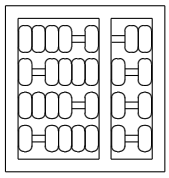

INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Rastreamento de viagens compartilhadas publicadas no facebook

Luiz F. Takakura *Leandro Aparecido Villas*

Relatório Técnico - IC-PFG-16-11 - Projeto Final de Graduação

December - 2016 - Dezembro

Rastreamento de viagens compartilhadas publicadas no facebook

Luiz F. Takakura* Leandro A. Villas*

Resumo

O compartilhamento de viagens intermunicipais é uma prática bastante difundida principalmente entre os estudantes de universidades públicas. A popularização de tal prática provocou um elevado crescimento das comunidades digitais destinadas a este fim presentes em redes sociais como, por exemplo, o *Facebook*. O crescimento das ofertas de compartilhamento de viagens provocou um bombardeamento intenso de informações na rede social fato que, ironicamente, prejudicou a prática do ponto de vista de quem busca a viagem. Utilizando a ideia por trás do conceito de redes de sensoriamento participativo e buscando a melhoria da experiência do usuário que busca o compartilhamento de viagens através do *Facebook*, propôs-se criar uma aplicação capaz de coletar o grande volume de informações geradas pelos sensores sociais para interpretá-las e então servi-las aos usuários de maneira organizada por intermédio de um messenger bot inteligente.

1 Introdução

Os estudantes que frequentam a Universidade Estadual de Campinas são em sua grande maioria de outras cidades ou mesmo até de outros estados. Este é um fato não só observado na Unicamp, mas em grande parte das universidades públicas brasileiras.

Muitos destes estudantes se mudam para as cidades sede das universidades que frequentam, mas retornam às suas cidades de origem esporadicamente, comportamento este que acaba gerando gastos elevados tanto aos que utilizam seu próprio meio de transporte como àqueles que dependem de transporte público rodoviário para se locomover.

Tendo em vista o alto custo dessas viagens, os próprios estudantes começaram a se organizar em pequenos grupos para compartilhar suas viagens e assim diminuir os custos para ambos os lados - tanto para quem possui carro como para os que não possuem. Estes grupos foram crescendo, se especializando em seus distintos pontos de origem e destino e extrapolando o âmbito estudantil, alguns alcançando as marcas de dezenas de milhares de participantes e denotando um grande número de pessoas interessadas em compartilhar viagens.

Com tamanho crescimento, ferramentas pré-existentes foram adotadas para suprir as necessidades dos grupos mencionados. A primeira destas ferramentas foi o *Google Groups* ou mesmo as já conhecidas listas de emails. Através destas os participantes (ingressados por

*Instituto de Computação, Universidade Estadual de Campinas

meio de convites de amigos previamente adicionados) começaram a divulgar suas ofertas de carona de maneira mais eficiente, atingindo rapidamente a totalidade dos participantes interessados do grupo. Estes últimos, porém, tinham bastante dificuldade para encontrar a oferta desejada devido à grande desorganização das informações divulgadas.

Com o surgimento das redes sociais (*Orkut* inicialmente e *Facebook* mais a frente) ocorreu um fenômeno de migração dos grupos antes estabelecidos nas listas de emails para estas plataformas. Este fenômeno facilitou o acesso aos grupos e popularizou ainda mais a prática de compartilhamento de viagens. Contudo, a organização das informações persistiu ineficiente, o que prejudica consideravelmente a busca por ofertas.

Neste contexto, foi observada a necessidade de se desenvolver um novo mecanismo que atuasse em conjunto com o já existente, para suprir a ineficiência da busca por caronas. Propôs-se então a criação de uma aplicação capaz de extrair toda a informação fornecida pelos membros dos grupos de compartilhamento de viagens, interpretá-la e então servi-la novamente à comunidade através de uma interface simples e intuitiva.

2 Justificativa

Apesar do aumento da prática de caronas, o rápido crescimento dos grupos anteriormente mencionados culminou no estabelecimento do caos. Buscar uma carona tornou-se uma tarefa extremamente desgastante do ponto de vista da experiência do usuário, pois começaram a surgir diversas ofertas de maneira desordenada, sem filtros, fazendo com que os participantes tenham que ler todas as postagens e comentários para que possam encontrar o que buscam. A principal razão deste acontecimento se deve à falta de especialização das redes sociais para tratar de forma eficiente a este tipo de finalidade.

Visando otimizar estas buscas por viagens compartilhadas e aproveitando as plataformas já existentes e os dados voluntariamente gerados pelos usuários delas, propôs-se criar uma aplicação capaz de extrair as informações dos grupos e organizá-las para que seus próprios participantes possam, através de uma interface objetiva, executar de maneira eficiente as tarefas relacionadas ao compartilhamento de viagens.

Esta proposta explicita uma aplicação específica do conceito de redes de sensoriamento participativo. Neste tipo de sistema é estabelecido um processo distribuído de coleta de dados, os quais são obtidos por meio da participação ativa das pessoas para compartilhar voluntariamente informações sobre determinado assunto [1, 2]. Este modelo se encaixa perfeitamente com a ideia da solução, pois é gerado nos grupos de caronas um número elevado de informações relevantes ao tráfego intermunicipal, já que as pessoas são as maiores interessadas no bom funcionamento do compartilhamento de viagens.

Assim vale ressaltar que, além do benefício diretamente relacionado à comunidade, a solução aqui proposta faz uso da técnica de sensoriamento participativo utilizando os próprios usuários como sensores para se obter informações valiosas acerca do tráfego intermunicipal do estado de São Paulo. Informações estas que não seriam facilmente obtidas utilizando sensores físicos [4].

3 Objetivo

Utilizando o conceito estabelecido de sensoriamento participativo, e a motivação de se criar um sistema otimizado para facilitar a busca por viagens compartilhadas, este projeto focou em utilizar os participantes de um número discreto de grupos de caronas presentes na rede social *Facebook* como sensores para extrair e interpretar informações acerca do compartilhamento de viagens intermunicipais entre as cidades de Campinas e suas adjacências.

Com o foco estabelecido, foram traçados os seguintes objetivos secundários:

- Obter o maior número de informações úteis sobre o compartilhamento de viagens, com o intuito de popular uma base de dados utilizada para dispor as informações sob demanda dos os usuário de maneira interpretada;
- Estudar e compreender os padrões de linguagem utilizados pelos participantes dos grupos de compartilhamento de viagens a fim de se classificar os dados obtidos através da coleta de informações dos sensores;
- Automatizar a busca por viagens compartilhadas para os participantes dos grupos utilizados no projeto através da interface do *Facebook messenger* por intermédio de um *messenger bot* capaz de entender a necessidade do usuário e realizar a busca adequada na base de dados com as informações interpretadas e classificadas.

4 Desenvolvimento do trabalho

Para a realização do trabalho, focamos na análise de 4 grandes grupos destinados ao compartilhamento de viagens na rede social Facebook. São eles:

1. Caronas - Campinas / São Paulo (18.325 membros) [15];
2. Caronas Sorocaba - Campinas (3.300 membros) [15];
3. Caronas Campinas Limeira (UNICAMP) (4.458 membros) [15];
4. Caronas Rio Claro - Campinas (2.339 membros) [15].

A seleção de tais grupos se deve a popularidade de seus trajetos, o que faz com que apresentem volume e frequência de postagens, fatores fundamentais para que as aplicações criadas possam ser avaliadas.

A seguir temos a visão em alto-nível da aplicação:

Figura 1: Visão de alto nível da aplicação

A aplicação ilustrada acima é subdividida em dois módulos distintos que são executados em máquinas diferentes. O primeiro destes é denominado ***Extractor*** e o segundo ***Interpreter***.

A seguir serão justificadas as escolhas tecnológicas da aplicação e posteriormente é apresentada uma visão detalhada sobre o funcionamento dos serviços e módulos desenvolvidos.

NodeJS

Node, como é chamado, é uma plataforma desenvolvida por *Ryan Dahl* em 2009 e funciona como um interpretador de *javascript* no lado do servidor. Ele é baseado na *engine V8* desenvolvida pelo *Google* em *C++*, a qual é utilizada como interpretador *javascript* no navegador *Chrome* [11, 12].

A plataforma permite que suas aplicações sejam desenvolvidas em *javascript*, é totalmente assíncrono permitindo um grande volume de operações I/O não-bloqueantes e direcionado a eventos. Para obter tal comportamento, a plataforma faz uso das bibliotecas *libev* e *libeio* de *C* [7, 8].

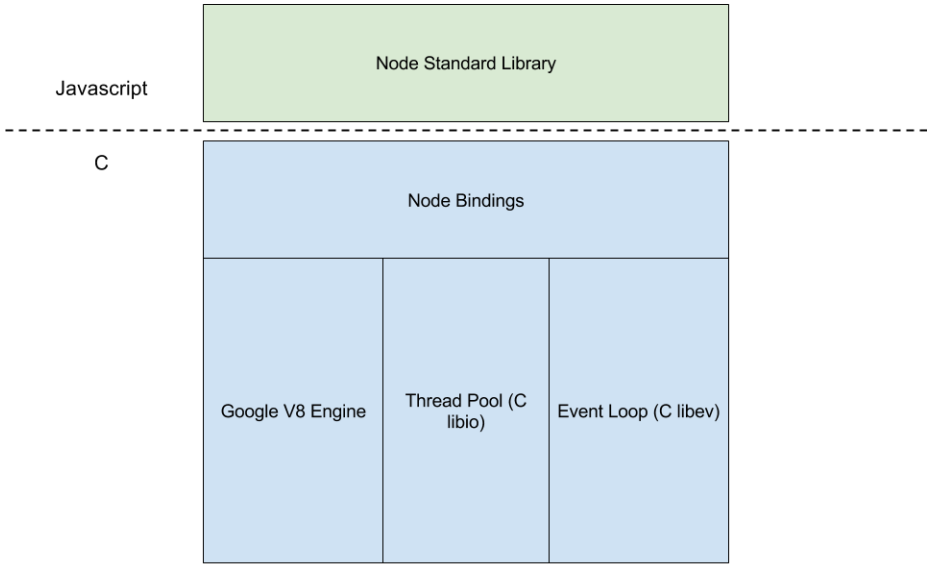

Figura 2: Arquitetur básica do *NodeJS*

Um aspecto importante do *framework Node* é a sua execução *single-threaded* (utilizando apenas um processo) orientada a eventos utilizando o *EventLoop*, que escuta eventos vindos do *front-end* e os envia para *handlers* que respondem de forma assíncrona utilizando o esquema de *libeio*callbacks muito utilizados na linguagem *javascript*. Isto permite com que a plataforma execute e forma não-bloqueante e elimina problemas de concorrência presentes em programas tradicionais baseados em *threads* [9, 10].

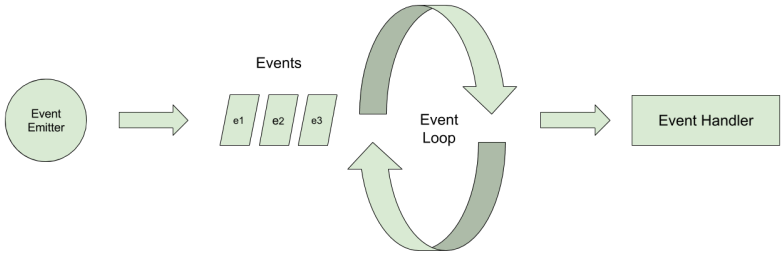

Figura 3: *Node Event Loop*

Outro aspecto importante sobre a plataforma é a presença do *NPM (Node package manager)*. Este gerenciador de pacotes já conta com mais de 350.000 módulos *open-source* utilizados por mais de 4 milhões de desenvolvedores no mundo.

A utilização do *NPM* também facilita o gerenciamento de dependências do projeto facilitando o *deploy* em diferentes ambientes.

A escolha do *Node* se deve principalmente à sua escalabilidade e à facilidade de integração do *framework* com diversos serviços *web* existentes, tais como o *Firebase* e a própria *API* de desenvolvimento do *Facebook*.

Base de Dados *Firebase*

O *Firebase* foi fundado em 2011 por *Andrew Lee* e *James Tamplim*. No início, o *Firebase* fornecia uma *API* para a integração de *online chats* em *websites*. O serviço evoluiu para um *realtime-database* de uso geral permitindo desenvolvedores sincronizarem seus dados em múltiplos clientes [13]. Hoje o *Firebase* fornece uma gama de serviços que auxiliam a criação de aplicações de diversas maneiras.

Neste projeto o uso do *Firebase* se restringe à sua base dados. A base de dados fornecida pelo serviço é distribuída na nuvem, *NoSQL* orientada a documentos (*JSON*) e de rápida sincronização. O serviço possui bibliotecas para integração com *NodeJS* e é estruturado em árvores *JSON* o que dispensa traduções dos dados recebidos nas requisições às *APIs* de terceiros.

Wit.ai

O *Wit.ai* é um serviço que fornece processamento de linguagens naturais para desenvolvedores. Ele possui uma *API REST* e um *SDK* para *NodeJS* disponível via *NPM*. Desde o início de 2015 a plataforma pertence ao próprio *Facebook*, o qual a disponibiliza sem custos [14].

A escolha do serviço para o projeto foi devido a fácil integração com o *framework* utilizado e fornece uma ferramenta extremamente importante para o funcionamento da interface de busca da aplicação.

APIs externas

Além das *APIs* descritas anteriormente, a aplicação faz uso dos seguintes serviços:

- *Facebook Messenger API* - Este serviço fornece a integração com o aplicativo *Facebook Messenger*, através do qual os usuários da aplicação fazem suas requisições;
- *Facebook Graph API* - Esta *API* possibilita a extração dos dados dos grupos da rede social programaticamente;
- *Google Places API* - Este serviço auxilia na obtenção de dados de geolocalização precisos a partir de nomes de locais obtidos pela extração dos dados da rede social.

Extractor

Definidas as tecnologias utilizadas, começaremos descrevendo a aplicação pelo módulo da plataforma denominado ***Extractor***.

Este módulo é responsável por se comunicar com a *Facebook Graph API*, extrair os textos relacionados às postagens e comentários nos grupos de viagens compartilhadas, processar os dados obtidos estruturando a informação relevante obtida, realizar consultas à *Google Places API* para obter dados precisos de geolocalização e finalmente armazenar a informação estruturada na base de dados (*Firebase*).

A figura 4 abaixo ilustra os submódulos do serviço:

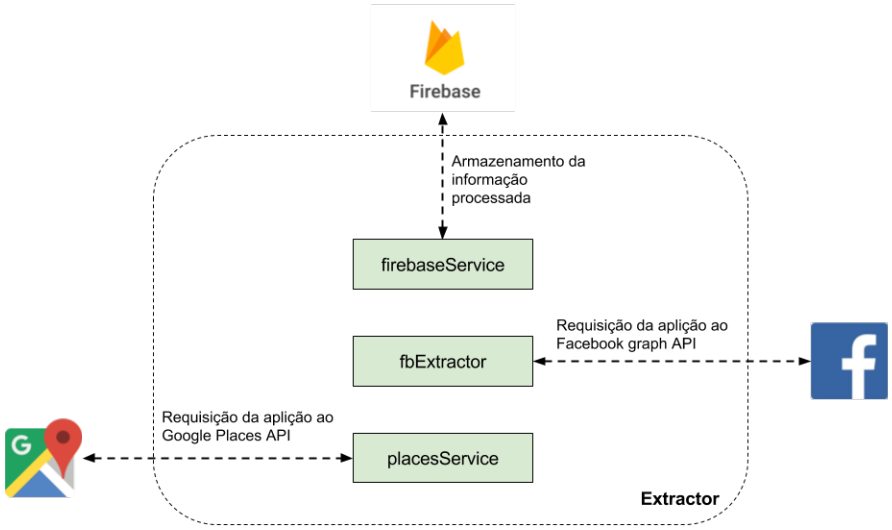

Figura 4: *Extractor* - arquitetura básica

Os serviços ***firebaseService*** e ***placesService*** ilustrados acima executam a função de interfaces para as *APIs* da base de dados e do Google Places respectivamente.

No caso da primeira, a lógica é bastante simplificada, sendo que fornece apenas um conjunto de métodos para leitura e escrita no *Firebase*. Já o ***placesService*** possui maior complexidade. O serviço deve executar a seguinte sequência de ações:

1. Realiza uma primeira requisição à *API Google* para obter o *place id* a partir do nome de uma localização enviada como parâmetro ao método responsável do serviço.

   Estrutura da requisição:

```
https://maps.googleapis.com/maps/api/place/textsearch/json
   ?query=<QUERY>&key=<API_KEY>&location=22.9098833,-47.06
   25812&radius=3000000
```

   Estrutura do *JSON* de resposta (resumido):

```
{
  "results": [{
    "place_id": "ChIJm2uboM3GyJQR45OqGNCqay4"
  }],
  "status": "OK"
}
```

2. Espera o retorno da primeira requisição e então realiza outra com o *place id* obtido para recuperar os detalhes necessários da localização;

   Estrutura da requisição:

```
https://maps.googleapis.com/maps/api/place/details/json?
   placeid=ChIJva8anaJZzpQRmckmgyhpZUQ&key=AIzaSyBZSnUtj5
   FrBDNcPayMSS5SNtg8Nbf1S8M&language=pt-BR
```

Estrutura do *JSON* de resposta (resumido):

```
{
  "result": {
    "address_components": [{
      "long_name": "Liberdade",
      "short_name": "Liberdade",
      "types": ["sublocality_level_1", "sublocality", "
         political"]
    }, {
      "long_name": "S  o  Paulo",
      "short_name": "S  o  Paulo",
      "types": ["locality", "political"]
    }],
    "geometry": {
      "location": {
        "lat": -23.5688767,
        "lng": -46.6401122
      },
      "viewport": {
        "northeast": {
          "lat": -23.5688716,
          "lng": -46.63967135
        },
        "southwest": {
          "lat": -23.568892,
          "lng": -46.64025915000001
        }
      }
    },
    "place_id": "ChIJva8anaJZzpQRmckmgyhpZUQ"
  },
  "status": "OK"
}
```

Os exemplos de resposta acima, ocorrem quando a localização é encontrada. No caso contrário, a requisição responde *'ZERO RESULTS'*.

O serviço ***fbExtractor*** é responsável por executar as requisições no *Facebook Graph API* e estruturar a informação obtida.Ele executa em uma máquina exclusiva em um intervalo pré-definido de 5 minutos através do crontab presente no *Unix*. A cada execução ele extrai e estrutura a informação referente às 50 postagens mais recentes dos 4 grupos mencionados anteriormente.

A seguir um exemplo de postagem:

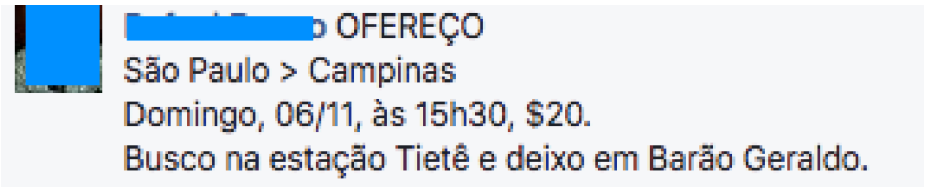

Figura 5: Exemplo de postagem de carona

O serviço executa os seguintes métodos em ordem:

1. ***filteroffer()*** - Filtra postagens que são exclusivamente ofertas (ignora as postagens de busca por carona);
2. ***filterfull()*** - Verifica se o autor da postagem marcou a sua carona como lotada e neste caso armazena um campo para identificá-la;
3. ***filterdatetime()*** - Extrai a hora e a data do oferecimento e checa se esta data e hora são válidas, ou seja, a carona ainda está vigente;
4. ***filterdata()*** - Extrai origem e destino da carona.

O módulo ***Extractor*** executa então a seguinte sequência de ações:

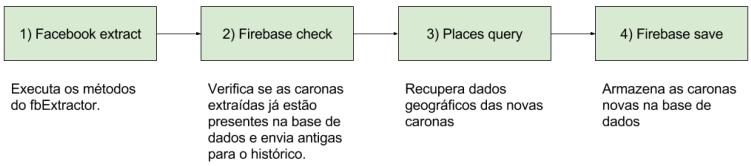

Figura 6: *Extractor* - Sequência de execução

Se qualquer um dos passos descritos na figura 6 falhar, a carona não é considerada válida e então não é armazenada. Após o passo 3, o seguinte objeto será armazenado:

```
{
 "timestamp": <int>,
 "datetime": <string>,
 "message": <string>,
 "author_id": <int>,
 "orig":
 {
   "locality": <string>,
   "geohash": <int>
 },
 "dest":
 {
   "locality": <string>,
   "geohash": <int>
 },
 "from": <string>,
 "to": <string>
}
```

Após todos os passos descritos anteriormente, a seguinte estrutura para a base de dados foi construída:

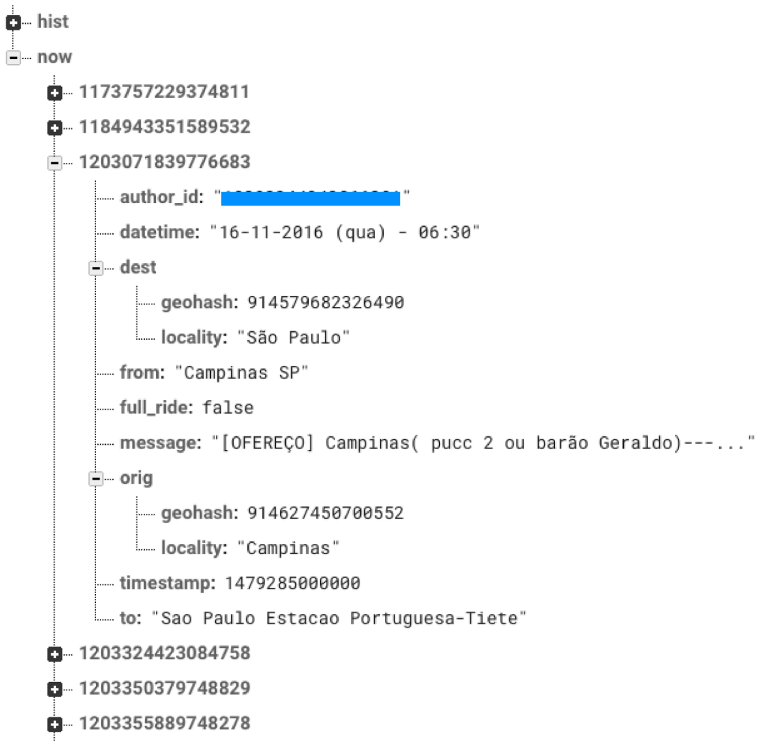

Figura 7: *Firebase* - Estrutura

As caronas são indexadas pelo identificador único da postagem de origem no *Facebook*. O grupo de dados abaixo de *now* representa as caronas vigentes enquanto que o grupo abaixo de *hist* representa o histórico de caronas registradas.

Interpreter

Para definir o módulo ***Interpreter***, primeiro é descrito a interface desenvolvida para auxiliar a interação dos usuários com a aplicação.

Como mencionado anteriormente, a interface baseia-se totalmente no *Facebook Messenger* e seu objetivo é facilitar a busca por caronas extraídas através do módulo ***Extractor***. Para tornar a experiência dos usuários a mais simples e prática possível - mesmo para quem não está habituado a interagir com interfaces de aplicativos móveis em geral - foi criado um bot capaz de assimilar qual as necessidades dos usuários e traduzi-las requisições inteligíveis para o módulo da aplicação. Desta forma, elimina-se a curva de aprendizado dos usuários pois estes não precisam se adaptar a uma nova interface, exigindo apenas que se comuniquem de forma clara utilizando linguagem natural.

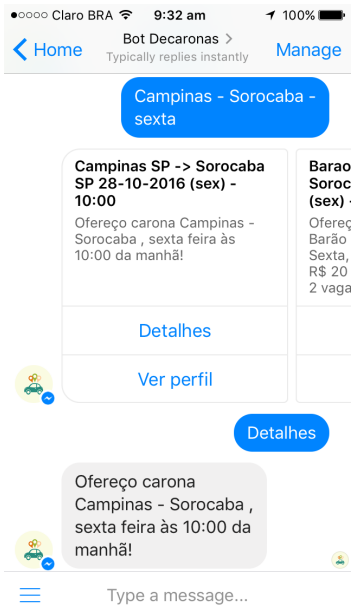

Figura 8: Exemplo de interação com o *bot*

Definido o modelo de interface escolhido, detalharemos o módulo em si. O Interpreter funciona como uma *API* aberta simplificada, pois possui apenas um único endpoint que trabalha como *webhook* para tratamento das mensagens encaminhadas ao *bot*. Toda mensagem de usuário chega no módulo com um *payload* similar ao do exemplo abaixo:

```
1 {
2   sender:
```

```
  {
    id: '1030963503679153'
  },
  recipient:
  {
    id: '1828226190795823'
  },
  timestamp: 1479604497473,
  message:
  {
    mid: 'mid.1479604497473:868bad5a47',
    text: 'Buscar'
  }
}
```

A primeira tarefa do módulo é pré-classificar o texto da mensagem do usuário, isto é, definir a direção que a mensagem deve tomar na aplicação. A direção diz respeito aos serviços onde a mensagem será de fato classificada. Em alguns casos, a mensagem pode ser classificada utilizando apenas os recursos da própria aplicação, em outros, pode ser que a mensagem necessite do poder de processamento de linguagens naturais presentes no serviço externo *Wit.ai*.

A figura abaixo ilustra o caminho da mensagem enviada pelo usuário:

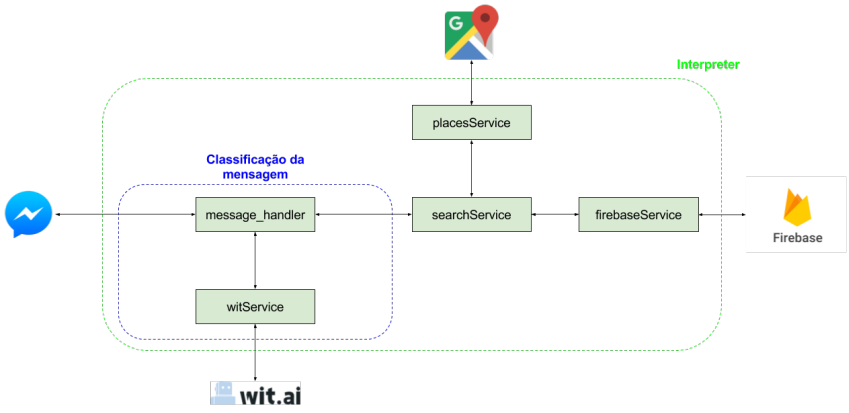

Figura 9: *Interpreter* - Arquitetura básica

O *message handler* é o primeiro a ter contato com a mensagem do usuário. Ao receber esta mensagem, o *message handler* age como um *parser* textual e verifica a presença de *keywords* pré-definidas na implementação do módulo que sejam capazes de definir a intenção do usuário. Por exemplo, se o usuário enviar uma mensagem com a palavra "ajuda", o *message handler* é capaz de classificar a mensagem sem o auxílio do *Wit.ai*. Neste exemplo específico ele não precisa executar nenhuma tarefa de busca devendo apenas enviar uma

resposta com recomendações de diálogos para direcionar o usuário para o caminho certo em direção à busca por viagens compartilhadas. Caso o *parser* não seja capaz de classificar a mensagem, ele a envia para o *witService*. Este serviço se comunica com a *API Wit* e implementa métodos que são invocados pelo *Wit.ai* para detectar a intenção do usuário e parametrizar a sua mensagem.

O *Wit.ai* deve ser configurado para classificar os diálogos com os usuários. Esta configuração se dá por meio da criação de histórias. Abaixo observamos uma dessas histórias:

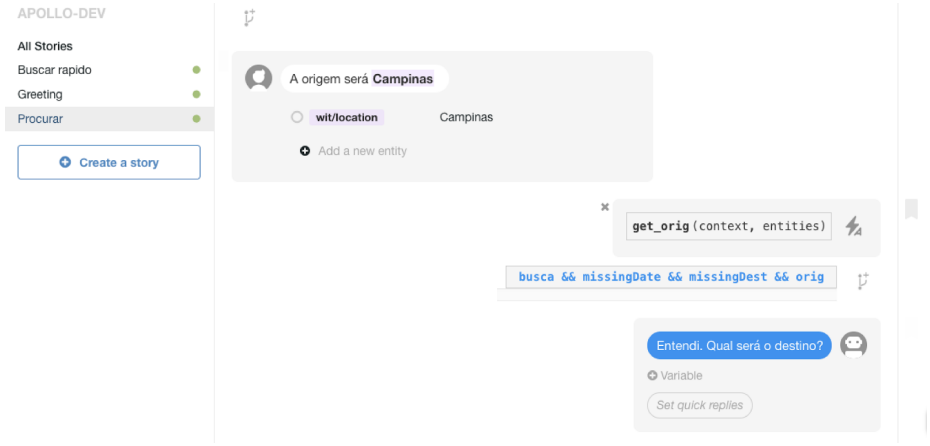

Figura 10: *Wit.ai* - trecho de história (interface *web Wit.ai*)

As histórias descrevem em alto nível como devem ser tratadas as interações com a aplicação. Nelas são descritas tanto as respostas do *bot* como as prováveis mensagens enviadas pelos usuários. Quando o *message handler* envia uma mensagem para o *witService* e este se comunica com a *API* do *Wit*, o serviço realiza o mapeamento da mensagem original com as histórias criadas. Na imagem acima está representado um trecho de uma das histórias criadas. Neste trecho o serviço espera que o usuário envie um texto cuja intenção é definir o ponto de origem de sua busca. Apesar de utilizar um texto específico ("A origem será Campinas"), o *bot* é treinado para identificar a intenção em diferentes textos. Do texto, o serviço assimila que a intenção é definir o ponto de origem e identifica a entidade location (no exemplo, Campinas). O próximo passo é invocar o método *getorig(context, entities)* o qual está definido no *witService* do módulo ***Interpreter***.

O parâmetro *context* representa um objeto gerenciado pela própria aplicação e que tem como função armazenar qual o contexto do diálogo em que o usuário se encontra. No caso do exemplo acima descrito, o objeto ao ser enviado como parâmetro possui a seguinte estrutura:

```
{
  busca: true,
  missingOrigin: true,
  missingDest: true,
  missingDate: true
}
```

O parâmetro *entities* representa as propriedades extraídas do texto recebido pelo serviço. No caso abordado, o objeto entities será composto apenas por *location: "Campinas"*. Assim sendo, após executar o método *getorig()*, o objeto context assumirá a seguinte forma:

```
{
  busca: true,
  missingDest: true,
  missingDate: true,
  orig: "Campinas"
}
```

Assim a mensagem é classificada conforme o diálogo progride. Ao terminar a classificação, o serviço possui as variáveis de contexto *orig*, *dest* e *date* preenchidas. Então o objeto *context* retorna ao *message handler* o qual encaminha os parâmetros para o *searchService*.

O serviço de busca recebe o contexto estruturado e possui a função de decorar este objeto, ou seja, adicionar as informações necessárias para realizar a busca na base de dados. A primeira ação do *searchService* é então obter os objetos completos de origem e destino da viagem com o auxílio do *placesService*, o qual envia uma requisição para o *Google Places API* com os nomes de origem e destino. O retorno desta requisição traz os dados geográficos de origem e destino, os quais são incorporados ao objeto original. A seguir o *searchService* realiza a busca na base de dados filtrando por município de origem e destino e ordenando os resultados por *timestamp* crescente. O resultado obtido é paginado no *searchService* e a resposta é enviada ao *message handler* em *chunks* de tamanho fixo, os quais são exibidos conforme requisição do usuário.

Quando a resposta retorna ao *message handler*, o módulo possui somente o trabalho de construir a resposta como ela deve ser exibida ao usuário.

Assim finaliza-se o ciclo de processamento do ***Interpreter***.

5 Resultados

O ***Extractor*** foi colocado em execução no dia 25/09/2016. Desde o *deploy* da aplicação até o dia em que este documento foi escrito passaram-se 55 dias.

Neste período a aplicação responsável por extrair as informações do sensor social foi executada em intervalos de de 5 minutos somando ao todo aproximadamente 15,840 execuções.

Durante as execuções, foram consideradas apenas as viagens que foram validadas, isto é, que puderam ser classificadas corretamente considerando as informações mínimas necessárias para se entender a intenção do participante que oferece a carona. Estas informações são: origem, destino, data e horário.

Tendo isto em mente, é natural que a aplicação não tenha extraído em sua totalidade todas as ofertas divulgadas nos grupos da rede social, seja por escassez de informação ou seja por dificuldade de interpretação das sentenças criadas pelos participantes. De fato, o Extractor obteve uma taxa de coleta de aproximadamente 82% de todos os *posts* nos grupos.

Esta taxa de coleta durante todo o processo de extração rendeu ao todo 2,021 viagens compartilhadas válidas armazenadas na base de dados. Destas, 815 (40.3%) são de São Paulo - SP para Campinas-SP ou vice-versa, 524 (25.9%) são de Campinas - SP para Sorocaba - SP ou vice-versa, 454 (22.5%) são de Campinas - SP para Limeira - SP ou vice-versa e 228 (11.3%) são de Campinas - SP para Rio Claro - SP ou vice-versa. Abaixo podemos observar os gráficos referentes às viagens válidas coletas por trajeto e também o dia e o horário com o maior número de ofertas de Campinas para São Paulo, denotando que o maior número de viagens ocorre na sexta-feira às 19h. Com o sistema desenvolvido, é possível obter informações com grau de granularidade variável de acordo com a necessidade e em tempo real para qualquer trajeto cujas informações sejam coletadas dos sensores. Por questões de simplificação, neste documento foram ilustradas as informações interpretadas sobre o trajeto Campinas/São Paulo.

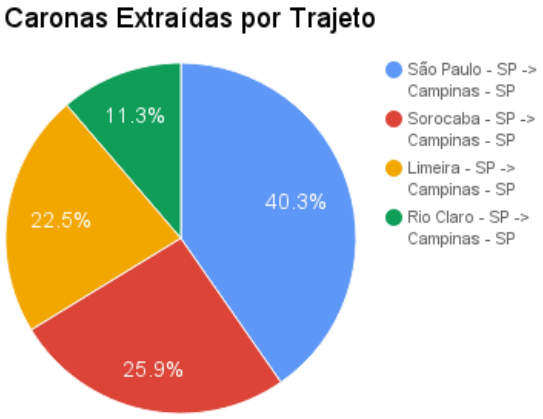

Figura 11: Caronas extraídas por trajeto

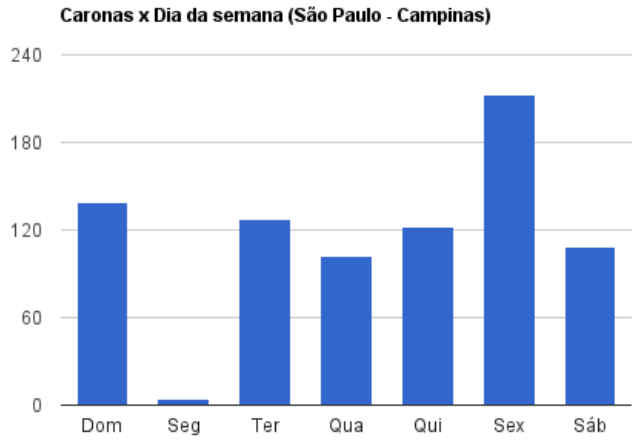

Figura 12: Caronas x Dia da semana (São Paulo - Campinas)

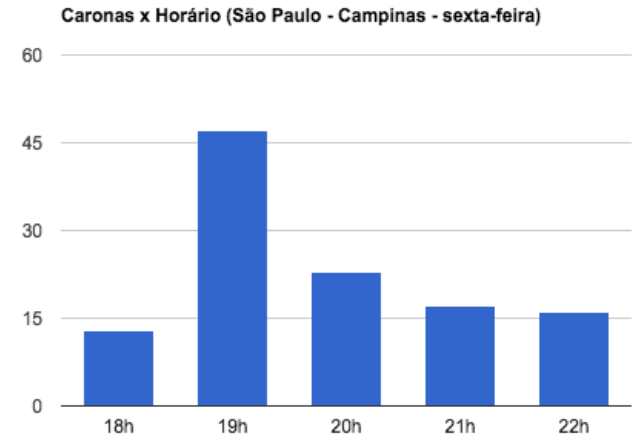

Figura 13: Caronas x Horário (São Paulo - Campinas - sexta-feira)

Já o módulo ***Interpreter*** foi disponibilizado no dia 06/11/2016 o que configura 13 dias no total desde o seu *deploy* até o dia em que este documento foi escrito. Neste período 218 usuários únicos interagiram com o *messenger bot*. Estas interações resultaram em 245 buscas distintas, ou seja, buscas que se diferenciam por pelo menos um critério mínimo (origem, destino, data ou horário). Isto resulta em uma taxa de aproximadamente 1,12 buscas distintas por usuário. Foram ao todo 165 usuários que realizaram pelo menos uma busca, o que indica que 53 pessoas não realizaram busca alguma, seja por não terem conseguido completar uma busca através da interface ou porque não indicaram interesse em buscar. Novamente, entre todas as mensagens trocadas por todos os usuários, as cidades mais mencionadas foram Campinas com 578 menções e São Paulo com 424.

6 Conclusões

A implementação da aplicação se mostrou eficiente tanto do ponto de vista da extração e tratamento dos dados obtidos através dos sensores sociais, como do ponto de vista da agilidade e facilidade de busca automatizada implementada com foco no participante dos grupos de compartilhamento de viagens hoje existentes. A grande maioria dos oferecimentos criados no *Facebook* foram extraídos com sucesso e assim foi possível criar um volume de ofertas relevante quando comparado ao volume presente na rede social, e com a vantagem de ser consideravelmente mais fácil de ser visualizado.

Os resultados obtidos com a execução da aplicação servem de exemplo do que é possível se obter com as informações extraídas dos sensores utilizados. Vale notar que a aplicação atuou sobre um volume de dados muito aquém do que é possível e por um período de tempo relativamente curto. Com um tempo maior de execução e sobre um volume de dados maior, é possível realizar análises mais profundas acerca do comportamento dos envolvidos com o compartilhamento de viagens.

Do ponto de vista do usuário da aplicação, é possível realizar diversas otimizações na forma com que hoje este tipo de atividade é realizada, como por exemplo criar um sistema de recomendações conforme a aplicação vai aprendendo sobre os hábitos de seus usuários.

Referências

[1] Benevenuto, F., Almeida, J. M., and Silva, A. S. (2011). Explorando redes sociais online: Da coleta e análise de grandes bases de dados às aplicações. SBRC'11, pages 63–94.

[2] Burke, J., Estrin, D., Hansen, M., Parker, A., Ramanathan, N., Reddy, S., and Srivastava, M. B. (2006). Participatory sensing. Workshop on World-Sensor-Web (WSW'06), pages 117–134, Boulder, USA.

[3] Cranshaw, J., Schwartz, R., Hong, J. I., and Sadeh, N. (2012). The Livehoods Project: Utilizing Social Media to Understand the Dynamics of a City. ICWSM'12, Dublin, Ireland.

[4] Fiore, M., Barcelo-Ordinas, J. M., Trullols-Cruces, O., and Uppoor, S. (2014). Generation and analysis of a large-scale urban vehicular mobility dataset. IEEE Transactions on Mobile Computing, 13(5):1–1.

[5] C. D. Manning, P. Raghavan, and H. Schütze. Introduction to Information Retrieval. Cambridge University Press., 2008.

[6] A. T. Campbell, S. B. Eisenman, N. D. Lane et al., "The rise of people-centric sensing," IEEE Internet Computing, vol. 12, no. 4, pp. 12–21, 2008.

[7] M. Lehmann. "Libev". http://software.schmorp.de/pkg/libev.html, 2016, acessado em Dezembro de 2016.

[8] M. Lehmann. "Libeio". http://software.schmorp.de/pkg/libeio.html, 2016, acessado em Dezembro de 2016.

[9] N. Trevor. "Understanding the Node.js Event Loop". https://nodesource.com/blog/understanding-the-nodejs-event-loop/, 2016, acessado em Dezembro de 2016.

[10] Node.js. "EventEmitter". http://nodejs.org/docs/latest/api/events.html, 2016, acessado em Dezembro de 2016.

[11] Google. "V8 javascript Engine". http://code.google.com/p/v8/, 2016, acessado em Dezembro de 2016.

[12] Node.js, "Node.js", http://nodejs.org/, 2016, acessado em Dezembro de 2016.

[13] Firebase, "Firebase", https://firebase.google.com/, 2016, acessado em Dezembro de 2016.

[14] Wit.ai, "Wit.ai", https://wit.ai/, 2016, acessado em Dezembro de 2016.

[15] Facebook, "Facebook", https://facebook.com/, 2016, acessado em Dezembro de 2016.