Pontifícia Universidade Católica
do Rio de Janeiro

Ezequiel Bertti

MIRA – Um ambiente para Interfaces dirigidas por modelos para aplicações REST

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa de Pós-Graduação em Informática da PUC-Rio.

Orientador: Prof. Daniel Schwabe

Rio de Janeiro

Março de 2015

Pontifícia Universidade Católica do Rio de Janeiro

Ezequiel Bertti

MIRA – Um ambiente para Interfaces dirigidas por modelos para aplicações REST

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa de Pós-Graduação em Informática da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Daniel Schwabe
Orientador
Departamento de Informática - PUC-Rio

Prof. Edward Hermann Haeusler
Departamento de Informática - PUC-Rio

Prof. Arndt Von Staa
Departamento de Informática - PUC-Rio

Prof. José Eugenio Leal
Coordenador Setorial do Centro
Técnico Científico - PUC-Rio

Rio de Janeiro, 02 de março de 2015

Ezequiel Bertti

Graduou-se em Sistemas de Informação na PUC-Rio (Pontifício Universidade Católica do Rio de Janeiro) em 2008.

Ficha Catalográfica

Bertti, Ezequiel

MIRA : um ambiente para interfaces dirigidas por modelos para aplicações REST / Ezequiel Bertti ; orientador: Daniel Schwabe. – 2015.

137 f. ; 30 cm

1. Dissertação (mestrado) – Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2015.

Inclui bibliografia

1. Informática – Teses. 2. Interfaces. 3. Adaptação. 4. Regras. 5. Desenvolvimento dirigido por modelos. 6. REST. I. Schwabe, Daniel. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

DEDICATÓRIA

A meus pais e irmãos, que sempre estiveram do meu lado acreditando e me dando forças para continuar e finalizar mais esta etapa.

AGRADECIMENTOS

Ao meu orientador Professor Daniel Schwabe pelo estímulo e parceria para a realização deste trabalho.

À PUC-Rio, pelos auxílios concedidos, sem os quais este trabalho não poderia ter sido realizado.

A W3C Brazil Office, NiC.br e Microsoft Brasil Open Source que apoiaram e financiaram este projeto.

Aos meus pais, pela educação, atenção e carinho de todas as horas.

Aos meus colegas da PUC-Rio.

Aos professores que participaram da Comissão examinadora.

A todos os professores e funcionários do Departamento pelos ensinamentos e pela ajuda.

A todos os amigos e familiares que de uma forma ou de outra me estimularam ou me ajudaram

Resumo

Bertti, Ezequiel; Schwabe, Daniel. **MIRA – Um ambiente para Interfaces dirigidas por modelos para aplicações REST**. Rio de Janeiro, 2015. 137p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Este trabalho apresenta um “framework” de software para construção de interfaces para aplicações REST, dirigido por modelos. Este “framework” permite a construção destas interfaces exigindo um mínimo de programação pelo projetista. Os modelos nos quais se baseia, e a interface gerada utilizam padrões do W3C. Uma avaliação qualitativa indica que há um aumento efetivo de produtividade e qualidade no projeto de interfaces através do ambiente, quando comparado com abordagens tradicionais para projeto e implementação de interfaces.

PALAVRAS CHAVES

Interfaces; Adaptação; Regras; Desenvolvimento Dirigido por Modelos; REST.

Abstract

Bertti, Ezequiel; Schwabe, Daniel (Advisor). **MIRA – A Model-Driven Interface Framework for REST Applications**. Rio de Janeiro, 2015. 137p. MSc. Dissertation – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro

This work presentes a Model-driven framework for the design of interfaces for REST applications. The framework allows building interfaces with minimal programming, The models used, as well as the generated interfaces are represented using W3C standards. A qualitative evaluation indicates that there are gains in both productivity and quality of the generated interfaces, when compared with traditional approaches.

KEYSWORDS

Interfaces; Adaptation; Rules; Model Driven Development; REST.

SUMARIO

LISTA DE FIGURAS

Lista de Quadros

LISTA DE TABELAS

LISTA DE GRÁFICOS

1 INTRODUÇÃO

O interesse no uso de modelos e abstrações de interface tem aumentado em diversas áreas de pesquisa justamente por deixar o projetista realizar suas tarefas junto ao sistema sem se preocupar com a construção das interfaces para tal, naquele momento.

Tipicamente, a interface gráfica, representa cerca de 48% do código de fonte, requer cerca de 45% do tempo de desenvolvimento e de 50% do tempo de aplicação, e atinge 37% do tempo de manutenção [1].

Em paralelo, observamos que a cada dia há lançamento de novos dispositivos e novas plataformas para a exibição de informação para seus usuários. Cada plataforma possui uma interface distinta para interagir com seus usuários. Um conceito relacionado que está vindo à tona é o de "Internet of Things" [2]. Por exemplo, televisões utilizam o controle remoto como interface de manuseio e fazem uso de telas grandes para exibir suas interfaces gráficas.

Como ilustração da extensão desta tendência, a figura a seguir mostra quase todos os tipos de dispositivos para os quais a empresa Netflix tem aplicações criadas para exibir os vídeos que ela provê.

Figura 01 – Dispositivos para os quais o Netflix possui interface com o usuário

1.1 **OBJETIVOS**

Nesta dissertação iremos aproveitar o trabalho desenvolvido no Synth, uma ferramenta de autoria de aplicação de web semântica, no que diz respeito à definição de interfaces, para criar um novo framework. Este framework se integrará a aplicações que disponibilizam uma API Web feita com REST, um padrão de comunicação simplificada amplamente utilizada em aplicações da Web. Em outras palavras, a partir de um modelo de interface no SHDM, de seleção de interfaces, interfaces abstratas e interfaces concreta, será possível mapear estas APIs gerando uma interface baseada em regras para manipular seus dados.

As regras criadas pelo projetista para mapear a API podem utilizar, além dos dados retornados pelas APIs, os recursos do navegador, que está acessando a aplicação, e o ambiente, que está executando o framework. Com isto, será possível criar interfaces para diversos dispositivos alterando seus Widgets conforme regras mapeadas pelo projetista da aplicação.

1.2 **ORGANIZAÇÃO DA DISSERTAÇÃO**

Os temas apresentados nessa dissertação estão organizados em capítulos da seguinte maneira:

- **Capítulo 2 – Fundamentos**: uma breve revisão das principais áreas de conhecimento envolvidas no desenvolvimento desse trabalho;
- **Capítulo 3 – Modelos do MIRA para de Especificação de Interface**: apresenta a forma de especificar a interface no framework proposto;
- **Capítulo 4 – Arquitetura e Implementação do Framework:**
- **Capítulo 5 – Comparação com outros frameworks de** interface

Métrica	**MIRA**	**Angular**	**Backbone**	**Jquery**
Média por função de Logical LOC	7	5,19	7,89	5,08
Média por função de Parameter count	3,27	1,55	1,27	1,43
Média por função de Cyclomatic complexity	1,87	2,26	3,25	3,13
Média por função de Halstead effort	1548,43	2907,37	5864,19	5986,7
Média por módulo Maintainability index	115,74	116,86	107,59	114,67

- Estudo de Caso e Avaliação**:** traz uma avaliação qualitativa de aplicações construídas utilizando o framework proposto

2 FUNDAMENTOS

2.1 **WEB API**

O desenvolvimento de sistemas na Web com integrações em diversas plataformas como Websites, Mobile, Desktop forçou que seus desenvolvedores fizessem o uso de uma API (Application Programming Interface) Web para se comunicar com seus diversos clientes, como forma de encapsular a "lógica de negócio" da aplicação, separando-a da lógica da interface. Esta abordagem acabou se firmando no mercado, utilizando sobretudo a comunicação através de requisições utilizando o protocolo HTTP.

Uma API é um conjunto de funções que servem para estabelecer uma comunicação de serviços e seus consumidores. Existem diversas formas de se implementar uma API. Na Web uma forma muito utilizada para dar acesso a uma API é através de um web service [3], e geralmente utiliza os formatos XML ou JSON para representar os dados trocados nas suas requisições. Dentre as técnicas de interação com web services os mais comuns são o protocolo SOAP (Simple Object Access Protocol) e a abordagem REST (Representation State Transfer). O SOAP está em desuso pela complexidade nas definições de esquema e overhead criado para uma simples troca de mensagens para que funcione em diversos protocolos além do HTTP.

2.2 **API RESTFUL**

O conceito de REST surgiu no ano de 2000 na da tese de Roy T. Fielding [4] como uma alternativa mais “leve” ao SOAP. Ele foi construído para funcionar apenas em cima do protocolo HTTP, diminuindo qualquer overhead de comunicação. Em outras palavras, não requer as abstrações adicionais acima do protocolo que trafega suas mensagens. Sistemas nele baseados são geralmente chamados de RESTful.

Um sistema RESTful se caracteriza por seguir estes fundamentos:

1. Um protocolo cliente servidor sem estado;

Cada mensagem de requisição HTTP enviada pelo cliente para o servidor deve conter todas a informações para suas requisições, como identificação de

sessão do cliente, URI que deseja navegar e a validação de sua permissão, já que o servidor não guarda o estado de navegação de cada cliente;

2. Um conjunto de operações bem definidas que se aplicam a todos os recursos de informação

As operações disponíveis são POST, GET, PUT e DELETE que são frequentemente associadas com operações CRUD do sistema (CREATE, READ, UPDATE e DELETE respectivamente).

3. Uma sintaxe universal para identificar os recursos;

Em um sistema REST, cada recurso é unicamente identificado através da sua URI.

4. Uso de hipermédia, tanto para a informação da aplicação como para as transições de estado dela.

Deve ser possível navegar desde um recurso REST a muitos outros, seguindo suas ligações sem requerer o uso de registros ou uma definição de esquema.

2.3 **INTERFACES DIRIGIDAS POR MODELOS**

Nos últimos anos, diversas abordagens têm sido desenvolvidas para se enfrentar os desafios existentes na geração de interfaces com o usuário, muitas delas através do uso de modelos. Porém, os desafios mais recentes estão relacionados em oferecer mecanismos para se projetar interfaces que possam atender as diversas plataformas, dispositivos e variados contextos de uso. Além disso J.Coutaz e Calvary apresentam a questão da plasticidade das interfaces, como sendo a capacidade delas se adaptarem aos contextos de uso preservando a usabilidade e os valores humanos [5].

2.4 **CAMELEON**

CAMELEON Reference Framework (CRF) é a referência mais aceita para projetar e classificar arquiteturas de interfaces baseadas em modelos, levando em conta o suporte a diversos contextos e plasticidade de interfaces.

O contexto de uso é formado por três entidades:

- **Usuário** – atributos que definem o perfil do usuário ou o usuário que utiliza um sistema
- **Plataforma** – características do dispositivo, como por exemplo, sistema operacional, características físicas e de software (dispositivo, aplicativo, funcionalidades, etc.)
- **Ambiente** – atributos relacionados ao espaço (lugar), tempo, funções e outras propriedades que caracterizem quando e onde a interação está ocorrendo. Por exemplo: localização, fonte de energia, disponibilidade de rede etc.

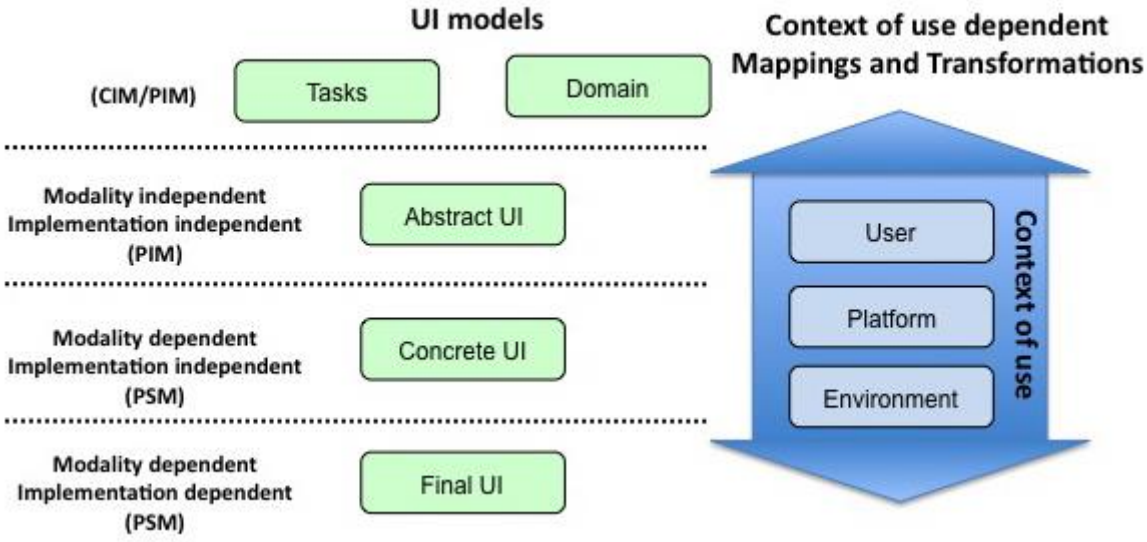

Figura 02 – Diagrama simplificado do CRF

Em relação aos modelos existentes no CRF, a Figura 02 mostra as relações entre eles e a influência do contexto de uso sobre cada etapa de modelagem, sendo essas:

- **Tarefas e domínio** – definem que objetos de domínio podem ser manipulados em uma interface e que tarefas que podem ser realizadas pelo usuário (por exemplo, ‘Exibir fotos’)
- **Interface Abstrata (AUI)** – define os elementos de interface que estão previstos em um nível abstrato, sem focar em que componente concreto irá ser utilizado. Por exemplo, ‘Exibir valor’. Esses elementos são independentes de implementação ou plataforma;

- **Interface concreta (CUI)** – define quais widgets concretos que possam satisfazer a tarefa. Por exemplo, "Widget de navegação" ou "Campo de texto". Os widgets concretos são dependentes do ambiente de execução;
- **Interface final** – é a interface final gerada e utilizável por um usuário. Como por exemplo, em HTML.

Existem outras abordagens para arquitetura de interfaces baseadas em modelos. Como: TERESA [6], MARIA XML [7] e USIXML [8], mas não entraremos no detalhe destas pois assim como o MIRA, todas são baseadas nos conceitos propostos pelo CAMELEON.

2.5 **OOHDM**

Dentre os diversos métodos propostos para o processo de desenvolvimento de aplicações [OOH, WebML], destacamos o OOHDM (Object-Oriented Hypermedia Design Method) [9], um método baseado em modelos e que utiliza práticas de orientação a objetos para o projeto de aplicações hipermídia. O grande diferencial deste método para construção de aplicações hipermídia, comparado a métodos tradicionais de engenharia de software, é a destacada relevância dada ao aspecto navegacional da aplicação, bem como o tratamento do projeto de interfaces de forma independente da navegação, e também das tecnologias e ambientes de execução. A aplicação é modelada em cinco etapas distintas: Levantamento de Requisitos, Modelagem Conceitual ou de Domínio, Modelagem Navegacional, Projeto de Interface Abstrata e Implementação [10]. O processo se dá de forma iterativa e incremental seguindo o desenvolvimento de cada etapa e produzindo novos modelos ou enriquecendo os já existentes.

2.6 **SHDM**

O SHDM [11] é uma evolução do método OOHDM que leva em consideração os formalismos e primitivas introduzidas pela Web Semântica. O método permite o projeto de aplicações hipermídia baseadas em ontologias, descritas através de meta dados. O método SHDM mantém a mesma estrutura

de fases de modelagem para o método OOHDM, mantendo a separação entre modelos de domínio, navegacionais e de interface, e demais conceitos do método original.

O modelo SHDM possui ontologias que representam modelos para o design de interfaces abstratas e concretas respectivamente, definindo Widgets Abstratos e Widgets Concretos [12]. A interface concreta é responsável por mapear os Widgets Abstratos para Widgets Concretos segundo regras definidas pelo projetista da aplicação. Além destas ontologias, o SHDM propõe também regras que permitem a síntese de uma interface dinamicamente, levando-se em conta diversos aspectos além do próprio modelo especificado - dados de contexto de uso, e os próprios dados sendo exibidos a cada requisição.

Baseado nestes modelos, foi desenvolvido o Synth [13], um ambiente de desenvolvimento que dá suporte à construção de aplicações modeladas segundo o método SHDM. Ele foi desenvolvido para permitir a manipulação de dados utilizando o modelo RDF [14], que ainda não é muito difundido no mercado. Este ambiente inclui um módulo com uma interface de autoria para modelos de interface que divide a modelagem em três etapas: as regras de seleção de interface, a descrição abstrata, as regras de composição e o mapeamento concreto da interface da aplicação final.

2.7 **FRAMEWORKS DE INTERFACE**

Há muitos frameworks para construção de interface para aplicações *cliente-side*. O MIRA, inclusive, se baseia no framework Backbone.js [15]. O Backbone.js é um framework Javascript que fornece componentes para melhorar a estrutura de aplicações web. Entre estes componentes encontram-se Models, Collections e Views, além de meios nativos de interagir com backends RESTful e JSON.

Dentre outros existentes com este conceito de client-side, um dos mais populares atualmente é o Angular.js [16], Angularjs é um framework javascript construído e mantido pelo grupo de engenheiros do Google, ele usa o HTML como uma “template engine”.

Mas como ele está preso à estrutura do HTML, ele se torna dependente de uma estrutura pré-determinada de elementos concretos, gerando a

necessidade de recriar a interface em caso de mudanças de contexto como dispositivo ou regras de negócio.

A funcionalidade de maior destaque do Angular, é o *Two-Way Data-Binding*, onde informações alteradas na visão, são refletidas no modelo, e as informações refletidas no modelo, são atualizadas na visão.

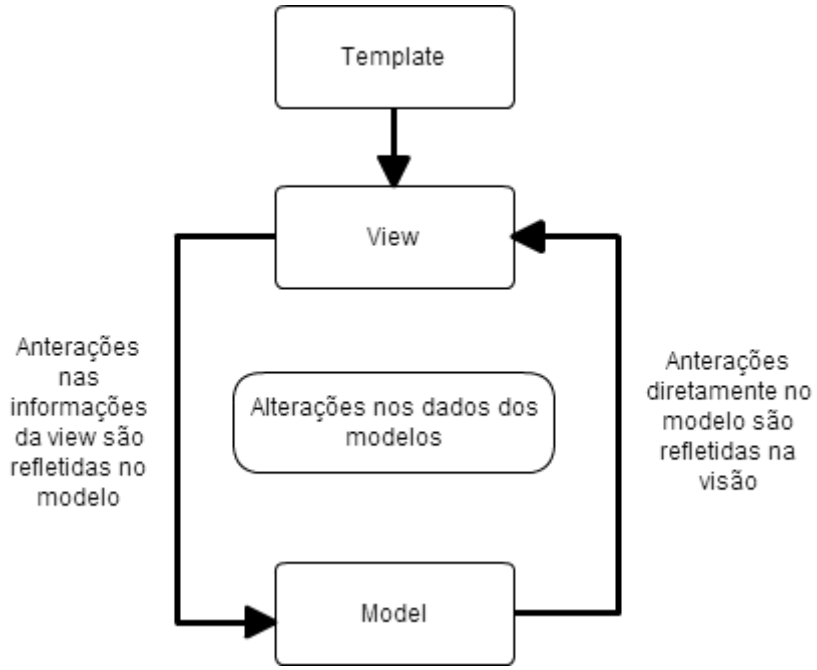

Figura 03 – Two-Way Data-Binding realizado pelo AngularJS

Os modelos, ou Model, representam os dados carregados e exibidos na tela. A View é uma estrutura para exibição criada a partir de um Template feito em HTML com atributos diferenciados que o Angular denomina de Diretivas. Quando algum dado é alterado na View, logo é refletida no modelo, que desencadeia a alteração da View e do HTML exibido na interface.

3 MODELOS DO MIRA PARA DE ESPECIFICAÇÃO DE INTERFACE

Os modelos do MIRA para especificar interfaces são as **Condições de Negócio** com a **Seleção de Interface**, e as especificações das **Interfaces Abstrata**, **e Concreta;** e a partir destes modelos a interface é gerada. Como será detalhado subsequentemente, diferentemente do Cameleon, o MIRA junta as camadas de Concreta e Interface Final em uma camada só.

A seguir será mostrado como o framework sintetiza interfaces a partir destes modelos.

No MIRA, a interface é montada a partir de uma solicitação de dados através da URI de uma API REST. Estes dados retornados são avaliados, junto com o contexto da aplicação, por regras que avaliam as condições de negócio para selecionar a interface mais apropriada para exibi-los. Em seguida, estes dados e o contexto serão utilizados para montar a interface abstrata com seus widgets abstratos. Finalmente, novas regras são avaliadas para realizar o mapeamento dos widgets concretos na interface concreta que será utilizada para exibir os dados para o usuário final, também levando em conta o contexto.

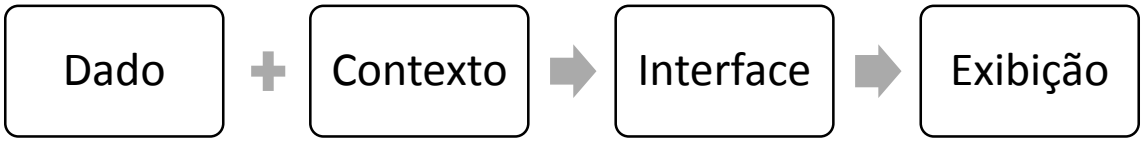

Figura 04 – Sequência de eventos para exibição de uma interface usando o MIRA

O MIRA não faz distinção da fonte dos dados solicitados, que podem vir de URIs completamente distintas e sem padrão em sua nomenclatura. O que é avaliado pelos modelos de interfaces do MIRA são as estruturas de dados retornados pelas APIs, junto com informações do contexto.

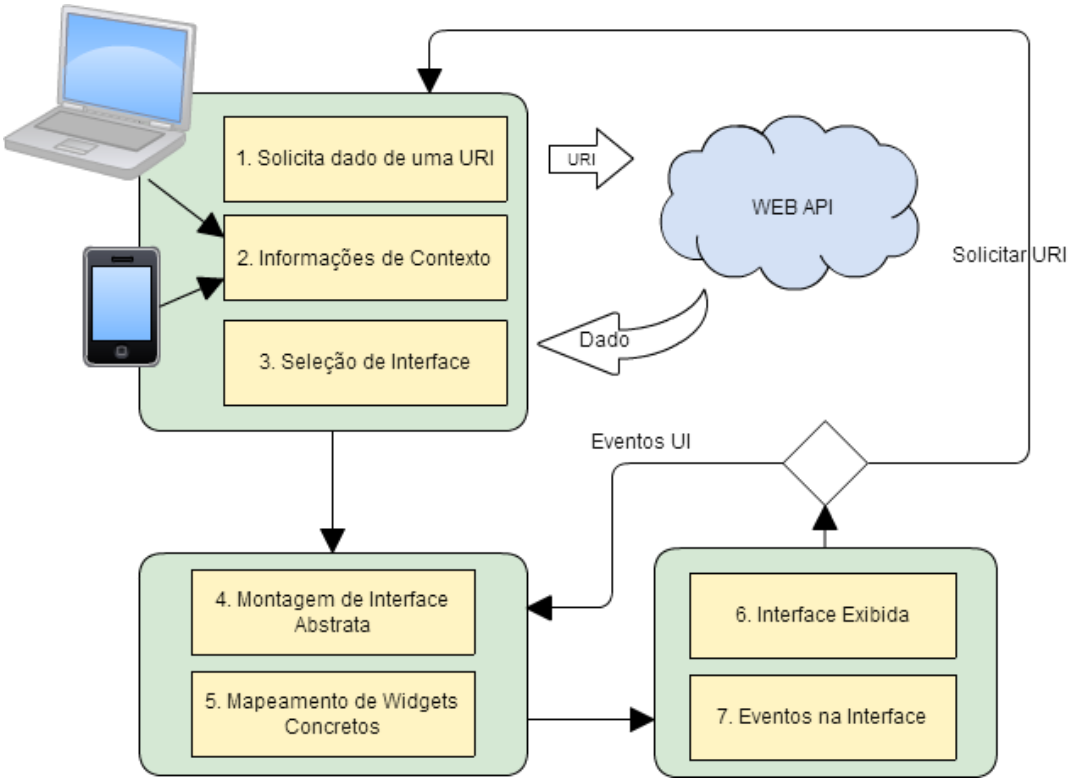

Figura 05 – Sequência de execução para exibição de uma interface

Em uma abordagem convencional de aplicações, a interface é tipicamente selecionada através da interpretação da URI utilizada para acesso, e a partir do contexto da aplicação, é feita a requisição de informação dentro de sua base de conhecimento para ser exibida. Se a URI não for conhecida pela aplicação, ela não consegue exibir qualquer informação.

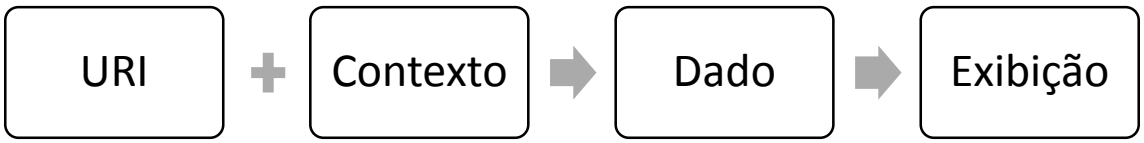

Figura 06 – Sequência de eventos para exibir uma interface em uma aplicação convencional

3.1 CONDIÇÕES DE NEGÓCIO

Os modelos do MIRA utilizam regras do tipo “condição → ação” para selecionar interfaces e para mapear interfaces abstratas em interfaces concretas. Desta forma, as definições de condições de negócio, ou regras de negócio, avaliam as informações do contexto para selecionar quais interfaces serão montadas e na sequência, que widgets concretos serão mapeados nos widgets abstratos.

Uma condição é composta pelos seguintes atributos:

- **Name***:* será o nome utilizado em todas as regras que utilizarão esta condição.
- **Validate:** é a condição que será avaliada.

Uma regra pode ser fatorada em diversas condições de negócio; a forma que representamos uma condição pode ser vista nos exemplos a seguir:

1) Se o tipo do dado retornado possuir o atributo com o valor ‘edit’ deve ser exibido um botão com a possibilidade de editar o conteúdo.

```
{
    name: 'PermissaoEdicao',
    validate: '$data.edit == true'
}
```

Quadro 01 – Condição de negócio que habilita edição

2) Se o dispositivo que está acessando a aplicação for um desktop, deve ser exibido um mapa dinâmico, com possibilidade de Zoom. Se for um dispositivo Móvel ou Tablet, devemos exibir um mapa estático.

```
{
    name: 'isMobileOrTablet',
    validate: '$env.device.mobile == true || $env.device.tablet == true'
},{
    name: 'isDesktop',
    validate: '$env.device.desktop == true'
}
```

Quadro 02 – Condição de negócio que detecta o tipo de dispositivo

3) Se o dado retornado pela URI contiver informações de ‘login’, deve ser montada a interface de usuário.

```
{
    name: 'isUser',
    validate: '$data.login != null'
}
```

Quadro 03 – Condição de negócio que detecta que o usuário está “logado”

O MIRA prevê a declaração das condições em uma seção separada das regras propriamente ditas. Este fatoramento das condições permite o seu reuso em diversas regras diferentes utilizadas na construção da interface. Visto que as regras das interfaces mudam com mais facilidade que as regras de negócio dos modelos da aplicação, a centralização destas condições permite ao projetista, em caso de alguma mudança, alterar somente as condições de forma

independente do comportamento da interface. Isto também permite verificar quais elementos da interface fazem o uso da mesma condição nas regras do modelo da interface.

3.2 CRITÉRIO DE SELEÇÃO DE INTERFACE

A seleção de interface se dá pelo uso de condições e da indicação que interface abstrata e concreta será montada para o usuário. Para simplificar a especificação das regras de seleção, caso o projetista não informe o nome da interface concreta, o MIRA assume que será utilizada a interface concreta com o mesmo nome da interface abstrata.

```
var selection = [
    {
        when: 'isUser',
        abstract: 'user'
    },{
        when: 'isRepository',
        abstract: 'repository'
    }
];
```

Quadro 04 – Exemplo de regra para seleção de interface

No exemplo do Quadro 04, se a condição *isUser* for válida, a interface abstrata e concreta com nome *user* será selecionada. Da mesma forma, se a condição *isRepository* for verdadeira, a interface selecionada será *repository*.

A seleção de interface é composta pelos seguintes parâmetros:

- **When:** Condição que precisa ser válida para que a interface seja selecionada. Se nenhuma condição for válida, será selecionada a interface com o nome de *“not_found”*
- **Abstract:** Nome da interface abstrata que será mapeada caso a seleção seja válida.
- **Concrete:** Opcional, nome da interface concreta que será utilizada para o mapeamento de Widgets Concretos da Interface Abstrata. Se nenhum nome for informado, será utilizado a Interface Concreta com o mesmo nome que a interface Abstrata.

A sequência de execução segue o diagrama a seguir:

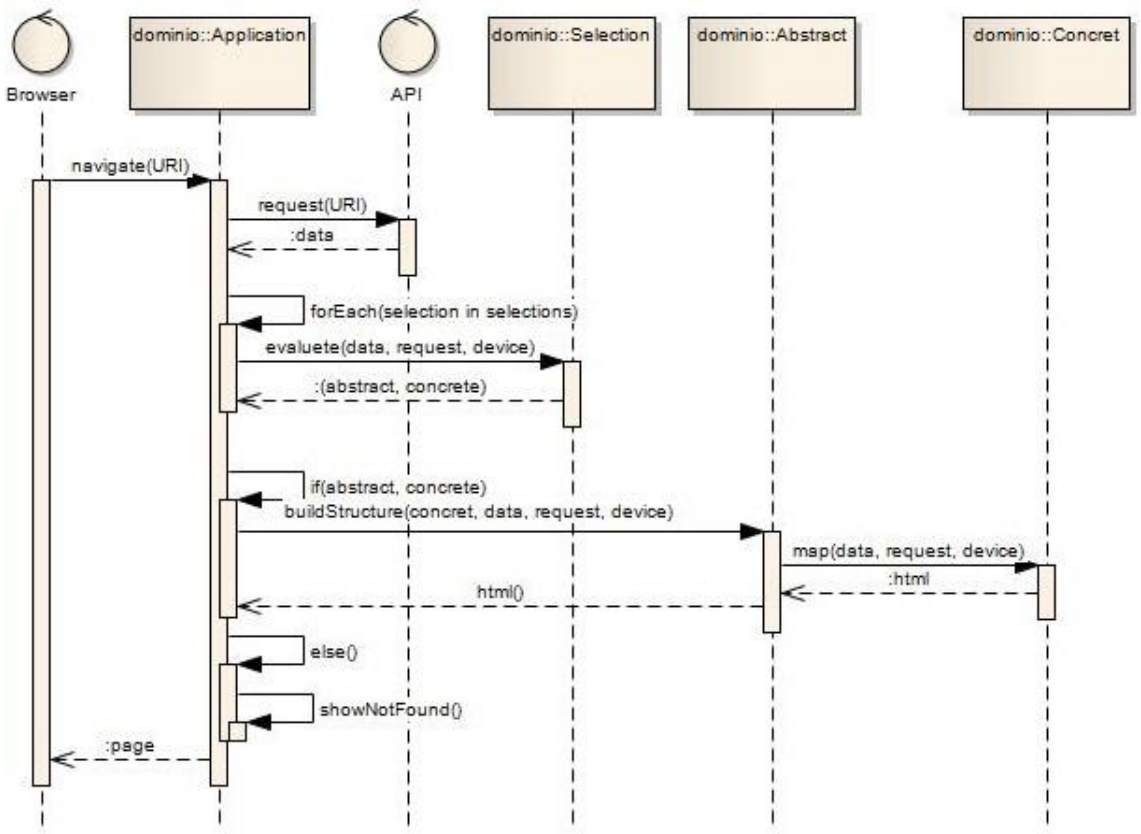

Figura 07 – Diagrama de sequência para a seleção de interface

Há duas formas para ser executar a requisição dos dados de uma API e a seleção de interface; a mais simples é funcionando totalmente no cliente (*client-side*), a outra forma é utilizando um servidor.

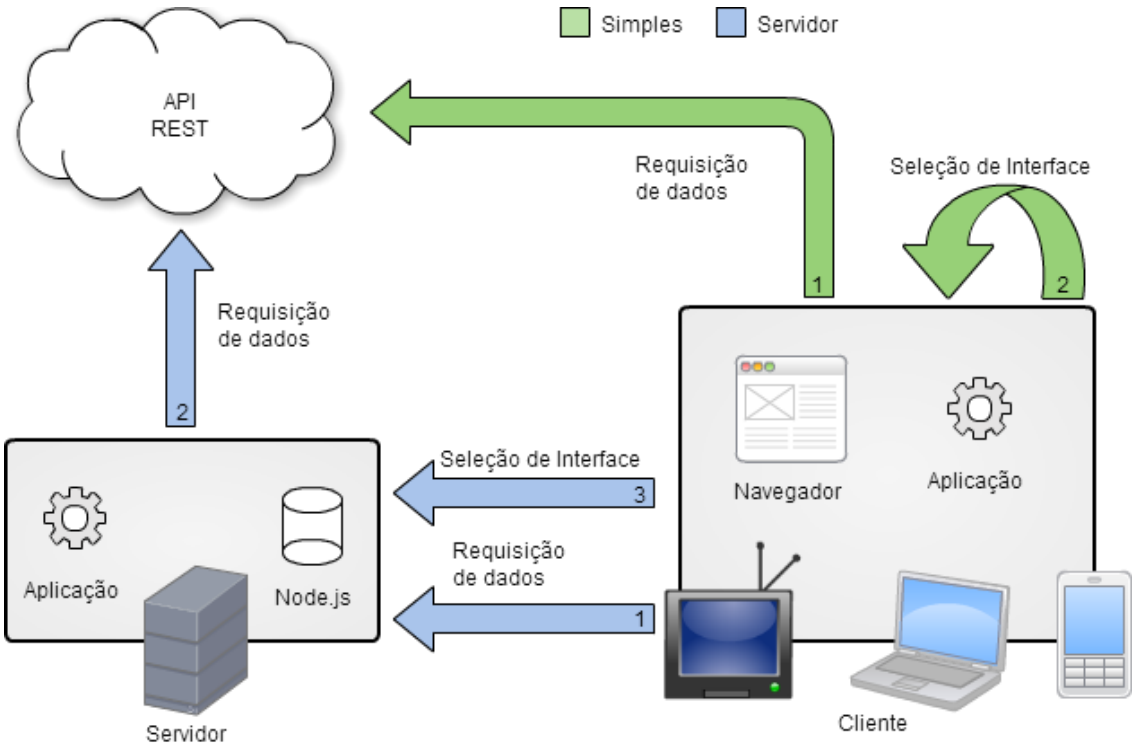

Figura 08 – Esquema de execução de seleção de interface no cliente ou no servidor

Utilizando o modelo simples, todas as regras e requisições de dados são executadas pelo navegador do usuário. Em caso de uso de regras elaboradas com inúmeras validações, que poderiam carregar o dispositivo, ou bloqueio de acesso à API por controle de Cross-site Scripting [17], que impede o acesso a URIs de domínio diferente do domínio do site acessado para proteger o usuário, é possível fazer a distribuição destas tarefas no servidor. Deste modo, diminui-se a carga de processamento do cliente que está acessando a aplicação.

O MIRA avalia todas as condições de interface listadas na seleção, e seleciona sempre a última que teve sua condição considerada válida, seguindo a ordem de declaração.

No exemplo a seguir, o dado retornado pela WEB API está contido na variável *uri_data*, e existem 2 regras habilitas, porém será a exibida a interface correspondente à última regra listada que foi habilitada listada.

```
var uri_data = { login: "ebertti", edit: true  /* ... */ };

var conditions = [{
  name: 'isUser', validate: '$data.login != null'
},{
  name: 'isUserEdit', validate: '$data.login != null && $data.edit == true'
},{
  name: 'isPhoto',  validate: '$data.photo_id != null'
},{
  name: 'isPhotoEdit', validate: '$data.photo_id != null && $data.edit == true'
}];

var selection = [{
    when: 'isUser', abstract: 'user' /* é válida */
},{
    when: 'isPhoto', abstract: 'photo'
},{
    when: 'isUserEdit', abstract: 'user-edit' /* é válida e foi selecionada */
},{
    when: 'isPhotoEdit', abstract: 'photo-edit'
}];
```

Quadro 05 – Exemplo de ordenação nos critérios de seleção de interface

3.3 MODELAGEM DE INTERFACE ABSTRATA

A interface abstrata no SHDM representa a interface do ponto de vista do fluxo de dados entre a aplicação e o usuário. Projetar uma interface abstrata em uma aplicação SHDM consiste basicamente em construir uma composição hierárquica de elementos disponíveis na ontologia de Widgets Abstratos, de tal forma que essa composição descreva como os objetos da aplicação serão apresentados ou terão seus valores alterados, especificando os elementos perceptíveis que estarão disponíveis para o projetista da interface concreta (“designer”) [12].

Os Widgets Abstratos representam os papéis possíveis para um Widget de interface em relação à troca de informações na interface, abstraindo a forma específica como esta troca é concretizada em um ambiente de execução.

Depois da interface ter sido selecionada, é montada a estrutura abstrata a ser mapeada no próximo passo. O projetista deverá montar suas interfaces com estruturas de Widges Abstratos (Abstract Interface Elements):

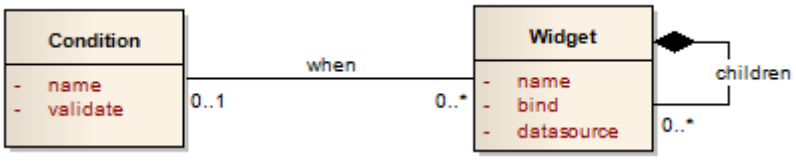

Figura 09 – Diagrama de classes com a hierarquia do widget abstrato

A seguir mostramos um exemplo de como o projetista especifica uma interface abstrata, usando como exemplo a tela na Figura 10

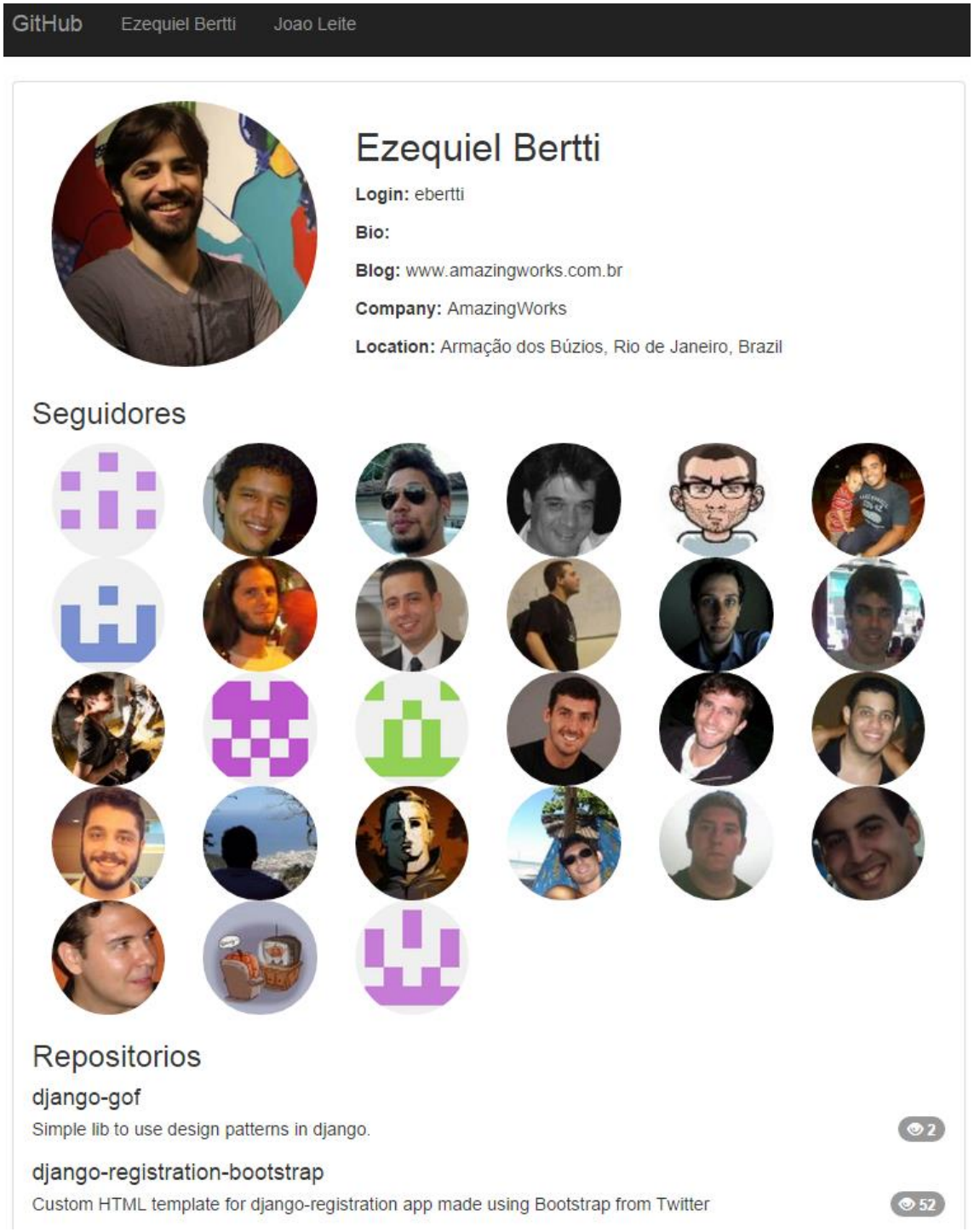

Figura 10 – Exemplo da interface montada pelos widgets mapeados

A estrutura da interface abstrata correspondente a esta tela é mostrada na Figura 11.

Figura 11 – Composição dos de Widgets Abstratos em uma interface exibida para o usuário.

```
var abstracts = [
    { name:"user",
        widgets:[ { name:"navigation",
            children:[ { name:"navigation-list",
                children:[ { name:"navigation-list-item" } ],
                datasource:"obj" } ] },
            { name:"content",
                children:[ { name:"user",
                    children:[ { name:"avatar" },
                        { name:"detail",
                            children:[ { name:"name" },
                                { name:"login" },
                                { name:"bio" },
                                { name:"blog" },
                                { name:"company" },
                                { name:"location" } ] } ] },
                    { name:"seguidores_panel",
                        children:[ { name:"seguidores_title" },
                            { name:"seguidores",
                                children:[ { name:"seguidor",
                                    children:[ { name:"avatar_seguidor" } ] } ],
                                datasource:"url:<%= $data.followers_url %>" },
                            { name:"seguidores_mais" } ] },
                    { name:"repositorios_panel",
                        children:[ { name:"repositorios_title" },
                            { name:"repositorios",
                                children:[ { name:"repositorio",
                                    children:[ { name:"nome" },
                                        { name:"descricao" },
                                        { name:"box",
                                            children:[ { name:"star" },
                                                { name:"watch" } ] } ] } ],
                                datasource:"url:<%= $data.repos_url %>" } ] } ] },
            { name:"footer",
                children:[ { name:"footer-content" } ] } ] }
];
```

Quadro 06 – Exemplo de construção de uma interface abstrata

Uma interface abstrata é definida através dos seguintes parâmetros:

- **Name:** Nome da interface abstrata, este nome será utilizado pela no parâmetro abstract de elementos da Seleção de Interface.
- **Widgets:** Uma lista de Widgets Abstratos que compõem a interface Interface Abstrata.

Um Widget Abstrato é composto pelos seguintes parâmetros:

- **Name:** Nome que será usado pelo mapeamento de Widgets Concretos.
- **When:** Condição que precisa ser válida para que este Widget Abstrato esteja disponível para o mapeamento de Widget Concreto, se não for informada nenhuma condição, o Widget será considerado como disponível para o mapeamento.

- **Datasource:** Opcional, URI de uma API ou uma variável que será utilizada para montar os filhos desse Widget Concreto.
- **Children:** Lista de Widgets Abstratos que são filhos deste Widget.

3.4 **MAPEAMENTO CONCRETO**

Os Widgets Concretos descrevem os possíveis elementos concretos de interface das aplicações SHDM. Um Widget Concreto WC corresponde a um elemento de interface capaz de ser processado em um dado ambiente de execução. Na Web padrão, corresponde a elementos definidos no HTML, tais como check box, pull-down menu, radio box, etc.

Cada Widget Abstrato deve ser mapeado para um determinado Widget Concreto afim de que seja incluído na interface final gerada a ser exibida ao usuário.

A seleção do Widget Concreto apropriado para a construção da interface final é baseada em regras pré-definidas pelo projetista da aplicação. Quando uma regra é satisfeita, ela irá mapear todos os Widgets Abstratos para os Widgets Concretos definidos por aquela regra.

Se um Widget Abstrato não possuir mapeamento para um Widget Concreto, nada será exibido na interface final para o usuário.

Em seguida, definimos os mapeamentos e utilizamos as regras definidas anteriormente, para a tela de exemplo exibida na Figura 10.

```
var concrete = {
    name: 'user', maps: [
    { name: 'navigation', widget: 'BootstrapNavigation', value:'"GitHub"'},
    { name: 'navigation-list', widget: 'BootstrapNavigationList'},
    { name: 'navigation-list-item', widget: 'BootstrapNavigationListItem',
value:'data.name', href:'navigate(data.link)'},

    { name: 'content', widget: 'ProfileContainer' },
    { name: 'user', widget: 'SimpleHtml', class:'clearfix' },
    { name: 'avatar', widget: 'ProfileImage', value:'data.avatar_url' },
    { name: 'detail', widget: 'SimpleHtml', class:'col-xs-12 col-sm-8' },
    { name: 'name', widget: 'SimpleHtml', tag: 'h2', value: 'data.name' },
    { name: 'login', widget: 'ProfileDetail', detail: 'Login', value:
'data.login'},
    { name: 'bio', widget: 'ProfileDetail', detail: 'Bio', value: 'data.bio'},
    { name: 'blog', widget: 'ProfileDetail', detail: 'Blog', value: 'data.blog'},
    { name: 'company', widget: 'ProfileDetail', detail: 'Company', value:
'data.company'},
    { name: 'location', widget: 'ProfileDetail', detail: 'Location', value:
'data.location'},
    { name: 'seguidores_panel', widget: 'SimpleHtml', class:'clearfix' },
    { name: 'seguidores_title', widget: 'SimpleHtml', tag: 'h3', class:'clearfix',
value:'"Seguidores"'},
    { name: 'seguidores', widget: 'SimpleHtml', tag: 'div'},
    { name: 'seguidor', widget: 'SimpleHtml', tag: 'a', href:
'navigate(data.url)'},
    { name: 'avatar_seguidor', widget: 'SimpleHtml', class:'col-md-2 col-xs-3 img-
circle img-responsive', tag: 'img', src: 'data.avatar_url + "s=80"',
alt:'data.login', title:'data.login'},
    { name: 'repositorios_panel', widget: 'SimpleHtml', class:'clearfix' },
    { name: 'repositorios_title', widget: 'SimpleHtml', tag: 'h3',
value:'"Repositorios"'},
    { name: 'repositorios', widget: 'SimpleHtml', tag: 'div'},
    { name: 'repositorio', widget: 'SimpleHtml', tag: 'div', class:'media'},
    { name: 'nome', widget: 'SimpleHtml', tag: 'h4', value: 'data.name',
class:'media-heading'},
    { name: 'descricao', widget: 'SimpleHtml', tag: 'span', value:
'data.description'},
    { name: 'box', widget: 'SimpleHtml', tag: 'ul', class:'nav nav-pills nav-
stacked pull-right'},
    { name: 'watch', widget: 'ProfileCount', icon:'eye-close',
value:'data.watchers_count'},
    { name: 'watch', widget: 'ProfileCount', icon:'eye-open',
value:'data.watchers_count', when:'haveWatch'},

    { name: 'footer', widget: 'SimpleHtml', tag:'div', class:'container' },
    { name: 'footer-content', widget: 'BootstrapFooter' }
]}];
```

Quadro 07 – Exemplo de construção da interface concreta

Uma Interface Concreta é definida através dos seguintes parâmetros:

- **Name:** Nome da interface concreta, este nome será utilizado pela no parâmetro *concrete* na Seleção de Interface.
- **Head:** Lista com Widgets Concretos utilizados para montar dependências da interface exibida para o usuário, tais como arquivos de estilo (CSS).

- **Maps:** Lista com Widgets Concretos para o mapeamento dos widgets abstratos da interface abstrata selecionada.

O mapeamento de Widget Concreto pode conter inúmeros parâmetros dependendo daquilo que o Widget Concreto utilizado no mapeamento requer. Mas, no mínimo, os seguintes parâmetros devem estar presentes:

- **Name:** Nome Widget Abstrato que será mapeado.
- **When:** Condição que precisa ser válida para que este mapeamento seja incluído na interface montada para o usuário.
- **Widget:** Nome do Widget Concreto que será utilizado para montar a interface do usuário.

Um exemplo de regra que foi utilizado na interface de exemplo é aquela que especifica que quando um repositório não possui observadores, este deve ser exibido com um ícone com um olho traçado com o valor 0, que é a contagem de observadores. E se possuir observadores, deve ser exibido um ícone com um olho aberto, com a contagem de quantos observadores o repositório em questão possui.

```
var conditions = [{
    name: 'haveWatch', validate: '$data.watchers_count > 0'
}];

var concrete = [{
name: 'user',
maps: [
    // ... //
    { name: 'watch',
        widget: 'ProfileCount', icon:'eye-close', value:'0'},
    { name: 'watch', when:'haveWatch',
        widget:'ProfileCount', icon:'eye-open', value:'$data.watchers_count' }
]}];
```

Quadro 08 – Exemplo de mapeamento de Widget Concreto

Figura 12 – Exemplo da interface montada pelos widgets mapeados pelas regras do Quadro 08

3.5 CONSTRUÇÃO DE WIDGETS CONCRETOS

Ao desenvolver uma interface nova para uma aplicação, muitas vezes os widgets concretos existentes na biblioteca do MIRA não serão suficientes para expressar todos os requisitos necessários da aplicação.

Para suprir esta necessidade, o MIRA pode ser estendido, através da ciração de novos widgets concretos para serem utilizados no mapeamento da interface concreta. Um exemplo é a criação de um widget chamado *CustomTitle*, onde será exibido para o usuário um elemento HTML do tipo *H1* com o atributo *class* contento o valor *title* e com conteúdo o valor contido no atributo *value* informado como parâmetro do mapeamento de widget concreto na interface concreta.

```
var custom_widgets = {
    CustomTitle: function($parent, name, $context, options, callback) {
        var element = document.createElement('h1');
        element.id = name;
        element.setAttribute('class', 'title');
        element.innerHTML = options.value;

        $parent.append(element);
        var $element = $(element);

        if(callback){
            callback({
                $children: $element,
                $element: $element
            })
        }
    }
};
Mira.Widget.register(custom_widgets);
```

Quadro 09 – Exemplo de construção e registro de um widget concreto customizado

A construtor de widgets recebe os seguintes parâmetros:

- **$parent:** Corresponde ao elemento de interface ao qual o widget deverá ser adicionado
- **name:** O nome do Widget Concreto que está sendo mapeado
- **$context:** Objeto contendo informações do ambiente, da requisição HTTP, do objeto carregado via API e o objeto do contexto atual.
- **options:** Parâmetros adicionais informados no mapeamento do Widget Concreto.
- **callback:** Método que deve ser chamado no final da montagem do Widget Concreto.

3.6 **TRATAMENTO DE EVENTOS**

O MIRA, diferente de outras abordagens de interface do SHDM [18] permite o tratamento explicito de eventos disparados pela interface. Com estes eventos é possível manipular e interagir com os dados carregados dinamicamente na interface em tempo de execução, por exemplo, quando o usuário entra com dados em algum widget de captura de dados (input).

Se durante o tratamento de algum evento houver modificação nos dados previamente utilizados na avaliação das condições, ou em informações exibidas na interface, o MIRA reavalia os Widgets Abstratos e Concretos que foram utilizados para montar a interface, atualizando a tela exibida com as possíveis alterações nas informações e/ou Widgets exibidos para o usuário.

Os eventos tratados são registrados como parte do mapeamento de Widgets Concretos.

```
{name:'status', value:'Aguardando', events:{click:'marcar_feito'}},
{name:'status', value:'Feito', when:'isDone'}
```

Quadro 10 – Exemplo de mapeamento de eventos

No exemplo do Quadro 10, quando houver um evento de clique no widget *status*, será feito uma chamada para a função *marcar_feito*:

```
window.marcar_feito = function(options){
    options.$dataObj.set('feito', true);
};
```

Quadro 11 – Exemplo de tratamento de um evento disparado por um widget

Todas as funções que tratam eventos, recebem um parâmetro chamado **options**, que é composto pelos seguintes atributos:

- **$env:** Informações sobre o contexto da aplicação, como request, device e dado carregado pelo request da URI.
- **$data:** Dados de leitura do objeto carregado da URI ou do contexto de um datasource de um Widget Abstrato
- **$dataObj:** Objeto para manipular informações dos dados que foram carregados da URI ou do contesto de um datasource de um Widget Abstrato
- **$element:** Elemento HTML do contexto do evento
- **$event:** Informações do evento disparado

- **$map:** Widget Concreto utilizado no mapeamento.

A mudança de valores nos dados carregados previamente na interface dispara a reavaliação de widgets, como descrito no diagrama de sequência a seguir:

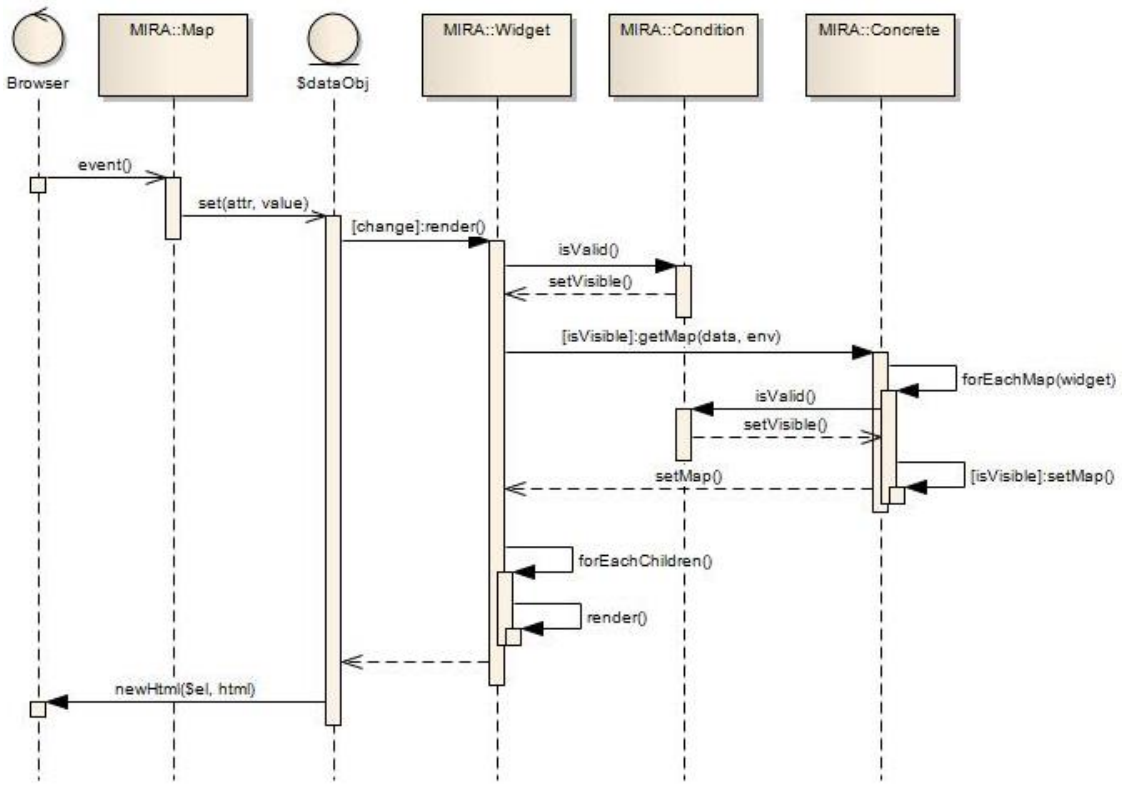

Figura 13 – Sequência de execução após mudança de valores.

Com esta forma de tratamento de eventos é possível criar a comunicação entre Widgets Concretos em tempo de exibição da página, capturando possíveis dependências entre eles. Por exemplo, num formulário de reserva de voos, o valor da data exibida no widget “data de regresso” deve ser no mínimo igual (ou até estritamente maior) que o valor da data no widget “data de partida”. Através dos eventos, é possível garantir que assim que o usuário selecionar uma data de partida, o widget correspondente à data de regresso já exibirá aquela data (ou uma data posterior, conforme ditar a regra de negócio).

A seguir, é apresentado um exemplo de uma lista de tarefas feitas utilizando a estrutura de eventos:

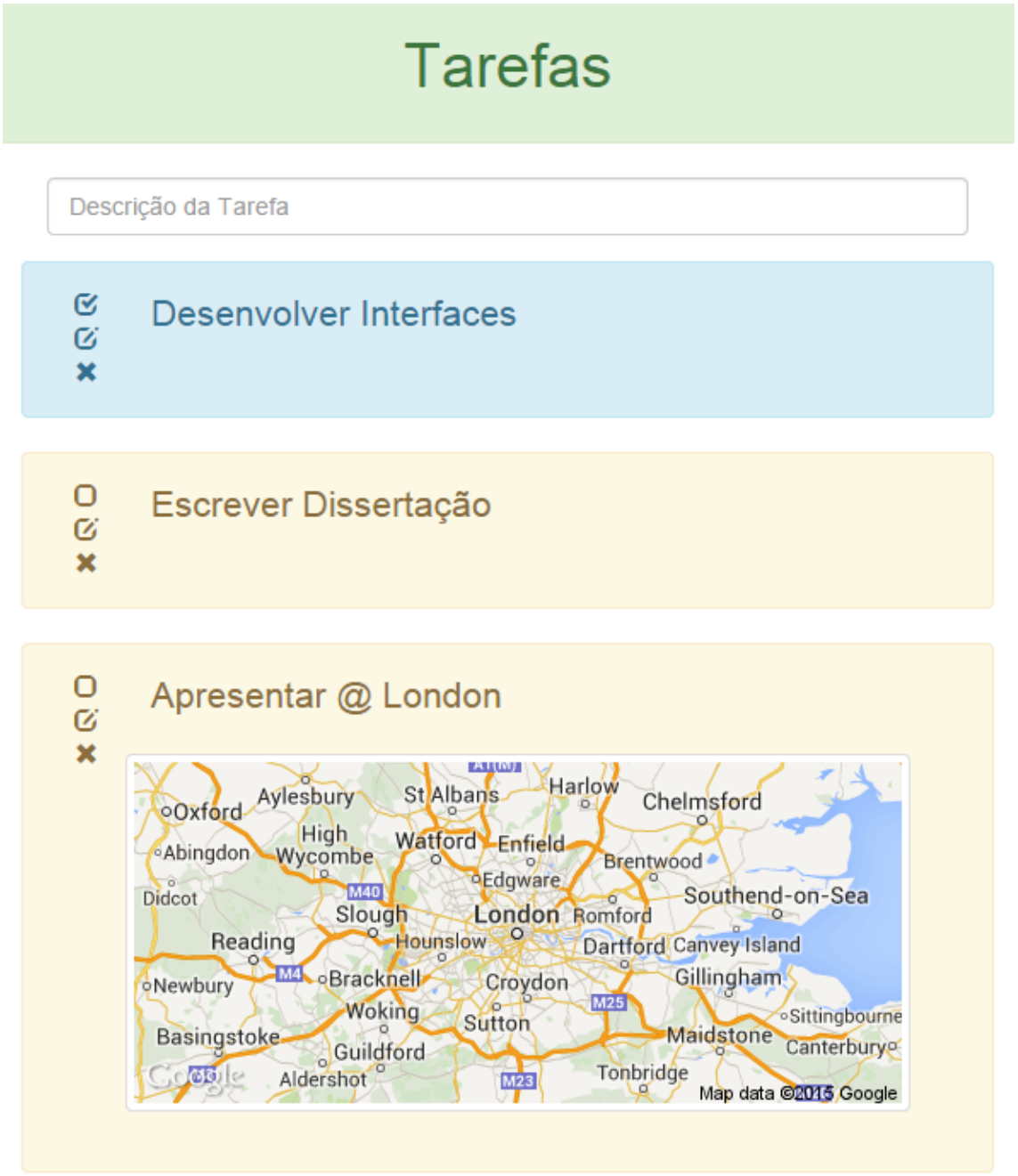

Figura 14 – Interface de tarefas exibida para o usuário

As informações que montam esta interface com uma lista de tarefa são estas:

```
var todo_list = [{
    tarefa: 'Desenvolver Interfaces', feito: true
},{
    tarefa: 'Escrever Dissertação', feito: false
},{
    tarefa: 'Apresentar @ London', feito: false
}];
```

Quadro 12 – Dados carregados na interface de tarefas

A interface de tarefas foi construída com a seguinte Interface Abstrata, representada graficamente para facilitar a compreensão:

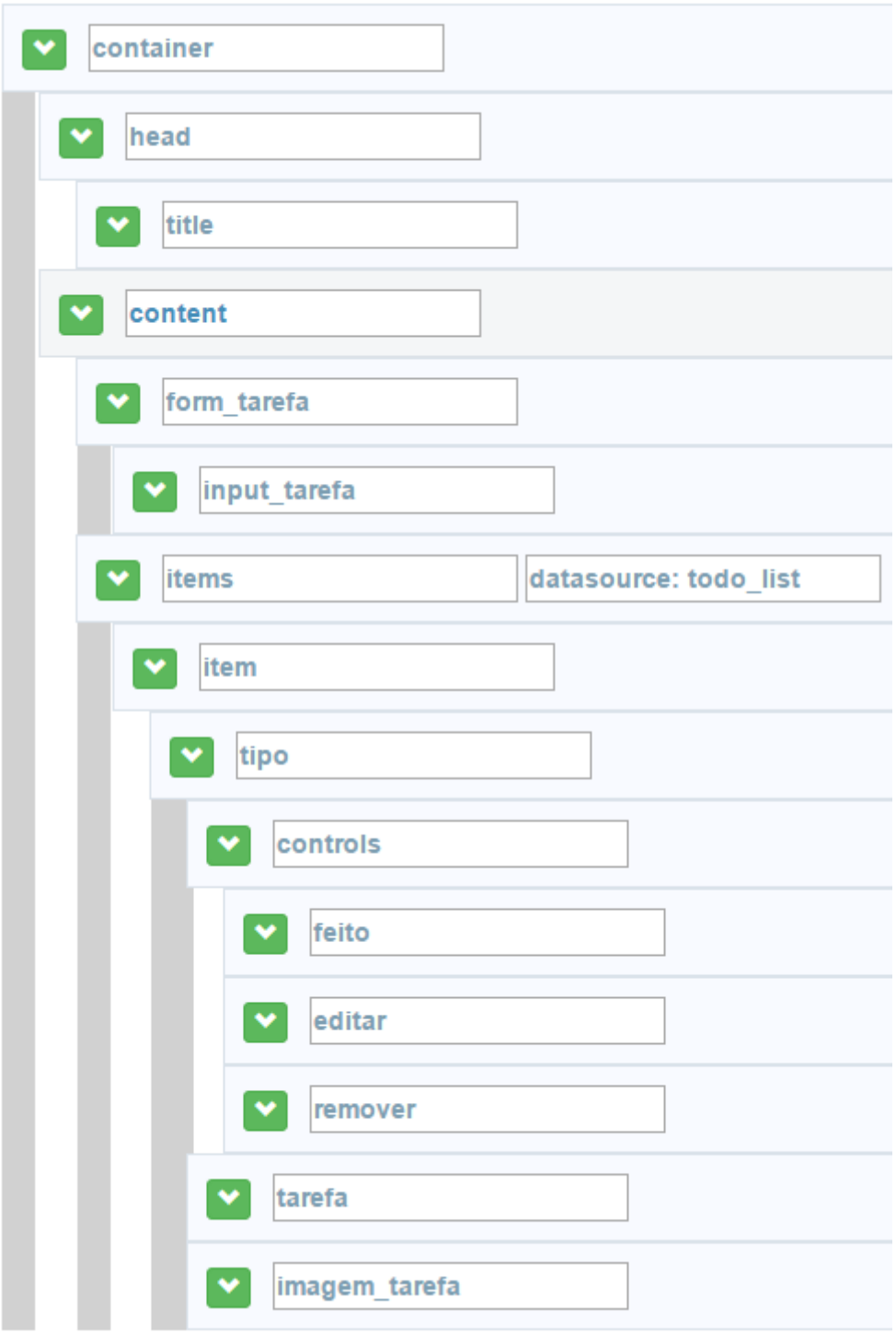

Figura 15 – Estrutura de widgets abstratos da interface de tarefas

Foram mapeados diversos Widgets concretos, em mapeamentos no qual o atributo *widget* não é informado, resultando no uso do widget padrão SimpleHtml.

```
{ name: 'container', class:'container' },
{ name: 'head', alert:'success' },
{ name: 'title', tag:'h1', text:'center', value:'"Tarefas"' },

{ name: 'content', class:'row', md:'6,offset-3' },
{ name: 'form_tarefa'},
{ name: 'input_tarefa', widget:'BootstrapFormControl',
    input:{type:'text', events:{ keydown:'adicionar' },
        placeholder:'Descrição da Tarefa'}
},
{ name: 'items' },
{ name: 'item', form:'horizontal' },
{ name: 'tipo', alert:"warning", class:'row' },
{ name: 'tipo', when:'isFeito', alert:"info", class:'row'},
{ name: 'controls', md:1 },
{ name: 'feito', widget:'BootstrapIcon', value:'unchecked',
    events: {click: 'marcar'}},
{ name: 'feito', when:'isFeito',
    widget:'BootstrapIcon', value:'check',
    events: {click: 'marcar'}},
{ name: 'editar', widget:'BootstrapIcon', value:'edit',
    events: {click: 'habilitar_edicao' }},
{ name: 'remover', widget:'BootstrapIcon', value:'remove',
    events: {click: 'remover'}},
{ name: 'imagem_tarefa', when:'hasLocation',
    widget:'MapStatic', size:'450x200', class:'thumbnail',
    value:'$data.tarefa.substring($data.tarefa.indexOf("@"))'},
{ name: 'tarefa', class:'lead', value:'$bind', md:'8',
    events: {click:'habilitar_edicao'}},
{ name: 'tarefa',  when:'inEdition',
    widget:'BootstrapFormControl', md:8,
    input:{ value:'$bind', events:{keydown:'editar'}}
}
```

Quadro 13 – Mapeamento de widgets concretos da interface de tarefas

No exemplo do Quadro 13 foi destacado a propriedade *events*, que são tratados pelas seguintes funções do Quadro 14:

```
event.marcar = function(options){
    options.$dataObj.set('feito', !options.$dataObj.get('feito'))
};

event.adicionar = function(options){
    if(options.$event.keyCode == 13 && options.$event.target.value) {
        todo_list.add({
            tarefa: options.$event.target.value,
            feito: false
        }, {at: 0});
    }
};

event.habilitar_edicao = function(options){
    if(!options.$dataObj.get('$edit')) {
        options.$dataObj.set('$edit', true);
    } else {
        options.$dataObj.set('$edit', undefined);
    }
};

event.editar = function(options){
    if(options.$event.keyCode == 13) {
        if(options.$event.target.value){
            options.$dataObj.set('tarefa', options.$event.target.value);
        } else {
            options.$dataObj.remove();
        }
        options.$dataObj.set('$edit', undefined);
    }
    if(options.$event.keyCode == 27){
        options.$dataObj.set('$edit', undefined);
    }
};

event.remover = function (options) {
    options.$dataObj.destroy();
}
```

Quadro 14 – Tratamento de eventos da interface de tarefas

Os eventos deste exemplo realizam alteração nos dados carregados na interface, e cada evento realiza as seguintes operações:

- **marcar***:* Alterna o valor do atributo *feito* do objeto, se for *true*, ele o transforma em *false*, e vice-versa.
- **adicionar**: Adiciona uma nova tarefa a lista de tarefas no momento que o usuário pressiona a tecla *enter*. Utiliza o valor digitado no widget *input_tarefa* como informação da tarefa.
- **habilitar_edicao**: Caso a tarefa possua o atributo de controle *$edit*, ele remove o atributo, caso não possua o atributo *$edit*, ele cria o atributo com o valor *true*.

- **editar:** Quando o usuário apertar a tecla *enter* e possuir algum valor no widget mapeado pelo evento, ele altera o valor do atributo tarefa do objeto. Se não possuir nenhum valor no widget, ele remove o objeto. Mas se o usuário apertar a tecla *esc*, ele cancela a edição do objeto.
- **remover**: Remove o objeto.

Os eventos não alteram diretamente os Widgets Concretos mapeados na interface, somente os dados carregados na interface, e a partir das seguintes condições:

```
var rules = [{
    name: 'isFeito', validate: '$data.feito == true'
},{
    name: 'hasLocation', validate: '$data.tarefa.indexOf("@") != -1'
},{
    name: 'inEdition', validate: '$data.$edit != null'
}];
```

Quadro 15 – Condições da interface de tarefas

A condição *isFeito* valida se a tarefa possui o atributo *feito* como verdadeiro, a regra *hasLocation* valida se na descrição da tarefa há o caractere *'@'* e a regra *inEdition* valida se foi declarada uma variável de controle na tarefa.

Será feita a troca de mapeamentos utilizados na interface, conforme ilustrado nas imagens a seguir:

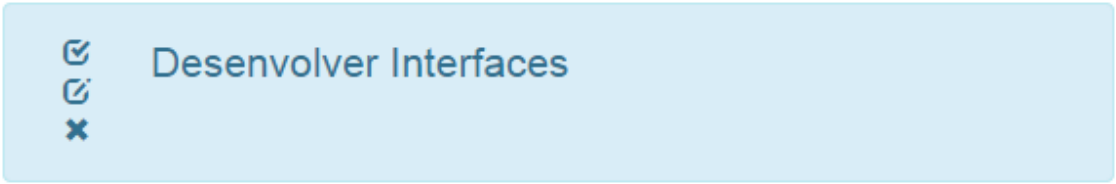

Figura 16 – Exemplo de tarefa feita

Se o widget *feito* for clicado, os widgets que possuem a condição *isFeito* serão alterados, como os widget *feito* e *tipo*, e o objeto passará a ser exibido desta maneira:

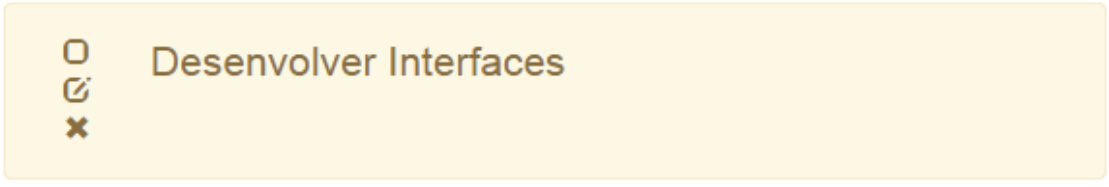

Figura 17 – Exemplo de tarefa não feita

Se o widget *editar* for clicado, os widgets que possuem a condição *inEdition* serão alterados, como o widget *tarefa*.

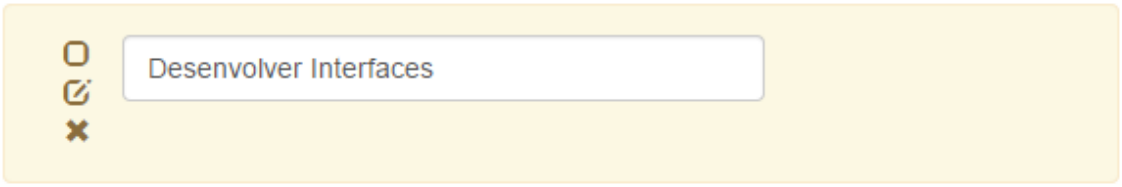

Figura 18 – Exemplo de tarefa em edição

Se o valor informado quando o usuário precisar a tecla *enter* possuir o caractere '@', os wigets que utilizam a condição *hasLocation* serão alterados, como o widget *imagem_tarefa*, que antes não possuía um mapeamento que habilitasse sua exibição na interface.

Figura 19 – Exemplo de tarefa com localização

Se for clicado novamente o widget *feito*, os widgets que possuem a condição *isFeito* serão alterados, como os widget *feito* e *tipo*, e o objeto passará a ser exibido desta maneira:

Figura 20 – Exemplo de tarefa feita com localização

E, finalmente, se o widget *remover* for clicado, o objeto deixará de existir e deixará de ser exibido na interface montada para o usuário.

4 ARQUITETURA E IMPLEMENTAÇÃO DO FRAMEWORK

O Framework MIRA é escrito utilizando a linguagem JavaScript [19] uma linguagem de padrão aberto do W3C originalmente feita para funcionar em modo *client-side* orientada a eventos, funcional, sem estrutura de módulos, e sem classes. No entanto, a utilização de frameworks como Backbone [15], que cria uma estrutura de classes e herança, e RequireJS [20], que cria uma estrutura organizada de módulos e dependências aproxima a programação em Javascript das linguagens OO mais tradicionais. Já com o uso de Node.js foi possível criar uma estrutura para o MIRA organizada em módulos, permitindo inclusive a distribuição da seleção de interfaces entre o servidor e orientada a objetos. Por estas razões, tem sido a linguagem de preferência para implementação de funcionalidades nos clientes Web.

4.1 ARQUITETURA

4.1.1 Organização em Módulos

Além das classes dos modelos de interface, o MIRA possui módulos e classes que controlam o comportamento da aplicação.

4.1.1.1 Application

O início da aplicação se dá com a instância desta classe. Ela é responsável por interpretar e armazenar os modelos das interfaces. Além de interpretar os parâmetros das URIs navegadas e iniciar o processo de seleção de interface. Construir o contexto utilizando nas avaliações de condições. Fazendo a requisição a WEB API e disparando a avaliação de regras da seleção de interface. Cada aplicação possui uma instância da classe aplicação para executar os modelos da interface.

4.1.1.2 Interface

Módulo Responsável por montar o HTML dos modelos de interface. Iniciando o processo de avaliação de widgets abstratos e avaliação de mapeamentos para widgets concretos.

4.1.1.3 Config

Módulo que contém o caminho para os arquivos de bibliotecas externas utilizadas pelo projeto.

4.1.1.4 Server

Módulo do servidor que é utilizado para criar um ambiente para desenvolvimento simples e também para validação das regras de seleção de interface distribuídas no servidor em produção.

4.1.1.5 Helper

Módulo com funções utilizadas em diversos módulos do MIRA e em suas extensões.

4.1.1.6 View

Módulo para isolar eventos e montar a exibição de cada objeto exibido na interface e facilitar o tratamento de eventos pelos widgets concretos. Diminuindo a carga de processamento ao alterar valores dos objetos exibidos na interface.

4.1.1.7 Init

Módulo com classes Base utilizada para herança de todos as classes da aplicação. Foi concebido utilizando o padrão de projeto GOF Prototype [21]

4.1.1.8 Render

Módulo responsável por indicar qual widget concreto será utilizado no mapeamento e realizar a execução de sua montagem na interface. Este módulo foi concebido utilizando o padrão de projeto GOF Flyweight [22], pois ele faz referência a todos os construtores de widgets concretos, que foram concebidos utilizando o padrão GOF Builder [23], assim evitando o uso excessivo de memória com referência de inúmeros mapeamentos utilizando o mesmo widget concreto.

4.1.2 Alternativas de Distribuição Cliente/Servidor

Todas as regras de interface do MIRA conseguem ser executadas unicamente no dispositivo que está acessando a aplicação feita com o framework. Mas em determinadas situações, o projetista da aplicação pode achar conveniente que suas regras sejam executadas no servidor da aplicação.

4.1.2.1 Seleção de Interface

A seleção de interface pode ser feita tanto no cliente que está navegando para uma URI, ou por um servidor. Este conceito foi criado com o intuito de diminuir a carga de processamento em situações onde a avaliação de interface poderia ser muito dispendiosa para o dispositivo que está acessando a aplicação.

4.1.3 Diagramas de Classe

4.1.3.1 Classes dos Modelos de Interface

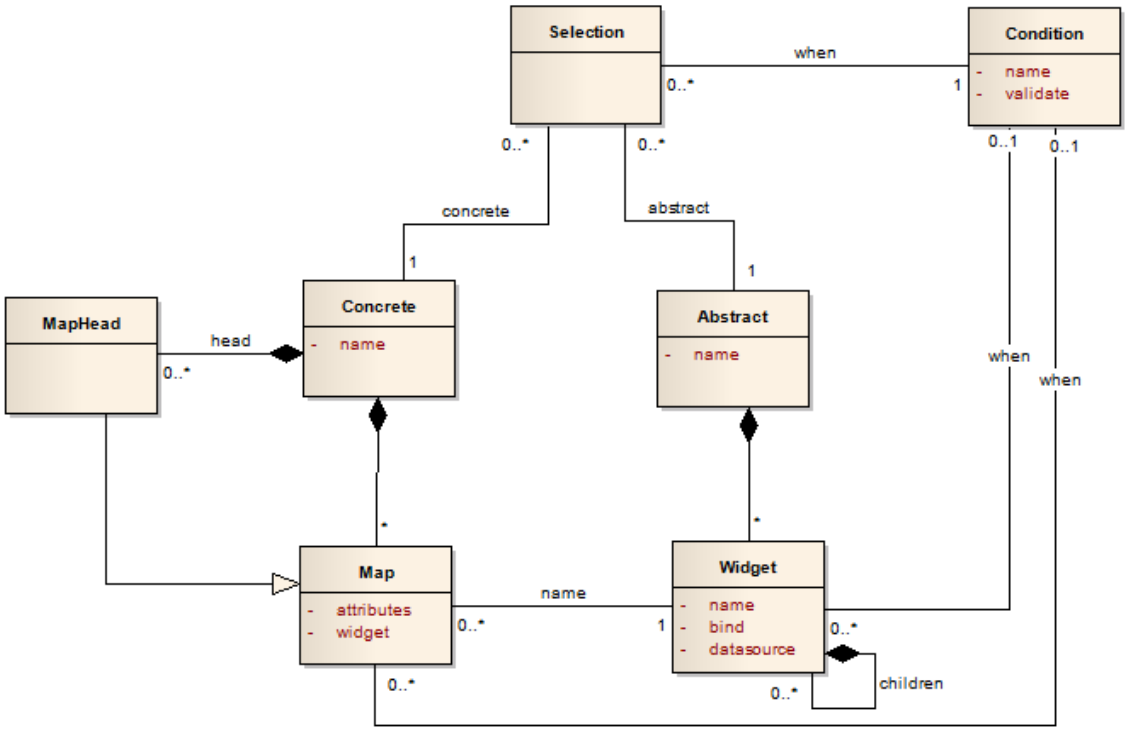

Figura 21 – Diagrama de classes dos modelos de interface do MIRA

4.1.3.2 Classes dos controladores do MIRA

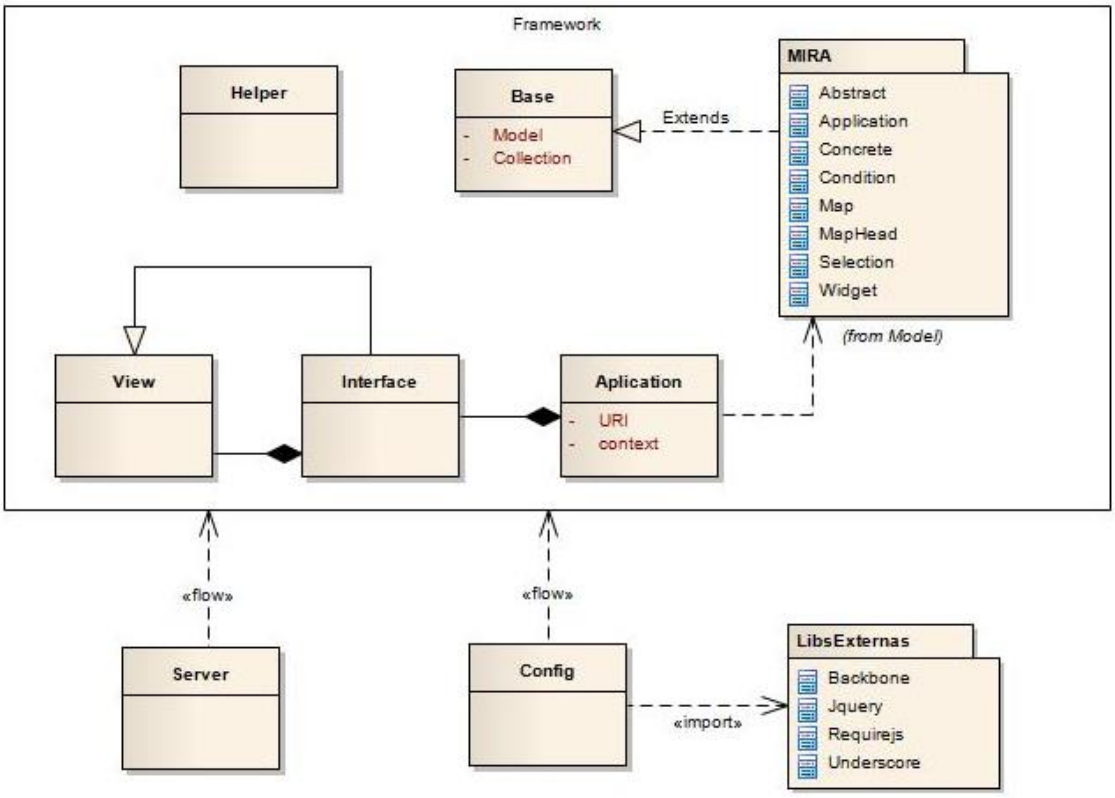

Figura 22 – Diagrama de classes dos controladores do MIRA

4.2 IMPLEMENTAÇÃO

O MIRA foi desenvolvido utilizando a linguagem de programação JavaScript, que foi escolhida por ser uma linguagem que funciona em navegadores, dispositivos móveis, televisões e, incluindo, ambiente de servidor utilizando Noje.js. Isto confere ao MIRA uma abrangência de uso para diversas plataformas.

4.2.1 Ambiente

O JavaScript foi concebido para ser executado somente em *client-side*, porém foi desenvolvido um ambiente de execução chamado Node.js que possibilita a execução de programas escritos em JavaScript em servidores de aplicação. Mas para tanto, há uma pequena diferença na escrita dos arquivos que compõem a aplicação.

Para que os mesmos arquivos que compõem do MIRA que executa em *client-side* possam ser executados no ambiente no servidor (*server-side*), foi necessária a utilização do padrão UMD (Universal Module Definition) [24].

O UMD consiste na união de dois padrões de desenvolvimento de módulos para JavaScript, o CommonJS [25] utilizado pelo Node.JS e o AMD (Asynchronous Module Definition) [26] utilizado pelo RequireJS. Assim, com uma simples modificação na forma de declarar os módulos, é possível executar o mesmo código tanto no cliente, quando no servidor de aplicação.

```
"use strict";
(function (root, factory) {
    if (typeof define === 'function' && define.amd) {
        define([
            'underscore',
            'backbone'
        ], factory);
    } else if (typeof exports === 'object') {
        module.exports = factory(
            require('underscore'),
            require('backbone')
        );
    }
}(this, function (_, Backbone) {

    var Model = Backbone.Model.extend({
        __name__: 'Base.Model'

    });

    var Collection = Backbone.Collection.extend({
        __name__: 'Base.Collection',

        model: Model
    });

    return {
        Model: Model,
        Collection: Collection,
    };
}));
```

Quadro 16 – Exemplo de uso do padrão UMD no módulo Base do MIRA

O método UMD verifica se o módulo está em ambiente de *client-side* ou *server-side* e faz a chamada da carga de módulos de acordo com cada padrão, RequireJS ou Node.js respectivamente.

4.2.2 Estrutura de arquivos

Os arquivos do MIRA são organizados em pastas conforme sua responsabilidade.

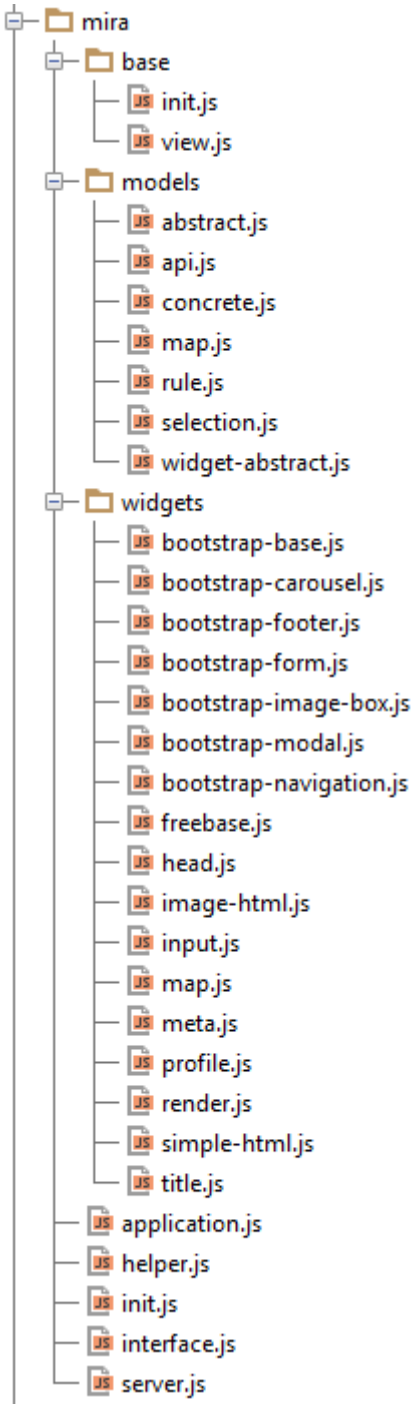

Figura 23 – Estrutura de pastas do MIRA

As pastas possuem as seguintes responsabilidades:

- **Mira:** Arquivos que controlem o comportamento do framework.
- **Base:** Arquivos que são herdados por outras classes de controle do framework.
- **Models:** Arquivos que controlam as informações dos modelos de interface e montam a estrutura do HTML exibido para o usuário.

- **Widgets:** Widgets Concretos disponibilizados pelo framework para construção de interface.

4.3 MÉTRICAS DO CÓDIGO

Foi utilizada a ferramenta **complexity-report** [27] [28] que analisa as seguintes métricas para estimar a qualidade de códigos escritos [29] em JavaScript:

- **Logical Loc** [30]**:** Contagem de linhas existentes em uma determinada porção de código fonte. Geralmente sumarizadas por métodos, classes ou assemblies, LOC nos dá de imediato uma clara visão do “tamanho” do software.
- **Parameter Count:** Analisa a quantidade de parâmetros das assinaturas das funções.
- **Cyclomatic Complexity** [31]: Caminhos (decisões) possíveis de execução dentro de um código. De fato, é uma métrica que expressa a complexidade (notadamente, utilizadas em métodos). Quanto maior este número, temos um código mais complexo de entender e de dar manutenção.
- **Halstead Effort** [32]**:** Conjunto de métricas que são calculadas com base no número de operadores distintos, o número de operandos distintos, o número total de operadores e o número total de operandos em cada função. A ferramenta utilizada para realizar a medição utiliza três medias de Halstead em particular:
    - **Dificuldade:**
      (# operadores dist. / 2) *
      (# operandos / # operadores dist.)
    - **Volume:**
      (# Operadores + # operandos) *
      log2 (# operadores dist. + # operandos dist.)
    - **Esforço:**
      dificuldade * volume

- **Maintainability Index** [33]: Calculada em todo o módulo a partir das médias de três outras métricas, utilizando a seguinte fórmula:
  171 -
  (3.42 * ln(esforço médio)) -
  (0.23 * ln(cyclomatic complexity médio)) -
  (16.2 * ln(Logical LOC médio))
- **First-order density:** A porcentagem de todas as possíveis dependências internas que são realizados no projeto.
- **Change cost:** O percentual de módulos afetados, em média, quando um módulo é alterado no projeto.
- **Core Size:** A percentagem de módulos que são amplamente dependem de si e dependem de outros módulos.

4.3.1 Todos os módulos do MIRA

Média por função de Logical LOC	7,00
Média por função de Parameter count	3,27
Média por função de Cyclomatic complexity	1,87
Média por função de Halstead effort	1548,43
Média por módulo Maintainability index	115,74
Change cost	3,23%

Tabela 1 - Resultado do complexity-report para todos os módulos do MIRA

4.3.2 Comparação entre Módulos

Métricas	**Mira**	**Base**	**Models**	**Widgets**
Média por função de Logical LOC	7,00	7,46	5,42	7,56
Média por função de parameter count	3,27	1,61	2,03	3,67
Média por função de cyclomatic complexity	1,87	2,06	1,68	2,02
Média por função de Halstead effort	1548,43	2108,47	1053,11	1812,88
Média por módulo Maintainability Index	115,74	112,19	120,21	114,00
Change cost	3,23%	50%	14,29%	5,88%

4.3.3 Gráficos de comparação

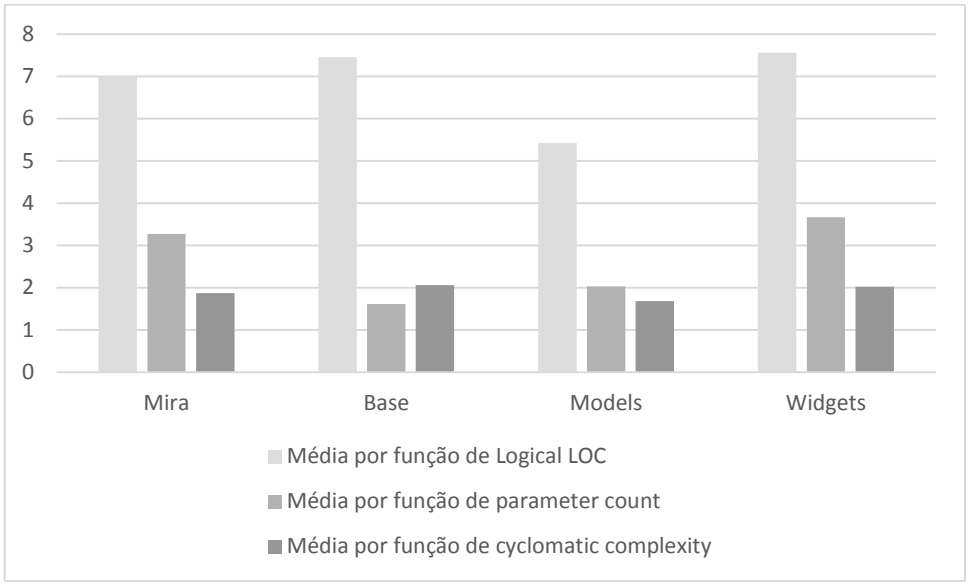

Gráfico 1 - Comparação entre métricas com medidas próximas

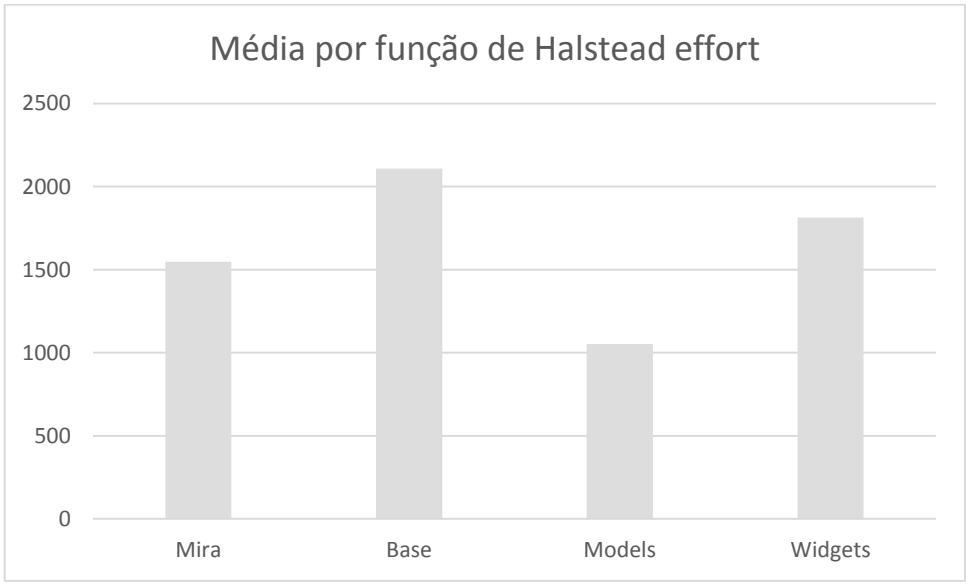

Gráfico 2 - Comparação de média por função com Halstead effort

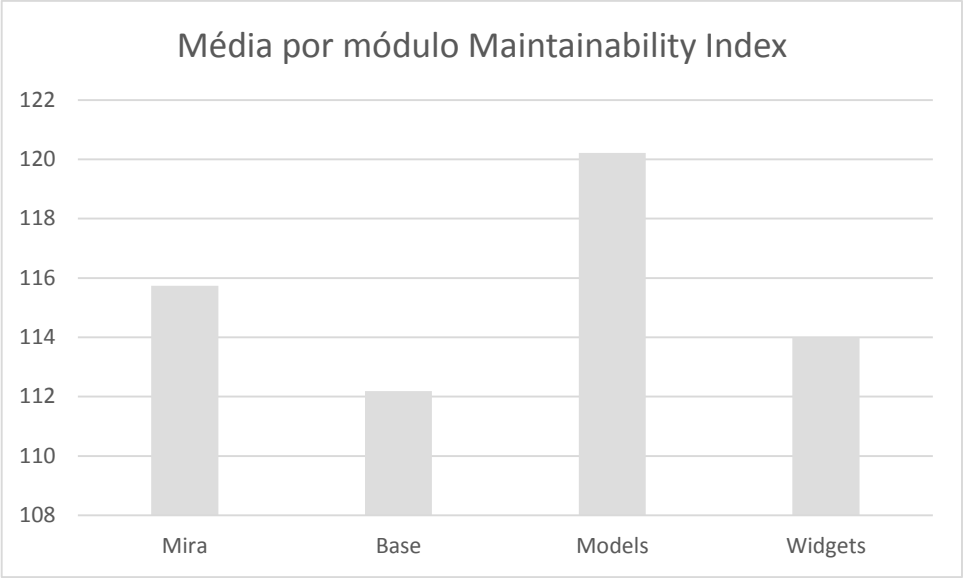

Gráfico 3 - Comparação entre módulos com Maintainability Index

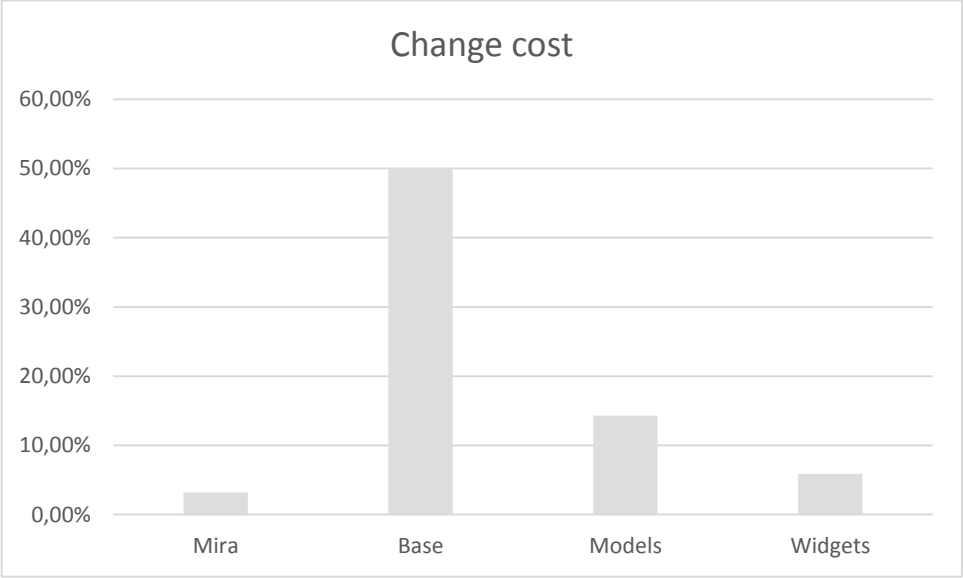

Gráfico 4 - Comparação entre módulos com base na métrica Change Cost

Com base nos resultados, concluímos que qualquer alteração nos módulos Base pode afetar praticamente todo o projeto, enquanto, todo o projeto possui um custo médio baixo de alteração. A dificuldade de manutenção nos models são as maiores e a complexidade dos códigos do Widgets são as maiores do Mira.

4.3.4 Métricas de cada arquivo

As métricas de cada arquivo podem ser vistas na seção de apêndice 9.3 Métricas de Código

4.3.5 Comparação com outros frameworks de interface

Métrica	MIRA	Angular	Backbone	Jquery
Média por função de Logical LOC	7	5,19	7,89	5,08
Média por função de Parameter count	3,27	1,55	1,27	1,43
Média por função de Cyclomatic complexity	1,87	2,26	3,25	3,13
Média por função de Halstead effort	1548,43	2907,37	5864,19	5986,7
Média por módulo Maintainability index	115,74	116,86	107,59	114,67

5 ESTUDO DE CASO E AVALIAÇÃO

A avaliação do MIRA foi realizada focando-se em dois aspectos. O primeiro visa determinar a expressividade da abordagem, e o segundo visa determinar a sua eficácia no uso por desenvolvedores. Para o primeiro aspecto, a avaliação foi feita através de estudos de caso, modelando-se uma interface real complexa encontrada no mercado. Para o segundo aspecto, foi feita uma avaliação qualitativa através de experimentos com desenvolvedores.

A seguir apresentamos cada uma delas.

5.1 ESTUDO DE CASO

Foi realizado um estudo de caso para avaliar a qualidade e aplicabilidade do MIRA, foram modelados vários exemplos de aplicação, com graus de complexidade variados. A seguir, apresentamos um estudo de caso onde avaliamos a construção de uma interface mais complexa baseada num site encontrado com uso de mercado, o Flickr[1].

5.1.1 Flickr

O Flickr é uma rede social onde é possível compartilhar fotos entre amigos e em grupos. Sua interface é rica em detalhes e o serviço possui uma API REST[2] que permite recriar grande parte de suas funcionalidades em uma nova interface.

[1] https://www.flickr.com

[2] https://www.flickr.com/services/api/

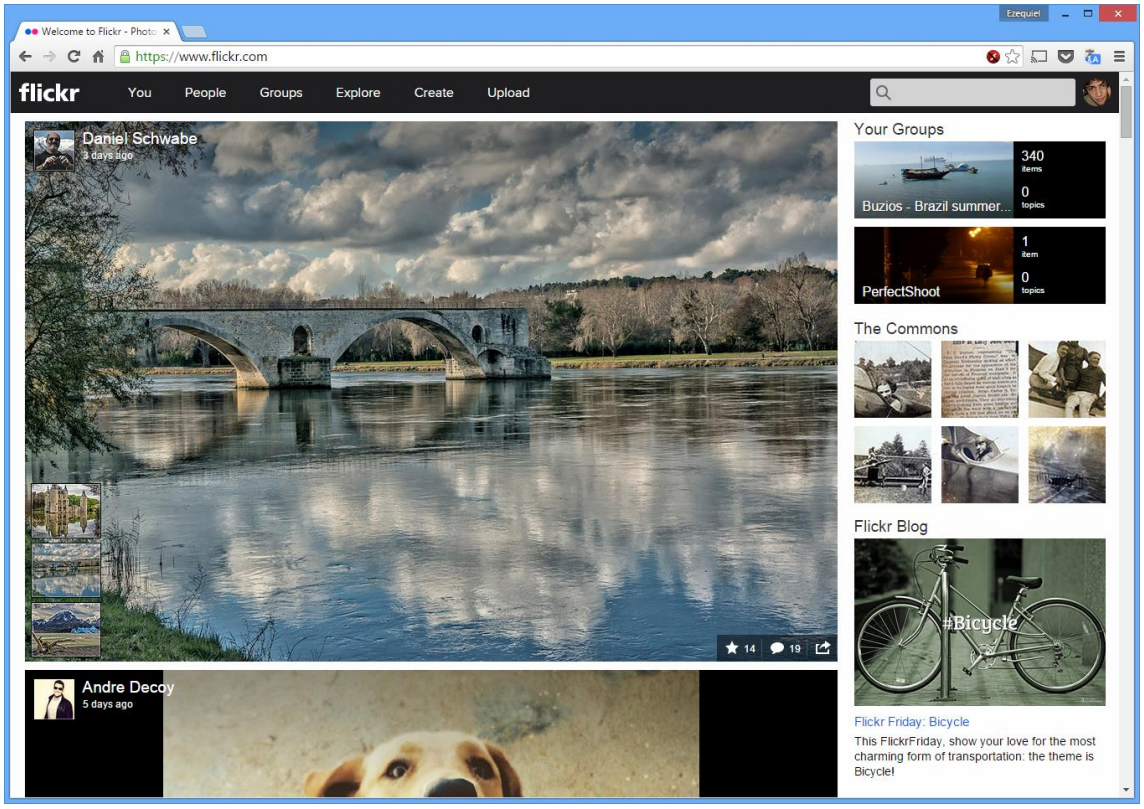

Figura 24 – Tela inicial do Flickr

O Flickr possuiu interfaces distintas, de acordo com o dispositivo que está acessando sua aplicação. Quando se está acessando uma URL do Flickr (http://www.flickr.com), ele avalia se é um smartphone ou se é um desktop ou tablet. Se for um smartphone, ele redireciona para uma URL com um domínio para smartphones (http://m.flickr.com).

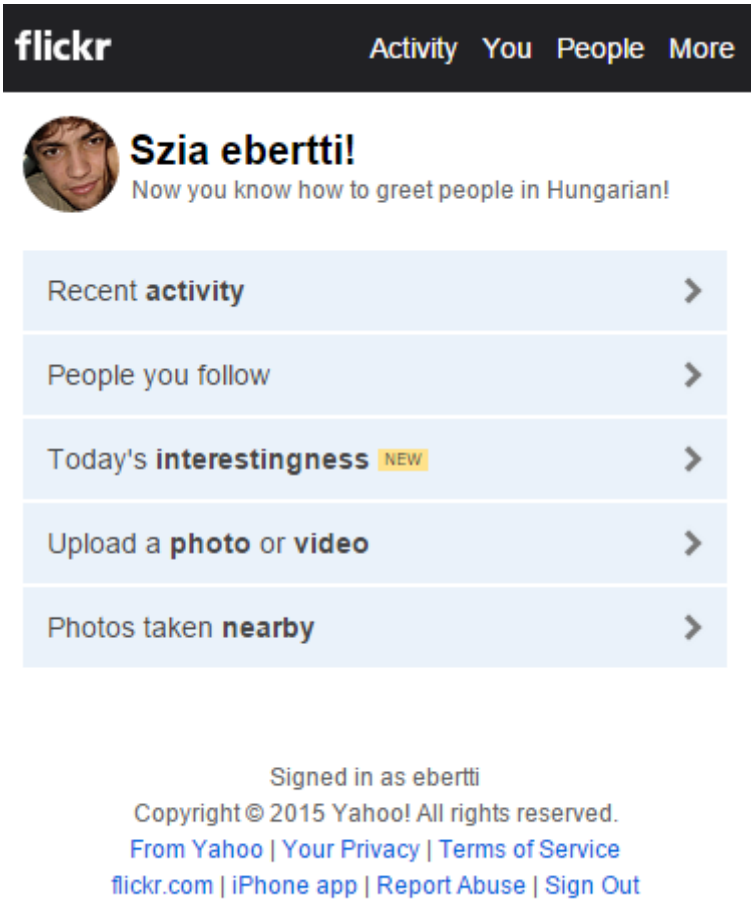

Figura 25 – Tela inicial do Flickr para smartphones

A segunda interface analisada é a de listagem de fotos de um usuário, na versão desktop ela exibe uma galeria de fotos com uma disposição de imagens variando de acordo com as dimensões de cada foto. Quando passa-se o mouse sobre uma imagem, são exibidas informações como nome da foto, quantidade de exibições e quantidade de comentários.

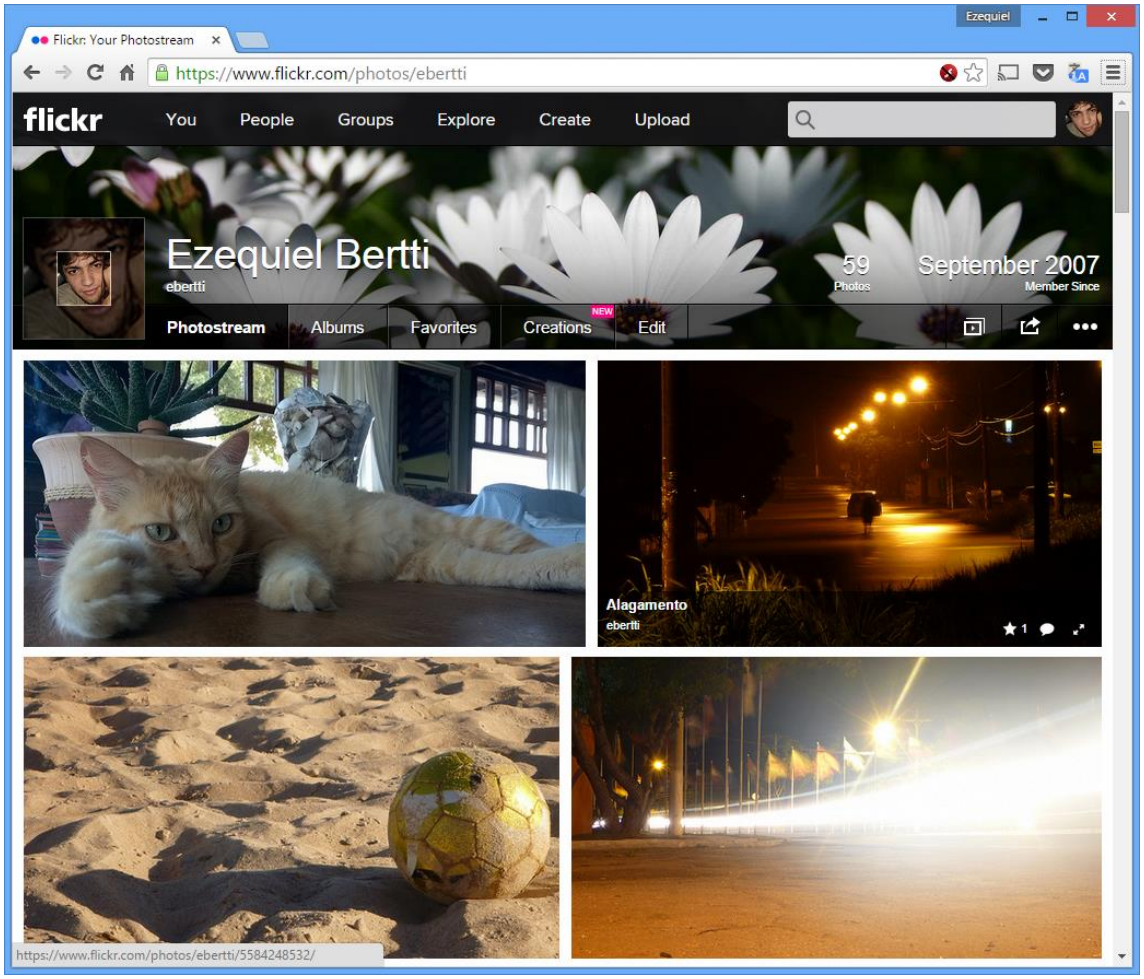

Figura 26 – Listagem de fotos de um usuário do Flickr

Já na versão mobile, são exibidas apenas miniaturas destas fotos no formato quadrado, com algumas informações como nome da foto, quantidade de visualizações e comentários.

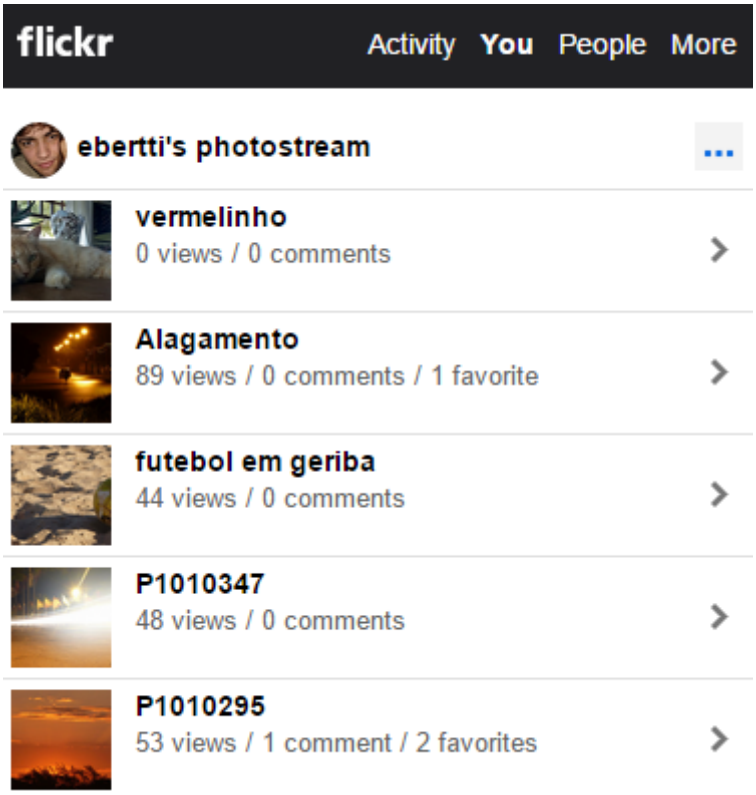

Figura 27 – Listagem de fotos de um usuário do Flickr em um smartphone

Na terceira interface analisada, mostramos a exibição em tamanho cheio de uma foto. Nesta interface são exibidas diversas informações sobre uma dada foto, entre elas: nome, descrição, comentários, pessoas que a curtiram, grupos que ela participa, tags, localização, quando e como a ela foi tirada (abertura, velocidade, ISO, flash). E se o usuário que está acessando esta interface for autor da foto, também é habilitada a possibilidade de adicionar tags, adicionar a grupos em que ele participa ou editar informações da foto como: nome, descrição, localização ou quando a foto foi tirada. Adicionalmente, se o usuário for moderador de algum grupo de fotos do Flickr, é habilitada a possibilidade de adicionar a foto a algum destes grupos.

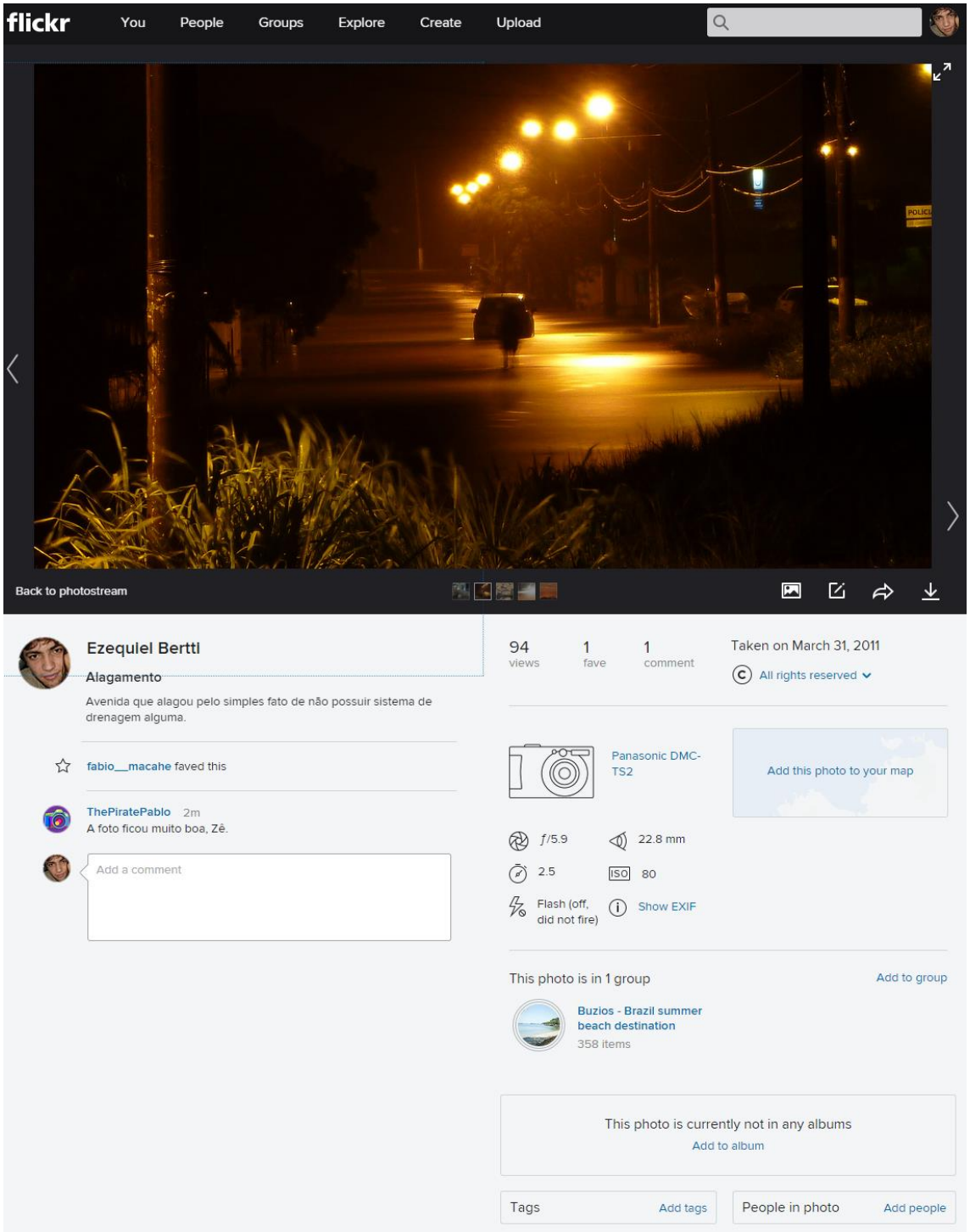

Figura 28 – Detalhes de uma foto no Flickr

Quando o usuário está visualizando as informações de uma foto no Flickr por um smartphone, as funcionalidades e as informações são limitadas apenas a exibição do nome, descrição, quantidade de curtidas, comentários e a uma listagem de miniaturas de outras fotos do usuário. Para realizar edições nas informações da foto, ele precisa navegar para outra interface.

Figura 29 – Detalhes de uma foto no Flickr em um smartphone

Além da exibição de imagens e coleções de fotos dos usuários, uma funcionalidade explorada neste estudo é o envio de fotos para grupos do Flickr. Esta funcionalidade só está disponível quando o usuário acessa o Flickr sem utilizar um smartphone.

A participação de um usuário num grupo depende de configurações do grupo como:

- Qualquer um pode participar de um grupo;
- Somente pessoas autorizadas podem participar de um grupo.

E para enviar fotos para estes grupos, um participante pode ser submetido a seguintes regras:

- Qualquer um pode publicar fotos no grupo;
- Somente moderadores podem publicar fotos no grupo;
- Apenas fotos autorizadas por moderadores podem ser publicadas no grupo.

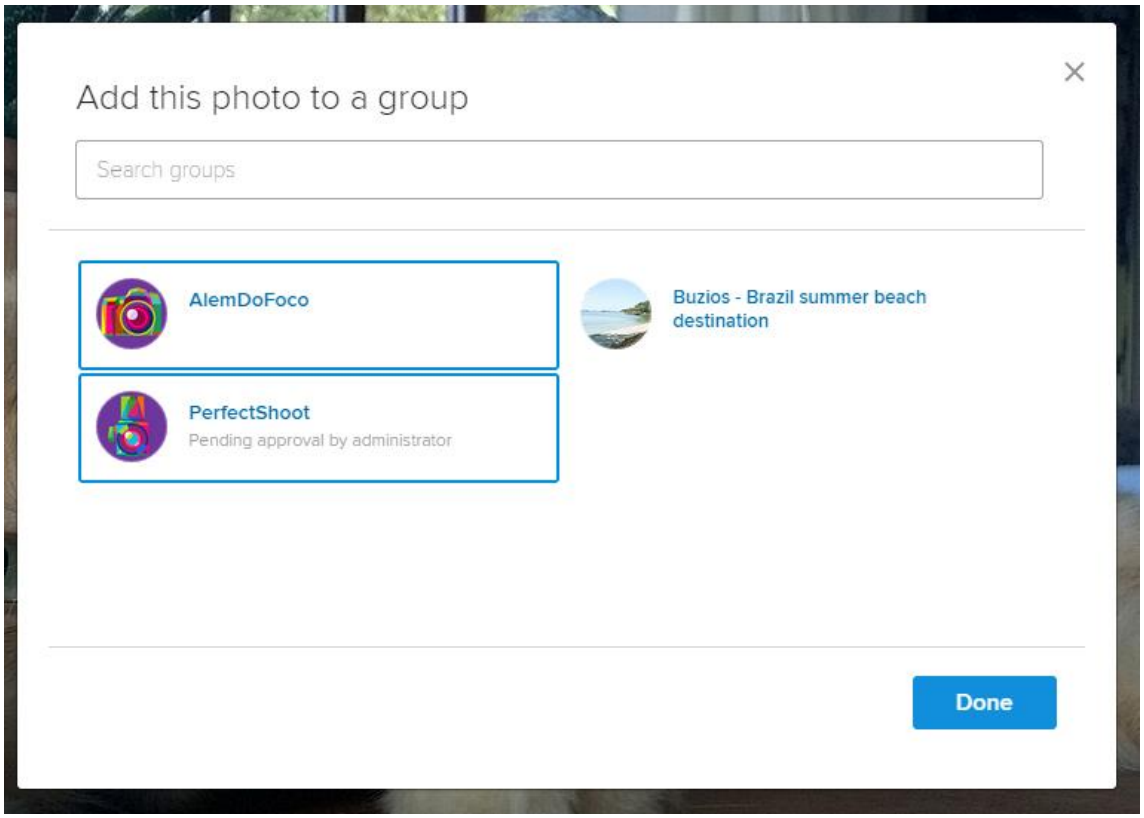

Figura 30 – Janela para adicionar fotos a um grupo

Quando a foto foi submetida a um grupo ou já faz parte de um grupo, a borda deste grupo fica em azul, se é necessária a aprovação da moderação do grupo, é exibida uma mensagem de “Pending approval by administrator” abaixo do nome do grupo após a submissão.

A partir desta análise das interfaces do Flickr, serão recriadas as mesmas interfaces com a mesma expressividade utilizando o MIRA.

5.1.2 Flickr com MIRA

Como forma de demonstrar a expressividade do MIRA, foram recriadas diversas interfaces do Flickr e alguns de seus comportamentos; estas interfaces criadas estão disponíveis online[3].

Foram levantadas Condições de Negócio, Seleção de Interface, Interface Abstrata, Interface Concreta, assim como a criação de Widgets Concretos Customizados para reconstruir a interface do Flickr.

[3] http://mira.tecweb.inf.puc-rio.br/?app=example/flickr

5.1.2.1 Condições de negócio

```
var conditions = [{
    name: 'isMe',
    validate: '$data.me == true'
},{
    name: 'isMobile',
    validate: '$env.device.mobile == true'
},{
    name: 'notMobile',
    validate: '$env.device.mobile == false'
},{
    name: 'isPerson',
    validate: '$data.person != null'
},{
    name: 'isPhoto',
    validate: '$data.type == "photo"'
},{
    name: 'actualPhoto',
    validate: '$env.$data.id == $data.id'
},{
    name:'isUserPhoto',
    validate: '$env.$login.user_nsid == $data.owner.nsid'
},{
    name:'groupToAdd',
    validate: '($data.admin == 1 || $data.moderador == 1) || ' +
              '$env.$login.user_nsid == $env.$data.owner.nsid'
},{
    name:'needApprove',
    validate: '$data.privacy == 2'
},{
    name:'waitingApprove',
    validate: '$data.$status == 2'
},{
    name:'approved',
    validate: '$data.$status == "ok"'
},{
    name:'groupShow',
    validate: '$data.$hide == null'
},{
    name:'hasPreviousPhoto',
    validate: '$data.prevphoto.id != 0'
},{
    name:'hasNextPhoto',
    validate: '$data.nextphoto.id != 0'
}];
```

Quadro 17 – Condições da aplicação do Flickr utilizando o MIRA

Dentre estas condições, temos condições utilizadas na Seleção de Interface, como:

- Seleção de Interface: *isPhoto, isPerson* e *isMe*
- Dispositivo: *isMobile* e *notMobile*
- Atributos da Foto: *isUserPhoto*, *hasPreviousPhoto e hasNextPhoto*
- Iteração com Grupo: *groupToAdd, needApprove, waitingApprove, approved* e *groupShow*

5.1.2.2 Interface de Timeline

A primeira interface recriada utilizando o MIRA foi a interface de linha do tempo (*timeline*) que aparece para o usuário assim que ele acessa o site do Flickr

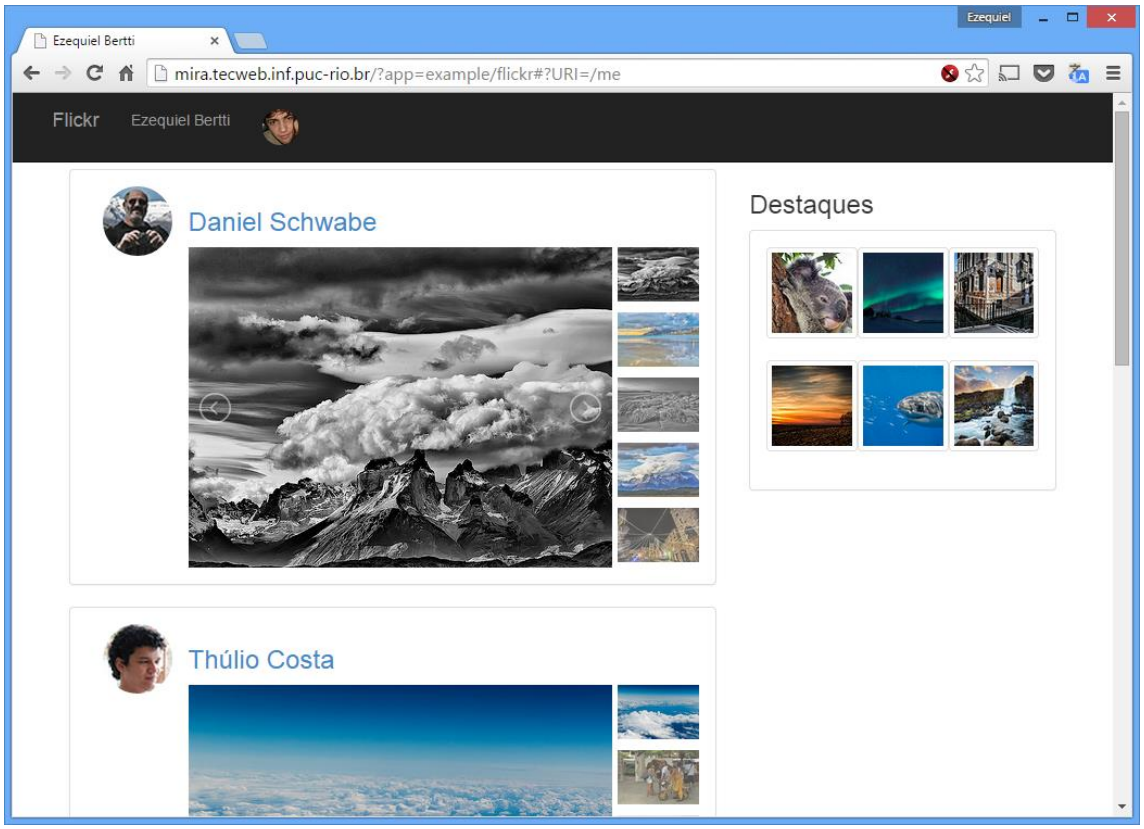

Figura 31 – Tela de timeline do Flickr construída com o MIRA

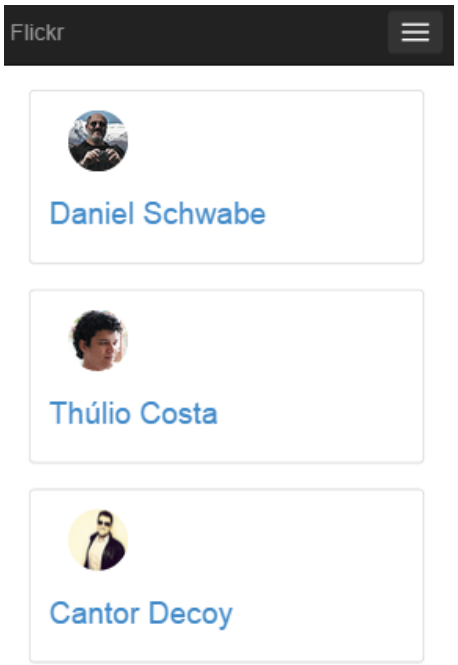

Figura 32 – Interface de timeline do Flickr para smartphone construída com o MIRA

Para construir esta interface, iniciamos com a seleção de interface abstrata e concreta:

5.1.2.2.1 Seleção de interface

```
var selection = [
    {
        when: 'isMe',
        abstract: 'timeline'
    },
    // outras //
];
```

Quadro 18 – Seleção de interface para timeline da aplicação do Flickr usando o MIRA

Nesta regra está sendo verificado se o dado retornado pela API atende a condição *isMe*, que testa se o dado retornado pela API possui o atributo *me* com o valor *true*. Caso seja verdade, a interface abstrata e concreta (por default de mesmo nome) com o nome de *timeline* será montada para o usuário.

5.1.2.2.2 Interface Abstrata

A mesma interface abstrata é utilizada para montar a interface para dispositivos como smartphones e para desktops e tablets.

```
{
name: "timeline",
widgets: [
    { name: 'navigation',
        children: [
            { name:'navigation-list',
                children:[
                    {name:'navigation-list-item'},
                    {name:'navigation-avatar',
                      children:[
                          {name:'navigation-avatar-img'}
                      ]
                    }
                ]
            }]
    },{
        name: "content",
        children: [{
            name: "amigos",
            parse:'$data.data',
            datasource: "url:/me/following",
            children: [{
                name: "amigo_box",
                children: [
                    {name: 'amigo_link',
                     children:[
                         {name: "amigo_avatar"},
                         {name: "amigo_name"}
                     ]},
                    {name: "slide_gallery",
                     parse:'$data.data',
                     datasource:"url:/photos?user_id=<%= $data.nsid %>",
                     children: [
                        {name: "slide_item",
                         children:[
                             {
                                 name: "slide_desc",
                                 children: [
                                     {name: "slide_share"},
                                     {name: "slide_title"}
                                 ]
                             }
                         ]
                        }
                     ]}
                ]
```

Quadro 19 – Modelo de interface abstrata para aplicação do Flickr usando o MIRA

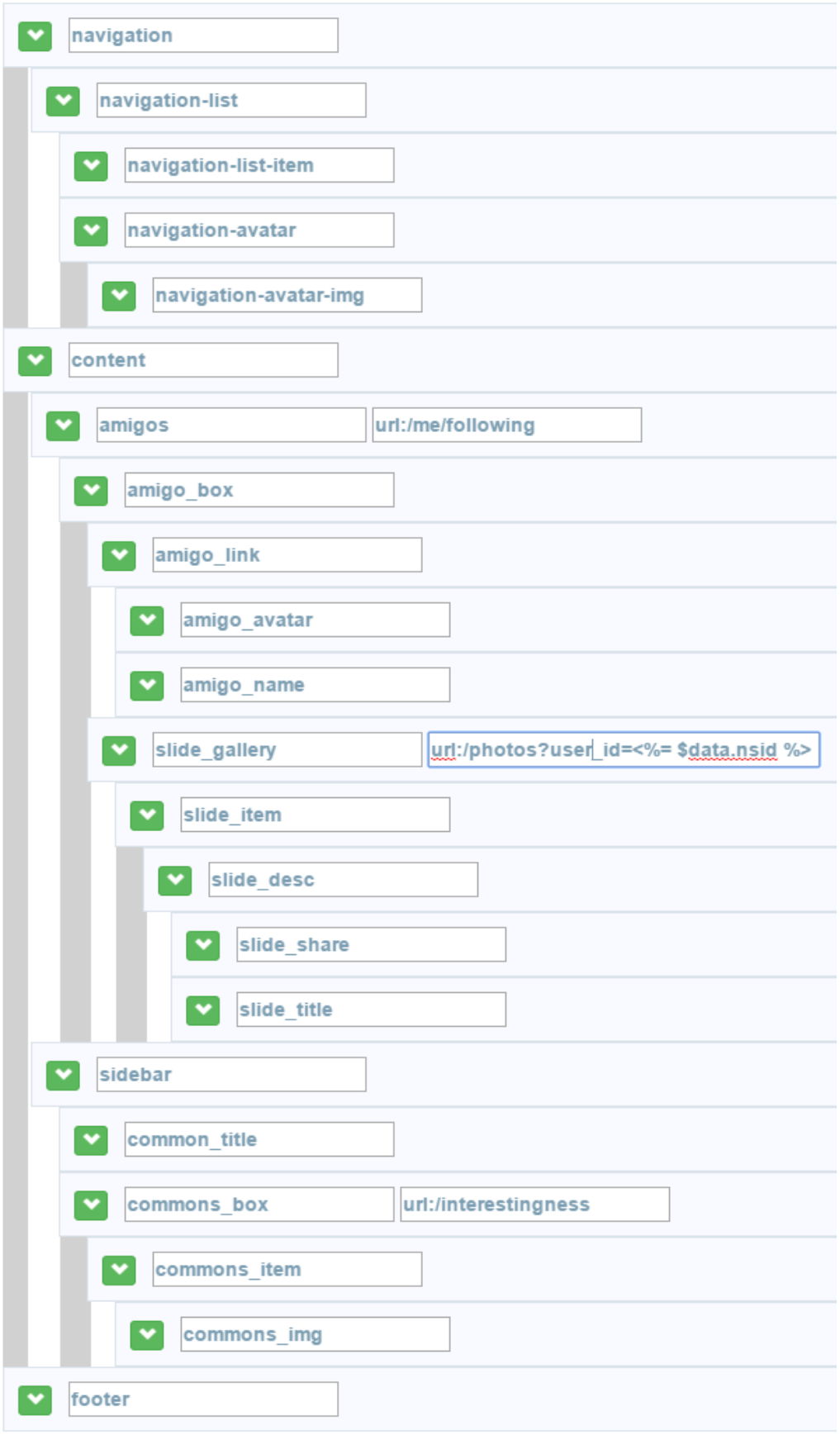

Figura 33 – Modelo de interface abstrata da aplicação do Flickr utilizando o editor de interface

5.1.2.2.3 Interface Concreta

A Interface Concreta da *timeline* do Flickr possui apenas alguns mapeamentos de Widgets Concretos que utilizam regras para se adaptar de acordo com tipo de dispositivo que está acessando a aplicação. Estes mapeamentos estão destacados na

```
{
name: "timeline",
head: head.concat([
    {name: 'title', widget:'Title', value: '$data.realname._content'},
    {name: 'galery_css', widget:'Head', tag:'style',
        href:'js/example/flickr/lightSlider.css'}
]),
maps: [
    { name: 'navigation', widget: 'BootstrapNavigation', value:'"Flickr"'},
    { name: 'navigation-list', widget: 'BootstrapNavigationList'},
    { name: 'navigation-list-item', widget: 'BootstrapNavigationListItem',
        value:'$env.$login.fullname', href:'navigate("/me")'},
    { name: 'navigation-avatar', widget: 'BootstrapNavigationListItem',
        tag:'a', class:'navbar-brand',
        href:'navigate("/user?user_id=" + $env.$login.user_nsid)'},
    { name: 'navigation-avatar-img', tag:'img', class:'img-circle',
        src:'$env.$login.info.picture', width:"33", height:'33'},
    { name: "busca" },
    { name: "foto", tag:'img', src:'' },
    { name: "content", class:'container' },
    { name: "amigos", md:8 },
    { name: "amigo_box", widget:'BootstrapPanelBody', class:'panel-default'},
    { name: "amigo_box_body", class:'panel-body'},

    { name: "amigo_link", tag:'a',
        href:'navigate("/user?user_id=" + $data.nsid)'},
    { name: "amigo_avatar", tag: 'img', md:"2", img:'circle,responsive',
        src:'$data.thumbnail'},
    { name: "amigo_name", tag:'h3', value:'$data.realname' },

    { name: "slide_gallery", widget:'FlickrGallery', when:'notMobile' },
    { name: "slide_item", widget:'FlickrGalleryItem',
        'data-thumb':'$data.thumbnail', img:{ src:"$data.picture"},
        link: { href:'navigate("/photo?photo_id=" + $data.id)' } },
    { name: "slide_desc",  class:"lslide-description" },
    { name: "slide_title", tag:'h4', value:'$data.title' },
    { name: "slide_share", widget:'BootstrapIcon', class:'pull-right',
        icon:'share' },

    { name: 'sidebar', md:4 },
    { name: 'commons_box', widget:'BootstrapPanelBody', class:'panel-default' },
    { name: 'common_title', tag:'h3', value:'"Destaques"' },
    { name: 'commons_item', tag:'a',
        href:'navigate("/photo?photo_id=" + $data.id)'  },
    { name: 'commons_img', tag: 'img', md:"4", class:'thumbnail',
        src:'$data.size.square' },
    { name: 'commons_img', tag: 'img', img:'responsive', class:'thumbnail',
        src:'$data.size.s320', when:'isMobile' },
```

Quadro 20 – Interface concreta da interface timeline da aplicação do Flickr utilizando o MIRA

Como podemos, notar nesta interface concreta, o mapeamento para o Widget Abstrato *slide_galery* só ocorre quando a condição *notMobile* é

verdadeira, logo, se a condição for falsa, nenhum dos Widgets que são filhos dele serão montados na interface do usuário.

Outro Widget Abstrato tem seu mapeamento alterado com a condição *isMobile* é o *commons_img,* mas ao contrário do que acontece no Widget *slide_galery*, temos um mapeamento sem regra acontecendo, então, se a condição *isMobile* for falsa, ele continuará sendo montado para o usuário.

5.1.2.2.4 Widget Concretos Customizados

Para representar esta interface com toda a expressividade apresentada pelo Flickr, foram construídos 2 Widget Concretos Customizados.

- **FlickrGallery:** Um componente JavaScript que envolve uma lista de fotos e realiza trocas com o clique e botões de próximo e anterior, ou com o clique na miniatura das imagens listadas.
- **FlickrGalleryItem:** São itens que compõem o Widget Concreto FlickrGallery. Este recebe como parâmetros a URL de uma imagem grande e de uma miniatura, e o link que deve ser navegado caso a foto que está sendo exibida cheia seja clicada.

5.1.2.3 Interface de fotos de um usuário

Para a interface de usuário foi criado um Widget Concreto Customizado para ajudar a redimensiona-las, conforme as imagens estejam sendo exibidas em um desktop ou tablet.

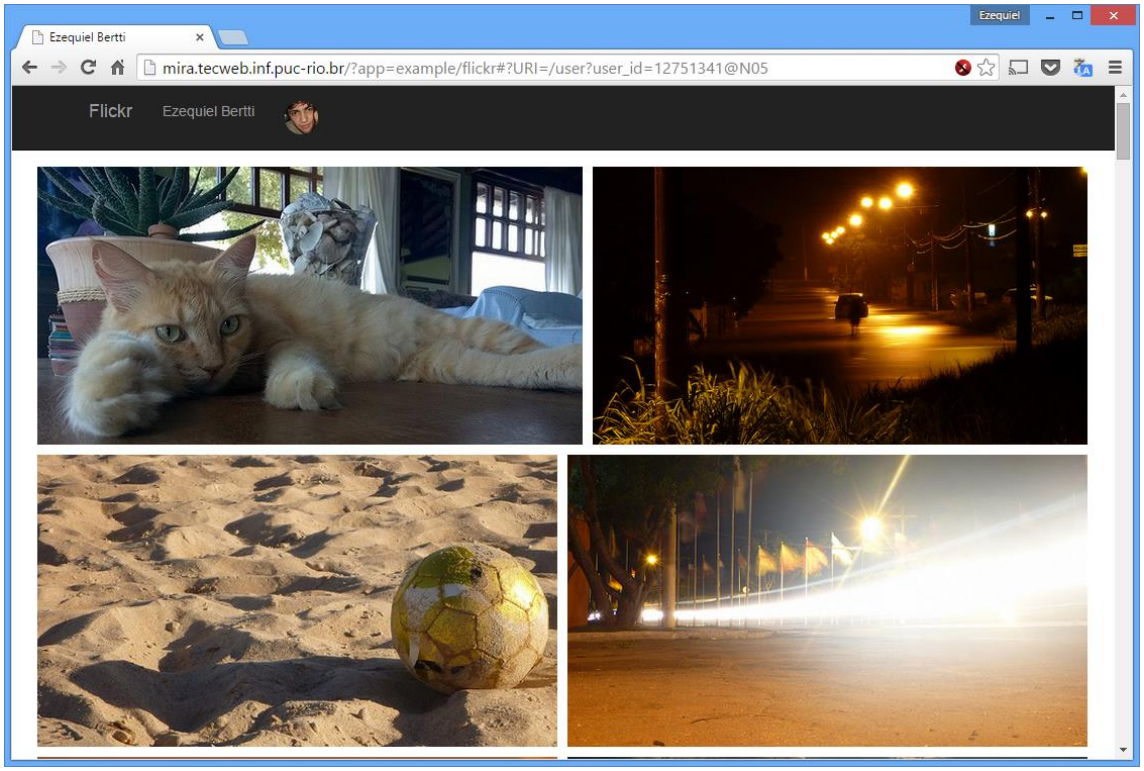

Figura 34 – Interface de fotos do usuário do Flickr construída com o MIRA

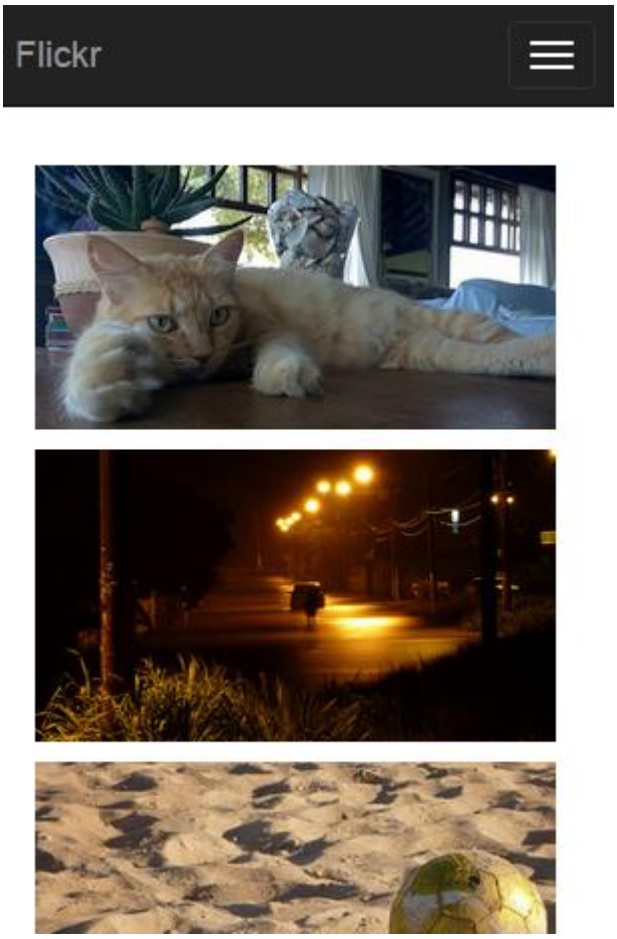

Figura 35 – Interface de fotos do usuário do Flickr construída com o MIRA para smartphone

5.1.2.3.1 Seleção de Interface

Para iniciar a construção desta interface, começamos com a regra de seleção de interface:

```
var selection = [
    {
        when: 'isPerson',
        abstract: 'user'
    },
];
```

Quadro 21 – Seleção de interface para user da aplicação do Flickr usando o MIRA

Nesta regra está sendo checado se o dado retornado pela API atende a condição *isPerson*, que testa se o dado retornado pela API possui algum valor no atributo *person*. Caso seja verdade, a interface abstrata e concreta com o nome de *user* será montada para o usuário.

5.1.2.3.2 Interface Abstrata

Como aconteceu na interface da timeline, será utilizada a mesma interface abstrata para smartphones e para desktops e tablets.

```
{   name:'user',
    widgets: [
        { name: 'navigation',
            children: [
                { name:'navigation-list',
                    children:[
                        {name:'navigation-list-item'},
                        {name:'navigation-avatar',
                            children:[ {name:'navigation-avatar-img'}]}]}]},
        {name: "content",
         children: [{
                name: "photos_box",
                datasource: "url:/photos?user_id=<%= $data.person.id %>”,
                parse:'$data.data',
                children: [{
                    name: "photo_item",
                    children: [
                        {name: "photo_title"},
                        {name: "photo_src"}]}]}]
        },{name: "footer"}]
}
```

Quadro 22 – Modelo de interface abstrata para a interface user do Flickr usando o MIRA

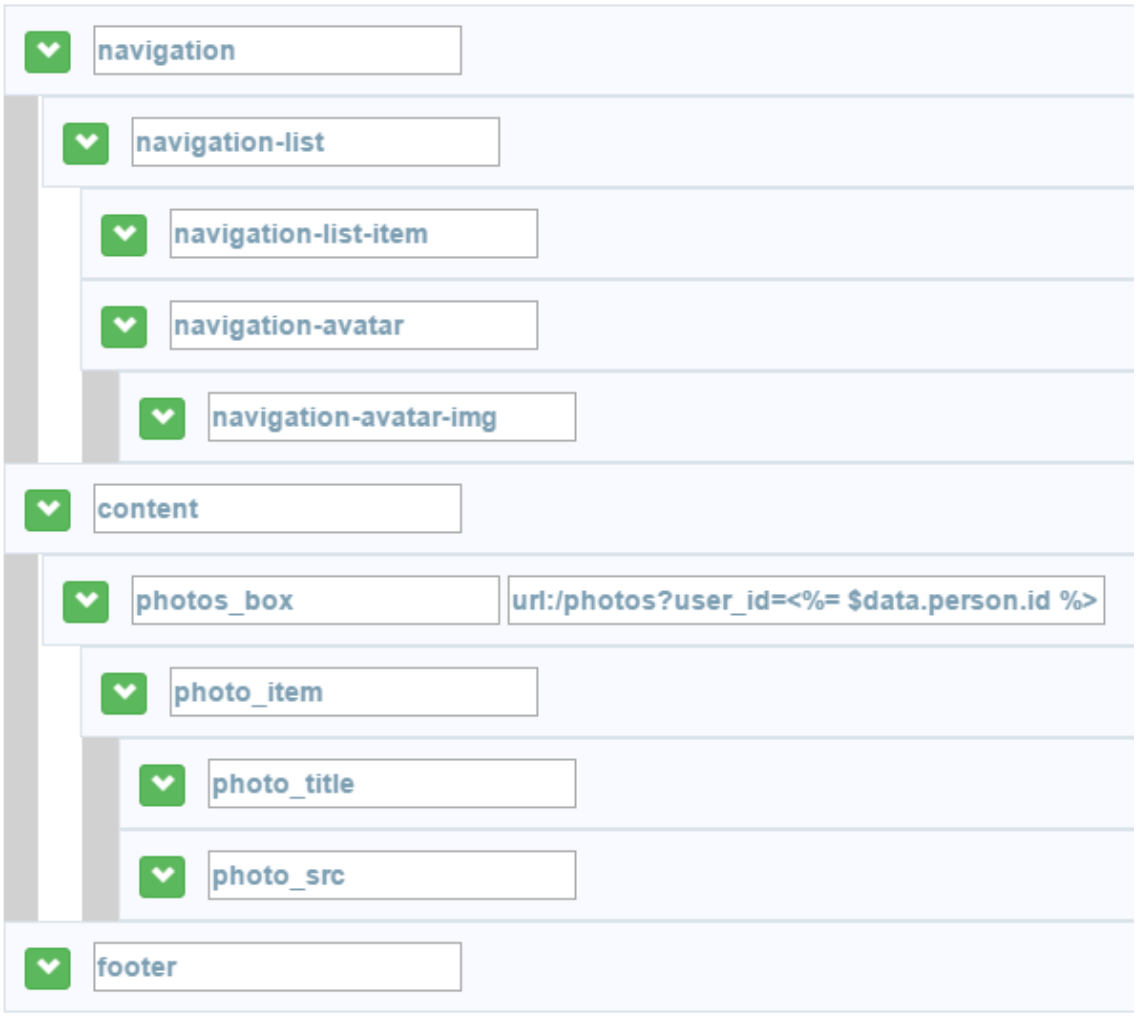

Figura 36 – Estrutura da interface abstrata da interface user do Flickr usando o MIRA

5.1.2.3.3 Interface Concreta

Nesta interface, apenas um Widget Abstrato precisou ter seu mapeamento alterado para se adaptar a um smartphone.

```
{
name: 'user',
head: [
    {name: 'title', widget:'Title', value: '$data.person.realname._content'}
],
maps:[
    { name: 'navigation', widget: 'BootstrapNavigation', value:'"Flickr"'},
    { name: 'navigation-list', widget: 'BootstrapNavigationList'},
    { name: 'navigation-list-item', widget: 'BootstrapNavigationListItem',
        value:'$env.$login.fullname', href:'navigate("/me")'},
    { name: 'navigation-avatar', widget: 'BootstrapNavigationListItem', tag:'a',
        class:'navbar-brand',
        href:'navigate("/user?user_id=" + $env.$login.user_nsid)'},
    { name: 'navigation-avatar-img', tag:'img', class:'img-circle',
        src:'$env.$login.info.picture', width:"33", height:'33'},
    { name: 'content', class:'container-fluid'},
    { name: 'photos_box', widget:'FlickrCollage'},
    { name: 'photo_item', tag:'a',
        href:'navigate("/photo?photo_id=" + $data.id)' },
    { name: 'photo_title' },

    { name: 'photo_src', tag:'img', src:'$data.size.s500'},
    { name: 'photo_src', when:'isMobile',
        tag:'img', src:'$data.size.s320', img:'responsive' }
]}
```

Quadro 23 – Interface concreta para a interface de user do Flickr usando o MIRA

O Widget *photo_src* foi mapeado com o mesmo Widget Concreto, porém a URL da foto informada como *source* da imagem foi uma foto de tamanho menor.

5.1.2.3.4 Widgets Concretos Customizados

Para esta interface foi construído um Widget Concreto Customizado chamado FlickrCollage, que redimensiona as fotos para que elas se encaixem ao lado uma das outras, produzindo o efeito que todas elas possuem a mesma altura.

5.1.2.4 Interface de detalhes de uma foto

Esta interface, além de exibir diversas informações de uma foto, possui as interações de publicação em grupos.

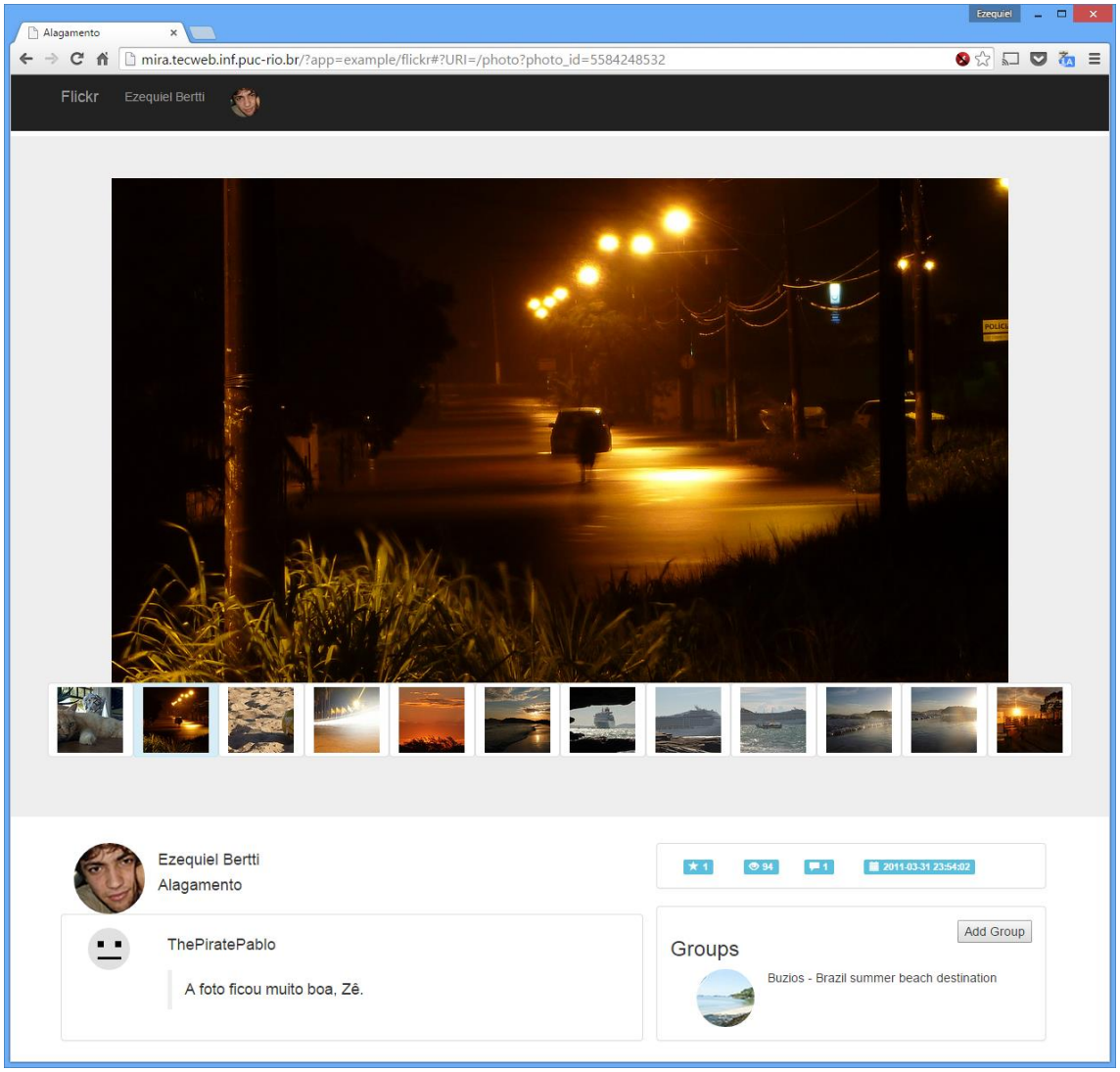

Figura 37 – Interface de detalhes de uma foto no Flickr construída com o MIRA

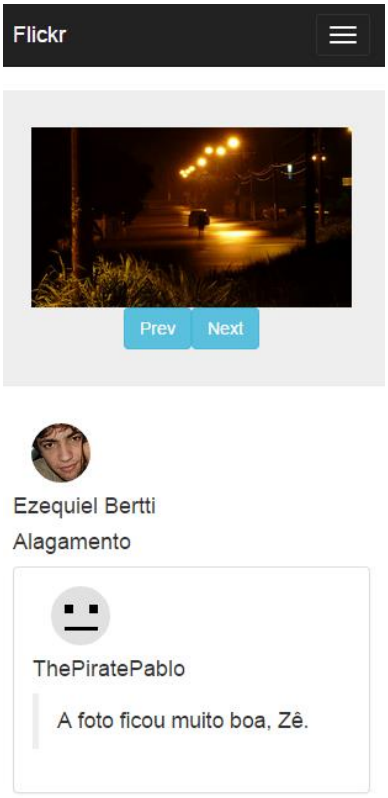

Figura 38 – Interface de detalhes de uma foto no Flickr construída com o MIRA para smartphone

5.1.2.4.1 Seleção de Interface

Para iniciar a construção desta interface, começamos com a regra de seleção de interface:

```
var selection = [
    {
        when: 'isPhoto',
        abstract: 'photo'
    },
];
```

Quadro 24 – Seleção de interface para user da aplicação do Flickr usando o MIRA

Nesta regra está sendo verificado se o dado retornado pela API atende a condição *isPhoto*, que testa se o dado retornado pela API possui o valor *photo* no atributo *type*. Caso seja verdade, a interface abstrata e concreta com o nome de *photo* será montada para o usuário.

5.1.2.4.2 Interface Abstrata

Devido à complexidade desta interface, ela foi dividida em dois quadros (Quadro 25 e Quadro 26) mostrando seu modelo de interface abstrata, para

facilitar sua visualização. Do mesmo modo que nas interfaces discutidas anteriormente, foi utilizada a mesma interface abstrata para smartphones e para desktops e tablets.

```
{name: "photo", widgets: [{ name: "navigation", children: [{
    name: "navigation-list",
    children: [{name: "navigation-list-item"},{
            name: "navigation-avatar",
            children: [{name: "navigation-avatar-img"}]}]}]},
{   name: "head", children: [{
        name: "image-box",
        children: [{name: "imagem-principal"},
        {   name: "imagem-context",
            datasource: "url:/cont?photo_id=<%= $data.id %>",
            children: [{
                name: "imagem-context-item",
                children: [{name: "imagem-context-item-prev"},
                    {name: "imagem-context-item-next"}]}]},
        {   name: "mais-box",
            datasource: "url:/photos?user_id=<%= $data.owner.nsid %>",
            parse: "$data.data",
            children: [{ name: "mais-item", children: [{name: "mais-img"}]}]}]}]},
```

Quadro 25 – Primeira parte do modelo de interface abstrata para a interface de foto usando o MIRA

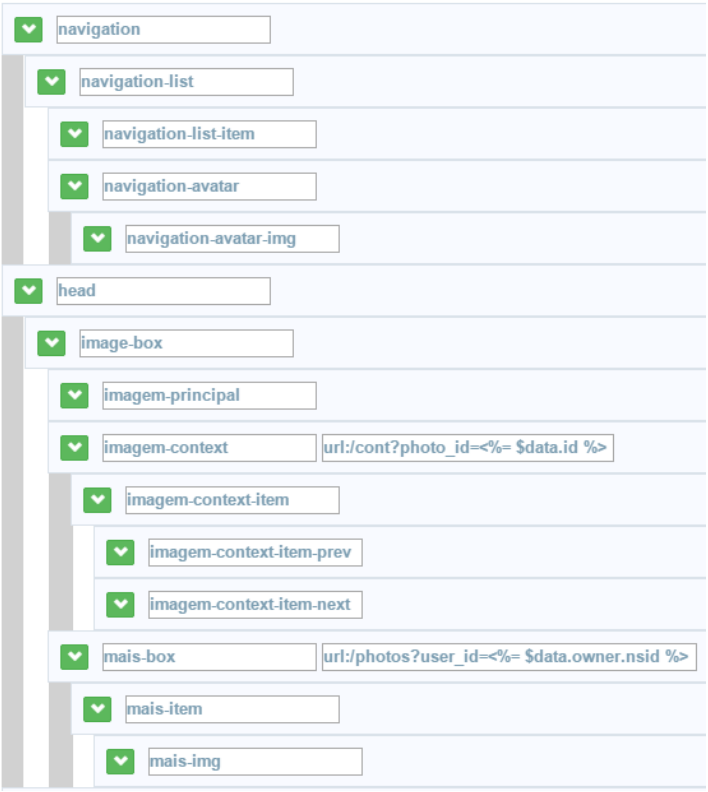

Figura 39 – Estrutura da primeira parte da interface abstrata de foto utilizando o MIRA

```
{   name: "content", children: [{name: "main", children: [{
        name: "user-box", children: [{
            name: "user-avatar-link",
            children: [{name: "user-avatar"}]},
            {name: "user-name"},
            {name: "photo-name"}]},
        {   name: "comment-box",
            datasource: "url:/comments?photo_id=<%= $data.id %>",
            parse: "$data.comments.comment",
            children: [{ name: "comment-item",
                children: [{name: "comment-author",
                    children: [{name: "comment-avatar"}]
                },  {name: "comment-author-name"},
                    {name: "comment-content"}]}]}]},
    {   name: "sidebar", children: [{name: "count-box", children: [{
            name: "star-box",
            datasource: "url:/favorites?photo_id=<%= $data.id %>",
            parse: "[$data]", children: [{name: "star-item",
                children: [{name: "star-desc"},{name: "star-qtd"}]}]},
            {   name: "view-count-box",
                children: [{
                    name: "view-count-item",
                    children: [{name: "view-count-desc"},
                        {name: "view-count-qtd"}]}]},
            {   name: "comment-count-box",
                children: [{
                    name: "comment-count-item",
                    children: [{name: "comment-count-desc"},
                        {name: "comment-count-qtd"}] }]},
            {   name: "taken-box",
                children: [{name: "taken-item",
                    children: [{name: "taken-desc"},{name: "taken-qtd"}]}]}]},
        {   name: "groups-box",
            children: [{name: "group-submit"},{name: "groups-title"},
            {name: "groups-itens",
             datasource: "url:/photo/context?photo_id=<%= $env.$data.id %>",
             parse: "$data.pool",
             children: [{
              name: "groups-item",
              children: [{name: "group-thumb"},
                  {name: "group-name"},
                  {name: "group-status"}]}]},
             {name: "group-submit-box",
             children: [{
                 name: "group-submit-box-header",
                 children: [{name: "group-submit-box-header-title"},
                     {name: "group-submit-box-header-title-filter"}]},
                 {name: "group-submit-box-body",
                  children: [{
                  name: "group-submit-box-body-groups",
                  datasource: "url:/photo/listGroups",
                  parse: "$data.groups.group",
                  children: [{
                    name: "group-submit-box-body-groups-item",
                    children: [{name: "group-submit-box-body-groups-thumb"},
                     {name: "group-submit-box-body-groups-name"},
                     {name: "group-submit-box-body-groups-status"}]
                  }]}]},{name: "group-submit-box-footer"}]}]}]}]
}, {name: "footer"}]};
```

Quadro 26 – Segunda parte da interface abstrata de foto do Flickr usando o MIRA

Figura 40 – Segunda parte da estrutura da interface abstrata de foto do Flickr utilizando o MIRA

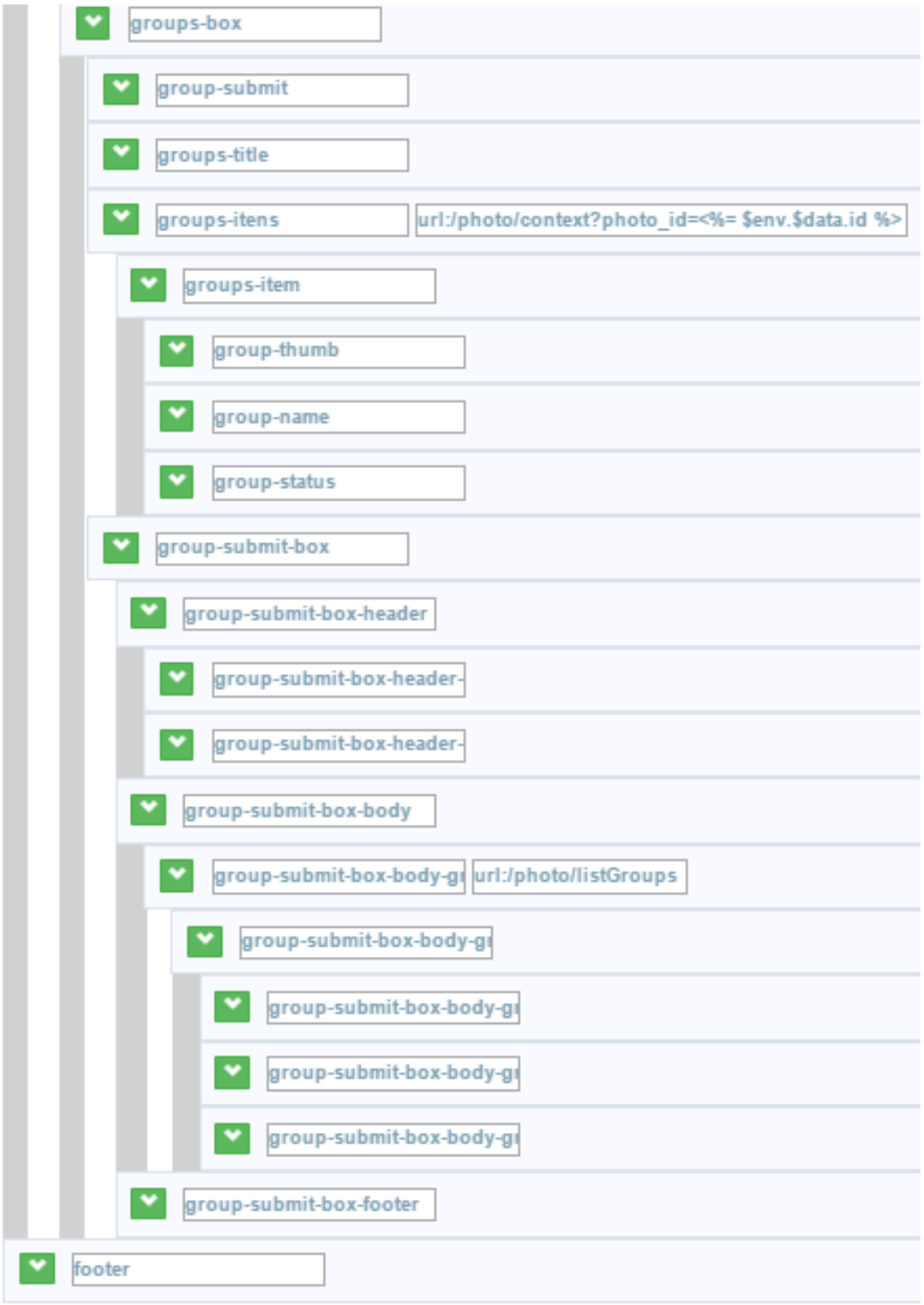

Figura 41 – Cont. da segunda parte da estrutura interface abstrata de foto usando o MIRA

5.1.2.4.3 Interface Concreta

```
{ name: "photo",
head: [
    {name: 'title', widget:'Title', value: '$data.title._content'}
],
maps: [
{ name: 'navigation', widget: 'BootstrapNavigation', value:'"Flickr"'},
{ name: 'navigation-list', widget: 'BootstrapNavigationList'},
{ name: 'navigation-list-item', widget: 'BootstrapNavigationListItem',
    value:'$env.$login.fullname', href:'navigate("/me")'},
{ name: 'navigation-avatar', widget: 'BootstrapNavigationListItem', tag:'a',
  class:'navbar-brand', href:'navigate("/user?user_id="+$env.$login.user_nsid)'},
{ name: 'navigation-avatar-img', tag:'img', class:'img-circle',
    src:'$env.$login.info.picture', width:"33", height:'33'},

{ name: 'head', class:'jumbotron'},
{ name: 'image-box', class:'container', text:'center'},
{ name: "imagem-principal", tag:'img', src:"$data.size.s1024" },
{ name: "imagem-principal",  when:'isMobile',
    tag:'img', src:"$data.size.s320", img:'responsive' },

{name:'imagem-context', when:'isMobile'},
{name:'imagem-context-item'},
{name:'imagem-context-item-prev', when:'hasPreviousPhoto', tag:'a', btn:'info',
    value:'Prev', href:'navigate("/photo?photo_id=" + $data.prevphoto.id)'},
{name:'imagem-context-item-next', when:'hasNextPhoto', tag:'a', btn:'info',
    value:'Next', href:'navigate("/photo?photo_id=" + $data.nextphoto.id)'},

{name:'mais-box', class:'row', when:'notMobile'},
{name:'mais-item', tag:'a', href:'navigate("/photo?photo_id=" + $data.id)',
    md:1, class:'thumbnail'},
{name:'mais-item', when:'actualPhoto', md:1, class:'thumbnail alert-info',
    tag:'a', href:'navigate("/photo?photo_id=" + $data.id)', },
{name:'mais-img', tag:'img', src:'$data.size.square'},

{ name: "content", class:'container' },

{ name: "main", md:7 },
```

```
{ name: 'sidebar', md:5, when:'notMobile'},
{ name: 'count-box', widget:'BootstrapPanelBody', class:'panel-default' },

{ name: "star-box", md:2  },
{ name: 'star-item', label:'info', title:'Favoritos'  },
{ name: 'star-desc', widget:'BootstrapIcon', icon:'star' },
{ name: 'star-qtd', tag:'span', value:'$data.photo.person.length' },

{ name: "view-count-box", md:2  },
{ name: 'view-count-item', label:'info', title:'Visualizações' },
{ name: 'view-count-desc', widget:'BootstrapIcon', icon:'eye-open' },
{ name: 'view-count-qtd', tag:'span', value:'$data.views' },

{ name: "comment-count-box", md:2  },
{ name: 'comment-count-item', label:'info', title:'Comentários' },
{ name: 'comment-count-desc', widget:'BootstrapIcon', icon:'comment' },
{ name: 'comment-count-qtd', tag:'span', value:'$data.comments._content' },

{ name: "taken-box", md:2  },
{ name: 'taken-item', label:'info', title:'Tirada em' },
{ name: 'taken-desc', widget:'BootstrapIcon', icon:'calendar' },
{ name: 'taken-qtd', tag:'span', value:'$data.dates.taken' },

{ name: 'groups-box', widget:'BootstrapPanelBody', class:'panel-default' },
{ name: 'groups-title', tag:'h3', value:'Groups' },
{ name: 'groups-itens' },
{ name: 'groups-item', md:12 },
{ name: 'group-name', value:'$data.title || $data.name' },
{ name: 'group-thumb', tag:'img', img:'circle', md:3,
    src:'getThumbnail($data.id, $data.iconserver, $data.iconfarm)' },
{ name: 'group-status' },
{ name: 'group-submit', value:'Add Group', tag:'button', class:'pull-right',
    onclick:'"show_modal(\'group-submit-box\')"' },
{ name: 'group-submit-box', widget:'BootstrapModalDialog' },
{ name: 'group-submit-box-header', widget:'BootstrapModalHeader' },
{ name: 'group-submit-box-header-title', tag:'h3', value:'Adicionar Grupos' },
{ name: 'group-submit-box-body', widget:'BootstrapModalBody' },
{ name: 'group-submit-box-footer', widget:'BootstrapModalFooter' },

{name: "group-submit-box-body-groups", class:'row' },
{name: "group-submit-box-header-title-filter", tag:'input',
    events:{ keyup:'filtrarGrupos'} },
{name: "group-submit-box-body-groups-item", when:'groupToAdd,groupShow', md:12},
{name: "group-submit-box-body-groups-thumb", tag:'img', img:'circle',
   pull:'left', src:'getThumbnail($data.id, $data.iconserver, $data.iconfarm)'},
{name: "group-submit-box-body-groups-name", value:'$data.name' },
{name: "group-submit-box-body-groups-status",
    tag:'button', value:'"Request"', btn:'default',
    events: {click: 'addGroup'} },
{name: "group-submit-box-body-groups-status", when: 'needApprove',
    tag:'button', value:'"Request"', btn:'warning',
    events: {click: 'addGroup'} },
{name: "group-submit-box-body-groups-status", when: 'waitingApprove',
    tag:'span', value:'Waiting', class:'disabled', btn:'warning' },
{name: "group-submit-box-body-groups-status", when: 'approved',
    tag:'span', value:'Added', class:'disabled', btn:'success' },

{ name: "footer" }
]}
```

Quadro 27 – Mapeamento de widgets concretos para a interface de foto do Flickr usando o MIRA

Assim como ocorre na interface *timeline* com o widget abstrato *slide_galery*, o widget abstrato *slidebar* só é exibido quando a condição n*otMobile* é válida, com isso, nenhum widget que é filho do widget abstrato *sidebar* é montado na interface do usuário.

Quando a condição *isMobile* é válida, o mapeamento para *imagem-context* passa a ser montado na interface, e seus widgets filhos passam a ser exibidos. Estes widgets são responsáveis pela navegação entre fotos de um usuário quando ele acessa por um smartphone. O *widget imagem-context-item-prev* só é montado caso haja uma foto anterior, e o *widget 'imagem-context-item-next* só é montado caso haja uma próxima foto.

Quando a condição *notMobile é válida*, outro widget fica responsável pela navegação entre fotos do usuário, o widget *mais-box*, este widget exibe uma lista de miniaturas de foto do usuário. E quando uma miniatura é clicada, é realizada uma navegação para outra foto. O widget *mais-item* é alterado pela condição *actualPhoto* que avalia se a foto que está sendo exibida em tamanho cheio é a mesma da miniatura, mudando a sua cor de fundo.

Quando o botão de adicionar grupo é clicado, é exibida uma modal para o usuário com a lista de grupo que ele pode adicionar a foto exibida, como vemos na Figura 42.

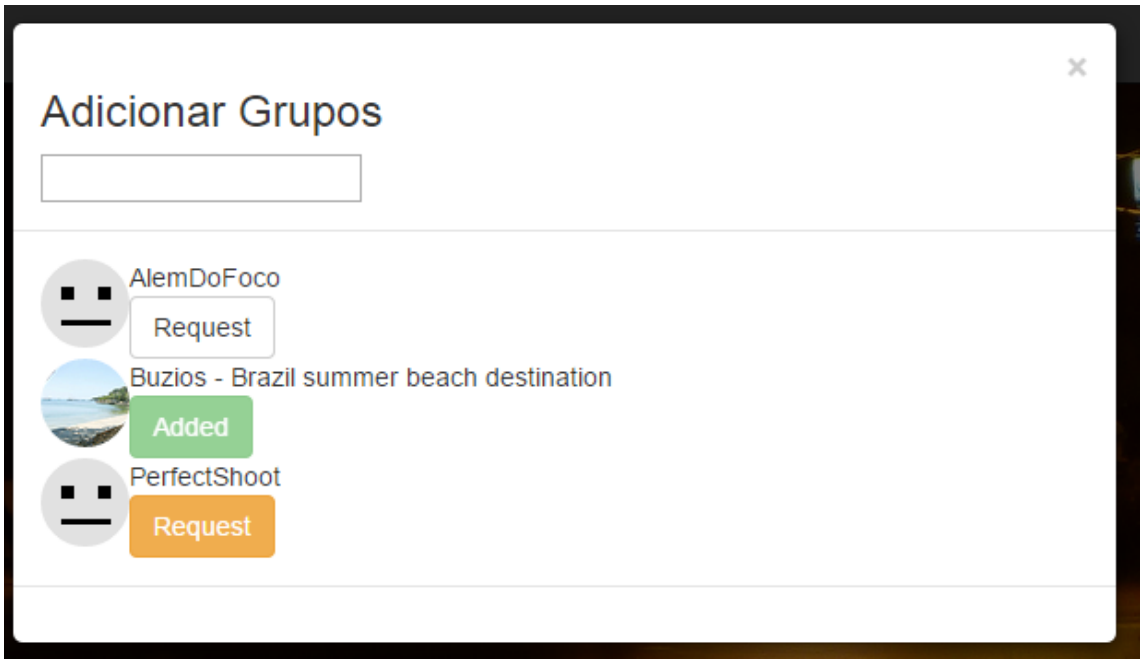

Figura 42 – Interface Modal para adicionar grupos em uma foto com a interface usando o MIRA

Nestes grupos de fotos que são listados, para serem exibidos, todos os grupos devem atender a condição *groupToAdd*, o grupo *Buzios – Brazil summer beach destination* também atendem a condição *approved*, logo, ele é exibido com um botão com fundo verde e marcado e desabilitado, O grupo *PerfectShot* atende a condição *needApprove,* logo, é exibido um botão com o fundo amarelo. O grupo *AlemDoFoco* atende exibe apenas o botão padrão para adição no grupo.

5.1.2.4.4 Eventos

Quando algum valor é inserido no widget *group-submit-box-header-title-filter*, é disparado o evento *filtrarGrupos*, valida se o nome do grupo contém o valor inserido no widget, se não possuir, ele adiciona o atributo *$hide,* como podemos ver no Quadro 27

```
events.filtrarGrupos = function(options){
    var text = options.$event.target.value.toLowerCase();
    var collection = options.$env.collections['group-submit-box-body-groups'];

    collection.each(function(model){
        if(text) {
            if (model.get('name').toLowerCase().indexOf(text) != -1) {
                model.set('$hide', undefined);
            } else {
                model.set('$hide', true);
            }
        } else {
            model.set('$hide', undefined);
        }
    });
};
```

Quadro 28 – Evento *filtrarGrupos* da interface de photo utilizando o MIRA

Se for inserido o valor *buzios*, apenas um grupo será exibido, como podemos ver na Figura 43.

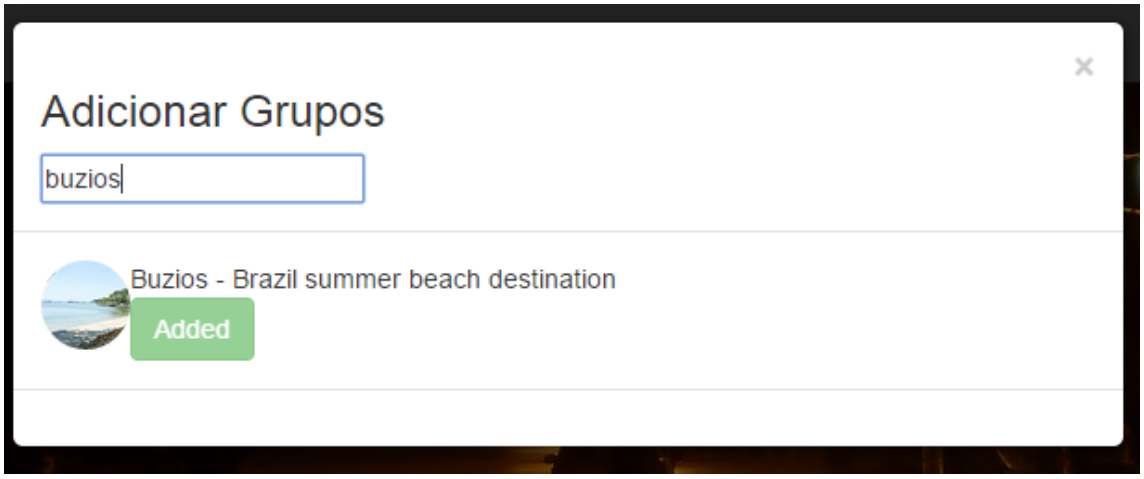

Figura 43 – Modal de grupo sendo filtrada por eventos do MIRA

Se o usuário clicar em um grupo que ainda não foi adicionado, será disparado o evento *addGroup*, que valida se há permissão para postar no grupo que foi clicado, se não tiver permissão, ele marca com *status* 2 que passa a ser válido para a regra *waitingApprove*. E se há permissão, ele adiciona na lista de grupos exibidos pela interface da photo.

```
window.addGroup = function (options) {
    if(options.$data.privacy == 2) {
        options.$dataObj.set('$status', 2);
    } else {
        options.$dataObj.set('$status', 'ok');
        var col = options.$env.collections['groups-itens'];
        col.add(options.$data);
    }
};
```

Se o usuário clicar nos outros 2 grupos de fotos que aquela foto ainda não faz parte, a interface que será montada, será com da Figura

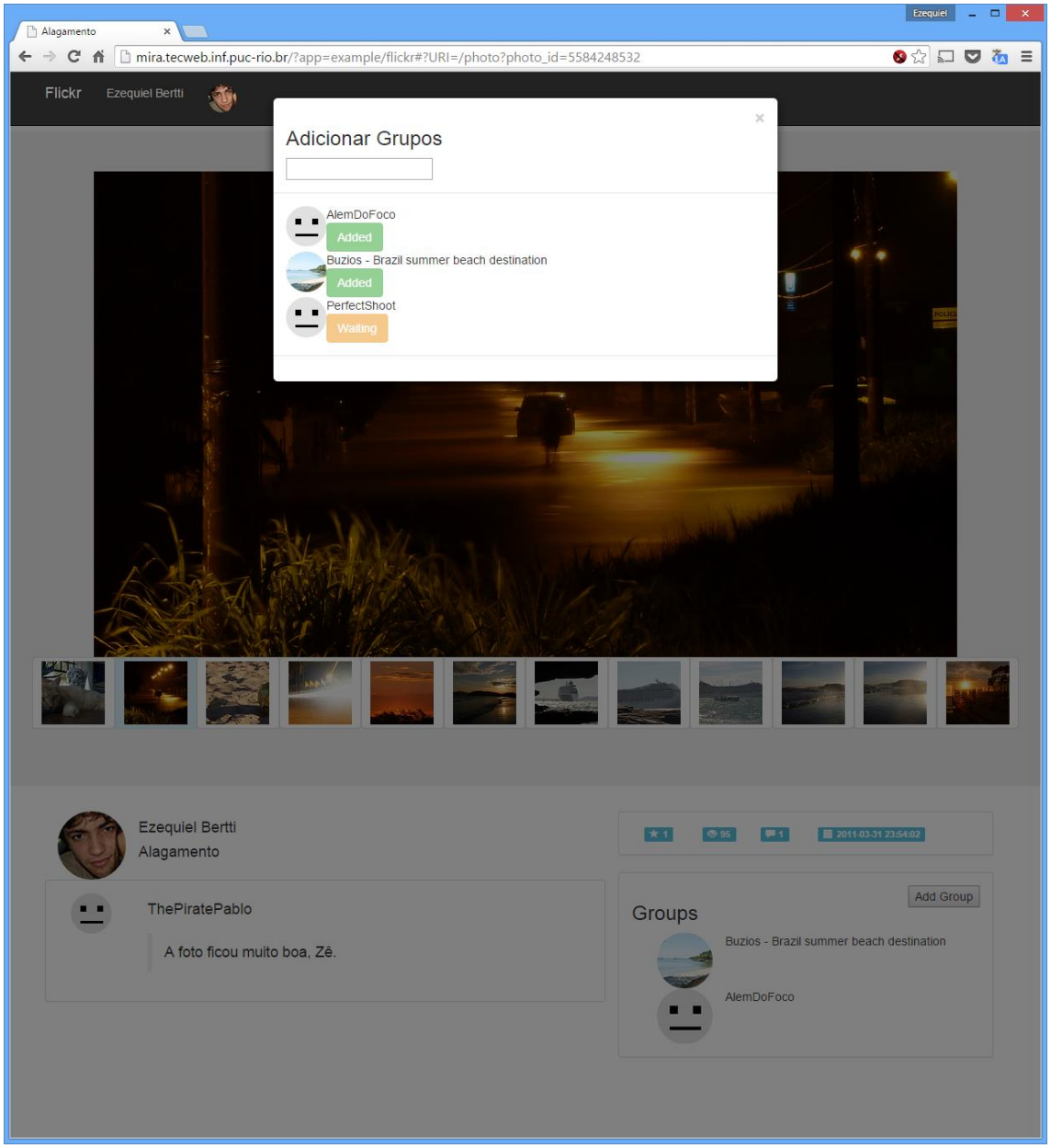

Figura 44 – Interface exibida após adicionar novos grupos a uma foto

Através do exemplo evidencia-se a expressividade dos modelos de interface propostos pelo MIRA. Analisando os códigos necessários para desenvolver a interface original do Flickr, que possui códigos e estruturas distintas para plataformas mobile e desktop, constata-se a simplicidade em montar a interface da aplicação utilizando o MIRA com a possibilidade de reutilizar condições e widgets sem a necessidade de se re-escrever os códigos para cada interface.

5.2 **AVALIAÇÃO QUALITATIVA**

Além de avaliar a expressividade, entendida como a capacidade de especificar, de forma mais simples as mesmas funcionalidades de uma aplicação web complexa, foi realizada também uma avaliação qualitativa com 5 desenvolvedores com conhecimentos variados sobre projeto de aplicações, desenvolvimento de Frontend, JavaScript e HTML.

A tarefa da avaliação foi o desenvolvimento da interface para 2 aplicações, uma construída utilizando o MIRA e outra utilizando seus conhecimentos prévios de construção de interfaces. O candidato poderia escolher para qual aplicação ele gostaria de construir a interface inicialmente.

A complexidade das interfaces para as duas aplicações era a mesma, apenas 2 telas, uma com um índice de itens, e quando o usuário da aplicação quisesse saber mais informações sobre um item do índice, ele clicaria no item e navegaria até uma tela de contexto do item, onde haveriam informações mais detalhadas sobre o item selecionado. A seção 5.2.3 apresenta mais detalhes. Ambas as interfaces possuíam adaptações a serem feitas que dependiam de que tipo de dispositivo estava sendo usado para acessar as interfaces da aplicação, alterando a resolução de imagens e forma de exibição de mapa de acordo com dispositivo.

Os candidatos foram divididos em 2 grupos, o GRUPO A desenvolveria a primeira interface utilizando seu conhecimento prévio de construção de interfaces, e em seguida, construiria a interface da segunda aplicação utilizando o MIRA. O GRUPO B desenvolveria a primeira interface utilizando o MIRA, e em seguida, a interface da segunda aplicação utilizando seus conhecimentos prévios de construção de interface.

Durante a avaliação foi solicitado aos candidatos que utilizassem a técnica Think Aloud [34], onde o candidato deve narrar em voz alta tudo que está pensando e pretendendo realizar durante a construção da aplicação.

Um documento com regras[4], um mockup[5] das telas e a estrutura de HTML foi fornecida para todos os candidatos de ambas as aplicações. Também foi

[4] http://mira.tecweb.puc-rio.br/docs

[5] http://mira.tecweb.puc-rio.br

disponibilizada uma página com toda a documentação de uso do MIRA, para auxiliar no desenvolvimento da interface.

Foi realizado um nivelamento prévio de conhecimento dos candidatos e toda a avaliação foi filmada. Foram observados os graus de satisfação, o sucesso na resolução de problemas, o tempo de desenvolvimento e a qualidade final da interface construída.

5.2.1 Perguntas Preliminares

Antes de se iniciar os testes, as seguintes perguntas foram feitas para nivelar seus conhecimentos prévios:

1) Anos como desenvolvedor
2) Nível de conhecimento como projetista de aplicação
3) Nível de conhecimento de desenvolvimento para Web
4) Nível de conhecimento de desenvolvimento com JavaScript
5) Nível de conhecimento de desenvolvimento para Web Mobile
6) Nível de conhecimento de desenvolvimento de interfaces responsivas
7) Nível de conhecimento do OOHDM, SHDM e/ou Interface concreta e abstrata

5.2.2 Nivelamento

Após as perguntas preliminares, foi realizada uma apresentação do modelo de interface do SHDM, com seleção de interface, interface abstrata e interface concreta. Em seguida foram apresentados os modelos do MIRA e a estrutura do framework junto com sua documentação[6].

5.2.3 Aplicações

Foi feita a proposta de desenvolver 2 interfaces, ambas com um nível de complexidade equivalente:

[6] http://mira.tecweb.inf.puc-rio.br/docs

5.2.3.1 Imobiliária

As regras de negócio para esta aplicação são descritas a seguir.

Esta é uma aplicação que lista imóveis oferecidos por uma imobiliária, mostrando detalhes de contrato, localização e informações mais detalhadas.

Interface de índice:

- Quando a negociação for de l*ançamento* o fundo do item de listagem será verde
- Quando a negociação for de *venda* o fundo do item de listagem será amarelo
- Quando a negociação for de *aluguel* o fundo do item de listagem será azul

Interface de contexto (detalhe) do imóvel

- A exibição da imagem deve carregar imagens para o tipo de dispositivo que está visualizando a aplicação, mobile, tablet ou desktop.
- Se o dispositivo for mobile ou tablet, deve ser exibido um mapa estático, onde apenas uma imagem com a localização é exibida.
- Se o dispositivo for desktop, deve ser exibido um mapa dinâmico, onde o usuário da aplicação pode dar zoom e interagir com o mapa da localização do imóvel.
- O local onde fica o título do imóvel, deve ter seu fundo assim como a listagem de imóveis do índice.
- A descrição do imóvel é apresentada na propriedade com o nome do tipo de contrato anunciado.
- Se o dispositivo for mobile ou tablet, deve ser exibido um mapa estático, onde apenas uma imagem com a localização é exibida.
- Se o dispositivo for desktop, deve ser exibido um mapa dinâmico, onde o usuário da aplicação pode dar zoom e interagir com o mapa da localização do imóvel.
-

As telas de *mockup* são mostradas a seguir:

Índice

Figura 45 – Mockup de índice da aplicação imobiliária

Contexto (detalhe)

Figura 46 – Mockup do contexto (detalhe) de um item da aplicação de imobiliária

5.2.3.2 Placar

Esta aplicação apresenta um placar de jogos de um time, de acordo com as seguintes regras:

- Inicialmente exibe-se uma lista de times.
- A propriedade pontos indica quantos pontos o time fez em sua última partida, se o valor for 3 ele ganhou, se for 1 ele empatou e for 0 ele perdeu

Interface de índice:

- Quando o time ganhou a última partida o fundo do item de listagem será verde
- Quando o time empatou a última partida o fundo do item de listagem será amarelo
- Quando o time perdeu a última partida o fundo do item de listagem será vermelho

Interface de contexto (detalhe) do time:

- Na listagem das últimas partidas de um time, o fundo do item deve seguir a mesma regra da listagem de times da interface de índice
- A informação nome do adversário na lista de partidas está disponível na propriedade com o mesmo nome que o campeonato que o time está disputando.
- Se na listagem de partidas o time atual for visitante, deve ser exibido primeiro o nome do time adversário, e logo em seguida, o seu nome, assim como o placar da partida e os gols de decisão de pênaltis. Se for mandante, deve aparecer o seu nome primeiro e em seguida do adversário, assim como gols e gols de decisão de pênaltis.
- No nome do time adversário deve existir um link para mostrar informações do time.

As telas de *mockup* são mostradas a seguir:

Índice

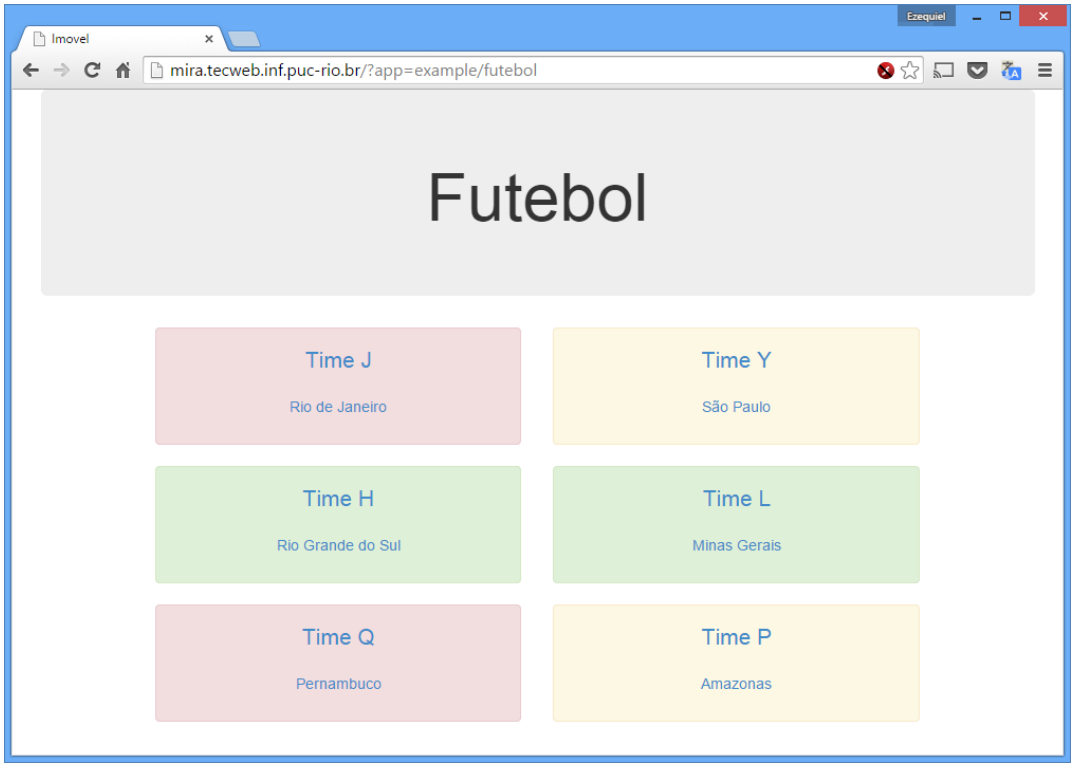

Figura 47 – Mockup do indice da aplicação de placar

Contexto

Figura 48 – Mockup do contexto de um item na aplicação de placar

5.2.4 Ferramenta auxiliar

Foi disponibilizada uma ferramenta gráfica para a edição das interfaces abstratas[7], que permite construir e dar manutenção com mais agilidade à estrutura utilizada para se criar as interfaces abstratas.

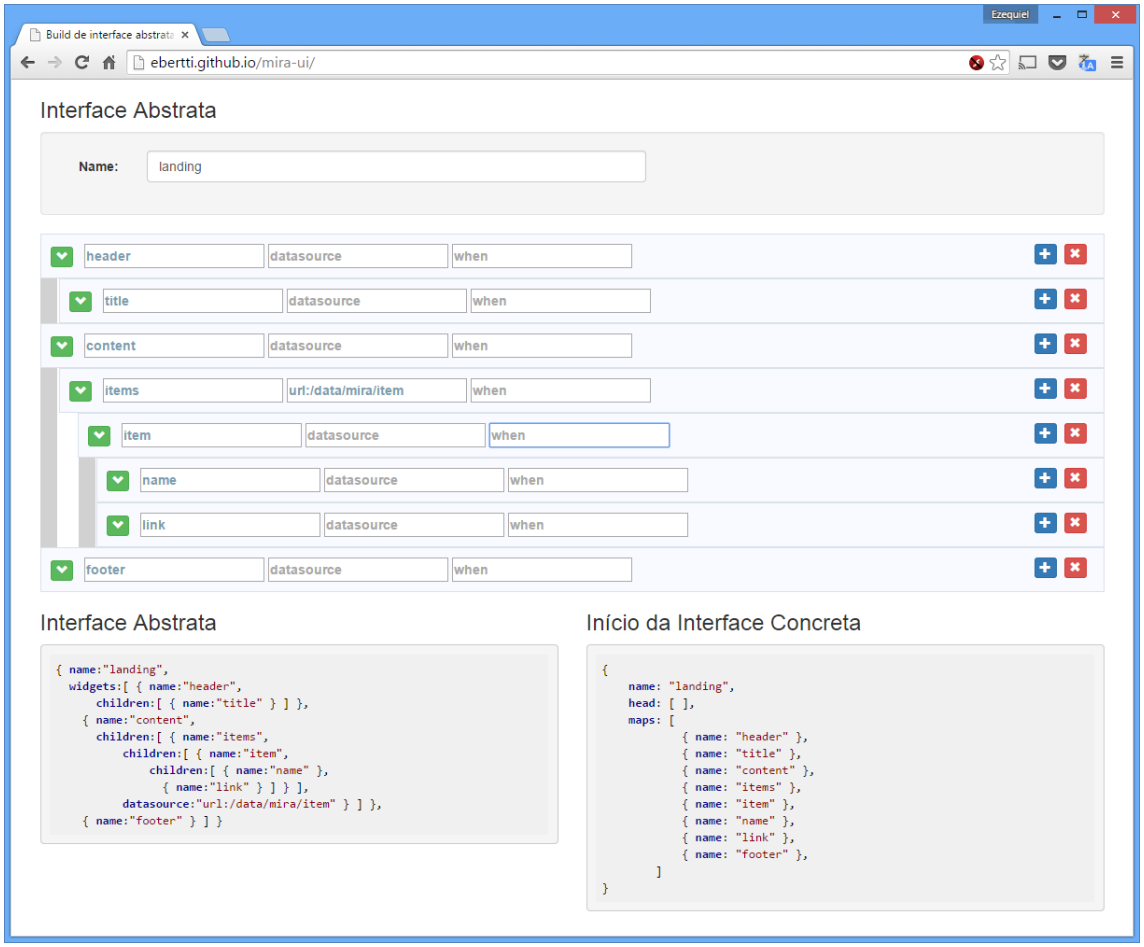

Figura 49 – Editor de interface abstrata

Além da estrutura de interface abstrata, ela gera a especificação de um esqueleto de interface concreta para facilitar a especificação do mapeamento de widgets para o projetista da aplicação.

[7] http://ebertti.github.io/mira-ui/

5.2.5 API REST

5.2.5.1 Imobiliária

Interface de índice

URI: /api/imovel/

Ação: Listar todos os imóveis

Estrutura:

```
[
  {
    "id": INTEIRO,
    "tipo": STRING,
    "nome": STRING,
    "bairro": STRING
  },
  ...
]
```

Quadro 29 – Estrutura JSON da interface de índice da aplicação imobiliária

Exemplo:

```
[
  {
    "id": 1,
    "tipo": "Aluguel",
    "nome": "Imóvel aluguel",
    "bairro": "Botafogo, RJ"
  },
  {
    "id": 2,
    "tipo": "Lançamento",
    "nome": "Lançamento - Apto com 2 suites",
    "bairro": "Ipanema, RJ"
  },
  {
    "id": 3,
    "tipo": "Venda",
    "nome": "Venda de Apartamento",
    "bairro": "Flamengo, RJ"
  },
  {
    "id": 4,
    "tipo": "Aluguel",
    "nome": "Imóvel de Casa",
    "bairro": "Lapa, RJ"
  }
]
```

Quadro 30 – Exemplo de retorno para interface de índice da aplicação imobiliária

Interface de contexto de um imóvel

URI: /api/imovel/{{ ID }}

Ação: Obter dados de um imóvel

Estrutura:

```
{
  "id": INTEIRO,
  "tipo": STRING EM ("Lançamento", "Venda", "Aluguel"),
  "nome": STRING,
  "bairro": STRING,
  "fotos": [
    {
      "desktop" : STRING,
      "tablet" : STRING,
      "mobile" : STRING
    },
    ...
  ],
  "localizacao": [
    {"item": STRING },
    ...
  ],
  "negociacao": [
    {"item": STRING },
    ...
  ],
  "{{ TIPO }}": STRING
}
```

Quadro 31 – Estrutura JSON da interface de contexto da aplicação imobiliária

Exemplo:

```
{
  "id": 1,
  "tipo": "Aluguel",
  "nome": "Imóvel aluguel",
  "bairro": "Botafogo",
  "fotos": [
    {
      "desktop" : "http://placehold.it/1980x600/f1f1f1/ff0000",
      "tablet" : "http://placehold.it/1024x400/f1f1f1/ff0000",
      "mobile" : "http://placehold.it/600x300/f1f1f1/ff0000"
    }
  ],
  "localizacao": [
    {"item": "Rio de Janeiro"},
    {"item": "Zona Sul"},
    {"item": "Botafogo"}
  ],
  "negociacao": [
    {"item": "30 meses"},
    {"item": "Somente com seguro"}
  ],
  "aluguel": "descrição do imóvel para aluguel"
}
```

Quadro 32 – Exemplo de retorno para interface de contexto de um item da aplicação imobiliária

5.2.5.2 Placar

Interface de índice

URI: /api/futebol/

Ação: Listar todos os times

Estrutura:

```
[
  {
    "id": INTEIRO,
    "nome": STRING,
    "ponto": INTEIRO,
    "estado": STRING
  },
  ...
]
```

Quadro 33 – Estrutura JSON da interface de índice da aplicação de placar

Exemplo:

```
[
  {
    "id": 1,
    "nome": "Time J",
    "ponto": 0,
    "estado": "Rio de Janeiro"
  },
  {
    "id": 2,
    "nome": "Time Y",
    "ponto": 1,
    "estado": "São Paulo"
  },
  {
    "id": 3,
    "nome": "Time H",
    "ponto": 3,
    "estado": "Rio Grande do Sul"
  },
  {
    "id": 4,
    "nome": "Time L",
    "ponto": 3,
    "estado": "Minas Gerais"
  }
]
```

Quadro 34 – Exemplo de retorno para interface de índice da aplicação de placar

Interface de contexto de um time

URI: /api/futebol/{{ ID }}

Ação: Obter dados de um time

Estrutura:

```
{
  "id": INTEIRO,
  "nome": STRING,
  "ponto": INTEIRO EM (0, 1 ou 3),
  "estado": STRING,
  "sede": STRING,
  "campeonato": STRING,
  "partidas": [
    {
      "gols_favor": INTEIRO,
      "gols_contra": INTEIRO,
      "local": STRING EM ("casa" ou "visitante"),
      "contra_id": INTEIRO,
      "{{ CAMPEONATO }}": STRING,
      "ponto": INTEIRO EM (0, 1 ou 3) ,
      "penaltis_favor": INTEIRO ou NULO,
      "penaltis_contra": INTEIRO ou NULO,
    },
    ...
  ]
}
```

Quadro 35 – Estrutura de retorno para interface de contexto da aplicação de placar

Exemplo:

```
{
  "id": 1,
  "nome": "Time J",
  "ponto": 0,
  "estado": "Rio de Janeiro",
  "sede": "Ipanena, RJ",
  "campeonato": "Brasileiro",
  "partidas": [
    {
      "gols_favor": "2",
      "gols_contra": "1",
      "local": "casa",
      "contra_id": 2,
      "brasileiro": "Time Y",
      "ponto": 3
    }
  ]
}
```

Quadro 36 – Exemplo de retorno para interface de contexto da aplicação de placar

5.2.6 Estrutura em HTML

Foi também disponibilizado um arquivo HTML com a estrutura básica de cada uma das telas de mockup, que podem ser vistas em detalhe no Apêndice 9.1

5.2.7 Perguntas

As perguntas realizadas após a construção das interfaces propostas foram

1) O layout final em ambas as aplicações ficou igual ao que foi proposto?
2) Houve diferença na documentação ou na dificuldade do entendimento das interfaces?
3) Quais foram as principais dificuldades que encontrou dentro da tarefa proposta?
4) Quais foram as dificuldades no uso do MIRA?
5) Quais foram os benefícios no uso do MIRA?
6) Se tivesse que refazer a interface que foi feita sem utilizar o MIRA, você acredita que faria em menos tempo que utilizando seus conhecimentos prévios de construção de interface para web?
7) Qual dificuldade haveria em modificar a interface da aplicação com novas regras de interface em ambas as aplicações?
8) Qual dificuldade você encontrou em ambas as aplicações para desenvolver uma interface que se adapta ao dispositivo utilizado?

5.2.8 Analise dos Resultados

Ao finalizar as avaliações, observou-se que os candidatos conseguiram realizar a avaliação praticamente o mesmo tempo para a construção da aplicação com um ambiente desconhecido, como é o caso do MIRA, e com o uso as ferramentas que já estavam habituados.

Candidato	Convencional	MIRA
1	1:35	1:55
2	1:31	1:41
3	N/A	1:57
4	1:57	1:26
5	1:18	1:07

Tabela 2 - Comparação de tempos durante a avalição

Esta observação pode ser considerada uma evidência da utilidade concreta do MIRA, considerando-se que o candidato teve de aprender a lidar com uma nova forma de construir interface baseada em uma estrutura de interface abstrata, concreta e seleção de interface e, além disto, precisou aprender a forma de funcionar do MIRA e sua estrutura. Mesmo com a curva de aprendizado, conseguiram realizar a tarefa empregando o mesmo tempo total que necessitaram utilizando outras técnicas com as quais já eram familiarizados e fluentes.

Observou-se também que os candidatos que não estavam habituados com o desenvolvimento de interfaces, ou com conhecimentos restritos, conseguiram se adaptar bem a nova estrutura e ao framework.

Todos os candidatos, ao utilizar o MIRA, não se sentiram como programadores, mas sim, como construtores. Eles destacaram a relevância da estrutura de Condições de Negócio pela centralização de condições utilizadas na interface, e de regras de Seleção de Interface por não ficar preso a estrutura de URI como em abordagens convencionais. Também foram comentados o uso das Interfaces Abstrata e Concreta pela possibilidade de estender a mesma estrutura de interface para diversos contextos sem precisar reconstruí-las para cada dispositivo ou ambiente.

As maiores dificuldades encontradas estavam relacionadas à manipulação da estrutura de interface abstrata, que por ser uma árvore, torna-

se complicada de manipular e organizar seus elementos. Este aspecto foi facilitado ao se criar uma ferramenta de edição de interfaces abstratas[8].

5.3 **DESEMPENHO**

Foi realizado uma comparação de desempenho para avaliar o tempo de carga de uma aplicação feita com o MIRA comparada a uma aplicação feita com HTML convencional, ambas exibindo a mesma interface final.

Nesta comparação foi utilizada a aplicação de Imobiliária mencionada na avaliação qualitativa. E as métricas analisadas foram o tempo de carga e de processamento da página inicial, com a lista de imóveis, e com a exibição de informações detalhadas, com fotos e mapa do um imóvel.

A coleta de dados foi feita pela ferramenta DevTools[9] contida no navegador Google Chrome.

O tempo é exibido em milésimos de segundo.

	MIRA - Index	MIRA - Segunda	HTML - Index	HTML - Segunda
Loading	10	7	14	31
JavaScript	308	33	55	406
Render	3	6	14	36
Painting	1	3	3	17
Outros	153	49	37	241
Total	475	98	123	731

Tabela 3 - Comparações de tempo de carga de páginas

Observa-se que no index, página inicial, o tempo de carga do MIRA é maior que a página seguinte, isto ocorre devido ao processamento de arquivos de estilo e de JavaScript do MIRA em sua carga inicial, já nas navegações seguintes, há um ganho no tempo e na quantidade de informações processadas pelo navegador.

[8] http://ebertti.github.io/mira-ui/

[9] https://developer.chrome.com/devtools

Já na aplicação com HTML puro, temos um tempo maior nas páginas seguintes, já que elas possuem mais informações e uma estrutura mais elaborada, bem como a cada requisição de navegação subsequente.

6 CONCLUSÕES

6.1 TRABALHOS RELACIONADOS

Alguns frameworks de interface possuem algumas características do MIRA quanto a funcionarem em modo *client-side*, mas a maioria deles, como AngularJS, React e Ember, estão fortemente atrelados a estrutura do HTML, limitando sua utilização apenas para a construção de páginas com HTML. Outros como BackboneJS e MarionetteJS criam uma estrutura de camadas para a interface com alguns facilitadores de integração com API REST.

O Callimachus [35] também é um framework de interface dirigida por modelos em JavaScript. A construção de sua interface se baseia em marcações de RDF realizadas no HTML.

O diferencial de maior destaque do MIRA em relação as outras ferramentas é facilidade de adaptação de acordo com o contexto da aplicação, como modelo, dispositivo, ambiente e requisição.

6.2 TRABALHOS FUTUROS

A maior dificuldade encontrada ao se utilizar as estruturas de modelos de interface do MIRA, foi a manipulação dos modelos utilizando a estrutura JSON. A partir dessa afirmação, é possível concluir que é necessário se facilitar a interação do projetista de interface com os modelos do MIRA.

Com isso, será proposto o desenvolvimento de uma ferramenta gráfica para construir interfaces e com pré-visualização das mesmas.

Já fora construído uma ferramenta gráficas[10] que ajuda a montar a estrutura de uma Interface Abstrata com seus Widget Abstratos e ainda inicia a estrutura de Interface Concreta com mapeamentos simples do Widgets Concretos.

Mas além de uma Interface Abstrata, esta ferramenta irá construir Condições de Negócio, Seleção de Interface e criar o mapeamento de Widgets Concretos disponíveis na Interface Concreta.

[10] http://ebertti.github.io/mira-ui/

Outra necessidade é a realização de mais testes, com mais usuários e com mais aplicações distintas.

7 CONTRUIBUIÇÕES

- Evolução do modelo SDHM, acrescentando a estrutura de eventos nos Widgets montados na interface do usuário. Dando mais expressividade as interfaces construídas com o modelo SHDM.
- Criação de novo modelo para descrever a distribuição do processamento runtime no cliente e no servidor, permitindo múltiplas estratégias de distribuição. Com isso, criando a possibilidade de realizar avaliações de condições de negócio que necessitam de um processamento de dados no servidor da aplicação, algo que seria incompatível com alguns dispositivos que acessam a aplicação utilizando o MIRA.
- Implementação e avaliação de um framework em JavaScript para criação de interface ricas e expressivas sem deixar de ser adaptável a diversos serviços de API REST. Tornando possível a integração de diversas APIs em uma única aplicação, mesmo sem conhecer sua estrutura de URIs, apenas conhecendo os modelos retornados por elas.

8 Referências Bibliográficas

1. BRAD A. MYERS, M. B. R. Survey on user interface programming, 1992.
2. ROLF H. WEBER, R. W. **Internet of Things**. [S.l.]: [s.n.], v. ISBN: 978-3-642-11709-1.
3. WEB Service. Disponivel em: <http://en.wikipedia.org/wiki/Web_service>.
4. ROY T. FIELDING, R. N. T. Principled design of the modern Web architecture. **Journal ACM Transactions on Internet Technology (TOIT)**.
5. COUTAZ, J. A. G. C. Hci and software engineering for user interface plasticity. **Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications**, n. Third Edition, 2012.
6. S., B. et al. The TERESA XML Language for the Description of Interactive Systems at Multiple Abstraction Levels. **Proceedings Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages**, Pisa, Italy, 2004. 103-110.
7. F., P.; C., S.; MARIA, S. L. D. A Universal, Declarative, Multiple Abstraction Level Language for Service-Oriented Applications in Ubiquitous Environments. **ACM Transactions on Computer-Human Interaction (TOCHI)**, November 2009.
8. LIMBOURG, Q. et al. USIXML: A Language Supporting Multi-path Development of User Interfaces. **Working Conference on Engineering for Human-Computer Interaction and International Workshop on Design Specification and Verification of Interactive Systems(EHCI/DS-VIS)**, Hamburg, Germany, 2004. 200-220.
9. ROSSI, G. **Um método orientado a objetos para o projeto de aplicações hipermidia**. Rio de Janeiro: PUC-Rio, 1996.
10. SCHWABE, D.; PONTES, R. A.; MOURA, I. OOHDM-Web: An Environment for Implementation of Hypermedia Applications in the WWW. **ACM SigWEB Newsletter**, Junho 1999.
11. LIMA, F. Modelagem semântica de aplicações na WWW, 2003.

12. MOURA, S. S. Desenvolvimento de Interfaces Governadas por Ontologias para Aplicações na Web Semântica, 2004.
13. SYNTH. Disponivel em: <http://www.tecweb.inf.puc-rio.br/synth>.
14. RDF. Disponivel em: <http://www.w3.org/RDF/>.
15. OSMANI, A. **Developing Backbone. js Applications**. [S.l.]: O'Reilly Media, Inc., 2013.
16. PB DARWIN, P. K. **AngularJS web application development**. [S.l.]: Packt Publ, 2013.
17. STERNE, B.; BARTH, A. Cross-site Scripting. **XSS**. Disponivel em: <http://pt.wikipedia.org/wiki/Cross-site_scripting>.
18. NASCIMENTO, V. B. **Modelagem e geração de interfaces dirigidas por regras.** Rio de Janeiro: PUC-Rio, 2013.
19. JAVASCRIPT WEB API. **JavaScript WEB API**, 2014. Disponivel em: <http://www.w3.org/standards/webdesign/script.html>.
20. HTTP://REQUIREJS.ORG/. **RequireJS**, 2014. Disponivel em: <http://requirejs.org/>.
21. PROTOTYPE. **Wikipedia**, 2014. Disponivel em: <http://pt.wikipedia.org/wiki/Prototype>.
22. FLYWEIGHT Pattern. **Wikipedia**, 2014. Disponivel em: <http://en.wikipedia.org/wiki/Flyweight_pattern>.
23. BUILDER Pattern. **Wikipedia**, 2014. Disponivel em: <http://en.wikipedia.org/wiki/Builder_pattern>.
24. UMD (Universal Module Definition) patterns for JavaScript modules that work everywhere. **UMD**, 2014. Disponivel em: <https://github.com/umdjs/umd>.
25. COMMONJS. **CommonJS**. Disponivel em: <http://www.commonjs.org/>.
26. AMD-API. **AMD**, 2014. Disponivel em: <https://github.com/amdjs/amdjs-api/wiki/AMD>.
27. JSCOMPLEXITY.ORG. **jscomplexity.org**, 2015. Disponivel em: <http://jscomplexity.org/>.

28. BOOTH, P. complexity-report. **complexity-report**, 2015. Disponivel em: <https://github.com/philbooth/complexity-report>.
29. MISRA; SANJAY; CAFER, F. Estimating Quality of JavaScript. **The International Arab Journal of Information Technology 9.2**, 2012.
30. NGUYEN, V. A SLOC counting standard. **COCOMO II Forum**, 2007.
31. MCCABE. A Complexity Measure. **EEE Transactions on Software Engineering**, 1976. 308–320.
32. M, H. Elemets of Software Science. **Elsevier North-Holland**, 1977.
33. P, O.; J, H.; DA, A. A Definition and Taxanomy for software Maintainability. **SETL Report #9! - 08 TR. University of Idaho**, Moscow, 1992.
34. MW VAN SOMEREN, Y. B. J. S. **The think aloud method:** A practical guide to modelling cognitive processes. [S.l.]: [s.n.], 1994.
35. CALLIMACHUS, a Linked Data management system. **Callimachus**, 2012. Disponivel em: <http://www.w3.org/2001/sw/wiki/Callimachus>.
36. NODE.JS. Disponivel em: <http://www.nodejs.org>.
37. CALVARY, G. . C. J. . B. L. . F. M. . L. Q. . M. L. . V. J. The CAMELEON reference framework. **Deliverable D1, 1.**, 2002.
38. S., B. et al.

9 APÊNDICE

9.1 HTML

Imóvel

Índice

```
<!DOCTYPE html>
<html>
<head lang="en">
    <title class="navigate_remove">Imovel</title>
    <link type="text/css" rel="stylesheet" href="css/bootstrap.css">
    <meta name="viewport" class="navigate_remove"
          content="width=device-width, initial-scale=1">
</head>
<body>
<div id="container" class="container">
    <div id="head" class="jumbotron">
        <h1 id="title" class="text-center">Escolha seu Imóvel</h1>
    </div>
    <div id="content" class="row col-md-10 col-md-offset-1">
        <div id="items" class="">
            <div id="item" class="col-md-6">
                <div id="tipo" class="panel-body alert-info alert">
                    <a id="link" href="#?URI=/api/imovel/1">
                        <p id="nome" class="lead text-center">
                            Imóvel aluguel</p>
                        <p id="bairro" class="text-center">
                            Botafogo, RJ</p>
                    </a>
                </div>
            </div>
            <div id="item" class="col-md-6">
                <div id="tipo" class="panel-body alert-info alert">
                    <a id="link" href="#?URI=/api/imovel/4">
                        <p id="nome" class="lead text-center">
                            Imóvel de Casa</p>
                        <p id="bairro" class="text-center">
                            Lapa, RJ</p>
                    </a>
                </div>
            </div>
            <div id="item" class="col-md-6">
                <div id="tipo" class="panel-body alert-warning alert">
                    <a id="link" href="#?URI=/api/imovel/6">
                        <p id="nome" class="lead text-center">
                            Terreno a Venda</p>
                        <p id="bairro" class="text-center">
                            Gávea, RJ</p>
                    </a>
                </div>
            </div>
        </div>
    </div>
</div></body></html>
```

Quadro 37 – Estrutura HTML do índice da aplicação de Imóvel

Contexto de um imóvel

```
<html>
<head>
    <title class="navigate_remove">Imovel | Aluguel | Imóvel aluguel</title>
    <link type="text/css" rel="stylesheet" href="css/bootstrap.css">
    <meta name="viewport" class="navigate_remove"
          content="width=device-width, initial-scale=1">
</head>
<body>
<div id="carousel">
    <!-- placeholder para javascript de carrocel de imagens -->
</div>
<div id="content" class="container">
    <h1 id="nome" class="text-center alert">Imóvel aluguel</h1>
    <div id="detalhes" class="col-md-8">
        <div id="row" class="row well">
            <div id="localizacao_box" class="col-md-6">
                <h3 id="localizacao_title" class="">Localização</h3>
                <ul id="localizacao_lista" class="">
                    <li id="localizacao_item" class="">Rio de Janeiro</li>
                    <li id="localizacao_item" class="">Zona Sul</li>
                    <li id="localizacao_item" class="">Botafogo</li>
                </ul>
            </div>
            <div id="negociacao_box" class="col-md-6">
                <h3 id="negociacao_title" class="">Contrato de Locação</h3>
                <ul id="negociacao_lista" class="">
                    <li id="negociacao_item" class="">30 meses</li>
                    <li id="negociacao_item" class="">Somente com seguro</li>
                </ul>
            </div>
        </div>
        <h3 id="descricao_title" class="text-center">Descrição</h3>
        <p id="descricao">
            Echo park banh mi gentrify organic art party leggings, put a bird on
            it american apparel marfa lo-fi readymade. Single-origin coffee
            marfa gluten-free, VHS sustainable salvia beard craft beer
            williamsburg. Vinyl four loko cred, seitan art party homo readymade
            fixie squid messenger bag lo-fi yr. Mixtape cardigan next level,
            bicycle rights sustainable beard VHS aesthetic cosby sweater. Homo
            bicycle rights iphone, lo-fi shoreditch sustainable put a bird on it
            vinyl. Twee pitchfork marfa etsy stumptown. Wayfarers cosby sweater
            3 wolf moon gentrify organic fap quinoa four loko gluten-free,
            butcher ethical readymade seitan.
        </p>
    </div>
    <div id="mapa_box" class="col-md-4">
        <!-- placeholder para javascript do mapa -->
    </div>
</div>
</body>
</html>
```

Quadro 38 – Estrutura HTML do contexto de um imóvel da aplicação de imóvel

Placar

Índice

```
<!DOCTYPE html>
<html>
<head>
    <title class="navigate_remove">Futebol</title>
    <link type="text/css" rel="stylesheet" href="css/bootstrap.css">
    <meta name="viewport" class="navigate_remove"
          content="width=device-width, initial-scale=1">
</head>
<body>
<div id="container" class="container">
    <div id="head" class="jumbotron">
        <h1 id="title" class="text-center">Futebol</h1>
    </div>
    <div id="content" class="row col-md-10 col-md-offset-1">
        <div id="items" class="">
            <div id="item" class="col-md-6">
                <div id="tipo" class="panel-body alert-danger alert">
                    <a id="link" href="#?URI=/api/futebol/1">
                        <p id="nome" class="lead text-center">Time J</p>
                        <p id="estado" class="text-center">Rio de Janeiro</p>
                    </a>
                </div>
            </div>
            <div id="item" class="col-md-6">
                <div id="tipo" class="panel-body alert-warning alert">
                    <a id="link" href="#?URI=/api/futebol/2" class="">
                        <p id="nome" class="lead text-center">Time Y</p>
                        <p id="estado" class="text-center">São Paulo</p></a>
                </div>
            </div>
            <div id="item" class="col-md-6">
                <div id="tipo" class="panel-body alert-success alert">
                    <a id="link" href="#?URI=/api/futebol/3" class="">
                        <p id="nome" class="lead text-center">Time H</p>
                        <p id="estado" class="text-center">Rio Grande do Sul</p>
                    </a>
                </div>
            </div>
        </div>
    </div>
</div>
</body>
</html>
```

Quadro 39 – Estrutura HTML do índice da aplicação de placar

Contexto de um time

```
<html>
<head>
    <title class="navigate_remove">Time J</title>
    <link type="text/css" rel="stylesheet" href="css/bootstrap.css">
    <meta name="viewport" class="navigate_remove"
          content="width=device-width, initial-scale=1">
</head>
<body>
<div id="display" class="container jumbotron">
    <h1 id="nome" class="text-center text-info">Time J</h1>
</div>
<div id="content" class="container">
    <div id="jogos_box" class="col-md-8">
        <h3 id="jogos_title" class="text-center">
            Partidas do Brasileiros
        </h3>
        <div id="jogos_lista" class="row">
            <div id="item_box" class="text-center alert-success alert">
                <h4 id="placar">
                    <span id="placar_texto" class="">Time J 2 X 1</span>
                    <a id="adversario" href="#?URI=/api/futebol/2">Time Y</a>
                </h4>
            </div>
            <div id="item_box" class="text-center alert-danger alert">
                <h4 id="placar">
                    <a id="adversario" href="#?URI=/api/futebol/3">Time K</a>
                    <span id="placar_texto">2 X 1 Time J</span>
                </h4>
            </div>
            <div id="item_box" class="text-center alert-success alert">
                <h4 id="placar">
                    <a id="adversario" href="#?URI=/api/futebol/4">Time M</a>
                    <span id="placar_texto">2 X 2 Time J</span>
                </h4>
            </div>
        </div>
    </div>
    <div id="mapa_box" class="col-md-4">
        <!-- placeholder para o mapa -->
    </div>
</div>
</body>
</html>
```

Quadro 40 – Estrutura HTML do contexto de um time da aplicação de placar

9.2 **AVALIAÇÕES**

9.2.1 Candidato 1

Formação: Engenharia da Computação na PUC-RIO

Área: Banco de Dados

9.2.1.1 Nivelamento

1) Anos como desenvolvedor
   2 anos

2) Nível de conhecimento com projetista de aplicação
   Iniciante

3) Nível de conhecimento de desenvolvimento para Web
   Básico

4) Nível de conhecimento de desenvolvimento com JavaScript
   Básico

5) Nível de conhecimento de desenvolvimento para Web Mobile
   Nenhum

6) Nível de conhecimento de desenvolvimento de interfaces responsivas
   Nenhum

7) Nível de conhecimento do SHDM e/ou Interface concreta e abstrata
   Nenhum

9.2.1.2 Interfaces

Placar – Sem o MIRA

Tempo: 1h35m

Ferramentas: Jquery

Ocorrências:

- Pelo candidato não ter muita experiência de desenvolvimento de web, ele precisou de ajuda para programar a interface.
- O candidato teve de ser avisado de que esqueceu as regras de negócio para a construção da interface.

- O candidato desistiu de implementar a mudança de mapa estático para dispositivos mobiles e o mapa dinâmico para desktops.
- O candidato esqueceu de modificar o *title* da página.

Imobiliária – Com o MIRA

Tempo: 1h55m

Observações:

- O candidato não contou com a ajuda do construtor de interface abstrata.
- Teve ajuda do instrutor no final para corrigir a estrutura da interface abstrata.

9.2.1.3 Perguntas:

1) O layout final em ambas as aplicações ficou igual ao que foi proposto?
   Na aplicação feita sem o MIRA ficou sem o mapa. Com o MIRA ficou igual.

2) Houve diferença na documentação ou na dificuldade do entendimento das interfaces?
   Estavam iguais.

3) Quais foram as principais dificuldades que encontrou dentro da tarefa proposta?
   Tive que reler regras de negócio e detalhes de exibição após ter parte da interface já construída. Esquecia das regras durante o desenvolvimento.

4) Quais foram as dificuldades no uso do MIRA?
   Entendimento da diferença da interface concreta e abstrata, mas quando diferenciei as duas, melhorou o desenvolvimento da aplicação

5) Quais foram os benefícios no uso do MIRA?
   Principal foi a organização. Para introduzir uma regra de negócio nova, não foi preciso reescrever o código, foi simples.

6) Se tivesse que refazer a interface que foi feita sem utilizar o MIRA, você acredita que faria em menos tempo que utilizando seus conhecimentos prévios de construção de interface para web?
   Sim

7) Qual dificuldade haveria em modificar a interface da aplicação com novas regras de interface em ambas as aplicações?
   Com o MIRA eu teria pouca dificuldade, pois seria apenas criar novos mapeamentos de interface concreta. Sem o MIRA teria que criar diversas condições e provavelmente deixaria o código desorganizado.

8) Qual dificuldade você encontrou em ambas as aplicações para desenvolver uma interface que se adapta ao dispositivo utilizado?
Com o MIRA foi apenas necessário criar uma nova regra checando o tipo do dispositivo e fazer o mapeamento para o widget concreto para a regra. Sem o MIRA eu nem tive coragem de fazer, pois sei que é bem complicado.

9.2.2 Candidato 2

Formação: Engenharia da Computação na INFNET em andamento
Área: Frontend Developer

9.2.2.1 Nivelamento

1) Anos como desenvolvedor
2 anos e meio

2) Nível de conhecimento com projetista de aplicação
Básico

3) Nível de conhecimento de desenvolvimento para Web
Avançado

4) Nível de conhecimento de desenvolvimento com JavaScript
Avançado

5) Nível de conhecimento de desenvolvimento para Web Mobile
Nenhum

6) Nível de conhecimento de desenvolvimento de interfaces responsivas
Básico

7) Nível de conhecimento do SHDM e/ou Interface concreta e abstrata
Nenhum

9.2.2.2 Interfaces

Imobiliária – Sem o MIRA

Tempo: 1h31m

Ferramentas: Backbone, Jquery, Underscore

Ocorrências:

- O candidato teve de ser avisado de que esqueceu as regras de negócio para a construção da interface. E teve algumas dificuldades para acrescentar as regras na estrutura que havia construído.
- O candidato esqueceu de modificar o *title* da página com informações do imóvel como nome e tipo de negócio.

Futebol – Com o MIRA

Tempo: 1h41m

Observações:

- Teve ajuda do instrutor no final para corrigir a estrutura da interface abstrata.
- O candidato esqueceu das regras de negócio durante o desenvolvimento da interface, mas ao ser aviso, conseguiu modificar de forma rápida e concluiu a tarefa.

9.2.2.3 Perguntas:

1) O layout final em ambas as aplicações ficou igual ao que foi proposto?
   Sim

2) Houve diferença na documentação ou na dificuldade do entendimento das interfaces?
   Sem diferenças

3) Quais foram as principais dificuldades que encontrou dentro da tarefa proposta?
   Aplicar as regras de negócio a interface.

4) Quais foram as dificuldades no uso do MIRA?
   Construir as interfaces abstratas e concretas. E a falta de uma forma de generalização e reutilização de trechos das interfaces.

5) Quais foram os benefícios no uso do MIRA?
   Realizar as tarefas foram mais fáceis por conta de como o MIRA lida com as regras de negócio. Como também utilizei um framework, as dificuldades de implementação foram as mesmas, mas quando estava usando o MIRA, na hora de lidar com as regras de negócio, foi mais fácil.

6) Se tivesse que refazer a interface que foi feita sem utilizar o MIRA, você acredita que faria em menos tempo que utilizando seus conhecimentos prévios de construção de interface para web?
   Sem dúvidas

7) Qual dificuldade haveria em modificar a interface da aplicação com novas regras de interface em ambas as aplicações?
   Com o MIRA precisaria apenas criar e mapear widgets para cada regra. Sem ele, eu teria que trabalhar muito em cima do código

8) Qual dificuldade você encontrou em ambas as aplicações para desenvolver uma interface que se adapta ao dispositivo utilizado?
   Com o MIRA parecia que estava trabalhando para a WEB, só foi adicionar uma regra e tudo se resolveu.

9.2.3 Candidato 3

Formação: Tecnólogo Processamento de Dado na PUC-Rio
Área: Analista de Sistemas e Gerente de Projetos

9.2.3.1 Nivelamento

1) Anos como desenvolvedor

11 anos sem desenvolver

2) Nível de conhecimento com projetista de aplicação
   Avançado

3) Nível de conhecimento de desenvolvimento para Web
   Como desenvolvedor muito básico, como projetista é avançado

4) Nível de conhecimento de desenvolvimento com JavaScript
   Nenhum

5) Nível de conhecimento de desenvolvimento para Web Mobile
   Nenhum

6) Nível de conhecimento de desenvolvimento de interfaces responsivas
   Nenhum

7) Nível de conhecimento do SHDM e/ou Interface concreta e abstrata
   Nenhum

9.2.3.2 Interfaces

Imobiliária – com o MIRA

Tempo: 1h57m

Ocorrências:

- O candidato teve dificuldade para entender como funcionavam os widgets diferentes que o padrão do MIRA.
- Não foi desenvolvido a interface sem o uso do MIRA por falta de conhecimento em programação em JavaScript pelo candidato.

9.2.3.3 Perguntas:

1) O layout final em ambas as aplicações ficou igual ao que foi proposto?
   Fiz somente uma aplicação e esta ficou igual.

2) Houve diferença na documentação ou na dificuldade do entendimento das interfaces?
   Fiz somente uma aplicação.

3) Quais foram as principais dificuldades que encontrou dentro da tarefa proposta?
   Foi simples, mas me senti desambientado por não desenvolver a muito tempo.

4) Quais foram as dificuldades no uso do MIRA?
   Foi o entendimento de quando utilizar o datasource, e a parte de abstração das interfaces

5) Quais foram os benefícios no uso do MIRA?
   Não parece que estou desenvolvendo, parece que estou apenas indicando o que quero que seja feito, sem programar. Organização que a estrutura de modelos de interface facilita muito o entendimento.

6) Se tivesse que refazer a interface que foi feita sem utilizar o MIRA, você acredita que faria em menos tempo que utilizando seus conhecimentos prévios de construção de interface para web?
   Não cheguei a fazer para comparar.

7) Qual dificuldade haveria em modificar a interface da aplicação com novas regras de interface em ambas as aplicações?
   Só fiz a aplicação utilizando o MIRA, e criar novas regras seria bem simples.

8) Qual dificuldade você encontrou em ambas as aplicações para desenvolver uma interface que se adapta ao dispositivo utilizado?
   No MIRA não me senti tendo que adaptar a interface para vários dispositivos, nem pareceu dar trabalho. Apenas criei a regra, fiz o mapeamento e já estava adaptado para vários dispositivos.

9.2.4 Candidato 4

Formação: Ciência da Computação na IST

Área: Frontend Developer

9.2.4.1 Nivelamento

8) Anos como desenvolvedor

5 anos

9) Nível de conhecimento com projetista de aplicação

Nenhuma

10) Nível de conhecimento de desenvolvimento para Web

Avançado

11) Nível de conhecimento de desenvolvimento com JavaScript

Intermediário

12) Nível de conhecimento de desenvolvimento para Web Mobile

Nenhum

13) Nível de conhecimento de desenvolvimento de interfaces responsivas

Básico

14) Nível de conhecimento do SHDM e/ou Interface concreta e abstrata

Nenhum

9.2.4.2 Interfaces

Imobiliária – Com o MIRA

Tempo: 1h26m

Ocorrências:

- O candidato esqueceu das regras de negócio durante o desenvolvimento da interface, mas ao ser aviso, conseguiu modificar de forma rápida e concluiu a tarefa.

Futebol – Sem o MIRA

Tempo: 1h57m

Ferramentas: Jquery, Underscore

Observações:

- O candidato teve de ser avisado de que esqueceu as regras de negócio para a construção da interface. E teve algumas dificuldades para acrescentar as regras na estrutura que havia construído.

9.2.4.3 Perguntas:

1) O layout final em ambas as aplicações ficou igual ao que foi proposto?
   Sim

2) Houve diferença na documentação ou na dificuldade do entendimento das interfaces?
   Sem diferenças

3) Quais foram as principais dificuldades que encontrou dentro da tarefa proposta?
   Aplicar as regras de negócio a interface.

4) Quais foram as dificuldades no uso do MIRA?
   Construir as interfaces abstratas e concretas. E a falta de uma forma de generalização e reutilização de trechos das interfaces.

5) Quais foram os benefícios no uso do MIRA?
   Realizar as tarefas foram mais fáceis por conta de como o MIRA lida com as regras de negócio. Como também utilizei um framework, as

dificuldades de implementação foram as mesmas, mas quando estava usando o MIRA, na hora de lidar com as regras de negócio, foi mais fácil.

6) Se tivesse que refazer a interface que foi feita sem utilizar o MIRA, você acredita que faria em menos tempo que utilizando seus conhecimentos prévios de construção de interface para web?
   Sem dúvidas

7) Qual dificuldade haveria em modificar a interface da aplicação com novas regras de interface em ambas as aplicações?
   Com o MIRA precisaria apenas criar e mapear widgets para cada regra. Sem ele, eu teria que trabalhar muito em cima do código

8) Qual dificuldade você encontrou em ambas as aplicações para desenvolver uma interface que se adapta ao dispositivo utilizado?
   Com o MIRA parecia que estava trabalhando para a WEB, só foi adicionar uma regra e tudo se resolveu.

9.2.5 Candidato 5

Formação: Ciência da Computação na Estácio de Sá – Cursando
Área: Frontend Developer

9.2.5.1 Nivelamento

1) Anos como desenvolvedor
   8 anos

2) Nível de conhecimento com projetista de aplicação
   Apenas em projetos pessoais

3) Nível de conhecimento de desenvolvimento para Web
   Avançado

4) Nível de conhecimento de desenvolvimento com JavaScript
   Avançado

5) Nível de conhecimento de desenvolvimento para Web Mobile
   Básico

6) Nível de conhecimento de desenvolvimento de interfaces responsivas
   Teórico

7) Nível de conhecimento do SHDM e/ou Interface concreta e abstrata
   Nenhum

9.2.5.2 Interfaces

Imobiliária – Com o MIRA

Tempo: 1h07m

Ocorrências:

- O candidato esqueceu das regras de negócio durante o desenvolvimento da interface.
- O candidato não usou a estrutura para reuso de regras.

Futebol – Sem o MIRA

Tempo: 1h18m

Ferramentas: AngularJS

Observações:

- Teve ajuda do instrutor no final para corrigir a estrutura da interface abstrata.
- O candidato teve de ser avisado de que esqueceu as regras de negócio para a construção da interface. E teve algumas dificuldades para acrescentar as regras na estrutura que havia construído.

9.2.5.3 Perguntas:

1) O layout final em ambas as aplicações ficou igual ao que foi proposto?
   Sim

2) Houve diferença na documentação ou na dificuldade do entendimento das interfaces?
   Não

3) Quais foram as principais dificuldades que encontrou dentro da tarefa proposta?
   Saber quais widgets utilizar.

4) Quais foram as dificuldades no uso do MIRA?
   Não tive dificuldades após entender o que é interface concreta e abstrata.

5) Quais foram os benefícios no uso do MIRA?
   Declaração unificadas das interfaces e reutilização de componentes.

6) Se tivesse que refazer a interface que foi feita sem utilizar o MIRA, você acredita que faria em menos tempo que utilizando seus conhecimentos prévios de construção de interface para web?
   Resolveria mais fácil o mesmo problema. E o código ficaria mais limpo.

7) Qual dificuldade haveria em modificar a interface da aplicação com novas regras de interface em ambas as aplicações?
   Com o MIRA seria simples, usando outras ferramentas, o código ficaria com regras desorganizadas, deixando o código mais difícil de realizar manutenção.

8) Qual dificuldade você encontrou em ambas as aplicações para desenvolver uma interface que se adapta ao dispositivo utilizado?

Com o MIRA precisei fazer uma checagem, usando Angular, eu tive que preparar uma estrutura para realizar a troca de elementos de acordo com o dispositivo.

9.3 **MÉTRICAS DE CÓDIGO**

Mean per-function logical LOC: 6.9957097222392
Mean per-function parameter count: 3.2681046366808326
Mean per-function cyclomatic complexity: 1.8713502725182367
Mean per-function Halstead effort: 1548.4304091033202
Mean per-module maintainability index: 115.74174637151181
First-order density: 0%
Change cost: 3.225806451612903%
Core size: 0%

- **File:mira\application.js**
  Physical LOC: 76
  Logical LOC: 43
  Mean parameter count: 3.2
  Cyclomatic complexity: 3
  Cyclomatic complexity density: 6.976744186046512%
  Maintainability index: 110.74472329087017
  Dependency count: 0
- **File:mira\helper.js**
  Physical LOC: 260
  Logical LOC: 180
  Mean parameter count: 1.9655172413793103
  Cyclomatic complexity: 46
  Cyclomatic complexity density: 25.555555555555554%
  Maintainability index: 116.30976505580924
  Dependency count: 2
- **File:mira\init.js**
  Physical LOC: 42
  Logical LOC: 14

Mean parameter count: 11

Cyclomatic complexity: 1

Cyclomatic complexity density: 7.142857142857142%

Maintainability index: 107.9679675516607

Dependency count: 0

- **File:mira\interface.js**

  Physical LOC: 42

  Logical LOC: 22

  Mean parameter count: 3

  Cyclomatic complexity: 1

  Cyclomatic complexity density: 4.545454545454546%

  Maintainability index: 123.26275507191727

  Dependency count: 0

- **File:mira\server.js**

  Physical LOC: 96

  Logical LOC: 70

  Mean parameter count: 2.4285714285714284

  Cyclomatic complexity: 11

  Cyclomatic complexity density: 15.714285714285714%

  Maintainability index: 125.83704318453314

  Dependency count: 13

- **File:mira\base\init.js**

  Physical LOC: 64

  Logical LOC: 32

  Mean parameter count: 2

  Cyclomatic complexity: 5

  Cyclomatic complexity density: 15.625%

  Maintainability index: 111.91490997351629

  Dependency count: 2

- **File:mira\base\view.js**

  Physical LOC: 122

  Logical LOC: 71

  Mean parameter count: 1.2222222222222223

  Cyclomatic complexity: 11

Cyclomatic complexity density: 15.492957746478872%

Maintainability index: 112.4700300212366

Dependency count: 0

- **File:mira\models\abstract.js**

  Physical LOC: 78

  Logical LOC: 43

  Mean parameter count: 1.6

  Cyclomatic complexity: 4

  Cyclomatic complexity density: 9.30232558139535%

  Maintainability index: 125.6372842668992

  Dependency count: 4

- **File:mira\models\api.js**

  Physical LOC: 29

  Logical LOC: 16

  Mean parameter count: 1.5

  Cyclomatic complexity: 3

  Cyclomatic complexity density: 18.75%

  Maintainability index: 116.41614525760419

  Dependency count: 1

- **File:mira\models\concrete.js**

  Physical LOC: 42

  Logical LOC: 22

  Mean parameter count: 2

  Cyclomatic complexity: 2

  Cyclomatic complexity density: 9.090909090909092%

  Maintainability index: 122.03478689069115

  Dependency count: 0

- **File:mira\models\map.js**

  Physical LOC: 41

  Logical LOC: 18

  Mean parameter count: 3

  Cyclomatic complexity: 2

  Cyclomatic complexity density: 11.11111111111111%

Maintainability index: 126.3047286977154

Dependency count: 0

- **File:mira\models\rule.js**

  Physical LOC: 50

  Logical LOC: 31

  Mean parameter count: 2.25

  Cyclomatic complexity: 4

  Cyclomatic complexity density: 12.903225806451612%

  Maintainability index: 114.95522385091832

  Dependency count: 2

- **File:mira\models\selection.js**

  Physical LOC: 51

  Logical LOC: 30

  Mean parameter count: 1.6

  Cyclomatic complexity: 7

  Cyclomatic complexity density: 23.333333333333332%

  Maintainability index: 119.99853117172802

  Dependency count: 2

- **File:mira\models\widget-abstract.js**

  Physical LOC: 208

  Logical LOC: 130

  Mean parameter count: 2.238095238095238

  Cyclomatic complexity: 22

  Cyclomatic complexity density: 16.923076923076923%

  Maintainability index: 116.13167433730607

  Dependency count: 5

- **File:mira\widgets\bootstrap-base.js**

  Physical LOC: 108

  Logical LOC: 66

  Mean parameter count: 2.3846153846153846

  Cyclomatic complexity: 17

  Cyclomatic complexity density: 25.757575757575758%

  Maintainability index: 121.26692534099308

  Dependency count: 0

- **File:mira\widgets\bootstrap-carousel.js**

  Physical LOC: 72

  Logical LOC: 35

  Mean parameter count: 3.2

  Cyclomatic complexity: 7

  Cyclomatic complexity density: 20%

  Maintainability index: 116.50287523576694

  Dependency count: 0

- **File:mira\widgets\bootstrap-footer.js**

  Physical LOC: 30

  Logical LOC: 14

  Mean parameter count: 4

  Cyclomatic complexity: 2

  Cyclomatic complexity density: 14.285714285714285%

  Maintainability index: 118.93860568485198

  Dependency count: 0

- **File:mira\widgets\bootstrap-form.js**

  Physical LOC: 54

  Logical LOC: 33

  Mean parameter count: 3

  Cyclomatic complexity: 5

  Cyclomatic complexity density: 15.151515151515152%

  Maintainability index: 116.50705486050882

  Dependency count: 0

- **File:mira\widgets\bootstrap-image-box.js**

  Physical LOC: 35

  Logical LOC: 19

  Mean parameter count: 4

  Cyclomatic complexity: 4

  Cyclomatic complexity density: 21.052631578947366%

  Maintainability index: 109.23427187494198

  Dependency count: 0

- **File:mira\widgets\bootstrap-modal.js**

  Physical LOC: 111

  Logical LOC: 41

  Mean parameter count: 3.2222222222222223

  Cyclomatic complexity: 8

  Cyclomatic complexity density: 19.51219512195122%

  Maintainability index: 125.15639203777454

  Dependency count: 0

- **File:mira\widgets\bootstrap-navigation.js**

  Physical LOC: 82

  Logical LOC: 46

  Mean parameter count: 4.5

  Cyclomatic complexity: 10

  Cyclomatic complexity density: 21.73913043478261%

  Maintainability index: 104.2485911035024

  Dependency count: 0

- **File:mira\widgets\freebase.js**

  Physical LOC: 41

  Logical LOC: 20

  Mean parameter count: 3.75

  Cyclomatic complexity: 2

  Cyclomatic complexity density: 10%

  Maintainability index: 122.49501377343076

  Dependency count: 0

- **File:mira\widgets\head.js**

  Physical LOC: 49

  Logical LOC: 31

  Mean parameter count: 3.5

  Cyclomatic complexity: 2

  Cyclomatic complexity density: 6.451612903225806%

  Maintainability index: 99.62433911253191

  Dependency count: 0

- **File:mira\widgets\image-html.js**

  Physical LOC: 25

  Logical LOC: 13

  Mean parameter count: 4

  Cyclomatic complexity: 3

  Cyclomatic complexity density: 23.076923076923077%

  Maintainability index: 118.85202631649337

  Dependency count: 0

- **File:mira\widgets\input.js**

  Physical LOC: 33

  Logical LOC: 18

  Mean parameter count: 3.3333333333333335

  Cyclomatic complexity: 3

  Cyclomatic complexity density: 16.666666666666664%

  Maintainability index: 119.9141579630936

  Dependency count: 0

- **File:mira\widgets\map.js**

  Physical LOC: 67

  Logical LOC: 40

  Mean parameter count: 3.25

  Cyclomatic complexity: 5

  Cyclomatic complexity density: 12.5%

  Maintainability index: 107.85156297919248

  Dependency count: 2

- **File:mira\widgets\meta.js**

  Physical LOC: 33

  Logical LOC: 19

  Mean parameter count: 3

  Cyclomatic complexity: 1

  Cyclomatic complexity density: 5.263157894736842%

  Maintainability index: 119.35484461706733

  Dependency count: 0

- **File:mira\widgets\profile.js**

  Physical LOC: 114

  Logical LOC: 71

  Mean parameter count: 4.666666666666667

  Cyclomatic complexity: 14

  Cyclomatic complexity density: 19.718309859154928%

  Maintainability index: 103.73736357193164

  Dependency count: 0

- **File:mira\widgets\render.js**

  Physical LOC: 96

  Logical LOC: 56

  Mean parameter count: 4.5

  Cyclomatic complexity: 4

  Cyclomatic complexity density: 7.142857142857142%

  Maintainability index: 109.65393850898415

  Dependency count: 0

- **File:mira\widgets\simple-html.js**

  Physical LOC: 42

  Logical LOC: 22

  Mean parameter count: 4.5

  Cyclomatic complexity: 6

  Cyclomatic complexity density: 27.27272727272727%

  Maintainability index: 105.1642421041176

  Dependency count: 0

- **File:mira\widgets\title.js**

  Physical LOC: 27

  Logical LOC: 13

  Mean parameter count: 3.5

  Cyclomatic complexity: 2

  Cyclomatic complexity density: 15.384615384615385%

  Maintainability index: 119.50636380927806

  Dependency count: 0