceitual, projeto navegacional e projeto da interface com o usuário [Vilain, 2002].

O UID possui uma notação simples e que evita influenciar nas opiniões sobre os requisitos dos usuários na etapa de levantamento de requisitos. Contudo, todas as trocas de informações entre o usuário e o sistema estão representadas nele e estas devem ser consideradas na etapa do projeto da interface, pois cada informação trocada, seja ela fornecida pelo usuário ou retornada pelo sistema, deve ter um objeto que a represente na interface concreta.

1.1. Motivação

Na tarefa de levantamento de requisitos o analista deve coletar as opiniões dos futuros usuários sobre as características, funcionalidades e requisitos que o sistema deverá apresentar. Para não influenciar no projeto da interface com o usuário durante a definição dos requisitos dos usuários, o analista pode fazer uso do UID como artefato

para representar as trocas de informações entre o sistema e o usuário, representando, de forma simples e compreensível, a interação dos usuários com o sistema.

Entretanto, as informações trocadas entre o usuário e o sistema, representadas no UID, podem ser utilizadas, posteriormente, no projeto da interface com o usuário facilitando assim a tarefa dos projetistas. Com regras de mapeamento das informações representadas nos UIDs para objetos concretos da interface, a tarefa de definição dos componentes que devem constar na interface pode ser parcialmente automatizada.

1.2. Justificativa

Os elementos do UID apresentam características semelhantes com as dos componentes comumente utilizados nas interfaces das aplicações, sejam as entradas de usuários que se assemelham com os elementos *input* das interfaces, as saídas do sistema que são semelhantes aos elementos *output* das interfaces. Considerando estas características, é possível definir um conjunto de regras que mapeiam os elementos do UID para componentes de uma interface concreta. Com este conjunto de regras é possível automatizar parcialmente a geração de interfaces a partir do UID.

O Java Server Faces (JSF) é um framework para desenvolvimento de aplicações Web que utilizam tecnologia Java. Ele trabalha principalmente nas camadas de Visão e Controle do modelo MVC [BUSCHMANN, MEUNIER, ROHNERT, SOMMERLAD e STAL 1996]. Seu principal objetivo é facilitar o trabalho dos desenvolvedores na tarefa de construção das páginas das aplicações Web [MARAFON 2006]. Por apresentar uma

clara separação entre o modelo de negócio e a visão, e, ainda, por ser altamente utilizado nas aplicações comerciais, o JSF foi a tecnologia escolhida para o desenvolvimento das interfaces concretas.

1.3. Objetivo Geral e Objetivos Específicos

1.3.1. Objetivo Geral

O objetivo geral deste trabalho consiste na definição do mapeamento dos UIDs (Diagramas de Interação do Usuário) para os componentes JSF (Java Server Faces) e no desenvolvimento de um protótipo de uma ferramenta que implemente este mapeamento.

1.3.2. Objetivos Específicos

Para alcançar o objetivo geral, este trabalho apresenta os seguintes objetivos específicos:

- Estudo das técnicas relacionadas com o trabalho: UIDs, Widgets Abstratos, Mapeamento dos UIDs para Widgets Abstratos e JSF;
- Definição das regras de mapeamento dos UIDs para componentes JSF.
- Desenvolvimento e validação da ferramenta, utilizando a linguagem Java.

1.4. Estrutura do Trabalho

Este trabalho está organizado da seguinte forma. No capítulo 2 são apresentadas as técnicas relacionadas com o trabalho: o UID, a ontologia de Widgets Abstratos, o mapeamento dos UIDs para Widgets Abstratos e o framework JSF.

No capítulo 3 é apresentado um estudo sobre algumas ferramentas para modelagem e geração de interfaces de usuários.

No capítulo 4 são definidas as regras do mapeamento dos UIDs para os componentes JSF. Inicialmente, as classes da ontologia de widgets abstratos são relacionadas com os componentes JSF. Em um segundo momento, são definidas as regras para geração das páginas JSF diretamente a partir dos UIDs.

No capítulo 5 é descrito o desenvolvimento da ferramenta que implementa as regras de mapeamento das páginas JSF a partir dos UIDs, com o intuito de automatizar este processo.

No capítulo 6 é apresentado um exemplo de aplicação da ferramenta desenvolvida na geração de páginas JSF.

No capítulo 7 são apresentadas as conclusões obtidas com este trabalho.

2. Fundamentação Teórica

Este capítulo apresenta as tecnologias estudadas para o desenvolvimento deste trabalho. Inicialmente são apresentados o UID e a ontologia de Widgets Abstratos. Em seguida, é apresentada a definição do mapeamento do UID para Widgets Abstratos. E por último, é apresentado o framework JSF com ênfase na biblioteca de componentes HTML utilizada para construção das páginas web.

2.1. Diagrama de Interação do Usuário

Definição:

> "Um Diagrama de Interação do Usuário ou UID (User Interaction Diagram) representa a interação entre o usuário e uma aplicação que apresenta intensa troca de informações e suporte à navegação. Este diagrama descreve somente a troca de informações entre o usuário e a aplicação, sem considerar aspectos específicos da interface com o usuário nem da navegação." [VILAIN 2003]
>
> "Um UID é composto por um conjunto de estados conectados através de transições. Os estados representam as informações que são trocadas entre o usuário e a aplicação, enquanto as transições são responsáveis pela troca do foco da interação de um estado para outro. Diz-se que um estado da interação é o foco da interação quando as informações contidas nesse estado representam as informações que estão sendo trocadas entre o usuário e a aplicação em um dado momento. As informações participantes da interação entre o usuário e a aplicação são apresentadas dentro dos estados de interação, mas algumas seleções e opções são associadas às transições. As transições são disparadas,

geralmente, pela entrada ou seleção de informações pelo usuário.” [VILAIN 2003]

O UID é uma ferramenta que apresenta graficamente a interação do usuário com o sistema, descrita no caso de uso. Porém, não apresenta graficamente os requisitos não funcionais. Tais requisitos podem ser descritos em notas textuais.

Os UIDs são elaborados na fase de levantamento de requisitos do processo de software e, juntamente com os casos de uso, contribuem com esta tarefa dos analistas. Por apresentarem foco apenas nas informações necessárias da interação, os UIDs não influenciam previamente na navegação e na interface com o usuário.

Após serem desenvolvidos no processo de levantamento de requisitos, os UIDs podem ser utilizados nas etapas de projeto conceitual, projeto navegacional, no caso de uma aplicação web, e projeto de interface, sendo assim uma ferramenta importante no processo de software.

2.1.1. Notação

O UID apresenta uma notação simples, de fácil compreensão tanto para o projetista quanto para o usuário, e independente da interface com usuário que será projetada.

2.1.1.1. Item de Dado

Um item de dado representa uma informação única que aparece durante a interação. Ele pode estar acompanhado do seu domínio, seguido por dois pontos e o nome do domínio. Geralmente o nome do item de dado é escrito em letras minúsculas. Se seu domínio não for especificado, é assumido domínio *Texto*.

<item de dado>:<domínio>

<item de dado>

2.1.1.2. Estrutura

Uma estrutura representa uma coleção de informações (itens de dados, estruturas, conjunto de itens de dados ou conjunto de estruturas) que estão relacionadas de alguma maneira. Graficamente uma estrutura possui seu nome, que é obrigatório, seguido das informações entre parênteses.

<Estrutura> (<item de dado 1>, <item de dado 2>...<item de dado n>)

<Estrutura> ( )

2.1.1.3. Conjunto

Representa um conjunto de itens de dados ou estruturas. Sua multiplicidade é representada por *min..max* em frente ao elemento. A multiplicidade padrão é 1..N e pode ser representada somente por reticências (...).

…<item de dado >

… <Estrutura> (<item de dado 1>, <item de dado 2> .. <item de dado n>)

… <Estrutura> ( )

1..5 <item de dado>

0..5 <item de dado>

2.1.1.4. Dado Opcional

Um dado opcional representa um item de dado, estrutura ou texto opcional. Graficamente os dados opcionais são representados pelo símbolo "?" ou por um conjunto de cardinalidade 0..1 ou ainda, no caso de uma entrada do usuário, por um retângulo com linhas pontilhadas.

<item de dado>?

…<item de dado>?

<Estrutura> (<item de dado 1>, <item de dado 2> .. <item de dado n>)?

" texto "?

0..1 <item de dado>

<item de dado>

2.1.1.5. Entrada do usuário

Uma entrada de usuário é um item de dado ou estrutura fornecido pelo usuário ao sistema. Graficamente uma entrada de usuário é representada pela informação colocada dentro de um retângulo.

<item de dado>

É usado o símbolo “OR” entre dois itens de dados que são dependentes entre si e que pelo menos um deles deve ser fornecido pelo usuário. No caso de somente um dos itens de dados poder ser fornecido, então é usado o símbolo “XOR” entre eles.

<item de dado1> OR <item de dado2>

<item de dado1> XOR <item de

2.1.1.6. Entrada do Usuário Enumerada

É uma entrada do usuário que deve ser selecionada a partir de uma lista de opções. Estas opções aparecem entre colchetes e separadas por vírgula. A multiplicidade é representada em frente ao nome do item de dado.

<item de dado> [<opção 1>, <opção 2>...<opção

<min>..<Max> <item de dado> [<opção 1>, <opção

2.1.1.7. Saída do Sistema

É toda informação retornada pelo sistema, item de dado ou estrutura, e deve ser colocada diretamente no estado de interação.

<item de dado> : <domínio>

<item de dado>

<item de dado >?

…<item de dado>

<Estrutura> (<item de dado>)

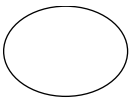
2.1.1.8. Texto

Representa um texto pré-definido, de caráter explicativo, apresentado pelo sistema durante a interação. Deve estar entre aspas duplas.

“<texto>”

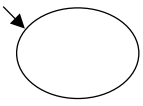
2.1.1.9. Estado da Interação

Um estado de interação é representado graficamente por uma elipse. As informações trocadas entre o usuário e o sistema estão inseridas nesta elipse.



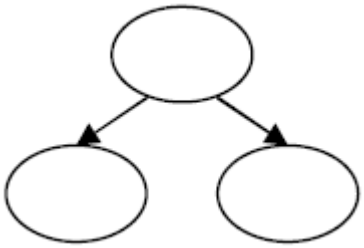
2.1.1.10. Estado Inicial da Interação

É representado por uma transição sem origem. É o estado no qual começa o fluxo das trocas de informações entre o usuário e o sistema.



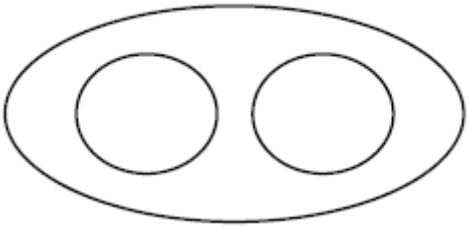
2.1.1.11. Estados Alternativos da Interação

É utilizado quando existem duas ou mais saídas alternativas a partir de um estado de interação. O estado da interação que se tornará foco da interação depende da informação fornecida pelo usuário ou da opção selecionada por ele.



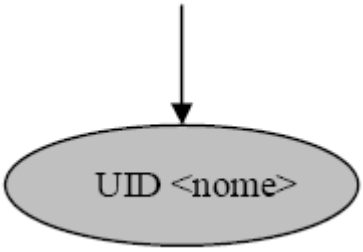
2.1.1.12. Sub-Estados de um Estado da Interação

São utilizados para representar partes excludentes de um estado de interação. O usuário deve optar por qual sub-estado vai seguir durante a interação.



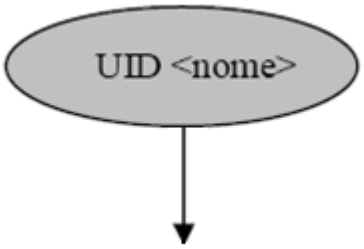
2.1.1.13. Chamada de outro UID

É utilizado para representar que o foco da interação foi transferido para outro UID.



2.1.1.14. Chamada a partir de outro UID

Representa que o foco da interação foi transferido a partir de outro UID.



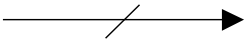
2.1.1.15. Transição do Estado da Interação

São utilizadas para representar a troca do foco da interação. O estado destino torna-se o novo foco da interação após o sistema retornar as informações necessárias e o usuário fornecer os dados requisitados no estado origem.

Uma transição deve ter como origem um estado da interação, um sub-estado de um estado da interação, ou ainda, um item de dado ou uma estrutura retornados pelo sistema.

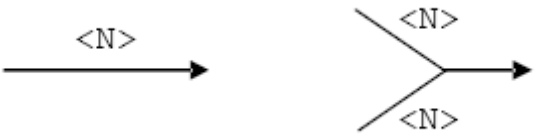
Para representar, explicitamente, que o usuário pode retornar ao estado de interação origem da transição, deve ser utilizada a seguinte representação:

Porém quando for necessário representar que o usuário não pode retornar ao estado origem da transição, então se usa a seguinte representação:

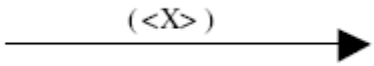


Uma transição pode apresentar três tipos de rótulos:

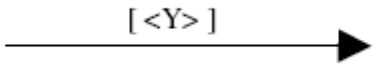
- *Transição com Seleção de N Elementos:* para uma transição receber o rótulo de quantos elementos devem ser selecionados para que o estado destino torne-se foco da interação, a origem dela deve ser um conjunto de itens de dados ou estruturas retornados pelo sistema. A seleção pode ser de elementos de conjuntos diferentes.



- *Transição com Seleção da opção X:* para que o estado destino torne-se foco da interação é necessário que a opção X seja selecionada pelo usuário.



- *Transição com Condição Y:* para que a mudança do foco da interação ocorra, a condição Y deve ser satisfeita.



2.1.1.16. Pré-Condições e Pós-Condições

Quando a execução da interação representada no UID requer que algumas condições sejam satisfeitas previamente, estas condições são representadas pelas pré-condições associadas ao UID.

```
Pré-condições: <condições>
```

Para representar as condições que precisam ser satisfeitas ao término da execução da interação representada no UID, usa-se as pós-condições.

```
Pós-condições: <condições>
```

2.1.1.17. Notas Textuais Anexadas ao UID

As notas textuais servem para especificar alguma informação importante que não pode ser representada graficamente no UID. Também são utilizadas para descrever os requisitos não-funcionais.

```
Nota: <nota textual>
```

2.1.2. Exemplo de UID.

A Figura 2.1 mostra o exemplo de um UID com os principais elementos apresentados na seção 2.1.1. Este exemplo representa a tarefa *Seleção de um CD a partir de um*

Título. No estado inicial o usuário entra com o título do CD. Após o usuário entrar com o título, o sistema apresenta o conjunto de CDs que combinam com o título fornecido. A partir deste conjunto, o usuário inclui na cesta de compras aquele que ele deseja comprar.

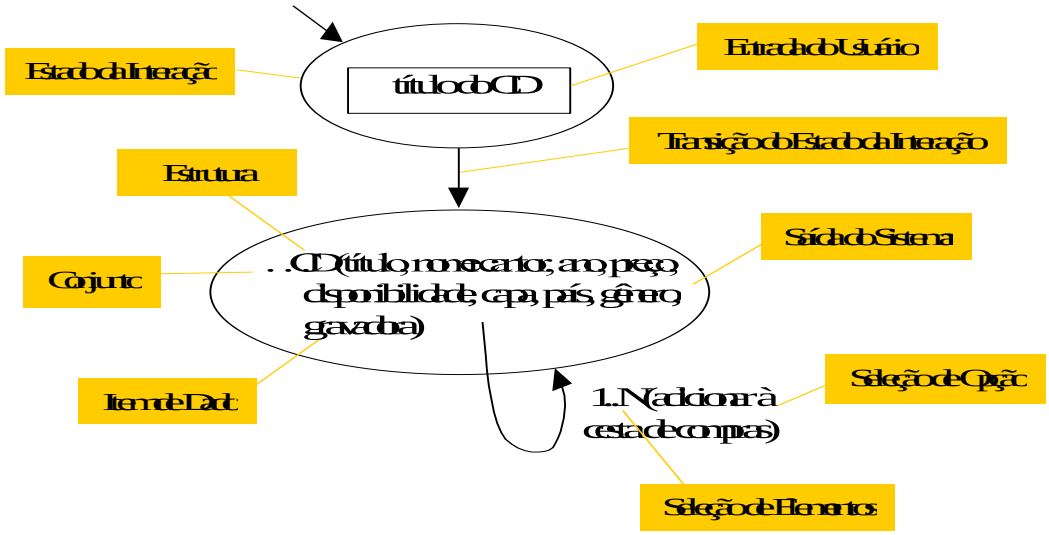


Figura 2.1 – Exemplo de UID. [VILAIN 2003]

2.2. Widgets Abstratos

"A ontologia de widgets abstratos é utilizada para especificar interfaces abstratas" (Moura, 2004). Nela estão representados os widgets essenciais que dizem respeito às trocas de informações entre o usuário e a aplicação, mas de forma independente das tecnologias e dos padrões de projetos utilizados.

A ontologia de widgets abstratos é composta por 11 conceitos (Figura 2.2), onde cada um representa um ou mais elementos da ontologia de widgets woncretos. Estes 11 elementos da ontologia de widgets abstratos possuem semelhança com as primitivas do UID.

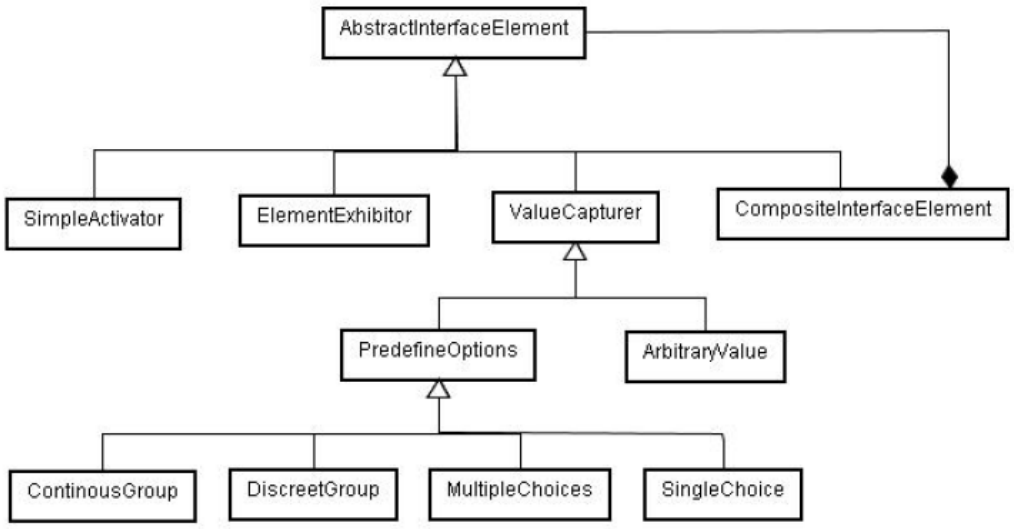


Figura 2.2 – Ontologia de Widgets Abstratos.

2.2.1. Classes da ontologia

As classes da ontologia de widgets abstratos representam os elementos das interfaces das aplicações hipermídia. Aplicações hipermídia são sistemas utilizados para criação, manipulação, apresentação e representação de informação, nos quais a informação é armazenada em uma coleção de estruturas multimídia, permitindo aos usuários navegar por estas estruturas [SCHWABE, ROSSI, 2003].

- *AbstractInterfaceElement:* define os possíveis elementos abstratos, esta classe é composta por 4 subclasses:
    - *SimpleActivator:* representa os elementos capazes de reagir a eventos.
    - *ElementoExhibitor:* representa os elementos que exibem algum conteúdo.
    - *ValueCapturer:* representa os elementos que possibilitam a entrada de algum valor ao sistema. Esta classe possui duas subclasses:

- *ArbitraryValue:* permite que o usuário insira dados através do uso do teclado.

- *PredefineOptions:* permite a seleção de um subconjunto de valores de um conjunto pré-definido. Quatro subclasses especializam esta:

    - *ContinousGroup:* permite a seleção de um valor de um conjunto infinito de valores.

    - *DiscreetGroup:* permite a seleção de um valor de um conjunto de valores enumeráveis.

    - *MultipleChoice:* permite a escolha de mais de um elemento de um conjunto enumerável.

    - *SingleChoice:* permite a seleção de apenas um elemento de um conjunto enumerável.

  - *CompositeInterfaceElement:* representa uma composição de AbstractInterfaceElements.

- *AbstractInterface:* é uma composição de elementos da classes "AbstractInterfaceElement". Define a interface abstrata final.

2.2.2. Mapeamento dos widgets abstratos para elementos concretos

No trabalho apresentado por [MOURA, 2004], os widgets abstratos formam um vocábulo para descrição de interfaces abstratas e devem ser mapeados para elementos de interface concreta.

A ontologia de widgets concretos apresenta o grupo de elementos concretos que irão representar os elementos abstratos, da ontologia de widgets abstratos, em uma interface concreta. A Figura 2.3 apresenta um exemplo da especificação dos widgets abstratos para um elemento de busca de uma aplicação hipermídia, e seus mapeamentos para elementos concretos da interface.

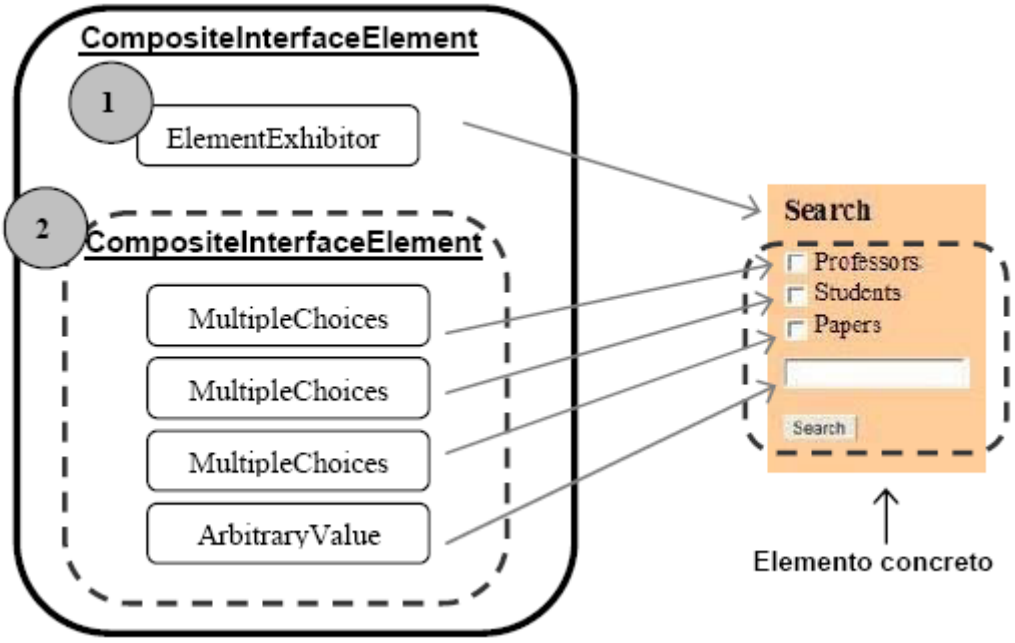


Figura 2.3 – Estrutura abstrata para o elemento busca [MOURA, 2004]

A definição do mapeamento dos elementos abstratos para elementos concretos é realizada pelo projetista, considerando os objetivos e necessidades do cliente, de forma que cada elemento da ontologia de widgets abstratos corresponda a somente um elemento concreto.

Neste trabalho, ao invés de utilizar a ontologia de widgets concretos, os componentes JSF serão utilizados como elementos da interface concreta.

2.3. Mapeamento dos UIDs para Widgets Abstratos.

Com o intuito de utilizar os UIDs na etapa de projeto de interface, [REMÁCULO 2005] propôs um conjunto de regras de mapeamento dos elementos do UID para os elementos da ontologia de widgets abstratos. Este mapeamento foi utilizado como base no desenvolvimento das regras de mapeamento dos elementos do UID para os componentes JSF.

- *Item de dado:* caso essa informação apenas exiba algum tipo de conteúdo, ela deve ser mapeada para a classe *ElementExhibitor*. Caso, além de exibir algum conteúdo, esta informação ofereça acesso a outra página ou parte do mesmo documento, ela deve ser mapeada para a classe *SimpleActivator*.

- *Estrutura:* se a estrutura for composta por elementos que tenham como objetivo apenas exibir alguma informação, ela pode ser mapeada de duas formas: cada elemento deve ser mapeado para um *ElementExhibitor* ou toda a estrutura deve ser mapeada a um *ElementExhibitor*.

  Se a estrutura for composta por elementos que, além de exibir informação, forneçam acesso a outra página ou parte do mesmo documento, ela deve ser mapeada para um único elemento *SimpleActivator*. Se cada elemento da estrutura leve a destinos diferentes, cada elemento deve ser mapeado para um *SimpleActivator*.

  Nos casos de estruturas em que cada elemento é mapeado para um *ElementExhibitor* ou um *SimpleActivator*, a estrutura deve ser mapeada para um *CompositeInterfaceElement*, onde serão incluídos os *ElementExhibitor* ou *SimpleActivator* referentes aos elementos da estrutura.

- *Conjunto:* como primeiro passo, o conjunto deve ser mapeado para um *CompositeInterfaceElement.* O segundo passo é mapear cada elemento do conjunto para *ElementExhibitor* ou *SimpleActivator*, se o conjunto for de itens de dados. No caso de ser um conjunto de estruturas, o segundo passo segue a proposta do mapeamento para Estrutura para cada estrutura do conjunto.

- *Dado Opcional:* um dado opcional pode ser mapeado para dois conceitos: *ValueCapturer*, quando o sistema solicita ao usuário uma informação opcional, e *ElementExhibitor*, quando o sistema retorna uma informação que é opcional ao usuário. No caso de *ValueCapturer*, um dado opcional pode ser mapeado para 5 classes:

    - *ArbitraryValue:* quando o sistema permite que o usuário insira dados através do teclado.

    - *ContinousGroup:* quando o sistema permite que o usuário faça a seleção de um valor de um conjunto finito de valores.

    - *DiscreetGroup:* quando o sistema permite que o usuário faça a seleção de um valor de um conjunto de valores enumeráveis, como por exemplo, o conjunto dos números reais.

    - *MultipleChoice:* quando o sistema permite que o usuário faça a escolha de mais de um elemento de um conjunto enumerável.

    - *SingleChoice:* quando o sistema permite que o usuário faça a escolha de apenas um elemento de um conjunto enumerável.

- *Entrada do Usuário:* a entrada de usuário é mapeada para a classe *ArbitraryValue*, visto que neste caso o usuário pode inserir os dados através do uso do teclado.

- *Entrada do Usuário Enumerada:* quando o usuário deve selecionar apenas um item de dado fornecido pelo sistema, o mapeamento deve ser realizado para o conceito *SingleChoice.* Quando o usuário pode selecionar mais de um item de dado fornecido pelo sistema, então seu mapeamento deve ser para o conceito *MultipleChoice.*

- *Saída do Sistema:* representa um item de dado ou uma estrutura, e seu mapeamento deve ser realizado segundo as definições citadas anteriormente.

- *Texto:* deve ser mapeado para a classe *ElementExhibitor.*

- *Estado de Interação:* um estado de interação deve ser mapeado para o conceito *CompositeInterfaceElement,* que corresponde a uma composição dos elementos abstratos da classe *AbstractInterfaceElement.*

- *Estado Inicial de Interação:* deve ser mapeado igual ao Estado de Interação.

- *Estados Alternativos da Interação:* assim como para Estado Inicial de Interação, os Estados Alternativos de Interação devem ser mapeados segundo a regra definida para o elemento Estado de Interação.

- *Sub-Estados de um Estado de Interação:* cada sub-estado deve ser mapeado para um *CompositeInterfaceElement.*

- *Transição com Seleção de N Elementos:* este mapeamento só é necessário caso não tenha sido realizado mapeamento de dado opcional. No contexto dos widgets

abstratos, os conceitos que suportam a seleção de um ou mais elementos especificados nos UIDs são:

  - *ContinousGroup*: quando o sistema permite que o usuário selecione um valor de um conjunto infinito de valores.

  - *DiscreetGroup*: quando o sistema permite que o usuário selecione um valor de um conjunto de valores enumeráveis, como por exemplo, o conjunto dos números reais.

  - *MulipleChoice:* quando o sistema permite que o usuário escolha mais de um elemento de um conjunto enumerável.

  - *SingleChoice*: quando o sistema permite que o usuário escolha apenas um elemento de um conjunto enumerável.

- *Transição com Seleção da Opção X*: significa que a opção X deve ser selecionada para que o estado de interação destino se torne o foco da interação. A opção que deve ser selecionada é pertencente à classe *SimpleActivator*, a qual representa qualquer elemento capaz de reagir a elementos externos que possua um evento associado a ele, tais como *link* e botão de ação. No caso da opção precisar de mais de um elemento para que o estado de interação destino se torne o foco da interação, a mesma deve ser mapeada para *MultipleChoice* e atrelada a um *SimpleActivator* para que a transição entre estados de interação ocorra.

Os demais elementos: *Chamada de Outro UID, Chamada a partir de Outro UID, Pré-Condições, Pós-Condições e Notas Textuais* não possuem mapeamento para a ontologia de widgets abstratos, pois não são elementos que compõe a interface abstrata.

A Tabela 2.1 apresenta, resumidamente, o mapeamento de cada elemento do UID para um elemento da ontologia de widgets abstratos.

Tabela 2.1 – Tabela do mapeamento UID para Ontologia de Widgets Abstratos. [REMÁCULO, 2005]:

Mapeamento UIDs para Ontologia *Widgets* Abstratos	
UIDs	***Widgets* Abstratos**
Item de dado e Item de dado personalizado	*ElementExhibitor* ou *SimpleActivator*
Estrutura	A estrutura toda pode ser mapeada para um *ElementExhibitor* ou *SimpleActivator*, assim como cada elemento da estrutura pode ser mapeado para um *ElementExhibitor* ou *SimpleActivator*. Neste caso, 1º deve-se mapear a Estrutura para um *CompositeInterfaceElement*, para que depois os elementos que compõem a Estrutura sejam mapeados para *ElementExhibitor* ou *SimpleActivator*.
Conjunto, Conjunto com conteúdo personalizado e Conjunto em disposição diferenciada	1º passo: mapear o conjunto para um *CompositeInterfaceElement*. 2º passo: mapear cada elemento do conjunto para um *ElementExhibitor* ou *SimpleActivator*, conforme o objetivo do item de dado ou estrutura que compõem o conjunto.
Dado opcional	*ElementExhibitor* ou *ValueCapturer* No caso de *ValueCapturer*, o elemento pode ser mapeado para: *ArbitraryValue*, *ContinuousGroup*, *DiscreetGroup*, *MultipleChoices* ou *SingleChoices*.
Entrada do Usuário	*ArbitraryValue*
Entrada do Usuário Enumerada	*SingleChoice* ou *MultipleChoice*.
Saída do Sistema	Recebe o mapeamento descrito para os elementos dos UIDs, conforme o tipo de elemento que representa a Saída do Sistema (Item de dado, Item de dado personalizado, Estrutura, Conjunto, Conjunto com conteúdo personalizado, Conjunto em disposição diferenciada, Dado Opcional, Entrada do Usuário e Entrada do Usuário Enumerada).
Texto	*ElementExhibitor*
Estado de Interação, Estado Inicial da Interação e Estados Alternativos da Interação.	*CompositeInterfaceElement*. Caso o Estado e Interação seja um conjunto de *CompositeInterfaceElement*, o estado de interação será a associação dos *CompositeInterfaceElements*.
Sub-estados de um Estado de Interação.	*CompositeInterfaceElement*
Transição com Seleção da Opção X e Transição com Seleção da Opção Restrita X.:	*SimpleActivator*. No caso da opção precisar de um outro elemento para que o estado de interação destino se torne o foco da interação, a mesma deve ser mapeada para *MulipleChoice* ou *SingleChoice* e atrelada a um *SimpleActivator* para que a transição entre Estados de Interação ocorra.
Transição com Seleção de N Elementos	Os conceitos que suportam a seleção de 1 ou mais elementos especificados nos UIDs são: *ContinuousGroup*, *DiscreetGroup*, *MultipleChoices* ou *SingleChoices*.
Chamada de Outro UID, Chamada a partir de Outro UID, Transição com Condição Y, Pré-Condições, Pós-Condições e Notas Textuais.	Não há mapeamento para a ontologia de *widgets* abstratos.

2.4. Java Server Faces (JSF)

O Java Server Faces (JSF) é um framework para desenvolvimento de aplicações Web que utilizam tecnologia Java e é baseado no padrão MVC (Model-View-Control). Seguindo a tecnologia MVC, JSF tem como um de seus diferenciais a clara separação entre o modelo de negócio e a visualização [MARAFON, 2006]. Neste trabalho será utilizada a versão 2.0 do framework.

Como este trabalho tem como objetivo o mapeamento dos elementos do UID para elementos visuais de páginas JSF, o foco será dado na biblioteca HTML do JSF que representa visualmente os objetos JSF.

2.4.1. FacesServlet

FacesServlet é o controlador do framework por onde devem passar todas as requisições das páginas HTML. Depois de passar por ele, a requisição começa a executar a primeira fase do ciclo de vida das requisições JSF, detalhada adiante.

2.4.2. View ID

É uma árvore que contém os componentes de uma página que instanciou uma requisição. Uma View ID pode estar em três estados:

- *New View:* quando acaba de ser crida e possui todas as informações dos componentes vazias.
- *Initial View:* quando a página é carregada e as informações dos componentes são preenchidas.

- *Post Back:* quando a view já existe e precisa apenas ser restaurada.

2.4.3. Faces Context

Guarda as View ID´s e todas as informações necessárias para o gerenciamento dos componentes de uma página pelo framework.

2.4.4. Managed Bean

São classes que mantém as informações dos valores dos componentes de uma página. É um objeto Java que representa um formulário do HTML.

2.4.5. Faces-config.xml

É um arquivo de configuração do framework, onde estão inseridas as regras de navegação utilizadas pela aplicação.

2.4.6. Ciclo de vida de uma requisição JSF

O ciclo de vida de uma requisição JSF é dividido em 6 fases (Figura 2.4):

- *Restaurar Apresentação (Restore View)*: é a primeira fase do ciclo de vida de uma requisição, é quando o controle examina a requisição e a página que a solicitou e constrói a View ID.

- *Aplicar Valores Requisitados (Apply Request Values):* é quando ocorre a recuperação do estado atual de cada componente.

- *Processar validações (Process Validations):* o objetivo desta fase é fazer a validação dos valores recebidos por cada componente. Caso ocorra alguma falha na validação, uma mensagem de erro é adicionada no *FacesContext.*

- *Atualizar Valores do Modelo (Update Model Values)*: neste estado ocorre a atualização dos valores dos Managed Beans.

- *Invocar Aplicação (Invoke Application)*: é nesta fase que o framework decide se o formulário será submetido e qual será a página mostrada ao usuário, além de executar as ações de lógica de interface e as ações de lógica de negócio.

- *Renderizar Resposta (Render Response)*: é nesta fase que a página que será mostrada ao usuário é reconstruída.

Este ciclo de vida pode sofrer alterações no fluxo, não executar alguma fase do ciclo ou executar alguma fase inserida pelo desenvolvedor. Na Figura 2.4 é apresentado o ciclo de vida de uma requisição, e as linhas tracejadas representam algumas possíveis alterações do fluxo.

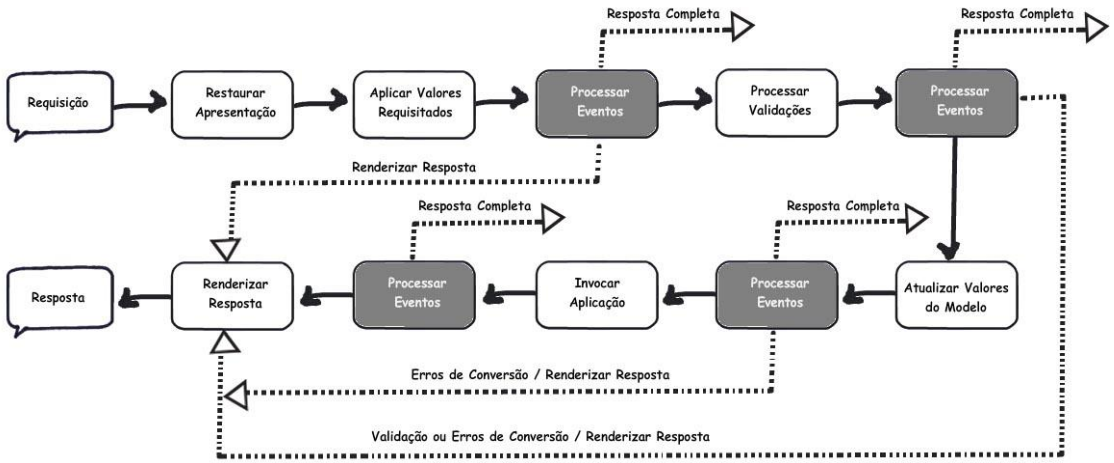


Figura 2.4 – Ciclo de vida das requisições JSF.

2.4.7. Componentes Visuais JSF

A arquitetura de componentes visuais do JSF permite que novos componentes além dos que estão definidos na especificação sejam desenvolvidos. Isso permitiu que bibliotecas de componentes mais "ricos" visualmente fossem desenvolvidas. Dentre estas bibliotecas se destacam a RichFaces (www.jboss.org/richfaces) e a PrimeFaces (http://www.primefaces.org).

A figura 2.5 apresenta o diagrama de classes do pacote de componentes visuais da JSF 2.0, todos os componentes visuais desenvolvidos para a JSF devem implementar a interface UIComponent ou estender a classe UIComponentBase.

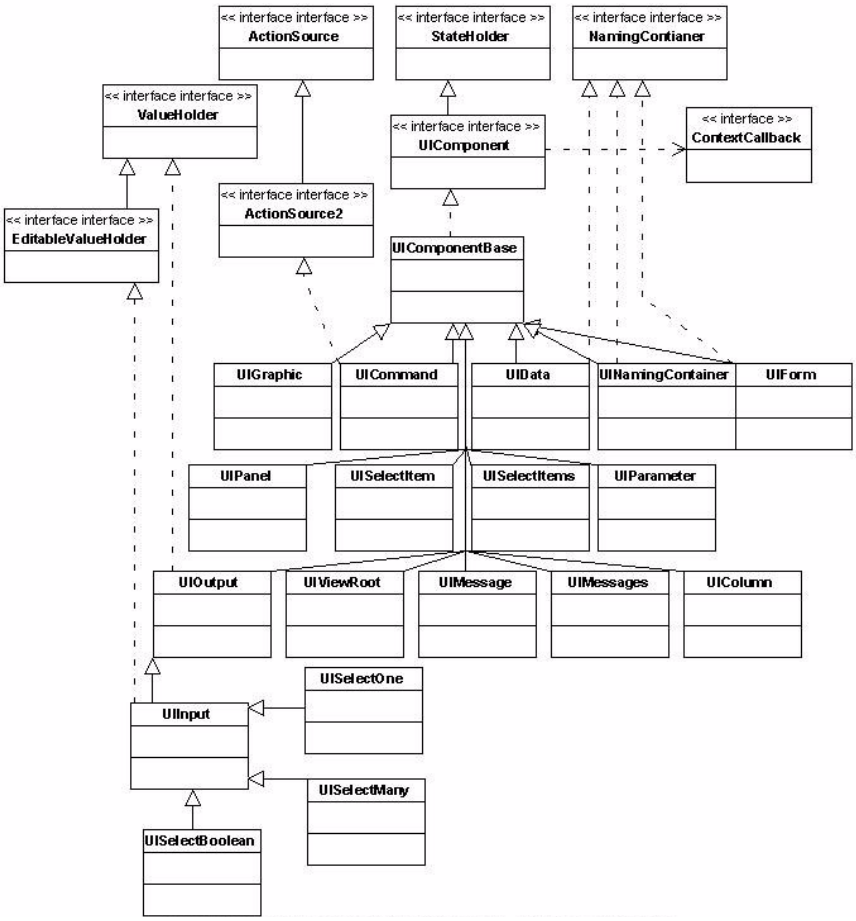


Figura 2.5 – Diagrama de classes do pacote de componentes.

Entretanto, neste trabalho, será feito o uso apenas dos componentes da especificação da JSF 2.0, cada um destes componentes possui representação direta de algum componente HTML, assim o framework dá suporte à "renderização" de páginas HTML diretamente a partir de seus componentes.

2.4.7.1. Componentes HTML

Os componentes da biblioteca HTML do JSF são os seguintes:

form: representa um formulário para envio de dados pelo JSF.

panelGrid: produz uma tabela para disposição organizada dos elementos nele agrupados.

panelGroup: tem como finalidade agrupar outros elementos JSF. Quando convertido para HTML, gerará um SPAM dos elementos, ou, então, caso o atributo *layout* esteja com valor "block", os elementos serão agrupados no elemento *div* do HTML.

column: representa a coluna de uma tabela.

dataTable: é formada por elementos do tipo column e usada para apresentar uma coleção de objetos organizados na forma de uma tabela, onde as colunas representam informações ou atributos dos objetos.

Title	Subject	Price ($)
JSF For Dummies	JSF	25.0
Struts For Dummies	Struts	22.95

outputText: produz um texto simples retornado pelo sistema.

10032

outputLabel: representa um rótulo de algum campo de entrada do usuário.

User Home Address

outputFormat: produz um texto simples, com informação formatada segundo parâmetros dinâmicos, como, por exemplo, a versão do browser.

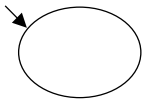
Hello Filipe!

message: produz um texto simples, a partir de uma mensagem JSF, por exemplo, uma mensagem de conversão ou de erro de validação.

messages: produz um texto simples a partir de um conjunto de mensagens.

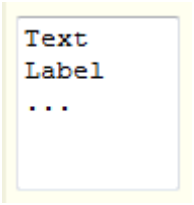
Enter address: Validation Error: Value is required.
Save

graphicImage: mostra uma imagem retornada pelo sistema ao usuário.



inputText: produz um campo de entrada de texto.

inputTextArea: produz uma área para entrada de texto do usuário.



inputSecret: produz um campo para entrada de texto de forma secreta. É muito utilizado para campos de entrada de senhas.

inputHidden: não produz nenhum componente visual. Tem como finalidade permitir ao programador incluir uma variável escondida na página.

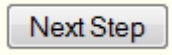
outputLink: produz um Hyperlink que levará para outra página, ou parte da mesma, sem produzir um evento de ação.

Next Page

commandLink: também produz um Hyperlink que levará para outra página ou parte da mesma. Contudo, diferentemente do outputLink, o commandLink produz uma ação e/ou chamada de um método do Managed Bean. Quando um commandLink é acionado, ele gera uma requisição que faz com que o formulário (form), em que ele está inserido, seja submetido ao sistema e os valores dos Managed Beans sejam atualizados (fase IV do ciclo de vida de uma requisição JSF).

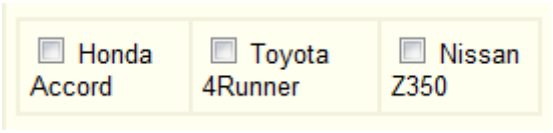
Next Page

commandButton: apresenta a mesma funcionalidade que o commandLink, contudo tem a aparência de um botão.

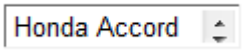


selectBooleanCheckbox: produz uma caixa de seleção que permite que o usuário altere o valor booleano de uma variável.

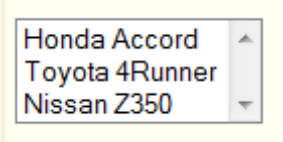
selectManyCheckbox: produz um conjunto de caixas de seleção que permite que o usuário selecione um subconjunto de um conjunto de itens.



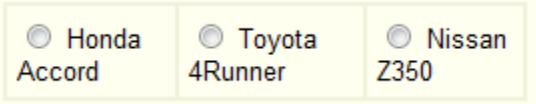
selectManyMenu: produz uma caixa de rolagem que permite a seleção de mais de um item.



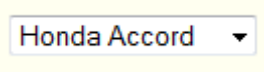
selectManyListbox: produz uma caixa de listagem que permite a seleção de mais de um item.



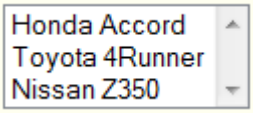
selectOneRadio: produz um conjunto de botões de radio, permitindo a seleção de apenas um botão pelo usuário.



selectOneMenu: produz uma caixa de rolagem que permite a seleção de apenas um item de um conjunto de itens.



selectOneListbox: permite a seleção de apenas um item de um conjunto de itens dispostos em uma caixa de listagem.



2.4.8. Facelets

O Facelets é um projeto com foco no reaproveitamento de software, tem como principal objetivo facilitar todo o processo de desenvolvimento e manutenção das interfaces de uma aplicação JSF. O Facelets já faz parte do JSF 2.0, sendo a tecnologia padrão para o desenvolvimento das interfaces da aplicação JSF.

As interfaces das aplicações JSF 2.0 podem ser definidas através de arquivos XHTML, estes arquivos são processados pelo Facelets que constrói as páginas finais da aplicação. São nestes arquivos que as tags de declaração dos componentes visuais são inseridas.

3. Trabalhos Relacionados

O desenvolvimento da interface de aplicativos independente do desenvolvimento do modelo lógico é uma área em crescente evolução. Pesquisas na área de geração de interfaces com o usuário a partir de artefatos criados nas etapas anteriores no processo de software têm sido bastante publicadas.

Becker [Becker, 2009] propôs um aplicativo para geração de interfaces, considerando os critérios ergonômicos de usabilidade, a partir do diagrama de classes do domínio.

Alguns trabalhos apresentam técnicas para modelagem de interfaces em uma linguagem abstrata que posteriormente são mapeadas para interfaces concretas de diferentes plataformas (dispositivos moveis, TV Digital, Web, etc). Como exemplo, o XIML (eXtensible Interface Makup Language) que "é uma linguagem de representação baseada em XML que visa o suporte universal de funcionalidade da aplicação ao longo de todo o ciclo de vida de uma interface de usuário. O foco principal de sua abordagem é o desenvolvimento fundamentado em modelos." [MOURA, 2004]

Com o desenvolvimento de aplicações para múltiplas plataformas, os Ambientes de Desenvolvimento de Interfaces de Usuários Baseados em Modelos passam a ser cada vez mais utilizados. A UIIMT (Ferramenta para Modelagem de Interação e Interface de Usuário) [LAVÔR, LEITE. 2011] é uma ferramenta para desenvolvimento de um modelo que representa o processo de interação usuário-sistema, a partir do qual o designer de interfaces pode analisar e encontrar possíveis problemas de usabilidade. A

ferramenta ainda permite que o designer acesse uma biblioteca de padrões, possibilitando o reuso de estruturas nos modelos de interação.

Existem também trabalhos que propõem o desenvolvimento de interfaces Web através da utilização de padrões. Um exemplo é a ferramenta G.I. (Gerador de Interfaces) [PETERMANN, BELLIN, KROTH, 2001]. Na época da sua proposta, com a evolução da internet, viu-se a necessidade de migrar os sistemas para a web, exigindo a realização de um processo de adaptação das interfaces existentes para o novo ambiente. A G.I. é uma ferramenta de auxílio ao desenvolvimento das interfaces para aplicações web, utilizando padrões definidos pelo desenvolvedor.

Entretanto, não foi encontrado nenhum aplicativo consagrado para geração de interfaces de usuário concretas a partir de artefatos elaborados na etapa de levantamento de requisitos do processo de software.

3.1. Proposta de Becker

A ferramenta [BECKER,2009] foi desenvolvida utilizando a linguagem C-Sharp (C#) e gera interfaces WPF (Windows Presentation Foundation, tecnologia da Microsoft) de forma semi-automática, considerando os aspectos ergonômicos de usabilidade, a partir de um diagrama de classes em arquivo XMI e de um arquivo XML contendo os parâmetros de usabilidade. Este arquivo de usabilidade pode ser criado com o auxílio da própria ferramenta.

Inicialmente a ferramenta gera um arquivo XAML (*eXtensible Application Markup Language*) a partir do diagrama de classes e dos parâmetros de usabilidade definidos inicialmente. Em um segundo momento estes arquivos XAML são mapeados

para interfaces WPF. O processo de geração das páginas é dividido em duas etapas, pois tem como objetivo permitir a modificação das interfaces de maneira fácil pelo usuário, não precisando assim que o usuário tenha conhecimento mais avançado da tecnologia WPF para realizar alguma eventual modificação nas interfaces geradas. A ferramenta desenvolvida dá suporte para à modificação destes arquivos.

A partir do diagrama de classes as interfaces são geradas segundo os seguintes critérios:

a) “classes sem relacionamentos geram somente interfaces com uma aba principal, denominada cadastro;

b) classes com relacionamentos do tipo associação n x n, geram interfaces com uma aba principal denominada cadastro e dentro desta aba, é criado para cada relacionamento uma nova aba como o nome da classe relacionada;

c) classes com relacionamentos do tipo agregação 1 x n, geram interfaces com uma aba denominada cadastro e para cada tipo de relacionamento de agregação, cria-se uma aba, sendo estas uma ao lado da outra;

d) atributos do tipo int, float e double geram campos textbox alinhados à direita;

e) atributos do tipo string e char geram campos texbox alinhados à esquerda;

f) atributos do tipo mascaraValor, mascaraData, mascaraHora e mascaraTelefone geram campos do tipo MyMaskedTextBox com o valor da mascara definida pelo usuário no aplicativo;

g) atributos do tipo class, exemplo UF:UF, são automaticamente trocado, pelo sistema, para Associado1x1, Associado1xN, AssociadoNxN ou Agregado1xN;

h) os atributos do tipo Associado1x1 e Associado1xN geram campos do tipo combobox dentro da aba pertencente a sua origem;

i) os atributos do tipo AssociadoNxN geram dentro da aba de sua origem um campo do tipo combobox, um listbox para adicionar os valores, um botão de adicionar, um botão para remover e um botão de procurar;

j) os atributos do tipo Agregado1xN, busca todos os atributos que estão no relacionamento de destino e para cada atributo geram um campo conforme o seu tipo, dentro da sua aba de origem.”

A geração da interface não é completamente automatizada, pois o intuito da ferramenta é fazer com que as interfaces geradas satisfaçam os critérios de usabilidade citados a seguir, e alguns destes critérios precisam de configuração do usuário:

- Usabilidade – inteligibilidade: interface organizada em grupos, mensagens explicativas ao posicionar o mouse sobre um botão, etc;

- Aspectos visuais: distribuição dos objetos, campos de entrada de dados compatíveis com a necessidade, identificação do formato dos campos de entrada de dados, diferenciação dos ícones habilitados e não habilitados com relação ao contexto, alinhamento dos campos alfa numéricos à esquerda e dos campos numéricos à direita;

- Exibição de mensagens de orientação ao usuário;

- Usabilidade – operacionalidade: definição de teclas de atalho;

- Prevenção de erros: impossibilita a entrada de dados alfa numéricos em campos numéricos, desativação das mensagens dos botões para não exercer ações não permitidas.

3.2. G.I. (Gerador de Interfaces)

O G.I. é uma ferramenta antiga e na época, devido as limitações da tecnologia, o custo de desenvolvimento de aplicações web era muito superior comparado com o desenvolvimento de aplicações desktop. Porém, já era defendida a padronização das interfaces através da utilização de templates.

> "O Gerador de Interface utiliza a especificação de interfaces armazenadas em uma base de dados para posterior geração de código. Desta forma, consegue-se o reuso de padrões já definidos. Assim, por exemplo, os formulários de uma mesma aplicação passam a apresentar uma interface baseada em regras definidas, resultando em uma aparência uniforme em todo o sistema." [Petermann, Bellin e Kroth, 2001]

A ferramenta gera as interfaces a partir de um template, um layout e um formulário. A padronização é garantida através dos templates e layouts. No template são definidas as características da aparência dos componentes de uma interface, tais como cor da fonte, tamanho da fonte, botões, labels, etc. O layout tem como finalidade a construção de um padrão para disposição dos componentes de uma determinada interface.

Cada interface da aplicação terá um formulário aonde são inseridos os componentes da interface, cada formulário deve ter um template e um layout associado a ele. A ferramenta proporciona um assistente para a criação destes formulários que

podem ser de um dos seguintes tipos: Formulário simples, Mestre-Detalhe ou Filtro. A ferramenta ainda cria uma página gerenciadora, onde contém um Menu, que permitirá o acesso aos diferentes formulários criados, assim obtendo-se uma interface completa para um sistema de informação.

3.3. XIML

O XIML é fundamentado em modelos para facilitar a portabilidade da interface de uma aplicação. Uma interface abstrata representada em XIML é composta de vários widgets abstratos, então o sistema busca as informações sobre a plataforma alvo e realiza o mapeamento dos widgets abstratos para widgets concretos.

O processo de projeto de uma interface XIML consiste na elaboração de três modelos: o modelo de tarefa, o modelo de plataforma e o modelo de apresentação. É no modelo de tarefa que são incluídas todas as tarefas que o usuário poderá realizar na aplicação, ele contém de maneira estruturada todas as informações relacionadas aos objetos, às pré-condições e às pós-condições de cada uma das tarefas.

O modelo de plataforma apresenta a declaração de cada elemento de uma plataforma específica, contém os atributos que descrevem suas características e restrições. Uma aplicação pode possuir suporte para mais de um tipo de interface desde que sejam declarados os elementos de todas as interfaces neste modelo.

A aparência visual da interface é detalhada no modelo de apresentação. É neste modelo que são definidos os widgets concretos (objetos de interface concretos) que irão representar os widgets abstratos (objetos da interface abstrata) em uma interface concreta. Ou seja, este modelo realiza a ligação entre o modelo de tarefa com o modelo

de plataforma, permitindo, assim, que a interface da aplicação seja implementada em diferentes plataformas.

3.4. UIIMT

O modelo de representação do processo de interação usuário-sistema desenvolvido na ferramenta UIIMT é especificado pela linguagem ALaDIM (*Abstract Language for Description of Interactive Message*). "Ela permite representar de forma diagramática, a interação usuário-sistema e os elementos funcionais do sistema." [LAVÔR, LEITE. 2011]

Em ALaDIM, o conceito de *espaço de interação* é utilizado para descrever um conjunto de *interações básicas* que serão realizadas pelo usuário a fim de utilizar uma *função do sistema*. Os elementos de *interações básicas* permitem a entrada de informações, o controle da execução de uma *função de domínio*, ou a visualização do progresso e dos resultados da função. Estes elementos de *interações básicas* podem ser agrupados e organizados por meio de *operadores de interação*, estes operadores são abstrações utilizadas pelo projetista para indicar como o usuário deve interagir. As *funções de sistema* são utilizadas para representar o controle da aplicação, ou seja, as regras de negócios da aplicação. Já os aspectos dinâmicos da interação, *fluxos de controle de apresentação*, são representados por setas direcionais.

Os operadores de interação do usuário podem ser *sequence*, *repeat*, *join* ou *combine*. Já os elementos de informações básicas podem pertencer aos conceitos *perceiveInformation*, *enterInformation*, *selectInformation*, *activate* ou *navigate*.

A Figura 3.1 apresenta um modelo de interação usuário-sistema para a função Copiar Arquivo de um sistema desenvolvido na ferramenta UIIMT. Na figura são apresentados dois exemplos de espaços de interação, o *Copiar Arquivo* e o *Copiando Arquivos.*

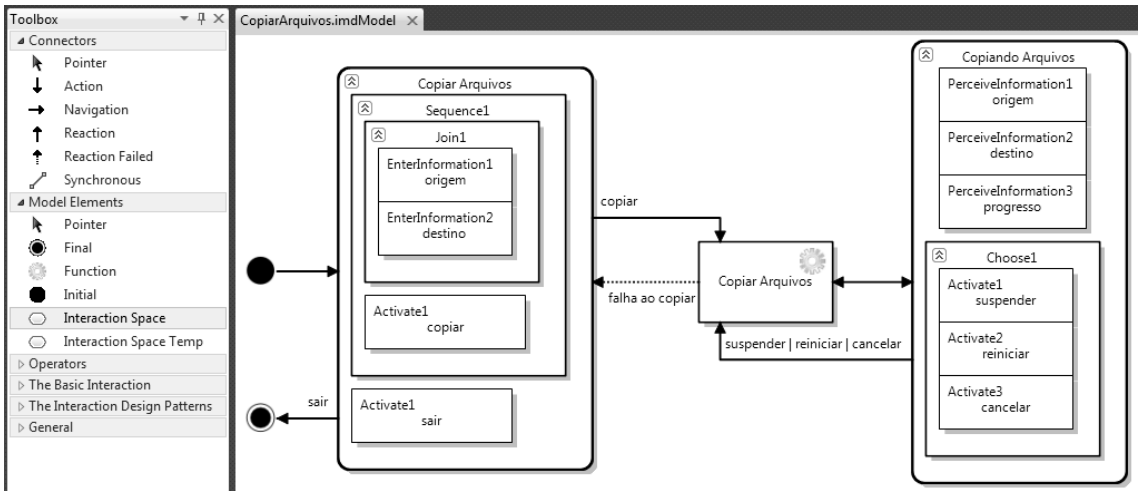


Figura 3.1 – Modelo de Interações na ferramenta UIIMT. [LAVÔR, LEITE. 2011]

Para auxiliar o projetista, a ferramenta possui uma biblioteca de padrões de modelos de interação, permitindo assim o reuso das estruturas na construção dos modelos mais complexos de interação usuário-sistema. A ferramenta ainda permite que o projetista vincule imagens de protótipos aos modelos de interação, a fim de auxiliar os desenvolvedores. Ao final do processo de projeto da interface, é possível visualizar os modelos e protótipos, aonde a ferramenta gera um conjunto de páginas HTML a partir de *templates* que capturam imagens dos modelos, suas informações e as possíveis imagens dos protótipos. A partir da página de um modelo é possível navegar para as páginas de outros modelos que possuem alguma relação com ele. A Figura 3.2 apresenta o exemplo da página para o modelo de interação “Autenticação”.

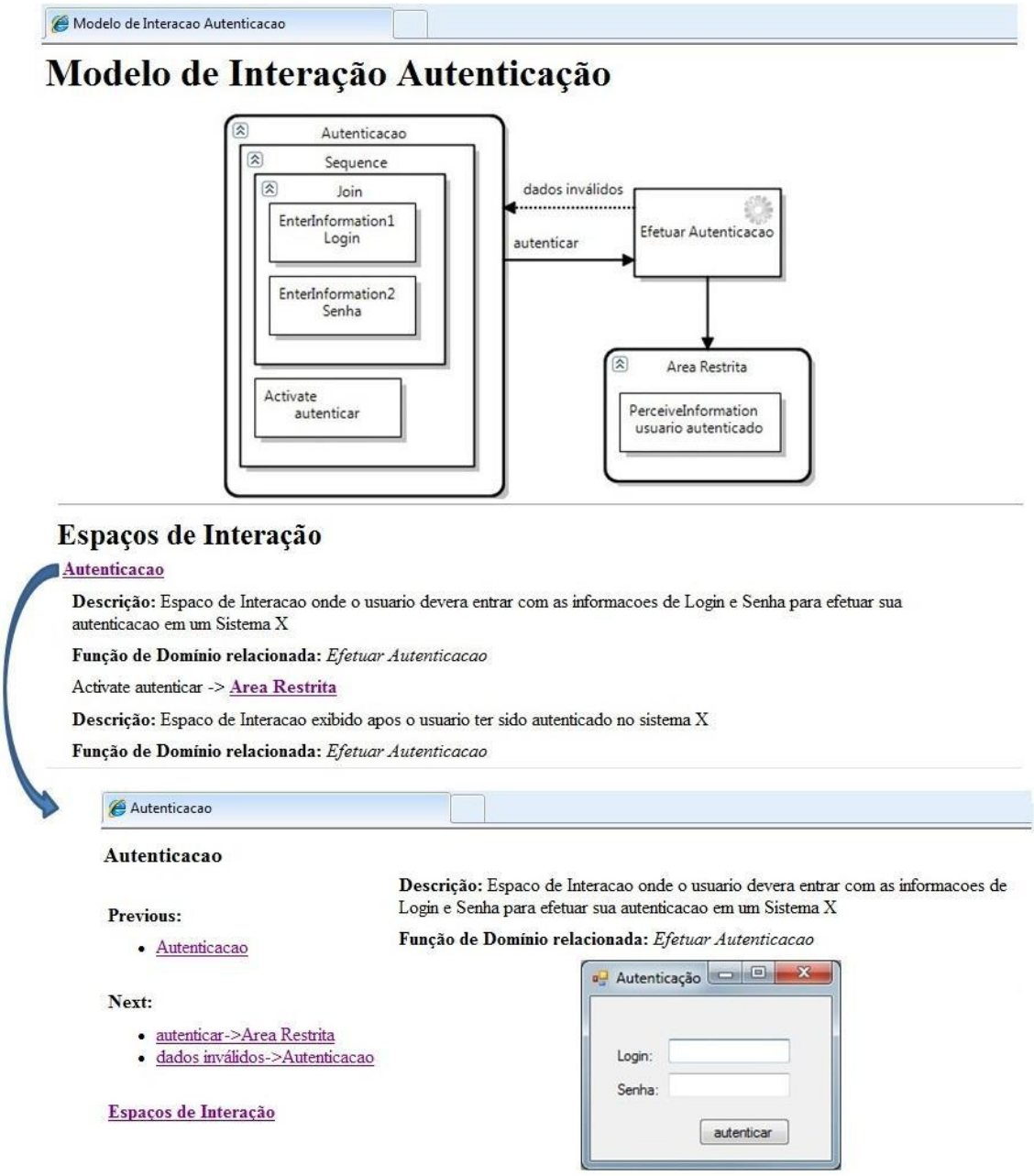


Figura 3.2 – Visualização do modelo integrado aos protótipos de interface. [LAVÔR, LEITE. 2011]

4. Projeto

Este capítulo apresenta a proposta de mapeamento dos elementos do UID para os componentes JSF. Este mapeamento é baseado no mapeamento dos UIDs para a ontologia de widgets abstratos, proposto por [REMÁCULO, 2005], e no relacionamento entre os widgets abstratos e os componentes HTML do JSF que é mostrado no Apêndice A.

4.1. Mapeamento dos elementos do UID para componentes JSF

Para cada componente dos UIDs é definida uma regra do seu mapeamento para os componentes JSF. Também é apresentado um exemplo de mapeamento e, se necessário, algumas considerações explicando como esta regra de mapeamento foi definida.

4.1.1. Saídas do Sistema

São as informações retornadas pelo sistema e que devem ser mostradas ao usuário.

4.1.1.1. Texto: o componente texto deve ser representando pelo componente *outputText* do JSF.

A Figura 4.1 mostra o exemplo de um estado de interação que contém o texto "Entre com o ano de lançamento dos CDs" e que é mapeado para o componente *outputText* do JSF.

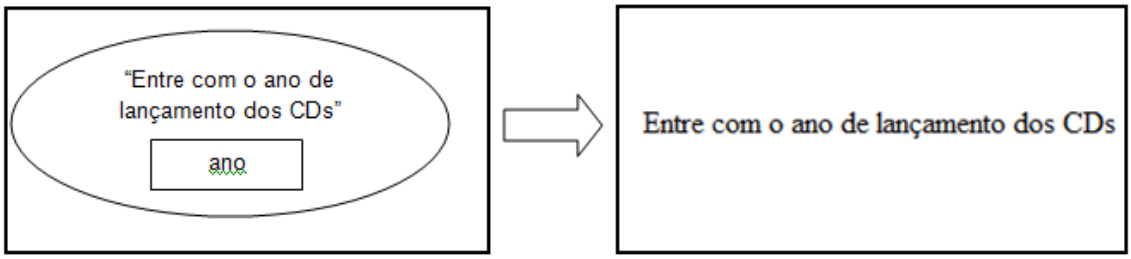


Figura 4.1 – Exemplo de mapeamento de Texto para *outputText*.

Considerações: na ontologia de widgets abstratos um texto é mapeado para o *ElementExhibitor* e no JSF o componente com tal finalidade é *outputText*. Portanto, seu mapeamento se restringe a tal elemento.

4.1.1.2. Item de Dado: um item de dado deve ser mapeado para o componente *commandLink* do JSF caso ele seja origem de alguma transição do UID. Caso contrário, deve ser mapeado para *outputText*, pois este tem como finalidade apenas exibir alguma informação de saída do sistema ao usuário.

A Figura 4.2 mostra o exemplo do mapeamento dos itens de dados "número do pedido", que é origem de uma transição, e "valor total" para os componentes *commandLink* e *outputText*, respectivamente.

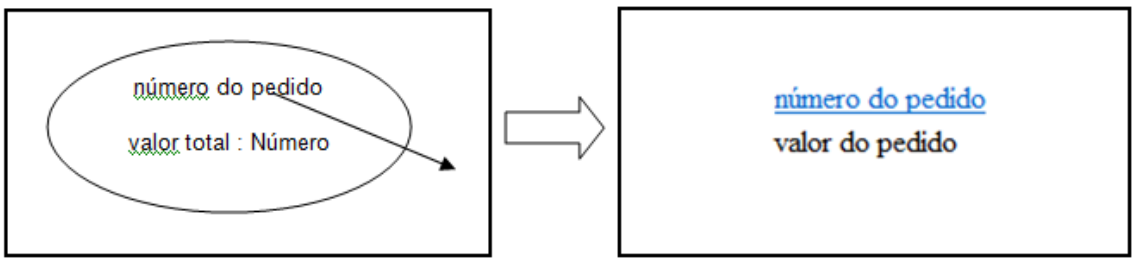


Figura 4.2 – Exemplo de mapeamento de itens de dados para os componentes *outputLink* e *outputText*.

Considerações: no mapeamento dos elementos dos UIDs para a ontologia de widgets abstratos, o item de dado deve ser mapeado para o componente *ElementExhibitor*. Porém, se além de exibir informação, o item de dado ainda fornecer acesso a outra página, ou parte de um mesmo documento, então ele deve ser mapeado para *SimpleActivator*. E no JSF os componentes capazes de representar o *ElementExhibitor* e o *SimpleActivator*, neste contexto, são o *outputText* e o *outputLink*, respectivamente.

4.1.1.3. Estrutura: uma estrutura deve ser mapeada para elementos JSF conforme as seguintes regras:

- Se a estrutura não for origem de nenhuma transição e não possuir elementos, então ela deve ser mapeada para *outputText*. Se ela não for origem de nenhuma transição mas possuir elementos, ela deve ser mapeada para um *panelGrid*, com um *outputLabel* com o nome da estrutura, e seus elementos devem ser mapeados conforme definido anteriormente nas seções 4.2.1.2 (mapeamento de item de dado) e 4.2.1.3 (mapeamento de estrutura).

- Se a estrutura for origem de alguma transição e não possuir elementos, então esta deve ser mapeada para *commandLink*. Porém, se ela for origem de alguma transição e possuir elementos, então ela deve ser mapeada para um *panelGrid*, com um *outputLabel* com o nome da estrutura, e seus elementos devem ser mapeados conforme definido anteriormente nas seções 4.2.1.2 (mapeamento de item de dado) e 4.2.1.3 (mapeamento de estrutura).

A Figura 4.3 mostra o exemplo de um estado de interação que contém duas estruturas, “Cliente”, que não possui elementos, e “CD”, que possui os elementos “nome do cd”, “nome do autor” e “preço”. A estrutura “Cliente” é mapeada para o componente *outputText* do JSF, enquanto a estrutura “CD” é mapeada para um *panelGrid* e seus elementos para o componente *outputText*, que estão incluídos no *panelGrid* referente a estrutura.

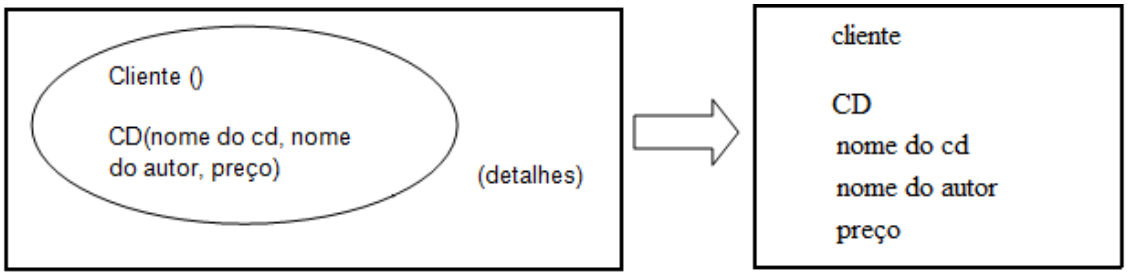


Figura 4.3 – Exemplo de mapeamento das estruturas para componentes JSF.

Considerações: no mapeamento dos UIDs para a ontologia de widgets abstratos [REMÁCUO, 2005] foram apresentadas duas formas de mapeamento de uma estrutura. A primeira é o mapeamento de toda a estrutura para os componentes *ElementExhibitor* ou *SimpleActivator*, e a segunda é o mapeamento da estrutura para o componente *CompositeInterfaceElement* e seus elementos para *ElementExhibitor* ou *SimpleActivator*. O mapeamento das estruturas dos UIDs para os componentes JSF seguiu tais diretivas.

Ainda existe a possibilidade de representar uma estrutura que contém elementos pelo componente *panelGroup* do JSF, porém faz-se necessário definir um estilo CSS (atributo style do componente) para que os elementos fiquem devidamente organizados

na página. Como o componente *panelGrid* já possui uma pré organização configurável dos elementos, ele foi escolhido como padrão.

Também foi definido que uma estrutura sem elementos, mas que é origem de uma transição, é mapeada para *ouputLink*, o componente mais simples que representa o *SimpleActivator*. Porém, ela também poderia ser mapeada para um *commandLink* ou *commandButton*.

4.1.1.4. Conjunto de Itens de Dado: se o conjunto não for origem de nenhuma transição, então ele deve ser mapeado para um *dataTable* com apenas uma coluna e seus elementos, os itens de dados, devem ser mapeados conforme a definição citada na seção 4.2.1.2 e adicionados na coluna da tabela. Caso contrário, isto é, o conjunto de itens de dado é origem de alguma transição, então seu mapeamento deve ser realizado juntamente com a transição, conforme descrito na seção 4.2.3.4.

A Figura 4.4 mostra o mapeamento de um estado de interação que contém o elemento texto "Assuntos relacionados" e o conjunto de itens de dados do tipo "Assuntos". O texto foi mapeado para o componente *outputText* e o conjunto de itens de dados foi mapeado para o componente *panelGrid*, onde cada item de dados pertencente ao conjunto foi mapeado para um *outputText* e está incluído no *panelGrid*.

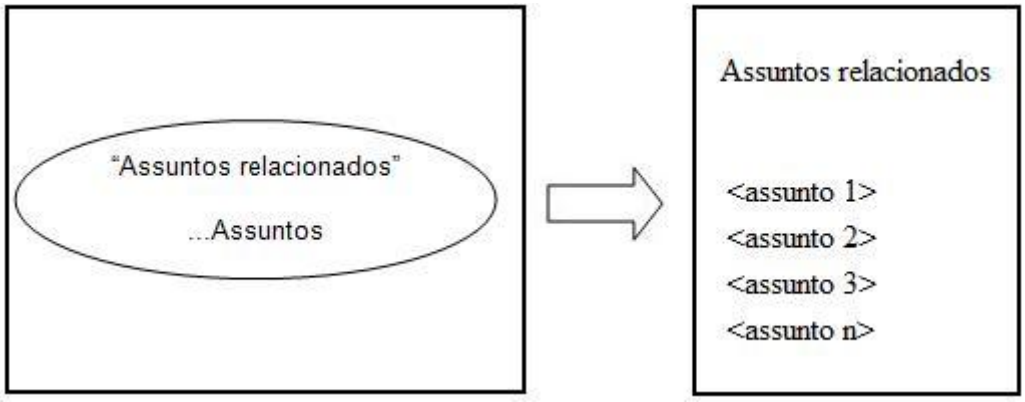


Figura 4.4 – Exemplo do mapeamento de um texto e de um conjunto de itens de dados para componentes JSF.

Considerações: se um conjunto retornado pelo sistema não for origem de nenhuma transição, este tem o papel de apenas expor alguma informação. Considerando esta característica e também o mapeamento para widgets abstratos, foram definidas as regras de mapeamento do conjunto para componentes JSF. Contudo, se o conjunto for origem de alguma transição, ele vai levar a outra página ou parte do mesmo documento. E neste caso o seu mapeamento deve ser realizado juntamente com a transição.

4.1.1.5. Conjunto de Estruturas: se o conjunto não for origem de nenhuma transição, então ele deve ser mapeado para um *dataTable* onde seus elementos, ou seja, as estruturas, serão as linhas da tabela. Os elementos da estrutura formarão as colunas da tabela. Caso contrário, isto é, o conjunto de estruturas é origem de alguma transição, então seu mapeamento deve ser realizado juntamente com a transição, conforme descrito na seção 4.2.3.4.

A Figura 4.5 mostra o exemplo de um estado de interação que contém um conjunto de estruturas do tipo "CD" que possui os elementos "nome do cd", "nome do artista",

“gênero” e “preço”. Este conjunto é mapeado para o componente *dataTable* do JSF e os elementos da estrutura formam as colunas da tabela e cada estrutura pertencente ao conjunto é uma linha da tabela.

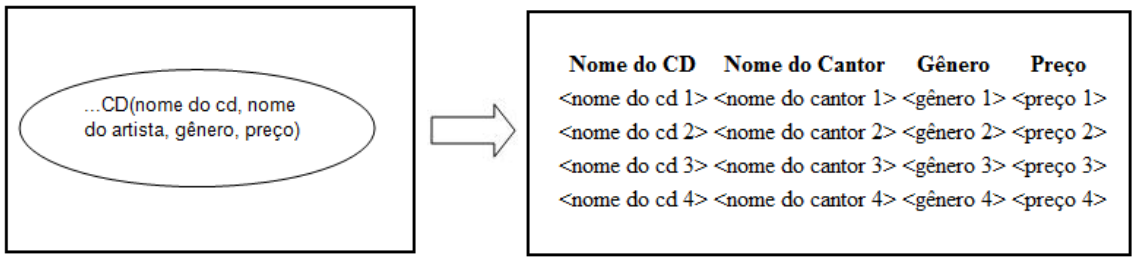


Figura 4.5 – Exemplo do mapeamento de um conjunto de estruturas para componentes JSF.

Considerações: pelo fato de uma estrutura agrupar informações que se relacionam de alguma forma, e um conjunto também agrupar elementos que apresentam alguma característica em comum, foi definido o mapeamento acima citado. Contudo, ainda existe a opção de mapear o conjunto para um *panelGrid* (ou *panelGroup*) e mapear as estruturas conforme a definição do item 3.1.3.

4.1.2. Entradas do Usuário

Em [REMÁCULO, 2005], as entradas de usuário são separadas em dois grupos: entradas de usuário e entradas do usuário enumeradas. No mapeamento dos UIDs para a ontologia de widgets abstratos, a entrada de usuário é mapeada para *ArbitraryValue*, pois é este o componente que permite a entrada de um dado pelo teclado, e as entradas do usuário enumeradas são mapeadas para *SingleChoice* ou *MultipleChoice*. No mapeamento dos UIDs para componentes JSF, cada tipo de entrada de usuário será

mapeado de forma isolada, apresentando uma regra de mapeamento para componente JSF.

Para cada entrada do usuário, independente se realizada por teclado ou não, pode ser definido um rótulo com o nome da mesma. Este rótulo é um *outputLabel* que terá como valor o nome da entrada e como elemento a entrada em questão. Este artifício será utilizado em alguns exemplos para melhor compreensão.

Por padrão, foi definido que as entradas do usuário realizadas pelo teclado serão mapeadas para o componente *inputText*. Porém, elas também podem ser mapeadas para o componente *inputSecret*, como no caso de senhas, e *inputTextArea*, no caso de textos longos como a descrição de um produto.

4.1.2.1. Item de Dado: a entrada de um item de dado deve ser mapeada para o componente *inputText*.

A Figura 4.6 mostra o exemplo de um estado de interação que contém o texto "Entre com o ano de lançamento dos CDs" e uma entrada do usuário do tipo item de dado "ano". O texto é mapeado para o componente *outputText*, como já foi definido antes. Em relação ao mapeamento do item de dado, ele é mapeado para o componente *inputText*, e ainda foi feito o uso de um *outputLabel*, onde o valor dele é o nome do item de dado "ano".

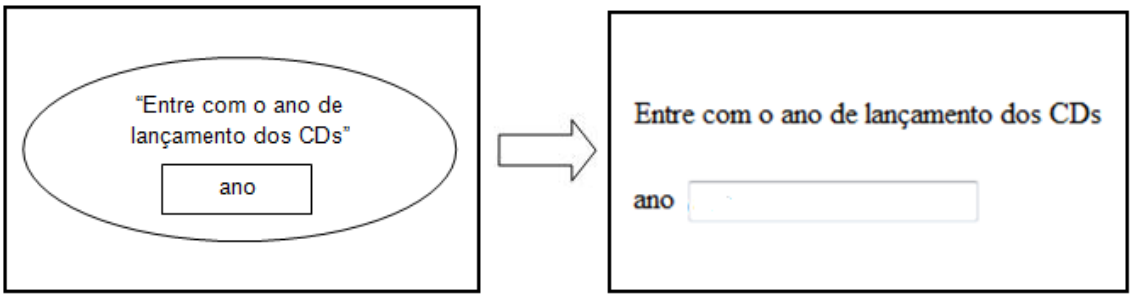


Figura 4.6 – Exemplo do mapeamento de um texto e de uma entrada de usuário do tipo item de dado para componentes JSF.

Considerações: vale lembrar que uma entrada de usuário do tipo item de dado ainda pode ser mapeada para os componentes *inputSecret* e *inputTextArea*. O *inputText* foi definido como padrão pois é o mais utilizado.

4.1.2.2. Estrutura: uma estrutura de entrada do usuário deve ser mapeada para o *panelGrid* do JSF com um *outputLabel* em que o valor será o nome da estrutura. Os elementos que forem itens de dados devem ser mapeados para *inputText* (item 2.2.1) e os elementos que forem outra estrutura devem ser mapeados novamente como uma estrutura. Todos os elementos devem ser inseridos no *panelGrid* para qual a estrutura foi mapeada.

Se a estrutura não possuir elementos, então esta deve ser representada por um *inputText* com um *outputLabel* em que o valor será o nome da estrutura.

A Figura 4.7 mostra o exemplo de um estado de interação que contém uma entrada de usuário do tipo estrutura "Cliente" que possui os elementos "nome", "sobrenome", "email", "login" e "senha". A estrutura é mapeada para o componente *panelGrid* e seus elementos, por serem todos entrada de usuário do tipo item de dados,

são mapeados para componentes *inputText*, que estão incluídos no *panelGrid* referente à estrutura. Ainda foi utilizado um *outputLabel* para o componente *panelGrid*, referente à estrutura, cujo valor é o nome da estrutura, "Cliente".

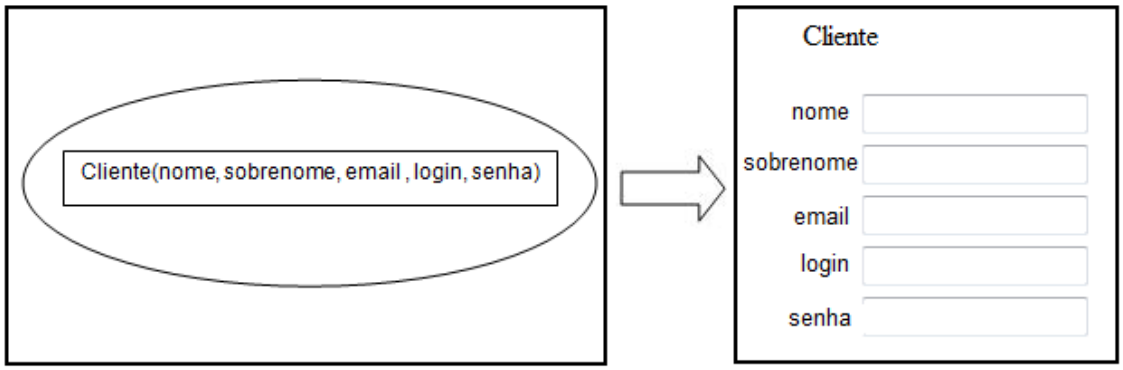


Figura 4.7 – Exemplo do mapeamento de uma entrada de usuário do tipo estrutura para componentes JSF.

4.1.2.3. Conjunto de Itens de Dado: existem duas possibilidades de mapeamento de um conjunto de entrada do usuário de itens de dado para componentes JSF. Como primeira alternativa, se o conjunto é finito, então deve ser criado um total de campos para entrada dos dados igual ao número limite de itens de dado que o usuário possa entrar. Isto é, replicar os *inputText* até o máximo permitido de entradas do conjunto.

Como segunda alternativa, se o conjunto é sem limite ou com limite superior muito alto, deve ser criado apenas um *inputText*. Assim o usuário entra com um item de dado por vez.

A Figura 4.8 mostra o exemplo de um estado de interação que contêm o texto "Entre com até 4 nomes de artistas favoritos" e, também, um conjunto de entrada de itens de dados "Nome", onde o número mínimo de entradas é zero e o máximo é quatro.

O texto foi mapeado para o componente *outputText* e o conjunto, por se tratar de um conjunto finito e de tamanho máximo pequeno, foi mapeado segundo a primeira definição, ou seja, foram criados 4 *inputText*. Ainda foram utilizados 4 *outputLabel*, um para cada *inputText*, referentes aos itens de dados, cujo valor é "Nome", ou seja, o nome do item de dado do conjunto de entrada.

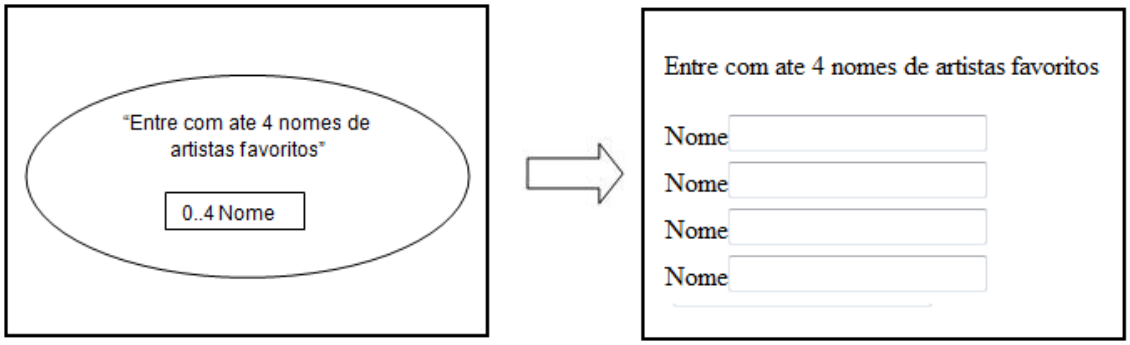


Figura 4.8 – Exemplo do mapeamento de um texto e de um conjunto de entrada de itens de dado para componentes JSF.

4.1.2.4. Conjunto de Estruturas: o mapeamento da entrada de um conjunto de estruturas para elementos JSF deve seguir as seguintes diretivas:

- Se o conjunto é finito, então seu mapeamento deve ser para um *dataTable*, em que cada elemento da estrutura é uma coluna da tabela e cada linha é uma entrada da estrutura. As células da tabela serão componentes do tipo *inputText*.

- Se o conjunto é finito e a estrutura não possui elementos, então seu mapeamento é idêntico ao do conjunto de entrada de itens de dado (seção 4.2.2.3).

- Se o conjunto não possui limite ou o limite superior é muito alto, deve ser criada apenas uma estrutura de entrada (seção 4.2.2.2). Assim, o usuário entra com todas as estruturas que forem necessárias, uma por vez.

A Figura 4.9 mostra o exemplo de um estado de interação que contém o texto "Indique até 3 amigos" e um conjunto de estruturas "Amigo Indicado", onde o número mínimo de entradas é zero e o máximo é três. A estrutura do conjunto possui os elementos "nome" e "email". O texto foi mapeado para o componente *outputText* e o conjunto, por se tratar de um conjunto finito e de tamanho máximo pequeno, foi mapeado segundo a primeira definição, onde foi criada uma tabela em que os elementos "nome" e "email" da estrutura viraram as colunas e as entradas da estrutura viraram as linhas. Neste exemplo não foi utilizado o *outputLabel* como nos exemplos anteriores, mas foi adicionado um cabeçalho na tabela com os nomes dos elementos da estrutura do conjunto.

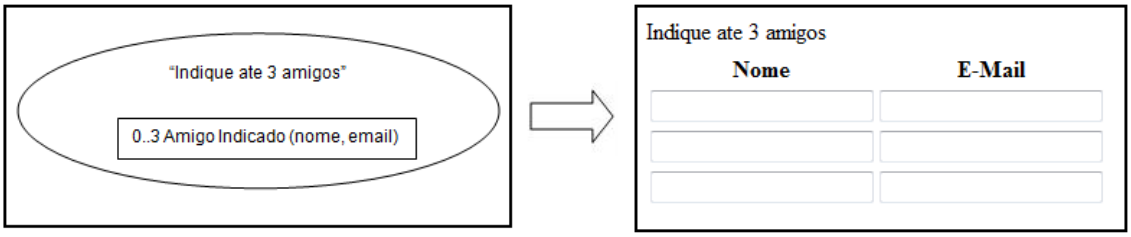


Figura 4.9 – Exemplo do mapeamento de um texto e de um conjunto de estruturas para componentes JSF.

4.1.2.5. Entrada de Usuário Enumerada: quando a escolha é de apenas um item, seu mapeamento deve ser para o componente *selectOneRadio*. Se mais de um item pode ser selecionado, então a entrada enumerada deve ser mapeada para o componente *selectManyCheckbox*.

A Figura 4.10 mostra o exemplo de um estado de interação que contém duas entradas de usuário do tipo enumeração. A primeira enumeração, “Pagamento”, é restrita à seleção de apenas um item dentre as duas opções, “cheque” ou “cartão de crédito”, e por isto foi mapeada para o componente *selectOneRadio*. A segunda enumeração, “Embalagem Extra”, permite a seleção de zero a duas das opções “presente” e “transporte”, e por permitir uma seleção diferente de uma opção, foi mapeada para o componente *selectManyCheckbox*.

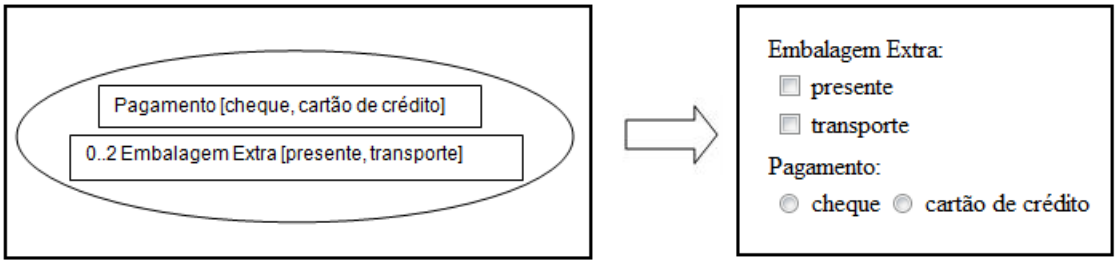


Figura 4.10 – Exemplo do mapeamento de duas enumerações para componentes JSF.

Considerações: o mapeamento para o componente *selectOneRadio* pode ser substituído por *selectOneListbox* ou *selectOneMenu*. E o elemento *selectManyCheckbox* também pode ser substituído por *selectManyMenu* e *selectManyListbox*.

Porém, como a enumeração dos UIDs geralmente apresenta um conjunto pequeno de opções, os componentes *selectOneRadio* e *selectManyCheckbox* do JSF são os que melhor expõem visualmente as opções disponíveis.

4.1.2.6. Seleção dentre dois Itens de Dados (OR): deve ser representado pelo elemento *selectManyCheckbox* do JSF.

A Figura 4.11 mostra o exemplo de um estado de interação que contém o texto “Opções de pesquisa” e a seleção de apenas um dos itens de dados ou dos dois itens de dados, “cantor” e “compositor”. O texto foi mapeado para o componente *outputText* e a seleção foi mapeada para o componente *selectManyCheckbox*.

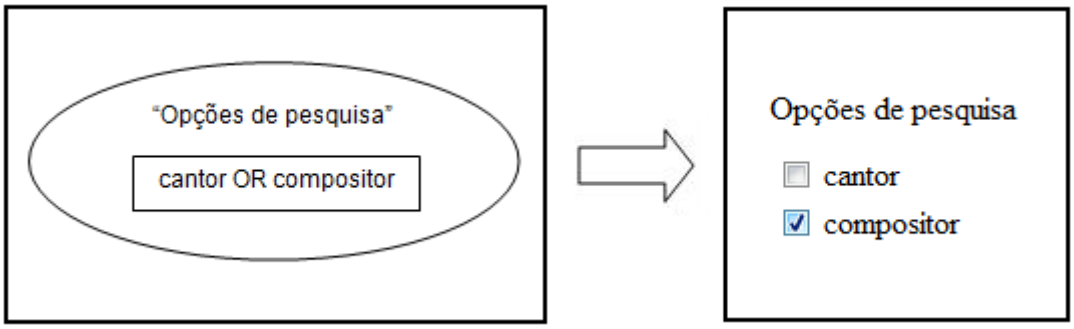


Figura 4.11 – Exemplo do mapeamento de um texto e uma seleção “or” de dois itens de dados para componentes JSF.

Considerações: além do elemento *selectManyCheckbox*, o mapeamento pode ser feito para os componentes *selectManyMenu* ou *selectManyListbox*. Porém, como são apenas duas opções de itens de dados, o componente *selectManyCheckbox* do JSF é o mais indicado.

4.1.2.7. Seleção de um Item de Dado (XOR): no JSF, a seleção exclusiva de um item de dado dentre duas opções deve ser representada pelo componente *selectOneRadio*.

A Figura 4.12 mostra o exemplo de um estado da interação que contêm o texto “Opção de pagamento” e a seleção exclusiva de um item de dado dentre as opções “cheque” e “cartão de crédito”. O texto foi mapeado para o componente *outputText* e a seleção foi mapeada para o componente *selectOneRadio*.

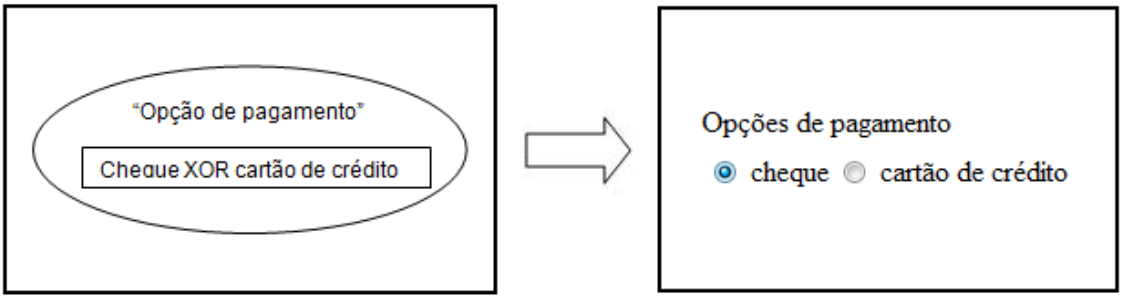


Figura 4.12 – Exemplo do mapeamento de um texto e uma seleção exclusiva de um item de dado para componentes JSF.

Considerações: entre as três formas possíveis de mapeamento de um item de dado XOR para componentes JSF (*selectOneMenu*, *selectOnListbox* e *selectOneRadio*), a que melhor representa que somente uma entre as duas únicas opções será escolhida é o componente *selectOneRadio* e, por isto, foi definido como padrão.

4.1.2.8. Entradas Opcionais: todas as entradas de usuário também podem ser opcionais. O mapeamento delas deve ser idêntico aos apresentados acima (seção 4.2.2.1 à seção 4.2.2.7).

4.1.3. Demais elementos do UID

O mapeamento dos elementos estado da interação, estado inicial da interação, sub-estado de um estado da interação, transição com seleção da opção X e transição com seleção de N elementos para componentes JSF é definido nesta seção.

Já para o grupo restante de elementos, composto por **chamada de outro UID, chamada a partir de outro UID, pré-condições, pós-condições, parâmetros e notas textuais**, não foi possível encontrar nenhuma representação por parte dos componentes

JSF, podendo ser utilizados como informação adicional para auxílio no mapeamento dos outros elementos.

4.1.3.1. Estado da Interação e Estado Inicial da Interação: os estados da interação e o estado inicial de interação de um UID devem ser mapeados segundo as seguintes definições:

a) Se o UID possuir um único estado, ou seja, possuir apenas o estado inicial, então este deve ser mapeado para uma nova página. Esta página deve conter o componente *form*, onde os elementos do estado serão incluídos.

b) Se o UID possuir um estado que possui elementos de entrada do usuário, e este estado possuir uma transição simples, isto é, uma transição sem a seleção de uma opção e sem a seleção de N elementos, que leva a outro estado que também possui elementos de entrada do usuário, e assim por diante, então estes estados devem ser mapeados para uma única página com um único componente do tipo *form*, onde serão incluídos os elementos de todos os estados envolvidos neste contexto de entrada de dados.

c) Se um estado que contém elementos de entrada de usuário do tipo item de dado, precede outro estado, isto é, estão ligados por uma transição simples, que possui um conjunto de estruturas e esta estrutura contém como atributos os itens de dado com os mesmos nomes dos itens de dados do estado anterior, então os dois estados são mapeados para uma única página com dois componentes do tipo *form*, um para cada estado, e deve ser adicionado no *form* do primeiro estado, além dos seus elementos, um botão de *submit* (componente *commandLink* ou

commandButton) que, ao ser acionado, fará com que as informações do segundo *form* sejam atualizadas.

d) Para os demais casos, cada estado deve ser mapeado para uma nova página com um componente *form*, onde os elementos do estado serão incluídos.

As Figuras 4.13 e 4.14 mostram o mapeamento dos estados 1 e 2 do UID para uma única página, onde cada estado foi mapeado para um *form*, pois o estado 1 possui uma entrada do usuário do tipo item de dado "nome do cantor" e o estado 2 possui a estrutura CD que possui o item de dado "nome do cantor" apresentado no estado 1. O primeiro *form* possui dois componentes, um *inputText* e um *commandButton* que foi definido com o nome de "Pesquisar". Já o segundo *form* possui uma *dataTable* referente ao conjunto de estruturas CD retornado pelo sistema. Na figura 4.13 o formulário ainda não foi renderizado com as informações disponíveis no estado 2 do UID, pois ainda não foi submetida nenhuma pesquisa ao sistema. Já na figura 4.14, pode-se observar o resultado da pesquisa referente às informações do estado 2 do UID, após o usuário entrar com o item de dado "cantor 2" e clicar no *commandButton* "Pesquisar". Nestes exemplos foi desconsiderada a chamada do UID "Mostrar Gravações de um CD" e a transição com a condição "mostrar CD" e seleção exclusiva de 1 CD. Por isto, o conjunto de CD do estado 2 foi mapeado para uma *dataTable*.

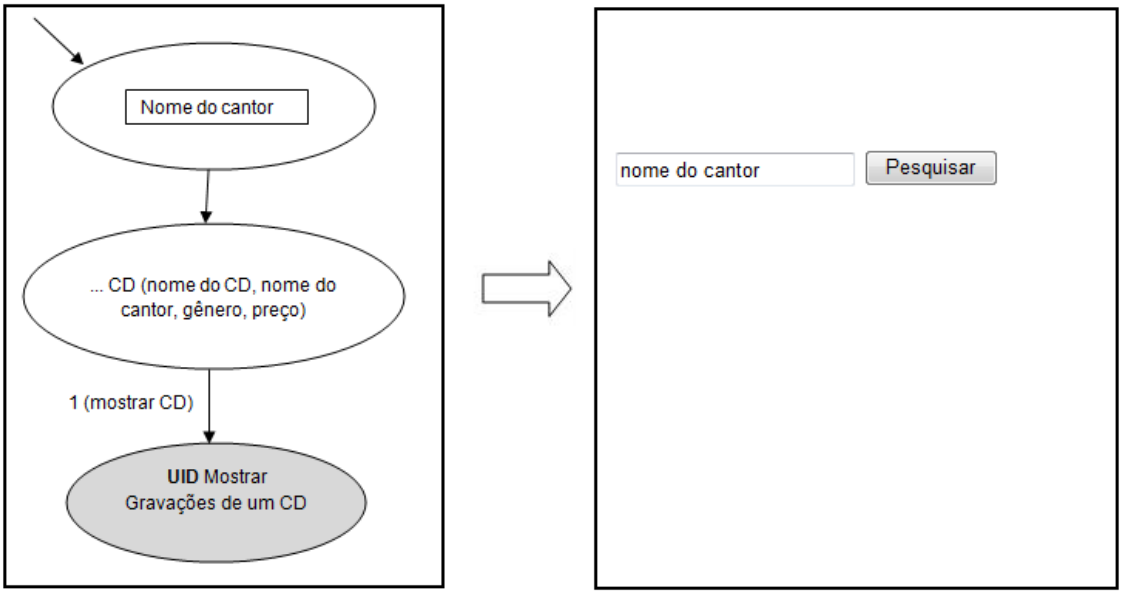


Figura 4.13 – Exemplo do mapeamento dos estados de um UID para páginas JSF com o estado da página antes de a pesquisa ser submetida.

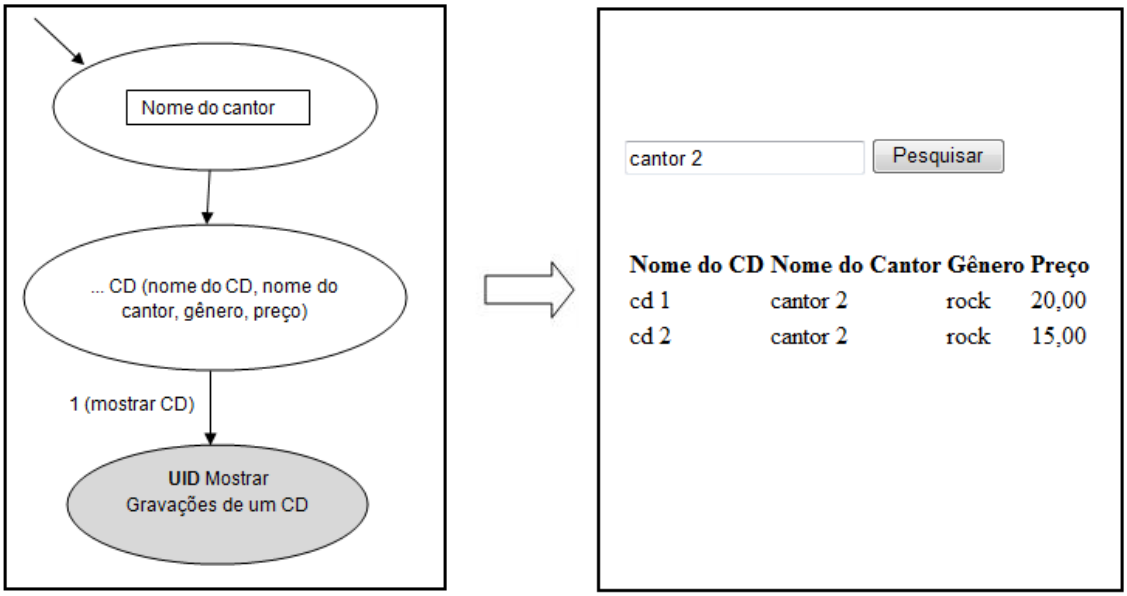


Figura 4.14 – Exemplo do mapeamento dos estados de um UID para páginas JSF com o estado da página após a pesquisa ser submetida.

A Figura 4.15 exemplifica o mapeamento dos estados de um UID para diferentes páginas. Como estes estados não apresentaram as características necessárias para que fossem mapeados segundo as regras **a**, **b** ou **c**, cada um deles foi mapeado para uma página diferente. O estado inicial possui apenas uma estrutura "Cliente" com os elementos "nome" e "email", e foi mapeado para uma página com um *form* que contém um *panelGrid* com dois *outputText* referentes aos elementos da estrutura "Cliente". Já o segundo estado possui quatro entradas de usuário opcionais do tipo item de dado, sejam elas: "nome", "email", "login" e "senha". Este estado foi mapeado para uma nova página que contém um *form* com quatro *inputText* referentes aos itens de dados opcionais.

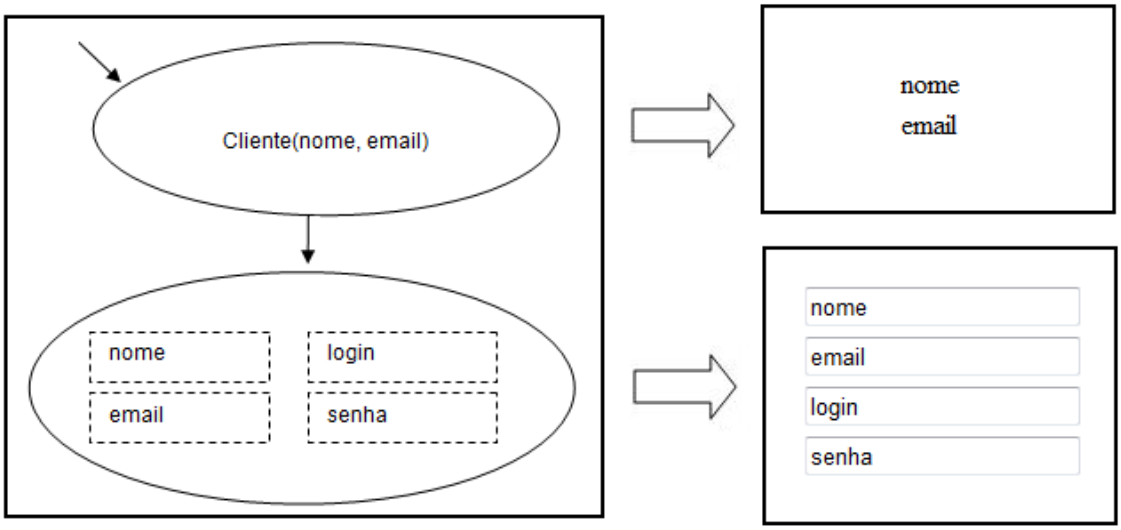


Figura 4.15 – Exemplo do mapeamento dos estados de um UID para diferentes páginas JSF.

Considerações: na prática, pode acontecer de dois estados de interação que poderiam aparecer em uma mesma página, ficarem em páginas distintas. Na geração

automática, devido à dificuldade de analisar a semântica dos elementos e as diferentes formas de construção dos UIDs, é complicado identificar a ocorrência desta situação. Entretanto, foram definidas as regras **a, b** e **c** que visam agrupar diferentes estados, possivelmente pertencentes a um mesmo contexto, em uma única página.

4.1.3.2. Sub-estado de um Estado de Interação: um sub-estado de um estado de interação deve ser mapeado para o elemento *form*.

A Figura 4.16 mostra o mapeamento do estado inicial de um UID que contêm dois sub-estados, para componentes JSF. O estado inicial foi mapeado para uma página e nesta página foram adicionados dois componentes do tipo *form* referentes aos sub-estados do estado inicial. No primeiro *form* foram inseridos dois *inputText* referentes às entradas de usuários do tipo item de dado "login" e "senha". No segundo *form* foi inserido um único *inputText* referente ao único elemento do segundo sub-estado, a entrada de usuário do tipo item de dado "nome do CD". As transições e demais componentes do UID foram desconsiderados neste exemplo.

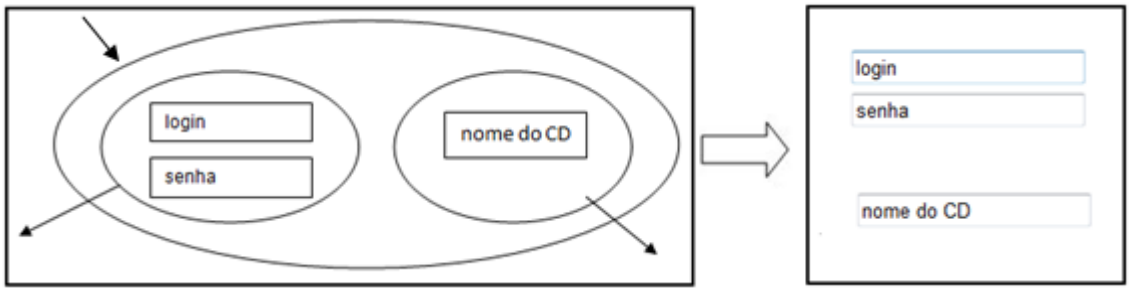


Figura 4.16 – Exemplo do mapeamento dos sub-estados de um estado para componentes JSF.

4.1.3.3. Transição com Seleção da Opção X: deve ser mapeada para elementos JSF segundo as regras:

- Se a transição tiver como origem um item de dado, então nada deve ser feito, pois ela já esta incorporada no elemento *commandLink* referente ao item de dado.

- Se a transição tiver como origem uma estrutura com atributos, então deve ser adicionado um componente *commandLink* no componente *panelGrid* referente à estrutura. Se a estrutura não possuir elementos, então ela deve ser mapeada para um componente *commandLink* qual já incorpora a transição.

- Se a transição tiver como origem um estado ou sub-estado da interação então deve ser adicionado um componente *commandLink* no componente *form* referente ao estado ou sub-estado de origem da transição.

A Figura 4.17 mostra o mapeamento de um UID para duas páginas, uma referente ao estado inicial e a outra referente ao outro estado do UID. O *form* da página do estado inicial contém um *panelGrid* e dois *outputText* referentes a estrutura "Cliente" e seus atributos "nome" e "email", e ainda um *commandLink* referente a transição com a opção "editar", pois a transição tem como origem uma estrutura que possui elementos, sendo então representada por um *commandLink* inserido no *panelGrid* referente a esta estrutura. Já o *form* da segunda página, referente ao outro estado do UID, contém quatro componentes *inputText* que representam as quatro entradas de usuário opcionais do tipo item de dados "nome", "email", "login" e "senha", e ainda possui um *commandLink* que representa a transição com seleção da opção "confirmar", já que transição tem como origem um estado e, por isso, deve ser representada por um *commandLink* inserido no *form* que representa este estado. Quando o usuário clicar no *commandLink* "editar", da página referente ao primeiro estado, ele será redirecionado para a página referente ao segundo estado, e uma vez na página do

segundo estado, ao clicar no *comandLink* "Confirmar", ele será redirecionado, novamente, à página referente ao primeiro estado.

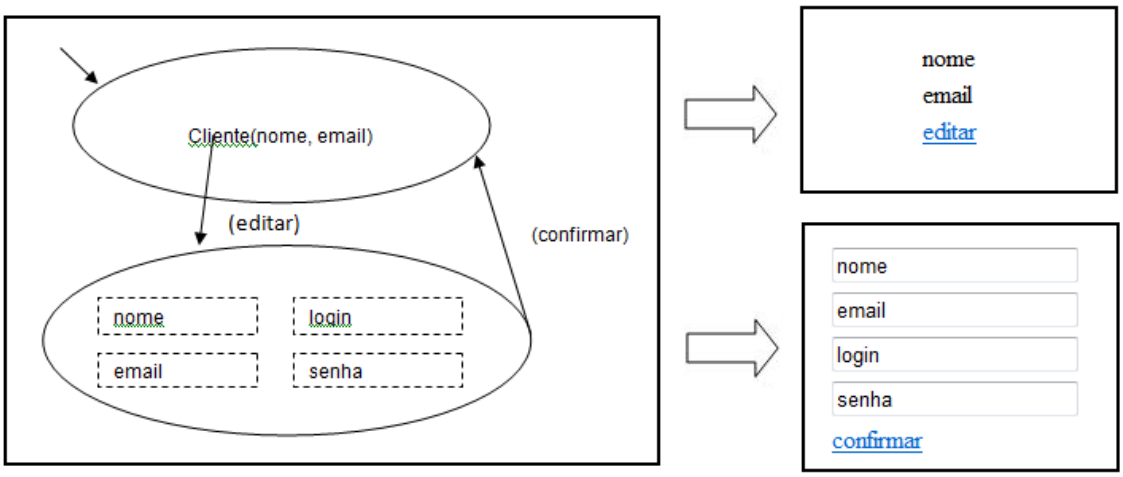


Figura 4.17 – Exemplo do mapeamento do UID que contém transição com seleção da opção X para componentes JSF.

Considerações: foi escolhido como padrão o componente *commandLink* por ser mais flexível para aplicação de estilos CSS. Porém, vale lembrar que o elemento *commandButton* apresenta as mesmas funcionalidades que o *commandLink*.

4.1.3.4. Transição com Seleção de N Elementos: para uma transição que tenha como origem um conjunto de itens de dado:

- Se a transição for de seleção de um único item, isto é, N igual a 1, então o conjunto de itens de dados deve ser mapeado para um *dataTabel* com uma única coluna onde cada item de dado será um *commandLink*.

- Para transições com valor de seleção diferente de 1, mapear o conjunto para o componente *selectManyCheckbox* onde cada item de dado será um *selectItem* do

selectManyCheckbox. Adicionar no mesmo *form* em que está o elemento *selectManyCheckbox* um *commandLink* referente a transição.

Já para uma transição que tenha como origem um conjunto de estruturas:

- Se a transição restringir a seleção de uma única estrutura, o conjunto deve ser mapeado para o componente *dataTable*, onde os elementos da estrutura serão as colunas. Deve ser adicionada uma coluna do tipo *commandLink* para representar esta transição.

- Se transição requer um valor de seleção diferente de 1, então mapear o conjunto para um *dataTable* em que os elementos da estrutura serão as colunas. Adicionar uma coluna do tipo *selectManyCheckbox* para a seleção da estrutura daquela linha. Ainda adicionar no mesmo *form* em que está inserida a tabela um *commandLink* para representar a transição.

A Figura 4.18 mostra o mapeamento do estado de um UID que contém um conjunto de estruturas “CD”, onde esta estrutura possui os elementos “nome do cd”, “nome do cantor” e “preço”. O conjunto ainda apresenta uma cardinalidade de 1 à N e é origem de uma transição com a seleção de 1 à N elementos e da opção “comprar”. O estado foi mapeado para uma página com um componente *form*. No *form* foram inseridos um *dataTable* referente ao conjunto de estruturas “CD” e um *commandLink* referente a transição “comprar”. As demais transições e componentes do UID foram desconsiderados neste exemplo.

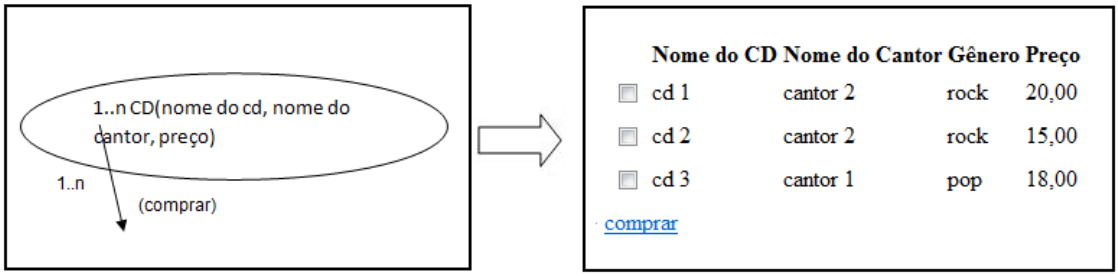


Figura 4.18 – Exemplo do mapeamento dos elementos de um estado de iteração que contém uma Transição com Seleção de N Elementos para componentes JSF.

Considerações: o *commandLink* pode ser substituído por um *commandButton*.

4.1.4. Tabela de mapeamento do UID para JSF

A Tabela 4.1 apresenta, resumidamente, o mapeamento de cada elemento do UID para componentes JSF.

Tabela 4.1 - Tabela de mapeamento dos elementos do UID para componentes JSF:

Entradas do Usuário	
UID	**JSF**
Item de Dado	Input Text
Estrutura	Input Text com um Output Label Panel Grid com um Output Label
Conjunto de Itens de Dado	Um Input Text para cada item de dado do conjunto Apenas um Input Text
Conjunto de Estruturas	Data Table com uma coluna do tipo Inpu Text para cada elemento da estrutura Apenas uma Estrutura de entrada de usuário
Entrada de usuário Enumerada	Select One Radio Select Many Checkbox
Seleção dentre dois Itens de Dados (OR)	Select Many Checkbox
Seleção de um Item de Dado (XOR)	Select One Radio

Saídas do Sistema	
UID	**JSF**
Texto	Output Text
Item de Dado	Output Text Command Link
Estrutura	Output Text Command Link Panel Grid com Output Label
Conjunto de Itens de Dado	Data Table com apenas uma coluna.
Conjunto de Estruturas	Data Table com uma coluna para cada elemento da estrutura.
Demais Elementos	
UID	**JSF**
Estado da Interação, Estado Inicial da Interação	Página XHTML com Form
Sub-estado de um Estado de Interação	Form
Transição com Seleção de N Elementos	Command Link Select Many Checkbox
Transição com Seleção da Opção X	Command Link

5. Desenvolvimento da ferramenta

Com o intuito de automatizar a geração das páginas JSF a partir dos UIDs, foi desenvolvido um protótipo da ferramenta a partir das regras de mapeamento definidas no capitulo 4. A ferramenta foi desenvolvida em linguagem Java com o auxílio do IDE (Integrated Development Environment) Eclipse.

O processo de desenvolvimento da ferramenta foi baseado em práticas ágeis de desenvolvimento de software. Foi tomado como base o framework de comparação e análise de diversos métodos ágeis [FAGUNDES 2005]. Este framework reúne práticas ágeis de diversos métodos ágeis, visando, assim, facilitar a definição de novos processos ágeis.

5.1. Definição do processo de desenvolvimento

O processo de desenvolvimento da ferramenta foi definido a partir da seleção de algumas práticas ágeis apresentadas pelo framework de comparação e análise dos métodos ágeis. As práticas ágeis que foram selecionadas são apresentadas a seguir, agrupadas por atividades:

- Atividades de Definição dos Requisitos
    - Lista de Requisitos – Elaboração de um documento contendo todos os requisitos do sistema.
- Atividades de Atribuição de Requisitos às Iterações
    - Planejamento das Iterações – No início de cada iteração deve ser feito um planejamento para definir quais requisitos serão implementados.

- Duração das Iterações – A partir dos requisitos que devem ser implementados em cada iteração, é definida sua duração. Neste trabalho cada iteração teve duração de duas semanas.

- Atividades de Projeto da Arquitetura do Sistema
  - Projeto Geral do Sistema – A partir dos requisitos conhecidos até o momento é elaborado o projeto geral do sistema. Para realização desta tarefa foi utilizado o diagramas de classes da UML.

- Atividades de Desenvolvimento do Incremento do Sistema
  - Implementação dos Requisitos durante cada Iteração – Consiste na geração de código para os requisitos pertencentes à iteração corrente.
  - Integração Paralela ao Desenvolvimento – Integração do código gerado na iteração corrente com os das iterações passadas.

5.2. Desenvolvimento

O UID já possui a especificação de uma DTD (Document Type Definition) para sua representação através de um arquivo XML (Extensible Markup Language). Esta especificação foi desenvolvida por [VILAIN, 2003], visando facilitar o armazenamento e também intercâmbio das instâncias dos UIDs entre aplicações. As interfaces serão geradas a partir destes arquivos XML.

5.2.1. Lista de Requisitos

Os requisitos se resumem, basicamente, na implementação das regras de mapeamento dos elementos do UID para os componentes JSF. Para cada regra, é gerada uma string de declaração de cada componente JSF – esta string é inserida nos arquivos das páginas JSF. Segue a lista dos 32 requisitos levantados para o desenvolvimento da ferramenta:

- Mapeamento do elemento Texto
- Mapeamento dos itens de dados de saída do sistema
- Mapeamento da estrutura de saída do sistema
- Mapeamento do conjunto de itens de dados de saída do sistema
- Mapeamento do conjunto de estruturas de saída do sistema
- Mapeamento do item de dados de entrada do usuário
- Mapeamento da estrutura de entrada do usuário
- Mapeamento do conjunto de itens de dados de entrada do usuário
- Mapeamento do conjunto de estruturas de entrada do usuário
- Mapeamento da entrada de usuário enumerada
- Mapeamento da seleção dentre dois itens de dados
- Mapeamento da seleção de um item de dado
- Mapeamento do estado inicial de interação
- Mapeamento do estado de interação
- Mapeamento do sub-estado de um estado de interação
- Mapeamento da transição com seleção da opção X
- Mapeamento da transição com seleção de N elementos
- Geração da string do componente column
- Geração da string do componente commandLink
- Geração da string do componente dataTable
- Geração da string do componente form
- Geração da string do componente inputText
- Geração da string do componente outputLabel
- Geração da string do componente outputText
- Geração da string do componente panelGrid
- Geração da string do componente selectItem
- Geração da string do componente selectItens
- Geração da string do componente selectManyCheckbox
- Geração da string do componente selectOneRadiobox
- Geração da string e dos arquivos das páginas JSF
- Integração do módulo de leitura dos arquivos dos UIDs
- Elaboração de uma interface simples que permita inserir os arquivos dos UIDs para posterior geração das páginas JSF.

Não houve necessidade de se elaborar casos de uso pra os requisitos mais complexos uma vez que eles já têm sua funcionalidade documentada no capítulo 3 deste trabalho.

5.2.2. Modelo de Projeto

No mesmo trabalho em que foi realizada a especificação da DTD para validação dos UIDs [VILAIN, 2003], foi desenvolvido um framework em Java que realiza a leitura dos arquivos XML dos UIDs criando instâncias de classes de objetos e ainda realiza a validação semântica dos mesmos.

O Modelo de Classes de Projeto do framework de leitura e validação dos UIDs (Figura 5.1) foi utilizado durante o desenvolvimento da ferramenta de mapeamento e serviu como ponto de partida para o projeto da ferramenta, uma vez que o projeto das classes do sistema foi desenvolvido de forma incremental ao longo das iterações.

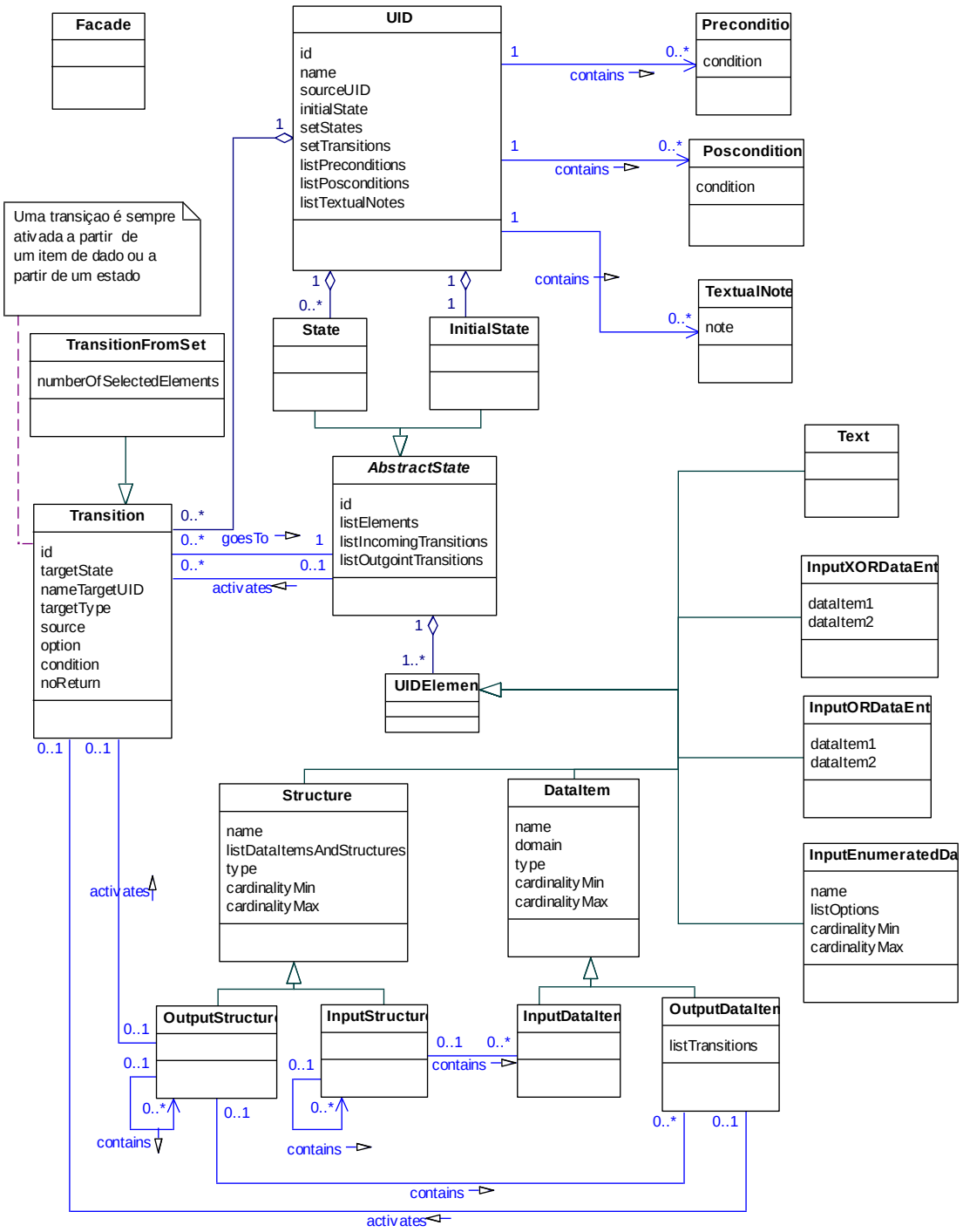


Figura 5.1 – Modelo de Classes de Projeto [Vilain, 2003]

Também foi desenvolvido um Modelo de Projeto, Figura 5.2, que representa a estrutura de uma página JSF, tal página pode ter apenas os componentes padrão das

regras de mapeamento, sejam eles: Form, OutputText, OutputLabel, OutputLink, CommandLink, InputText, PanelGrid, DataTable, Column, SelectManyCheckbox, SelectOneRadio, SelectItem e SelectItems.

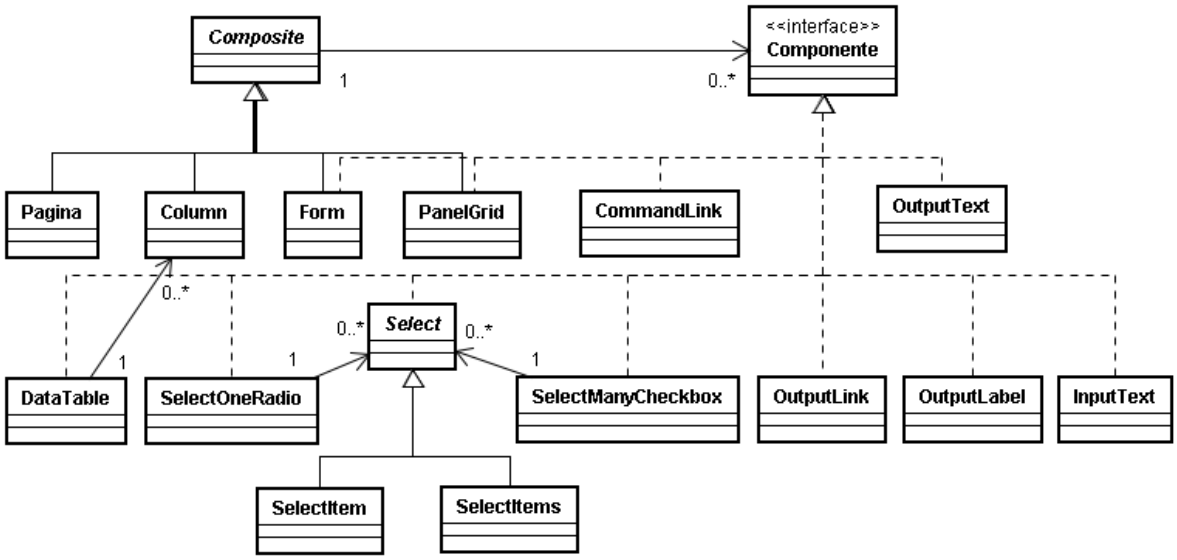


Figura 5.2 – Modelo de Projeto para uma Página JSF com componentes utilizados nas regras de mapeamento.

5.2.3. Primeira iteração

Durante o planejamento da primeira iteração viu-se a necessidade de começar o desenvolvimento da ferramenta a partir do requisito *Integração do módulo de leitura dos arquivos do UID,* seguido dos requisitos *Mapeamento do estado e estado inicial da iteração* e *Mapeamento do sub-estado de um estado de iteração.* Dentre os demais requisitos, optou-se por selecionar todos os requisitos de entrada do usuário, a fim de se familiarizar com as classes do UID.

No início desta iteração foram desenvolvidas as classes Java *MapeamentoUIDparaJSF* (Figura 5.3), *Form, PanelGrid, CompoenenteAbstrato*

InputText, Select, SelectItem, SelectItens, SelectOneRadio e *SelectManyCheckbox,* e ainda, as interfaces Componente e Composite.

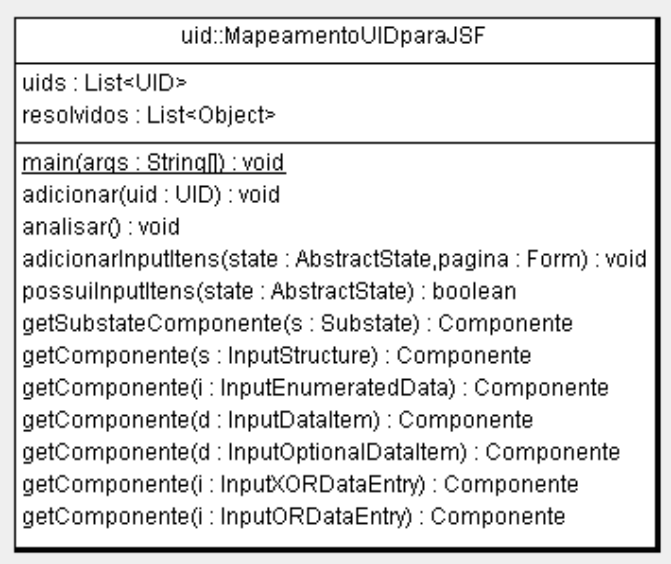


Figura 5.3 – Classe Mapeamento UID para JSF.

5.2.4. Segunda iteração

Para a segunda iteração foram selecionados os requisitos de mapeamento dos demais elementos do UID. Esta iteração foi a que apresentou maior complexidade. A programação das regras de mapeamento dos conjuntos de estruturas juntamente com as transições, como já era esperada, foi o que ocasionou o maior grau de dificuldade. Ao término da iteração, a classe *MapeamentoUIDparaJSF* teve alguns métodos modificados e outros adicionados, como pode ser visto na Figura 5.4, e ainda, cada classe do modelo de classes de projeto da Figura 5.2 possuía uma classe Java que manteve-se com a mesma estrutura até o final.

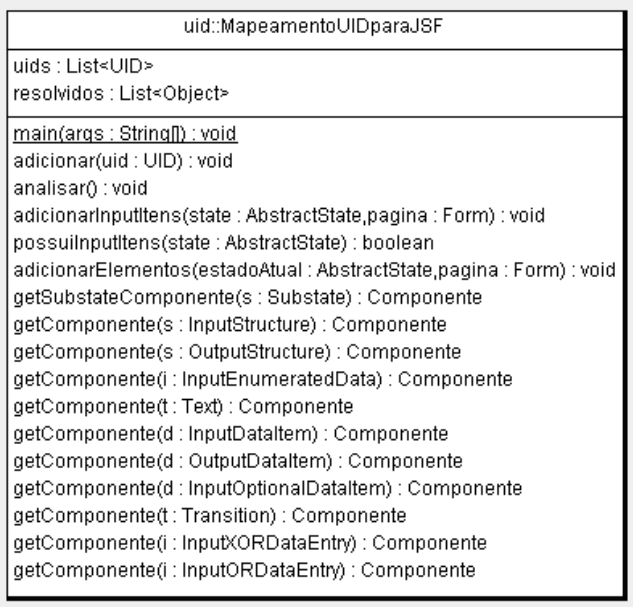


Figura 5.4 – Classe Mapeamento UID para JSF após a segunda iteração.

5.2.5. Terceira iteração

Para a terceira iteração foram selecionados todos os requisitos restantes, que consistia na implementação do método getJSF, para cada classe do tipo Componente. Também foram selecionados os requisitos de criação dos arquivos XHTML das páginas JSF e de elaboração de uma interface simples para facilitar a inserção dos arquivos dos UIDs.

Para criar o arquivo da página JSF basta invocar o getJSF na raiz da árvore de componentes que este irá propagar a chamada do método getJSF para os demais componentes, e assim sucessivamente até chegar em todos os nodos folha da árvore de componentes. Quando o nodo raiz, componente da classe “Página”, retornar sua string, esta será a página JSF que representa toda a árvore de componentes, bastando agora salva-lá em um arquivo .xhtml. A figura 5.5 apresenta o diagrama de sequência para o requisito de criação dos arquivos XHTML das páginas JSF.

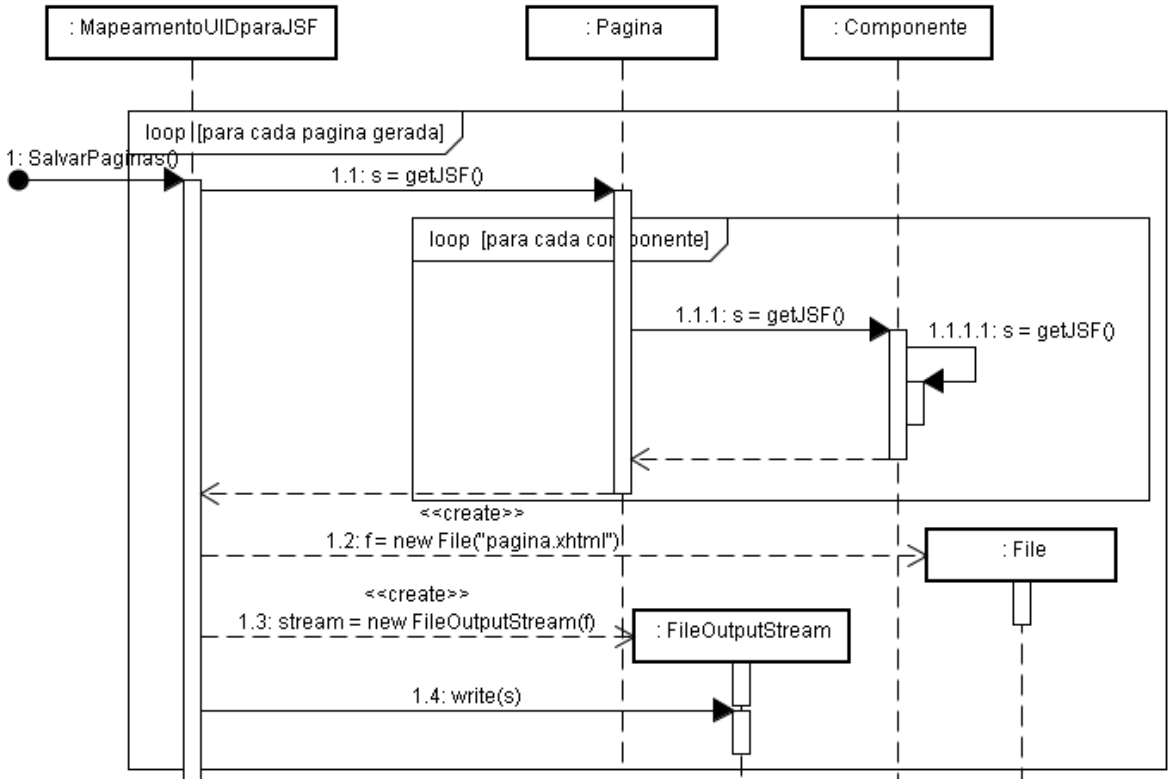


Figura 5.5 – Diagrama de sequência para criação dos arquivos das páginas

A Figura 5.6 mostra o diagrama de classes resultante ao término da terceira, e ultima, iteração.

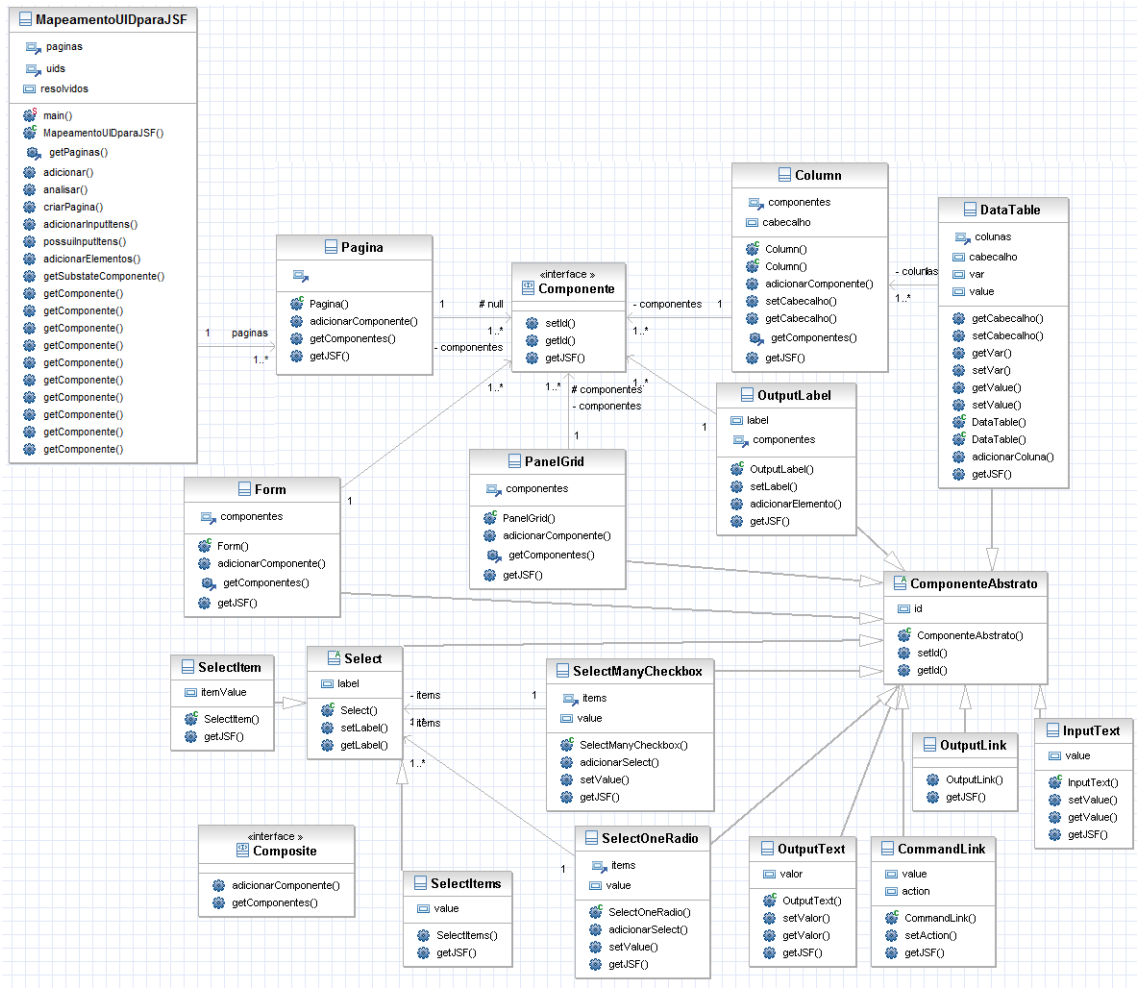


Figura 5.6 – Diagrama de classes de projeto resultante.

A interface gráfica desenvolvida para a ferramenta, mostrada na Figura 5.7, teve o intuito apenas de facilitar a geração das páginas JSF a partir dos arquivos dos UIDs sem se preocupar com design e usabilidade e foi desenvolvida para apenas que o usuário tenha que entrar com linhas de comando pelo console.

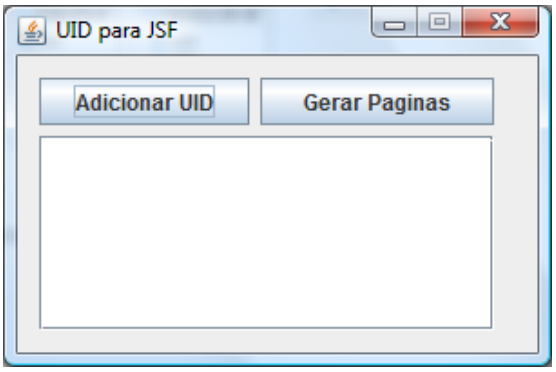


Figura 5.7 – Interface gráfica da ferramenta.

6. Exemplo de aplicação das regras

Com o intuito de validar as regras de mapeamento dos UIDs para páginas JSF, propostas neste trabalho, serão geradas páginas para alguns UIDs especificados em [Vilain, 2002]. Os UIDs foram especificados para o projeto de uma aplicação de venda de CDs musicais através de uma loja virtual. Eventualmente algum UID poderá ser alterado para colocar em teste alguma regra de mapeamento que não seria aplicada na atual especificação.

Ainda foi elaborado um UID a partir do site de compras online www.submarino.com.br. Com o uso da ferramenta de mapeamento desenvolvida neste trabalho, foram geradas as páginas JSF para este UID. Estas páginas foram, então, comparadas com as páginas originais do site.

6.1. Mapeamento dos UIDs da aplicação de venda de CDs

Neste seção são apresentados 3 UIDs (*Mostrar gravações de um CD*, *Comprar um CD a partir do nome de uma música* e *Comprar um CD a partir do gênero e consultar os cantores de um gênero*) e para cada um destes UIDs é mostrado o mapeamento para páginas JSF geradas a partir da ferramenta desenvolvida.

- **UID: Mostrar gravações de um CD**

O UID da Figura 6.1 foi especificado para representar o caso de uso Mostrar gravações de um CD:

1. Para um dado CD, o sistema mostra um conjunto com todas as suas gravações. Para cada música, é apresentado o nome da música, tempo de duração, cantor, compositor e letra.

2. Se o usuário desejar, uma gravação pode ser selecionada e um trecho seu pode ser escutado.

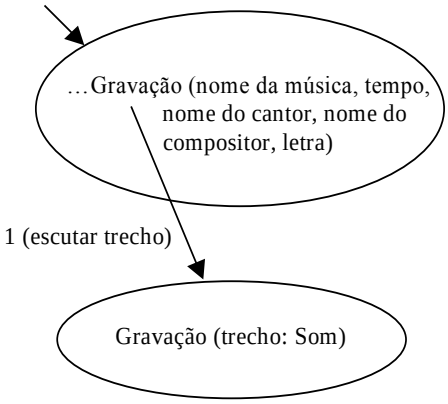
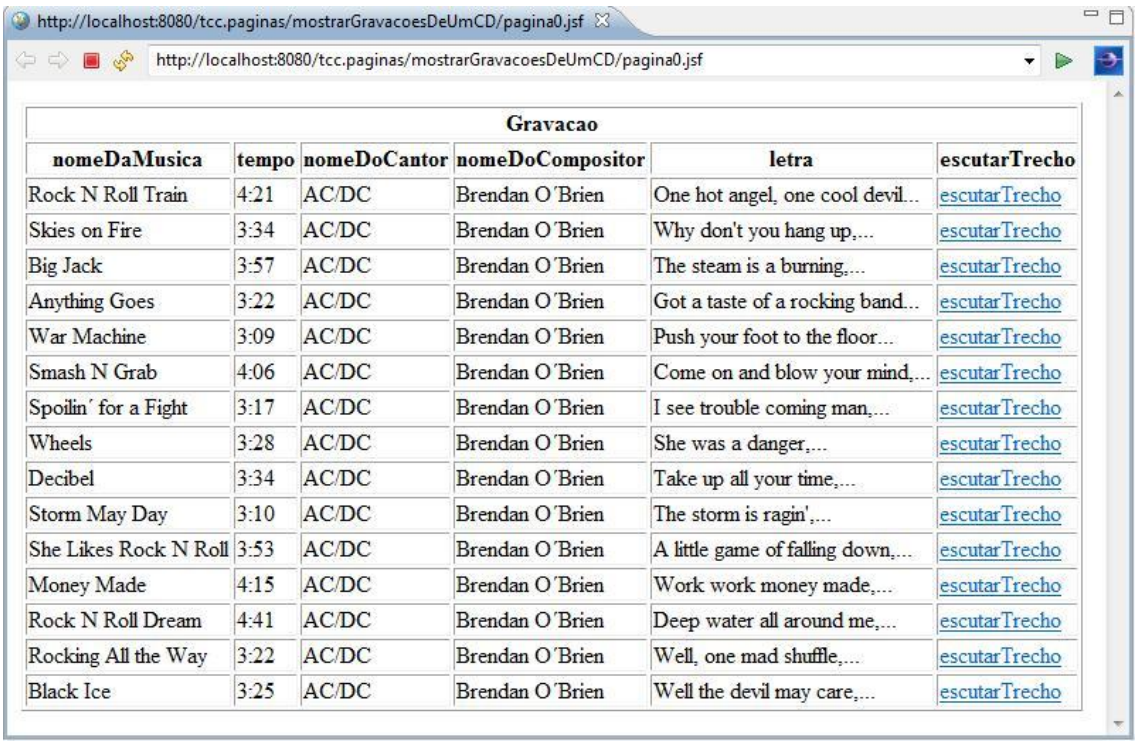


Figura 6.1 – Mostrar gravações de um CD

A Figura 6.2 mostra a página JSF gerada a partir do estado inicial de interação deste UID. O estado é representado por um componente *form*, não visível na imagem. O conjunto de estruturas deu origem à tabela e cada item de dado da estrutura "Gravação" é representado por uma coluna da tabela, assim como a transição "escutar trecho" que é representada pela última coluna. Já os elementos do conjunto, isto é, as estruturas do tipo gravação, são as linhas da tabela.



Gravacao					
nomeDaMusica	**tempo**	**nomeDoCantor**	**nomeDoCompositor**	**letra**	**escutarTrecho**
Rock N Roll Train	4:21	AC/DC	Brendan O´Brien	One hot angel, one cool devil...	escutarTrecho
Skies on Fire	3:34	AC/DC	Brendan O´Brien	Why don't you hang up,...	escutarTrecho
Big Jack	3:57	AC/DC	Brendan O´Brien	The steam is a burning,...	escutarTrecho
Anything Goes	3:22	AC/DC	Brendan O´Brien	Got a taste of a rocking band...	escutarTrecho
War Machine	3:09	AC/DC	Brendan O´Brien	Push your foot to the floor...	escutarTrecho
Smash N Grab	4:06	AC/DC	Brendan O´Brien	Come on and blow your mind,...	escutarTrecho
Spoilin´ for a Fight	3:17	AC/DC	Brendan O´Brien	I see trouble coming man,...	escutarTrecho
Wheels	3:28	AC/DC	Brendan O´Brien	She was a danger,...	escutarTrecho
Decibel	3:34	AC/DC	Brendan O´Brien	Take up all your time,...	escutarTrecho
Storm May Day	3:10	AC/DC	Brendan O´Brien	The storm is ragin',...	escutarTrecho
She Likes Rock N Roll	3:53	AC/DC	Brendan O´Brien	A little game of falling down,...	escutarTrecho
Money Made	4:15	AC/DC	Brendan O´Brien	Work work money made,...	escutarTrecho
Rock N Roll Dream	4:41	AC/DC	Brendan O´Brien	Deep water all around me,...	escutarTrecho
Rocking All the Way	3:22	AC/DC	Brendan O´Brien	Well, one mad shuffle,...	escutarTrecho
Black Ice	3:25	AC/DC	Brendan O´Brien	Well the devil may care,...	escutarTrecho

Figura 6.2 – Página referente ao estado inicial do UID Mostrar gravações de um CD.

A Figura 6.3 apresenta a página JSF referente ao segundo estado de interação deste UID. O estado de iteração foi mapeado para um componente *form*, não visível; a estrutura “Gravação” foi mapeada para um *panelGrid*, com um *outputLabel* onde seu valor é o nome da estrutura; e o item de dado “trecho” da estrutura foi mapeado para um *outputText* onde o valor é o nome do item de dado. Como a ferramenta desenvolvida não leva em consideração os domínios dos itens de dados, e simplesmente para cada um gera um *outputText*, o projetista terá que modificar o componente *outputText* referente ao item de dado “trecho” para, por exemplo, um player de música.

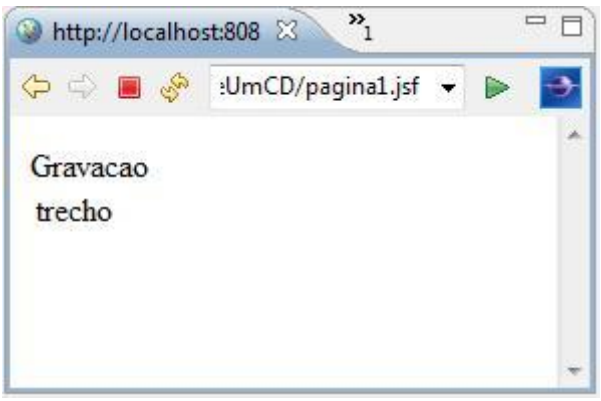


Figura 6.3 – Página referente ao segundo estado de iteração do UID Mostrar gravações de um CD.

- **UID: Comprar um CD a partir do nome de uma musica**

O UID da Figura 6.4 foi especificado para representar o caso de uso Comprar um CD a partir do nome de uma música:

1. O usuário entra com o nome de uma música.
2. O sistema retorna um conjunto de gravações que combinam com a entrada. Para cada gravação, é apresentado o nome da música, tempo de duração, nome do cantor e nome do compositor.
3. O usuário seleciona uma gravação e o sistema retorna um conjunto de CDs que apresentam a gravação selecionada. Para cada CD, é apresentado o nome do CD, nome do cantor, ano, preço, disponibilidade, capa, país, gênero e gravadora.
3. Se o usuário desejar, ele pode acessar as gravações em um CD (use case Mostrar gravações em um CD).
4. Caso o usuário deseje comprar um CD, ele o inclui na lista de compras para mais tarde efetuar a compra.

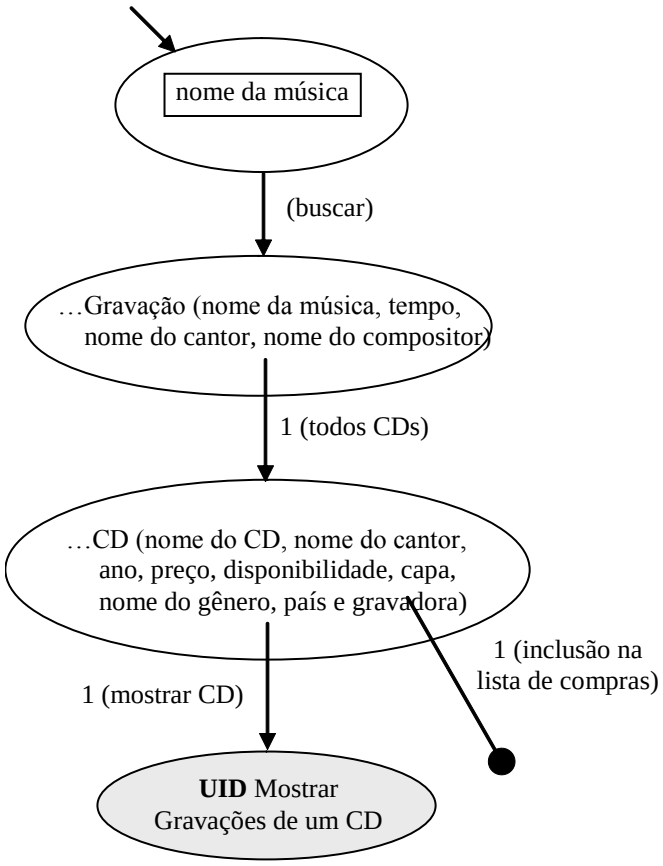


Figura 6.4 – Comprar um CD a partir do nome de uma musica

A Figura 6.5 apresenta a página JSF correspondente ao estado inicial e ao segundo estado de interação do UID, no contexto de que nenhuma busca foi realizada. Nesta página estão visíveis apenas os componentes pertencentes ao primeiro estado da interação: o item de dado "nome da musica" está representado pelos componentes *outputLabel*, com o nome do item de dado, e *inputText*, componente que permite que o usuário entre com o dado, e a transição "buscar" foi mapeada para o componente *commandLink*. Na figura 6.6 também é possível ver os componentes pertencentes ao segundo estado de interação, significando que uma busca foi submetida ao sistema. O conjunto de estruturas foi mapeado para a tabela, onde cada coluna da tabela representa

um item de dado da estrutura Gravação bem como a transição “todosCDs’, e as linhas da tabela representam as estruturas pertencentes ao conjunto .

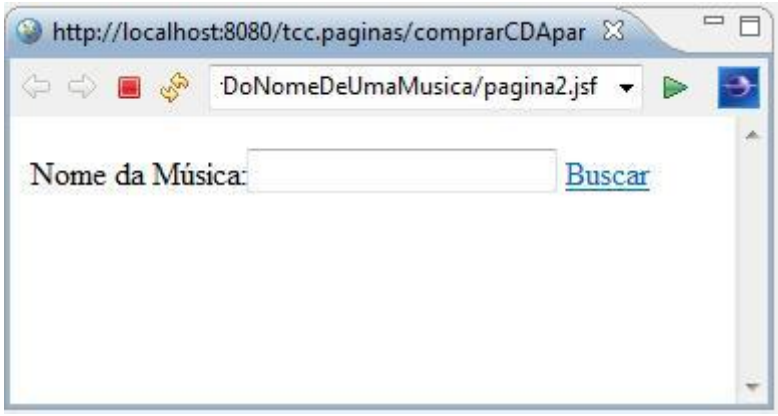


Figura 6.5 – Página referente ao estado inicial e segundo estado da iteração do UID Comprar um CD a partir do nome de uma musica, antes de uma busca ser submetida ao sistema.

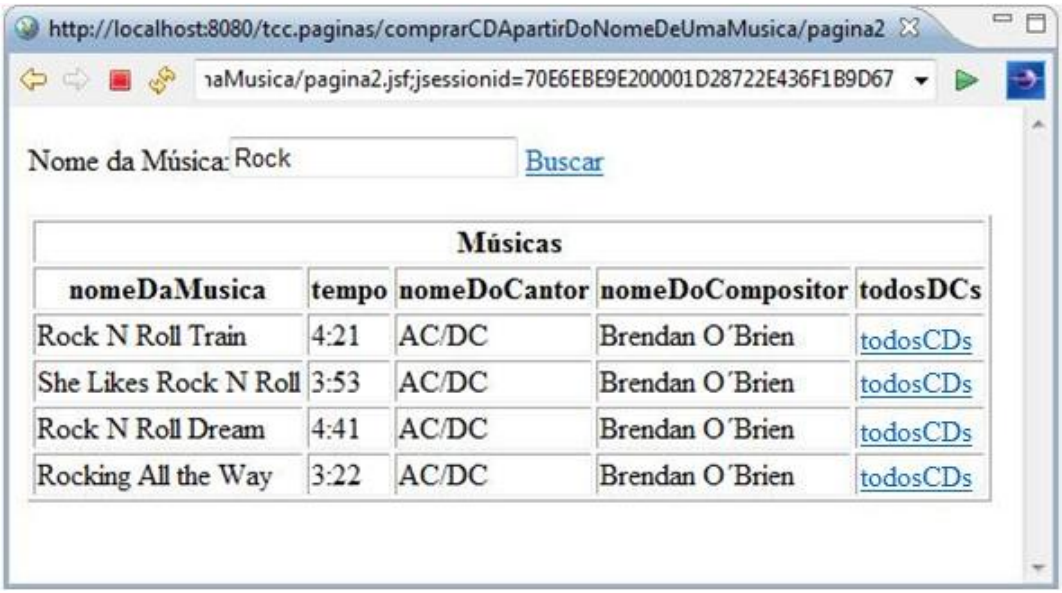


Figura 6.6 – Página referente ao estado inicial e segundo estado da iteração do UID Comprar um CD a partir do nome de uma musica, após a submissão de uma busca ao sistema.

A página JSF referente ao terceiro estado de interação é apresentada na Figura 6.7. O estado de iteração foi mapeado para o componente *form*, não visual. O conjunto

de estruturas foi mapeado para a tabela, cada coluna da tabela representa os itens de dados da estrutura CD e as duas últimas colunas representam as transições “mostrar CD” e “inclusão na lista de compras”. As estruturas pertencentes ao conjunto formam as linhas da tabela.

O item de dado “capa” foi mapeado para um *outputText*, contudo ele é uma imagem. Como a ferramenta considera todo item de dado como uma string e realiza seu mapeamento para o componente *outputText*, os campos da tabela referentes às capas dos CDs ficaram vazios.

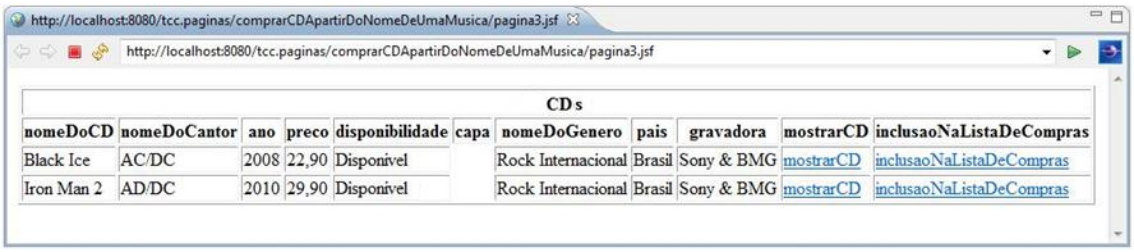


Figura 6.7 – Página referente ao terceiro estado da iteração do UID Comprar um CD a partir do nome de uma música.

- **UID: Comprar um CD a partir do gênero e consultar os cantores de um gênero**

O UID da Figura 6.8 foi especificado para representar o caso de uso Comprar um CD a partir do gênero e consultar os cantores de um gênero:

1. O sistema apresenta um conjunto dos gêneros de música (rock, clássica, new age, axé, pagode, samba, country, italiana, francesa, etc).
2. O usuário seleciona um gênero e, se desejar, o ano.
3. Se o usuário desejar verificar todos os cantores do gênero, o sistema retorna um conjunto de cantores que pertencem ao gênero escolhido.
4. O usuário seleciona o cantor desejado.
6. O sistema retorna um conjunto de CDs do cantor. Para cada CD, é apresentado o nome, cantor, ano, preço, disponibilidade, capa, país, gênero e gravadora.

6. Se o usuário desejar verificar todos os CDs do gênero ou sugestões, o sistema apresenta os CDs em ordem alfabética. Para cada CD, é apresentado o nome, cantor, ano, preço, disponibilidade, capa, gênero, país e gravadora.
7. Se o usuário desejar, ele pode acessar as gravações em um CD (use case Mostrar gravações em um CD).
8. Caso o usuário deseje comprar um ou mais CDs, ele o inclui na lista de compras para mais tarde efetuar a compra.

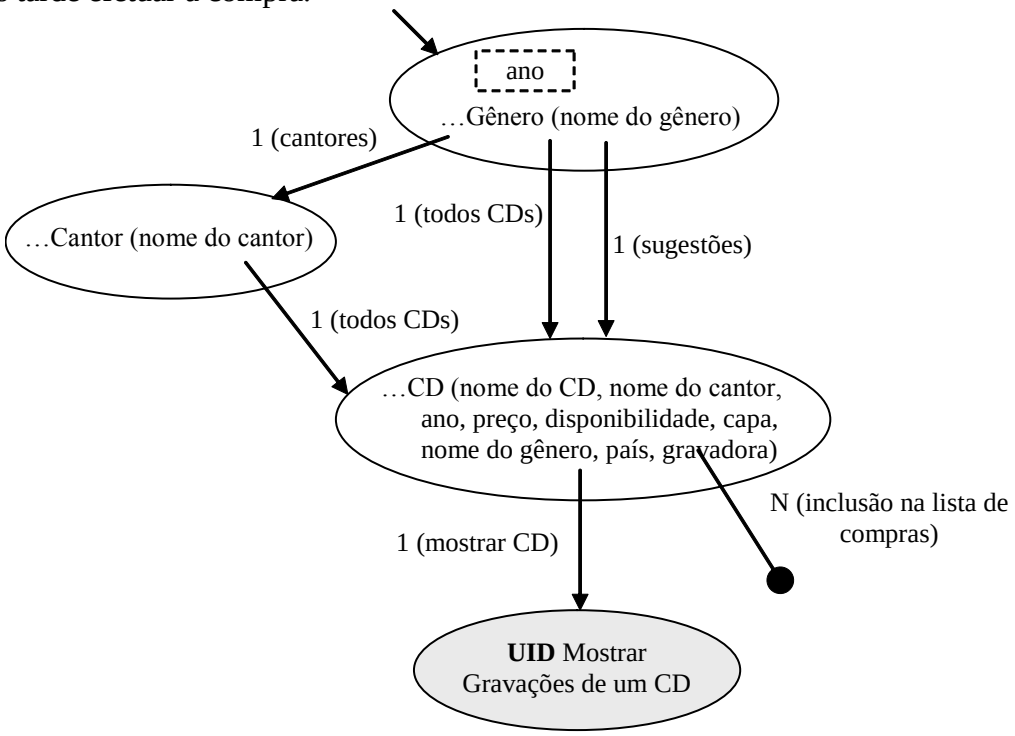


Figura 6.8 – Comprar um CD a partir do gênero e consultar os cantores de um gênero

O resultado do mapeamento para uma página JSF do estado inicial de interação deste UID é mostrado na Figura 6.9. A entrada do tipo item de dado é representada pelos componentes *inputText* e *outputLabel*, onde o nome do item de dado "ano" é o valor do *outputLabel*. Já o conjunto de estruturas do tipo Gênero foi mapeado para uma tabela, onde as colunas da tabela são os itens de dados da estrutura e também as transições que possuem ela como origem. Já as estruturas pertencentes ao conjunto formam as linhas da tabela.

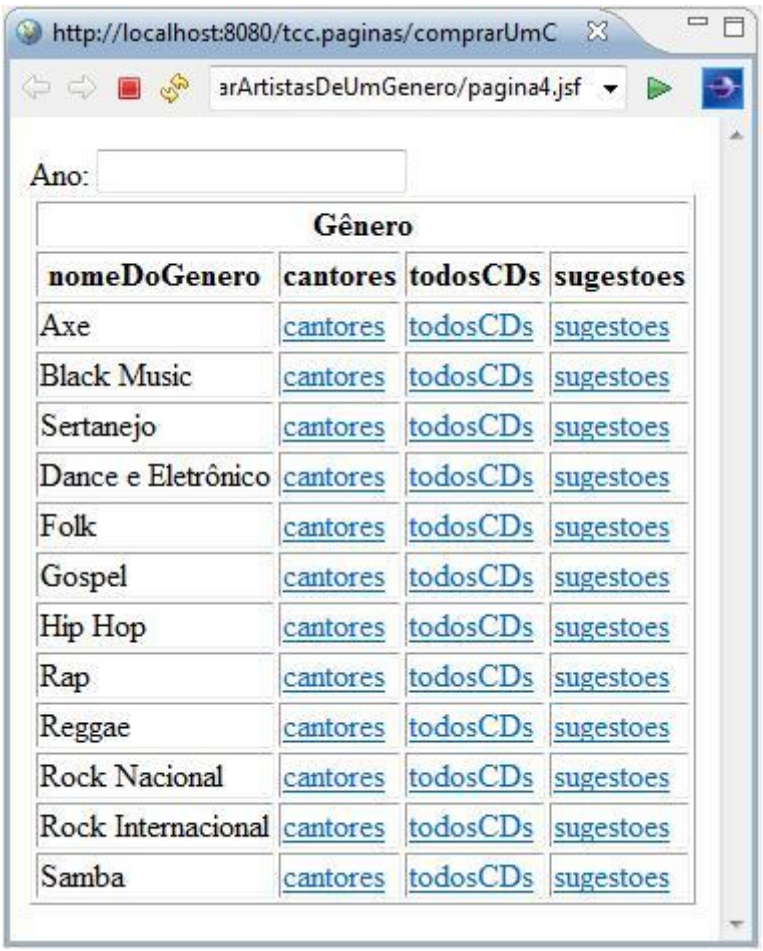


Figura 6.9 – Página referente ao estado inicial do UID Comprar um CD a partir do gênero e consultar os cantores de um gênero.

Na Figura 6.10 está a página JSF resultante do mapeamento do segundo estado de interação do UID. O estado de interação foi mapeado para o componente *form*, não visível. O conjunto de estruturas do tipo Cantor foi mapeado para uma tabela, onde as colunas da tabela representam o item de dado "nome do cantor" da estrutura e a transição "todos CDs", que possui como origem a estrutura. Cada linha da tabela é uma estrutura pertencente ao conjunto.

Figura 6.10 – Página referente ao segundo estado do UID Comprar um CD a partir do gênero e consultar os cantores de um gênero.

A Figura 6.11 mostra a página JSF resultante do mapeamento do terceiro estado de interação do UID Comprar um CD a partir do gênero e consultar cantores de um gênero. Seu mapeamento foi semelhante ao do terceiro estado do UID Comprar um CD a partir do nome da música (Figura 6.7), distinguindo apenas na última coluna da tabela. Pode ser verificado que nesta página, a última coluna é composta por caixas de seleção, componente *selectItem* do JSF, que representa a seleção do CD, correspondente à linha da tabela, para posterior inclusão na lista de compras, ou seja, o usuário seleciona todos

os CDs que deseja comprar e depois submete a compra “clicando” no *comandLink* que representa a transição “inclusão na lista de compras”.

No UID Comprar um CD a partir do nome da música (Figura 6.4), a transição “incluir na lista de compras” é para seleção de apenas 1 elemento, portanto o *commandLink* fica diretamente na tabela, significando que o CD representado pela mesma linha em que o *commandLink* está inserido é adicionado à lista de compras.

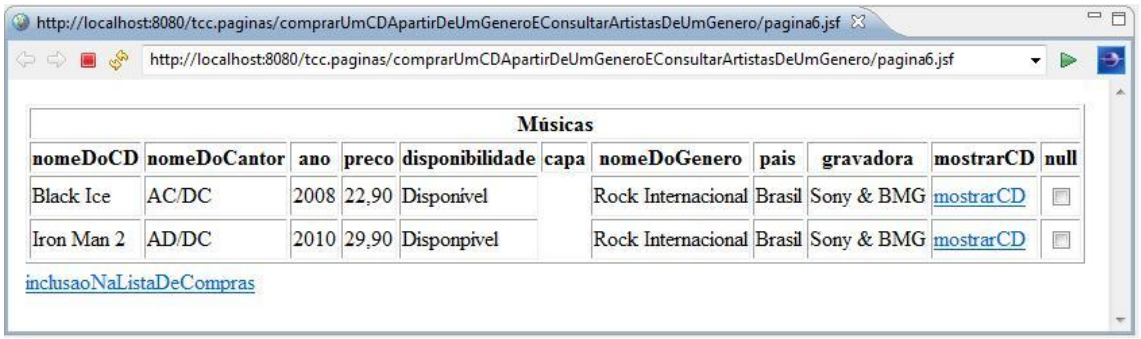


Figura 6.11 – Página referente ao terceiro estado do UID Comprar um CD a partir do gênero e consultar os cantores de um gênero.

6.2. Comparação com páginas de um site comercial

A validação das regras desenvolvidas neste trabalho também foi feita através de uma comparação das páginas geradas automaticamente pela ferramenta com páginas de sites comerciais. Como não foi encontrado nenhum site de vendas online que implementasse, por completo, algum dos três UIDs mostrados na seção anterior, foi elaborado, a partir do site www.submarino.com.br, um UID que representa o caso de uso *Comprar um CD a partir da busca avançada*, apresentado abaixo. Este UID é mostrado na Figura 6.12.

1. O usuário entra com o nome de um artista ou o nome do álbum ou o ano de lançamento ou, ainda, escolhe a seção (gênero) de um CD a partir de um conjunto de opções apresentado pelo sistema.

2. O sistema retorna um conjunto de CDs que combinam com as entradas. Para cada CD, é apresentado o nome do CD, o nome do artista, a avaliação dos clientes, uma breve descrição e o preço.

3. Se o usuário desejar, ele pode ver as informações detalhadas do CD, incluindo a descrição completa, o ano de lançamento, o país de origem, o label (gravadora), o número do ISSN e suas músicas.

4. Se o usuário desejar ele pode, ainda, escutar um trecho de cada música do cd.

5. Caso o usuário deseje comprar um CD, ele pode escolher uma de duas opções de compra, a compra normal ou a compra "com um click".

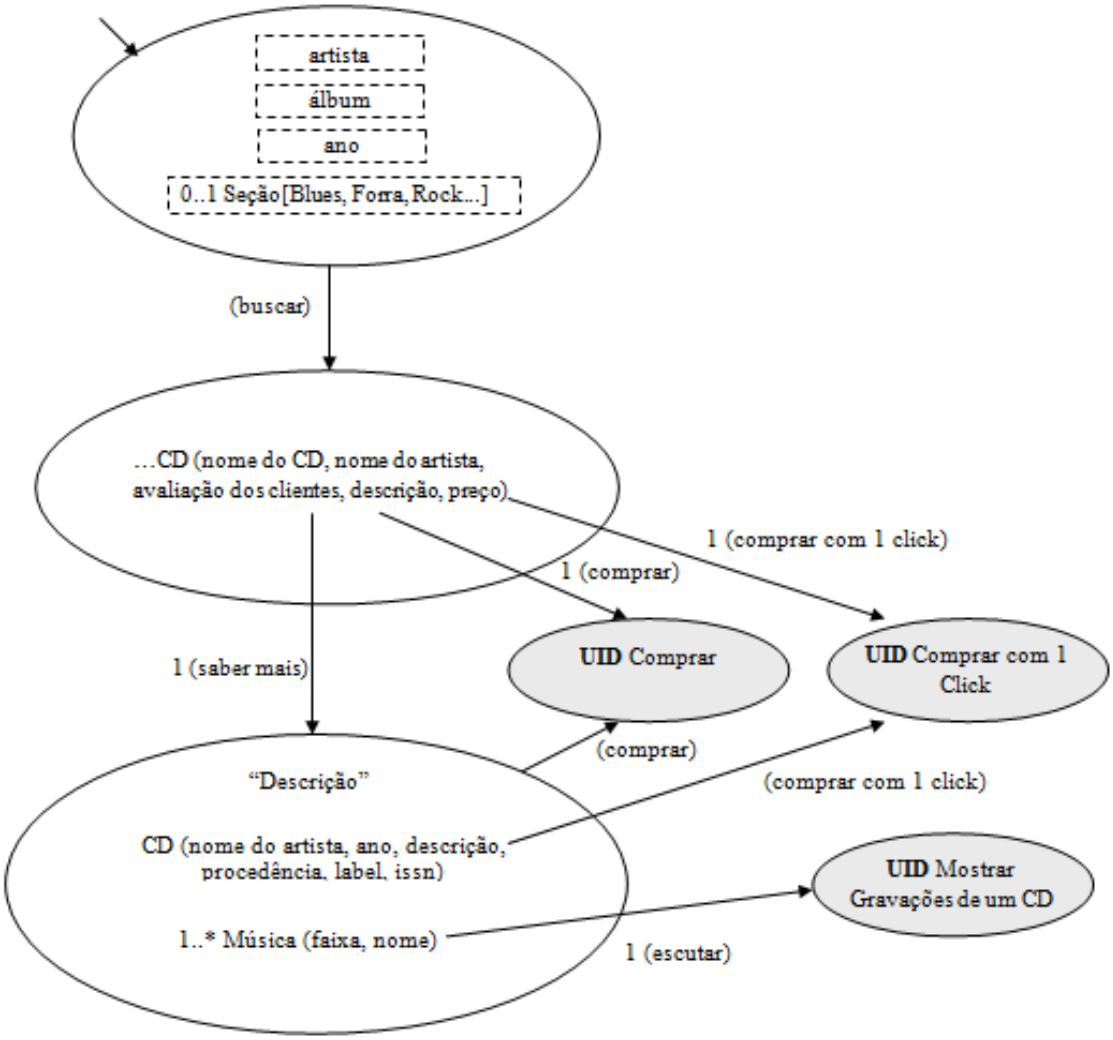


Figura 6.12 – Comprar um CD a partir da busca avançada

A página original do site correspondente ao estado inicial do UID está representada na Figura 6.13, enquanto a página gerada automática está representada na Figura 6.14. Tanto na página original quanto na página gerada automaticamente, existem 3 campos de entradas de texto para que o usuário possa entrar com o nome do artista, o nome do álbum e o ano de lançamento. A seleção da seção na página original é realizada através da seleção de um item de uma lista, enquanto na página gerada automaticamente a enumeração de seções foi mapeada para um conjunto de botões de rádio. E na página gerada automaticamente, o botão "Buscar" da página original é representando pelo link "Buscar".

Figura 6.13 – Página de busca avançada do site www.submarino.com.br.

A Figura 6.14 mostra os elementos do estado inicial, uma vez que nenhuma busca ainda havia sido realizada. Assim que uma busca for realizada os componentes do segundo estado de interação também ficarão visíveis, como mostrado na Figura 6.16.

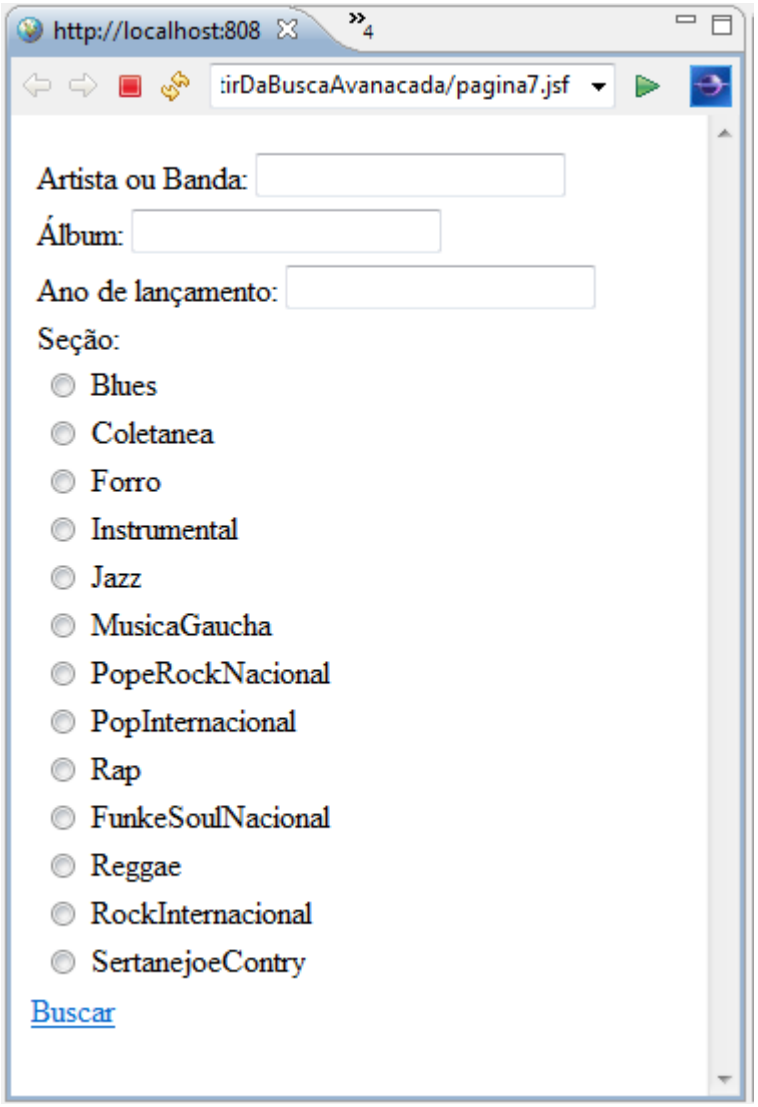


Figura 6.14 – Página gerada a partir do estado inicial do UID Comprar um CD a partir da busca avançada.

A Figura 6.15 é a parte da página de resultados de uma busca avançada realizada no site www.submarino.com.br, na qual a busca foi realizada informando o nome da banda “AC/DC” e ainda foi selecionada a seção “Rock Internacional”, e os campos para entrada do nome do álbum e o ano de lançamento foram deixados em branco. A mesma busca foi realizada na página gerada automaticamente e a página resultado pode ser visto na Figura 6.16.

Figura 6.15 – Página de resultados de uma busca avançada do site www.submarino.com.br.

A ferramenta realizou o mapeamento do estado inicial e do segundo estado de interação para a mesma página JSF mantendo assim a funcionalidade de busca avançada ainda na página de resultados. Diferentemente, o site original possui páginas distintas para busca avançada e para o resultado da busca, entretanto, ele ainda mantém um campo de busca simples na página de resultados.

A lista de CDs, resultado da busca, do segundo estado de interação foi mapeada para uma tabela, aonde cada resultado é uma linha e as informações são as colunas (Figura 6.16). No site original o design é um pouco diferente, mas segue basicamente o mesmo principio.

Tanto na página original quando na gerada automaticamente, cada resultado possui um link para “mais informações”, um para “comprar” e um terceiro para “comprar com 1 click”.

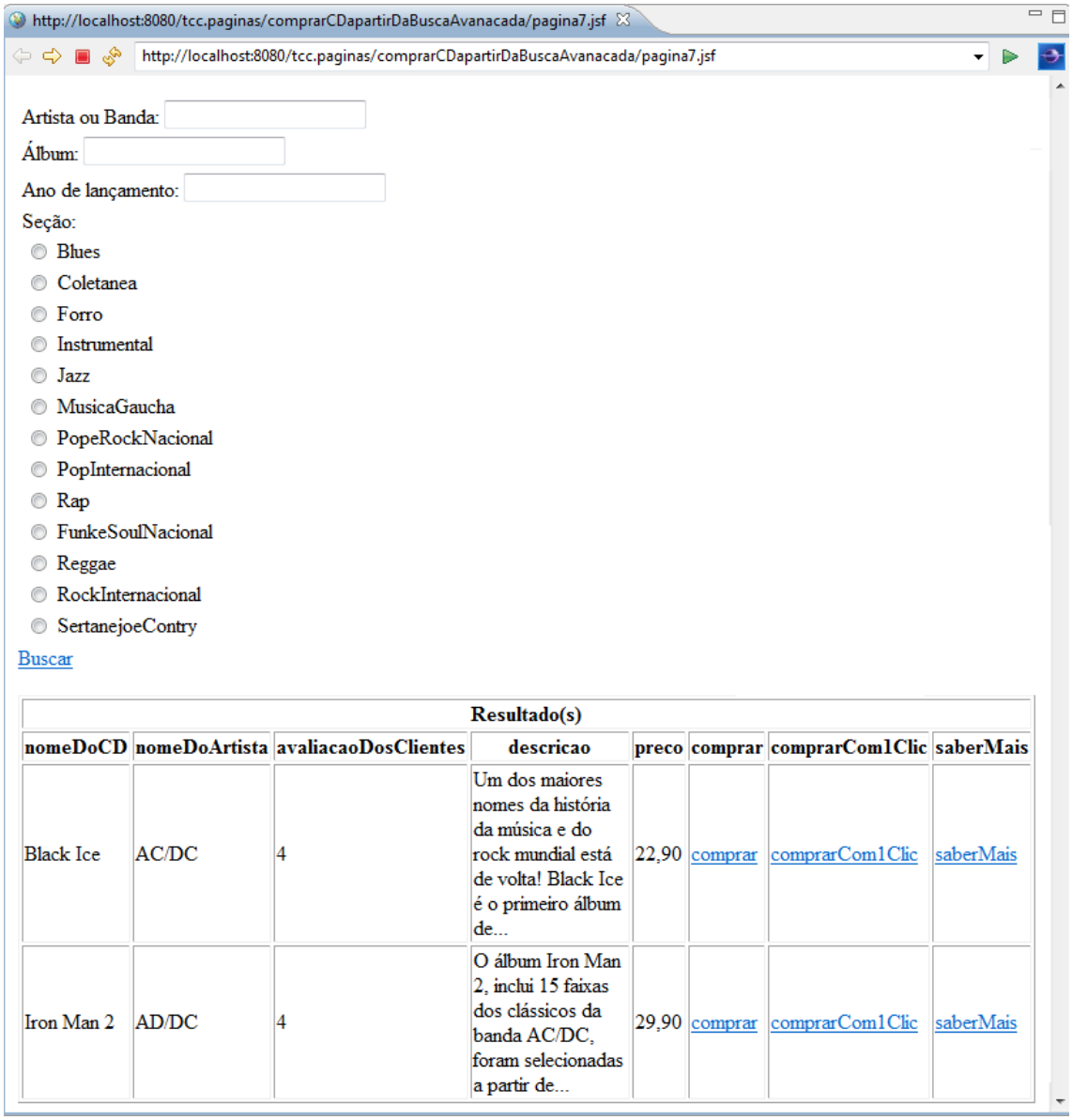


Figura 6.16 – Página com os elementos do segundo estado de interação do UID Comprar um CD a partir da busca avançada.

A Figura 6.17 é a página original referente ao terceiro estado de interação do UID. Nela não é possível ver os links para “comprar” e “comprar com ‘ click” pois estes

estão localizados na parte inicial do documento, e os detalhes do CD estão localizados na parte final. Entre estas duas regiões do documento são feitas diversas propagandas pelo site, devido a isto foi optado por omitir a parte inicial do documento, mostrando apenas o que é relevante.

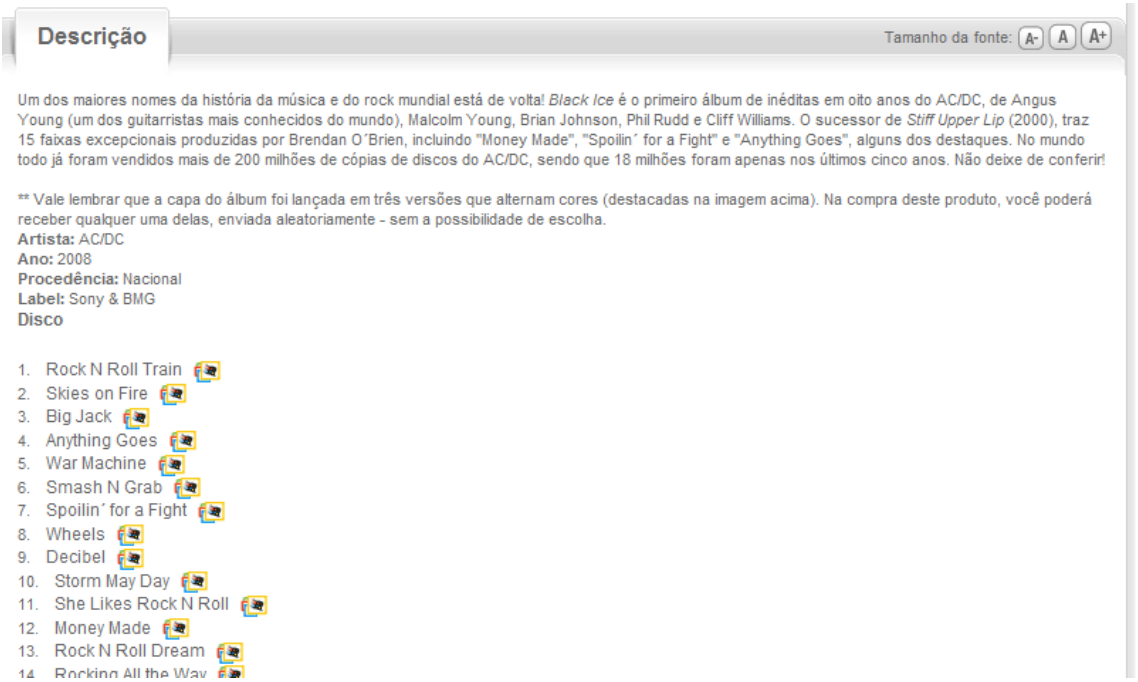


Figura 6.17 – Página de detalhes de um CD selecionado a partir do resultado de uma busca avançada do site www.submarino.com.br.

A página gerada automaticamente pela ferramenta que representa o terceiro estado de interação do UID é apresentada na Figura 6.18. A grande diferença entre esta página e a página original (Figura 6.17), é que na página original cada informação do CD possui um *label* informando qual sua natureza, enquanto na página gerada automaticamente as informações estão sem o *label*.

Como já mencionado anteriormente, os links “comprar” e “comprar com 1 click”, visíveis na página gerada automaticamente, estão ocultos na página original. Já

os links “escutar” da página gerada automaticamente são vistos como uma imagem, ícone do player, localizada ao lado de cada música no site original.

Figura 6.18 – Página gerada a partir do terceiro estado de interação do UID Comprar um CD a partir da busca avançada.

7. Considerações Finais

Este trabalho apresentou uma proposta de mapeamento dos UIDs (User Interaction Diagrams) para páginas JSF (Java Server Faces). Tomando como base as regras já definidas de mapeamento dos UIDs para os conceitos da ontologia de Widgets abstratos e as similaridades entre os elementos do UID e os componentes das páginas JSF, foram definidas as regras de mapeamento de cada elemento do UID para os componentes das páginas JSF.

Também foi desenvolvido um protótipo de uma ferramenta para automatizar este mapeamento. O protótipo desenvolvido obteve um bom desempenho na automatização das regras, gerando páginas JSF sintaticamente válidas, para as quais é necessário que o usuário implemente os Managed Beans de controle e rode a aplicação em desenvolvimento. Para que a geração automática ocorra corretamente é importante que o desenvolvedor elabore UIDs completos, coloque as opções nas transições, verifique a cardinalidade e se preocupe em atribuir o mesmo nome a itens de dados que possuem a mesma função. Todavia, como verificado nos testes realizados, alguns ajustes podem ser necessários nas páginas JSF geradas pela ferramenta.

A maior contribuição deste trabalho é o auxílio na etapa de projeto da interface com o usuário. A partir da especificação de UIDs, o projetista pode tomar como base as páginas geradas automaticamente, uma vez que elas possuem todos os componentes necessários para a troca de informação entre a aplicação e o usuário, bastando que elas sejam adaptadas para apresentarem as características desejadas.

Este trabalho também pode ser útil no desenvolvimento de protótipos para serem utilizados durante a etapa de levantamento de requisitos. Após a especificação dos UIDs, o analista poderia utilizar as páginas geradas automaticamente durante a interação com os usuários para confirmar os requisitos funcionais da aplicação.

As regras de mapeamento desenvolvidas neste trabalho podem ser utilizadas como base para o mapeamento dos UIDs para interfaces de outros frameworks. Como, por exemplo, geração de interfaces para WPF (Windows Presentation Foundation) a partir dos UIDs.

A Tabela 8.1 apresenta um comparativo entre a ferramenta desenvolvida neste trabalho com as apresentadas no capítulo de trabalhos relacionados. A comparação foi realizada considerando os seguintes critérios: se o processo de geração da interface é automático; se a interface é gerada a partir de um modelo abstrato; se a interface é gerada a partir de um modelo de interação; e se a interface gerada é a interface final da aplicação.

Tabela 8.1 – Tabela de comparativo das ferramentas:

	Becker	**G.I.**	**XIML**	**UIIMT**	**UID para JSF**
Geração Automática	Semi-automática	Semi-automática	Não	Não	Sim
Modelos Abstrato	Não	Não	Sim	Não	Não
Modelo de Interação	Não	Não	Não	Sim	Sim
Suporte a Padrões	Sim	Sim	Não	Sim	Não
Interface final da aplicação	Sim	Sim	Sim	Não	Parcial

Como trabalho futuro fica a sugestão de melhorias na ferramenta implementada, incluindo:

- a configuração, por parte do usuário, dos componentes para o qual cada elemento do UID é mapeado;
- a consideração do domínio do item de dado durante o mapeamento realizado;
- o mapeamento para componentes de bibliotecas de extensão como, por exemplo, RichFaces; e
- a geração dos managed beans das páginas JSF.

8. Referências Bibliográficas

BECKER, André Luis. **Ferramenta para Construção de Interfaces de Software a Partir de Diagrama de Classes.** FURB/Brasil, 2009.

LAVÔR, Renato B; LEITE, Jair C. **Ferramenta para Modelagem de Interação e Interface de Usuário.** CIbSE, 2011.

MARAFON, Diego Luiz. **Integração JavaServer Faces e Ajax, Estudo da integração das tecnologias JSF e Ajax**. UFSC/Brasil, 2006.

MOURA, Sabrina S. **Desenvolvimento de Interfaces Governadas por Ontologias para Aplicações na Web Semântica**. PUC-Rio, 2004. Dissertação de Mestrado.

PETERMANN, Rafael Jordan; BELLIN, Fernando; KROTH, Eduardo. **Uma ferramenta para geração de interfaces de sistemas de informação em ambiente WEB**. XV Simpósio Brasileiro de Engenharia de Software, 2001.

BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAD, P.; STAL, M. **Pattern-oriented software architecture: a system of patterns**. 1996

REMÁCULO, Luanda Philippi. **Personalização de Diagramas de Interação do Usuário e Mapeamento para a Ontologia de Widgets Abstratos**. UFSC/Brasil, 2005.

SCHWABE, D; ROSSI, G. **The Object-Oriented Hypermedia Design Model (OOHDM).** PUC-Rio, 2003.

Site **Edson Gonçalves JSF 2.0, O ciclo de vida do JSF.** Disponível em http://www.edsongoncalves.com.br/category/javaserver-faces-2-0/#LifeCicle. Acessado em 13/05/2011.

Site **JSF Tollbox, JavaServer Faces for Dreamwaver. JSF Tag Reference**. Disponível em http://www.jsftoolbox.com/documentation/help/12-TagReference/html/index.jsf. Acessado em 13/05/2011.

Site oficial **JavaServer Faces Technology**. Disponível em http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html. Acessado em 13/05/2011.

Site **PrimeFaces**. Disponível em www.primefaces.org. Acessado em 12/07/2011.

Site **RichFaces.** Disponível em http://www.jboss.org/richfaces. Acessado em 12/07/2011.

VILAIN, Patrícia. **Modelagem da Interação com o Usuário em Aplicações Hipermídia**. PUC-Rio, 2002. Tese de Doutorado.

VILAIN, Patrícia. **Relatório Final, Funpesquisa 2003. Implementação de um Framework para Suporte à Representação de Requisitos Funcionais no Processo de Software**. UFSC/Brasil, 2003.

Apêndice A - Mapeamento dos Widgets Abstratos para componentes JSF

A seguir são definidos, para cada widget abstrato da ontologia apresentada na seção 2.2, os possíveis componentes concretos do JSF que podem representá-lo em uma interface concreta:

- **Simple Activator:** um *simple activator* deve ser mapeado para um *outputLink*, *commandLink* ou *commandButton*, pois qualquer um destes três componentes é capaz de reagir a eventos.

- **Element Exhibitor:** um *element exhibitor* deve ser mapeado para um elemento capaz de exibir algum tipo de conteúdo no JSF. Os componentes *outputText*, *outputLabel* e *outputFormat* têm a funcionalidade de exibir textos. Já os componentes *message* e *messages* tem como função exibir mensagens retornadas pelo sistema e o *graphicImage* serve para exibir imagens.

- **Arbitrary Value:** um *arbitraty value* pode ser representado por três componentes no JSF: *inputText*, *inputTextArea* ou *inputSecret*. Estes três componentes permitem a entrada de dados através do teclado.

- **Single Choice:** no JSF os componentes que permitem a seleção de um único valor a partir de um conjunto pré definido de valores são: *selectOneRadio*, *selectOneMenu* e *selectOneListbox*. Portanto, um *single choice* deve ser representado, em JSF, por algum destes três componentes.

- **Multiple Choice:** no JSF os componentes *selectManyCheckbox*, *selectManyMenu* e *selectManyListbox* são os responsáveis por permitir a seleção de um subconjunto a partir de um conjunto de dados pré definido. Então, um *multiple choice* deve ser representado por algum destes três componentes.

- **ContinuousGroup e DiscreetGroup:** não existem componentes na biblioteca básica HTML do JSF que representem estes dois widgets abstratos.

- **Composite Interface Element:** os componentes *form*, *panelGrid*, *panelGroup* e *dataTable* são os responsáveis por agrupar outros componentes.

A Tabela A.1 apresenta, resumidamente, as opções de mapeamento de cada elemento da ontologia de widgets abstratos para componentes JSF.

Tabela A.1 - Tabela de mapeamento dos Widgets Abstratos para componentes JSF:

Widgets Abstratos		**Componentes JSF**
Simple Activator		OutputLink commandButton commandLink
Element Exhibitor		OutputText outputLabel outputFormat message messages graphicImage
Value Capturer	ArbitraryValue	InputText inputTextArea inputSecret
	Single Choice	SelectOneRadio selectOneMenu selectOneListbox

	Multiple Choice	SelectManyCheckbox selectManyMenu selectManyListbox
Composite interface element		Form panelGrid panelGroup dataTable

Como observado acima, para cada widget abstrato existe mais de um componente JSF possível de representá-lo. Tomando como base só as informações disponíveis no widget abstrato, a escolha de sua representação em uma página JSF fica comprometida pois, neste caso, algumas informações foram perdidas durante o mapeamento dos elementos do UID para os widgets abstratos. Tendo em vista que a partir de um elemento do UID é possível extrair mais informações do que a partir de um widget abstrato, foi decidido fazer o mapeamento direto dos elementos do UID para os componentes do JSF.

Apêndice B – Código Fonte

Frame.java

```
package visao;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import uid.MapeamentoUIDparaJSF;
import uid.UIDXmlReader;
import uid.elements.UID;
import uid.jsf.Pagina;

/**
 *
 * @author Filipe
 */
public class Frame extends javax.swing.JFrame {

    private List<UID> listaDeUIDs;

    /** Creates new form Frame */
    public Frame() {
        listaDeUIDs = new ArrayList<UID>();
        initComponents();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
    private void initComponents() {

        jfAbrir = new javax.swing.JFileChooser();
        jPanel1 = new javax.swing.JPanel();
        jbAdicionarUID = new javax.swing.JButton();
        jScrollPane1 = new javax.swing.JScrollPane();
        jTextListaDeUIDs = new javax.swing.JTextArea();
        jbGerar = new javax.swing.JButton();
```

```
        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("UID para JSF");
        setLocationByPlatform(true);
        setName("Mapeamento UID para JSF"); // NOI18N
        setResizable(false);

        jbAdicionarUID.setText("Adicionar UID");
        jbAdicionarUID.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jbAdicionarUIDActionPerformed(evt);
            }
        });

        jTextListaDeUIDs.setColumns(20);
        jTextListaDeUIDs.setRows(5);
        jTextListaDeUIDs.setEnabled(false);
        jScrollPane1.setViewportView(jTextListaDeUIDs);

        jbGerar.setText("Gerar Paginas");
        jbGerar.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jbGerarActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(jScrollPane1, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 251, Short.MAX_VALUE)
                    .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
jPanel1Layout.createSequentialGroup()
                        .addComponent(jbAdicionarUID, javax.swing.GroupLayout.PREFERRED_SIZE,
116, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jbGerar, javax.swing.GroupLayout.PREFERRED_SIZE, 129,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jbAdicionarUID)
                    .addComponent(jbGerar))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
```

```
            .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 107,
Short.MAX_VALUE))
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );

        getAccessibleContext().setAccessibleName("Mapeamento UID para páginas JSF");
        getAccessibleContext().setAccessibleDescription("UID para JSF...");

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void jbAdicionarUIDActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jbAdicionarUIDActionPerformed
        int status = jfAbrir.showOpenDialog(null);
        if (status == jfAbrir.APPROVE_OPTION) {
            File arquivo = jfAbrir.getSelectedFile();
            try {
                UID uid = new UIDXmlReader().read(new FileInputStream(arquivo));

                uid.setName(arquivo.getName());

                this.listaDeUIDs.add(uid);

                String uids = "";

                for (UID u : listaDeUIDs) {
                    uids += u.getName();
                    uids += "\n";
                }


                this.jTextListaDeUIDs.setText(uids);


            } catch (FileNotFoundException ex) {
                Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);
            }
```

```
        }

    }//GEN-LAST:event_jbAdicionarUIDActionPerformed

    private void jbGerarActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jbGerarActionPerformed

        MapeamentoUIDparaJSF map = new MapeamentoUIDparaJSF();

        for (UID uid : listaDeUIDs) {
            map.adicionar(uid);
        }
        String saida = "Páginas geradas: \n";
        map.analisar();
        int i = 0;
        for (Pagina p : map.getPaginas()) {

            File f = new File(
                    "pagina"
                    + i + ".xhtml");

            saida += "   " + f.getAbsolutePath() + "\n";

            FileOutputStream stream = null;
            try {
                stream = new FileOutputStream(f);

                stream.write(p.getJSF().replaceAll("\\#\\{BEAN.", "#{pagina" + i + "MB.").getBytes());
            } catch (FileNotFoundException ex) {
                Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);
            } catch (IOException ex) {
                Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);
            } finally {
                if (stream != null) {
                    try {
                        stream.close();
                    } catch (IOException ex) {
                        Logger.getLogger(Frame.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
            }
            i++;
        }

         this.jTextListaDeUIDs.setText(saida);

    }//GEN-LAST:event_jbGerarActionPerformed

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
```

```
            public void run() {
                new Frame().setVisible(true);
            }
        });
    }
    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JPanel jPanel1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextArea jTextListaDeUIDs;
    private javax.swing.JButton jbAdicionarUID;
    private javax.swing.JButton jbGerar;
    private javax.swing.JFileChooser jfAbrir;
    // End of variables declaration//GEN-END:variables
}
```

MapeamentoUIDparaJSF.java

```
package uid;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import uid.elements.AbstractState;
import uid.elements.DataItem;
import uid.elements.InputDataItem;
import uid.elements.InputEnumeratedData;
import uid.elements.InputORDataEntry;
import uid.elements.InputOptionalDataItem;
import uid.elements.InputStructure;
import uid.elements.InputXORDataEntry;
import uid.elements.OutputDataItem;
import uid.elements.OutputStructure;
import uid.elements.Structure;
import uid.elements.Substate;
import uid.elements.Text;
import uid.elements.Transition;
import uid.elements.TransitionFromSet;
import uid.elements.UID;
import uid.jsf.Column;
import uid.jsf.CommandLink;
import uid.jsf.Componente;
import uid.jsf.DataTable;
import uid.jsf.Form;
import uid.jsf.InputText;
import uid.jsf.OutputLabel;
import uid.jsf.OutputText;
import uid.jsf.Pagina;
import uid.jsf.PanelGrid;
import uid.jsf.SelectItem;
import uid.jsf.SelectItens;
import uid.jsf.SelectManyCheckbox;
import uid.jsf.SelectOneRadio;

public class MapeamentoUIDparaJSF {

	List<Pagina> paginas;
	List<UID> uids;

	List<Object> resolvidos;

	/**
	 * @author Filipe Bianchi Damiani
	 * @param args
```

```
 * @throws IOException
 */
public static void main(String[] args) throws IOException {

	MapeamentoUIDparaJSF map = new MapeamentoUIDparaJSF();

	UIDXmlReader xmlReader = new UIDXmlReader();
	FileInputStream in;

	for (int i = 0; i < args.length; i++) {
		in = new FileInputStream(args[i]);
		map.adicionar(xmlReader.read(in));
	}

	map.analisar();

	int i = 0;
	for (Pagina p : map.getPaginas()) {

		File f = new File("pagina" + i + ".xhtml");

		FileOutputStream stream = new FileOutputStream(f);

		stream.write(p.getJSF()
				.replaceAll("\\#\\{BEAN.", "#{pagina" + i + "MB.")
				.getBytes());

		i++;
	}
}

public MapeamentoUIDparaJSF() {
	paginas = new ArrayList<Pagina>();
	uids = new ArrayList<UID>();
	resolvidos = new ArrayList<Object>();
}

public List<Pagina> getPaginas() {
	return paginas;
}

public void adicionar(UID uid) {
	this.uids.add(uid);
}

public void analisar() {

	for (UID uid : uids) {
		resolvidos = new ArrayList<Object>();

		AbstractState estadoAtual = uid.getInitialState();

		Pagina p = new Pagina();
```

```
                this.criarPagina(estadoAtual, p);

                this.criarPagina(estadoAtual, p);

                paginas.add(p);

                List<AbstractState> estados = uid.getSetStates();
                for (AbstractState s : estados) {
                        if (!resolvidos.contains(s)) {
                                p = new Pagina();
                                this.criarPagina(s, p);
                                paginas.add(p);
                        }
                }
        }
}

private void criarPagina(AbstractState estadoAtual, Pagina p) {

        if (!resolvidos.contains(estadoAtual)) {
                for (Substate s : estadoAtual.getAllSubStates()) {
                        p.adicionarComponente(this.getSubstateComponente(s));
                }

                Form pagina = new Form(estadoAtual.getId());

                p.adicionarComponente(pagina);

                this.adicionarElementos(estadoAtual, pagina);

                boolean r = false;

                for (Transition t : estadoAtual.getAllOutgoingTransitions()) {

                        if (t.getTargetType() == Transition.STATE
                                        && t.getCondition() == null
                                        && !resolvidos.contains(t.getTargetState())
                                        && this.possuiInputItens(t.getTargetState())) {
                                this.adicionarInputItens(t.getTargetState(), pagina);
                                r = true;
                        }
                }

                if (r) {
                        resolvidos.add(estadoAtual);
                        return;
                }

                for (Transition t : estadoAtual.getAllOutgoingTransitions()) {

                        if (t.getTargetType() == Transition.STATE
                                        && t.getCondition() == null
                                        && !resolvidos.contains(t.getTargetState())) {
                                for (DataItem d : estadoAtual.getAllDataItens()) {
```

```
                                                    if (d instanceof InputDataItem
                                                            || d instanceof
InputOptionalDataItem) {
                                                        for (Structure s : t.getTargetState()
                                                                .getAllStructures()) {
                                                            if (s instanceof OutputStructure)
{
                                                                for (DataItem e :
s.getDataItems()) {

        d.getName().equals(e.getName());

        this.criarPagina(t.getTargetState(), p);
                                                                        r = true;
                                                                }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                    resolvidos.add(estadoAtual);
                                }

                                private void adicionarInputItens(AbstractState state, Form pagina) {
                                    if (!resolvidos.contains(state)) {
                                        this.adicionarElementos(state, pagina);
                                        for (Transition t : state.getAllOutgoingTransitions()) {
                                            if (t.getCondition() == null
                                                    && !resolvidos.contains(t.getTargetState())
                                                    && this.possuiInputItens(t.getTargetState())) {

                                                this.adicionarInputItens(t.getTargetState(), pagina);
                                            }
                                        }
                                        resolvidos.add(state);
                                    }
                                }

                                private boolean possuiInputItens(AbstractState state) {
                                    for (DataItem d : state.getAllDataItens()) {
                                        if (d instanceof InputDataItem) {
                                            return true;
                                        }
                                        if (d instanceof InputOptionalDataItem) {
                                            return true;
                                        }
                                    }
                                    for (Structure s : state.getAllStructures()) {
                                        if (s instanceof InputStructure) {
                                            return true;
                                        }
                                    }
```

```
		if (state.getAllInputEnumeratedData().size() > 0)
			return true;
		if (state.getAllInputORDataEntries().size() > 0)
			return true;
		if (state.getAllInputXORDataEntries().size() > 0)
			return true;
		return false;
}

private void adicionarElementos(AbstractState estadoAtual, Form pagina) {

		resolvidos.add(estadoAtual);

		for (Text t : estadoAtual.getAllTexts()) {

			pagina.adicionarComponente(this.getComponente(t));

		}

		for (DataItem d : estadoAtual.getAllDataItens()) {

			if (d instanceof InputDataItem) {

				pagina.adicionarComponente(this
						.getComponente((InputDataItem) d));

			}

			if (d instanceof InputOptionalDataItem) {

				pagina.adicionarComponente(this
						.getComponente((InputOptionalDataItem) d));
			}

			if (d instanceof OutputDataItem) {

				pagina.adicionarComponente(this
						.getComponente((OutputDataItem) d));
			}
		}

		for (Structure s : estadoAtual.getAllStructures()) {
			if (s instanceof InputStructure) {
				pagina.adicionarComponente(this
						.getComponente((InputStructure) s));

			}

			if (s instanceof OutputStructure) {
				pagina.adicionarComponente(this
						.getComponente((OutputStructure) s));
			}
		}
```

```
		for (InputEnumeratedData i : estadoAtual.getAllInputEnumeratedData()) {
			pagina.adicionarComponente(this.getComponente(i));

		}

		for (InputORDataEntry i : estadoAtual.getAllInputORDataEntries()) {

			pagina.adicionarComponente(this.getComponente(i));

		}

		for (InputXORDataEntry i : estadoAtual.getAllInputXORDataEntries()) {

			pagina.adicionarComponente(this.getComponente(i));

		}

		for (Transition t : estadoAtual.getAllOutgoingTransitions()) {

			if (t.getTargetType() == Transition.STATE
					&& t.getCondition() == null && t.getOption() != null) {
				if (!t.getOption().replaceAll(" ", "").equals("")) {
					pagina.adicionarComponente(this.getComponente(t));
				}
			}
		}

	}

	private Componente getSubstateComponente(Substate s) {
		if (resolvidos.contains(s)) {
			Form componente = new Form(s.getId());

			for (Text t : s.getAllTexts()) {

				componente.adicionarComponente(this.getComponente(t));

			}

			for (DataItem d : s.getAllDataItens()) {

				if (d instanceof InputDataItem) {

					componente.adicionarComponente(this
							.getComponente((InputDataItem) d));

				}

				if (d instanceof InputOptionalDataItem) {

					componente.adicionarComponente(this

	.getComponente((InputOptionalDataItem) d));
				}
```

```
                        if (d instanceof OutputDataItem) {

                                componente.adicionarComponente(this
                                                .getComponente((OutputDataItem) d));
                        }
                }

                for (Structure st : s.getAllStructures()) {
                        if (st instanceof InputStructure) {
                                componente.adicionarComponente(this
                                                .getComponente((InputStructure) st));

                        }

                        if (st instanceof OutputStructure) {
                                componente.adicionarComponente(this
                                                .getComponente((OutputStructure) st));
                        }
                }

                for (InputEnumeratedData i : s.getAllInputEnumeratedData()) {
                        componente.adicionarComponente(this.getComponente(i));

                }

                for (InputORDataEntry i : s.getAllInputORDataEntries()) {

                        componente.adicionarComponente(this.getComponente(i));

                }

                for (InputXORDataEntry i : s.getAllInputXORDataEntries()) {

                        componente.adicionarComponente(this.getComponente(i));

                }

                resolvidos.add(s);
                return componente;
        }
        return null;
}

private Componente getComponente(InputStructure s) {

        Componente componente;

        if (s.getType() == 1 | s.getType() == 3 || s.getCardinalityMax() > 5) {

                if (s.getDataItems().size() + s.getStructures().size() > 0) {
                        OutputLabel label = new OutputLabel(s.getName() + "label");

                        PanelGrid grid = new PanelGrid(s.getName());
```

```
                                label.adicionarElemento(grid);

                                for (DataItem d : s.getDataItems()) {

                                        if (d instanceof InputDataItem) {

                                                grid.adicionarComponente(this
                                                                .getComponente((InputDataItem)
d));

                                        }

                                        if (d instanceof InputOptionalDataItem) {

                                                grid.adicionarComponente(this

        .getComponente((InputOptionalDataItem) d));
                                        }

                                        if (d instanceof OutputDataItem) {

                                                grid.adicionarComponente(this

        .getComponente((OutputDataItem) d));
                                        }

                                }

                                for (Structure st : s.getStructures()) {
                                        if (st instanceof InputStructure) {
                                                grid.adicionarComponente(this
                                                                .getComponente((InputStructure)
st));

                                        }

                                        if (st instanceof OutputStructure) {
                                                grid.adicionarComponente(this

        .getComponente((OutputStructure) st));
                                        }
                                }

                                resolvidos.add(s);

                                componente = label;

                        } else {
                                OutputLabel label = new OutputLabel(s.getName() + "label");

                                label.adicionarElemento(new InputText(s.getName(), "value"
                                                + s.getName()));
```

```
                        this.resolvidos.add(s);

                        componente = label;
                    }

                } else {

                    if (s.getDataItems().size() + s.getStructures().size() > 0) {

                        DataTable dt = new DataTable(s.getName(), s.getName());

                        Column c;
                        for (DataItem d : s.getDataItems()) {
                            c = new Column(d.getName());
                            c.adicionarComponente(new InputText(d.getName(),
"value"
                                    + d.getName()));
                            dt.adicionarColuna(c);
                        }
                        for (Structure st : s.getAllStructuresAndInternalStructures()) {

                            if (st.getDataItems().size() > 0) {
                                for (DataItem d : st.getDataItems()) {
                                    c = new Column(d.getName());
                                    c.adicionarComponente(new
InputText(d.getName(),
                                            "value" + d.getName()));
                                    dt.adicionarColuna(c);
                                }
                            } else {
                                c = new Column(st.getName());
                                dt.adicionarColuna(c);
                            }

                        }

                        this.resolvidos.add(s);
                        componente = dt;

                    } else {

                        PanelGrid grid = new PanelGrid(s.getName());

                        for (int i = 0; i < s.getCardinalityMax(); i++) {
                            OutputLabel label = new OutputLabel(s.getName() +
"label"
                                    + i);

                            label.setLabel(s.getName());

                            label.adicionarElemento(new InputText(s.getName() + i,
                                    "value" + s.getName() + i));

                            grid.adicionarComponente(label);
```

```
                        }
                        this.resolvidos.add(s);
                        componente = grid;

                }
        }
        return componente;
}

private Componente getComponente(OutputStructure s) {

        Componente componente;
        if (s.getType() == 1) {

                if (s.getTransitions().size() > 0) {

                        if (s.getAllStructuresAndInternalStructures().size()
                                        + s.getDataItems().size() > 0) {

                                OutputLabel label = new OutputLabel(s.getName());
                                label.setLabel(s.getName());
                                PanelGrid g = new PanelGrid(s.getName());

                                label.adicionarElemento(g);

                                for (DataItem d : s.getDataItems()) {

                                        if (d instanceof InputDataItem) {

                                                g.adicionarComponente(this

.getComponente((InputDataItem) d));

                                        }

                                        if (d instanceof InputOptionalDataItem) {

                                                g.adicionarComponente(this

.getComponente((InputOptionalDataItem) d));
                                        }

                                        if (d instanceof OutputDataItem) {

                                                g.adicionarComponente(this

.getComponente((OutputDataItem) d));
                                        }
                                }

                                for (Structure st : s.getStructures()) {
                                        if (st instanceof InputStructure) {
                                                g.adicionarComponente(this
```

```
        .getComponente((InputStructure) st));

                                        }

                                        if (st instanceof OutputStructure) {
                                                g.adicionarComponente(this

        .getComponente((OutputStructure) st));
                                        }
                                }

                                for (Transition t : s.getTransitions()) {
                                        if (!resolvidos.contains(t)
                                                        && t.getOption() != null
                                                        && !t.getOption().replaceAll(" ",
"")
                                                                        .equals("")) {
                                                g.adicionarComponente(new
CommandLink(s.getName()
                                                                        + ":" + t.getId(),
t.getOption(), "action"
                                                                        + t.getOption()));
                                                resolvidos.add(t);
                                        }
                                }

                                componente = label;

                        } else {
                                componente = new CommandLink(s.getName(),
s.getName(),
                                                "action" + s.getName());

                        }

                } else {
                        if (s.getAllStructuresAndInternalStructures().size()
                                        + s.getDataItems().size() > 0) {

                                OutputLabel label = new OutputLabel(s.getName());
                                label.setLabel(s.getName());
                                PanelGrid g = new PanelGrid(s.getName());

                                label.adicionarElemento(g);

                                for (DataItem d : s.getDataItems()) {

                                        if (d instanceof InputDataItem) {

                                                g.adicionarComponente(this

        .getComponente((InputDataItem) d));
```

```
                                        }

                                        if (d instanceof InputOptionalDataItem) {

                                                g.adicionarComponente(this

        .getComponente((InputOptionalDataItem) d));
                                        }

                                        if (d instanceof OutputDataItem) {

                                                g.adicionarComponente(this

        .getComponente((OutputDataItem) d));
                                        }
                                }

                                for (Structure st : s.getStructures()) {
                                        if (st instanceof InputStructure) {
                                                g.adicionarComponente(this

        .getComponente((InputStructure) st));

                                        }

                                        if (st instanceof OutputStructure) {
                                                g.adicionarComponente(this

        .getComponente((OutputStructure) st));
                                        }
                                }

                                componente = label;

                        } else {
                                componente = new OutputText(s.getName(),
"#{BEAN.value"
                                                + s.getName() + "}");
                        }
                }
        } else {

                DataTable dt = new DataTable(s.getName(), "#{BEAN.cabecalho"
                        + s.getName() + "}");

                dt.setValue(s.getName());
                dt.setVar("var");

                componente = dt;

                Column c = new Column();

                for (DataItem d : s.getDataItems()) {
                        c = new Column();
```

```
                        c.adicionarComponente(new OutputText(d.getName(), "#{"
                                + dt.getVar() + "." + d.getName() + "}"));
                        c.setCabecalho(d.getName());
                        dt.adicionarColuna(c);
                }

                for (Structure si : s.getAllStructuresAndInternalStructures()) {

                        if (si.getDataItems().size() > 0) {
                                for (DataItem d : si.getDataItems()) {
                                        c = new Column();
                                        c.adicionarComponente(new
OutputText(d.getName(), "#{"
                                                        + dt.getVar() + "." + d.getName()
+ "}"));
                                        c.setCabecalho(d.getName());
                                        dt.adicionarColuna(c);
                                }
                        } else {
                                c = new Column();
                                c.adicionarComponente(new OutputText(si.getName(),
"#{"
                                                + dt.getVar() + "." + si.getName() +
"}"));
                                c.setCabecalho(si.getName());
                                dt.adicionarColuna(c);
                        }
                }

                PanelGrid grid = new PanelGrid(s.getName() + "grid");

                grid.adicionarComponente(dt);

                for (Object o : s.getTransitions()) {
                        TransitionFromSet t = (TransitionFromSet) o;

                        if (t.getNumberOfSelectedElements() == 1) {

                                c = new Column();
                                c.adicionarComponente(new CommandLink(s.getName()
+ ":"
                                                + t.getId(), t.getOption(), "action"
                                                + t.getOption()));

                                c.setCabecalho(t.getOption());

                                dt.adicionarColuna(c);

                                resolvidos.add(t);
                        } else {

                                componente = grid;
```

```
                                        SelectManyCheckbox smc = new
SelectManyCheckbox(s.getName()
                                                        + ":" + t.getId(), "#{BEAN.valueSelect"
                                                        + s.getName() + "}");

                                        smc.adicionarSelect(new SelectItem("", "", "var"));

                                        c = new Column();
                                        c.adicionarComponente(smc);

                                        dt.adicionarColuna(c);

                                        CommandLink cl = new CommandLink(s.getName() + ":"
                                                        + t.getId(), t.getOption(), "action"
                                                        + t.getOption());

                                        t.setResolvido(true);

                                        grid.adicionarComponente(cl);

                                }
                        }

                }

                resolvidos.add(s);

                return componente;
        }

        private Componente getComponente(InputEnumeratedData i) {

                Componente componente;
                if (i.getSelectionMax() == 1) {

                        SelectOneRadio radio = new SelectOneRadio(i.getName(),
                                        "#{BEAN.value" + i.getName() + "}");

                        for (String opcao : i.getListOptions()) {

                                radio.adicionarSelect(new SelectItem(opcao, opcao, opcao));

                        }

                        componente = radio;

                } else {

                        SelectManyCheckbox check = new SelectManyCheckbox(i.getName(),
                                        "#{BEAN.value" + i.getName() + "}");

                        for (String opcao : i.getListOptions()) {

                                check.adicionarSelect(new SelectItem(opcao, opcao, opcao));
```

```
                }

                componente = check;
        }
        return componente;
}

private Componente getComponente(Text t) {
        return new OutputText(t.getId(), t.getText());
}

private Componente getComponente(InputDataItem d) {
        Componente componente;
        if (d.getType() == 1 || d.getCardinalityMax() > 5) {
                OutputLabel label = new OutputLabel(d.getName() + "label");

                label.setLabel(d.getName());

                label.adicionarElemento(new InputText(d.getName(), "value"
                                + d.getName()));

                componente = label;
                resolvidos.add(d);
        } else {

                PanelGrid grid = new PanelGrid(d.getName());

                for (int i = 0; i < d.getCardinalityMax(); i++) {
                        OutputLabel label = new OutputLabel(d.getName() + "label" + i);

                        label.setLabel(d.getName());

                        label.adicionarElemento(new InputText(d.getName() + i, "value"
                                        + d.getName() + i));

                        grid.adicionarComponente(label);
                }
                resolvidos.add(d);
                componente = grid;

        }

        return componente;
}

private Componente getComponente(OutputDataItem d) {
        Componente componente = null;
        if (d.getType() == 1) {
                if (d.getTransitions().size() == 0) {
                        componente = new OutputText(d.getName(), d.getName());
                        resolvidos.add(d);
                } else {
                        Transition t = (Transition) d.getTransitions().get(0);
```

```
                componente = new CommandLink(d.getName(), d.getName() + " "
                        + t.getCondition(), "action" + t.getOption());

                resolvidos.add(t);
                resolvidos.add(d);
            }

        } else {

            if (d.getTransitions().size() == 0) {

                Column c = new Column();
                c.adicionarComponente(new OutputText(d.getName() +
"coluna1", d
                        .getName()));

                DataTable dt = new DataTable(d.getName());
                dt.adicionarColuna(c);

                componente = dt;
                resolvidos.add(d);

            } else if (d.getTransitions().size() == 1) {

                TransitionFromSet t = (TransitionFromSet) d.getTransitions()
                        .get(0);

                if (t.getNumberOfSelectedElements() == 1) {
                    Column c = new Column();
                    c.adicionarComponente(new CommandLink(d.getName()
                            + "coluna1", t.getOption(), "action"
                            + t.getOption()));

                    DataTable dt = new DataTable(d.getName());
                    dt.adicionarColuna(c);

                    componente = dt;
                    resolvidos.add(t);
                    resolvidos.add(d);
                } else {

                    SelectManyCheckbox smc = new SelectManyCheckbox(
                            d.getName(), "#{BEAn." + d.getName()
+ "}");
                    smc.adicionarSelect(new SelectItens("", "", ""));

                    CommandLink c = new CommandLink(d.getName() + ":"
                            + t.getId(), t.getCondition(), "action"
                            + t.getOption());

                    PanelGrid grid = new PanelGrid(d.getName());

                    grid.adicionarComponente(smc);
                    grid.adicionarComponente(c);
```

```
                                t.setResolvido(true);
                                componente = grid;
                                resolvidos.add(d);
                        }

                } else {
                        System.out.println("Item de Dado com mais do que 1 transição.");
                }

        }
        return componente;
}

private Componente getComponente(InputOptionalDataItem d) {
        Componente componente;
        if (d.getType() == 1 || d.getCardinalityMax() > 5) {
                OutputLabel label = new OutputLabel(d.getName() + "label");

                label.setLabel(d.getName());

                label.adicionarElemento(new InputText(d.getName(), "value"
                                + d.getName()));

                componente = label;

                resolvidos.add(d);
        } else {

                PanelGrid grid = new PanelGrid(d.getName());

                for (int i = 0; i < d.getCardinalityMax(); i++) {
                        OutputLabel label = new OutputLabel(d.getName() + "label" + i);

                        label.setLabel(d.getName());

                        label.adicionarElemento(new InputText(d.getName() + i, "value"
                                        + d.getName() + i));

                        grid.adicionarComponente(label);

                }

                resolvidos.add(d);
                componente = grid;

        }

        return componente;
}

private Componente getComponente(Transition t) {
        return new CommandLink(t.getId(), "#{BEAN.value" + t.getOption() + "}",
                        "action" + t.getOption());
```

```
	}

	private Componente getComponente(InputXORDataEntry i) {
		SelectOneRadio radio = new SelectOneRadio(i.getDataItem1().getName()
					+ "XOR" + i.getDataItem2().getName(), "#{BEAM."
					+ i.getDataItem1() + i.getDataItem2() + "}");

		radio.adicionarSelect(new SelectItem(i.getDataItem1().getName(), i
					.getDataItem1().getName(), i.getDataItem1().getName()));
		radio.adicionarSelect(new SelectItem(i.getDataItem2().getName(), i
					.getDataItem2().getName(), i.getDataItem2().getName()));

		return radio;
	}

	private Componente getComponente(InputORDataEntry i) {
		SelectManyCheckbox check = new SelectManyCheckbox(i.getDataItem1()
					.getName() + "OR" + i.getDataItem2().getName(), "#{BEAM."
					+ i.getDataItem1().getName() + i.getDataItem2().getName() +
"}");

		check.adicionarSelect(new SelectItem(i.getDataItem1().getName(), i
					.getDataItem1().getName(), i.getDataItem1().getName()));
		check.adicionarSelect(new SelectItem(i.getDataItem2().getName(), i
					.getDataItem2().getName(), i.getDataItem2().getName()));

		return check;
	}

}
```

Column.java

```
package uid.jsf;

import java.util.ArrayList;
import java.util.List;

/**
 * @author Filipe Bianchi Damiani
 */
public class Column implements Composite {

	private List<Componente> componentes;
	private String cabecalho;

	public Column() {
		componentes = new ArrayList<Componente>();
	}

	public Column(String cabecalho) {
		this.cabecalho = cabecalho;
		componentes = new ArrayList<Componente>();
	}

	@Override
	public void adicionarComponente(Componente comp) {

		this.componentes.add(comp);

	}

	public void setCabecalho(String cabecalho) {
		this.cabecalho = cabecalho;
	}

	public String getCabecalho() {
		return cabecalho;
	}

	@Override
	public List<Componente> getComponentes() {
		return this.componentes;
	}

	public String getJSF() {
		String r = "<h:column>" + "\n";
		r += "<f:facet name=\"header\">" + "\n";
		r += new OutputText(this.cabecalho, this.cabecalho).getJSF() + "\n";
		r += "</f:facet>" + "\n";

		for (Componente c : this.componentes) {
			r += c.getJSF() + "\n";
		}
```

```
                r += "</h:column>" + "\n";

                return r;
        }
}
```

CommandLink.java

```
package uid.jsf;
/**
 * @author Filipe Bianchi Damiani
 */
public class CommandLink extends ComponenteAbstrato {

	private String value;
	private String action = "pagina0.xhtml";

	public CommandLink(String id, String value, String action) {
		super(id);
		this.value = value;
		this.setAction(action);
	}

	public void setAction(String action) {
		this.action = action;
		// this.action = "pagina0.xhtml";
	}

	@Override
	public String getJSF() {

		return "<h:commandLink value=\"" + this.value + "\" action=\"#{BEAN."
				+ this.action + "}\"/>" + "\n";
	}
}
```

Componente.java

```
package uid.jsf;
/**
 * @author Filipe Bianchi Damiani
 */
public interface Componente {

	public void setId(String id);

	public String getId();

	public String getJSF();

}
```

ComponenteAbstrato.java

```
package uid.jsf;

/**
 * @author Filipe Bianchi Damiani
 */
public abstract class ComponenteAbstrato implements Componente {

	String id;

	public ComponenteAbstrato(String id) {
		this.id = id;
	}

	@Override
	public void setId(String id) {
		this.id = id;
	}

	@Override
	public String getId() {
		return this.id;
	}

}
```

Composite.java

```
package uid.jsf;

import java.util.List;
/**
 * @author Filipe Bianchi Damiani
 */
public interface Composite {

	public void adicionarComponente(Componente comp);

	public List<Componente> getComponentes();
}
```

DataTable.java

```
package uid.jsf;

import java.util.ArrayList;
import java.util.List;
/**
 * @author Filipe Bianchi Damiani
 */
public class DataTable extends ComponenteAbstrato {

	private List<Column> colunas = new ArrayList<Column>();
	private String cabecalho;
	private String var;
	private String value;

	public String getCabecalho() {
		return cabecalho;
	}

	public void setCabecalho(String cabecalho) {
		this.cabecalho = cabecalho;
	}

	public String getVar() {
		return var;
	}

	public void setVar(String var) {
		this.var = var;
	}

	public String getValue() {
		return value;
	}

	public void setValue(String value) {
		this.value = value;
	}

	public DataTable(String id) {
		super(id);
	}

	public DataTable(String id, String cabecalho) {
		super(id);
		this.cabecalho = cabecalho;
	}

	public void adicionarColuna(Column c) {
		this.colunas.add(c);
	}
```

```
	@Override
	public String getJSF() {
		String r = "<h:dataTable value=\"#{BEAN.value" + this.value
				+ "}\" var=\"" + this.var + "\"  border=\"1\" >" + "\n";
		r += "<f:facet name=\"header\">" + "\n";
		r += new OutputText(this.cabecalho, this.cabecalho).getJSF() + "\n";
		r += "</f:facet>" + "\n";

		for (Column c : this.colunas) {
			r += c.getJSF() + "\n";
		}

		r += "</h:dataTable>" + "\n";
		return r;
	}
}
```

Form.java

```
package uid.jsf;

import java.util.ArrayList;
import java.util.List;

/**
 * @author Filipe Bianchi Damiani
 */
public class Form extends ComponenteAbstrato implements Composite {

	private List<Componente> componentes;

	public Form(String id) {
		super(id);
		componentes = new ArrayList<Componente>();
	}

	@Override
	public void adicionarComponente(Componente comp) {
		this.componentes.add(comp);
	}

	public List<Componente> getComponentes() {
		return this.componentes;
	}

	@Override
	public String getJSF() {
		String r = "<h:form>" + "\n";

		for (Componente c : componentes) {
			r += c.getJSF() + "\n";
		}

		r += "</h:form>";

		return r;
	}
}
```

InputText.java

```
package uid.jsf;
/**
 * @author Filipe Bianchi Damiani
 */
public class InputText extends ComponenteAbstrato {

	private String value = "";

	public InputText(String id, String value) {
		super(id);
		this.value = value;
	}

	public void setValue(String value) {
		this.value = value;
	}

	public String getValue() {
		return value;
	}

	@Override
	public String getJSF() {
		return "<h:inputText value=\"#{BEAN." + getValue() + "}\" />" + "\n";
	}

}
```

OutputLabel.java

```
package uid.jsf;

import java.util.ArrayList;
import java.util.List;

/**
 * @author Filipe Bianchi Damiani
 */
public class OutputLabel extends ComponenteAbstrato {

	private String label;
	private List<Componente> componentes;

	public OutputLabel(String id) {
		super(id);
		componentes = new ArrayList<Componente>();
	}

	public void setLabel(String label) {
		this.label = label;

	}

	public void adicionarElemento(Componente c) {

		this.componentes.add(c);

	}

	@Override
	public String getJSF() {
		String r = "<h:outputLabel value=\"#{BEAN.label" + this.label + "}\">"
				+ "\n";

		for (Componente c : componentes) {
			r += c.getJSF() + "\n";
		}

		r += "</h:outputLabel>" + "\n";

		return r;
	}
}
```

OutputText.java

```
package uid.jsf;
/**
 * @author Filipe Bianchi Damiani
 */
public class OutputText extends ComponenteAbstrato {

	private String valor;

	public OutputText(String id, String valor) {
		super(id);
		this.setValor(valor);
	}

	public void setValor(String valor) {
		this.valor = valor;
	}

	public String getValor() {
		return valor;
	}

	@Override
	public String getJSF() {
		// TODO Auto-generated method stub
		return "<h:outputText value=\"" + this.getValor() + "\" />"
				+ "\n";
	}

}
```

PanelGrid.java

```
package uid.jsf;

import java.util.ArrayList;
import java.util.List;

/**
 * @author Filipe Bianchi Damiani
 */
public class PanelGrid extends ComponenteAbstrato implements Composite {

	protected List<Componente> componentes;

	public PanelGrid(String id) {
		super(id);
		componentes = new ArrayList<Componente>();
	}

	@Override
	public void adicionarComponente(Componente comp) {
		this.componentes.add(comp);
	}

	@Override
	public List<Componente> getComponentes() {
		return this.componentes;
	}

	@Override
	public String getJSF() {

		String r = "<h:panelGrid columns=\"1\">" + "\n";

		for (Componente c : componentes) {
			r += c.getJSF() + "\n";
		}

		r += "</h:panelGrid>";

		return r;
	}
}
```

Select.java

```
package uid.jsf;

/**
 * @author Filipe Bianchi Damiani
 */
public abstract class Select extends ComponenteAbstrato {

	private String label;

	public Select(String id, String label) {
		super(id);
		this.setLabel(label);
		// TODO Auto-generated constructor stub
	}

	public void setLabel(String label) {
		this.label = label;
	}

	public String getLabel() {
		return label;
	}

}
```

SelectItem.java

```
package uid.jsf;
/**
 * @author Filipe Bianchi Damiani
 */
public class SelectItem extends Select {

	private String itemValue;

	public SelectItem(String id, String label, String itemValue) {
		super(id, label);
		this.itemValue = itemValue;
	}

	@Override
	public String getJSF() {
		String r = "<f:selectItem itemValue=\"" + this.itemValue
				+ "\" itemLabel=\"" + this.getLabel() + "\" />";
		return r;
	}
}
```

SelectItens.java

```
package uid.jsf;
/**
 * @author Filipe Bianchi Damiani
 */
public class SelectItens extends Select {

	private String value;

	public SelectItens(String id, String value, String label) {
		super(id, label);
		this.value = value;
	}

	@Override
	public String getJSF() {
		String r = "<f:selectItems value=\"" + this.value + "\" itemLabel=\""
				+ this.getLabel() + "\" />";
		return r;
	}

}
```

SelectManyCheckbox.java

```
package uid.jsf;

import java.util.ArrayList;
import java.util.List;

/**
 * @author Filipe Bianchi Damiani
 */
public class SelectManyCheckbox extends ComponenteAbstrato {

	private List<Select> items;
	private String value;

	public SelectManyCheckbox(String id, String value) {
		super(id);
		this.items = new ArrayList<Select>();
		this.setValue(value);
	}

	public void adicionarSelect(Select select) {
		this.items.add(select);
	}

	public void setValue(String value) {
		this.value = value;
	}

	@Override
	public String getJSF() {

		String r = "<h:selectManyCheckbox layout=\"pageDirection\" value=\""
				+ this.value + "\">" + "\n";

		for (Select s : this.items) {
			r += s.getJSF() + "\n";
		}

		r += "</h:selectManyCheckbox>" + "\n";

		return r;
	}
}
```

SelectOneRadio.java

```
package uid.jsf;

import java.util.ArrayList;
import java.util.List;

/**
 * @author Filipe Bianchi Damiani
 */
public class SelectOneRadio extends ComponenteAbstrato {

	private List<Select> items;
	private String value;

	public SelectOneRadio(String id, String value) {
		super(id);
		this.items = new ArrayList<Select>();
		this.setValue(value);
	}

	public void adicionarSelect(Select select) {
		this.items.add(select);
	}

	public void setValue(String value) {
		this.value = value;
	}

	@Override
	public String getJSF() {

		String r = "<h:selectOneRadio value=\"" + this.value + "\">" + "\n";

		for (Select s : this.items) {
			r += s.getJSF() + "\n";
		}

		r += "</h:selectOneRadio>" + "\n";

		return r;
	}
}
```