Capítulo

4

IFML: Linguagem de Modelagem de Fluxo de Interação[1]

Raul Sidnei Wazlawick

Abstract

This chapter shows how to design front-end interfaces for information systems using a hypertext specification language known as IFML: Interaction Flow Modeling Language. Initially, the chapter presents the information view components of IFML, which are used in connection to the design class diagrams to manage the presentation of the information. The integration of view components and flows into pages and areas is also presented, as well as some interface design patterns. Operation components may also be used in connection to view components in order to define basic operations over objects and to connect the view to the operations implemented by the controller. The chapter concludes with the presentation of a CRUD interface modeled in IFML and a high-level design for a system use case.

Resumo

Este capítulo mostra como projetar uma interface com usuário para sistemas de informação usando uma linguagem de especificação de hipertexto conhecida como IFML: Interaction Flow Modeling Language. Inicialmente, os componentes de informação da IFML são apresentados, os quais são usados em conexão com diagramas de classe de design para gerenciar a apresentação de informação. A integração dos componentes de visão e fluxos em páginas e áreas é também apresentada, bem como alguns padrões de design de interface. Componentes de operação podem ser usados em conexão com os componentes de visão para definir operações básicas sobre objetos, bem como associar a visão com operações implementadas pelo controlador. O capítulo termina com a apresentação de uma interface CRUD modelada em IFML e um design em alto nível para um caso de uso de sistema.

[1] Este texto é uma adaptação do capítulo 12 do livro "Análise e Design Orientados a Objetos para Sistemas de Informação: Modelagem com UML, OCL e IFML", do mesmo autor, publicado pela Elsevier em 2015 e também disponível em inglês com o título "Object-Oriented Analysis and Design for Information Systems: Modeling with UML, OCL, and IFML", publicado pela Morgan Kaufmann em 2014.

4.1. Introdução ao Design da Camada de Interface

Sistemas de software, especialmente aplicações voltadas para sistemas de informação, usualmente são projetadas em camadas, de forma que diferentes preocupações sejam abordadas em diferentes partes do design. O modelo de três camadas (Eckerson, 1995) é, assim, bastante popular: ele separa a arquitetura do sistema nas camadas de *apresentação* (interface com usuário), *domínio* (lógica de transformação de dados) e *persistência* (armazenamento de dados).

O processo de análise e design de sistemas orientados a objetos, em linhas gerais, parte de uma análise de requisitos a partir da qual pode-se inicialmente definir a camada de domínio como um modelo usualmente representado por um diagrama de classes (Miles & Hamilton, 2006). A camada de persistência normalmente é derivada desse modelo já que, via de regra, cada classe que represente informação de domínio (entidade) será representada, por exemplo, através de uma tabela relacional, embora outras formas sejam possíveis. Como o mapeamento entre objetos e tabelas relacionais nem sempre é tão direto (Ireland *et al*., 2009), os responsáveis pelo design de banco de dados acabam usando diagramas como o Entidade-Relacionamento (Chen, 1976) para representar as tabelas que vão conter as informações gerenciadas pelo sistema. Várias ferramentas CASE (Computer Aided Software Engineering) já permitem há muitos anos a geração de código a partir destes diagramas, facilitando, assim, o trabalho dos programadores.

Já em relação à camada de apresentação, a abordagem usual acaba sendo baseada no desenho de interfaces acompanhado de comentários que explicam como cada janela ou tela deve se comportar. Eventualmente, o fluxo de interação do usuário com a interface acaba sendo modelado com diagramas de atividades, máquina de estados ou mesmo fluxogramas.

Mais recentemente, porém, sentiu-se a necessidade de definir um diagrama específico para modelagem de fluxos de interação entre usuário e computador, o qual pudesse, de forma clara e sintética, representar os diferentes componentes de uma interface com usuário e como os dados transitam entre esses componentes (Object Management Group, 2015). Além disso, seria necessário também que tal diagrama permitisse associar os eventos de interface, como, por exemplo, o pressionar de um botão, com chamadas a operações de sistema implementadas na camada de domínio e usualmente encapsuladas por uma classe controladora-fachada (Gamma *et al*., 1995).

A partir dessa necessidade é que foi desenvolvida a Linguagem de Modelagem de Fluxo de Interação ou *Interaction Flow Modeling Language* (IFML), a qual é uma linguagem compatível com a UML (*Unified Modeling Language*) que pode ser usada para modelar interfaces com usuário nas quais há uso intensivo de dados. Ela é uma evolução de WebML (Ceri, *et al*., 2003) e foi adotada como padrão pela OMG em 2015 (Object Management Group, 2015).

Este capítulo tem como objetivo apresentar os conceitos mais fundamentais de IFML e seu potencial de modelagem. Mais informação pode ser encontrada no site oficial da linguagem[2] e no livro de Brambilla e Fraternali (2014). A notação usada neste capítulo corresponde à usada na ferramenta WebRatio®[3], a qual é a primeira a implementar o padrão.

---

[2] A versão 1.0 da especificação oficial de IFML foi publicada em março de 2015. Disponível em: http://www.omg.org/spec/IFML/1.0/PDF/.

[3] http://www.webratio.com

A Seção 4.2 apresenta uma visão geral dos diferentes modelos associados com o uso da linguagem. A Seção 4.3 apresenta os componentes de visão, ou seja, os elementos que permitem especificar que dados serão apresentados ou coletados em uma interface gráfica. A Seção 4.4 introduz os conceitos de organização de um *site* com o uso de contêineres, ou seja, áreas e páginas. A Seção 4.5 apresenta os tipos de *fluxos* que podem ser definidos para navegação entre páginas e componentes de visão, bem como para obtenção de dados. A Seção 4.6 apresenta alguns padrões de interface *Web* bastante comuns em aplicações reais. A Seção 4.7 apresenta os componentes de operação que permitem especificar transformações nos dados, bem como conectar os componentes de visão com operações de sistema a serem executadas na camada de domínio. A Seção 4.8 apresenta a especificação completa de um CRUD (*Create*, *Retrieve*, *Update* e *Delete*) com IFML. A Seção 4.9 mostra como modelar uma interface com usuário em IFML a partir de um caso de uso expandido. A Seção 4.10 apresenta as considerações finais do capítulo.

4.2. A Linguagem

IFML é voltada para modelagem de interfaces, especialmente aquelas que fazem uso intensivo de dados, tais como sistemas de informação. O desenvolvimento de aplicações com IFML considera que pelo menos cinco modelos sejam usados:

- *Modelo estrutural*: ele representa as estruturas e a organização dos dados. Ceri *et al*. (2003) utilizam o diagrama entidade-relacionamento para definir o modelo estrutural. Entretanto, o diagrama de classes de UML e outros podem ser usados com os mesmos propósitos.
- *Modelo de derivação*: ele foi originalmente proposto para modelar dados que podem ser calculados a partir de outros dados. O modelo de derivação pode ser entendido como o conjunto de definições de atributos e associações derivadas que usualmente são definidos no diagrama de classes e podem ser escritos, por exemplo, com a linguagem OCL (Object Management Group, 2010).
- *Modelo de composição*: ele inclui a definição de contêineres como páginas (Seção 4.4.1) e áreas (Seção 4.4.2) que agregam os de componentes de visão mais básicos (Seção 4.3).
- *Modelo de navegação*: ele permite definir os fluxos de navegação entre as páginas e componentes de visão (Seção 4.5.1).
- *Modelo de apresentação*: ele permite definir o posicionamento dos componentes de visão da interface e sua aparência.

Este capítulo trata particularmente dos modelos de *composição* e *navegação*. Os modelos estrutural e de derivação podem ser encontrados nos trabalhos de Ceri *et al*. (2003) ou Wazlawick (2015). Para o modelo de apresentação, a ferramenta WebRatio®[4] permite o uso de planilhas de estilo XSL (*Extensible Stylesheet Language* – Linguagem Extensível de Folha de Estilos)[5]. A ferramenta de geração de código usa essas regras de apresentação para produzir páginas de acordo com o design produzido com outras ferramentas. Para os exemplos apresentados neste capítulo, são mostradas as interfaces padrão renderizadas pela ferramenta sem o uso de estilos ou design especial. Mesmo os

[4] A primeira ferramenta compatível com IFML.

[5] Disponível em: <www.w3.org/Style/XSL/>.

nomes dos campos foram deixados idênticos aos nomes dos atributos das classes com o objetivo de promover a compreensão mais do que a qualidade visual da apresentação.

4.3. Componentes de Visão

Componentes de visão são os elementos mais básicos em uma especificação IFML. Componentes de visão podem ser associados a classes de design e permitem a visualização de instâncias dessas classes. Componentes de visão em IFML podem ser especializados. Alguns exemplos de especializações de componentes de visão são:

- *Detalhes*: mostra informação sobre um único objeto.
- *Detalhes múltiplos*: mostra informação sobre uma coleção de instâncias da mesma classe.
- *Lista simples*: mostra múltiplas instâncias de uma classe na forma de uma lista.
- *Lista*: um melhoramento da lista simples que permite *scrolling* e ordenação dinâmica.
- *Lista de múltipla escolha*: uma lista que permite seleção múltipla.
- *Hierarquia*: mostra múltiplas instâncias de diferentes classes organizadas em uma hierarquia, tal como todo-parte ou mestre-detalhe (Seffah, 2015).
- *Hierarquia recursiva*: mostra múltiplas instâncias de uma única classe organizadas hierarquicamente, como no caso de estruturas organizacionais.
- *Scroller*: permite as operações típicas de *scroll* sobre uma sequência de objetos.
- *Formulário*: permite entrada de dados baseada em formulário.
- *Formulário múltiplo*: similar ao formulário, mas permite a edição de múltiplos objetos de cada vez.
- *Mensagem*: permite imprimir mensagens em uma página.
- *Calendário*: mostra um calendário perpétuo.

Existem outros componentes de visão e novos componentes podem ser definidos pelo usuário usando estereótipos (Brambilla & Fraternali, 2014). Este capítulo apresenta informações sobre os mais fundamentais dentre eles. Para cada tipo de componente de visão uma definição gráfica é apresentada, bem como a possível aparência da página renderizada por WebRatio®. Todos os exemplos são baseados no diagrama de classes de design (DCD) da Figura 4 .1.

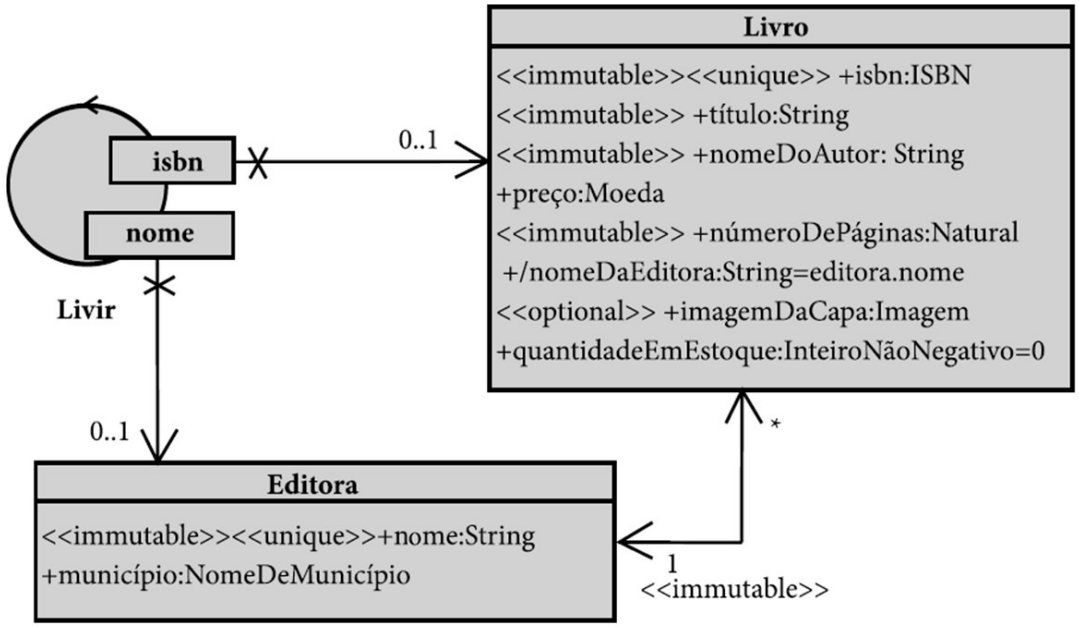


Figura 4.1. Diagrama de classes de design de referência.

Na Figura 4.1, as classes *Editora* e *Livro* representam conceitos ou entidades e a classe *Livir* (Livraria Virtual) é uma classe controladora de sistema, ou seja, ela é fachada que encapsula os objetos em relação à camada de interface. O estereótipo <<*immutable*>> significa que o atributo ou associação depois de definido não pode mais ser modificado, <<*unique*>> significa que o atributo não pode ter seu valor repetido em diferentes instâncias da classe e <<*optional*>> significa que o atributo pode ter valor indefinido (todos os demais atributos devem ser obrigatoriamente definidos desde a criação do objeto ou ele estará inconsistente). A barra "/" indica que o atributo ou papel de associação é *derivado*, ou seja, obtido através de um cálculo.

4.3.1. Componente de Visão *Detalhes*

O componente de visão *detalhes* apresenta informação sobre um único objeto de uma dada classe. Ele é definido pelas seguintes propriedades:

- Um *nome* que é escolhido pelo designer.
- Uma *entidade* ou *fonte de dados* que é usualmente uma classe do diagrama de classes de design.
- *Atributos exibidos* que é uma lista de atributos da classe a serem exibidos quando o componente detalhes é renderizado.

A maioria dos componentes de visão como *detalhes* também permite a definição de uma ou mais *expressões condicionais* que funcionam como seletores ou filtros sobre o conjunto de instâncias a serem exibidas. Existem três tipos de expressões condicionais:

- *Condição-chave*: cada objeto no modelo de dados é identificado por um OID (*Object IDentifier*) – um atributo que é único, obrigatório e imutável. Ele não deve ser confundido com nenhum dos atributos da classe conceitual, mesmo os estereotipados com <<*unique*>> e <<*immutable*>>, tais como ISBN ou CPF. O OID é um código gerado internamente e corresponde à chave primária de um objeto. OIDs não podem ser conhecidos ou atualizados pelo usuário. Uma

condição-chave permite que sejam selecionadas uma ou mais instâncias de uma dada classe baseando-se no valor do seu atributo OID.

- *Condição de atributo*: ela permite a seleção de uma ou mais instâncias baseando-se nos valores de seus atributos. Operações como *maior que*, *igual* ou *contém* podem ser usadas para comparar o valor de um atributo com um dado parâmetro.
- *Condição de papel*: ela permite a seleção de uma ou mais instâncias baseando-se nas suas ligações.

A Figura 4 .2 exibe em seu lado esquerdo um componente de visão *detalhes* que deve mostrar dados sobre uma instância de um livro para um dado ISBN. A expressão condicional *isbn*=0517149257 determina qual instância é mostrada. Se esta instância não existisse, então nenhuma informação seria renderizada por aquele componente. Como o ISBN é um atributo que é único e obrigatório, é seguro assumir que não mais do que um livro é apresentado por este componente de visão detalhes. O lado direito da figura mostra uma possível renderização Web para o componente de visão detalhes.

Figura 4.2. Um componente de visão detalhes com uma condição de atributo simples e sua renderização.

A renderização final da página pode variar porque é possível adicionar definições de estilo de forma que interfaces com diferentes design e aparência possam ser produzidas.

Se o componente de visão detalhes da Figura 4 .2 for mudado para permitir que apenas os atributos *título* e *nomeDoAutor* apareçam, então ele produziria o resultado renderizado na Figura 4 .3.

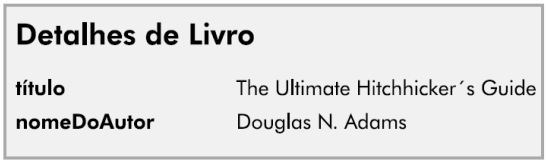


Figura 4.3. Uma possível renderização de um componente de visão detalhes com um número menor de atributos selecionados.

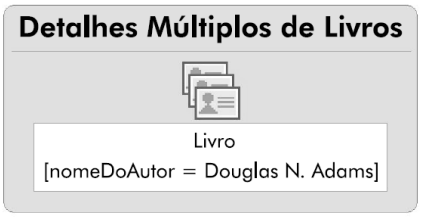
Usualmente, a seleção dos atributos a serem mostrados não é indicada graficamente no diagrama IFML. Com o uso de uma ferramenta de edição, essa informação fica oculta nos próprios componentes de visão e pode ser consultada quando se exibe informações internas sobre o componente.

4.3.2. Componente de Visão *Detalhes Múltiplos*

Detalhes múltiplos apresenta um grupo de instâncias de uma única classe de uma vez. Além das três propriedades já mostradas para o componente de visão detalhes, ele adiciona as seguintes propriedades:

- *Atributos de ordenação*: são os atributos usados para ordenar as instâncias.
- *Máximo de resultados*: especifica o número máximo de instâncias que podem ser recuperadas. Por *default*, todas as instâncias da classe são recuperadas.
- *Distinto*: especifica que elementos duplicados não serão mostrados mais do que uma vez.

O componente de visão detalhes múltiplos também permite a definição e o uso de expressões condicionais para selecionar quais elementos devem ser mostrados. A Figura 4 .4 apresenta um exemplo de detalhes múltiplos e sua renderização considerando que apenas três livros satisfazem a expressão condicional e que a ordenação acontece pelo título.



Detalhes Múltiplos de Livros

isbn	título	nomeDoAutor	preço	número DePáginas	imagem DaCapa	quantidade EmEstoque
0671746723	Dirk Gently's Holistic Detective Agency	Douglas N. Adams	6.99	306		12
0671742515	The Long Dark Tea-Time of the Soul	Douglas N. Adams	6.99	307		4
0517149257	The Ultimate Hitchhicker's Guide	Douglas N. Adams	122	815		4

Figura 4.4. Um componente de visão detalhes múltiplos e sua renderização.

Em oposição ao componente de visão detalhes que mostra apenas uma instância de cada vez, *detalhes múltiplos* apresenta todos os objetos que satisfazem a expressão condicional.

Se for necessário usar mais do que um atributo para determinar os objetos a serem apresentados pelo componente, então uma combinação de expressões condicionais pode ser usada. Essa combinação pode usar os operadores *and* e *or*. A Figura 4 .5 apresenta um componente de visão detalhes múltiplos que mostra instâncias que têm *númeroDePáginas* maior do que 100 e *preço* menor do que R$ 150,00.

A ferramenta usada para gerar o elemento apenas mostra a lista de condições usadas, sem especificar se elas são combinadas com *and* ou *or*. Para consultar essa informação é necessário abrir o componente e verificar sua especificação.

Figura 4.5. Um componente de visão detalhes múltiplos com uma expressão de condição composta.

4.3.3. Componente de Visão *Lista Simples*

O componente de visão *lista simples* apresenta um ou mais atributos de um conjunto de instâncias de uma classe na forma de uma lista. Enquanto o componente *detalhes múltiplos* é normalmente usado para visualizar os detalhes de vários objetos em uma mesma página, a lista simples e suas variações são usadas quando uma lista ou menu são necessários para o usuário selecionar um ou mais elementos e realizar ações com eles. As propriedades das listas simples são as mesmas do componente de visão detalhes múltiplos.

É possível, por exemplo, definir listas simples para selecionar livros com base em seus títulos. A Figura 4 .6 apresenta o componente de visão para uma lista simples de títulos de livros e sua possível renderização. A ausência de uma expressão condicional no componente de visão de lista simples implica que os títulos de todas as instâncias de *Livro* serão mostrados.

Figura 4.6. Lista Simples e sua renderização.

4.3.4. Componente de Visão *Lista*

O componente de visão *lista* é uma versão aprimorada da lista simples com operações de *scroll* e reordenação dinâmica dos elementos a partir de um simples clique no cabeçalho da respectiva coluna de atributo. Além das propriedades já definidas para listas simples, ele inclui as seguintes propriedades específicas:

- *Ordenável*: um atributo booleano que quando é verdadeiro define que a lista pode ser ordenada dinamicamente.

- *Atributos de ordenação default*: se a lista for ordenável, define os atributos de ordenação que podem ser selecionados pelo usuário para ordenar a lista e também o critério default de ordenação quando a lista é renderizada pela primeira vez.
- *Tamanho do histórico de ordenação*: especifica quantas ações de ordenação do usuário devem ser lembradas pela lista. Assim, por exemplo, se o usuário ordenar pela coluna 3, depois pela coluna 1 e depois pela coluna 5, se apenas uma ação de ordenação for lembrada, ele poderá desfazer a última ordenação (pela coluna 5) e retornar à configuração com a ordenação pela coluna 1, mas não poderá mais retornar para a ordenação pela coluna 3 com a ação de desfazer.
- *Assinalável*: se verdadeiro, uma *checkbox* é renderizada para cada elemento da lista.
- *Fator de bloco*: especifica quantos elementos são mostrados de cada vez. Se não foi especificado ou se definido como -1, todos os elementos são mostrados. Se o fator de bloco for definido, então os comandos de *scrolling* são renderizados para a lista.

A Figura 4 .7 mostra o componente de visão IFML para uma lista e sua renderização. Esta lista é definida para a classe *Livro* com três atributos: *nomeDoAutor*, *título* e *preço*. O fator de bloco é definido como 3.

Observe no topo da renderização na Figura 4.7 os comandos de *scroll* para a lista do primeiro até o último grupo de três elementos. Neste momento, os primeiros três elementos estão sendo mostrados, ordenados pelo seu título.

Figura 4.7. Uma lista e sua possível renderização.

Se o usuário clicar em uma das ligações no topo de cada coluna, então os elementos da lista serão rearranjados e ordenados por aquela coluna. Se o usuário clicar, por exemplo, em *nomeDoAutor*, as linhas serão ordenadas pelos nomes dos autores.

4.3.5. Componente de Visão *Lista de Seleção Múltipla*[6]

A *lista de seleção múltipla* permite que o usuário selecione um conjunto de elementos de uma lista. Fora isso, ela é muito similar a uma lista simples. A Figura 4 .8 apresenta a definição IFML de uma lista múltipla e sua renderização.

[6] *Checkable list*.

Figura 4.8. Lista de seleção múltipla e sua renderização.

Se um evento for associado à seleção de múltiplos objetos então a lista também renderizará um botão para ativar o evento.

4.3.6. Componente de Visão *Formulário*

Um *formulário* é usado para entrada de dados. Ele é muito útil para a criação de novas instâncias e também para fornecer parâmetros para pesquisas, consultas e comandos.

Um formulário é composto por um conjunto de *campos*. Cada campo contém um valor alfanumérico. Existem campos textuais, campos de seleção e campos de seleção múltipla, embora outros possam ser definidos.

Um formulário pode ser ou não associado a uma classe. Se ele for associado a uma classe, então seus campos podem ser associados aos atributos da classe.

Formulários que são usados para coletar parâmetros para uma consulta ou comando usualmente não precisam ser associados a classes. Por exemplo, um formulário que aceita palavras-chave para pesquisar livros não precisa estar associado a qualquer classe.

Os campos de um formulário têm as seguintes propriedades (além de outras):

- *Nome*: o nome do campo atribuído pelo designer.
- *Atributo*: se o formulário é associado a uma classe, então o campo pode ser associado a um de seus atributos.
- *Tipo de conteúdo*: define o tipo do campo, que pode ser *string*, *text*, *blob* ou *url*.
- *Pré-carregado*: o campo pode ser pré-carregado com valores que são mostrados ao usuário quando o formulário é renderizado.
- *Modificável*: define se o valor inicial de um campo pode ser trocado ou não. O *default* é que o campo pode ser atualizado.

A Figura 4 .9 apresenta um formulário que é associado à classe Livro e sua renderização.

Figura 4.9. Um formulário e sua renderização.

4.3.7. Componente de Visão *Hierarquia*

Hierarquias são usadas para mostrar dependências entre objetos com dois ou mais níveis. Elas são úteis, por exemplo, para mostrar objetos que se relacionam por uma associação de um para muitos, como *Editora* e *Livro*, ou parte-todo como *Livro* e *Capítulo*. A Figura 4 .10 mostra uma definição para uma hierarquia de dois níveis com *Editora* no topo e *Livro* no segundo nível.

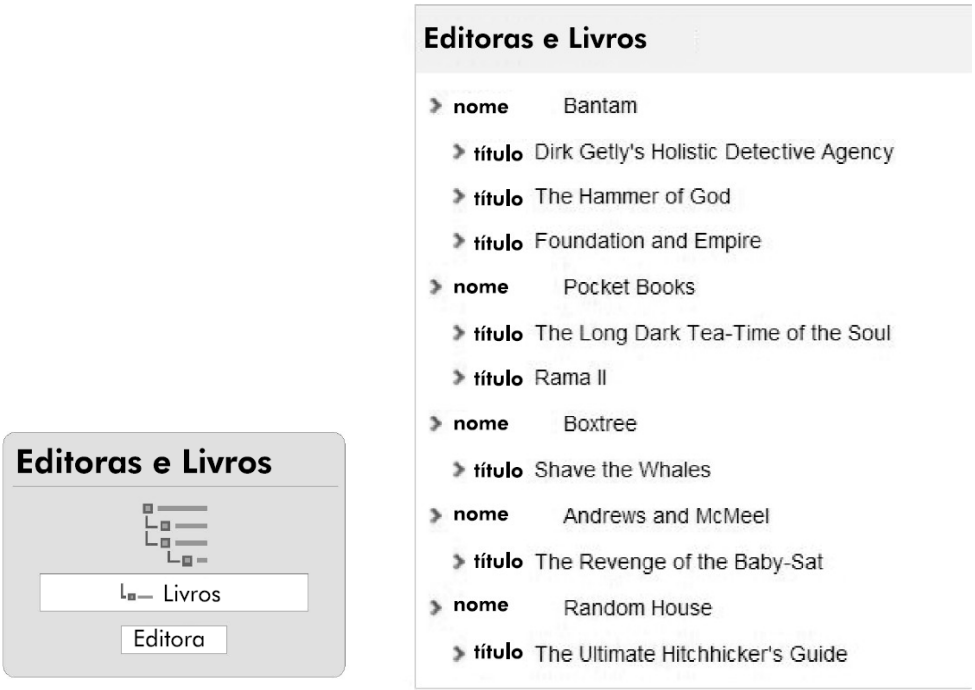


Figura 4.10. Uma hierarquia e sua renderização.

O segundo nível usualmente define uma *expressão condicional baseada em papel*. No exemplo, a expressão condicional baseada em papel define que apenas livros que estiverem ligados a uma dada editora são mostrados abaixo dela. Na ausência dessa

expressão condicional, todos os livros seriam mostrados abaixo de cada editora, mas agora este não é o caso. A renderização da Figura 4 .10 mostra uma hierarquia com livros ligados às suas respectivas editoras.

Existem situações que requerem a aplicação do padrão *hierarquia organizacional* (Fowler, 2003) onde um conceito tem subcomponentes que pertencem à mesma classe, a qual pode ser recursivamente aninhada. Para esses casos, IFML fornece o componente de visão chamado *hierarquia recursiva*. A Figura 4 .11 mostra uma classe que define uma estrutura organizacional com subestruturas e o componente de visão *hierarquia recursiva* definido para essa classe. Note que a fonte dos dados para essa visão é o nome de papel da associação e não o nome da classe.

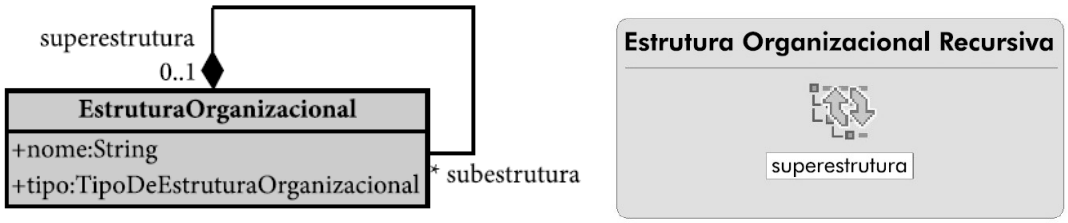


Figura 4.11. Uma classe com associação para si própria e uma visão de hierarquia recursiva para ela.

A definição e a renderização de uma hierarquia recursiva são similares à definição e renderização de uma hierarquia normal. As únicas diferenças são que a classe detalhe e a classe mestre são as mesmas e o número de níveis é indeterminado.

4.4. Organização de Hipertexto

Os componentes de visão são os elementos de interface mais locais. Entretanto, é também necessário criar modelos mais amplos para uma interface, incluindo grupos de páginas inter-relacionadas, regiões de navegação, *landmarks,* e assim por diante. Este tipo de modelo é descrito nesta seção e é referenciado aqui como *organização de hipertexto*.

A IFML organiza os componentes de visão em estruturas chamadas *contêineres*. Há basicamente três tipos de contêineres, as páginas, as áreas e as visões de site, as quais são explicadas, nas seções seguintes.

4.4.1. Páginas

Páginas são os elementos de interface que são efetivamente acessados pelo usuário. Páginas são, assim, um tipo de contêiner porque uma página pode conter um ou mais componentes de visão e possivelmente outros contêineres.

A representação IFML de uma página contém um nome e um conjunto opcional de componentes de visão ou subpáginas incluídas. Por exemplo, a Figura 4 .12Erro: Origem da referência não encontrada apresenta uma página que contém uma lista de editoras e uma lista de livros e sua renderização.

O símbolo na Figura 4 .12Erro: Origem da referência não encontrada indica que esta página é uma *homepage*, ou seja, é a página inicial de uma aplicação. O outro símbolo no canto superior direito da página não tem nenhum significado especial em IFML; trata-se apenas de um comando específico da ferramenta para destacar ou obscurecer elementos gráficos.

Os componentes de visão que aparecem dentro da página na Figura 4.12 são independentes um do outro porque não há fluxos (Seção 4.5) entre eles.

Por *default*, componentes de visão definidos na mesma página são renderizados um abaixo do outro e ocupam a largura total da página, mas é possível usar o *modelo de apresentação* como mencionado anteriormente, para definir componentes de visão em diferentes posições e tamanhos. Na ferramenta WebRatio, o modelo de apresentação consiste basicamente em uma grade na qual os componentes de visão são posicionados e dimensionados dentro de uma página. No exemplo da Figura 4 .13Erro: Origem da referência não encontrada, os componentes de visão são colocados lado a lado.

> Principal

Lista de Editoras

nome	município
Andrews and McMeel	Kansas City
Bantam	New York
Boxtree	London
Pocket Books	New York
Random House	New York

Lista de Livros

isbn	título	nomeDoAutor	preço	número DePáginas	quantidade EmEstoque
0671746723	Dirk Getly's Holistic Detective Agency	Douglas N. Adams	6.99	306	12
0553293370	Foundation and Empire	Isaac Asimov	5.99	282	3
0553286587	Rama II	Arthur C. Clarke and Gentry Lee	6.99	466	2
0752208497	Shave the Whales	Scott Adams	6.99	128	7
055356871X	The Hammer of God	Arthur C. Clarke	6.99	240	1
0671742515	The Long Dark Tea-Time of the Soul	Douglas N. Adams	6.99	307	21
0836218663	The Revenge of the Baby-Sat	Bill Waterson	8.95	127	4
0517149257	The Ultimate Hitchhicker's Guide	Douglas N. Adams	129	813	6

Figura 4.12. Uma página com dois componentes de visão e sua renderização.

Figura 4.13. Um arranjo de renderização diferenciado para a página mostrada na Figura 4 .12Erro: Origem da referência não encontrada.

4.4.2. Áreas

Áreas são grupos de contêineres que têm afinidades, tais como um conjunto de funcionalidades relacionadas a um objetivo de usuário. Áreas podem conter grupos de páginas relacionadas ou ainda conter outras áreas, caso a interface deva ser estruturada de forma hierárquica. No modelo IFML da Erro: Origem da referência não encontradaErro: Origem da referência não encontrada, as áreas aparecem como retângulos sombreados e as páginas como retângulos brancos.

Áreas são por default renderizadas como menus situados no topo das aplicações. As páginas (ou outras áreas) contidas em uma área de nível mais alto são renderizadas como submenus.

4.4.3. Visões de Site

Uma *visão de site* é um pacote que contém uma aplicação inteira que pode ser disponibilizada para usuários. Ela usualmente contém um grupo de *áreas* e *páginas* para a aplicação. A Erro: Origem da referência não encontradaErro: Origem da referência não encontrada apresenta um exemplo de uma visão de site com uma página (*Principal*) e duas áreas (*Administração* e *Cliente*).

Figura 4.14. Uma visão de site.

A Erro: Origem da referência não encontrada mostra uma visão de site em estágio inicial de especificação, onde apenas a estrutura de mais alto nível é mostrada. Uma

visão completamente especificada mostraria adicionalmente os componentes de visão dentro das páginas.

Essa estrutura seria renderizada como um menu de alto nível com três opções: *Principal*, *Administrativo* e *Compradores*. O menu *Administrativo* teria como opções *Relatórios* e *Manutenção*. O menu *Compradores* teria como opções *Acesso de Comprador*, *Pedidos* e *Registro*. Já o menu *Principal* não teria opções pois ele acessaria diretamente a página *Principal*.

4.4.4. Home, Landmark e Default

Na Erro: Origem da referência não encontradaErro: Origem da referência não encontrada, alguns contêineres têm propriedades especiais:

- [*H*] ou  indica a *homepage*. A homepage é a página inicial de uma aplicação. Quando um usuário acessar o *Website*, ele vai acessar a *homepage* por default. Apenas páginas podem ter essa propriedade.
- [*D*] indica a página *default* de uma área. Quando um usuário navega para uma área específica, ele vai iniciar na página default. Assim, a página default é uma espécie de "homepage" para uma área. Apenas páginas podem ter essa propriedade.
- [*L*] representa uma área ou página *landmark*. Isso significa que a página ou área é diretamente acessível de qualquer outra página dentro da mesma área. Isso evita a necessidade de criar muitos fluxos para páginas que devem ser acessadas de vários lugares.

Apenas uma *homepage* pode existir para uma dada visão de site e apenas uma página *default* para uma dada área. Entretanto, qualquer número de páginas ou áreas *landmark* podem existir em qualquer contêiner.

4.5. Fluxos

Um *fluxo* é uma conexão orientada entre duas páginas ou componentes de visão. Fluxos podem ser usados não apenas para definir possibilidades de navegação entre páginas, mas também para definir dependências de dados. Por exemplo, selecionar um elemento em um componente de visão lista pode fazer com que informação sobre o elemento apareça em um componente de visão detalhes. Isso pode ser conseguido pela definição de um fluxo indo da lista para o componente detalhes.

Existem dois tipos de fluxos em IFML, os fluxos normais de navegação e os fluxos de dados, os quais serão explicados nas subseções 4.5.1 e 4.5.2. Já as subseções 4.5.3 e 4.5.4 descrevem os vínculos de parâmetro simples e multivalorado, que permitem que fluxos normais de navegação e fluxos de dados transportem informação de um componente para outro.

4.5.1. Fluxo Normal de Navegação

Fluxos normais de navegação são um tipo de fluxo que é renderizado como uma ligação clicável em sua página ou componente de visão de origem. Quando uma ligação é clicada, ela provoca a navegação para a respectiva página ou componente de visão. O fluxo é representado por uma seta.

A Figura 4 .15 mostra um exemplo de um fluxo de navegação normal entre duas páginas. A renderização resultante da página *Editoras* contém uma ligação para a

página *Livros* (no canto superior direito). Podemos ver na figura que o fluxo não é ligado aos componentes de visão, mas as páginas. Portanto, a ligação será renderizada apenas uma vez na página na qual a lista de livros é visualizada.

Se o fluxo tem sua origem no componente de visão de lista, como na Figura 4 .16, então cada linha na lista deve ter uma ligação individual para a página *Livros*.

Porém, esta lista é ainda apenas um conjunto de ligações que tem o mesmo efeito: navegar para a página *Livros*. As seguintes seções mostram como este tipo de fluxo pode ser usado para transportar informação de um componente de visão para outro, tornando possível, por exemplo, visualizar apenas os livros de uma dada editora.

Figura 4.15. Um uxo de navegação normal entre páginas e a renderização da página de origem.

Figura 4.16. Um uxo de um componente de visão para uma página e sua renderização.

4.5.2. Fluxo de Dados

Fluxos de dados são um tipo de fluxo que não é renderizado, mas que define dependências entre componentes de visão[7]. Setas tracejadas representam fluxos de dados.

Fluxos de dados não definem navegação entre componentes de visão. Eles são usados para obter valores de um componente sem estar navegando a partir dele. Eles são particularmente úteis quando um componente necessita de dados de mais do que um único componente, já que o usuário só poderá estar navegando a partir de um deles.

4.5.3. Vínculo de Parâmetro Simples

Um *vínculo de parâmetro simples* ou simplesmente *vínculo de parâmetro* é uma peça de informação que é transmitida da origem para o destino de um fluxo, seja ele fluxo normal de navegação ou fluxo de dados. Um vínculo de parâmetro tem um *nome* e um *valor*, o qual é obtido no componente de origem.

Um fluxo pode ter vários vínculos de parâmetro, cada um deles com um nome e um valor.

Usualmente, chaves ou valores de atributos de objetos são passados por vínculos de parâmetros. Por *default*, um fluxo de um componente de visão vincula a chave primária do objeto selecionado no componente de origem e a envia ao destino, onde ela pode ser usada, por exemplo, por uma expressão condicional.

Nos exemplos anteriores, apenas constantes foram usadas em expressões condicionais para selecionar os elementos a serem mostrados nos componentes de visão tais como listas e detalhes. Agora, com fluxos e vínculos de parâmetro, é possível usar um componente de visão para definir quais elementos são mostrados em outro componente de visão.

O diagrama IFML da Figura 4 .17 mostra uma lista de livros com um fluxo para um componente de visão detalhes. O componente detalhes tem uma condição-chave [*oid*=?]. Como a chave para o livro selecionado na lista é vinculado ao fluxo que vai para o componente detalhes, apenas aquele livro será mostrado quando o componente detalhes for renderizado.

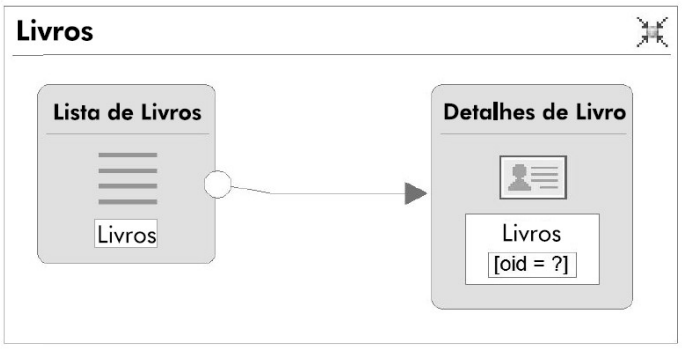


Figura 4.17. Um uxo de navegação normal conectando dois componentes de visão.

[7] E *operações,* como explicado adiante.

Quando o usuário seleciona um livro na lista, ao clicar na respectiva ligação, o componente detalhes apresenta informações sobre o livro. A Figura 4 .18 mostra o estado da página quando a ligação associada ao primeiro livro é clicada.

O vínculo de parâmetro também pode ser usado para associar um formulário a outro componente de visão. Por exemplo, o formulário pode fornecer um campo de pesquisa e uma lista poderia apresentar apenas os elementos que satisfazem o critério de seleção.

A Figura 4 .19 apresenta um exemplo no qual um campo de pesquisa é usado para encontrar os livros de um dado autor. O usuário pode digitar o nome do autor ou parte dele no campo de pesquisa e os resultados são mostrados na lista *Livros*.

O paralelogramo que está próximo ao fluxo indica que este fluxo possui um vínculo de parâmetro que não é o *default*, ou seja, a chave de um objeto selecionado no componente de visão da origem. Embora a ferramenta não mostre o texto associado ao vínculo, ele poderia ser consultado clicando-se no paralelogramo e, possivelmente, seria algo como "*campoX*→*texto*" indicando que o valor contido no campo *campoX* do componente da origem é passado como o parâmetro cujo nome é *texto* pelo fluxo e pode ser usado no componente de destino em uma expressão condicional como [*nomeDoAutor* Contains *texto*], onde *Contains* é um predicado de comparação de cadeias de caracteres que retorna verdadeiro se e somente se a cadeia à esquerda contém integralmente a cadeia à direita.

Na renderização mostrada na Figura 4 .20, o usuário digitou "Clarke" no campo de pesquisa e os resultados produzidos são mostrados na Figura 4 .21.

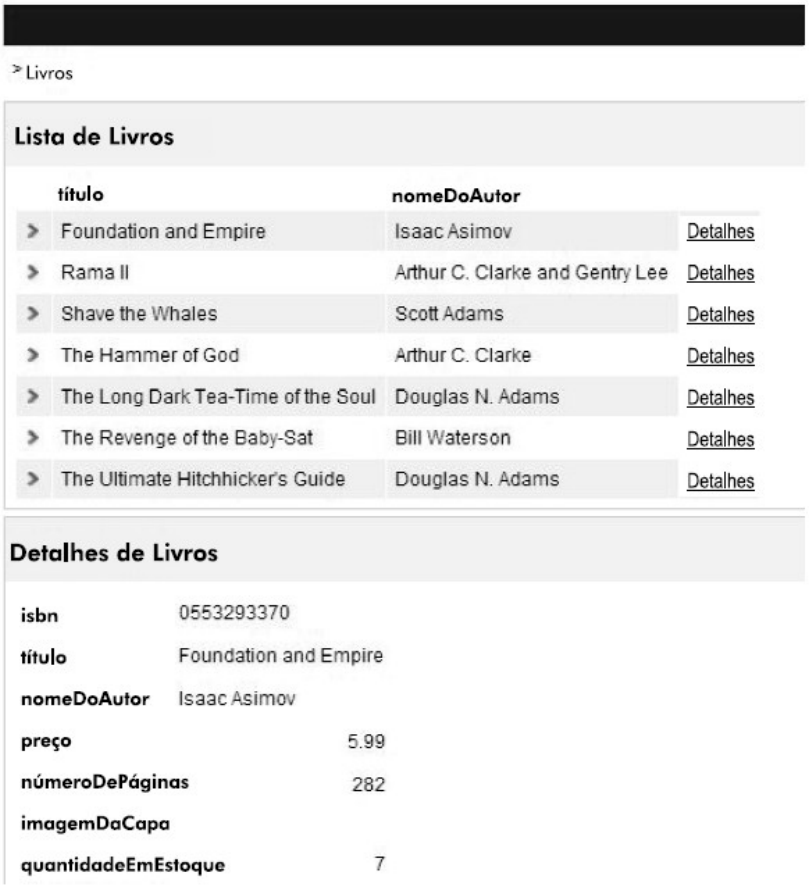


Figura 4.18. Renderização do modelo da Figura 4 .17 quando o primeiro livro da lista é selecionado.

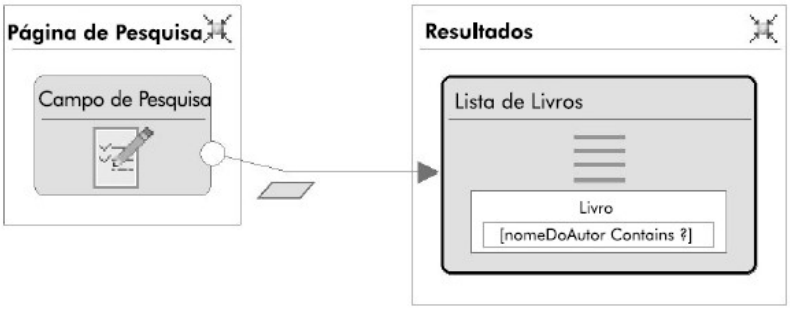


Figura 4.19. Um uxo que transporta dados de um formulário para uma lista.

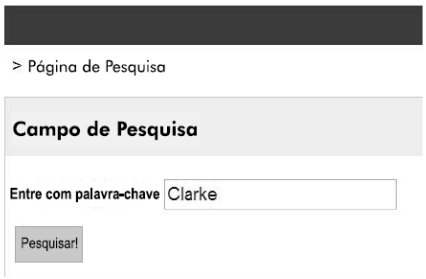


Figura 4.20. Renderização do modelo da Figura 4 .19 antes de iniciar a pesquisa.

Figura 4.21. Renderização do modelo da Figura 4 .19 após a pesquisa ser realizada.

4.5.4. Vínculo de Parâmetro Multivalorado

É possível definir fluxos a partir de componentes de escolha múltipla tais como listas múltiplas. Nesses casos, um vínculo de parâmetro passa um conjunto de valores (as chaves dos elementos selecionados, por exemplo).

Se uma lista múltipla é ligada a um componente detalhes múltiplos como na Figura 4 . 22, o usuário pode selecionar vários elementos da lista múltipla como na Figura 4 .23 e pressionar o botão "Ver detalhes" para navegar para a página que mostra os detalhes apenas dos livros selecionados (Figura 4 .24). O componente detalhes múltiplos tem uma condição-chave que estabelece que apenas livros cujo OID pertença ao conjunto que é vinculado ao fluxo são mostrados.

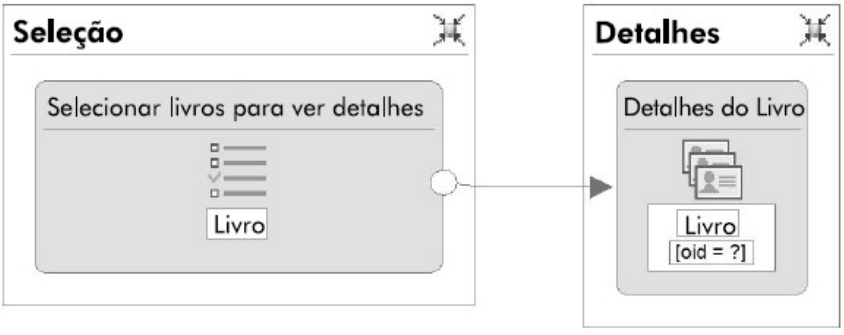


Figura 4.22. Um uxo para vinculação de parâmetro multivalorada.

Figura 4.23. Renderização do modelo da Figura 4 .22 antes de clicar o botão.

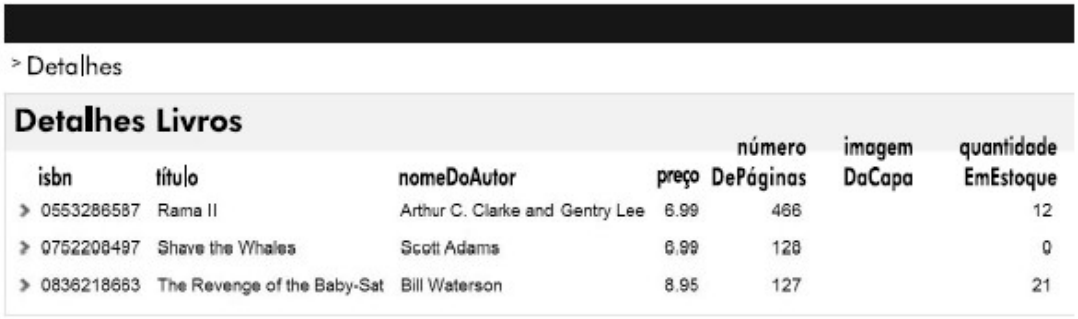


Figura 4.24. Renderização do modelo da Figura 4 .22 após clicar o botão.

4.6. Padrões de Interface Web

Esta seção apresenta alguns padrões de interface considerados bastante comuns em aplicações Web. A maioria dos padrões de interface apresentados nesta seção é adaptada de Ceri *et al*. (2003) e Tidwell (2005). São eles: "índice em cascata", "índice filtrado", "tour guiado", "pontos de vista", "visão geral mais detalhes" e "navegação em alto nível".

Não é interesse desta seção esgotar os possíveis padrões, visto que muitos outros existem e ainda podem vir a ser criados, mas mostrar ao leitor a viabilidade de modelar em IFML alguns dos padrões de interface mais comuns, bem como chamar a atenção para a importância de definir padrões de interface de forma a facilitar e padronizar a construção de aplicações.

4.6.1. Índice em Cascata

Um *índice em cascata* é uma sequência de menus baseados em listas que no final atingem um componente detalhes. Por exemplo, um usuário pode iniciar selecionando

uma editora; então ele pode selecionar um livro a partir de uma lista da editora e, finalmente, acessar os detalhes de um livro. A Figura 4 .25 ilustra este padrão.

Figura 4.25. Exemplo de um índice em cascata.

No diagrama da Figura 4 .25, como os vínculos de parâmetro são *default*, os fluxos carregam os OIDs dos elementos selecionados. Estes OIDs podem ser usados pelas expressões condicionais no componente de visão destino para definir quais elementos devem ser mostrados.

O componente de visão *Lista de Livros* usa uma expressão condicional baseada em papel que mostra apenas os livros ligados à editora cujo OID é vinculado ao fluxo de entrada que vem de *Lista de Editoras*. A expressão condicional em *Lista de Livros* é baseada no papel *EditoraParaLivro*, ou seja, apenas livros ligados à editora na origem do fluxo são mostrados por *Lista de Livros.*

O componente de visão *Livro* mostra a informação completa sobre o livro selecionado em *Lista de Livros*. Os OIDs dos livros selecionados são vinculados ao fluxo que vem de *Lista de Livros* para o componente detalhes. A Figura 4 .26 mostra uma sequência de interfaces renderizadas que ilustram este padrão. Em (a), uma editora pode ser selecionada. Em (b), após selecionar a segunda editora, um de seus livros pode ser selecionado. Finalmente, em (c), o terceiro livro foi selecionado e seus detalhes são mostrados.

Uma variação deste padrão consiste em mostrar alguns dados do objeto selecionado em uma das listas intermediárias. A Figura 4 .27 ilustra essa situação. Nela, quando uma editora é selecionada, a interface mostra os detalhes da editora e uma lista de livros para seleção. Note que na página intermediária, *Detalhes da Editora* está conectado à *Lista de Livros* por um fluxo de dados. Não é necessário que o usuário navegue de *Detalhes da Editora* para sua lista de livros: ambos os componentes de visão são mostrados ao mesmo tempo quando a página *Selecionar Livro* for acessada. A Figura 4 .28 mostra como a página intermediária seria renderizada.

Figura 4.26. Renderização de *Lista de Editoras*, *Lista de Livros* e *Livro*.

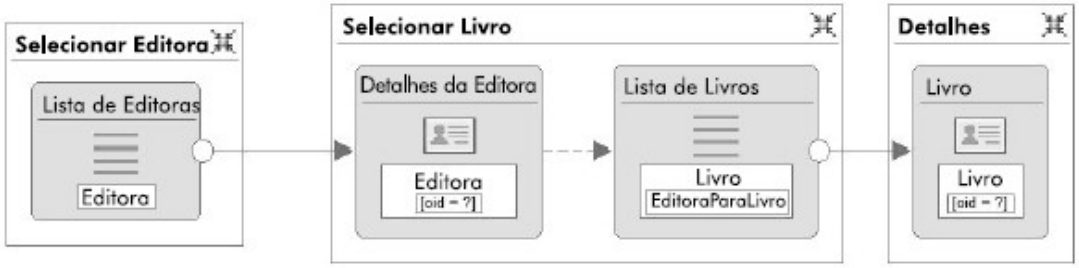


Figura 4.27. Índice cascata com apresentação de dados intermediários.

Figura 4.28. Renderização da página intermediária de um índice cascata com dados intermediários.

4.6.2. Índice Filtrado

Em alguns casos, pode ser necessário mostrar uma lista com elementos selecionados. Por exemplo, se a livraria tiver milhares de livros em seu catálogo, não seria viável para um usuário simplesmente procurar por um livro em uma lista. Uma lista filtrada poderia ser usada no lugar disso, por exemplo, pedindo ao usuário para fornecer uma palavra-chave de forma a apresentar uma lista de livros reduzida.

Este padrão é realizado por uma conexão entre um formulário, uma lista e um componente de visão detalhes, como mostrado na Figura 4 .29.

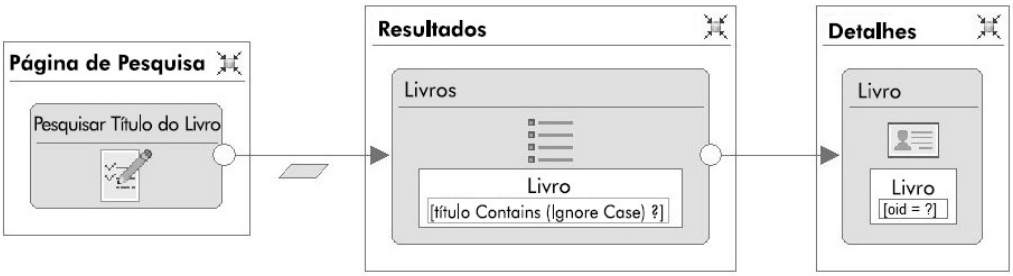


Figura 4.29. Um exemplo de um índice Itrado.

Como os resultados são mostrados em uma lista (não uma lista *simples*), um fator de bloco de *scroll* pode ser definido, bem como a possibilidade de dinamicamente ordenar os resultados. Além disso, o número máximo de resultados pode ser definido para esta lista de forma que listas desnecessariamente grandes não são apresentadas.

4.6.3. *Tour* Guiado

Um *tour guiado* é um padrão que permite que se visualize os detalhes de um conjunto de objetos, um de cada vez, usando operações de *scroll*. Ele é implementado pelo uso do componente *scroller* com *fator de bloco* 1 ligado a um componente detalhes, conforme mostra o lado esquerdo da Figura 4 .30.

O lado direito da Figura 4 .30 mostra uma renderização desta especificação na qual o quinto de oito livros foi selecionado pelo usuário.

4.6.4. Pontos de Vista

Algumas vezes, pode ser interessante apresentar diferentes faces de certos objetos em diferentes momentos. Por exemplo, dados resumidos sobre um livro podem ser apresentados por *default*, mas esses dados podem ser expandidos ou contraídos novamente se o usuário desejar.

Para implementar o padrão de pontos de vista, dois ou mais componentes detalhes podem ser definidos para a mesma classe, com diferentes conjuntos de atributos. Fluxos podem ser definidos entre os diferentes componentes para permitir ao usuário mudar de uma visão para outra, como mostra a Figura 4 .31.

O lado esquerdo da Figura 4 .32 mostra a aparência do ponto de vista resumido e o lado direito mostra o ponto de vista da informação completa.

Figura 4.30. Especi cação de um *tour* guiado e sua renderização.

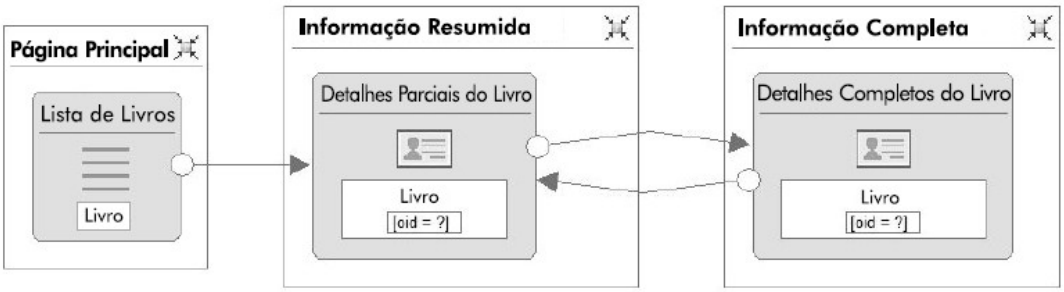


Figura 4.31. Especi cação de pontos de vista.

Figura 4.32. Dois pontos de vista para um mesmo objeto.

4.6.5. Visão Geral mais Detalhe

O padrão *Visão geral mais detalhe* é muito comum em aplicações tais como visualizadores de e-mail, mas ele pode ser aplicado a uma variedade de outras situações. A ideia é ter na mesma página uma lista com *scroll* de elementos e um componente detalhes de um elemento selecionado em uma região próxima da lista.

A Figura 4 .33 mostra uma definição IFML que usa este padrão com uma lista com fator de bloco 3 e um componente detalhes ligado a ela por um fluxo. A Figura 4 .34 mostra uma renderização para a página especificada.

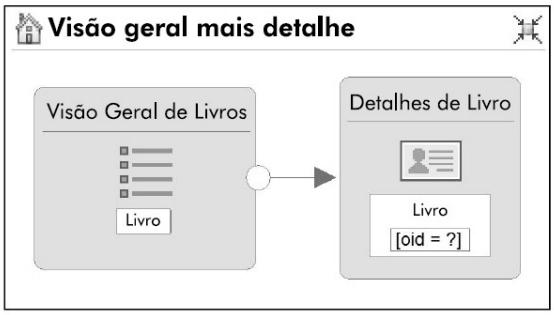


Figura 4.33. Especi cação do padrão *visão geral mais detalhe*.

Figura 4.34. Exemplo de renderização de uma página usando o padrão *visão geral mais detalhe*.

4.6.6. Navegação em Alto Nível

Navegação em alto nível é um padrão frequente em páginas Web nas quais um menu de alto nível permite acesso a diferentes áreas do site. O exemplo na Figura 4 .35 ilustra navegação em alto nível sendo definida por um conjunto de áreas *landmark*, cada uma contendo uma página (elas poderiam conter mais páginas se necessário).

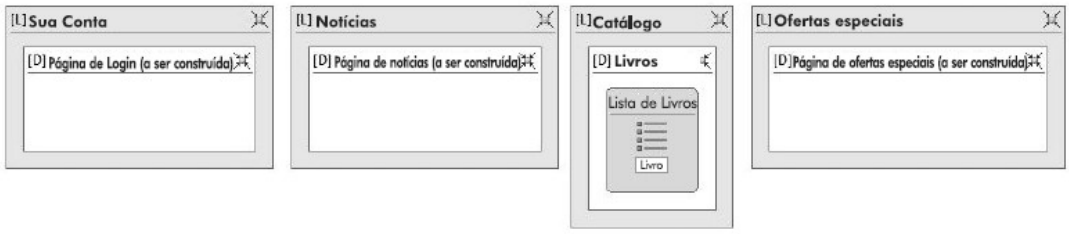


Figura 4.35. Exemplo de especi cação de navegação em alto nível.

Para o exemplo, apenas a terceira área foi parcialmente definida. Uma renderização deste modelo é mostrada na Figura 4 .36, em que a terceira área, "Catálogo", é selecionada do menu de alto nível mostrando a página *default Livros*.

Figura 4.36. Exemplo de renderização de uma página usando navegação de alto nível.

Uma variação desse padrão é a navegação de alto nível baseada em menus. Isso pode ser conseguido pela adição de um conjunto de páginas *landmark* a cada uma das áreas *landmark* de alto nível, como mostrado na Figura 4 .37.

A Figura 4 .38 mostra a renderização quando o mouse é posicionado sobre o item de menu "Catálogo".

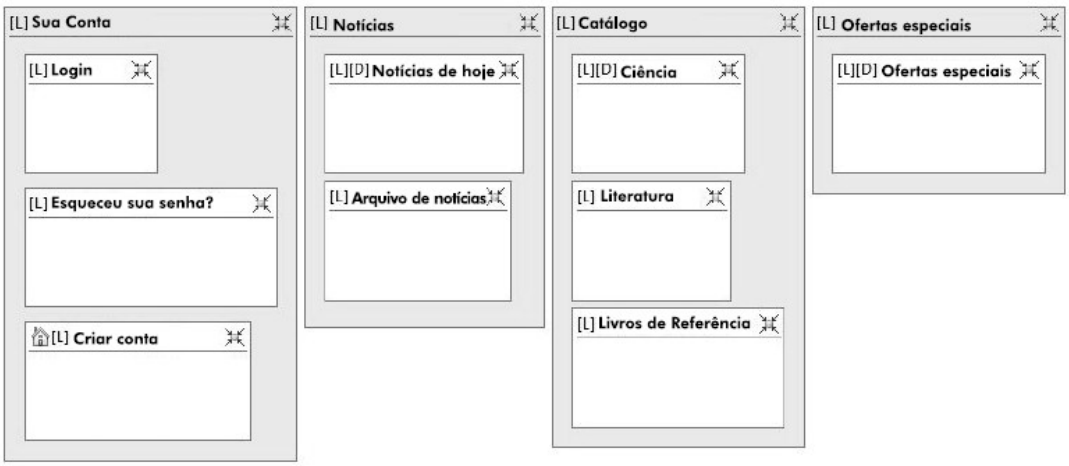


Figura 4.37. Exemplo de especi cação de uma navegação baseada em menu de alto nível.

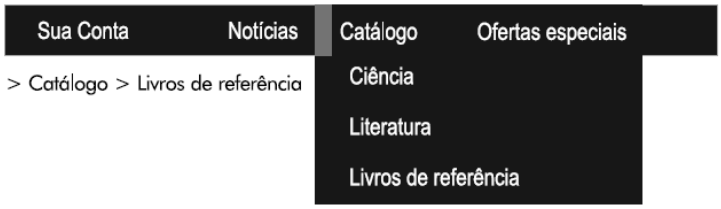


Figura 4.38. Exemplo de renderização de uma página usando navegação baseada em menu de alto nível.

4.7. Modelagem de Operações na Interface

IFML permite a modelagem de operações básicas com o uso de *componentes de operação* que permitem a criação, destruição, atualização, conexão, desconexão e reconexão[8] de instâncias. Esses componentes realizam as operações mais básicas que podem ser efetuadas sobre objetos. A Figura 4 .39 mostra a representação gráfica dessas operações.

Figura 4.39. Componentes de operação: *criar*, *deletar*, *atualizar*, *conectar*, *desconectar* e *reconectar*.

Os componentes de operação não contêm nem renderizam informação: eles apenas a *processam*. Portanto, eles são representados graficamente fora das páginas.

Cada componente de operação deve ter pelo menos um fluxo de entrada. Algumas vezes, toda a informação que a operação precisa é obtida de um único componente de visão, mas, algumas vezes, mais do que um componente de visão pode ser necessário para fornecer todos os dados necessários para uma operação.

[8] Substituição de uma ligação.

Componentes de operação têm fluxos de saída especiais que podem ser de dois tipos:

- *Fluxo OK*, que define para onde vai o foco quando a operação foi executada com sucesso sobre todos os objetos.
- *Fluxo KO*, que define para onde vai o foco quando a operação falha para pelo menos um dos objetos afetados por ela.

Os componentes de operação são baseados em classes ou associações, as quais são suas fontes de dados. As operações *criar*, *deletar* e *atualizar* são definidas sobre uma única classe, enquanto *conectar, desconectar e reconectar* são definidas sobre uma associação e, portanto, têm uma classe *fonte* e uma classe *alvo*.

Por exemplo, uma operação *deletar* poderia ser definida sobre a classe *Livro* como na Figura 4 .39; esta operação seria responsável pela destruição de instâncias de *Livro*.

Uma operação *conectar,* no entanto, poderia ser definida sobre uma associação tal como *LivroParaEditora*, como mostrado na Figura 4 .39. Portanto, ela seria responsável por conectar (adicionar ligações entre) instâncias da classe *Livro* (fonte) e instâncias da classe *Editora* (alvo).

4.7.1. Operação *Criar*

A operação *criar* cria novas instâncias de uma dada classe. Um fluxo de navegação ou de dados pode ser usado para fornecer dados de inicialização para a nova instância. Usualmente, dados de inicialização são obtidos de um formulário baseado na mesma classe. Um fluxo do formulário para a operação *criar* passa os valores necessários para todos os atributos, exceto pelo OID, o qual é gerado automaticamente pela operação *criar*. Por exemplo, a Figura 4 .40 mostra uma estrutura para criar instâncias de *Livro*.

Como podemos ver, o usuário pode digitar no formulário valores de inicialização para um novo livro. O fluxo do formulário vincula dados e leva esses dados para a operação de criação. A operação de criação então cria a nova instância e define seus atributos com os valores recebidos do fluxo.

A Figura 4 .41 mostra a renderização do formulário que está na origem do fluxo para a operação *criar*. O fluxo é renderizado como o botão *Salvar*.

Se a operação tiver sucesso, então o fluxo OK é seguido para um componente detalhes que mostra o livro que acaba de ser criado. O fluxo OK vincula automaticamente o OID do novo livro de forma que a expressão condicional no componente detalhes pode ser usada para definir qual livro apresentar.

Se a operação falhar, o controle vai para uma página de erro, seguindo o fluxo KO.

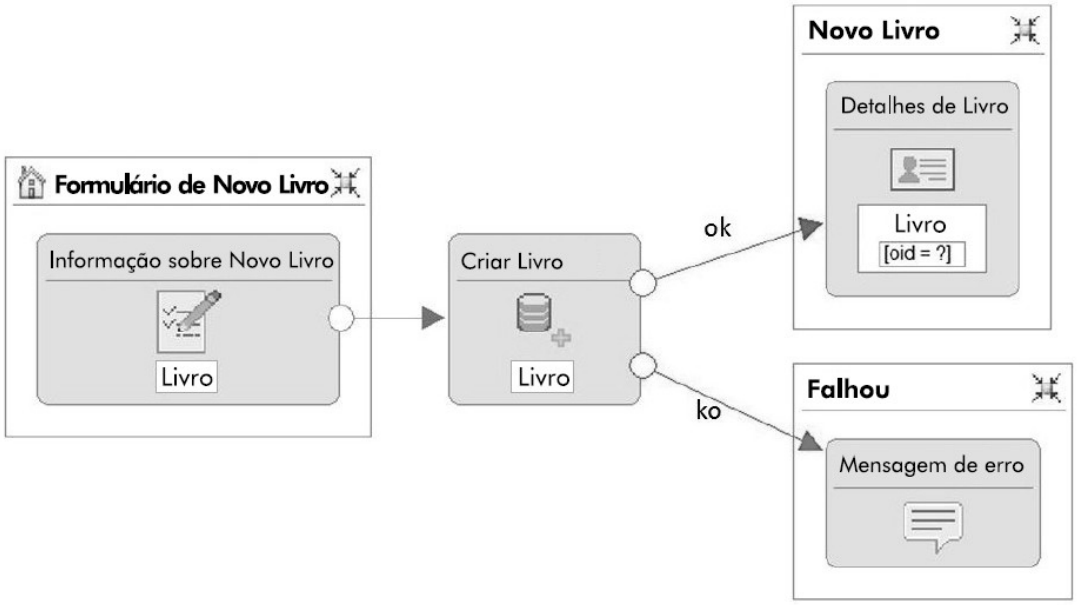


Figura 4.40. Exemplo de aplicação da operação *criar*.

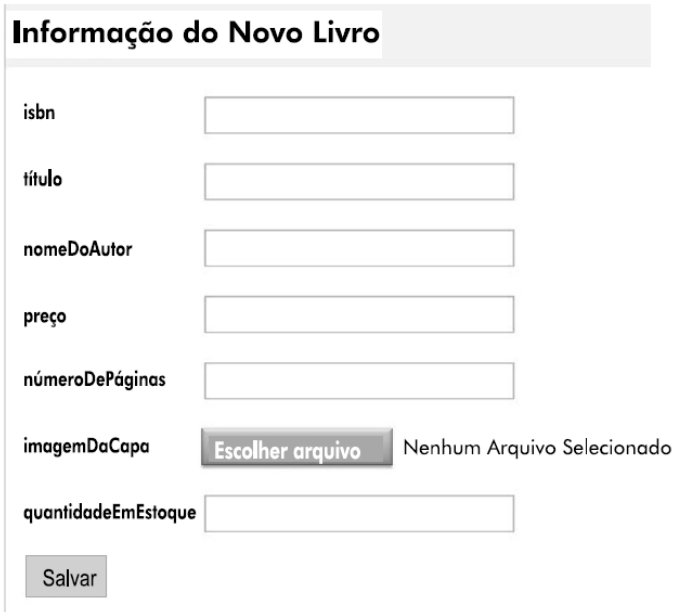


Figura 4.41. Renderização de um formulário com um uxo para uma operação *criar*.

4.7.2. Operação *Deletar*

A *operação deletar* pode ser realizada sobre um ou mais objetos que satisfazem a expressão condicional que a define.

A operação *deletar* pode ser usada em conjunto com uma lista (qualquer tipo de lista), na qual o usuário pode selecionar os objetos a serem deletados. A Figura 4 .42 mostra um exemplo no qual o usuário seleciona um livro para deletar de uma lista simples. Note que o seletor da operação deletar é baseado no OID do livro, o qual é recebido como um parâmetro do fluxo de entrada.

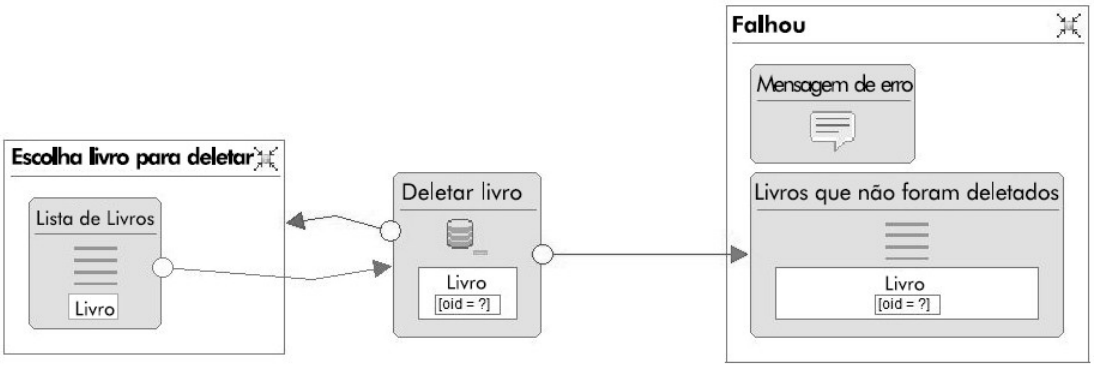


Figura 4.42. Uma operação *deletar* sendo ativada a partir de uma lista.

Esta página pode ser renderizada como na Figura 4 .43, na qual a palavra "deletar" corresponde ao nome do fluxo.

> Escolha livro para deletar

Lista de Livros

	título	nomeDoAutor	
>	Dirk Getly's Holistic Detective Agency	Douglas N. Adams	deletar
>	Foundation and Empire	Isaac Asimov	deletar
>	Rama II	Arthur C. Clarke and Gentry Lee	deletar
>	Shave the Whales	Scott Adams	deletar
>	The Hammer of God	Arthur C. Clarke	deletar
>	The Long Dark Tea-Time of the Soul	Douglas N. Adams	deletar
>	The Revenge of the Baby-Sat	Bill Waterson	deletar
>	The Ultimate Hitchhicker's Guide	Douglas N. Adams	deletar

Figura 4.43. Renderização de uma lista simples associada a uma operação *deletar* por um uxo chamado "deletar".

Quando todos os objetos que se qualificam para remoção são realmente removidos, o fluxo OK é seguido. Quando pelo menos um objeto não puder ser deletado, o fluxo KO é seguido e passa como parâmetro vinculado os OIDs dos objetos que não foram deletados. Assim, o fluxo KO poderia, por exemplo, levar a uma lista ou um componente detalhes múltiplos no qual os objetos que não puderam ser removidos são mostrados, como visto na Figura 4 .42.

4.7.3. Operação *Atualizar*

Uma operação *atualizar* contém um seletor para um ou mais objetos da mesma classe e um conjunto de atribuições para atualizar os atributos destes objetos. Usualmente, novos valores para atributos são recebidos por fluxos de navegação normais ou fluxos de dados.

A Figura 4 .44 mostra como usar uma operação *atualizar* para atualizar o preço de um livro existente.

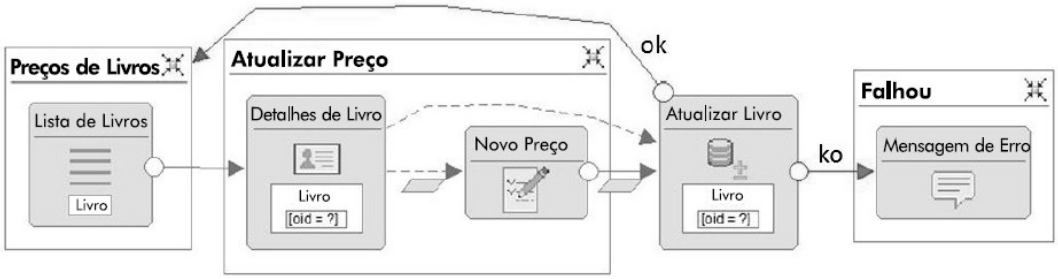


Figura 4.44. Uma operação *atualizar* sendo usada para trocar o preço de um livro.

Na página *Preços de Livros,* um livro é selecionado de uma lista (Figura 4 .45). Então uma nova página (*Atualizar Preço*) mostra detalhes do livro e um campo pré-carregado com o preço antigo. O usuário pode mudar este valor e atualizá-lo pressionando "Salvar" (Figura 4 .46).

> Preços de Livros

Lista de Livros

título		preço	
The Ultimate Hitchhicker's Guide	Douglas N. Adams	129	atualizar preço
Dirk Getly's Holistic Detective Agency	Douglas N. Adams	6.99	atualizar preço
Shave the Whales	Scott Adams	6.99	atualizar preço
The Long Dark Tea-Time of the Soul	Douglas N. Adams	6.99	atualizar preço
Rama II	Arthur C. Clarke and Gentry Lee	6.99	atualizar preço
The Hammer of God	Arthur C. Clarke	6.99	atualizar preço
Foundation and Empire	Isaac Asimov	5.99	atualizar preço
The Revenge of the Baby-Sat	Bill Waterson	8.95	atualizar preço

Figura 4.45. Renderização da página *Preços de Livros*.

O antigo preço é carregado no campo *Novo Preço* por um fluxo de dados de *Detalhes de Livro*. A operação *atualizar* é ativada pelo fluxo de navegação que é renderizado como o botão *Salvar* na página *Atualizar Preço*. Este fluxo de navegação tem um parâmetro vinculado que passa o novo preço do livro do respectivo campo no formulário *Novo Preço* para a operação *atualizar*. Outros atributos do livro, incluindo seu OID, são recebidos por um fluxo de dados de *Detalhes de Livro.*

Na Figura 4 .45, o fluxo OK da operação *atualizar* leva de volta para a lista na página *Preços de Livros*, na qual o novo preço pode ser visualizado. O fluxo KO leva para uma janela de erro. Novamente, o fluxo OK é seguido quando todos os objetos qualificados pela expressão condicional são atualizados. Se pelo menos um objeto não puder ser atualizado, o fluxo KO é seguido com o conjunto de OIDs dos objetos que não puderam ser atualizados como parâmetros vinculados.

Figura 4.46. Renderização da página *atualizar preço*.

4.7.4. Operações *Conectar*, *Desconectar* e *Reconectar*

As operações *conectar*, *desconectar* e *reconectar* têm como fonte de dados uma *associação*. As operações *conectar* e *desconectar* têm duas expressões condicionais: uma para o papel de origem e outra para o papel-alvo da associação. A operação *reconectar* pode ser entendida como uma combinação das anteriores. Ela tem três expressões condicionais: uma para o papel de origem, outra para os objetos no papel-alvo que vão ser desconectados e uma terceira para os objetos no papel-alvo que vão ser conectados aos objetos da origem.

As operações *conectar*, *desconectar* e *reconectar* podem adicionar, remover e substituir ligações entre conjuntos de objetos. Se, por exemplo, cinco objetos satisfazem a expressão condicional da origem de um *conectar* e três objetos satisfazem a expressão condicional de destino, então uma operação de conexão iria criar 15 ligações: cada um dos cinco objetos na origem seria ligado a cada um dos três objetos no destino.

O exemplo da Figura 4 .47 mostra como uma ligação entre um livro e uma editora pode ser criada. Primeiramente, a ligação é obrigatória do livro para a editora. Assim, não é possível simplesmente pegar uma lista de livros e uma lista de editoras e escolher quais serão ligados. Uma ligação entre um livro e uma editora deve ser criada assim que o livro for criado. Além disso, essa ligação é imutável e não poderá mudar mais tarde.

Na primeira página, *Escolher Editora*, existe uma lista simples de editoras. O usuário deve escolher uma e ser levado para a *Página da Editora*. Lá, os detalhes da editora e uma lista de livros já ligados a ela são mostrados. Note que *Detalhes da Editora* tem uma condição-chave e um fluxo a partir de *Lista de Editoras* e, portanto, ela mostra a editora selecionada na lista. A lista *Livros da Editora* tem um fluxo de dados que vem de *Detalhes da Editora* e uma expressão condicional baseada em papel, de forma que apenas livros que são ligados à editora de *Detalhes da Editora* são mostrados. A Figura 4 .48 mostra a renderização de *Página da Editora* após a editora “Bantam” ser selecionada da lista.

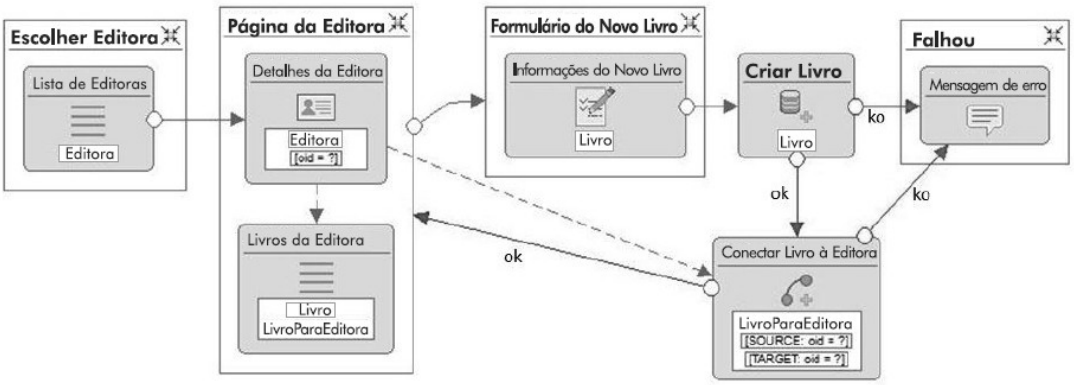


Figura 4.47. Modelo IFML mostrando uma interface na qual livros podem ser criados e ligados a editoras.

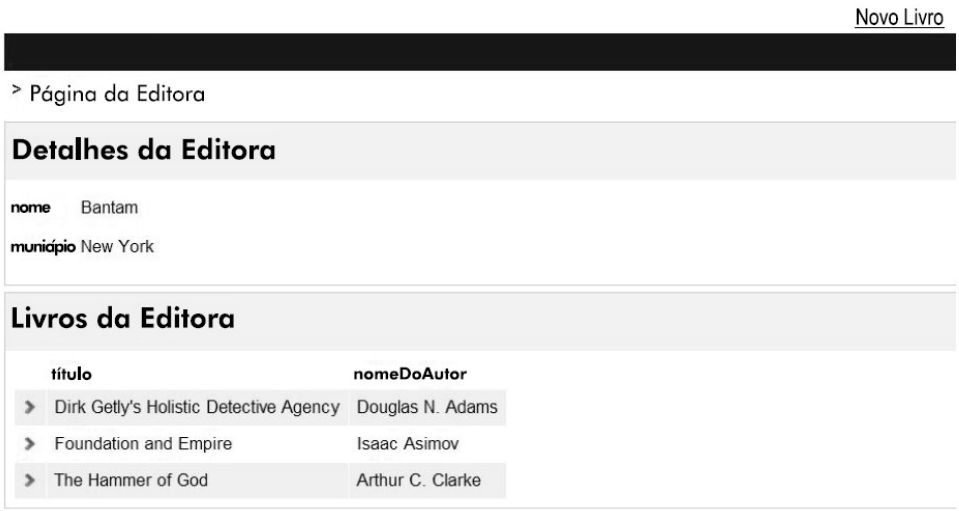


Figura 4.48. Renderização da *Página da Editora*.

A partir da *Página da Editora,* o usuário pode navegar para o *Formulário de Novo Livro* usando um fluxo de navegação normal entre ambas as páginas, o qual é renderizado como a ligação *Novo Livro*. Nessa página, o usuário pode preencher o formulário com informações sobre o novo livro, o qual será automaticamente ligado à editora apresentada na *Página da Editora*. Inicialmente, a operação *criar* cria uma nova instância de livro. Seu fluxo OK leva para uma operação *conectar* que é baseada na associação *LivroParaEditora*. O fluxo que sai da operação *criar* tem a chave do novo livro como parâmetro vinculado; esta chave é usada na expressão condicional do papel fonte. A editora é o papel-alvo e ela é obtida pelo fluxo de dados que vem de *Detalhes da Editora.*

Se a operação de conexão for um sucesso, ela retorna o foco para a *Página da Editora*, na qual o novo livro pode ser visto na lista de livros da editora corrente. Se a operação *criar* ou *conectar* falhar, então o foco vai para uma página com uma mensagem de erro.

Entretanto, o design da Figura 4 .47 não é seguro. Se um livro for criado e a operação de conexão falhar, este livro ficará inconsistente em relação à sua definição. Para solucionar este problema, IFML permite que uma *transação* seja definida como um grupo de operações. A Figura 4 .49 mostra um *grupo de operações* sendo definido para incluir tanto a operação *criar* quanto a operação *conectar.* Como o grupo de operações

pode ser definido como uma transação ([*T*]), então ou todas as operações dentro do grupo obtêm sucesso ou o grupo todo falha. No caso, um único fluxo KO saindo do grupo seria seguido.

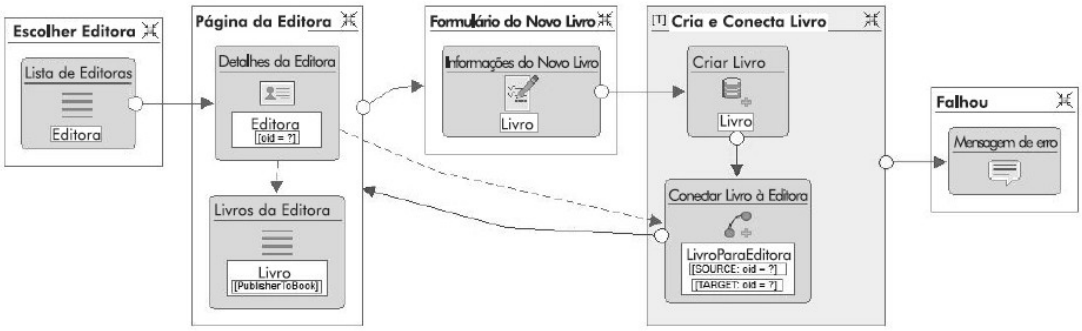


Figura 4.49. Um design mais seguro para o modelo da Figura 4 .47 usando um grupo de operações.

4.8. Modelo IFML para operações CRUD

Esta seção mostra passo a passo como implementar uma interface no estilo CRUD para gerenciar informação sobre objetos da classe *Editora*, usando componentes de visão e operações IFML.

A página principal do CRUD *Gerenciar Editoras* apresenta uma lista de editoras existentes. A criação de uma nova editora é iniciada por um clique em uma ligação na página principal que dá acesso a uma nova página com um formulário. Se a criação de uma nova editora tiver sucesso, o controle retorna para a página principal, e a nova editora aparece na lista. Se houver um erro, o controle vai para uma página de erro, a partir da qual é possível retornar à página principal (Figura 4 .50).

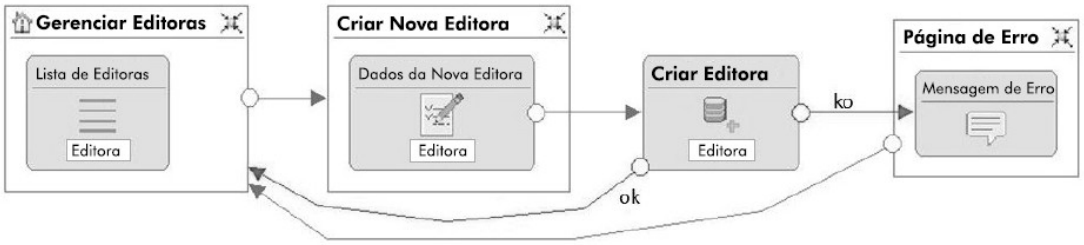


Figura 4.50. De nição de uma interface para *criar*.

A segunda operação de um CRUD, *consultar*, permite a visualização de todos os atributos de uma dada editora. Esta consulta pode ser modelada por um fluxo da lista principal levando para uma página com um componente detalhes, como mostra a Figura 4 .51.

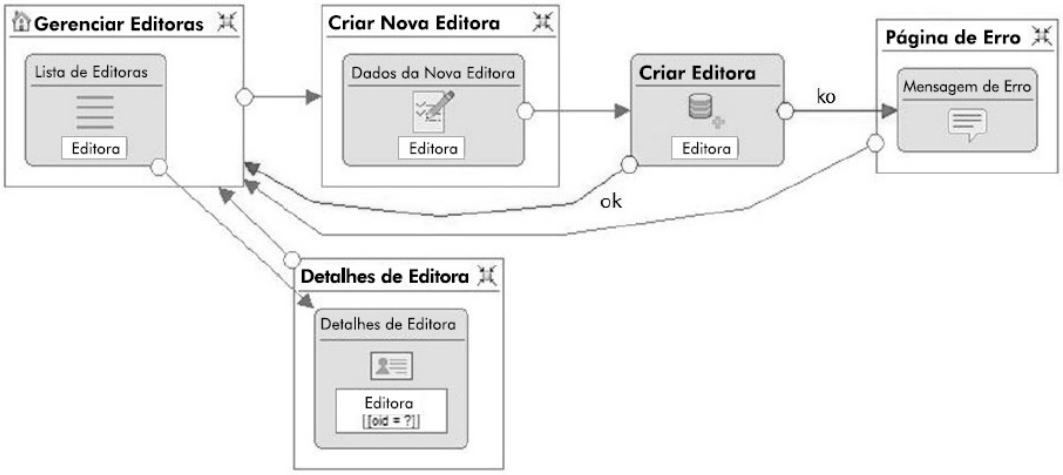


Figura 4.51. *Criar* e *consultar* de nidos.

A terceira operação, *atualizar*, permite que uma editora seja selecionada na lista principal e os atributos da editora selecionada são usados para preencher os campos de um formulário com *Editora* como classe fonte. Ali eles podem ser editados pelo usuário e salvos, conforme mostrado na Figura 4 .52.

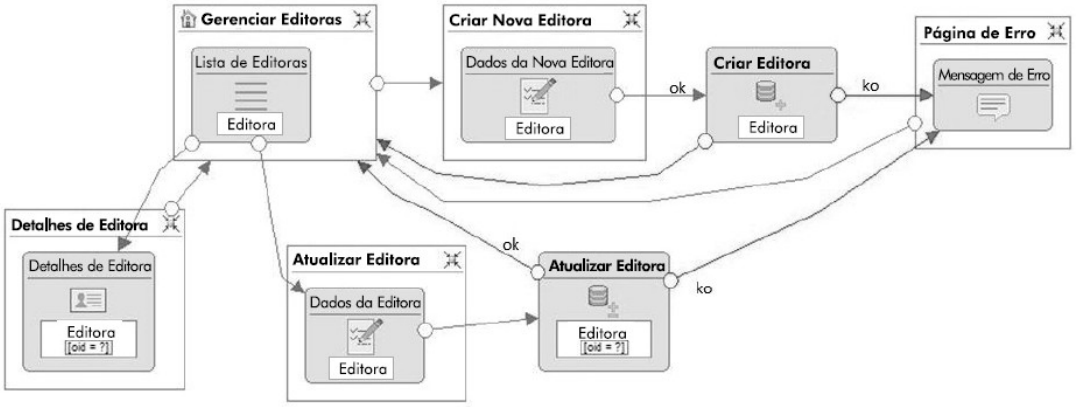


Figura 4.52. *Criar*, *consultar* e *atualizar* de nidos.

Finalmente, a operação para *deletar* uma editora é adicionada, como mostra a Figura 4 . 53. Uma operação de deleção pode ser acessada a partir da lista principal.

A Figura 4 .54 mostra a renderização da página principal do CRUD de *Editora* como definido na Figura 4 .53. No topo da página, é possível ver a ligação para a criação de uma nova editora, e logo a seguir está a lista principal, na qual as opções para consulta, atualização e deleção de cada editora estão disponíveis.

A Figura 4 .55 mostra a renderização para a página de consulta, a qual é acessada pela seleção da ligação *detalhes* para a terceira editora na lista da página principal. A ligação *retornar* no topo da página permite a um usuário voltar para a página principal.

A Figura 4 .56 apresenta a página de atualização da editora, a qual é acessada pelo clique na ligação *atualizar* na lista da página principal. Esta página é idêntica à página para criação de uma nova editora, exceto pelo fato de que ela pré-carrega dados em vez de estar em branco e que, como o atributo *nome* é imutável, seu respectivo campo é definido como não modificável.

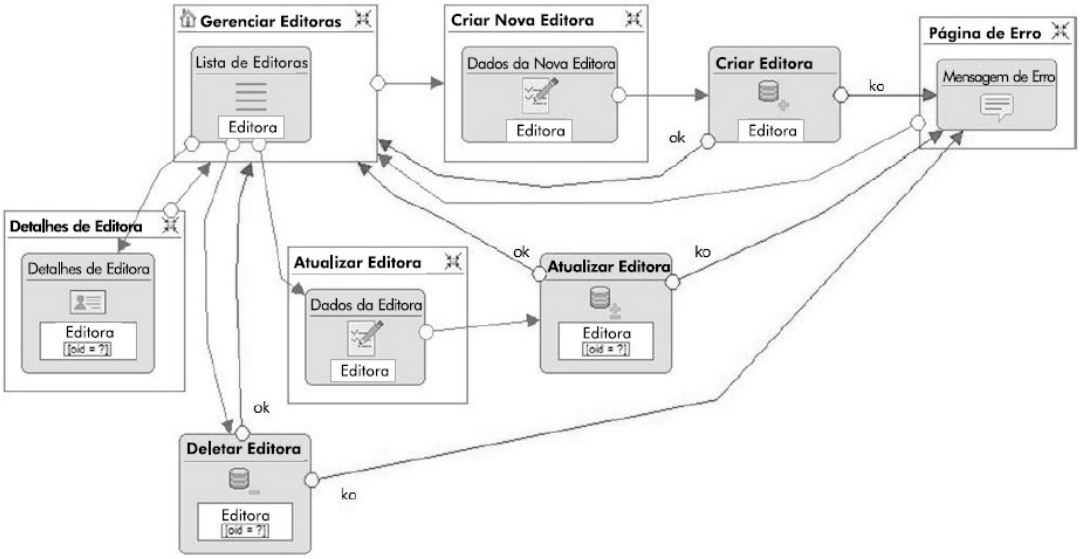


Figura 4.53. Especi cação completa de uma interface CRUD.

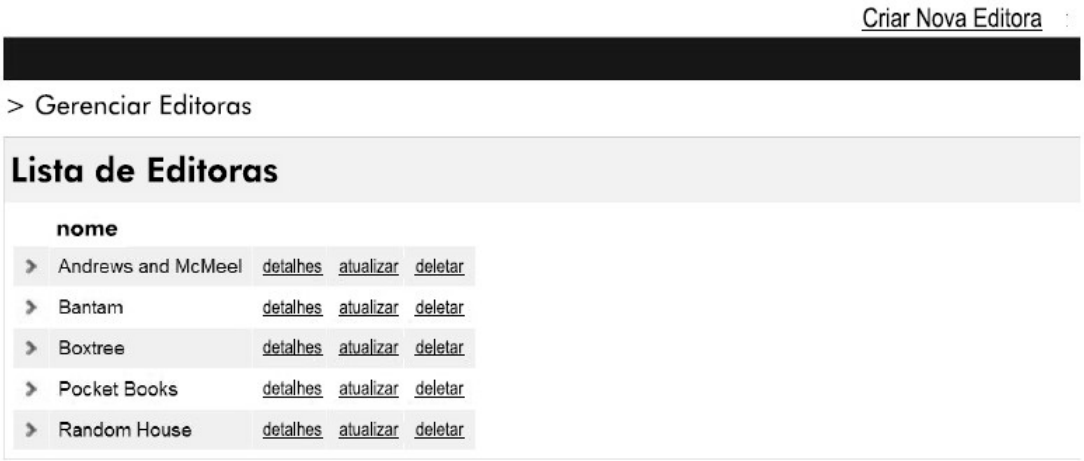


Figura 4.54. Janela principal para o CRUD de *Editora*.

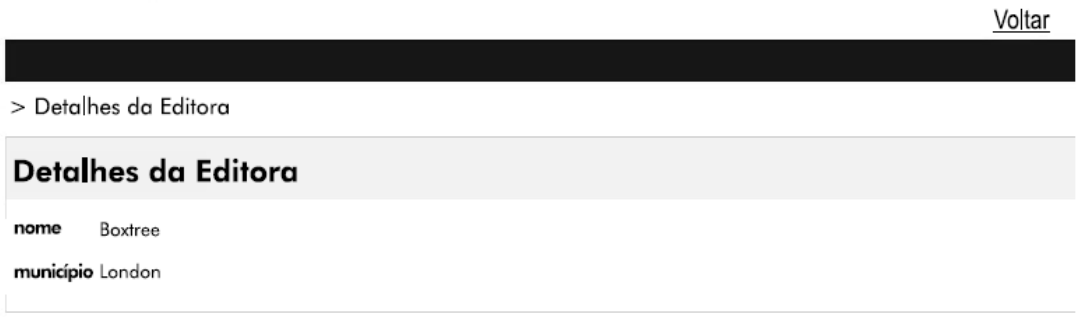


Figura 4.55. Renderização para a página de consulta de editora.

Figura 4.56. Renderização para a página de atualização de editora.

4.9. Modelagem de interfaces de casos de uso com IFML

Conforme visto na seção anterior, trabalhar com as operações básicas neste nível pode rapidamente tornar o diagrama muito complexo para ser facilmente entendido. Uma opção para lidar com essa complexidade é separar o design da interface em diferentes áreas e lidar com uma de cada vez.

Isso será mais eficaz se os *casos de uso de sistema* (Cockburn, 2001) já tiverem sido explorados e a equipe já tiver um entendimento sobre a estrutura do sistema e necessidades do usuário, o que permite que ela faça um design aceitável para a interface.

O design da interface com usuário pode iniciar com um diagrama de caso de uso de sistema como o que é mostrado na Figura 4 .57 que corresponde a um pequeno fragmento do diagrama completo, mas que será suficiente aqui para explorar essa questão. O estereótipo <<*report*>> indica que se trata de um caso de uso muito simples que apenas retorna dados aplicando derivações como totalização, ordenação ou filtragem, ou seja, é um caso de uso típico de sistemas de informação usualmente conhecido como *relatório*. Já o estereótipo <<*crud*>> indica que se trata de um caso de uso padrão também típico, no qual uma entidade simples pode ser submetida a quatro operações: criar, alterar, consultar e deletar.

Baseando-se neste diagrama de caso de uso, um possível design de interface que poderia ser preparado teria três áreas principais:

- *Carrinho de compras*: pedidos e pagamento.
- *Minha conta*: *login* e dados pessoais.
- *Relatórios*.

Este é ainda um design bastante cru, no sentido de que nenhuma questão de usabilidade foi considerada para decidir sobre a melhor organização possível. Entretanto, ele é efetivo. A Figura 4 .58 mostra o design IFML dessas áreas.

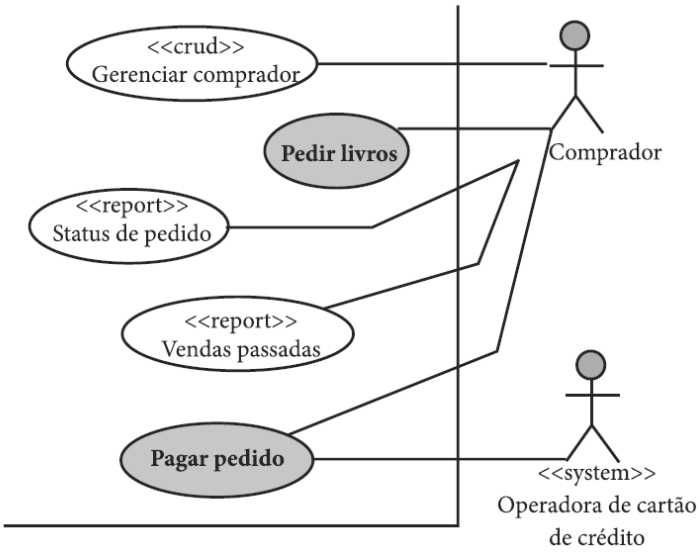


Figura 4.57. Diagrama de casos de uso parcial com casos de uso associados a um ator *Comprador*.

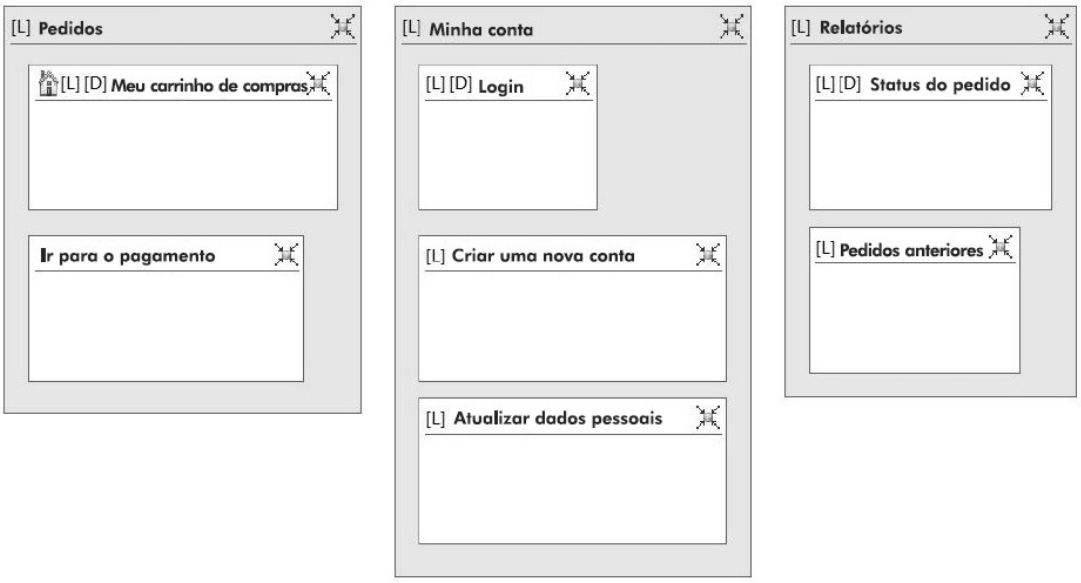


Figura 4.58. Design inicial de alto nível para uma visão de site.

Agora vamos olhar para o caso de uso *Pedir livros* que é mostrado na Figura 4 .59.

Caso de uso: *Pedir livros*

1. O comprador fornece palavras-chave para pesquisar livros.

2. O sistema gera uma lista de livros para venda que satisfaz as palavras-chave incluindo pelo menos título, autor, preço, número de páginas, editora, ISBN e imagem de capa.
3. O comprador seleciona livros da lista e indica a quantidade desejada para cada um.
4. O sistema gera um resumo do pedido (título, autor, quantidade, preço unitário e subtotal para cada livro) e o valor total.
5. O comprador finaliza o pedido.

Exceção 5a: O comprador ainda não se identificou.
5a.1. O comprador fornece uma identificação válida.
5a.2. Retorna ao passo 5.

Variante 5b: O usuário deseja pesquisar mais livros.
5d.1. Retorna ao passo 1.

Figura 4.59. Um caso de uso de sistema.

Para preparar um design de interface baseado nesse caso de uso de sistema, a equipe deve identificar as entradas e saídas de dados (comandos e consultas) necessárias para um ator poder executar o caso de uso. Tanto os comandos quanto as consultas poderão ter parâmetros repassados pelo ator através da interface. As consultas, porém, também irão retornar informações que precisam ser exibidas na interface. Assim, via de regra:

- Operações de sistema, sejam comandos ou consultas, devem ser ativadas em algum lugar da interface. A ativação pode ser obtida pelo clique em um botão ou ligação ou pela seleção de um item em uma lista, entre outras opções. Algumas vezes, o mesmo evento pode ativar mais do que uma operação de sistema: se duas ou mais operações de sistema acontecem imediatamente após um evento de sistema, então elas, possivelmente, são ativadas pela mesma ação do usuário.

- Valores para os parâmetros das operações de sistema precisam ser obtidos de algum lugar. Alguns parâmetros poderiam ser obtidos de outras fontes (tais como um relógio, por exemplo), mas a maioria dos parâmetros seria obtido na interface. Assim, para a maioria dos parâmetros, haverá um componente de visão na interface (usualmente um campo de formulário) que permite que o usuário introduza os dados necessários.

- Resultados obtidos por consultas de sistema devem ser apresentados. Alguns resultados podem estar escondidos do usuário, em alguns casos, mas, usualmente, eles são apresentados. Componentes de visão, tais como listas, detalhes e outros, podem ser usados para mostrar os resultados de consultas de sistema.

Para proceder com o design de interface, as recomendações anteriores são seguidas e os componentes de visão são criados para conter todos os parâmetros e resultados necessários. Ações de operações de sistema (representadas como hexágonos) são adicionadas ao diagrama também. A Figura 4 .60 apresenta a evolução do design de interface para a página *Meu carrinho de compras*, a qual implementa parte do caso de uso *Pedir livros*.

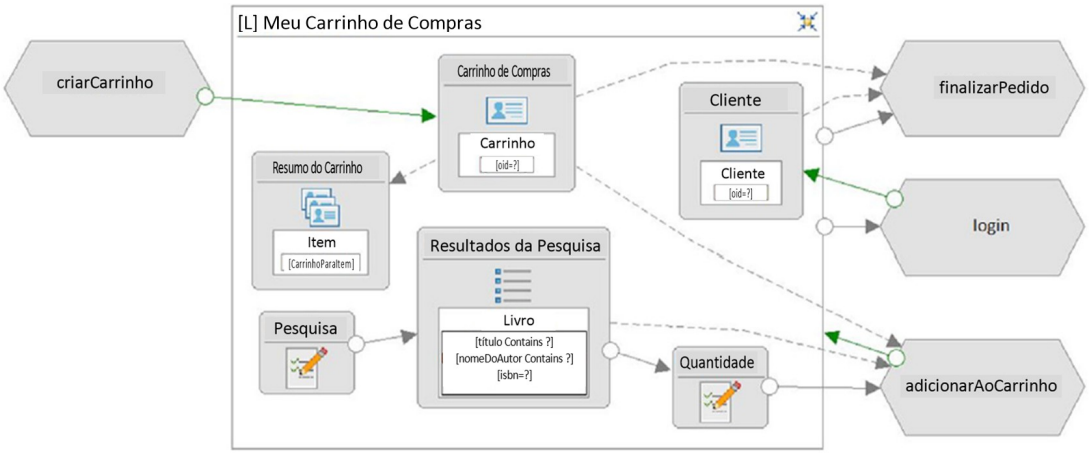


Figura 4.60. Re namento do design de interface para a página *Meu carrinho de compras*.

A operação de sistema *criarCarrinho* produz um novo carrinho que é apresentado pelo componente de visão detalhes *Carrinho de Compras*. O usuário pode usar o formulário *Pesquisa* para pesquisar por livros, os quais são apresentados na lista *Resultados de Pesquisa*. Lá, o usuário pode selecionar um livro e definir a quantidade desejada no formulário *Quantidade*. Isso ativa a operação de sistema *adicionarAoCarrinho*. Essa operação toma parâmetros de *Carrinho de Compras* (o *oid* do carrinho), *Resultados da Pesquisa* (o *isbn*) e *Quantidade* (a *quantidade*). Após um livro ser inserido no carrinho, ele é automaticamente mostrado no componente detalhes múltiplos *Resumo do Carrinho*.

Antes de finalizar o pedido, o comprador deve fazer *login*, caso ainda não o tenha feito. Finalmente, o pedido pode ser finalizado pela execução da operação de sistema *finalizarPedido*, a qual toma parâmetros de *Comprador* (o *cpf* do comprador) e *Carrinho de Compras* (o *oid* do carrinho).

Assim, em vez de usar operações básicas da IFML para definir operações de sistema, a equipe pode usar operações definidas externamente, como na Figura 4 .60. Nesse caso, a equipe pode implementá-los em sua linguagem de programação favorita. Isto é interessante por separar a lógica de transformação dos dados da lógica de interface. As operações definidas externamente podem realizar lógica complexa sobre os dados, mas apenas as entradas e saídas são associadas com os componentes de visão no modelo IFML.

4.10. Considerações Finais

Vimos aqui os conceitos mais básicos e o potencial de modelagem do novo padrão IFML que vem cobrir uma lacuna da linguagem UML por permitir a modelagem do fluxo de interação do usuário com um sistema. Conforme visto, a linguagem de modelagem é gráfica e independente de plataforma. Isto permite uma melhor organização do trabalho de desenvolvimento da parte *front-end* de aplicações computacionais, já que até agora não havia uma notação padronizada para definir tais características, as quais acabam sendo, na maioria das vezes, definidas de maneira informal e implementadas diretamente em código (Object Management Group, 2015).

IFML permite a especificação formal de diferentes perspectivas da aplicação, incluindo, conteúdo de interface, opções de navegação, quais operações são ativadas por quais eventos de interface e até aspectos de sua apresentação.

4.11. Referências

Brambilla, M., Fraternali, P. (2014). *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML*. Morgan Kaufman.

Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., & Matera, M. (2003). *Designing Data-Intensive Web Applications*. Morgan Kaufmann.

Chen, Peter (1976). The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems* **1**(1): 9–36.

Cockburn, A. (2001). *Writing Effective Use Cases*. Addison-Wesley.

Eckerson, W. W. (1995). Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications. *Open Information Systems* 10, January.

Fowler, M. (2003). *Patterns of Enterprise Application Architecture*. Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design Patterns: Elements of reusable object-oriented software. Addison-Wesley.

Ireland, C., Keynes, M. Bowers, D., Newton, M., Waugh, K. (2009). A Classification of Object-Relational Impedance Mismatch. *Advances in Databases, Knowledge, and Data Applications, DBKDA* '09. p. 36-43. Gosier.

Miles, R., & Hamilton, K. (2006). *Learning UML* 2.0. O'Reilly.

Object Management Group (2010). *OCL* 2.3.1 Specification. OMG.

Object Management Group (2015). *Interaction Flow Modeling Language – Version 1.0*. OMG. Disponível em: http://www.omg.org/spec/IFML/1.0/.

Seffah, A. (2015). *Patterns of HCI Design and HCI Design of Patterns: Bridging HCI Design and Model-Driven Software Engineering*. Springer-Verlag.

Tidwell, J. (2005). *Designing Interfaces: Patterns for effective interaction design*. O'Reilly Media.

Wazlawick, R. S. (2015). *Análise e Design Orientados a Objetos para Sistemas de Informação: Modelagem com UML, OCL e IFML*. 3ª edição. Elsevier. (1ª edição em 2004 com o título "Análise e Projeto de Sistemas de Informação Orientados a Objetos")