UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

JAIRO DE JESUS NASCIMENTO DA SILVA JUNIOR

UMA ARQUITETURA ORIENTADA A SERVIÇOS PARA VISUALIZAÇÃO DE DADOS EM DISPOSITIVOS INTELIGENTES

Belém - PA
2014

JAIRO DE JESUS NASCIMENTO DA SILVA JUNIOR

UMA ARQUITETURA ORIENTADA A SERVIÇOS PARA VISUALIZAÇÃO DE DADOS EM DISPOSITIVOS INTELIGENTES

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciências da Computação, da Universidade Federal do Pará, como parte dos requisitos para a obtenção do título de Mestre em Ciências da Computação.

ORIENTADOR: Prof. Dr. Bianchi Serique Meiguins

Belém - PA

2014

Jairo de Jesus Nascimento da Silva Junior

Uma Arquitetura Orientada a Serviços para Visualização de Dados em Dispositivos Inteligentes

Dissertação submetida à Banca Examinadora do Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas e Naturais pertencente à Universidade Federal do Pará (UFPA) para obtenção do Grau de Mestre em Ciência da Computação

BANCA EXAMINADORA:

_____
Prof. Dr. Bianchi Serique Meiguins (PPGCC – UFPA)
Orientador

_____
Prof. Dr. Nelson Cruz Sampaio Neto (PPGCC – UFPA)
Membro Interno

_____
Prof. Dr. Mario Massakuni Kubo (Faculdade Alvorada – DF)
Membro Externo

RESUMO

A evolução tecnológica dos smartphones e tablets, a grande quantidade de informações armazenadas eletronicamente, e a necessidade de tomada de decisão, individual ou colaborativa, em qualquer lugar e momento demandam a concepção e desenvolvimento de serviços para visualização desses dados. A Internet tem desempenhado um importante papel como rede de compartilhamento de conhecimento, e neste contexto tem surgido aplicações com arquiteturas orientadas a serviços (SOA) nos mais diversos campos de estudo. Assim, este trabalho tem como objetivo a concepção e desenvolvimento de um agregado de serviços que favoreçam a ubiquidade e pervasividade em aplicações de visualização de dados, permitindo que o usuário construa visualizações de dados sobre um determinado domínio de problema de maneira fácil e intuitiva. Com este tipo de serviço é possível construir aplicações de visualizações de dados em diferentes dispositivos inteligentes, tais como: smartphones, tablets, desktop, TV Digital e etc. Esta abstração é conseguida através de uma API Web que suporta as principais características de aplicações de visualização de informações em diferentes plataformas. O modelo arquitetural de comunicação utilizado na concepção do serviço foi o REST (**RE**presentational **S**tate **T**ransfer), as aplicações cliente e servidora foram desenvolvidas na linguagem Java, e foi utilizado um motor de geração de visualizações de dados denominado PRISMA. Como cenário de uso, foi desenvolvida uma aplicação cliente Android para testar os serviços criados. Por fim, serão apresentados dados iniciais sobre testes de usabilidade realizados na aplicação desenvolvida.

Palavras-chave: Visualização de dados, SOA, REST, dispositivos inteligentes.

ABSTRACT

Technological developments of smartphones and tablets, the great amount of electronic stored information and the need for decision-making, individually or collaboratively, anywhere and anytime, demands the conception and development of services for data visualization. The Internet has played an important role as a knowledge-sharing network and in this context, some service oriented architecture (SOA) applications have emerged in all kind of study fields. Therefore, this work aims the conception and development of a service aggregation that will favor ubiquity and pervasiveness in data visualization applications, allowing users to build domain-specific data visualizations in an easy and intuitive way. With this kind of service, it is possible to build data visualization applications for different smart devices such as smartphones, tablets, desktop, smart TV's, etc. A Web API that supports the main characteristics of an information visualization tool in different platforms reaches this abstraction. REST (**RE**presentational **S**tate **T**ransfer) Style was employed in the service conception as the architectural communication model. Client-side and server-side applications were developed using Java with a data visualization generator engine called PRISMA. As a use case, was developed a client Android application in order to test the created services. Lastly, initial data about usability essays performed in the developed application will be presented.

Keywords: Data Visualization, SOA, REST, smart devices.

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

Publicações

PRISMA Mobile: An Information Visualization Tool for Tablets. Proceedings of 16th International Conference on Information Visualisation (IV), 2012.

SUMÁRIO

1 INTRODUÇÃO

A quantidade de dados armazenados no mundo continua crescendo exponencialmente, acompanhada em grande parte pelo aumento de dispositivos inteligentes conectados à internet. Em alguns países, a quantidade desses dispositivos ultrapassou sua população, evidenciando a utilização de múltiplos dispositivos por usuário.

Neste cenário as empresas têm buscado constantemente alternativas para transformar dados brutos em informações úteis. O domínio dessa informação tem se mostrado um importante aliado, seja para ganhar vantagem no mercado ou para resolução de diversos problemas da sociedade. Porém, a dificuldade de análise dos dados tem se mostrado uma constante à medida que o fator de crescimento dos dados aumenta, assim como a complexidade dos problemas a serem resolvidos, tornando mais difícil fornecer informações precisas em tempo real, em qualquer lugar e momento.

Dentre as técnicas utilizadas para análise de dados, podemos destacar a visualização da informação, definida como um conjunto de tecnologias que utiliza a computação visual para amplificar a capacidade humana de trabalhar com informações abstratas (JACKO, 2012). A visualização da informação promete ajudar-nos a acelerar o entendimento e tomada de ações em um mundo com um crescente volume de informações.

Dispositivos inteligentes tem se tornado parte do dia a dia e do mundo de negócios. Usuários desses dispositivos vem utilizando-os como forma de dar continuidade as suas tarefas em qualquer local que estejam, suportando tarefas nos mais diversos campos de conhecimentos.

A internet tem atuado como ponto central para armazenamento das informações do usuário de forma a permitir essa continuidade, e dessa forma, desempenhado um importante papel como rede de compartilhamento de conhecimento, dando oportunidade para o surgimento de aplicações com arquitetura orientada a serviços nos mais diversos campos de estudo.

Embora esses dispositivos inteligentes possuam algumas características interessantes como mobilidade ou conectividade, sendo capazes de até mesmo substituir computadores de entrada, estes possuem recursos limitados, como telas reduzidas, baterias de curta duração, baixo poder de processamento, baixa capacidade de armazenamento, tráfego de dados limitados e mecanismos de interação ainda pouco explorados.

Os contextos físicos e sociais estão envolvidos no processo de visualização de dados. Enquanto o contexto físico é uma fonte de dados, o contexto social determina de maneiras sutis e complexas o que é coletado e como estes dados são interpretados. Logo, pode-se utilizar estes dispositivos inteligentes não apenas para tomar decisões em tempo real, mas para melhorar a percepção humana, de maneira individual ou através de um processo colaborativo.

Neste trabalho um conjunto de serviços de visualização de dados foram projetados e construídos, serviços estes que têm como objetivo abstrair as etapas do processo de visualização de dados, desde a análise e filtragem dos dados até o mapeamento destes para suas representações visuais e a sua própria representação. Buscou-se seguir os princípios SOA e REST durante o desenvolvimento desses serviços, com intuito de construir uma arquitetura reúsavel e interoperável com componentes autônomos que minimizem a retenção de estado ao máximo.

A partir da introdução dessa camada de serviço em uma ferramenta de visualização de informações pré-existente, o PRISMA (GODINHO, MEIGUINS, *et al.*, 2007), foi possível remodelar o modelo conceitual em alto nível, de forma a ocultar detalhes de implementação, permitindo que fosse criada uma API de interação motores de visualização de dados, viabilizando a utilização de outros provedores de visualização de dados.

Nessa arquitetura orientação a serviços, de um lado temos a API de interação com provedores de visualização que trabalha em conjunto com adaptadores para converter as interfaces de um provedor para as interfaces agnósticas extraídas do modelo conceitual, evitando dessa forma que os conceitos introduzidos no modelo conceitual sejam corrompidos com detalhes de implementação.

Do outro lado, a API de interação com os atores do sistema possui adaptadores que realizam conversões entre o modelo conceitual e alguma tecnologia específica, no caso deste trabalho, REST foi utilizado, mas assim como o provedor de visualizações, poderia ser substituído com a implementação de novos adaptadores.

Para interagir com estes serviços, um cliente leve foi desenvolvido na plataforma *Android* baseado em duas principais diretrizes: (1) não possuir lógica de negócio, apenas lógica da interface do usuário e (2) atingir os requisitos de uma boa ferramenta de visualização de informações (visão geral, filtros, zoom e detalhes sob demanda).

Ao final, testes de usabilidades preliminares foram realizados com a ferramenta para dispositivos inteligentes foi construída.

1.1 Objetivos

O objetivo principal desta dissertação é construir uma arquitetura orientada a serviços para visualização de dados em dispositivos inteligentes.

1.1.1 Objetivos específicos

Como objetivos específicos destacam-se:

- Modelar o domínio do processo de visualização de dados e de uma ferramenta de visualização de informações como um serviço.

- Construir um serviço de visualização de dados.
- Construir um cliente deste serviço para dispositivos móveis.
- Realização de testes preliminares de usabilidade.

1.2 Organização

O texto da dissertação está organizado na forma que segue:

No capítulo 2 temos um apanhado geral sobre visualização de informações, processo de visualização de dados, tipos de dados, técnicas, ferramentas de visualização de informações e múltiplas visões coordenadas.

No capítulo 3 tem-se uma breve introdução a arquitetura orientada a serviços com foco no estilo arquitetural REST e seus princípios.

O capítulo 4 irá expor aspectos conceituais do protótipo desenvolvido, tanto do consumidor (cliente) quanto do serviço (servidor), bem como detalhes da arquitetura e projeto do serviço.

No capítulo 5 entraremos em detalhes sobre o projeto e implementação do cliente, tecnologias utilizadas e buscando mostrar como este atende aos requisitos de uma boa ferramenta de visualização de informações.

O capítulo 6 irá apresentar a metodologia utilizada e os resultados dos testes de usabilidade preliminares realizados.

Por último serão apresentadas as considerações finais, desafios encontrados e propostas de trabalhos futuros.

2 VISUALIZAÇÃO DA INFORMAÇÃO

2.1 Definição de Visualização da Informação

Desenvolver sistemas de visualização de dados baseando-se na capacidade humana de percepção e sistemas de processamento de informações é um desafio crescente, demandado pela quantidade de dados gerados e armazenados eletronicamente a cada dia em diversos formatos e fontes de dados.

Quando se propõe novas técnicas de visualização de informações e mecanismos de interação com os dados, é necessário entender melhor como o ser humano interage com a informação, como ele a percebe visualmente, como a mente trabalha quando está procurando por informações conhecidas e desconhecidas, como a mente resolve os problemas e como o ser humano compreende os dados apresentados (Brath, 1997). Como qualquer outro software, um software para de visualização de dados deve possibilitar interfaces flexíveis que se adeque aos diversos tipos de perfis de usuário, do iniciante ao mais experiente, e incluindo ferramentas de navegação, métodos de pesquisa, e modelo de dados apropriados para cada um dos tipos de usuários existentes, para realização de suas tarefas.

De acordo com Few (Few, 2009), de maneira geral, o termo Visualização é aplicado para representação visual da informação, e pode ser associado a três palavras principais, criando três termos com significados um pouco diferentes, são eles: Visualização de Dados, Visualização da Informação e Visualização Científica.

Ainda de acordo com (Few, 2009), o termo Visualização de Dados pode ser utilizado como um termo guarda-chuva para cobrir todos os tipos de representações visuais que suportam a exploração, análise e comunicação de dados. Independente da representação, contanto que seja visual, e independente do que ela representa, contanto que seja uma informação, ela constitui-se uma Visualização de Dados. Os termos Visualização da Informação e Visualização Científica são subconjuntos de Visualização de Dados. Eles se referem a tipos particulares de representações visuais com propósitos específicos.

(Card, Mackinlay, & Shneiderman, 1999) e (Spence, 2007) definem e diferenciam os termos Visualização de Informação e Visualização Cientifica. Visualização de Informação é o uso de um ambiente computacional interativo, que possibilita representação visual de dados abstratos para amplificar a cognição. A Visualização Científica é definida como representação visual de dados científicos que são geralmente de natureza física, em vez de abstratos. Por exemplo, uma imagem de ressonância magnética e uma imagem de raio-x são considerados Visualização

Científica porque eles apresentam dados que possuem forma física, buscando representar fielmente esta forma de uma maneira que seja fácil de visualizar, reconhecer e compreender.

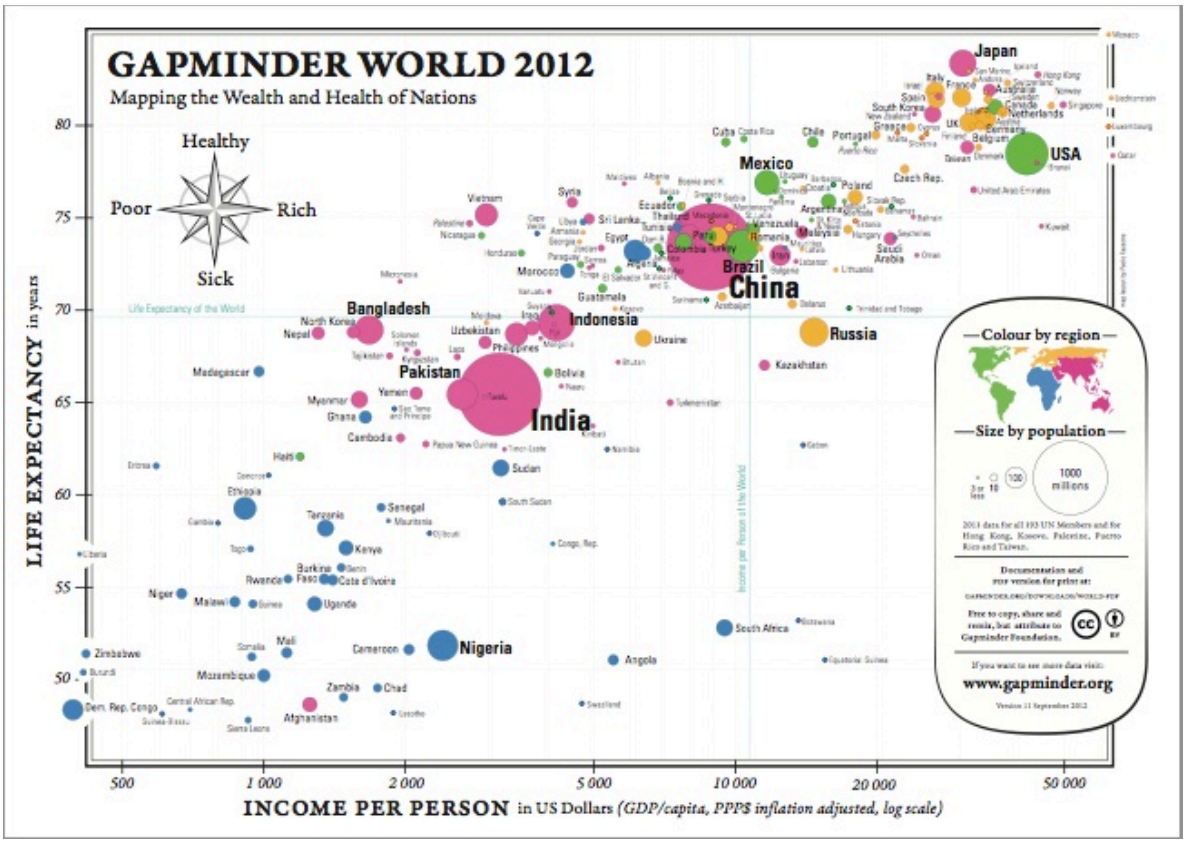

Figura 1: Exemplo de visualização de informações (GAPMINDER, 2012).

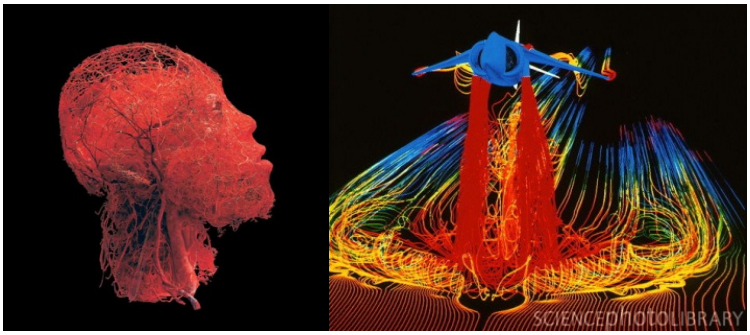

Figura 2: Exemplos de visualização científica (BODYWORLDS, 2013).

A Figura 1 exemplifica uma visualização de informações, onde possuímos um gráfico onde as cores representam regiões do planeja, o eixo X representa a renda per capita e o eixo Y a expectivata de vida. Já a Figura 2 mostra um exemplo de visualização científica, onde dados físicos são renderizados para facilitar o entendimento e ilustrção de conceitos.

2.2 Processo de Visualização de Dados

As etapas de processo de visualização de dados são apresentadas na Figura 3.

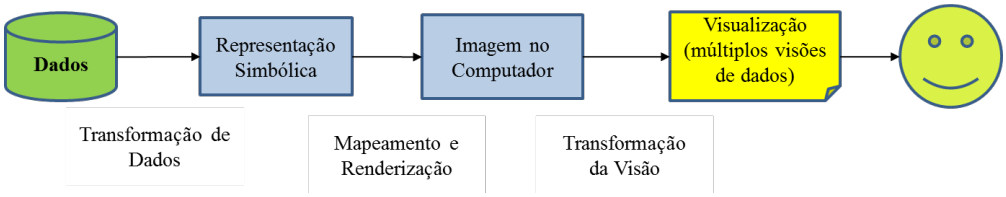

Figura 3: Etapas do processo de visualização. (Ward, Grinstein , & Keim, 2010)

Normalmente o projetista concentra seus esforços iniciais nos tipos de dados que estão disponíveis para visualização e que tipo de informação o usuário espera extrair da visualização que será proposta.

Um passo posterior importante para efetiva visualização de dados é definição de como os dados serão representados visualmente, e como essas representações serão inseridas no contexto da técnica visualização selecionada. Os atributos dos dados são utilizados para definir os objetos gráficos que serão visualizados, tais como: pontos, linhas, e formas, bem como as características destes objetos, tais como: tamanho, posição, orientação e cor. Alguns autores apresentam outras formas de representação, tais como: textura, transparência, animação, som, etc.

Outra importante etapa do processo de visualização é a definição dos controles de interação para visualização do conjunto de dados e seus relacionamentos. Os controles de interação são importantes porque o processo de visualização apoiado por computador é bastante dinâmico, na qual o usuário pode interferir em todas as etapas do processo, restringindo o conjunto de dados de análise, analisando em detalhes um item de dados específico, entrar itens similares de dados, etc.

2.2.1 Pipeline de Visualização

Os estágios do pipeline de visualização são os seguintes (Figura 4):

- **Modelagem de dados**: o dado para ser visualizado, de um arquivo ou banco de dados, tem que estar estruturado para facilitar a sua visualização. O nome, tipo, faixa de valor, e a semântica para cada atributo ou campo de um dado gravado devem estar disponíveis no formato que garanta rápido acesso e fácil modificação.
- **Seleção de Dados**: a seleção de dados envolve identificar um subconjunto dos dados potências que serão visualizados. Isto pode ocorrer com total controle do usuário ou via

métodos algorítmicos, por exemplo, detectar automaticamente características de interesse do usuário.

- **Mapeamento Visual do Dado**: um dos estágios mais importantes do pipeline de visualização é execução do mapeamento visual do dado gravado, que pode ser mapeado para característica de tamanho, posição, cor, forma, etc. Esse mapeamento envolve a aplicação de técnicas de computação gráfica, tais como: transformações de escala, rotação, translação, etc.
- **Configuração de parâmetros da Cena (Transformação da Visão)**: como um gráfico tradicional, o usuário deve especificar vários atributos da visualização que são relativamente independentes dos dados. Alguns exemplos: mapeamento de cor (para muitos domínios de problema, certas cores já estão claramente definidas), mapeamento de som (quando houver informações a serem transmitidas por canal de áudio, por exemplo, a completude correta de uma tarefa), especificações de luz (para visualizações tridimensionais), etc.
- **Renderização ou geração da visualização**: a projeção especifica ou renderização da visualização varia de acordo com o mapeamento que está sendo utilizado, que podem envolver técnicas de mapeamento de sombra e textura, embora muitas técnicas de visualização necessitam somente desenhar linhas e polígonos com sombreamento uniforme. Além disso, a maioria das visualizações incluem informações complementares para facilitar a interpretação dos dados, tais como: eixos, anotações, imagens, etc.

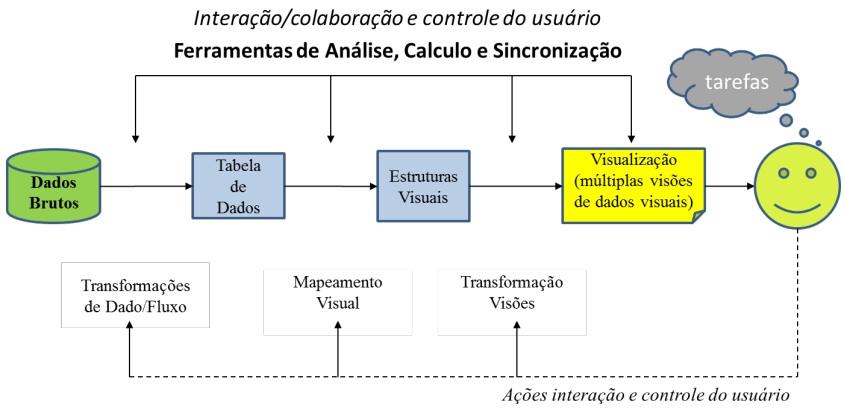

Figura 4: Pipeline de visualização. (Ward, Grinstein , & Keim, 2010)

2.3 Tipos de Dados X Tipos de Técnicas de Visualização de Informação

O formato visual com que as informações são disponibilizadas ao usuário tem influência direta na tarefa de extração do conhecimento de um determinado sistema de informação, e é dependente dos dados. (Shneiderman, 1996) classificou os dados em 7 (sete) tipos diferentes, e para

cada um deles pode-se descrever uma visualização diferente, são eles:

- **1-Dimensão:** este tipo de dado é representado por texto ou dados similares, como linhas de código. Pode haver outras informações associadas a ele, como data da criação, tamanho, data da última modificação, etc. Uma técnica bastante associada a esse tipo de dado é o uso de linhas com cores e larguras variadas, representando outros atributos. Como no exemplo da Figura 5.

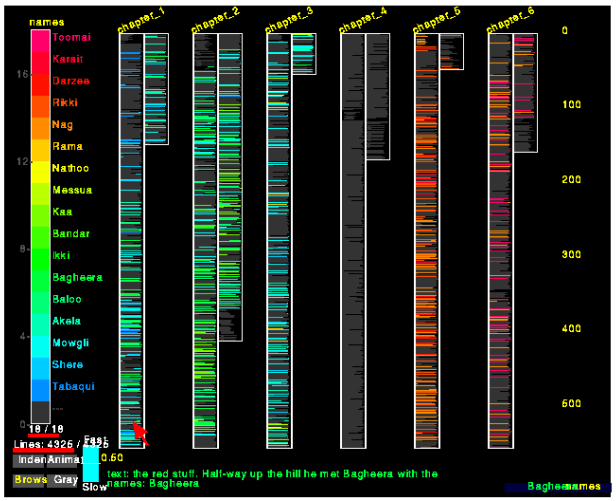

Figura 5: Software SeeSoft apresentando a localização de personagens nos capítulos de um livro (EICK, 1994).

- **2-Dimensões:** Este tipo de dado inclui dados geográficos, plantas de engenharia, etc. Pode-se associar uma grande quantidade de atributos com uso de cores, tamanhos e formas diferentes. A visualização da Figura 6 ilustra as leituras de temperatura do Mar Cáspio ao longo de sua extensão.

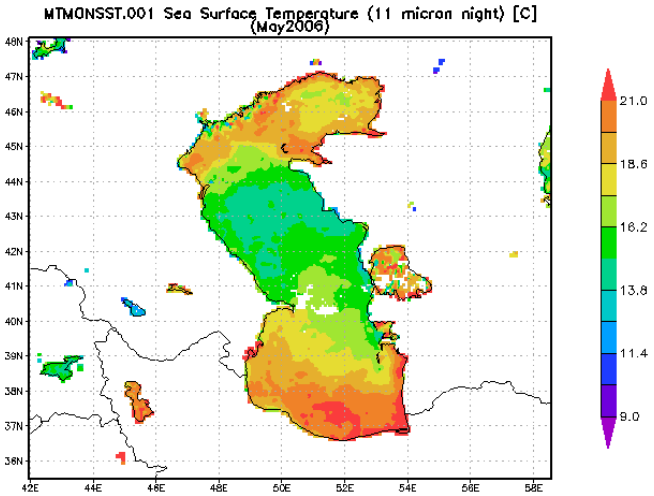

Figura 6: Temperatura do Mar Cáspio (NASA, 2006).

- **3-Dimensões:** aqui o volume de um objeto torna-se importante, um atributo a mais. Se o contexto do mundo real puder ser incluído para melhorar a percepção do usuário é mais indicado ainda. Não se deve deixar de mencionar problemas inerentes a uma visualização 3D, como a oclusão, quando parte de um dado esconde outro. Para isso técnicas de visões diferenciadas, transparência (Figura 7) e *slicing* são necessárias.

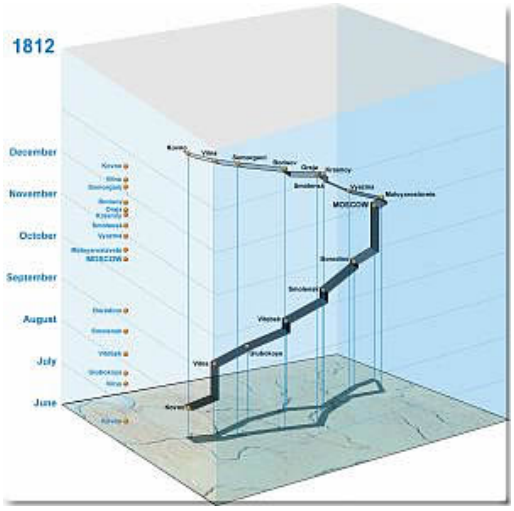

Figura 7: Visualização 3D com transparência (ITC, 2001).

- **Temporal:** este tipo de dado reúne todas as características dos dados acima mais o atributo tempo. Para o atributo tempo o mais indicado é formar uma dimensão. Os gráficos "tempo versus algum atributo" são bastante utilizados e conhecidos. A animação deve ser considerada quando há uma grande quantidade de dados. A Figura 8 utiliza o eixo X para formar uma nova dimensão com a variação anual.

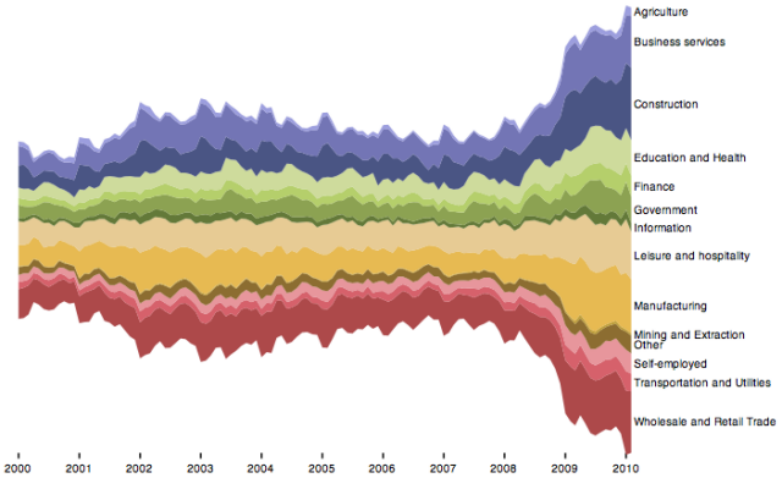

Figura 8: Visualização com variação temporal no eixo X (U.S. BUREAU OF LABOR STATISTICS, 2011).

- **Multidimensional:** base de dados relacional ou estatística pode ser considerada como pontos em um espaço multidimensional. Técnicas como consultas dinâmicas e diagramas de dispersão (Figura 9) são bastante úteis.

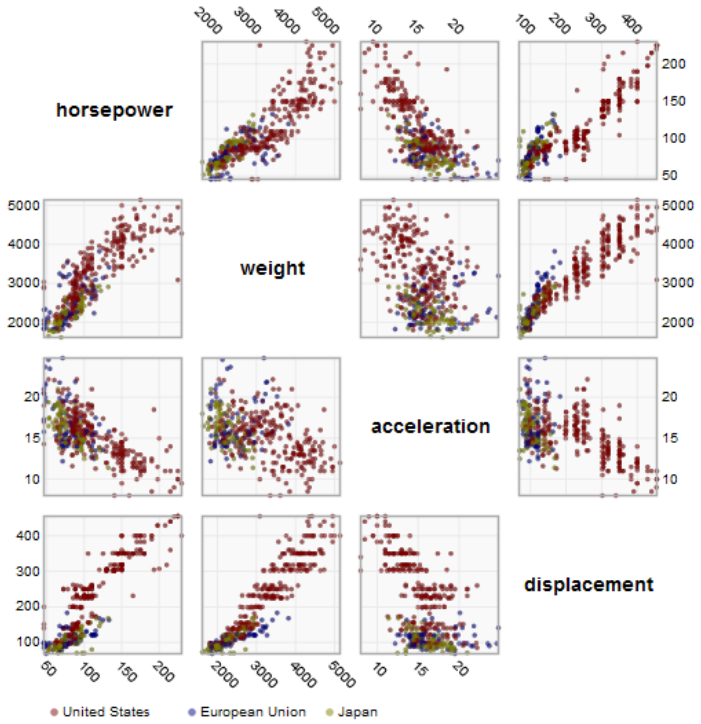

Figura 9: Visualização multidimensional representando 4 atributos (HEER, BOSTOCK e OGIEVETSKY, 2010).

- **Hierárquico:** muito útil para classificação de dados. Normalmente é representado por diagramas com nós, com ligações entre os mesmos, como na Figura 10.

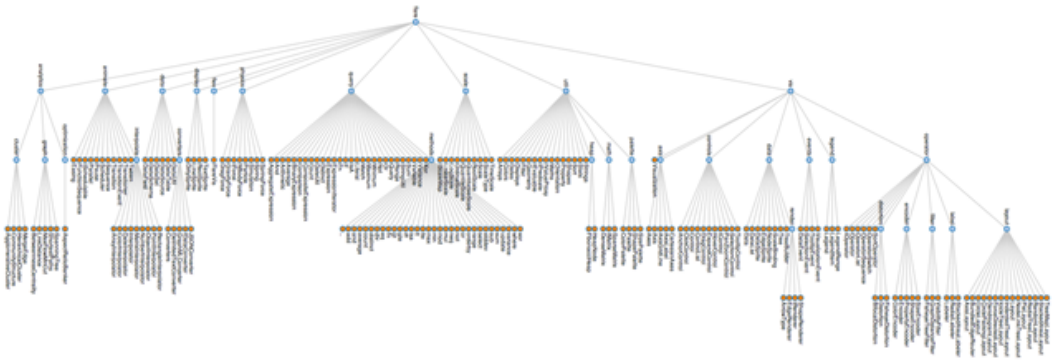

Figura 10: Visualização de dados hierárquicos (HEER, BOSTOCK e OGIEVETSKY, 2010).

- **Rede:** dados de rede são nós conectados por *links* previamente definidos. Esses *links* podem ser organizados em árvores ou em hierarquias, e a melhor maneira de manipulação é permitindo mudar o foco sobre os nós. A Figura 11 exemplifica estes dados de rede interconectados.

Figura 11: Visualização de uma rede em um grafo (GRADNJEAN, 2014).

2.4 Técnicas de Visualização de Dados

Esta seção descreve as três técnicas de visualização que foram utilizadas neste trabalho, são elas: dispersão de dados, treemap e coordenadas paralelas.

2.4.1 Dispersão de Dados

O gráfico de dispersão visualiza uma relação (correlação) entre duas variáveis X e Y, por exemplo, peso e altura. São representados pontos de dados individuais no espaço bidimensional onde os eixos representam as variáveis (X no eixo horizontal e Y no eixo vertical) (Figura 12). Os "pontos do gráfico" X-Y são ícones configuráveis através dos atributos de cor, tamanho, forma, entre outras, permitindo aos usuários reconhecer diversas informações simultaneamente.

Figura 12: Técnica de dispersão em anos diferentes representando o relacionamento de indicadores do IDH.

2.4.2 Treemap

Treemap é uma técnica de visualização de informação de preenchimento de áreas capaz de representar grandes coleções hierárquicas de dados quantitativos. A técnica divide a área de exibição em uma sequência aninhada de retângulos, cujas áreas correspondem a um valor de atributo do conjunto de dados (Figura 13) (SHNEIDERMAN, 2001). E a visualização do treemap é dependente do algoritmo a ser utilizado, destacam-se: *slice and dice*, *cluster* e *squarified*. A Figura 13 apresenta o resultado da aplicação do algoritmo *squarified* em uma base de dados, e tem como objetivo subdividir o espaço de apresentação em retângulos que possuam um bom aspecto relacional, ou seja, que possuam um tamanho de largura e altura próximos.

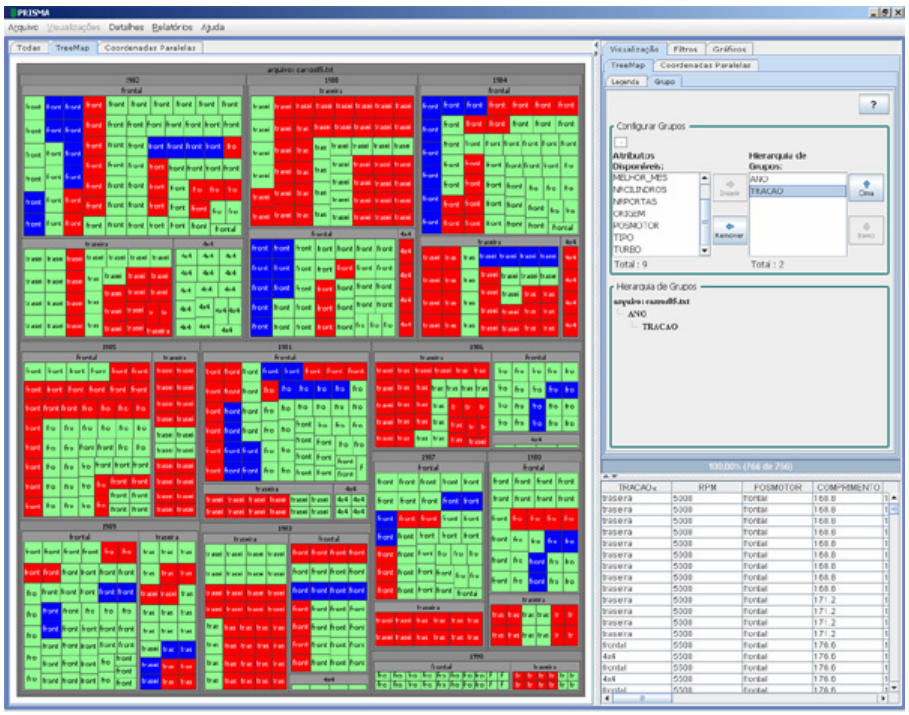

Figura 13: Técnica Treemap no PRISMA.

2.4.3 Coordenadas Paralelas

A técnica de coordenadas paralelas foi proposta por Alfred Inselberg em 1981 como uma nova maneira de visualizar informações multidimensionais (INSELBERG, 1990). Nessa técnica, cada dimensão é representada por um eixo vertical. Ao contrário da técnica de dispersão de dados, os eixos devem estar paralelos e equidistantes entre si. Dessa forma, é possível visualizar muitas variáveis simultaneamente.

A técnica consiste em interligar os eixos aos seus adjacentes através de linhas retas (Figura 14). Essas linhas são traçadas de acordo com os registros da base de dados. Um eixo possui uma escala que contém todos os valores possíveis para um determinado atributo. Nessa abordagem é considerado um atributo uma coluna da base de dados com seus respectivos valores.

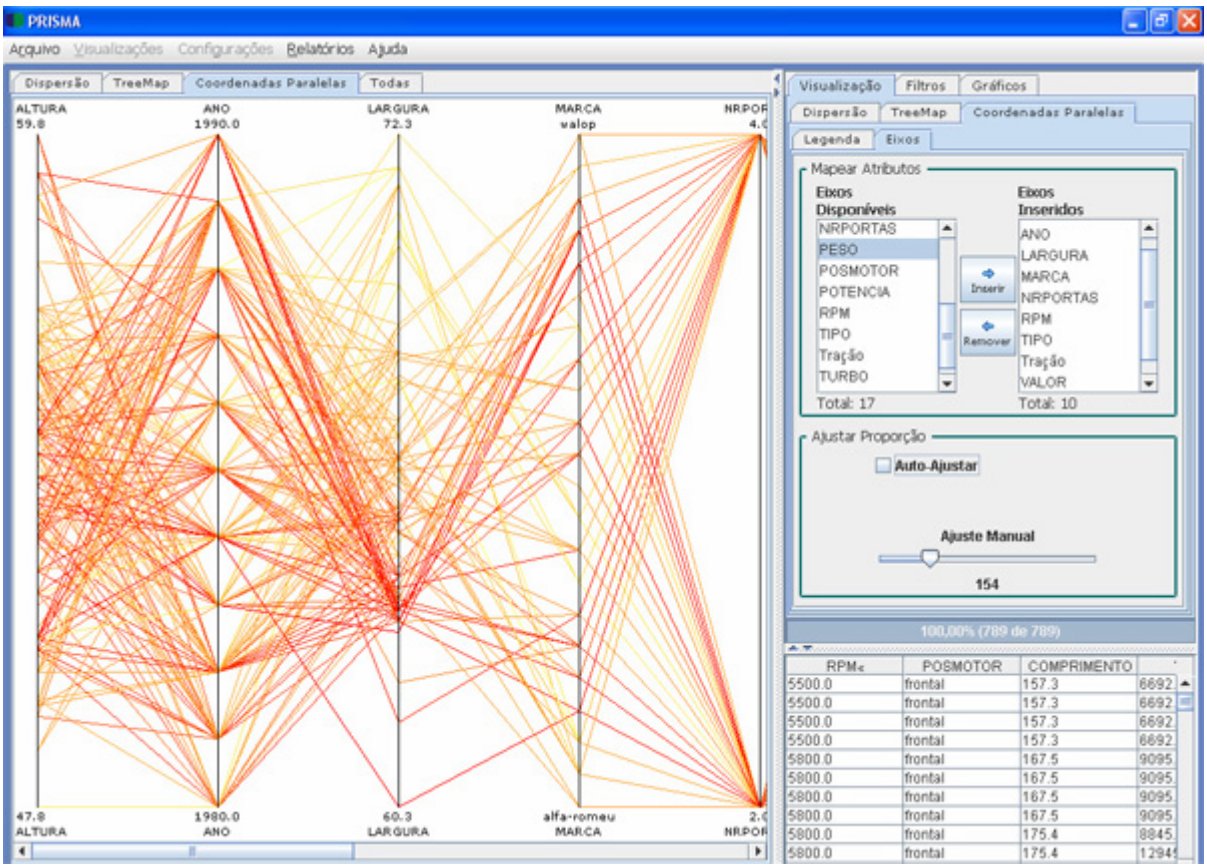

Figura 14: Técnica de coordenadas paralelas na ferramenta PRISMA (GODINHO, 2007).

Em certos casos, em que a base de dados é muito grande, a visualização pode se tornar confusa, devido à grande quantidade de linhas que ficarão sobrepostas. Assim, a solução para esse tipo de situação é a realização de filtros dinâmicos e atribuição de cores para a redução e diferenciação do conjunto de dados.

2.5 Requisitos para uma boa Ferramenta de Visualização

Segundo (Shneiderman, 1996), uma boa ferramenta de visualização deve possuir características definidas em função das tarefas executadas pelo usuário, entre elas destaca-se:

- **Visão geral:** o usuário precisa ganhar uma noção sobre todos os dados que serão analisados. Essa noção está baseada nos parâmetros que o usuário escolheu para a visualização, nos limites do dispositivo gráfico usado e de sua percepção. Os atributos gráficos mais usados são: posição, cor, tipo de representação e tamanho.
- **Zoom:** a técnica de zoom é importante porque permite focar em certo subconjunto dos dados para análise, ou seja, analisar um determinado contexto. Além disso, conforme se aplica o zoom, mais detalhes sobre uma determinada visão dos dados são mostrados, o que se chama de zoom semântico.

- **Filtro:** usuários frequentemente precisam reduzir o tamanho do conjunto de dados, eliminando itens com base em seus atributos. Uma das maneiras mais eficientes é o uso de consultas dinâmicas.
- **Detalhes sob demanda:** quando o usuário está explorando um conjunto de dados, ele necessita ver detalhes sobre um item em particular. Isto é normalmente feito usando o clique do mouse, onde as informações adicionais podem aparecer em uma janela auxiliar, ou na própria visão dos dados (visualização).

Carr (1999) sugere duas outras características que uma boa ferramenta de visualização de informação deveria ter:

- **Relacionamentos:** se o usuário descobre um item de interesse, ele pode precisar saber sobre outros itens com atributos similares, a ferramenta então poderia apontar esses itens similares.
- **Histórico:** manter o histórico de ações para suportar desfazer ou refazer e refinamento progressivo.

2.6 Múltiplas Visões Coordenadas

Sistemas de múltiplas visões usam duas ou mais visões distintas para auxiliar o processo de investigação de uma única entidade conceitual (BALDONADO, WOODRUFF e KUCHINSKY, 2000).

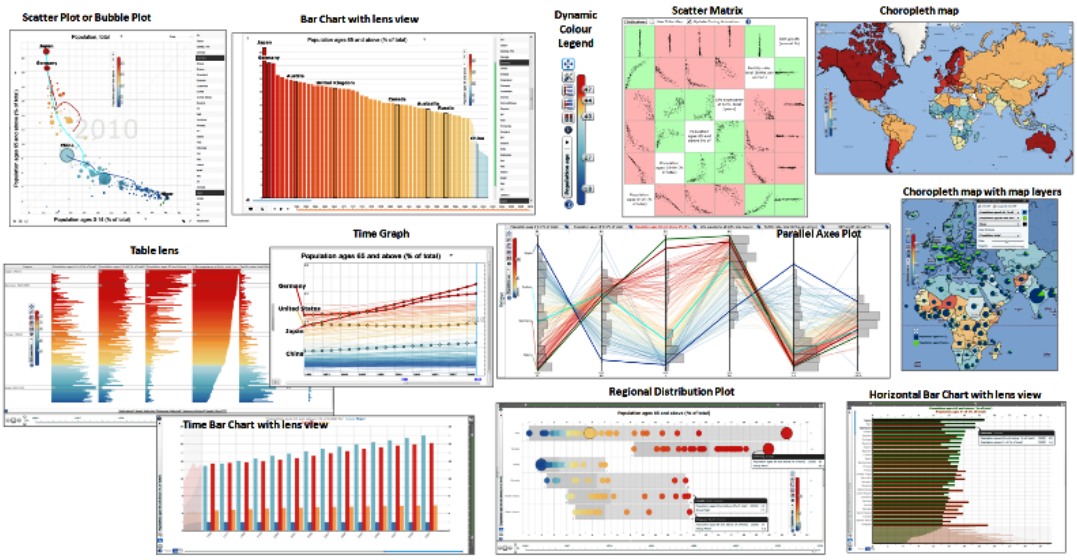

Figura 15: Exemplo de múltiplas visões coordenadas (BALDONADO, WOODRUFF e KUCHINSKY, 2000).

As principais características de coordenação em visualização de informação são flexibilidades em relação aos dados - utilização de mais de um conjunto de dados, visões - escolher tipo de representação visual para determinado conjunto de dados, e coordenação - definir as características da coordenação entre cada par de visões (North e Shneiderman, 2000).

Para o desenvolvimento de sistemas de visualização de informação com múltiplas visões coordenadas, são frequentemente encontradas as seguintes recomendações (Baldonado et al, 2000):

a) Usar múltiplas visões quando há uma diversidade de atributos, modelos, perfis de usuário, níveis de abstração ou gênero;

b) Usar múltiplas visões quando as visões diferentes destacam correlações ou disparidades;

c) Usar múltiplas visões quando os dados são complexos, recomenda-se particionar os dados em múltiplas visões para criar visões de dados gerenciáveis e fornecer ideias durante a interação entre as diferentes dimensões;

d) Usar múltiplas visões minimamente, justificar o uso de múltiplas visões versus custo de aprendizado do usuário e espaço de visualização.

As principais possibilidades de coordenação entre as visões são apresentadas a seguir (Pillat, et al. 2005):

a) Seleção: itens de dados selecionados em uma visão são destacados em outras visões.

b) Filtro: itens de dados com valores dentro de um intervalo específico não terão suas formas desenhadas nas visões.

c) Cor, Transparência e tamanho: itens utilizados para representar a variação de valores de um dado atributo do conjunto de dados (escolhido pelo usuário), pode se usar o mesmo critério para representar uma mesma informação em várias visões.

d) Ordenação: a ordem que em que os dados são desenhados na visão pode ser determinada pelos valores de um atributo no conjunto de dados. A sobreposição de imagens dos itens de dados nas visões indica a relação entre eles.

e) Rótulo: determina como e que conteúdo os rótulos exibirão para cada item de dados das visões, e deve ser especificado pelo usuário.

f) Manipulação de Atributos: Permite ao usuário para adicionar/remover atributos da representação visual. Além disso, o usuário também pode modificar a ordem na qual são exibidos tais atributos.

2.7 Trabalhos relacionados

Dentre os trabalhos relacionados, podemos destacar três principais:

- Em (LEE, MOCTEZUMA e LASTRA, 2013) os autores apresentam uma proposta visualização de informações em um sistema de controle de produção orientado a serviços. Neste trabalho uma prova de conceito foi desenvolvida para geração de contratos de serviços de visualização de informação utilizando informações que consumiam via web services de outros sistemas, além do monitoramento destes, dados dos usuários e ambiente, com intuito de aperfeiçoar o processo de geração dos contratos. A ideia dos contratos dinâmicos permitia um nível de abstração um pouco acima da pipeline de visualização, pois permitia que o usuário recebesse um contrato personalizado para consumir serviços de visualização de dados.

- (THIEDE, TOMINSKI e SCHUMANN, 2009) desenvolveram uma arquitetura orientada a serviços onde diferentes fases da pipeline de visualização eram realizados por componentes distribuídos, utilizando um ambiente inteligente em conjunto, ambiente este que consistia em uma sala de reuniões com a presença de alguns dispositivos inteligentes interconectados. Um aspecto importante da arquitetura apresentada é a utilização de técnicas de visualização diferentes para cada tipo de dispositivo evidenciando que os requisitos de uma visualização variam bastante de acordo com o seu contexto de uso. Alguns dos princípios da orientação a serviço são aplicados no protótipo apresentado, como autonomia e abstração, embora se baseie em uma tecnologia pouco interoperável.

- (POMBINHO, 2010) desenvolveu um trabalho sobre visualização de informação em ambientes móveis, evidenciando as limitações desses ambientes, relacionadas a recursos computacionais e mecanismos de interação. Após apresentar as limitações seu principal objetivo foi explorar formas de adaptar uma visualização a partir de dados do contexto físico, temporal e histórico de uso. As mesmas limitações apontadas por Pombinho ainda se aplicam e apesar da evolução tecnológica dos dispositivos estes ainda continuam sendo utilizados como um cliente leve, embora as interfaces venham ficando cada vez

mais ricas.

3 ARQUITETURA ORIENTADA A SERVIÇOS

3.1 Caracterização

De acordo com (ERL, 2005) SOA (*Service-Oriented Architecture*) é um termo que representa um modelo em que a automação da lógica é decomposta em pequenas e distintas unidades de lógica. Coletivamente, essas unidades formam uma grande peça de automação de negócios. Individualmente, essas unidades podem ser distribuídas.

Já para (SIMON, 2005), SOA é um framework técnico e organizacional que possibilita uma empresa entregar funcionalidades auto descritivas e independentes de plataforma, tornando-as elementos necessários para construção das aplicações atuais e futuras.

A orientação a serviços possui alguns princípios chaves que serão empregados no decorrer deste trabalho, são eles:

- **Fraco acoplamento**: Minimizar dependências entre o consumidor e o serviço.
- **Contrato de serviço**: Serviços utilizam um contrato que estabelece como se dará a comunicação entre consumidor e serviço.
- **Autonomia**: Serviços exercem um alto nível de controle sobre o ambiente de execução destes.
- **Abstração**: Contratos de serviços contém apenas informação essencial e são limitados ao que está contido no contrato.
- **Reusabilidade**: Serviços possuem e expressam lógica agnóstica de tecnológica e podem ser classificados como recursos corporativos reusáveis.
- **Componibilidade**: Serviços são participantes eficientes de composições.
- **Ausência de estado**: Serviços minimizam a retenção de informação.
- **Habilidade de poder ser descoberto**: Serviços são enriquecidos com *metadados* pelos quais eles podem ser descobertos e interpretados.

Web Services podem implementar uma arquitetura orientada a serviços. Um Web Service (WS) é um método de comunicação entre duas possíveis aplicações heterogêneas através da World Wide Web (WWW), para acesso por um cliente a funcionalidades de um servidor, sendo este acesso independente de plataforma e linguagem de programação. O W3C (World Wide Web Consortium) define WS como um “sistema de software projetado para suportar interoperabilidade máquina-a-máquina sobre uma rede de computadores. Tem uma interface descrita em um formato que pode ser processado por uma máquina. Outros sistemas interagem com WS utilizando mensagens SOAP (Simple Object Access Protocol), transmitidos via HTTP com serialização XML

em conjunto com outros padrões Web relacionados".

SOAP é uma especificação de protocolo para troca de informação estruturada para implementação de Web Services usa XML no formato das mensagens, e é dependente do protocolo de transporte (HTTP, FTP, TCP, UDP, etc.). Uma mensagem SOAP contém o dado, a ação que deve ser executada, o cabeçalho, e os detalhes de erro em caso de falha. SOAP disponibiliza mecanismos para os serviços se auto descreverem aos clientes (WSDL) e informar a existência dos serviços (UDDI - *Universal Description, Discovery and Integration*). Também fornece mecanismos de segurança e confiabilidade fim a fim, respectivamente através dos protocolos WS-Security e WS-ReliableMessaging.

O problema do padrão SOAP é que ele adiciona um overhead considerável, tanto por ser em XML quanto por adicionar muitas tags de meta-informação. Além disso, a serialização e desserialização das mensagens pode consumir um tempo considerável. Uma alternativa ao uso do SOAP é protocolo de comunicação REST (Representational State Transfer).

REST é um estilo arquitetural para projetar aplicações distribuídas, como as aplicações na Web/Internet, sendo caracterizada por envolver clientes e servidores enviando mensagens de pedidos e mensagens de reposta a esses pedidos, e não impõe restrições ao formato da mensagem, apenas sobre o comportamento dos componentes envolvidos. Assim, a maior vantagem do protocolo REST é sua flexibilidade, podendo adaptar-se a qualquer tipo de formato de mensagem, e quase sempre Web Services que usam REST são mais "leves" e, portanto, mais rápidos.

REST reconhece tudo como recurso. Cada recurso implementa uma interface uniforme padrão (interface HTTP), possuem nomes e endereços (URI), e possuem uma ou mais representações (JSON, XML, TXT, etc), e a representação do recurso é distribuída pela rede utilizando-se HTTP. E dá-se o nome de RESTFul a implementação de Web Services no estilo arquitetural REST.

3.1.1 Conceitos e Propriedade de uma Arquitetura REST:

É composta por quatro conceitos. (RICHARDSON, 2007):

- **Recursos**: é algo que pode ser armazenado em um computador e representado por uma sequência de bits, por exemplo, um documento, uma imagem, um registro de uma tabela ou o resultado da execução de um algoritmo.
- **URIs**: representa o nome e endereço de um recurso, o item não pode ser considerado um recurso se não possuir uma URI. As URIs devem ser descritivas, possuir uma correspondência clara com seus recursos, e intuitivas para os usuários. Ex: http:/www.exemplo.com/search/webservices

- **Representações**: é uma serie de bits em uma linguagem especifica, que podem ser transportados pela Web, e que representam um estado atual de um determinado recurso. Ou seja, um recurso é apenas um conceito abstrato de algo real, já a representação são os itens de dados que informam algo sobre o recurso.
- **Links entre elas**: há hyperlinks entre os recursos disponibilizados.

E possui quatro propriedades:

- **Endereçamento**: o recurso é endereçável quando um usuário pode acessar os seus dados através de uma determinada URI.
- **Falta de Estado**: significa que toda requisição HTTP feita por um cliente ao servidor tem incluída todas as informações necessárias para seu processamento e resposta. Desta forma, o servidor nunca guarda as informações das requisições passadas, pois para uma nova requisição o cliente deverá mandar todas as informações necessárias.
- **Encadeamento**: os recursos podem permitir acesso a outros dados e recursos através de links com esses recursos.
- **Interface Uniforme**: é responsável pode definir as operações possíveis para um recurso. O HTTP fornece as seguintes operações:
    - GET: recupera uma representação de um recurso;
    - PUT: cria ou atualiza um recurso
    - DELETE: apaga um recurso existente
    - POST: cria um recurso filho
    - HEAD: recupera metadados de uma representação
    - OPTIONS: verifica quais operações um determinado recurso suporta

3.2 RESTFul APIs

REST é um conjunto de princípios independentes de tecnologias, exceto pelo requisito de ser baseado no protocolo HTTP. Para ser chamado de RESTFul é necessário aderir aos seguintes princípios:

- Todos os componentes são unicamente identificados por um link hypermedia (URL), que por sua vez, possuem representações em diferentes formatos (XML, JSON, TXT e etc).

- Todas as comunicações não mantêm estado, ou seja, nenhum dado do cliente é armazenado no servidor, toda a informação necessária é enviada junto da requisição do cliente ao servidor, seja na URL, cabeçalhos ou corpo da requisição.

- A arquitetura é distribuída e o cache dos dados pode ser feito em qualquer camada. Em outras palavras, o cliente pode realizar cache das respostas, o servidor pode realizar cache dos resultados de requisições e elementos intermediários, como *proxies*, podem ser inseridos na arquitetura para realizar cache.
- Uma arquitetura cliente/servidor é utilizada, reduzindo o acoplamento entre esses componentes.
- A arquitetura é distribuída em camadas de forma que em momento algum o cliente pode diferenciar se está conectado com o serviço finalístico ou um intermediário.
- Todos os componentes do sistema se comunicam através de uma interface com métodos claramente definidos (operações HTTP) e código dinâmico.

Uma API (Application Programming Interface) especifica um componente de software em termos das suas operações, entradas, saídas e tipos associados. Seu principal objetivo é definir um conjunto de funcionalidades independente da forma como são implementados, permitindo que a definição e implementação sejam modificadas sem comprometer uma a outra.

No contexto Web, pode-se definir um API em termos de um conjunto de mensagens trocadas utilizando o protocolo HTTP e formatos abertos, como XML e JSON. Assim, dá-se o nome de RESTFul APIs, as APIs Web que aderem aos princípios REST.

A tabela 1 apresenta os métodos HTTP que são tipicamente utilizados para implementação de uma RESTful API:

Tabela 1: Exemplo de RESTFul API.

Recursos	GET	PUT	POST	DELETE
Coleção de URI http://exemple.com/resources	Lista as URIs e talvez outros detalhes dos itens da coleção	Substitui toda coleção por outra coleção	Cria uma nova entrada na coleção	Deleta uma coleção inteira
Elemento URI http://exemple.com/resources/item17	Recupera a representação de um item endereçado de uma coleção, expresso	Substitui um item endereçado de uma coleção, ou se ele não existe, cria o	Não é geralmente usado.	Deleta um item endereçado da coleção

	em um tipo de mídia de Internet apropriado	mesmo.		

Os métodos do HTTP podem ser classificados quanto a sua segurança e idempotência. Uma solicitação é considerada segura, quando o método utilizado não altera o estado do recurso. Quando uma mesma solicitação ao ser executada várias vezes não altera o estado do recurso a denominamos idempotente. Todo método seguro é também idempotente (RICHARDSON, 2007). Os métodos GET, HEAD e OPTIONS são seguros e idempotentes, pois não alteram o estado do recurso. PUT e DELETE não são seguros, mas são idempotentes. O método POST é o único método que não é seguro nem idempotente.

4 ASPECTOS DE CONCEPÇÃO E MODELAGEM

O cenário de uso para criação de serviços aderindo aos princípios REST, foi da criação de uma aplicação móvel cliente-servidor, onde é possível gerar visualizações interativas a partir de um conjunto de dados em formato estruturado. Ou seja, o cliente móvel realiza requisições através da interação do usuário, a resposta a essa requisição é um recurso (imagem) que representa uma visualização de dados, a partir daí, novas requisições são feitas com intuito de manipular a visualização. Serão apresentados mais detalhes de concepção e modelagem da aplicação, para um melhor entendimento da aplicação criado para consumir este serviço. Esta aplicação cliente é baseada no trabalho desenvolvido em (DA SILVA JUNIOR, MEIGUINS, *et al.*, 2012).

4.1 Aspectos Conceituais da Aplicação Cliente

O processo descrito anteriormente pode ser divido em duas etapas: (1) extração de dados e (2) geração de visualizações. A extração de dados consiste na obtenção de dados em um formato estruturado que servirá como entrada para o motor de geração de visualizações do PRISMA (GODINHO, MEIGUINS, *et al.*, 2007).

Pode-se ainda desmembrar esse protótipo em três componentes, (1) cliente para dispositivos móveis, (2) serviço web para visualização de informações e (3) motor de geração de visualizações, que neste trabalho será o PRISMA.

A Figura 16 demonstra a separação de responsabilidades destes componentes, onde o cliente (1) irá consumir um serviço exposto pelo serviço para visualização de informações (2), que por sua vez, utilizará o PRISMA (3) como motor para geração dessas visualizações.

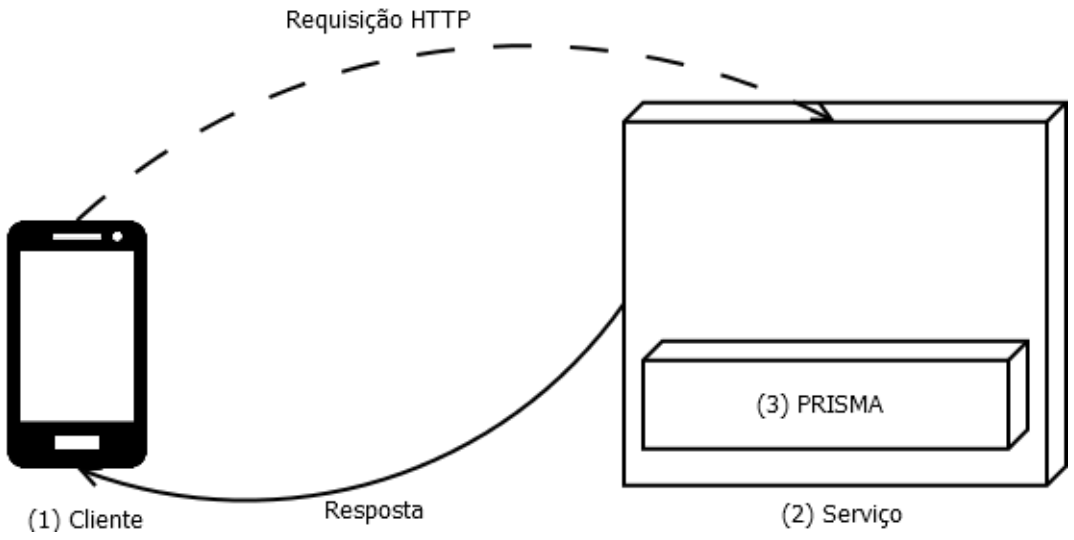

Figura 16: Conjunto de aplicações do protótipo

Do ponto de vista do usuário, o módulo cliente do protótipo é composto de 5 (cinco) casos de uso básicos (Figura 17):

1. Carregar conjunto de dados: permite o usuário carregar um novo conjunto de dados a partir de um formato estruturado (XML, JSON, CSV e etc).
2. Filtrar dados: permite transformar um conjunto de dados em um subconjunto a partida aplicação de filtros.
3. Criar visualização: permite criar a representação visual de um conjunto de dados.
4. Visualizar representação de dados, onde será possível interagir com zoom e detalhes sob demanda.
5. Configurar técnica de visualização: permite fazer ajuste fino de uma técnica. (E.g. hierarquia no *treemap*, eixo no gráfico de dispersão, etc.).

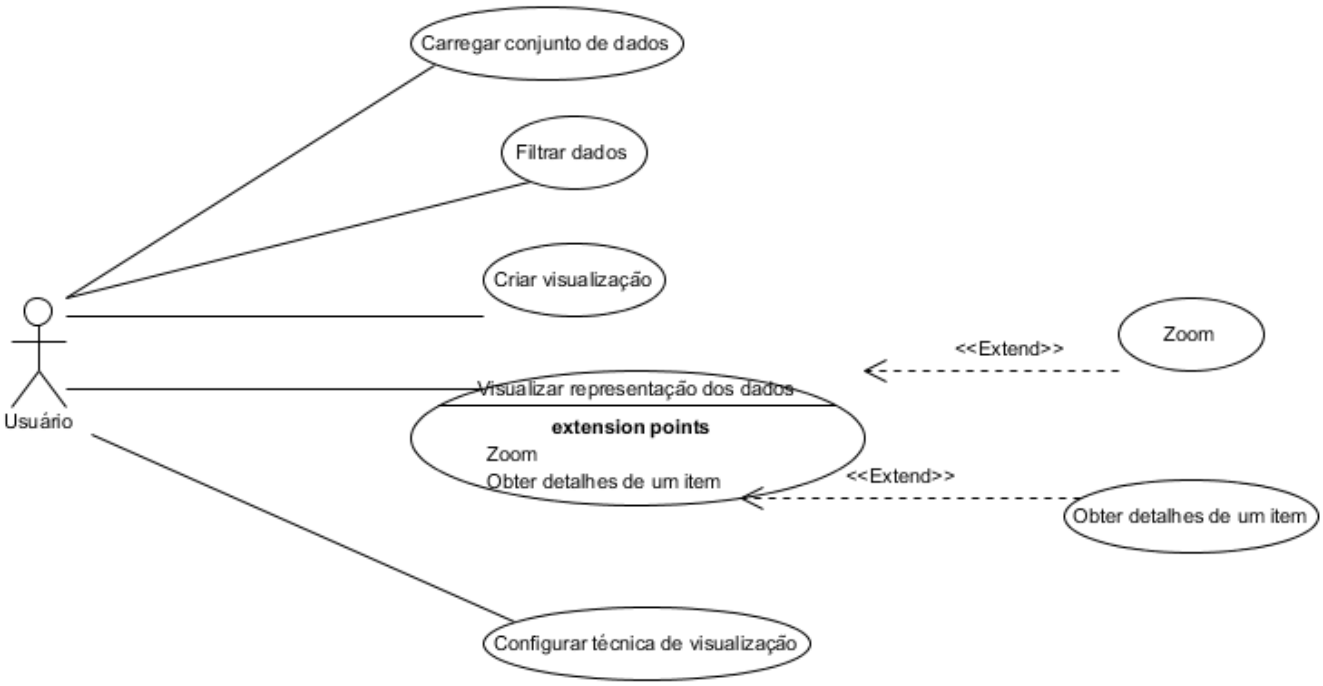

Figura 17: Diagrama de Casos de Uso do Protótipo

4.1.1 Modelo Conceitual

O modelo conceitual pode ser visto como uma representação do domínio da aplicação, mas é essencial para o entendimento do relacionamento entre o cliente e servidor e para modelagem do contrato de serviço em termos específicos do seu domínio. Este modelo representa a visão da

aplicação cliente ao consumir os recursos do servidor, desta forma, os termos empregados aqui serão empregados na construção do código fonte de ambas as aplicações (cliente e servidor).

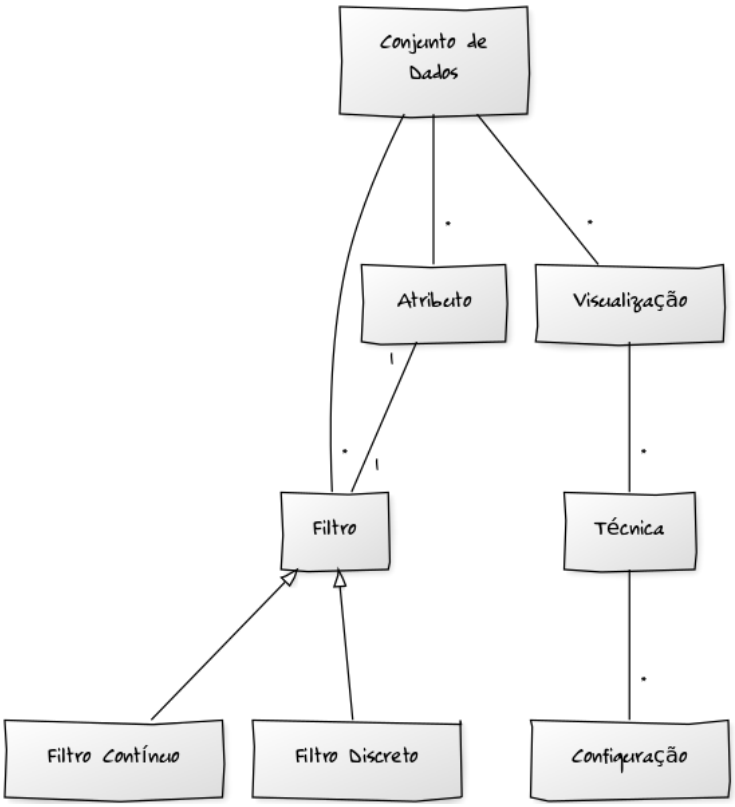

Figura 18: Modelo conceitual do domínio

O modelo conceitual descrito na Figura 18, utilizando uma sintaxe similar a UML, representa a visão do usuário ao interagir com uma ferramenta de visualização de informações.

É importante especificar as entidades do diagrama antes de explicar os seus relacionamentos, podemos caracterizá-las de uma forma como:

- **Conjunto de dados**: representa os dados disponíveis para visualização.
- **Atributo**: caracteriza uma dimensão de um conjunto de dados.
- **Filtro:** (Contínuo ou Discreto): representa um conjunto de valores para um atributo que deve ser incluído na visualização, este grupo pode ser formado por valores discretos ou uma faixa de valores para dados contínuos.
- **Visualização**: representa uma visualização de dados.
- **Técnica**: representa uma técnica de visualização de dados (e.g. treemap, dispersão de dados, coordenadas paralelas)
- **Configuração**: permite personalizar a codificação visual dos dados em uma técnica específica.

Quanto aos relacionamentos entre as entidades, podemos observar que a partir de um

conjunto de dados é possível ter n-visualizações, ou seja, diferentes representações derivadas de um mesmo conjunto de dados. Estas visualizações, por sua vez, são compostas de n-técnicas de representação visual, que terão suas dimensões visuais (i.e. rótulo, formato, eixos, etc.) definidas em um estado específico a partir de suas configurações.

Por outro lado, tem-se um conjunto de dados, composto de atributos e filtros para estes, onde pode-se manipulá-lo, seja com intuito de reduzir o número de dimensões (atributos) ou transformá-lo em um subconjunto utilizando filtros (para valores discretos e contínuos).

Por último, é válido ressaltar a ligação entre uma visualização e um conjunto de dados, permitindo que ambos evoluam, mas apenas mudanças no conjunto de dados impactem nas visualizações.

Essa decisão representa um ponto crítico no modelo, pois se ao vincular uma visualização a conjunto de dados podemos facilitar a coordenação entre as visualizações desse mesmo conjunto, do outro podemos gerar efeitos colaterais indesejados (interferir na visualização de um terceiro que compartilhe o mesmo dataset) a partir de mudanças no mesmo. A saída, neste caso, seria a criação de cópias de um conjunto de dados quando necessário.

O mesmo se aplica ao relacionamento entre uma técnica e suas configurações, que poderia ser reformulado como um relacionamento entre visualização e configurações, onde alterações em configurações de uma visualização seriam refletidas em todas as técnicas daquela visualização, facilitando a coordenação. Porém, nem sempre é possível coordenar configurações entre diferentes técnicas, visto que existem configurações que se aplicam apenas a uma ou um pequeno grupo de técnicas, a imagem que representa a coordenação entre técnicas da ferramenta PRISMA exemplifica bem esse cenário (Figura 19).

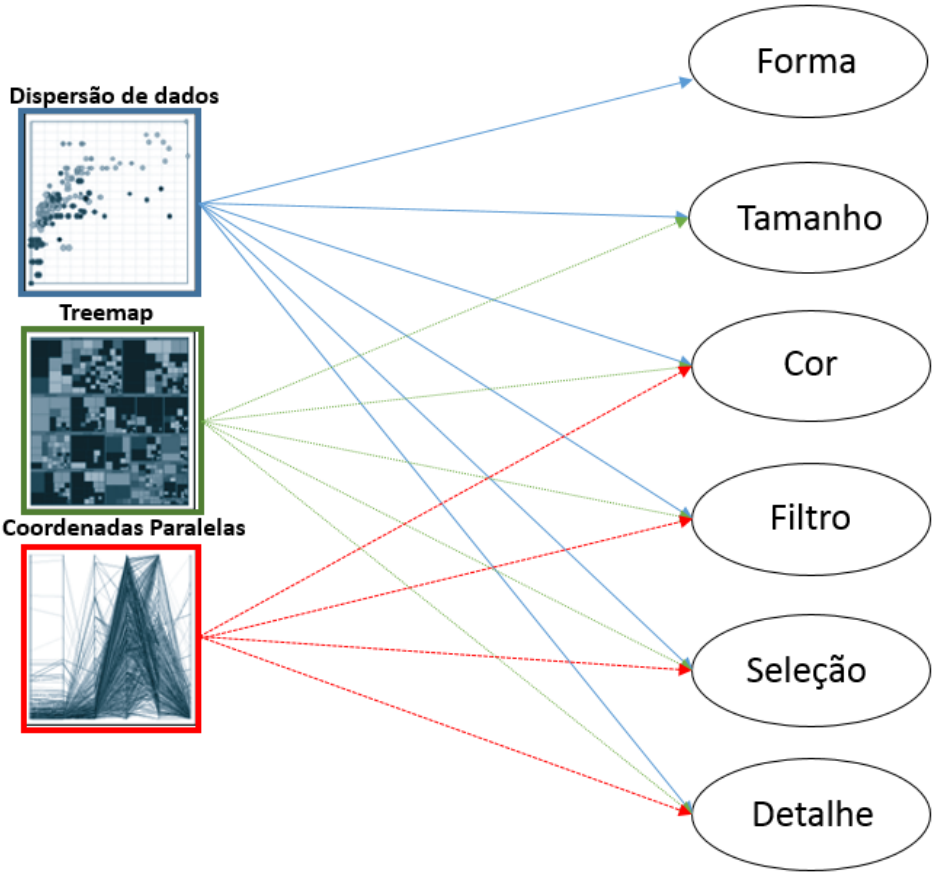

Figura 19: Coordenação entre visualizações da ferramenta PRISMA (*GODINHO, MEIGUINS,* et al., *2007)*

O modelo conceitual desenvolvido é compatível com os requisitos básicos de uma boa ferramenta de visualização de informações: (1) visão geral, (2) zoom, (3) filtros e (4) detalhes sob demanda. Onde a visão geral é obtida de uma técnica, assim como os detalhes sob demanda, já os filtros são aplicados a um *dataset* (conjunto de dados), enquanto o zoom será uma responsabilidade do cliente.

O principal objetivo desse modelo conceitual é servir como uma linguagem ubíqua (EVANS e FOWLER, 2003) para comunicação entre os interessados, facilitando a compreensão e completude do modelo, composto de elementos simples para expressar ideias complexas, como o funcionamento de uma ferramenta de visualização.

Presume-se que um modelo de domínio nunca está pronto e sempre há espaço para melhorias e evoluções, por isso acredita-se que esse modelo poderá ser incrementado ou restruturado para incorporar outros conceitos de visualização de informações não cobertos nesse trabalho, dessa forma, o modelo apresentando é apenas um subconjunto simplificado que atende aos requisitos básicos de uma ferramenta de visualização de informações.

4.2 PRISMA

PRISMA (GODINHO, MEIGUINS, *et al.*, 2007) é uma ferramenta de visualização da informação baseada em múltiplas visões coordenadas para explorar conjuntos de dados multidimensionais usando as seguintes técnicas: *treemap*, dispersão de dados e coordenadas paralelas.

Como ferramenta, o PRISMA possibilita a realização de tarefas inerentes de uma boa ferramenta de visualização da informação, como visão geral, zoom, filtros, detalhes sob demanda, relacionamentos e extração. A ferramenta aceita várias fontes de dados, desde que estejam em um formato estruturado, como arquivos de texto puro, *CSV*, conexão com base de dados e visualizações pré-configuradas no seu formato proprietário.

Entretanto, o PRISMA também pode ser visto como um framework para desenvolvimento de técnicas de visualização de informações, à medida que permite que o desenvolvedor possa adicionar facilmente novas técnicas ao projeto, implementando o contrato de interfaces pré-definidas e estendendo classes do projeto, reduzindo bastante o esforço de desenvolvimento.

Para este trabalho, o PRISMA não será utilizado com uma ferramenta de visualização de informações para propósitos gerais, ou como um framework para desenvolvimento de técnicas de visualização. Neste trabalho, o PRISMA será utilizado como um provedor de serviços.

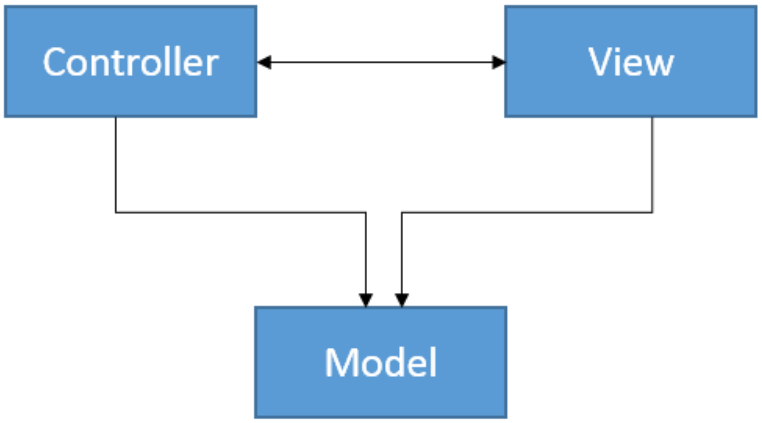

Figura 20: Padrão arquitetural MVC.

Considerando a arquitetura atual do PRISMA seguindo o padrão *MVC (Model-View-Controller) (*FOWLER, 2002), apresentado na Figura 17, foram removidas para o escopo desse trabalho todas as suas *views* e os relacionamentos dos *controllers* com estes (Figura 20).

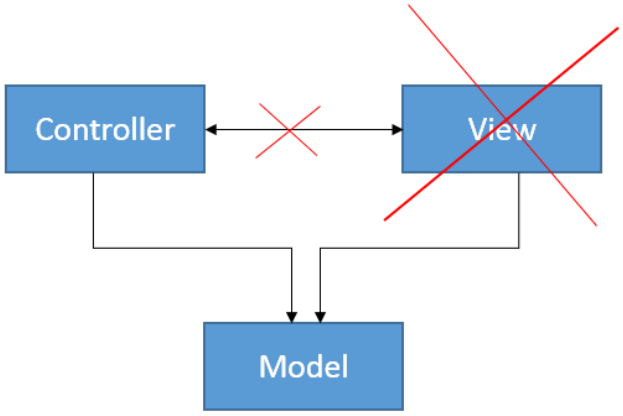

Figura 21: MVC sem as *views*.

Com a conclusão do processo de isolamento das funcionalidades do PRISMA sem a sua interface, ter-se-á apenas os elementos não riscados da Figura 21. A esse conjunto daremos o nome de *Núcleo PRISMA*.

De posse do núcleo, este será utilizado como uma dependência da nova fronteira introduzida, o Serviço PRISMA (Figura 22). Essa fronteira irá atenderá os requisitos de interoperabilidade e colaboração deste trabalho e reaproveitará as funcionalidades já implementas no PRISMA.

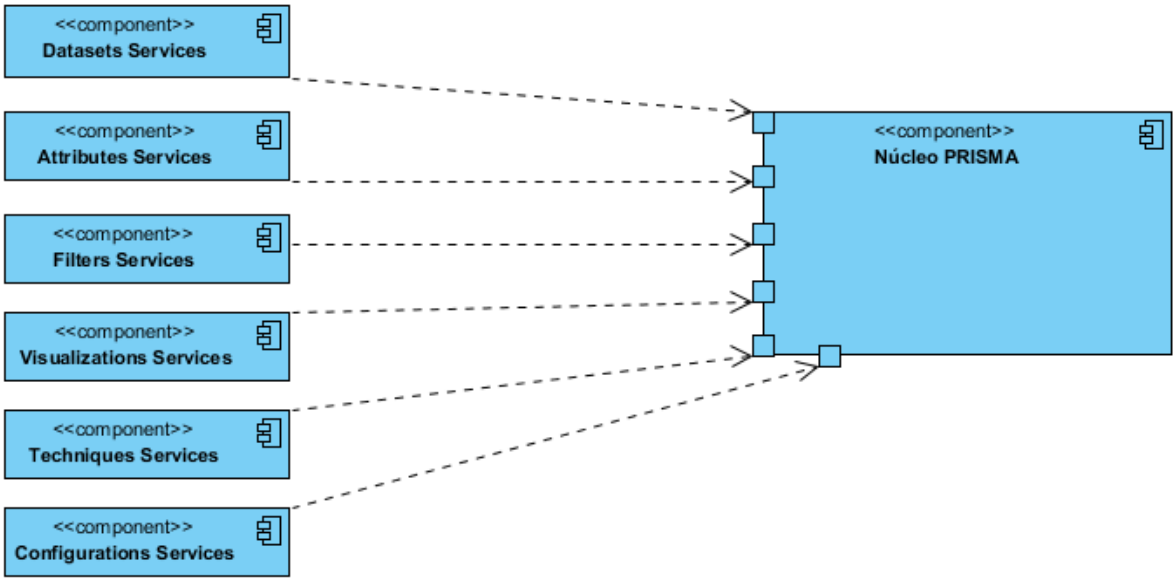

Figura 22: Dependência entre os serviços e o núcleo do PRISMA

4.3 Serviço

O modelo conceitual descrito nas seções anteriores nos remete a uma ideia de manipulação de recursos (textos, imagens, documentos) gerenciados por um sistema remoto. Essa troca de representações de recursos entre cliente e servidor torna a utilização de uma *Resource API (*DAIGNEAU, 2011) com uma arquitetura *REST* (**RE**presentational **S**tate **T**ransfer) o modelo ideal para comunicação entre o cliente e servidor do protótipo.

Ao seguir este modelo é possível explorar o *HTTP* como um protocolo de aplicação completo, fornecendo métodos padronizados (*GET, POST, PUT e DELETE*) para manipulação de recursos, utilização de *URI* (*Universal Resource Identifier*) para localização dos mesmos, além de códigos padronizados para respostas (*HTTP Status Codes*).

Estas características favorecerão a interoperabilidade, abstração, escalabilidade e fraco acoplamento entre cliente e servidor. Permitindo que os objetivos de colaboração e ubiquidade de uma visualização deste trabalho sejam atingidos.

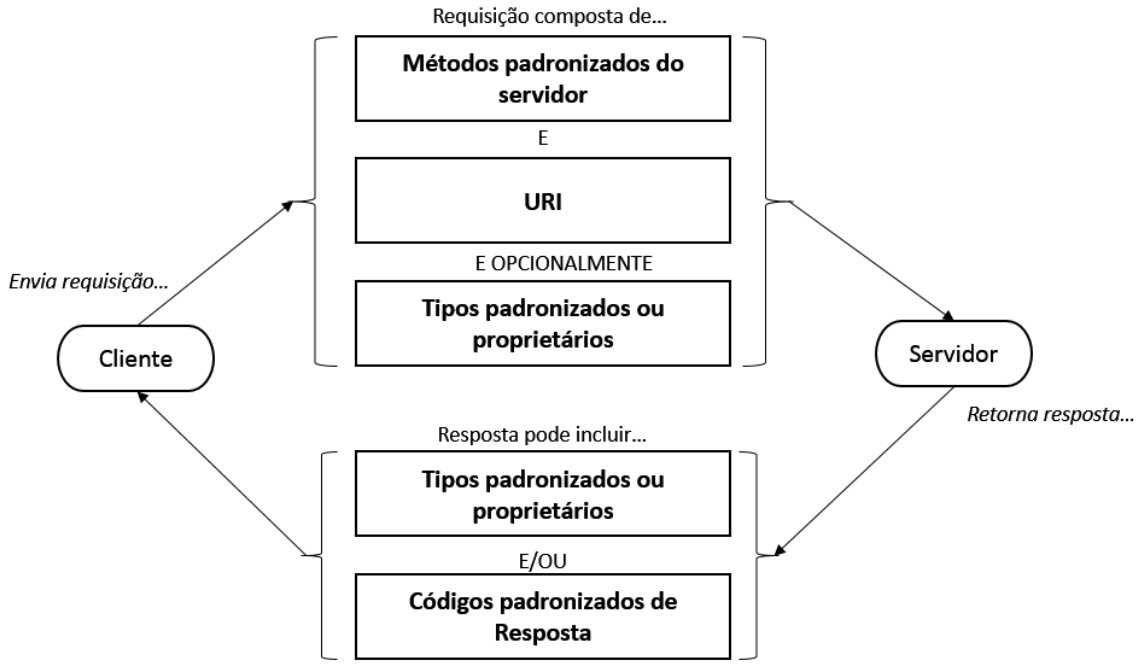

Figura 23: Representação do funcionamento de uma API baseada em recursos. ***(DAIGNEAU, 2011)***

O esquema apresentado na Figura 23 descreve a interação sobre o protocolo HTTP, entre o consumidor e o serviço. Nessa conversa a troca de mensagens é realizada utilizando um tipo de dados padronizado, o *XML* (*eXtensible Markup Language*), e as operações são representadas por

uma combinação de URI e método HTTP, além de códigos padronizados de resposta do protocolo.

A Tabela 2 descreve as operações disponíveis para interação com a API, onde a coluna Recurso irá representar o Verbo HTTP e Modelo de URI para comunicação com o servidor e a outra coluna, denominada descrição, fornecerá uma breve explicação sobre o seu funcionamento.

Tabela 2: Descrição dos recursos disponíveis na API.

Recurso	**Descrição**
GET /datasets	Retorna todos os *datasets* disponíveis.
GET /datasets/{name}	Retorna o *dataset* com o nome especificado.
POST /datasets/{name}	Carrega o *dataset* com o nome especificado.
PUT /datasets/{name}	Atualiza o *dataset* com o nome especificado.
DELETE /datasets/{name}	Exclui o *dataset* com o nome especificado.
GET /datasets/{name}/attributes	Retorna todos os atributos para o dataset com o nome especificado.
GET /datasets/{name}/filters	Retorna todos os filtros discretos e contínuos, habilitados e desabilitados para um dataset.
GET /datasets/{name}/filters/{attributeName}	Retorna a representação de um filtro para um determinado atributo de um *dataset*.
POST /datasets/{name}/filters/{attributeName}	Insere um filtro para um determinado atributo de um *dataset*.
PUT /datasets/{name}/filters/{attributeName}	Atualiza um filtro para um determinado atributo de um *dataset*.

DELETE /datasets/{name}/filters/{attributeName}	Excluir um filtro para um determinado atributo de um *dataset*.
GET /datasets/{datasetName}/visualizations/{name}	Obtém dados de uma visualização com o nome especificado para um determinado *dataset*.
POST /datasets/{datasetName}/visualizations/{name}	Cria uma visualização com o nome especificado para um determinado *dataset*.
PUT /datasets/{datasetName}/visualizations/{name}	Atualiza uma visualização com o nome especificado.
DELETE /datasets/{datasetName}/visualizations/{name}	Exclui uma visualização com o nome especificado.
GET /datasets/{datasetName}/visualizations/{name}/{techniqueName}?width=&height=	Retorna uma visualização de um determinado *dataset* utilizando uma técnica específica, com um tamanho x (altura) e y (largura) em pixels.
GET /datasets/{datasetName}/visualizations/{name}/techniques	Retorna uma lista de técnicas de visualização disponíveis para uma visualização de um conjunto de dados.
GET /datasets/{datasetName}/visualizations/{name}/techniques/{techniqueName}/{x}/{y}	Retorna detalhes de um determinado item sob um ponto x e y de uma visualização para determinada

	técnica.
GET /datasets/{datasetName}/visualizations/{name}/techniques/{techniqueName}/configurationsForAttribute	Retorna uma lista de configurações para atributos de uma determinada técnica.
PUT /datasets/{datasetName}/visualizations/{name}/techniques/{techniqueName}/configurationsForAttribute/{configurationName}/{attributeName}	Atualiza o atributo utilizado para uma configuração. *O nome da configuração deve ser um dos nomes disponíveis retornados pelo método GET.*
GET /datasets/{datasetName}/visualizations/{name}/techniques/{techniqueName}/configurationsForEnumeration	Retorna uma lista de configurações para valores pré-definidos de uma determinada técnica.
PUT /datasets/{datasetName}/visualizations/{name}/techniques/{techniqueName}/configurationsForEnumeration/{configurationName}/{attributeName}	Atualiza uma configuração com o nome especificado para um valor pré-definido. *O nome da configuração deve ser um dos nomes disponíveis retornados pelo método GET.*
GET /datasets/{datasetName}/visualizations/{name}/techniques/{techniqueName}/hierarchyConfigurations	Retorna uma lista de elementos adicionados na hierarquia, onde o primeiro representa o topo.
POST /datasets/{datasetName}/visualizations/{name}/techniques/{techniqueName}/hierarchyConfigurations	Adiciona um atributo no final da hierarquia.
DELETE /datasets/{datasetName}/visualizations/{name}/techniques/{techniqueName}/hierarchyConfigurations	Remove um atributo da hierarquia.

Além de URI's e verbos HTTP, este serviço também utilizará códigos padronizados de resposta HTTP (*HTTP Status Codes*), amplamente utilizados e conhecidos na Internet, portanto,

como em qualquer outra arquitetura REST, serão empregados na comunicação entre o cliente e servidor deste trabalho.

Em linhas gerais, o cliente precisa conhecer ao menos a classificação dos códigos de resposta descritos na Tabela 3, que os classifica pelo seu prefixo. Caso necessite entender um código específico, poderá consultar a RFC 7231 (FIELDING e RESCHKE, 2014), responsável pela padronização do protocolo HTTP 1.1.

Tabela 3: Significado dos códigos de status HTTP.

Prefixo	**Significado**
1xx	Informativo
2xx	Sucesso
3xx	Redirecionamento
4xx	Erro no cliente
5xx	Erro no servidor

(RICHARDSON, 2007) propôs uma classificação de serviços para a internet que vem sendo utilizada para medir a maturidade de um serviço, quantificando essa maturidade em valores de zero a três. Este trabalho tem como objetivo atingir o nível 2 (dois) de maturidade da classificação proposta, ou seja, servir vários recursos através de diferentes *URI's*, suportando diferentes verbos HTTP para manipulação de recursos através da rede, e por consequência tirar o maior proveito da robustez da internet para coordenar interações entre o cliente e o servidor.

A Figura 24 indica o principal elemento de cada nível, o nível 0 e nível 1 estão relacionadas com o uso de *URI's*, única no caso do nível inicial e múltiplas no caso do primeiro. Já o nível 2, envolve o uso de Verbos HTTP para coordenar as interações, enquanto o nível 3 utiliza controles hipermídia, em outras palavras, utiliza hipermídia como motor de estado da aplicação (HATEOAS).

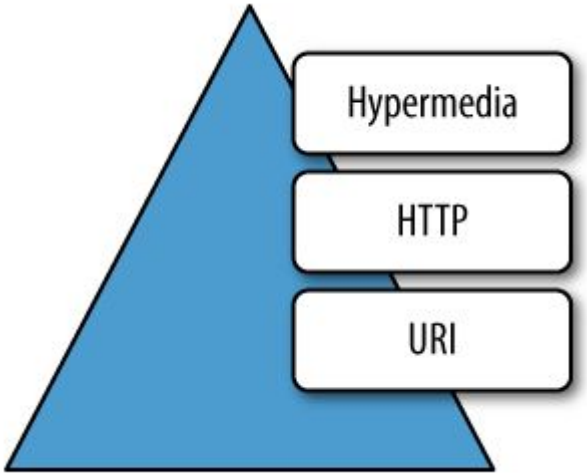

Figura 24: Modelo de maturidade de serviços REST. ***(WEBBER, PARASTATIDIS e ROBINSON, 2010)***

Segundo (WEBBER, PARASTATIDIS e ROBINSON, 2010), o modelo de Richardson auxilia na escolha do tipo de serviço HTTP que se quer construir a partir das características que cada nível nos proporciona. Como o último nível proporciona apenas a facilidade de descobrimento de operações e aumenta o poder de auto documentação do serviço, optou-se por não atingir, para construção desse serviço, este nível, permanecendo no segundo nível, onde podemos quebrar as responsabilidades e encapsular variações no tratamento das interações.

Dessa forma, o serviço apresentado define um conjunto de operações e coordena a delegação dessas tarefas ao *Núcleo PRISMA*, do carregamento dos dados brutos até a representação visual ao final do processo.

Pode-se dizer que a API abstraí os detalhes de implementação do Núcleo para o nível de granularidade da *pipeline de visualização*. Pipeline esse, expresso na Figura 25, que descreve o processo para criação de representações visuais dos dados, composto por quatro principais passos: (1) análise de dados, (2) filtragem, (3) mapeamento e (4) representação.

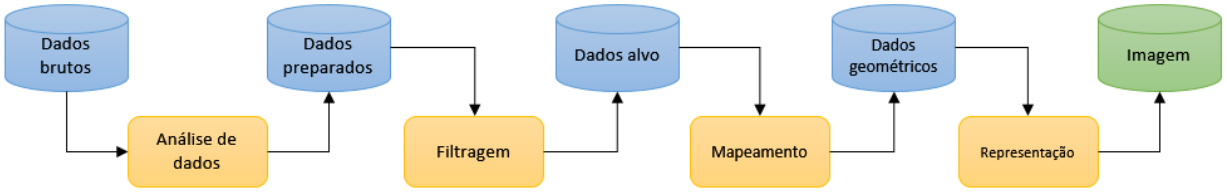

Figura 25: Pipeline de Visualização. ***(DOS SANTOS e BRODLIE, 2004)***

É possível mapear os passos do pipeline de visualização diretamente com as operações disponíveis na API apresentada, facilitando o entendimento da mesma pelos seus desenvolvedores e consumidores.

Tabela 4: Mapeamento entre operações da API e Pipeline de Visualização.

Recurso	**Passo na Pipeline de Visualização**
POST /datasets/{name}	Análise de dados
POST/PUT/DELETE /datasets/{name}/filters/{attributeName}	Filtragem
POST */datasets/{datasetName}/visualizations/{name}*	Mapeamento
GET /datasets/{datasetName}/visualizations/{name}/techniques/{techniqueName}	Representação

4.3.1 Arquitetura do Serviço

A arquitetura do serviço baseia-se no padrão ECB (*Entity-Control-Boundary*) (BUSCHMANN , MEUNIER, *et al.*, 1996), uma variação do padrão arquitetural MVC, onde as entidades (*entities*) são uma representação do modelo conceitual, os controladores (*controllers*) manipulam as entidades para realização de operações e as fronteiras (*boundaries*) representam a camada mais externa acessível pelos consumidores do serviço (Figura 26).

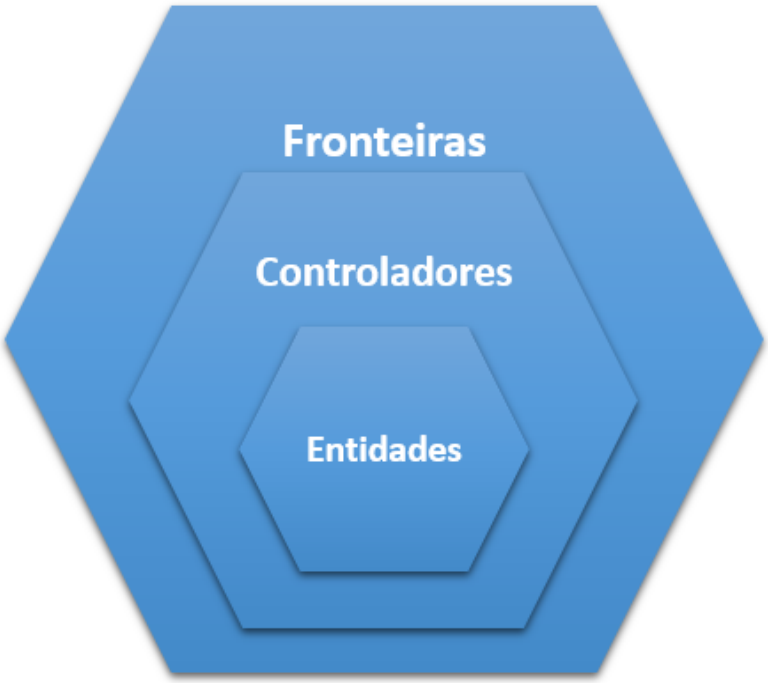

Figura 26: Padrão ECB.

(COCKBURN, 2009) construiu uma abstração em cima do padrão ECB, e consequentemente MVC, e denominou-a de Arquitetura Hexagonal. Esta arquitetura tem como principal propósito construir aplicações para trabalhar sem uma interface do usuário (UI) e/ou sem uma base de dados, permitindo dessa forma, o desacoplamento entre a lógica de negócio e a interação com as interfaces externas. Por este motivo, essa arquitetura também é chamada de "portas e adaptadores" (*ports and adapters*), pois permite que se utilizem adaptadores para converter as interfaces atuais dos *controladores* em novas portas da aplicação (Figura 27).

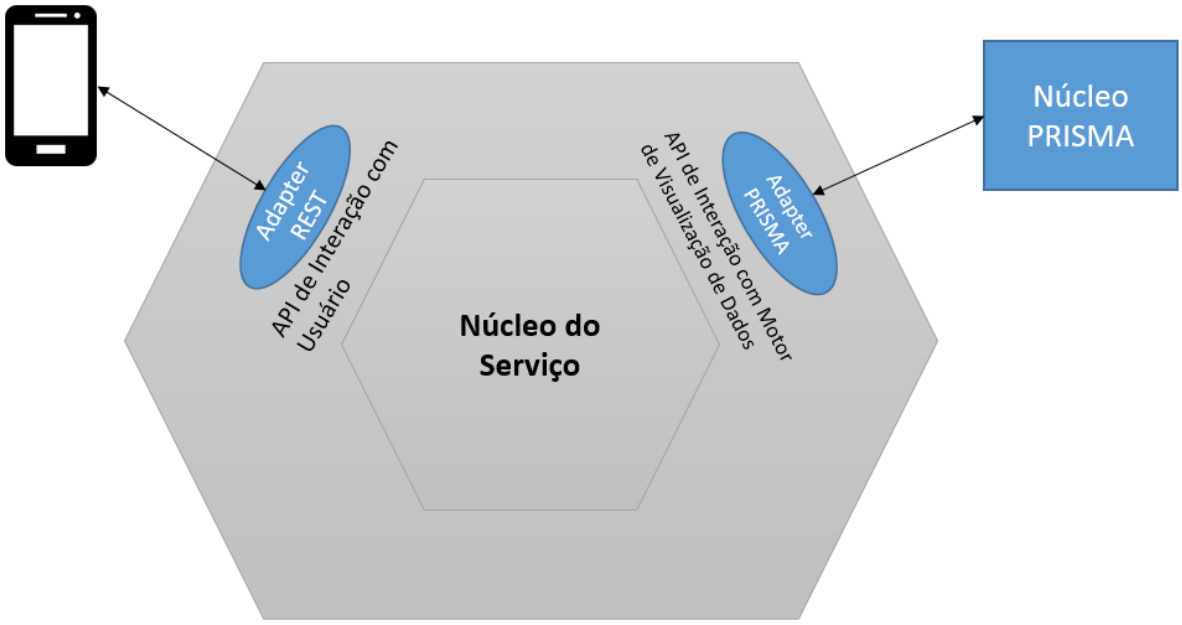

Figura 27: Exemplificação de Arquitetura Hexagonal no contexto do serviço desenvolvido

Dentre as principais vantagens dessa arquitetura podemos destacar:

- Não faz distinção entre interfaces de usuário e interfaces para outros tipos de atores, como servidores e dispositivos. Permitindo a construção de uma API agnóstica de uma tecnologia específica de interface do usuário ou persistência de dados.
- Um objeto de fronteira pode ser visto como qualquer objeto que implementa a interface de um *controlador*, e essa interface, por sua vez, pode ser vista como uma "porta" na fronteira do hexágono intermediário. Facilitando a construção de testes, visto que podemos facilmente implementar objetos de fronteira, reais ou simulados. A grande diferença em relação a outros padrões arquiteturais é a preocupação com portas secundárias (*backend*), além de portas primárias (*frontend*).

As vantagens descritas na arquitetura hexagonal também se aplicam ao padrão ECB, desde que haja separação entre a lógica de negócios e interfaces externas.

Dessa forma, ao utilizar esse padrão arquitetural no serviço, foi possível extrair uma API agnóstica de tecnologia (camada de controladores) que qualquer provedor de visualização de dados poderá implementar para expor um serviço de visualização de dados utilizando REST (camada de fronteira). Ou seja, neste estilo arquitetural, temos um adaptador (*adapter)* dos controladores para uma porta (*port*) na tecnologia REST, assim como também temos um adaptador da camada de

controladores para o Núcleo PRISMA.

Novos adaptadores podem ser inseridos nessa arquitetura para suportar outros provedores de visualização de dados ou outras tecnologias para interação com o núcleo do serviço.

4.3.2 Projeto do Serviço

O projeto do serviço é composto pelas classes integrantes das camadas de entidades, controladores e fronteiras.

No diagrama de classes do pacote de entidades (Entidades da arquitetura *ECB*) (Figura 28) pode-se perceber que esse modelo assemelha-se bastante com o modelo conceitual, porém com detalhes da implementação das duas estratégias de filtro, discreto (*DiscreteFilter*) e contínuo (*ContinuousFilter*), onde o primeiro possui um mapa de valores de um atributo que devem ou não ser filtrados de um conjunto de dados e o segundo define uma faixa de valor (mínimo e máximo) para um atributo, excluindo aqueles que não fazem parte do grupo. Estereótipos de entidade (<*entity*>) foram omitidos do diagrama para maior legibilidade, visto que todas as classes do pacote possuem o mesmo estereótipo.

Outra representação importante ilustrada no diagrama (Figura 28) é o funcionamento das configurações, agrupadas em *configurações para um atributo (ConfigurationForAttribute)*, como cor e rótulo, *configurações para um valor pré-definido (ConfigurationForEnumeration)* como tamanho e formato e *configuração de hierarquia (HierarchyConfiguration)* que possui uma lista de elementos ordenados onde o valor inicial indica o topo da hierarquia e os valores subsequentes representam valores abaixo desse na organização hierárquica.

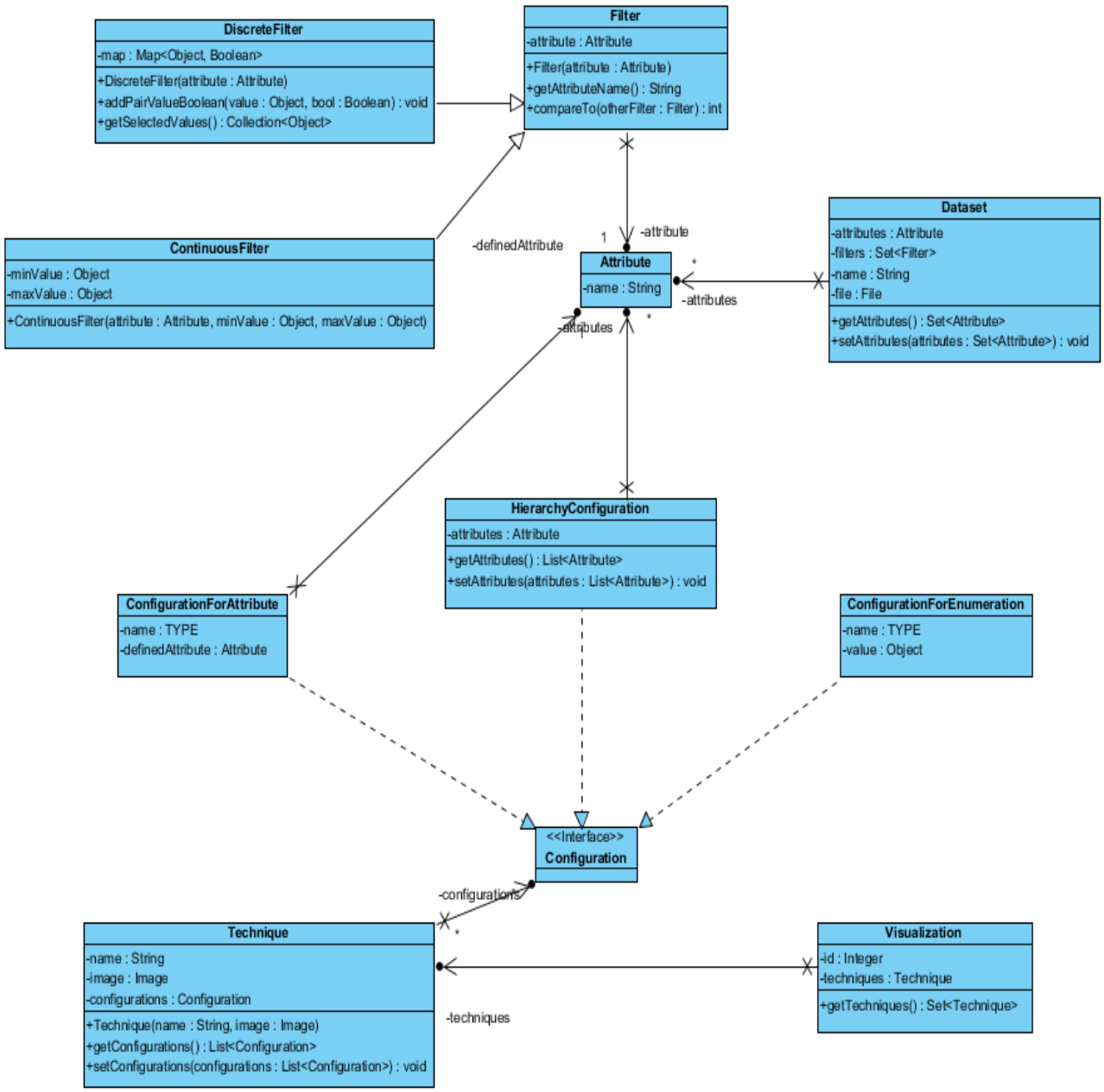

Figura 28: Diagrama de classes do pacote de entidades.

A camada de entidades fornecerá o primeiro nível de abstração, visto que as interfaces dos controladores serão definidas em termos dessas *entidades*. Logo, as interfaces da camada de controladores podem ser vistas, neste cenário, como o alvo da conversão entre o modelo conceitual e um provedor de visualizações de informações, realizada pelos *adapters (*GAMMA, HELM, *et al.*, 1994), permitindo dessa forma que o provedor de visualizações seja facilmente substituído com o custo apenas da implementação de *adapters* que satisfaçam o contratto da camada de *controladores*.

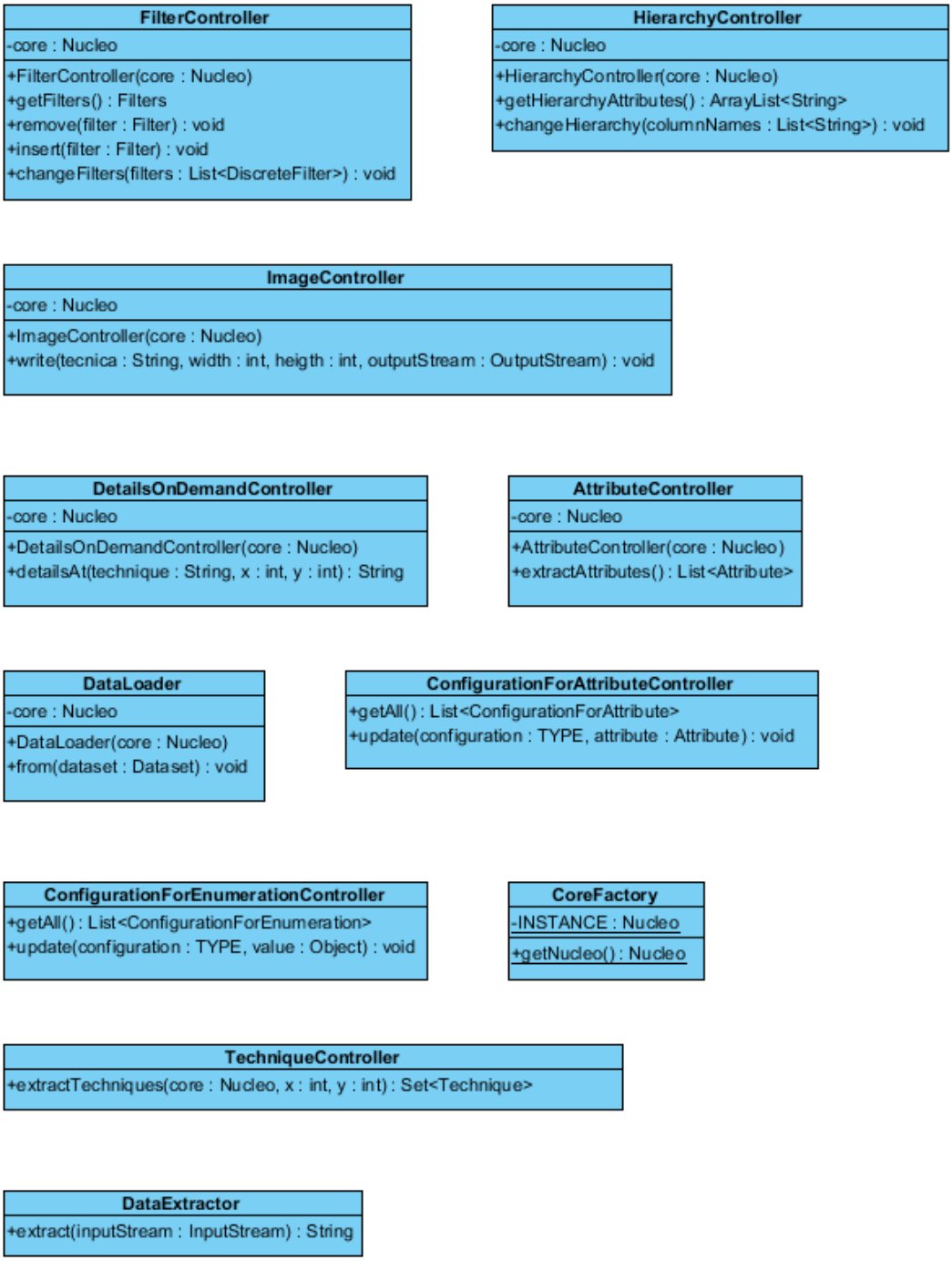

Figura 29: Diagrama de classes do pacote de controladores.

O diagrama de classes do pacote de controladores (Controles da arquitetura *ECB*) (Figura 29) ilustra o baixo acoplamento entre elementos da mesma camada e a definição dos seus contratos em termos da camada de entidade, tornando estes os principais responsáveis pela manipulação das mesmas, mediando a comunicação entre fronteiras e entidades de forma a orquestrar a execução dos comandos oriundos da primeira. O estereótipo UML para controladores (<*controller*>) também foi omitido do diagrama para simplificação, já que todos os objetos do pacote possuem este estereótipo.

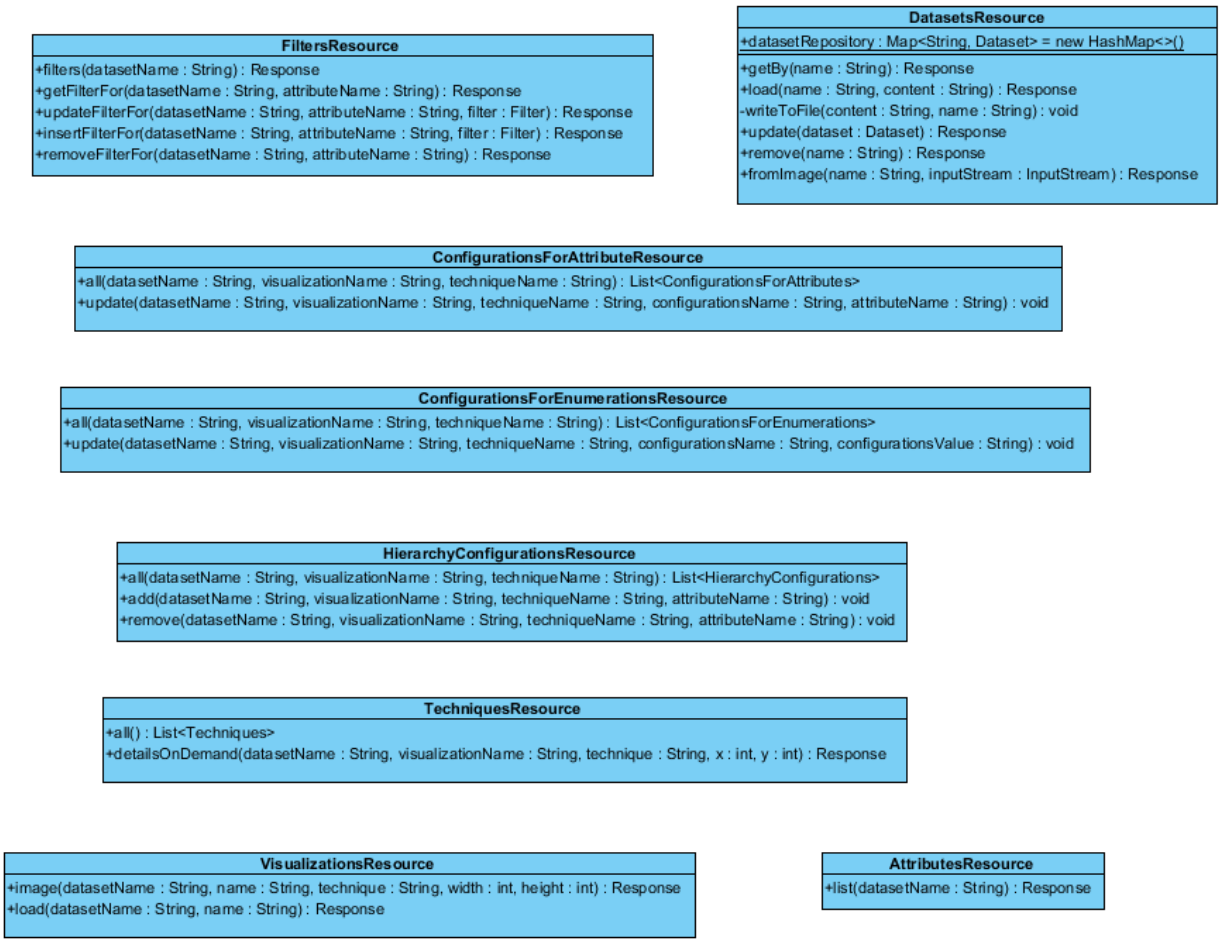

Figura 30: Diagrama de classes do pacote de fronteiras.

Por último, tem-se a fronteira da aplicação, responsável por disponibilizar um conjunto de operações e coordenar a resposta da aplicação para cada operação. Logo, todos os objetos desse pacote poderiam receber o estereótipo UML para fronteira (<*boundary*>), pois são objetos que interagem diretamente com os atores do sistema. Além de definir as operações, a fronteira do sistema também irá definir qual tecnologia (porta) ia será utilizada para integração entre os clientes e o serviço, neste caso, serviços *RESTFul*.

Apenas um pacote do serviço foi omitido dos diagramas, devido a sua simplicidade, pois este agrupa apenas as classes para fim de agregação de entidades para efeitos de facilitação das operações de *marshall* e *unmarshall* das coleções de objetos e consequentemente permitir a troca de mensagens complexas entre o cliente e o servidor.

4.3.3 API de Interação com Provedor de Visualização de Dados

A API de interação com usuário foi descrita em detalhes durante a apresentação da API REST e da camada de fronteiras do serviço. Já a API a interação com o provedor de visualização de dados pode ser compreendida a partir da camada de controles do serviço.

Se extrairmos as interfaces públicas da implementação atual dessa camada, ficaremos com uma API agnóstica de tecnologia e definida em termos do modelo conceitual, possibilitando que qualquer provedor de visualização de dados possa implementar *adapters* entre a sua representação do pipeline de visualização de dados e a representação proposta neste trabalho. Facilitando a exposição do seu núcleo de visualização de dados como um serviço REST ou qualquer nova porta (*port*) que se deseja introduzir.

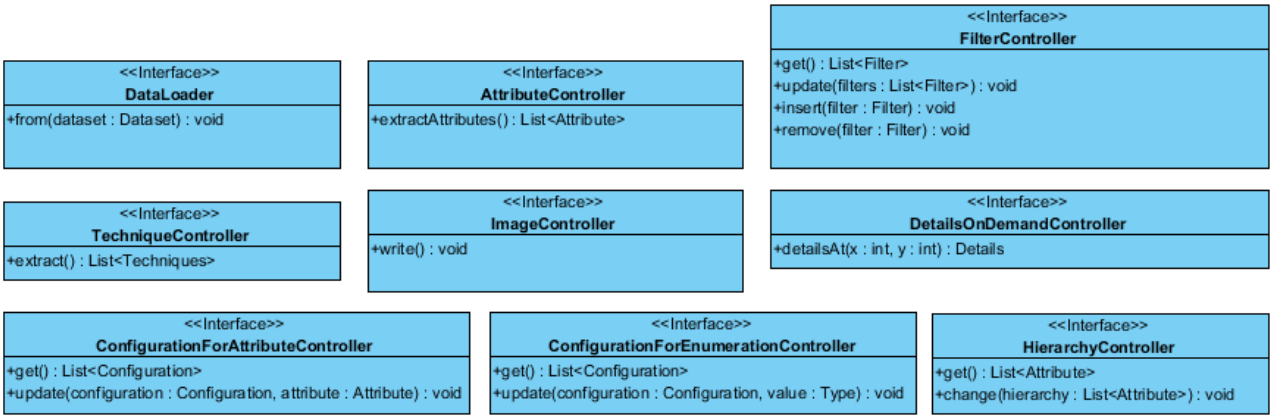

Figura 31: Interfaces da API de Interação com Provedor de Visualização de Dados

O diagrama da figura 31 representa as interfaces extraídas da implementação atual. Suas responsabilidades podem ser melhores descritas de acordo com a seguinte estrutura:

- *DataLoader*: Ponto de entrada de dados para o motor de visualização.
- *AttributeController:* Extrai os atributos de um conjunto de dados.
- *FilterController:* Insere, atualiza, exclui e obtêm filtros de uma visualização.
- *TechniqueController*: Extrai técnicas disponíveis de uma visualização.
- *ImageController:* Escreve a representação visual de uma técnica em uma saída.
- *DetailsOnDemandController:* Obtêm detalhes sob demanda de um ponto (x,y) específico.
- *HierarchyController*: Manipula a configuração de hierarquia.
- *ConfigurationForAttributeController*: Manipula configurações para atributos.

- *ConfigurationForEnumerationController*: Manipula configurações para valores pré-definidos.

5 APLICAÇÕES CLIENTE E SERVIDOR

Serão apresentados aspectos de modelagem da aplicação cliente e servidor, bem como funcionalidade da aplicação cliente, e as tecnologias utilizadas no desenvolvimento das mesmas.

5.1 Tecnologias Utilizadas

Para o desenvolvimento do serviço optou-se por tecnologias de código aberto e que pudessem facilmente interoperar com a base de código do PRISMA, desenvolvido em Java. As tecnologias empregadas estão listadas a seguir:

- JDK (Java Development Kit) versão 7, um ambiente de desenvolvimento para construção de aplicações utilizando a linguagem de programação Java.
- Framework *Grizzly* utilizado para construção de um servidor com suporte ao protocolo HTTP e alto nível de abstração.
- *Jersey*, a implementação de referência da especificação JAX-RS (*Java API for RESTFul Web Services*).

5.2 Cliente para Dispositivos Móveis

O cliente desempenha um papel finalístico no processo de geração de visualizações, ou seja, é responsável por fornecer uma entrada de dados, apresentá-los visualmente e interagir com suas representações.

Pode-se então, classificar este cliente como um cliente leve (*thin cliente)*, dependente do seu servidor para executar todo o processamento necessário, sendo responsável apenas por delegar chamadas a este.

Clientes para API's REST podem ser desenvolvidos para qualquer linguagem ou plataforma que suporte comunicação via protocolo HTTP, ou seja, é possível ter clientes para as principais

plataformas de dispositivos móveis como Android, iOS, Windows Phone, ou mesmo clientes ricos para Web utilizando JavaScript, Microsoft Silverlight ou Adobe Flex.

Devido a facilidade de desenvolvimento e popularidade da plataforma, optou-se por desenvolver uma aplicação *Android* (Figura 32) para servir o propósito de um cliente para dispositivos móveis do serviço.

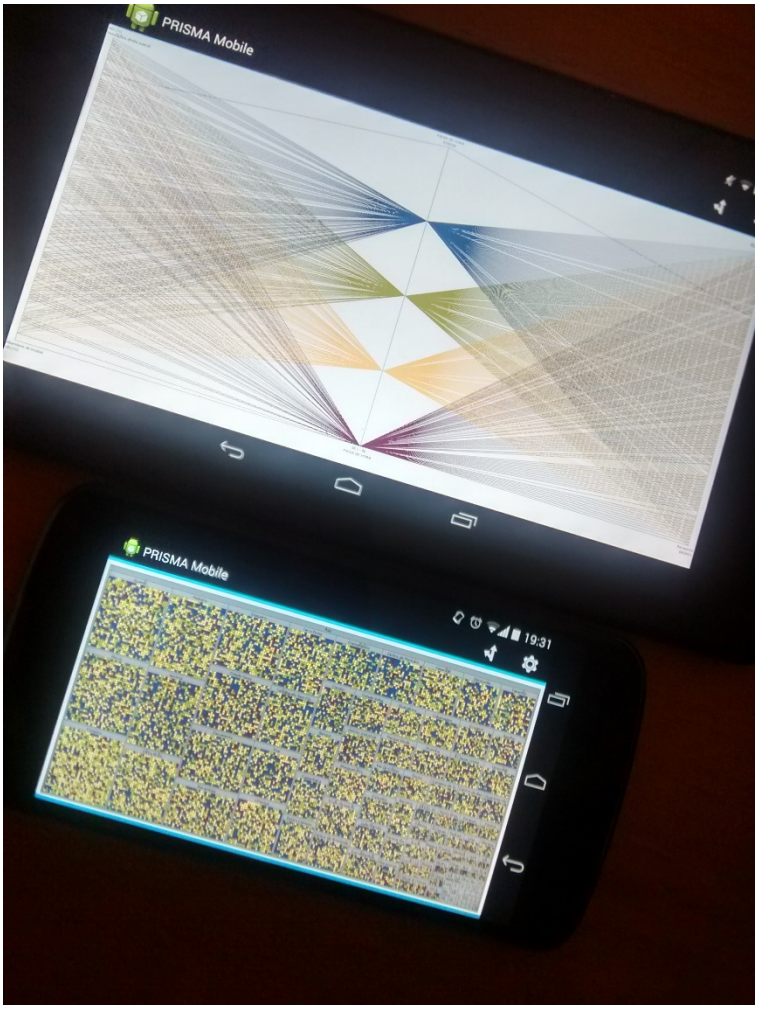

Figura 32: Dois clientes na plataforma Android trabalhando na mesma visualização.

5.3 Funcionalidades

5.3.1 Visão Geral

A Figura 33 demonstra uma visão inicial da aplicação, após o carregamento de um conjunto de dados e criação da visualização para o mesmo. Na visualização é possível perceber que não há controles na tela, maximizando o espaço disponível para a representação visual, permitindo que as interações sejam realizadas a partir dos ícones no canto superior direito ou no menu deslizável da lateral esquerda.

A partir dos controles da direita é possível filtrar o conjunto de dados representado ou

aplicar configurações específicas de uma técnica, enquanto a partir do menu deslizável da esquerda (Figura 34), pode-se escolher qual técnica utilizar dentre as disponíveis.

As técnicas disponíveis são obtidas do servidor a partir de critérios simples como o tipo dos dados carregados. Da mesma forma as configurações são carregadas de acordo com a visualização selecionada.

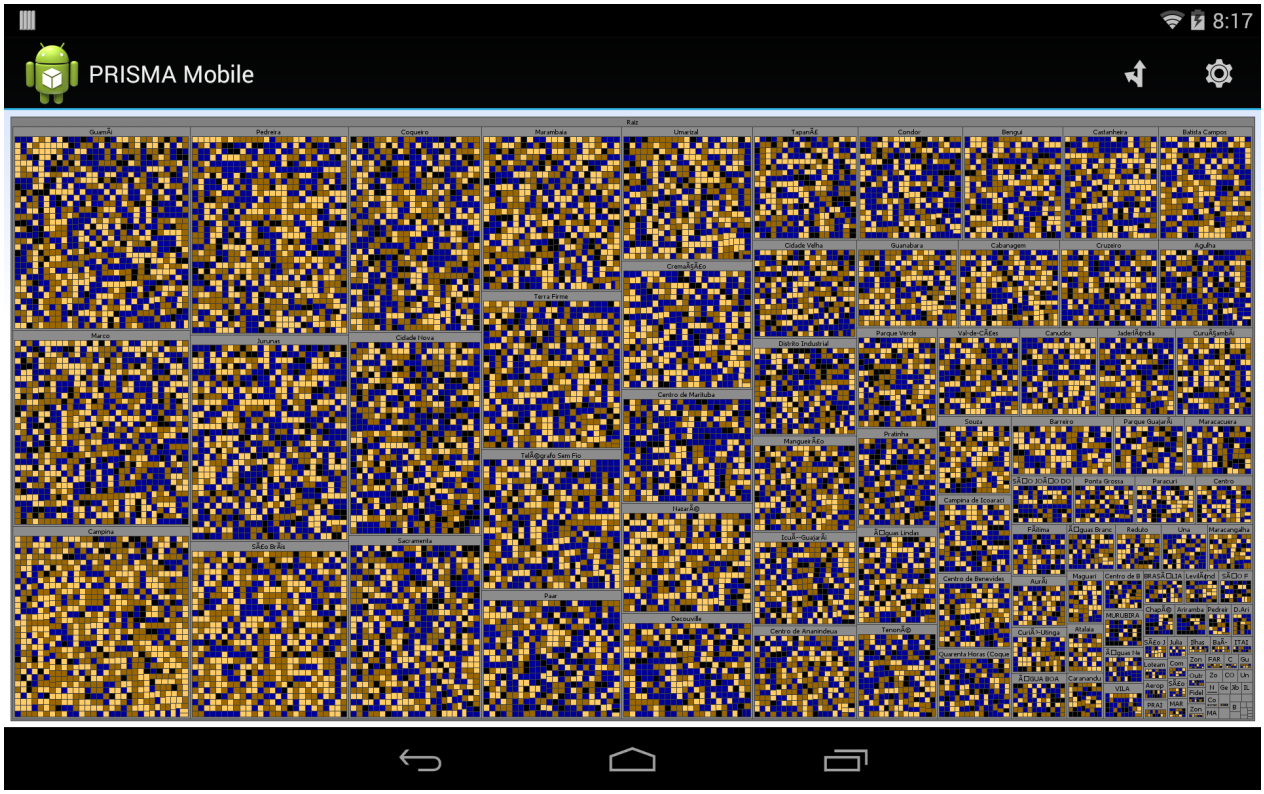

Figura 33: Visão de interação com uma visualização.

Figura 34: Menu deslizável lateral para seleção de técnica.

5.3.2 Zoom

Dispositivos móveis top de linha são dotados de telas de alta resolução, porém com tamanho físico reduzido se comparado com computadores pessoais e laptops, com isso os itens na tela tem o seu tamanho real inferior nesses dispositivos, além de possuir dificuldades de selecionar itens da visualização, geralmente por toque, que é menos preciso que o clique de um mouse. No modelo conceitual foi delegada a responsabilidade do zoom ao cliente, pois essa necessidade é dependente da plataforma utilizada, no caso de dispositivos móveis o zoom é essencial para visualização e seleção de itens devido a densidade de pontos por polegada (DPI) na tela desses dispositivos (Figura 35).

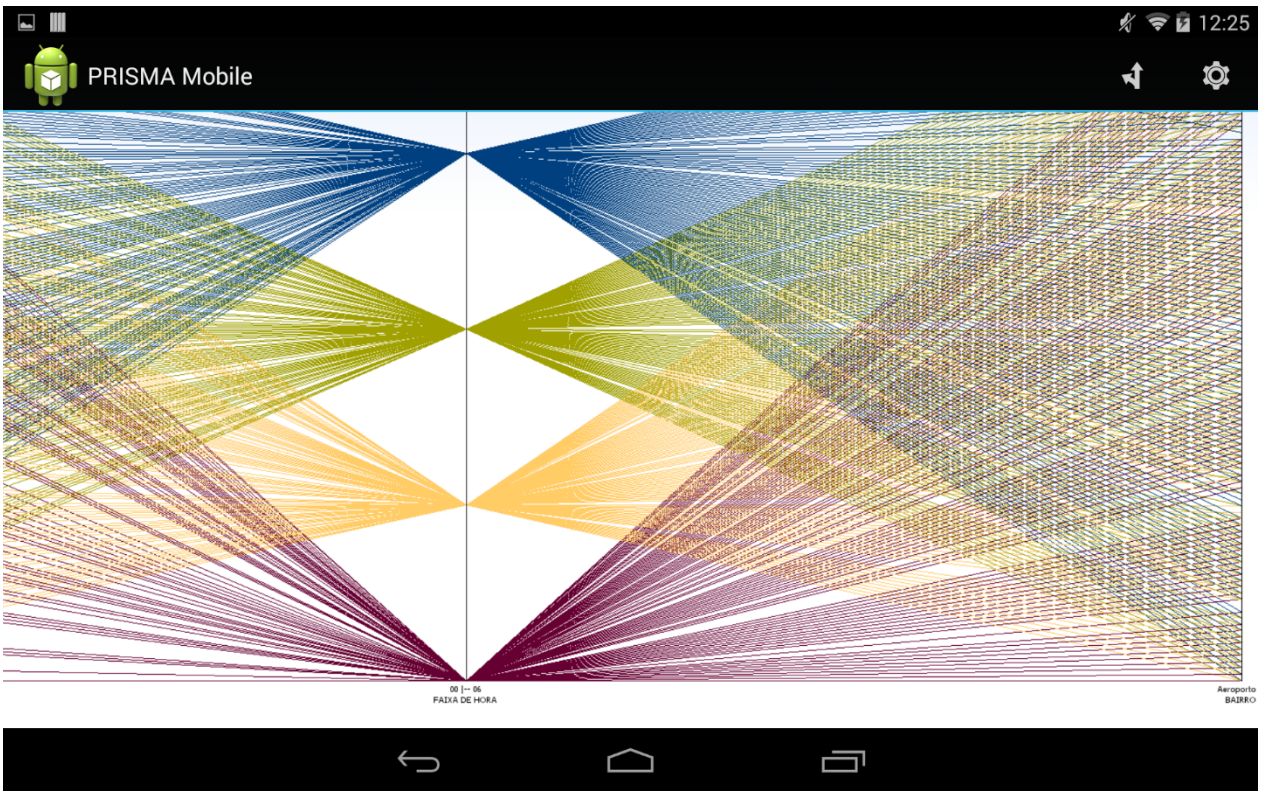

Figura 35: Coordenadas paralelas com Zoom.

A operação de zoom pode ser realizada utilizando o gesto de pinça, amplamente conhecido e adotado em dispositivos com telas sensíveis ao toque e *touchpads*. Esse gesto consiste em unir o polegar o indicador sobre um ponto e afastá-los para realizar a operação de zoom-in e aproximá-los para a operação de zoom-out. Exemplo de zoom aplicada a técnica treemap (Figura 36).

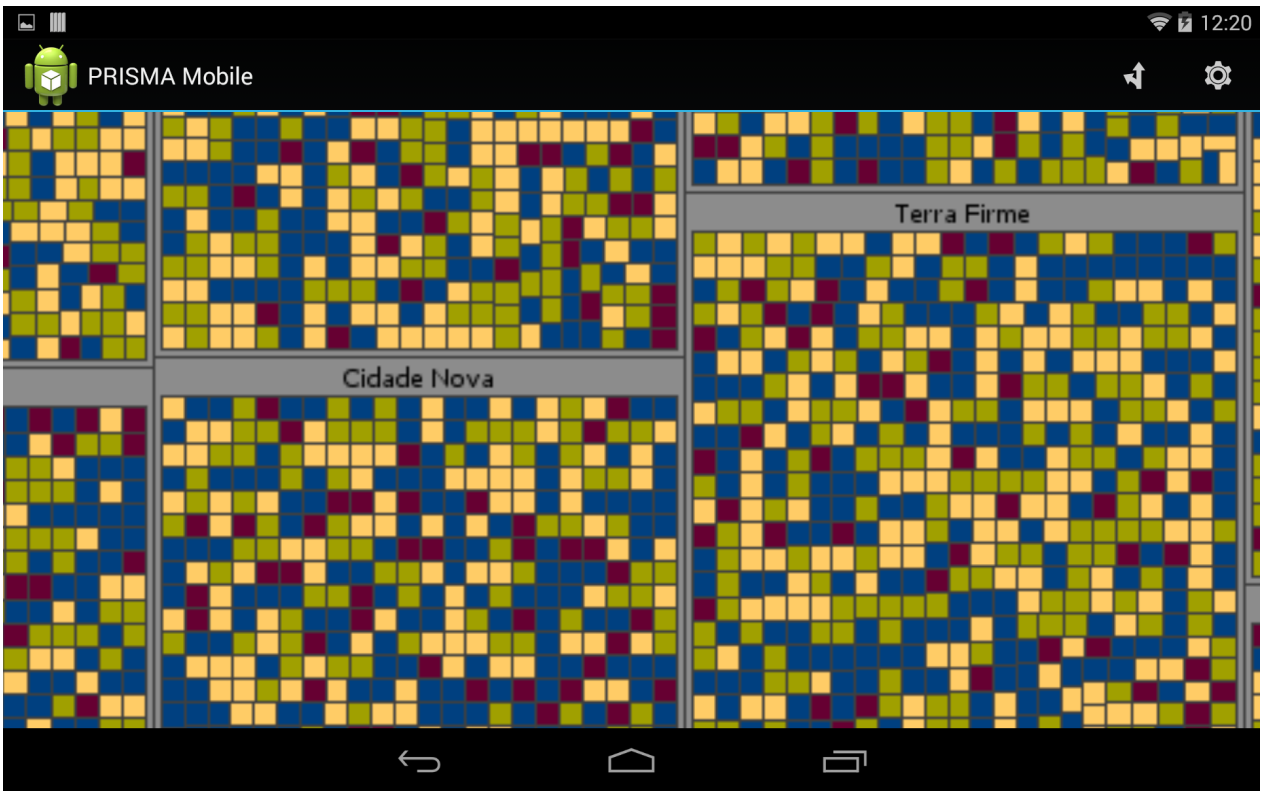

Figura 36: Zoom em funcionamento na técnica Treemap.

5.3.3 Detalhes sob Demanda

É possível obter informações a mais de um item visual ou de um grupo de itens visuais sob demanda. A ideia principal é identificar um item ou grupo de interesse através da visualização de dados, e obter mais dados sobre aquele item ou grupo de interesse (Figura 37).

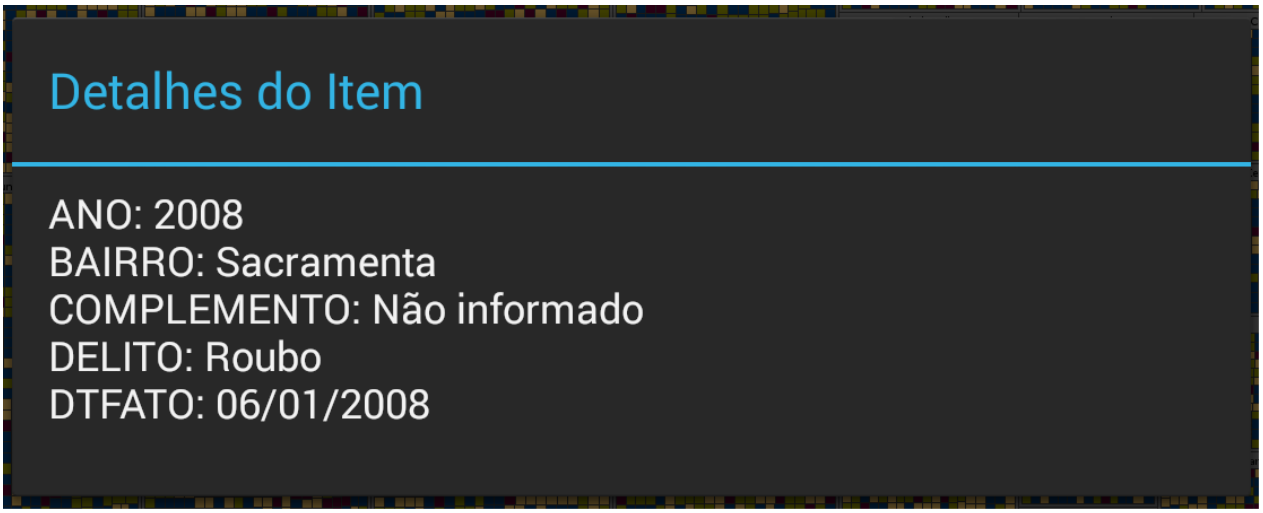

Figura 37: Detalhes sob demanda para um item selecionado.

5.3.4 Filtros

A aplicação apresenta duas modalidades de filtros, uma para atributos discretos e outra para

atributos contínuos, ambos acessíveis a partir da barra de ações no canto superior direito.

Os filtros discretos (Figura 38) permitem selecionar, dentre um conjunto de valores, quais destes farão parte do *dataset* utilizado para representação visual. Dessa forma, valores marcados de um atributo estão presentes naquela representação visual, enquanto valores não marcados estão de fora.

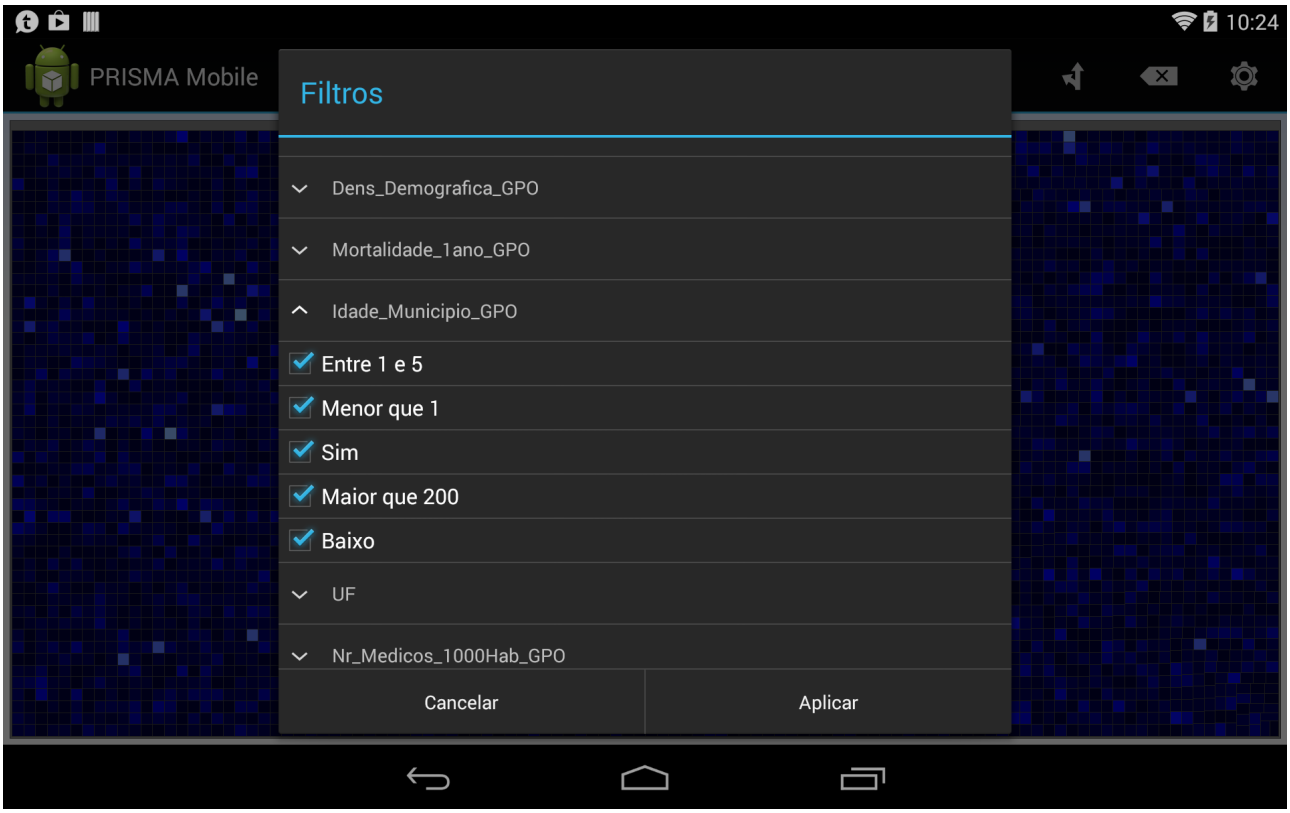

Figura 38: Controle para manipulação de filtros discretos.

Já para os atributos contínuos (Figura 39), tem-se um controle que permite definir o limite superior e inferior dos valores que estão contidos na visualização, podendo, dessa forma, excluir valores ao aumentar o limite inferior ou reduzir o limite superior.

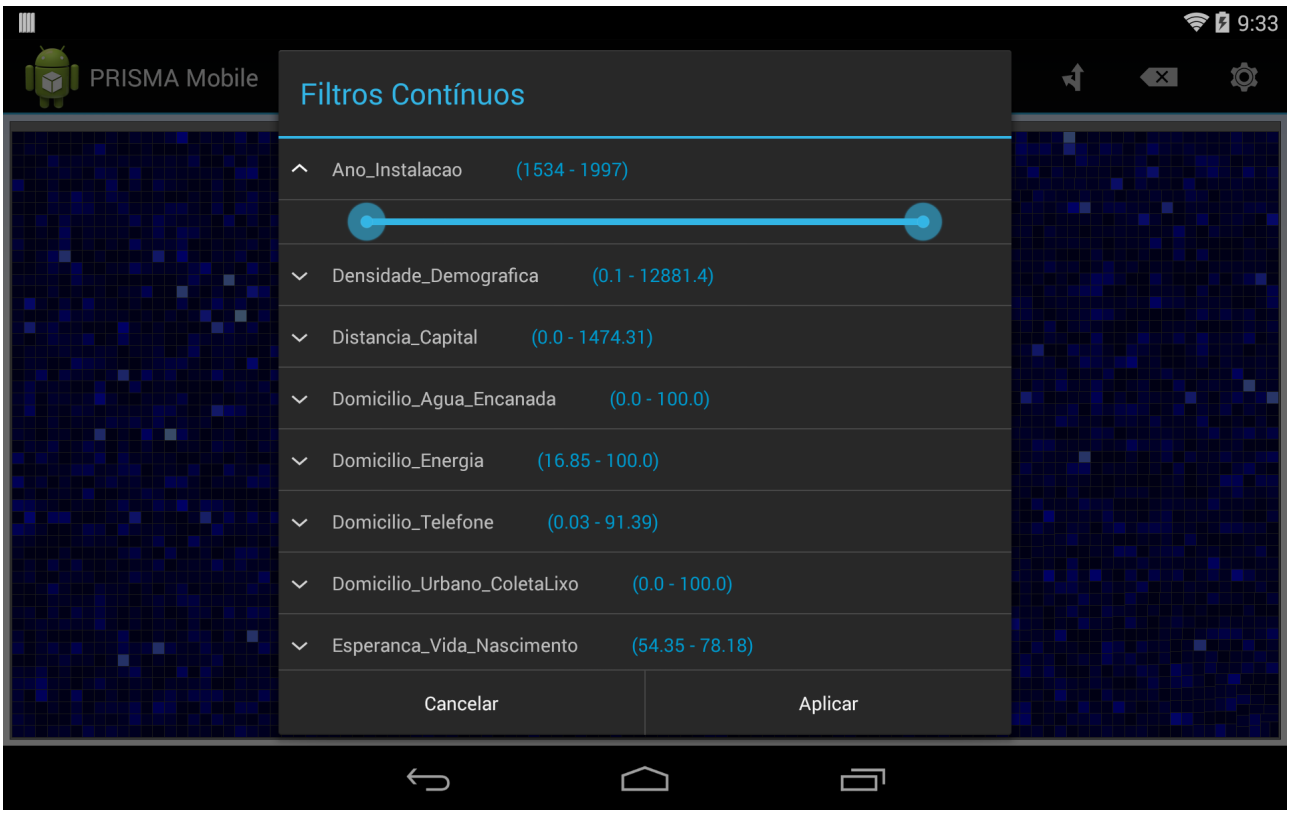

Figura 39: Controle para manipulação de filtros contínuos.

As manipulações nos filtros são aplicadas somente quando o botão aplicar é selecionado, permitindo que várias mudanças sejam aplicadas de uma única vez.

5.3.5 Configuração de técnicas

As configurações de técnicas, também acessíveis a partir da barra de ações do canto superior direito, estão divididas em quatro abas: (1) geral, onde se tem as configurações que podem ser aplicadas a mais de uma visualização com coordenação (Figura 40), (2) *treemap*, onde é possível configurar a hierarquia que será utilizada nessa técnica (Figura 41), (3) coordenadas paralelas, permitindo a adição ou remoção de eixos, (4) dispersão de dados, com opção de configurar quais atributos serão utilizados para os eixos X e Y (Figura 42).

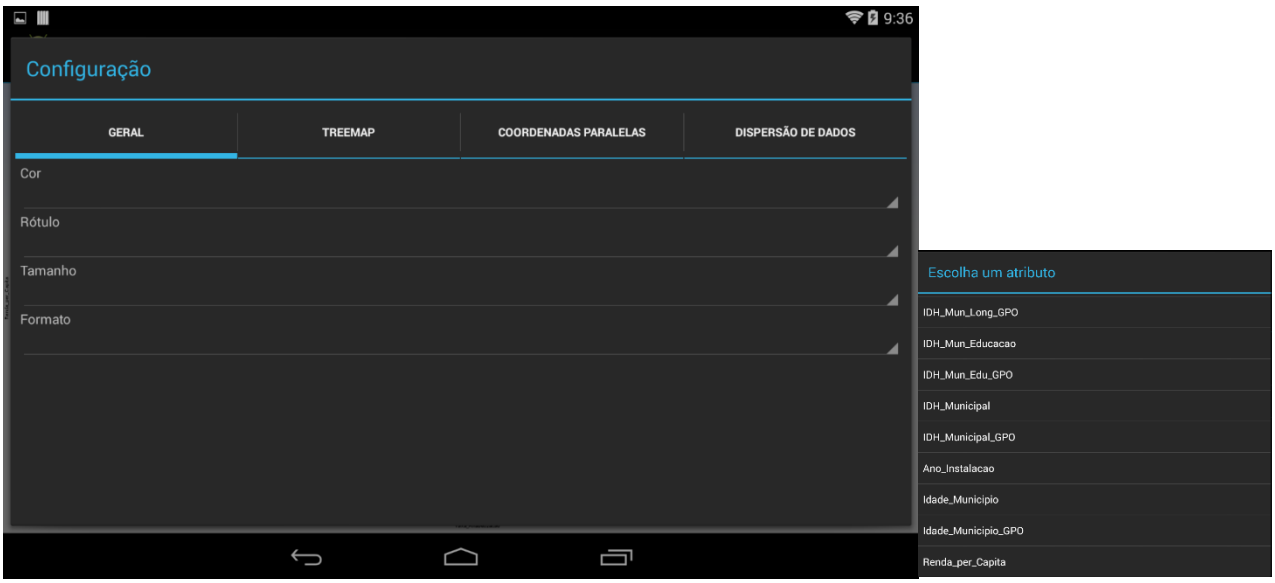

Figura 40: Configuração geral com seleção do atributo para cor, rótulo e tamanho.

Na guia de configuração geral (Figura 40) podemos configurar qual atributo será utilizado para cor, rótulo e tamanho. Ao selecionar uma configuração, uma caixa de diálogo será aberta exibindo quais atributos estão disponíveis.

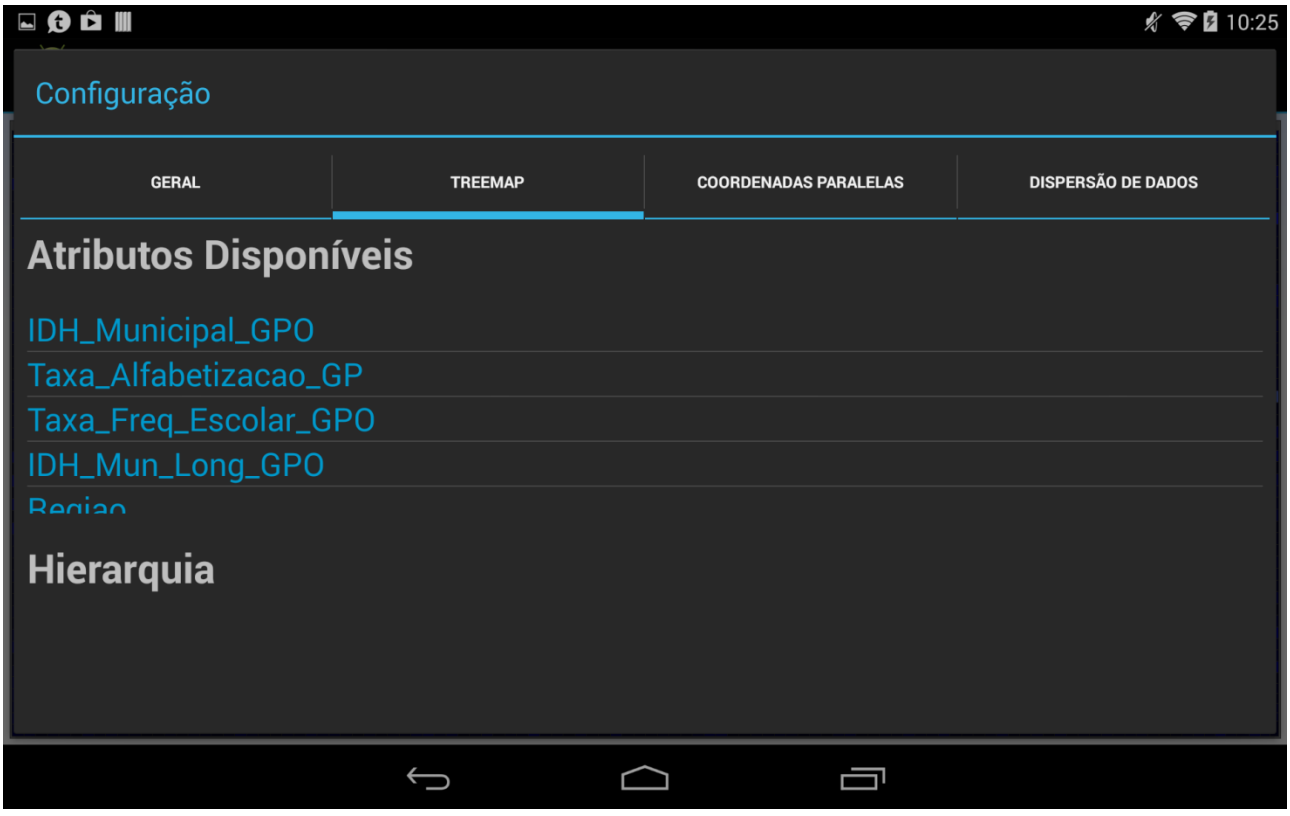

Figura 41: Configuração da hierarquia para o treemap.

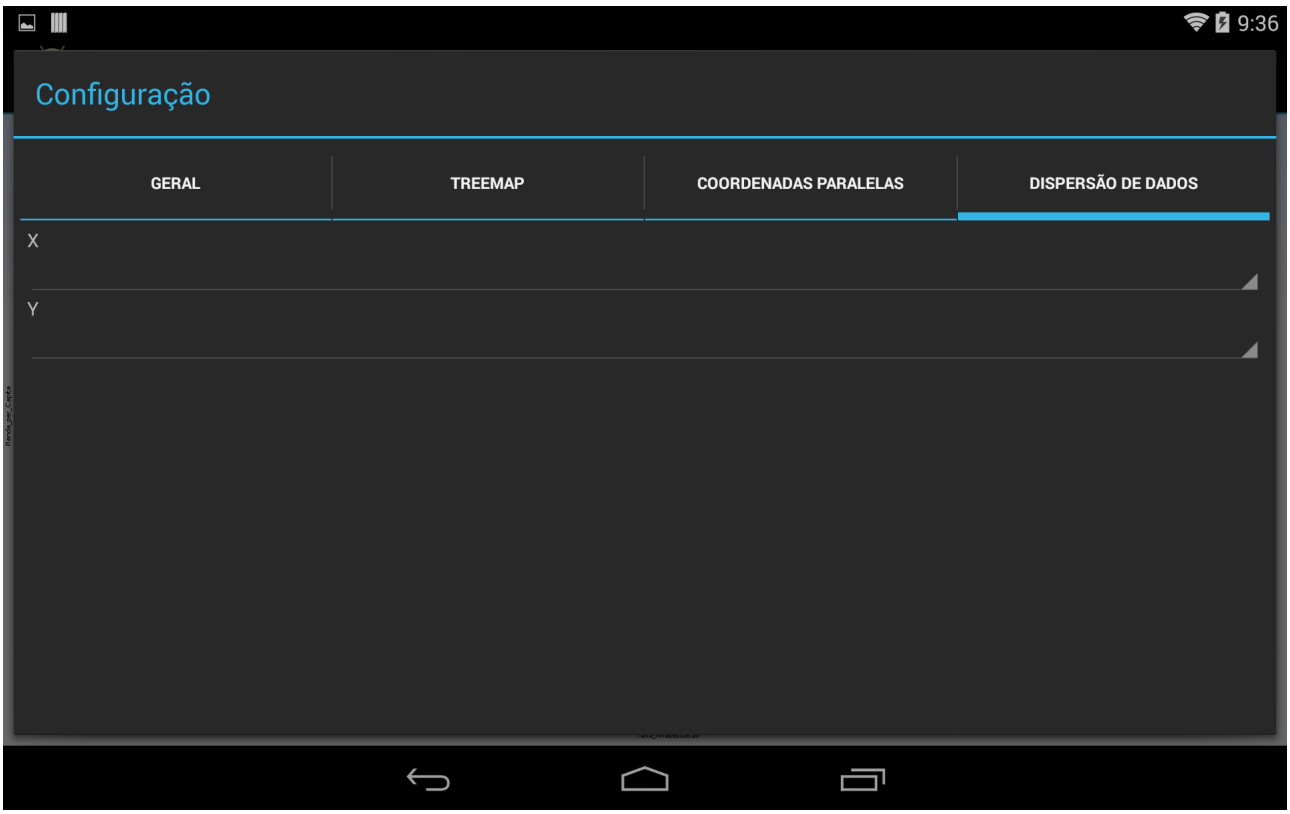

Figura 42: Configuração dos eixos X e Y para dispersão de dados.

5.4 Arquitetura do Cliente

A arquitetura do cliente é baseada no padrão arquitetural MVC (Model-View-Controller) em vez de ECB (Entity-Controller-Boundary), pois esse cliente leve é apenas um ator que interage com um serviço e não possui lógica de negócio, apenas lógica de interface do usuário. Padrões de interação entre cliente-servidor, como o Request/Response (Requisição/Resposta) (DAIGNEAU, 2011) e Asynchronous Response Handler (Tratamento Assíncrono de Resposta) (DAIGNEAU, 2011) também foram utilizados na construção desse cliente.

Sendo responsável por consumir uma API Web exposta pelo servidor, recebendo comandos da interface e notificando-a de eventos que possam causar alterações na mesma. Esse modelo de comunicação entre interface do usuário (UI) e os controladores baseiam-se no padrão Observer (GAMMA, HELM, *et al.*, 1994), desempenhando um papel chave para que as visões (views) sejam notificadas pelos controladores (controllers).

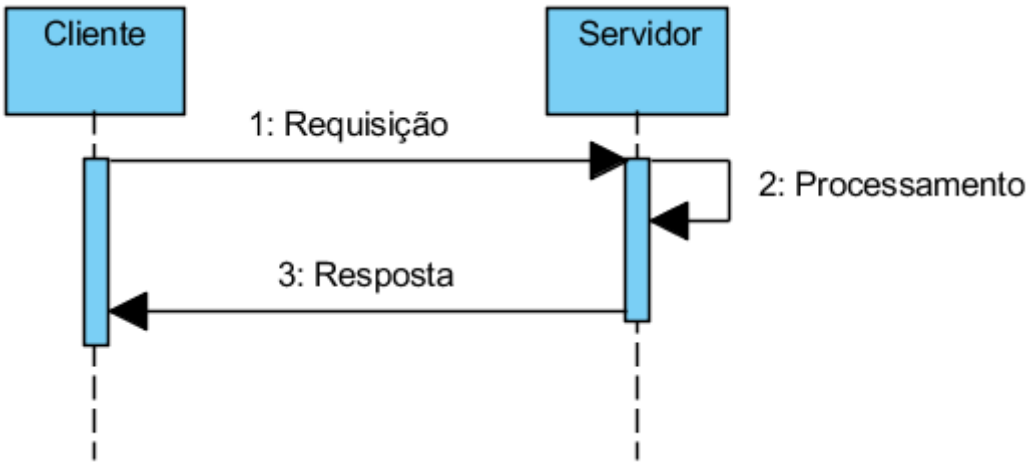

Figura 43: Interação cliente servidor no modelo Requisição/Resposta.

A maioria das interações entre o cliente e o serviço seguem o modelo requisição/resposta ilustrado na Figura 43, ou seja, a requisição e a resposta são feitas em uma mesma conexão, causando bloqueio no cliente, um modelo síncrono de comunicação. Esse modelo é imediato, pois o cliente precisa de respostas imediatas as suas requisições de interação na interface (i.e. filtros, configurações, seleção de técnica e etc.).

Para evitar que o bloqueio se reflita na interface do usuário e impeça que esse possa interagir com a aplicação, requisições sempre são disparadas ao servidor de forma assíncrona, em *backround*.

A única exceção a essa regra é o carregamento de um conjunto de dados que utiliza o tratamento assíncrono de resposta, devido ao alto custo de processamento da operação.

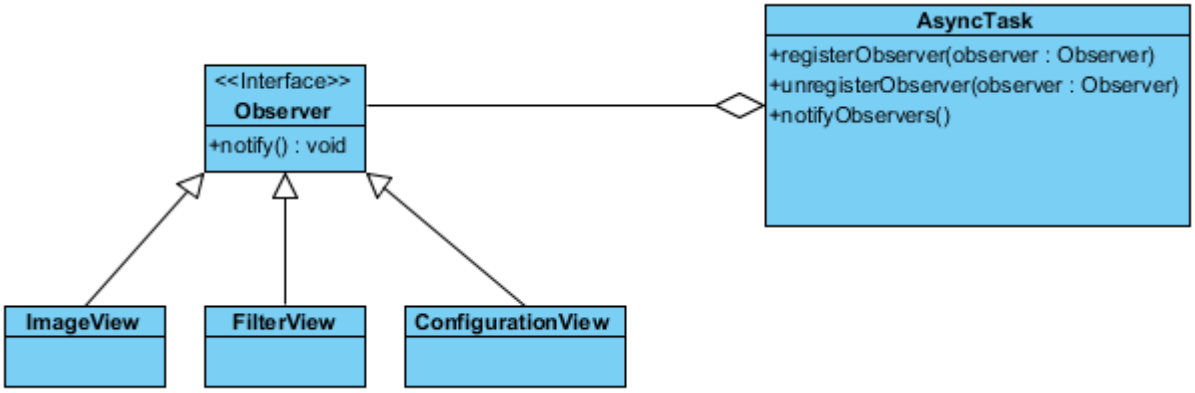

Figura 44: Padrão Observer aplicado na arquitetura do cliente.

O funcionamento da interface pode ser ilustrado pelo diagrama de classes da Figura 44, onde todas as visões (views) implementam a interface *Observer* e são notificadas quando alguma tarefa assíncrona (*AsyncTask*) é executada. Podemos citar como exemplo a tarefa de carregamento de uma

técnica, que consulta o servidor para obter uma imagem da representação visual desta e notifica a visão (view) da necessidade de ser atualizada quando a tarefa está pronta.

6 TESTES DE USABILIDADE

Usabilidade engloba aspectos relacionados à interface e a interação do usuário com o computador, e objetiva descrever a qualidade da interação de usuários específicos com uma interface de software específica, e que pode ser mensurada, segundo a ISO 9241-11 (ISO, 2013), pelos critérios de efetividade, eficiência e satisfação. De acordo com ISO 9241:

- **Efetividade**: deve mensurar o quanto os usuários alcançaram seus objetivos iniciais de interação, avaliando a conclusão da tarefa e a qualidade da mesma;
- **Eficiência**: deve mensurar a quantidade de esforço e recursos necessários para se chegar a um determinado objetivo;
- **Satisfação**: deve mensurar o nível de conforto que o usuário sente ao utilizar a interface com a meta de alcançar seus objetivos.

6.1 Objetivo e Metodologia

A abordagem utilizada neste projeto foi a combinação de teste de usabilidade e experimentos controlados, baseado em tarefas de usuários (Park & Bohner, 2012):

- **Avaliação de Experimentos Controlados**: Trata-se de um experimento de curto prazo, que abrange certo número de experimentos controlados. Estes experimentos medem o tempo utilizado para realizar determinada tarefa, e a qualidade da solução utilizada. Pode ser útil também para comparação de ferramentas de software com as mesmas características, e é sensível a qualidade dos dados.
- **Avaliação de Usabilidade**: é abordagem considerada de curto prazo, e tipicamente foca em problemas de usabilidade relativos a interface da ferramenta. Este tipo de avaliação identifica problemas, proporciona feedback nos problemas identificados e ajuda a desenvolver uma nova solução.

6.2 Plano de Teste

O plano de teste definido para este projeto apresenta os seguintes quesitos:

- Perfil do usuário alvo;
- Ambiente de teste: hardware, software e estado inicial da aplicação;
- Descrição de aplicação do teste;
- As tarefas definidas para os testes, que devem conter: a descrição da tarefa, tempo limite, objetivo, solução, complexidade e métricas de êxito e tempo de realização;

O perfil dos usuários escolhidos, para esse estudo de usabilidade preliminar, foi basicamente

alunos do curso de ciência da computação e sistema de informação da UFPA com alguma experiência no uso de smartphones e/ou tablets.

O ambiente de teste foi realizado em um Sansung Galaxy Tab 3 de 8", com Android 4.2.2. Os testes foram realizados individualmente. Foi realizada uma pequena introdução de cinco minutos sobre interação nos tablets, sobre as técnicas de visualização da informação e, mais cinco minutos sobre o funcionamento e objetivos da aplicação e base de dados utilizada, nada se apresentou da interface em específico.

Quanto à coleta de informações, o avaliador acompanhou a interação do usuário junto à aplicação, sem a possibilidade de intervir, com exceção do tempo limite das tarefas. O avaliador realizou anotações informais a respeito de aspectos não-definidos, baseado em observações.

Os testes foram realizados no âmbito do campus na Universidade Federal do Pará, utilizando dados sobre carros, tais como: marca, tipo de combustível utilizado, tipo de tração, peso, potência, consumo, valor, etc, ou seja, dados de um domínio de problema conhecido pelos usuários. Ao total a base apresenta 14 atributos, sendo destes 8 categóricos e 6 numéricos, com 191 registros.

Foram definas 3 tarefas. As tarefas foram classificadas em três categorias: tarefas de baixa complexidade (requer uma resposta simples, como um "sim" ou "não"), tarefa de média complexidade (não requer a análise de vários atributos, no máximo dois) e tarefas de alta complexidade (requer uma resposta mais elaborada e a análise de três ou mais atributos) (MACIEL, MEIGUINS, *et al.*, 2008)

As tarefas são descritas a seguir:

- Tarefa 1: Baixa complexidade
    - **Pergunta**: A maioria dos carros com potência acima de 200 é movida a gasolina? Resposta: SIM.
    - **Cenário:**
        - Filtro potência
        - Configurar combustível
    - **Tarefas Visualização**:
        - Identificar / Configurar – potência maior 200
        - Visualizar / Configurar - combustível
        - Comparar – combustível
        - Localizar – qual combustível é representado pelo maior número de itens.
    - **Tempo Sugerido**: 1:30 min

- Tarefa 2: Média Complexidade
    - **Pergunta**: De maneira geral, dentre os carros 4x4, qual o tipo de carro apresenta maior valor e qual o tipo de carro apresenta maior peso?

- **Resposta**: Peso – Wagon e valor - hatch
- **Cenário:**
  - Filtro – tração
  - Configurar tipo, valor e peso, podendo utilizar filtro para valor e peso.
- **Tarefas de Visualização:**
  - Identificar / Configurar – tração 4x4
  - Visualizar / Configurar – tipo
  - Visualizar / Configurar – valor
  - Visualizar / Configurar – peso
  - Comparar – valor / peso
  - Localizar – qual tipo tem maior valor e qual tipo tem maior peso
- **Tempo Sugerido**: 3 min

- Tarefa 3: Alta Complexidade
  - **Pergunta**: Compare os carros do tipo hatch e sedan fabricados nos anos de 1980 e 1990, da marca Honda, no que se refere à potência, peso e valor.
  - **Resposta**: 1980: mais pesados, mais caros e mais potentes, 1990: menos pesados, menos caros e menos potentes
  - **Cenário:**
    - Filtro – tipo
    - Filtro – ano
    - Filtro – marca
    - Configurar ano, potência, peso e valor.
  - **Tarefas de Visualização**
    - Identificar / Configurar – tipo hatch e sedan
    - Identificar / Configurar – ano 80 e 90
    - Identificar / Configurar – marca honda
    - Visualizar / Configurar – potência
    - Visualizar / Configurar – peso
    - Visualizar / Configurar – valor
    - Visualizar / Configurar – tipo
    - Comparar – peso / valor / potência
    - Localizar – maior peso / valor / potência
  - **Tempo Sugerido**: 5 min

6.3 Resultados Obtidos

O perfil dos usuários foi alunos do curso de computação com alguma experiência na utilização de smartphones e tablets. Idade entre 20 e 25 anos, com total de 5 alunos, entre homens e mulheres.

Em relação a execução com sucesso ou não das tarefas, o índice de sucesso foi alto, em torno de 87% (Figura 45), o que se pode concluir que após o treinamento, o entendimento e a interação do usuário da com interface foi excelente. Contudo, houve relatos para a necessidade de pistas para descobrir as funcionalidades. O tempo de execução das tarefas foi bom, os valores foram arredondados para cima ou para baixo, a tarefa de tempo mais alto foi a Tarefa 3, de certa maneira esperada por ser a tarefa mais complexa (Figura 46).

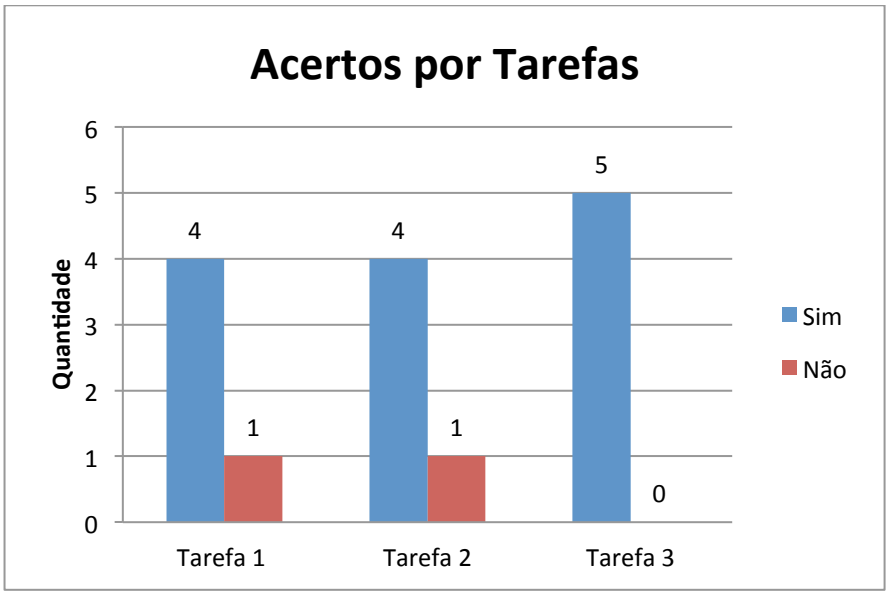

Figura 45: Quantidade de acertos por tarefa

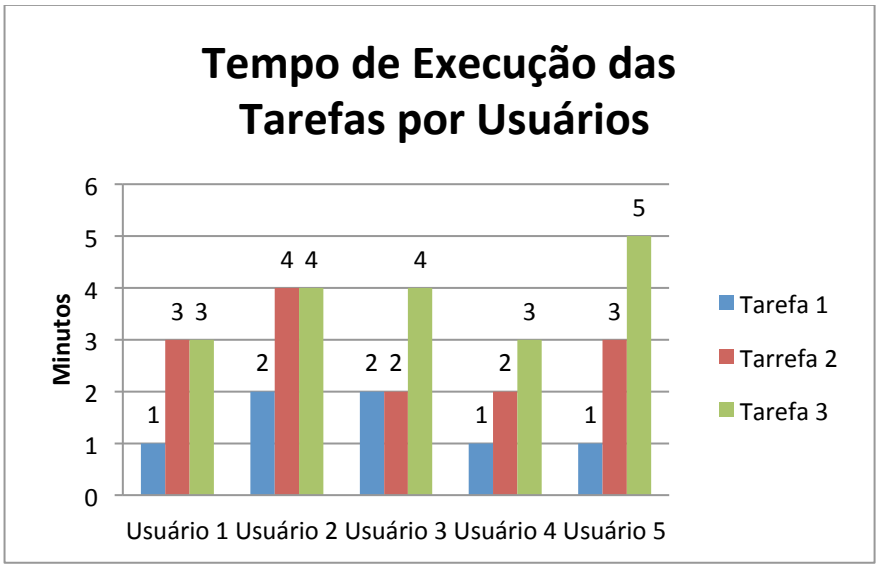

Figura 46: Tempo de execução (em minutos) das tarefas.

De maneira geral, a aplicação foi considerada de fácil utilização, mas os usuários recomendaram dicas ou atalhos para achar as funcionalidades mais facilmente. E poucos usuários utilizaram o recurso de múltiplas visões coordenadas.

7 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

A necessidade de mobilidade para executar tarefas e tomar decisões em qualquer lugar e tempo, tem motivado a construção de serviços computacionais, nos mais variados domínios e direcionado o desenvolvimento de software ao redor do mundo.

As características dessa nova modalidade de software baseados em serviços permite que as corporações entreguem funcionalidades de negócio em pequenos blocos aos seus clientes, facilitando a substituição e evolução independente destes, bem como a construção de novas aplicações a partir desses blocos iniciais.

Este trabalho teve como objetivo principal a construção de um serviço de visualização de dados para propósitos gerais de forma a favorecer a colaboração e o aproveitamento de diferentes contextos físicos e sociais dos potenciais consumidores desse serviço, contextos estes inerentes da tarefa exploração de dados.

A construção deste serviço permitiu:

- Estabelecimento de um modelo conceitual de uma ferramenta de visualização de informações com as principais características como filtro, zoom e detalhes sob demanda.
- Construção de uma arquitetura orientada a serviços (SOA) para abstração do motor de geração de visualização de dados do PRISMA.
- Validação do serviço a partir da construção de um cliente na plataforma *Android*, com componentes para filtro discreto, filtro contínuo, zoom, detalhes sob demanda, configurações gerais e configurações específicas para as técnicas de disponíveis: dispersão de dados, *treemap* e coordenadas paralelas.
- Realização de testes de usabilidades preliminares com o protótipo desenvolvido mostrando uma boa aceitação do usuário.

Este trabalho apresentou como maiores desafios:

- Modelar o domínio de uma ferramenta de visualização como uma API REST.
- Especificar um contrato agnóstico de plataforma para prover um serviço de visualização de dados para construção de ferramentas de visualização clientes.
- Realizar a transição de uma ferramenta *standalone* para uma arquitetura cliente-servidor.
- Extrair uma API de interação com provedores de visualização para permitir a criação de um adaptador para o Núcleo do PRISMA e possibilitar a construção de novos adaptadores.

- Manter o cliente tão leve quanto possível, evitando desperdício de banda ao não recarregar frequentemente itens com menor volatilidade e ao delegar processamento pesado ao servidor e utilizar armazenamento de dados na "nuvem". Além da limitação dos recursos de um dispositivo inteligente, outra grande preocupação que permeou o desenvolvimento do cliente foi a não inclusão de lógica de negócio na sua base de código, apenas lógica da interface do usuário.

Como potenciais de trabalho futuro destacam-se os seguintes pontos:

- Atingir o nível 3 de maturidade de Richardson, facilitando a descoberta e poder de auto documentação dos serviços.
- Substituir o PRISMA por outro motor de visualização de dados, existente ou novo para avaliar a abstração desenvolvida e o grau de reúso da API.
- Medir o tráfego de rede e o consumo de recursos com intuito de avaliar a escalabilidade da aplicação.
- Expandir as capacidades de carregamento de dados da ferramenta para incluir formatos não estruturados como imagens de outras visualizações estáticas.
- Avaliar técnicas de visualização mais apropriadas para diferentes dispositivos, levando em consideração os seus recursos como: largura de banda, tamanho da tela, mecanismos de interação.
- Realizar testes de usabilidade com maior número de usuários, outros perfis, outras bases de dados, e o uso da ferramenta de modo colaborativo.
- Captar dados do contexto físico, via sensores, além de informações sociais do usuário com intuito de aprimorar a escolha das técnicas padrões e permitir que a geração da interface cliente seja personalizada.
- Aumentar a imersão do usuário na visualização, eliminando controles da tela e utilizando mais espaço para visualização.
- Incorporar feedback dos testes de usabilidade na aplicação cliente, aumentando a ajuda contextual e favorecendo o entendimento do modelo conceitual.

8 REFERÊNCIAS

BALDONADO, M. Q. W.; WOODRUFF, A.; KUCHINSKY, A. **Guidelines for Using Multiple Views in Information Visualization**. Proceedings of the working conference on Advanced visual interfaces. Palermo, Italy: ACM Press. 2000. p. 110-119.

BODYWORLDS. BodyWorlds. **BodyWorlds**, 2013. Disponivel em: <http://www.bodyworlds.com/en/media/picture_database/thumbnails.html?category=1>. Acesso em: 08 Agosto 2014.

BRATH, R. K. **Effective Information Visualization Guidelines and Metrics for 3D Interactive Representations of Business**. [S.l.]. 1999.

BUSCHMANN , et al. **Pattern-Oriented Software Architecture Volume 1:** A System of Patterns. [S.l.]: Wiley, v. 1, 1996.

CARD , S. ; MACKINLAY , ; SHNEIDERMAN , B. **Readings in Information Visualization:** Using Vision to Think. [S.l.]: [s.n.], 1999.

CARR, D. A. **Guidelines for Desegning Information Visualization Applications**. Proceedings of Ericsson Conference in Usability Engineering. [S.l.]: [s.n.]. 1999. p. 1-7.

COCKBURN, A. Alistair Cockburn. **Hexagonal Architecture**, 2009. Disponivel em: <http://alistair.cockburn.us/Hexagonal+architecture>. Acesso em: 02 Outubro 2014.

DA SILVA JUNIOR, J. D. J. N. et al. **PRISMA Mobile:** An Information Visualization Tool for Tablets. Proceedings of 16th International Conference on Information Visualisation (IV'12). Montpellier: [s.n.]. 2012. p. 182-187.

DAIGNEAU, R. **Service Design Patterns:** Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services. [S.l.]: Addison-Wesley Professional, 2011.

DOS SANTOS, S. R.; BRODLIE, K. **Gaining understanding of multivariate and multidimensional data through visualization.** Computer & Graphics. [S.l.]: [s.n.]. 2004. p. 311-325.

EICK, S. G. Graphically displaying text. **Journal of Computational and Graphical Statistics**, Junho 1994. 127-142.

ERL, T. **Service-Oriented Architecture (SOA):** Concepts, Technology, and Design. [S.l.]: Prentice Hall, 2005.

EVANS, E.; FOWLER, M. **Domain-Driven Design:** Tackling Complexity in the Heart of

Software. [S.l.]: Prentice Hall, 2003.

FEW, S. **Now You See It:** Simple Visualization Techniques for Quantitative Analysis. [S.l.]: [s.n.], 2009.

FIELDING, R.; RESCHKE, J. RFC 7231 - Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. **Internet Engineering Task Force (IETF)**, 2014. Disponivel em: <https://tools.ietf.org/html/rfc7231>. Acesso em: 05 maio 2014.

FOWLER, M. **Patterns of Enterprise Application Architecture**. [S.l.]: Addison-Wesley Professional, 2002.

GAMMA, E. et al. **Design Patterns:** Elements of Reusable Object-Oriented Software. [S.l.]: Addison-Wesley Professional, 1994.

GAPMINDER. Gapminder. **Gapminder World 2012**, 2012. Disponivel em: <http://www.gapminder.org/downloads/world-pdf/>. Acesso em: 15 Setembro 2014.

GODINHO, P. I. A. et al. **PRISMA – A Multidimensional Information Visualization Tool Using Multiple Coordinated Views**. Proceedings of 11th International Conference Information Visualization (IV '07). Zurich: [s.n.]. 2007. p. 23-32.

GRADNJEAN, M. La connaissance est un réseau. **Les Cahiers du Numérique**, 2014. 37-54.

HEER, J.; BOSTOCK, M.; OGIEVETSKY, V. A Tour through the Visualization Zoo. **Communications of the ACM**, New York, Junho 2010. 59-67. Disponivel em: <http://www.ggobi.org/>.

HUMAN-COMPUTER INTERACTION LAB / UNIVERSITY OF MARYLAND. Treemap: Home Page. **Treemap**, 2003. Disponivel em: <http://www.cs.umd.edu/hcil/treemap/>. Acesso em: 31 out. 2011.

INSELBERG, A.; DIMSDALE, B. **Parallel Coordinates:** a tool for visualizing multi-dimensional geometry. Proceedings of the 1st conference on Visualization. San Francisco: IEEE Computer Society Press. 1990. p. 361-378.

ISO. ISO 9241-11:1998. **ISO - International Organization for Standardization**, 2013. Disponivel em: <http://www.iso.org/iso/catalogue_detail.htm?csnumber=16883>. Acesso em: 09 set. 2014.

ITC. Geovisualization and Minard's map. **International Institute for Aerospace Survey and Earth Sciences (ITC)**, 2001. Disponivel em: <http://www.itc.nl/personal/kraak/1812/minard-itc.htm>. Acesso em: 2 Outubro 2014.

JACKO, J. **Human Computer Interaction Handbook:** Fundamentals, Evolving Technologies, and Emerging Applications. [S.l.]: CRC Press, 2012.

LEE, N. A.; MOCTEZUMA, L. E.; LASTRA, J. L. **Visualization of Information in a Service-Oriented Production Control System**. Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE. Vienna: [s.n.]. 2013. p. 4422 - 4428.

MACIEL, M. D. P. D. S. et al. **The Impact of Multiple Coordinated Views on the Visual Data Exploration and Analysis**. Information VIsualization. London: [s.n.]. 2008. p. 113-119.

NASA. Goddard Earth Science Data and Information Services Center. **EarthData**, 2006. Disponivel em: <http://disc.sci.gsfc.nasa.gov/gesNews/caspian_sea_sst_animation>. Acesso em: 13

Julho 2014.

NORTH, C.; SHNEIDERMAN, B. **Snap-Together Visualization:** A User Interface for coordinating Visulizations via Relational Schemata. Advanced visual interfaces. [S.l.]: [s.n.]. 2000. p. 23-26.

PARK, A.; BOHNER, S. **Intended Use Evaluation Approach for Information Visualization**. [S.l.]: AV Akademikerverlag , 2012.

PILLAT, R. M.; VALIATI, E. R.; FREITAS, C. M. D. **Experimental Study on Evaluation of Multidimensional Information Visualization Techniques**. CLIHC'05. Cuernavaca - Mexico: [s.n.]. 2005. p. 20-30.

POMBINHO, P. **Information visualization on mobile environments**. MobileHCI '10 Proceedings of the 12th international conference on Human computer interaction with mobile devices and services. New York: [s.n.]. 2010. p. 493-494.

RICHARDSON, L. **RESTful Web Services**. [S.l.]: O'Reilly Media, 2007.

ROCHA, H. V.; BARANAUSKAS, M. C. **Design e Avaliação de Interfaces Humano-Computador**. Campinas: NIED/UNICAMP, 2003.

SHNEIDERMAN, B. **The eyes have it:** a task by data type taxonomy for information visualizations. Proceedings of the IEEE Symposium on Visual Languages. [S.l.]: [s.n.]. 1996. p. 336 -343.

SIMON, C. A. Killer SOA Definition. **Carl's Consulting Adventures**, 2005. Disponivel em: <http://carlaugustsimon.blogspot.com.br/2005/09/killer-soa-definition.html>. Acesso em: 01 set. 2014.

SPENCE, R. **Information Visualization:** Design for Interaction (2nd Edition). [S.l.]: [s.n.], 2007.

THIEDE, C.; TOMINSKI, ; SCHUMANN,. **Service-Oriented Information Visualization for Smart Environments**. IV '09 Proceedings of the 2009 13th International Conference Information Visualisation. Washington: [s.n.]. 2009. p. 227-234.

U.S. BUREAU OF LABOR STATISTICS. **Bureau of Labor Statistics**, 2011. Disponivel em: <http://www.bls.gov/>. Acesso em: 15 Setembro 2014.

WARD, M. O.; GRINSTEIN , G.; KEIM,. **Interactive Data Visualization:** Foundations, Techniques, and Applications. [S.l.]: [s.n.], 2010.

WARE, C. **Information Visualization:** Perception for Design. 2ª. ed. San Francisco: Elsevier, 2004.

WEBBER, J.; PARASTATIDIS, ; ROBINSON, I. **REST in Practice:** Hypermedia and Systems Architecture. [S.l.]: O'Reilly Media, 2010.