Pós-Graduação em Ciência da Computação

"UBIMID: UM MIDDLEWARE DE INTEGRAÇÃO E SENSÍVEL AOCONTEXTO VOLTADO PARA APLICAÇÕES E SISTEMAS INTELIGENTES DE TRANSPORTE"

Por

Gilson Medeiros de Oliveira Junior

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, JUNHO/2014

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

GILSON MEDEIROS DE OLIVEIRA JUNIOR

"UBIMID: UM MIDDLEWARE DE INTEGRAÇÃO E SENSÍVEL AO CONTEXTO VOLTADO PARA APLICAÇÕES E SISTEMAS INTELIGENTES DE TRANSPORTE"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADORA: ANA CAROLINA SALGADO
CO-ORIENTADOR: CARLOS ANDRÉ GUIMARÃES FERRAZ

RECIFE, JUNHO/2014

Catalogação na fonte
Bibliotecário Jefferson Luiz Alves Nazareno, CRB 4-1758

Oliveira Junior, Gilson Medeiros de.
UBIMID: um middleware de integração e sensível ao contexto voltado para aplicações e sistemas inteligentes de transporte. – Recife: O Autor, 2014.
81 f. : fig., tab.

Orientadora: Ana Carolina Brandão Salgado.
Dissertação (Mestrado) - Universidade Federal de Pernambuco. Cin. Ciência da Computação, 2014.
Inclui referências.

1. Middleware. 2. Serviço da Web. I. Salgado, Ana Carolina Brandão. (orientadora). II. Título.

005.376 (22. ed.) MEI 2014-91

Dissertação de Mestrado apresentada por **Gilson Medeiros de Oliveira Junior** à Pós Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**UbiMid: Um Middleware de Integração e Sensível ao Contexto Voltado para Aplicações e Sistemas Inteligentes de Transporte**" orientada pela **Profa. Ana Carolina Brandão Salgado** e aprovada pela Banca Examinadora formada pelos professores:

Profa. Patricia Cabral de Azevedo Restelli Tedesco
Centro de Informática / UFPE

Prof. Vagner José do Sacramento Rodrigues
Departamento de Estatística e Informática / UFG

Profa. Ana Carolina Brandão Salgado
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 26 de fevereiro de 2014.

Profa. Edna Natividade da Silva Barros
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

À minha família

Agradecimentos

Agradeço aos meus pais Gilson Medeiros e Socorro Almeida pelo incentivo, paciência e amor em todos os momentos da minha vida.
Ao meu irmão, Luiz Felipe, pelo companheirismo.
À minha namorada, Ivna Valença, pelo companheirismo, apoio, amor, carinho e incentivo em todos os momentos.
Aos meus excelentes orientadores, Ana Carolina Salgado e Carlos Ferraz pelos ensinamentos e pela paciência durante o período de orientação.
Aos professores examinadores da banca Vagner Sacramento e Patrícia Tedesco pelas contribuições com o trabalho.
Ao professor Kiev Gama pelas dicas e ideias em etapas fundamentais do projeto.
Aos integrantes do projeto UbiBus, Diogo, Wanderley e Luiz pela parceria e contribuição com o desenvolvimento do projeto.
Enfim, agradeço a todas as pessoas que não foram aqui mencionadas, mas participaram de forma direta ou indireta no projeto, minha gratidão a todos.

Smile, though your heart is aching
Smile, even though it's breaking
When there are clouds in the sky
you'll get by
If you smile through your fear and sorrow
Smile and maybe tomorrow
You'll see the sun come shining through
for you...

—CHARLES CHAPLIN (Smile, 1954)

Resumo

A mobilidade urbana é uma questão preocupante que vem ganhando bastante atenção nos últimos anos, principalmente, devido a alguns fatores como, a proximidade da realização de eventos internacionais como a Copa do Mundo de Futebol de 2014 e as Olimpíadas de 2016, além do aumento no número de carros, a má qualidade do transporte público urbano, a falta de infra-estrutura viária e o aumento da demanda do transporte público, graças ao aumento populacional.

Aliado a resolução ou a amenização desses problemas está o crescimento tecnológico, principalmente no que diz respeito a tecnologia móvel, através da popularização e do uso de dispositivos como: *smartphones* e *tablets*, por exemplo. Esse crescimento tecnológico torna a computação cada vez mais presente na vida e no cotidiano das pessoas e, com isso, surge um novo paradigma da computação chamado de Computação Ubíqua.

Nesse cenário de utilização de tecnologia móvel e da Computação Ubíqua, destacam-se as aplicações sensíveis ao contexto, que podem ser acessadas em qualquer lugar e a qualquer momento, levando em consideração informações estáticas e dinâmicas dos seus usuários. Essas aplicações, na maioria das vezes, precisam de uma infra estrutura distribuída especializadas provida através de *middleware* para o gerenciamento (processamento, aquisição e armazenamento) de informações contextuais.

Este trabalho propõe uma arquitetura de *middleware* que busca prover suporte ao desenvolvimento de aplicações ubíquas e sensíveis ao contexto. A arquitetura proposta foi definida após o estudo do estado da arte e dos requisitos necessários para o desenvolvimento desse novo tipo de aplicações. A arquitetura é baseada, principalmente, na criação e acoplamento de componentes, tornando fácil sua extensão e manutenção. A implementação da arquitetura proposta é validada através da desenvolvimento de um *middleware* chamado UbiMid que atua no domínio dos sistemas inteligentes de transporte público, desenvolvido no âmbito do projeto UbiBus.

Palavras-chave: Sistemas Inteligentes de Transporte, *Middleware* Sensíveis ao Contexto, Barramento de Serviços Empresarial, Computação Sensível ao Contexto, Serviços Web, Arquitetura Orientada a Serviços, Sistemas Distribuídos, Integração de Sistemas.

Abstract

Urban mobility is an issue of concern that has been gaining much attention in recent years, especially in Brazil, due to the proximity of international events such as the FIFA World Cup 2014 and the Olympic Games in 2016. Also, thanks to the population growth, the increase in the number of cars, the poor quality of public transport, the lack of good roads infrastructure and the increased demand for public transport increase the concerns about urban mobility.

Mitigation or even resolution of these problems is very much related to the growth of technological advances, particularly with regard to mobile technology, through the popularization and use of devices such as smartphones and tablets. This technological growth makes computing more and more present in peopleís lives. This way, a new computing paradigm emerges, called Ubiquitous Computing.

In this use scenario of mobile technology and ubiquitous computing, there are the context sensitive applications that can be accessed anywhere and at any time, taking into account static and dynamic information about their users. These applications, in most cases, require a distributed infrastructure provided through specialized middleware for management (processing, acquisition and storage) of contextual information.

This work proposes a middleware architecture that seeks to provide support to the development of ubiquitous and context-aware applications. The proposed architecture was defined after the study of the state of the art and of the requirements for the development of this new type of applications. The architecture is based mainly on the creation and coupling of components, making it easy extension and maintenance. The implementation of the proposed architecture is validated through the development of a middleware called UbiMid that operates in the field of intelligent public transportation systems, developed under the UbiBus project.

Keywords: Intelligent Transportation Systems, Context-Aware Middleware, Enterprise Service Bus, Context-Aware Computing, Web Services, Service Oriented Architecture, Distributed Systems, Systems Integration.

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

EDA	Event-Driven Architecture
CRM	Customer Relationship Management
AS	Análise de Sentimentos
ATAM	Architecture-based Tradeoff Analysis Method
APTS	Advanced Public Transportation Systems
ATMS	Advanced Trafc Management Systems
ATIS	Advanced Traveller Information Systems
CVO	Commercial Vehicle Operations
AVCS	Advanced Vehicle Control Systems
ARTS	Advanced Rural Transportation Systems
DRIVE	Dedicated Road Infrastructure for Vehicle Safity
ERP	Enterprise Resource Planning
ESB	Enterprise Service Bus
HTTP	Hyper-Text Transfer Protocol
IPI	Imposto sobre Produtos Industrializados
ITS	Intelligent Transportation Systems
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
MOM	Message Oriented Middleware
PROMETHEUS	Programme for European Traffic with Highest Efficiency and Unprecedented Safity
QoS	Quality Of Service
REST	Representational State Transfer
SAAM	Scenario-based Software Architecture Analysis Method
SALUTA	Scenario-based Architecture Level UsabiliTy Analysis
SCM	Supply Chain Management
SEMOB	Superintendência de Mobilidade Urbana
SGBD	Sistema de Gerenciamento de Banco de Dados

SIT	Sistemas Inteligentes de Transporte
SO	Sistema Operacional
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language

Sumário

1 Introdução

1.1 Motivação

A população brasileira vem crescendo bastante nos últimos anos, principalmente a população urbana, e agregado ao aumento populacional está o aumento na demanda por transporte. A quantidade de veículos vem crescendo num ritmo acelerado, entre 2001 e 2009 a frota aumentou 76,5% em todo o país (OLIVEIRA, 2009), um equivalente a 24 milhões de veículos, entre carros, caminhões e motocicletas. Houve cidades em que a frota de veículos aumentou quase 250% nesses 8 anos. O aumento do poder aquisitivo do brasileiro, somado com a redução do Imposto sobre Produtos Industrializados (IPI) em 2012 contribuiu ainda mais com o aumento do número de veículos nas ruas (AMATO, 2012). Assim, as cidades metropolitanas vêm sofrendo cada vez mais com questões relacionadas à mobilidade urbana.

Diversos problemas como: poluição ambiental, que ocasionam aumento do número de pessoas com problemas respiratórios, engarrafamentos constantes, falta de investimentos em infraestrutura dos sistemas viários (e.g. buracos, má sinalização) e a baixa qualidade dos serviços de transporte público tornam a vida da população ainda mais estressante e complicada.

Estimular o uso do transporte público é essencial para o funcionamento da mobilidade urbana, porque reduz o número de veículos nas vias, diminuindo o congestionamento, além de melhorar a qualidade de vida da população, reduzir os danos ambientais causados pela poluição dos veículos, dentre vários outros problemas. Para isso, o uso de tecnologia da informação para planejar, gerenciar e monitorar o transporte público tem se mostrado como uma solução para esses problemas.

O interesse das empresas de transporte público e urbano em utilizar tecnologia da informação é incerto, já que a primeira vista tal investimento não resultará em aumento de receita e mesmo que essa implementação fosse viável, provavelmente acarretaria um aumento nas tarifas das passagens dos ônibus. Assim, o uso da tecnologia da informação para melhorar a mobilidade urbana deve ocorrer através de soluções tecnológicas independentes do envolvimento das empresas de transporte.

Vários tipos de aplicações podem ser e estão sendo desenvolvidas para auxiliar os

passageiros do transporte público urbano, tendo como uma das características principais a mobilidade. Para dar suporte a essas aplicações, a tecnologia de *middleware* desempenha um papel fundamental, pois facilita a comunicação e coordenação de componentes de software distribuídos e muitas vezes heterogêneos, além de tratar, de forma transparente, as dificuldades e a complexidade introduzidas, por exemplo, pela mobilidade e pela comunicação sem fio (MURALIDHARAN et al., 2008).

Além dos desafios relacionados à mobilidade e comunicação, compreender o contexto atual dos indivíduos e dispositivos é uma tarefa de extrema importância para que as aplicações possam atender da melhor forma às necessidades dos usuários. Mais importante que considerar informações estáticas como, por exemplo, horários de saídas dos ônibus predeterminados pelas suas empresas, é cada vez mais necessário levar em consideração informações dinâmicas e, principalmente, externas que podem afetar o desempenho do ônibus durante o seu trajeto no decorrer da viagem. Esses fatores tanto podem envolver os próprios ônibus (velocidade atual, localização) como podem estar relacionados às condições das vias (engarramentos, acidentes, obras), horários de pico e condições meteorológicas, por exemplo.

1.2 UbiBus: Um sistema de transporte público inteligente, ubíquo e sensível ao contexto

O *middleware* proposto nesta dissertação, chamado UbiMid, faz parte de um projeto maior chamado UbiBus, cujo objetivo geral é propor técnicas, modelos, algoritmos e ferramentas em um sistema de transporte público inteligente, ubíquo e sensível ao contexto. Além disso, o UbiBus visa oferecer o acesso inteligente de informações de transporte público a passageiros, em tempo real, baseado em informações dinâmicas de contexto relacionadas aos próprios meios de transporte, às vias públicas, aos passageiros, ou a condições ambientais (SALGADO, 2010).

Dentre as metas a serem alcançadas neste projeto está o desenvolvimento de um *middleware* contendo soluções para as camadas de aquisição, processamento (ou gerenciamento de contexto) e comunicação/disseminação, bem como sua utilização através de aplicações existentes. Os detalhes referentes a arquitetura do *middleware*, juntamente com suas camadas serão abordados com mais detalhes no capítulo 4.

1.3 Objetivo

Esta dissertação de mestrado tem como objetivo geral a proposição de uma arquitetura de *middleware* sensível ao contexto voltado para aplicações na área de sistemas inteligentes de transporte. Esse arquitetura, por sua vez, tem como objetivo propor serviços sensíveis ao contexto dos seus usuários através da utilização de seus componentes. Esses componentes fornecem serviços que buscam resolver problemas relacionados a complexidade de aquisição,

enriquecimento e transformação das informações necessárias para execução dos serviços, bem como problemas relacionados à sua indisponibilidade.

O resultado esperado é a disponibilização de um conjunto de serviços que fornecem informações de melhor qualidade e personalizados aos usuários das aplicações clientes, uma vez que leva em consideração informações contextuais, bem como diminuir o retrabalho de desenvolvimento, reduzindo a complexidade de comunicação entre as aplicações e aumentando o reuso de funcionalidades para diferentes fins. Para alcançar tal objetivo, será feito um levantamento inicial do estado-da-arte com foco em *middlewares* sensíveis ao contexto, bem como *middleware* implementados utilizando uma arquitetura SOA (*Service Oriented Architecture*). Ao final, será feita a especificação e implementação de um protótipo que permitirá a validação da solução proposta.

Os seguintes objetivos podem ser citados:

a) Identificar serviços já existentes em aplicações já desenvolvidas no contexto do projeto UbiBus e integrá-los ao *middleware* para que outras aplicações também possam utilizá-los.

b) Propor uma arquitetura e desenvolver o protótipo de um *middleware* sensível ao contexto voltado para a área de transporte público inteligente que seja flexível e facilmente extensível.

c) Prover mecanismos de transformação e enriquecimento no conteúdo das mensagens enviadas ao *middleware*, permitindo que as aplicações consumidoras tenham uma maior facilidade e flexibilidade na chamada dos serviços.

d) Prover sensibilidade ao contexto durante o processo de chamada de um determinado tipo de serviços através da implementação de mecanismos de roteamento de requisições com o objetivo de tornar transparente para usuário qual serviço ele está solicitando.

e) Prover mecanismos de qualidade de serviço com o objetivo de garantir a entrega das informações solicitadas, além de proporcionar um melhor desempenho dos serviços requisitados.

1.4 Organização da dissertação

Este trabalho está organizado da seguinte forma:

- O **Capítulo 2**: Esse capítulo apresentará uma revisão bibliográfica com os principais conceitos teóricos necessários para o entendimento e desenvolvimento deste trabalho. Serão abordados conceitos relacionados às áreas de transporte inteligente, sistemas distribuídos, mais especificamente *middleware*, e sistemas sensíveis ao contexto.

- O **Capítulo 3**: Nesse capítulo serão mostrados os trabalhos relacionados e será feito um estudo comparativo entre eles, para a partir deles, serem mostrados os diferenciais do trabalho em questão.

- O **Capítulo 4**: Esse capítulo irá conter uma visão abstrata do *middleware* a ser desenvolvido, bem como sua arquitetura e os detalhes de funcionamento dos serviços que foram desenvolvidos.

- O **Capítulo 5**: Nesse capítulo serão apresentadas todas as etapas realizadas durante o desenvolvimento do projeto, bem como todos os aspectos e detalhes de implementação do mesmo, como tecnologias utilizadas, por exemplo.

- O **Capítulo 6**: Esse capítulo apresentará algumas conclusões sobre o projeto, as limitações, contribuições e importância deste, bem como, o grau de sucesso do mesmo. Além disso, serão sugeridas pesquisas futuras a serem realizadas.

2

Fundamentos Teóricos

Neste capítulo serão abordados alguns conceitos importantes que serão necessários para uma melhor compreensão deste trabalho.

Inicialmente será dada uma introdução sobre o que seriam os Sistemas Inteligentes de Transporte (SIT), para que servem, como e onde surgiram. Será mostrado também uma breve distinção entre os diferentes tipos de SIT existentes na literatura e, dentre eles, será especificado o foco deste trabalho.

Após a introdução sobre os sistemas inteligentes de transporte iremos falar sobre sistemas distribuídos, sua importância e características. Mais especificamente será falado sobre *middleware* (camada intermediária de *software* que fica entre os sistemas distribuídos e suas plataformas) e seus requisitos.

Mais adiante será falado sobre contexto computacional, onde serão mostradas definições clássicas de contexto, características, o que poderiam ser consideradas informações e elementos contextuais. Além disso, também serão apresentados exemplos de sistemas sensíveis ao contexto.

Por fim, será falado sobre *middleware* sensíveis ao contexto para fechar o ciclo, uma vez que o objetivo deste trabalho é a especificação da arquitetura e implementação de um *middleware* sensível ao contexto para sistemas inteligentes de transporte público e urbano.

2.1 Sistemas Inteligentes de Transporte

Os sistemas de transporte vêm passando por uma rápida mudança nos últimos anos em decorrência do avanço tecnológico dos equipamentos eletrônicos e de comunicação (SILVA, 2000). A junção do uso de tecnologia da informação com os meios de transporte tornou-se uma área de estudo, conhecida mundialmente como SIT, do inglês ITS (*Intelligent Transportation Systems*), que visa a melhoria de aspectos relativos ao transporte.

É difícil afirmar quando e onde nasceu a ideia de utilizar tecnologia em sistemas de transporte. Os Estados Unidos utilizam equipamentos eletrônicos para controle do tráfego urbano desde a década de 60. O desenvolvimento de ITS iniciou na Europa na década de 70, na década de 80 surgiu o projeto PROMETHEUS (*Programme for European Traffic with Highest Efficiency*

and Unprecedented Safity, 1986) financiado pelo setor privado e o DRIVE I e II (*Dedicated Road Infrastructure for Vehicle Safity*, 1988), conduzido pelo setor público e tinha como objetivo atender toda união europeia (SILVA, 2000). O termo ITS parece ter surgido no final da década de 80 nos Estados Unidos quando um grupo passa a se reunir almejando uma nova visão para o sistema de transporte americano (CALDAS; VIEIRA, 2010).

Um ITS pode atuar em diferentes vertentes como, por exemplo, fornecer informações em tempo real a passageiros de transporte público ou mesmo motoristas. Além disso, um ITS pode proporcionar uma maior rapidez durante um processo de tomada de decisão em situações de emergência como, por exemplo, mudança de trajetos causadas por vias onde existem obras em execução ou devido ao acontecimento de acidentes (CALDAS; VIEIRA, 2010).

Apesar da similaridade dos problemas enfrentados nos mais diversos países, existem diferentes peculiaridades dependendo da região. Dessa forma, podem ser adotadas diversas abordagens e empregadas diferentes tecnologias na elaboração de um ITS para a resolução e tratamento de problemas semelhantes. Dentre os diversos tipos de tecnologias que podem ser utilizados, destacam-se: sensoriamento remoto, processamento e armazenamento de informações, comunicação entre diferentes tipos de plataformas e dispositivos e sistemas distribuídos (e.g. *middleware*). O objetivo do uso dessas tecnologias é prover informações de qualidade a passageiros e motoristas do transporte urbano, além de melhorar a segurança e eficiência do uso das vias, dessa forma estimulando o aumento na mobilidade, reduzindo o tempo de espera nas paradas, congestionamentos e impactos ambientais.

Como falado anteriormente, os sistemas de transporte inteligentes podem ser desenvolvidos utilizando várias tecnologias nos mais diversos setores da área de transporte, os quais podem ser categorizados como (SILVA, 2000):

- Sistemas avançados de transporte público (APTS, do inglês *Advanced Public Transportation Systems*): aplicável a processos que visam trazer benefícios aos usuários de transporte da rede pública como: redução do tempo de espera nas paradas por meio do provisionamento de informações precisas e atualizadas em tempo real sobre a localização e os itinerários dos ônibus e facilidade no pagamento das tarifas com a utilização de cartões eletrônicos, por exemplo.

- Sistemas avançados de gerenciamento de tráfego (ATMS, do inglês *Advanced Traffic Management Systems*): são responsáveis pelo gerenciamento do tráfego de forma global. Por exemplo, em virtude do congestionamento o sistema pode sugerir rotas alternativas ao motorista, podendo também ser aplicados em sistemas de sinalização (semáforos). O maior objetivo desse tipo de sistema é que o tráfego ocorra da forma mais eficiente possível.

- Sistemas avançados de informação ao viajante (ATIS, do inglês *Advanced Traveller Information Systems*): são responsáveis por prover os mais variados tipos de informações aos viajantes, em casa ou no veículo, informações sobre as vias, trânsito,

incidentes, condições meteorológicas, rota ótima, ou seja, informações relevantes que dão suporte a tomada de decisões com relação às opções disponíveis para sua viagem.

- Operação de veículos comerciais (CVO, do inglês *Commercial Vehicle Operations*): sistemas usados por veículos comerciais como, por exemplo, companhias de táxis, vans ou ônibus, visando um maior gerenciamento, eficiência e produtividade da frota.
- Sistemas avançados de controle veicular (AVCS, do inglês *Advanced Vehicle Control Systems*): são sistemas que oferecem um maior controle do veículo em determinadas situações, um bom exemplo desse tipo de sistemas são sistemas de alerta a colisão (sensores de ré).
- Sistemas avançados de transporte rural (ARTS, do inglês *Advanced Rural Transportation Systems*): existem vários desafios com relação aos tipos de sistemas que podem ser aplicados nessa área, existem dificuldade com relação à comunicação, perda de sinal, ou seja, é um setor que ainda requer alguns esforços em pesquisa e investimentos.

O foco de interesse deste trabalho serão os APTS, no que diz respeito à realidade brasileira, onde a necessidade de utilização de um transporte público é realidade na vida de muitos cidadãos, e contribui diretamente na melhoria da qualidade de vida da população.

Os APTS têm um conjunto de objetivos específicos que se distinguem dos demais ITS (SILVA, 2000):

- Aumentar o controle dos usuários sobre as viagens, aumentando a confiabilidade dos horários;
- Proporcionar alta qualidade dos serviços para que possa melhor competir com o setor privado;
- Contribuir para um sistema tarifário integrado;
- Aprimorar o sistema de informação do passageiro;
- Proporcionar maior segurança e qualidade de serviço ao passageiro;

Alcançar esses objetivos não é uma tarefa trivial, pois não existe investimento suficiente em tecnologia e infraestrutura, além disso, a cobrança aos donos de empresas de ônibus por parte do governo é muito baixa. Dessa forma, é necessário um comprometimento de todas as partes envolvidas (governo e empresas de ônibus) para que ocorra uma melhoria do serviço de transporte, além de uma reestruturação dos sistemas existentes para que informações relevantes possam ser providas de forma eficaz, eficiente e com qualidade para os usuários de transporte público.

2.2 Sistemas Distribuídos

A computação distribuída consiste em unir o poder de processamento de um conjunto de computadores interligados através de uma rede. Essa união de computadores com objetivo de compartilhar recursos para execução de tarefas é chamada de sistema distribuído. Segundo TANENBAUM; VAN STEEN (2002), um sistema distribuído é "uma coleção de computadores independentes que se apresentam ao usuário como um sistema único e coerente". Para ASTLEY; STURMAN; AGHA (2001), um sistema distribuído é composto por uma coleção de elementos computacionais autônomos que interagem entre si através da rede.

Devido ao grande número de dispositivos e diferentes tipos de plataformas cada vez mais se faz necessário a utilização de sistemas distribuídos. Eles podem ser utilizados de diversas formas diferentes, dentre elas estão: integrar sistemas legados, preservando o investimento já feito, proporcionar escalabilidade dos sistemas, facilitar a comunicação entre diferentes dispositivos, realizar processamento paralelo de informações, além de prover o compartilhamento de informações de maneira controlada e eficiente.

O grande entrave para a criação de um sistema realmente distribuído é a dificuldade e complexidade de desenvolvimento se comparado a sistemas locais. Essa complexidade se dá porque o sistema distribuído deve possuir algumas características como, por exemplo: paralelismo, concorrência de recursos (várias requisições são feitas por diferentes fontes), transparência (recursos distribuídos devem ser acessados como se fossem locais), tolerância a falhas e segurança, uma vez que os sistemas são distribuídos e a informação trafega pela rede é necessário o uso de mecanismos mais eficientes de segurança.

Os *middleware* são exemplos de sistemas distribuídos, seus conceitos e funcionalidades serão abordados com mais detalhes na seção a seguir.

2.3 *Middleware*

O *middleware* ou mediador é uma camada de *software* intermediária que se encontra entre as aplicações distribuídas e suas plataformas (Figura 2.1), sendo as plataformas compostas por processadores e suas arquiteturas, além de funções de baixo nível e APIs (*Application Programming Interface*) dos sistemas operacionais (SO) (BERNSTEIN, 1996). Uma definição clássica define um SO como sendo "o *software* que torna o hardware usável", analogamente a essa definição, um *middleware* pode ser considerado como o *software* que torna um sistema distribuído programável (BAKKEN, 2001). Este mesmo autor define um *middleware* como sendo um tipo de tecnologia de *software* projetado para ajudar a gerenciar a complexidade e heterogeneidade inerentes aos sistemas distribuídos.

A ideia relativa à construção ou utilização de um *middleware* para sistemas distribuídos é semelhante à ideia de utilização de um sistemas de gerenciamento de bancos de dados (SGBDs) para sistemas de informação. Ambos permitem aos engenheiros de *software* abstraírem detalhes

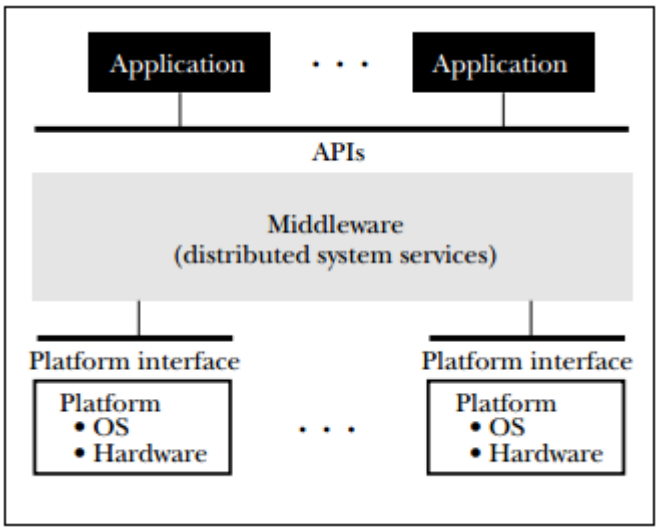

Figura 2.1: Visão do *middleware*. Extraído de (BERNSTEIN, 1996)

de implementação de baixo nível como controle de concorrência, gerenciamento de transações e comunicação na rede, permitindo assim um maior foco no desenvolvimento da aplicação (EMMERICH, 2000).

Alguns requisitos devem ser levados em consideração durante a elaboração de um *middleware*. Vale salientar, que alguns desses requisitos precisam ser revistos e adaptados caso o *middleware* seja usado em sistemas móveis.

- Comunicação: O *middleware* deve garantir a troca de informações entre os sistemas integrantes de forma transparente, dando a impressão ao usuário que o sistema é único e integrado (BRUNEO; PULIAFITO; SCARPA, 2007). Para isso, o *middleware* deve possuir a capacidade de interpretar e traduzir as informações passadas pelo componente solicitante e deve fornecê-las num formato que possa ser enviado por um protocolo de transporte como, por exemplo, uma sequência de bytes. Essa transformação de parâmetros se refere ao processo de *marshaling - unmarshaling* (EMMERICH, 2000).

- Coordenação: Em virtude do fato de componentes residirem em diferentes *hosts*, os sistemas distribuídos possuem vários pontos de acesso. Geralmente, várias *threads* são executadas concorrentemente em um *host* e todas devem ser sincronizadas com o solicitante de forma a não gerar um estado inconsistente (BRUNEO; PULIAFITO; SCARPA, 2007). A sincronização pode se dar de várias formas como, por exemplo, bloqueando um componente enquanto outro executa uma requisição.

- Confiabilidade: Os diferentes protocolos de transporte possuem diferentes graus de confiabilidade. Dependendo do protocolo, não há garantia que um pacote enviado será recebido, nem mesmo se a ordem de recebimento foi preservada. Dessa forma, para aumentar a confiabilidade é necessário fazer *trade-offs*, nesse caso do envio

de pacotes, a adoção de políticas para melhoria da qualidade do serviço como *best effort* impactam direta e negativamente na performance, entretanto, aumentam a confiabilidade. Outra forma de aumentar a confiabilidade seria utilizar técnicas de replicação de serviços em diferentes *hosts*. Assim, mesmo tendo um *host* indisponível por razões internas ou externas, o serviço continuaria disponível em outro *host*.

- Escalabilidade: É a capacidade de suportar o aumento de carga e requisições feitas pelos consumidores de serviços ao *middleware* no futuro. Em sistemas centralizados o limite da carga é definido pelo *server host*, porém nos sistemas distribuídos uma nova requisição pode ser enviada para diferentes servidores dependendo da necessidade, e isso deve ocorrer de forma transparente para o usuário. Esse balanceamento de carga é feito por mecanismos de *load-balancing*, cujo objetivo é prover redução ou aumento de carga, dependendo se um servidor for iniciado ou parado.

- Segurança: O *middleware* deve prover mecanismos de segurança como criptografia de dados, por exemplo, uma vez que dados privados trafegam na rede e podem ser interceptados por *softwares* mal intencionados.

2.4 ESB - *Enterprise Service Bus*

O conceito de ESB está diretamente ligado ao conceito de *middleware*. Um ESB é considerado um barramento que surgiu com a necessidade da integração de aplicações ponto a ponto, atividade que demonstrou ser bastante difícil de se executar, manter e gerir.

Integrações ponto a ponto resultam em uma comunicação customizada entre aplicações, causando o espalhamento de código entre os sistemas, gerando dependências e com frequência criando o chamado "código espaguete" (Figura 2.2). Dessa forma, um ESB é uma camada de abstração que combina características de uma arquitetura orientada a eventos (EDA - *Event-Driven Architecture*) com SOA (*Service Oriented Architecture*) (Figura 2.3) e tem como objetivo simplificar o processo de integração de unidades de negócio, estabelecendo uma ponte de comunicação entre plataformas e ambientes heterogêneos (MARÉCHAUX, 2006).

Segundo BAI et al. (2007), um ESB provê uma infraestrutura de serviços para troca e roteamento de mensagens em uma arquitetura orientada a serviços. Outra definição interessante segundo PAPAZAGLOU; HEUVEL (2003), um ESB é um barramento de mensagens, projetado para possibilitar a implementação, desenvolvimento e gerenciamento de soluções baseadas em SOA com o foco no empacotamento, desenvolvimento e gestão de serviços distribuídos.

Um ESB é o *backbone* ideal para implementação de um arquitetura orientada a serviços, porque ele provê um mecanismo universal para interconectar todos os serviços que compõem uma solução de negócio sem comprometer a segurança, confiabilidade, performance e escalabilidade (MENGE, 2007). Um ESB pode apresentar as seguintes características:

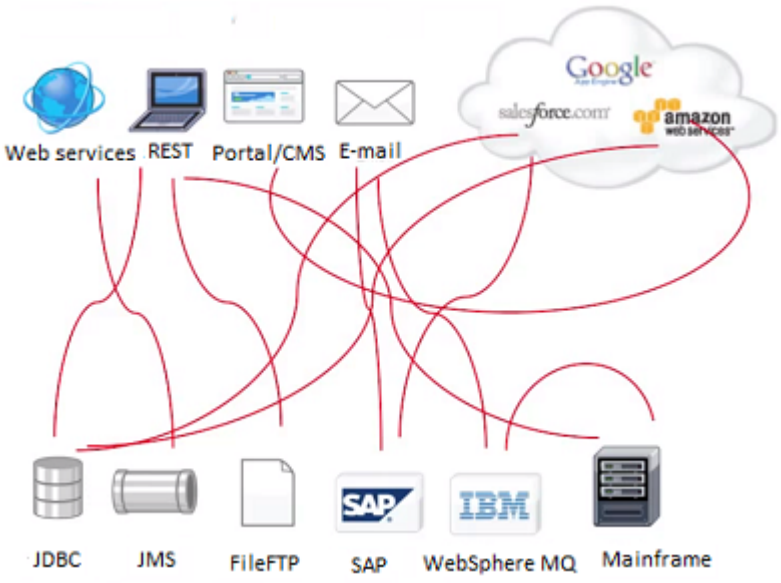

Figura 2.2: Integração de aplicações ponto a ponto. Extraído de http://www.mulesoft.org/

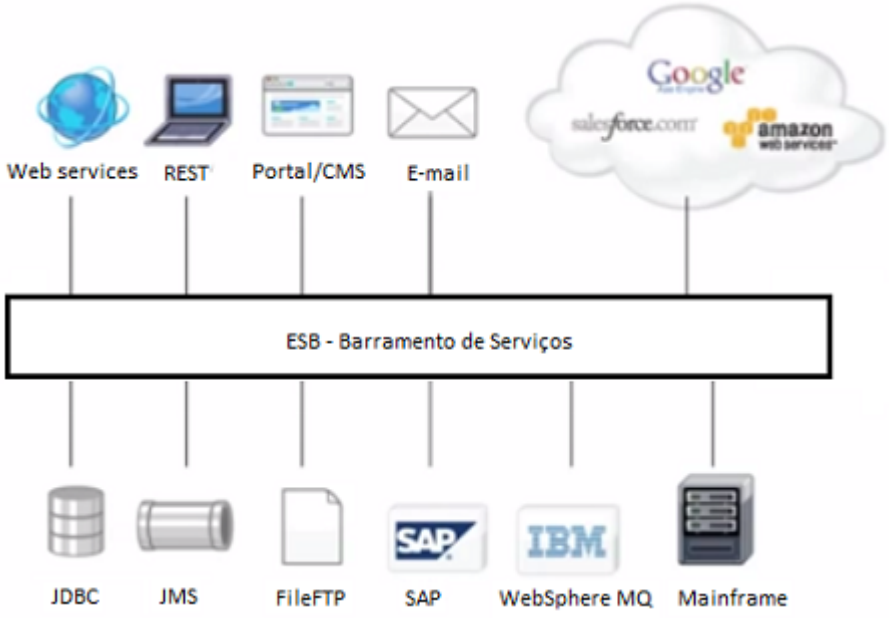

Figura 2.3: Integração de aplicações por meio da utilização de um ESB. Adaptado de http://www.mulesoft.org/

- Invocação: invocação é a habilidade do ESB enviar uma requisição e receber uma resposta para integração de serviços e recursos integrados. Isso significa que um ESB deve ser familiarizado e suportar padrões de comunicação web, além de suportar integração de aplicações usando MOM (*Message Oriented Middleware*) e outros protocolos de comunicação através da rede.

- Roteamento: essa habilidade decide o destino de uma mensagem durante o seu transporte. O roteamento é uma característica essencial de um ESB, pois ele permite o desacoplamento da fonte das mensagens e o seu destino. Essa característica foi levada bastante em consideração durante a concepção da arquitetura do UbiMid e será mais detalhada adiante.

- Mediação: refere-se a todas as traduções ou transformações sofridas entre os diferentes recursos, incluindo o protocolo de transporte, o formato e o conteúdo da mensagem. Essa transformação é bastante importante quando se fala em aplicações heterogêneas, pois as aplicações dificilmente concordam com o formato dos dados usados.

- Adaptadores: muitas soluções que utilizam um ESB são compostas por um conjunto de adaptadores, alguns prefabricados, o que reduz o esforço necessário para integrar aplicações utilizando uma arquitetura SOA. Os adaptadores estabelecem essa conexão por meio de APIs ou intefaces nativas que facilitam o reuso de código e lógica de negócio das aplicações. Eles podem ser usados para se conectar a um ERP (*Enterprise Resource Planning*), SCM (*Supply Chain Management*), ou a um CRM (*Customer Relationship Management*).

- Orquestração: vários componentes existentes são agregados para compor um único serviço composto em uma camada superior. Isso pode ser feito para alcançar a granularidade desejada, promover a reutilização e gerenciamento de componentes.

- Processamento de eventos complexos: uma mensagem assíncrona pode ser considerada um evento, especialmente quando é usada em um canal *publish/subscribe*. Assim, um ESB pode possuir ainda mecanismos de interpretação de eventos, correlação de eventos, além de eventos associados a padrões, permitindo, dessa forma, o desenvolvimento de uma arquitetura orientada a eventos.

É importante ter em mente os conceitos e características de ESB para melhor entendimento do processo de elaboração e concepção da arquitetura do *middleware* proposto, bem como do processo de implementação.

2.5 Contexto

O uso da palavra contexto na Web vem crescendo bastante nos últimos anos, passando de 5% em 1997 para 15% em 2004, segundo BAZIRE; BRÉZILLON (2005). Dessa forma, é possível encontrar várias definições para o significado da palavra contexto, algumas complementares, outras restritas à área de interesse. Existe uma definição clássica de contexto bastante referenciada na literatura dada por DEY (2001) onde ele diz que contexto é qualquer informação que possa ser usada para caracterizar a situação de uma entidade, esta pode ser uma pessoa, lugar ou objeto que seja considerado relevante para a interação entre o usuário e a aplicação, incluindo o próprio usuário e a aplicação.

Em (BAZIRE; BRÉZILLON, 2005) foi feito um estudo considerando um conjunto de definições de contexto (cerca de 150) encontradas na Web em diferentes domínios. Os autores chegaram à conclusão que a definição de contexto depende da área à qual pertence, não existindo assim uma definição absoluta, mas sim, definições que se adaptam às necessidades das áreas às quais são aplicas. Outra conclusão que os autores chegaram foi que o contexto atua como um conjunto de restrições que influenciam o comportamento de um sistema (um usuário ou um computador) envolvido numa data tarefa ou atividade.

Embora existam várias definições para contexto, este existe apenas quando está associado a alguma entidade (tarefa, agente ou interação). Segundo VIEIRA; TEDESCO; SALGADO (2009), na verdade, existem dois conceitos distintos: contexto e elemento contextual.

- Um elemento contextual é qualquer dado, informação ou conhecimento que permite caracterizar uma entidade em um dado domínio.
- O contexto é um conjunto de elementos contextuais instanciados que são necessários para apoiar a tarefa atual.

Por exemplo, a localização, é considerada uma informação relevante e, portanto, parte do contexto, ou seja, deve ser usada como um elemento contextual em um sistema de sugestão para almoço, um vez que o sistema deverá sugerir restaurantes próximos ao local onde o usuário se encontra.

Todo esse estudo relacionado a contexto vem sendo bastante utilizado recentemente, inclusive na área de sistemas distribuídos. Com o surgimento do contexto, tem-se buscado cada vez mais enriquecer a semântica dos serviços providos aos sistemas de forma a melhorar a interação usuário máquina ou mesmo diminuir o número de interações feitas pelo usuário, tornando o sistema mais agradável e eficiente.

Sistemas genéricos e pouco flexíveis que não levam em consideração o contexto dos usuários tendem a perder espaço. Estamos numa era onde a tendência é que os sistemas sejam cada vez mais fáceis de usar. Além de flexíveis e adaptáveis, eles devem levar em consideração as preferências dos usuários, suas necessidades, informações externas como: clima, horário do

dia, enfim, informações relevantes ao contexto ao qual os seus usuários estão inseridos. Porém, desenvolver aplicações sensíveis ao contexto não é uma atividade trivial, é importante identificar e compreender bem o contexto, sendo necessário saber exatamente qual o cenário desejado e quais informações servem para descrevê-lo.

2.6 Sistemas Sensíveis ao Contexto

O primeiro termo utilizado nessa vertente foi computação ciente de contexto (SCHILIT; THEIMER, 1994) para descrever um *software* que possui a capacidade de se adaptar a sua localização de uso, ao conjunto de pessoas e objetos próximos, bem como, às mudanças que ocorrem com estes ao passar do tempo (CALDAS; VIEIRA, 2010). Desde então, principalmente com o crescimento acelerado nos últimos anos da computação móvel, esta temática se tornou ainda mais popular. O termo em inglês é conhecido como *context-aware systems* (sistemas sensíveis ao contexto) tendo outros termos como sinônimo, como: sistemas baseados em contexto, aplicações adaptativas e aplicações dirigidas à resposta.

Aplicações tradicionais que não usam contexto, são sistemas computacionais que utilizam puramente como entrada informações e solicitações fornecidas explicitamente pelo usuário. Já aplicações sensíveis ao contexto (Figura 2.4), além de considerar as informações explícitas fornecidas pelo usuário, também levam em consideração informações históricas e implícitas, inferidas por meio de raciocínio, e observação do ambiente em que o usuário se encontra. Essas informações implícitas são armazenadas em bases de conhecimento contextuais, e são consideradas informações contextuais.

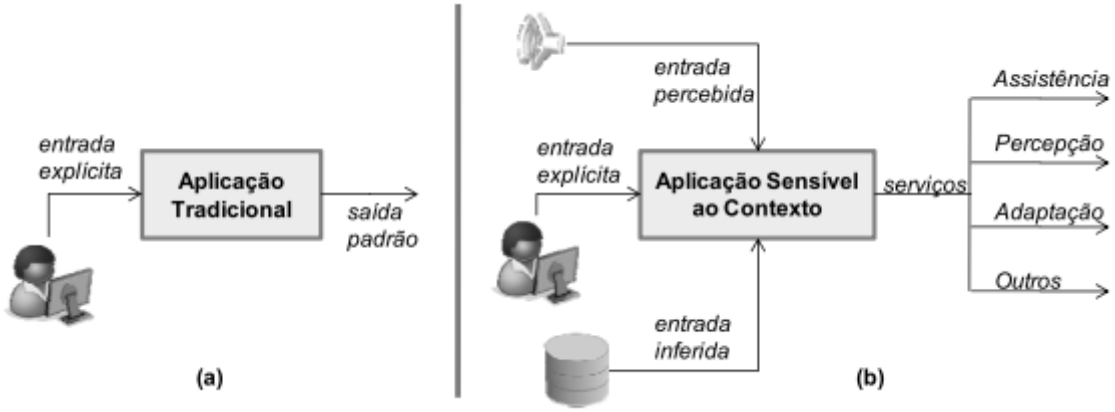

Figura 2.4: Visão geral de uma aplicação tradicional (a) e de uma aplicação sensível ao contexto (b) (VIEIRA; TEDESCO; SALGADO, 2009)

Segundo VIEIRA; TEDESCO; SALGADO (2009), com base nas informações contextuais, a aplicação pode enriquecer semanticamente a solicitação explícita do usuário e, com isso, executar serviços mais próximos às suas necessidades. Dentre estes serviços estão: (1) assistência na execução da tarefa sendo realizada como, por exemplo, alertar o usuário sobre ações que ele

deve executar para alcançar seus objetivos, ou recomendar recursos existentes relacionados à tarefa; (2) percepção do contexto, que se refere a notificar o usuário sobre o contexto associado a pessoas e interações do seu interesse, relativos à tarefa em execução, apoiando-o a coordenar suas próprias ações; (3) adaptação, ou variação do comportamento do sistema, respondendo de forma oportuna às mudanças ocorridas no ambiente e às ações e definições dos usuários (e.g. personalização de interfaces e conteúdo); e (4) outros serviços, como o uso do contexto para enriquecer semanticamente o conhecimento gerenciado pela aplicação.

Há várias aplicações que podem auxiliar o usuário na execução de uma tarefa exemplificando o serviço de assistência, dentre elas existe o MapLink Trânsito disponível para Android na Google Play [1] (Figura 2.5). Os usuários do MapLink podem consultar as condições do trânsito em tempo real para saber qual rota tomar para se deslocar do local de origem para o local de destino desejado. Para isso a aplicação fornece a situação do trânsito, além de prover informações das imagem das câmeras e notícias das vias (congestionamentos, acidentes) em tempo real.

Figura 2.5: MapLink - Exemplo de um serviço de assistência

A percepção é exemplificada na Figura 2.6. Quando se tenta acessar o Facebook[2] pelo *browser* por um celular, ele percebe que a requisição é feita por um dispositivo móvel e sugere ao usuário a mudança de contexto, utilizando o aplicativo do Facebook de forma a melhor auxiliar o usuário em suas tarefas e otimizando a navegação.

Outra ferramenta bastante utilizada atualmente que faz uso constante de contexto é o Gmail[3]. Ele utiliza informações contextuais extraídas através de palavras-chave presentes nos e-mails para recomendar *links* de propagandas que são adaptados de acordo com o e-mail que o usuário está lendo no momento (Figura 2.7).

Vários outros serviços podem ser propostos e desenvolvidos para integrar uma aplicação sensível ao contexto. Na área de transporte público, por exemplo, podem ser desenvolvidos sistemas de sugestão de rotas de acordo com a preferência do usuário e seu histórico de utilização.

[1] play.google.com - loja virtual mantida pela Google para distribuição de aplicações, jogos, filmes, música e livros para a plataforma Android.

[2] www.facebook.com.br - rede social gratuita com a maior quantidade de usuários da web.

[3] www.gmail.com - serviço de webmail gratuito fornecido pela google.

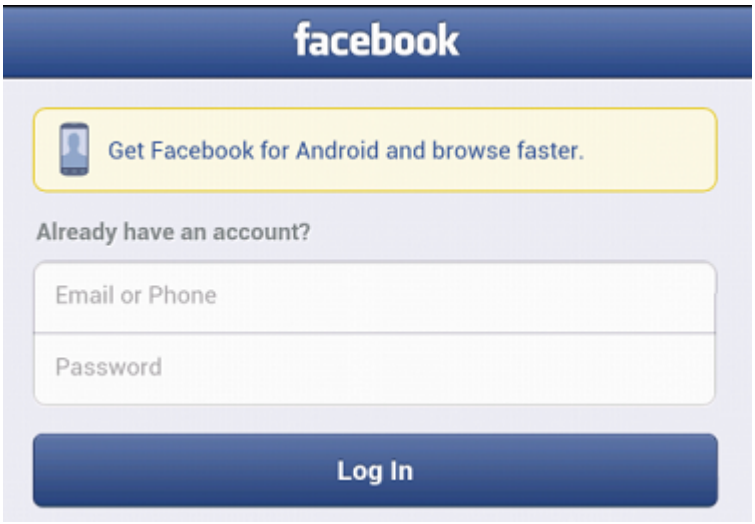

Figura 2.6: Facebook - Exemplo de um serviço de percepção no acesso a uma rede social

Figura 2.7: Gmail - Exemplo de um serviço de adaptação por meio de propagandas

Esse tipo de sistema auxiliaria o usuário na tomada de decisão sobre qual ônibus pegar baseado no trânsito, distância, tempo de espera, custo, sendo possível saber quanto tempo o ônibus demorará para chegar na parada e no destino desejado.

2.7 *Middleware* Sensível ao Contexto

O desenvolvimento de um *middleware* sensível ao contexto não é uma tarefa trivial. Além das questões relacionadas ao desenvolvimento de um *middleware* como transparência, heterogeneidade dos sistemas, segurança e comunicação, existem também várias questões relacionadas a modelagem do contexto de maneira flexível, eficiente e, acima de tudo, viável. Essas questões influenciam diretamente o desenvolvimento do *middleware*.

KJÆR (2007) fez um levantamento dos principais *middleware* sensíveis ao contexto e baseado nessa pesquisa os categorizou (Figura 2.8), levando em consideração suas características, da seguinte forma: ambiente, armazenamento, qualidade, composição, migração e adaptação.

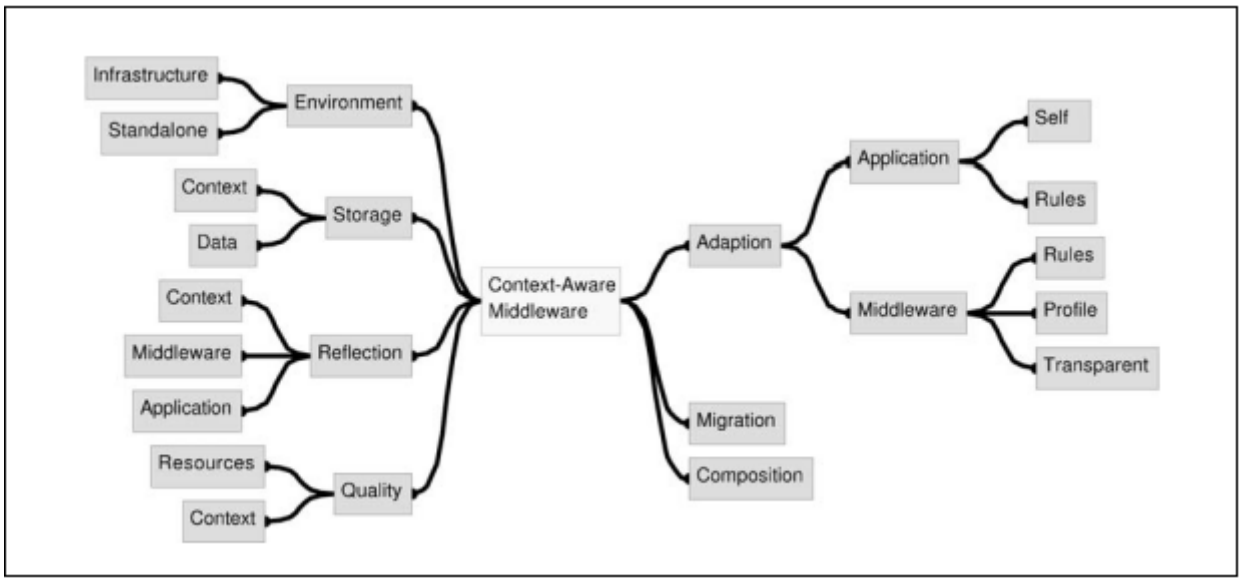

Figura 2.8: Categorização de *middleware* sensíveis ao contexto (KJÆR, 2007)

- Ambiente: o *middleware* sensível ao contexto assume de forma implícita ou explícita as condições do ambiente em que ele se encontra. Alguns *middleware* assumem a existência de uma infraestrutura que oferece os serviços que o *middleware* e a aplicação necessitam. Já outros sistemas assumem que os dispositivos possuem um método de comunicação e não dependem de serviços externos.

- Armazenamento: alguns sistemas proveem armazenamento de informações contextuais. Por exemplo, alguns sistemas utilizam sistema de arquivo para armazenar o contexto atual, outros utilizam sistemas de armazenamento centralizados, como banco de dados, para facilitar a recuperação de informação.

- Qualidade: está relacionada à performance do sistema ou quão boa é a informação provida pelo mesmo. Com relação a *middleware* sensível a contexto, a qualidade está mais relacionada à qualidade do serviço provido.

- Composição: alguns *middleware* fazem composições de componentes baseados em eventos contextuais. Assim, uma entidade do *middleware* pode ser formada pela junção de um conjunto de componentes ou pode mudar graças a ocorrência de um determinado evento, baseado no contextual. Por exemplo, um determinado serviço pode ser formado por um conjunto de outros serviços, onde este conjunto pode variar de acordo com a localização da requisição feita pela aplicação cliente.

- Migração: alguns sistemas proveem mecanismos de migração de código em execução dependendo da aplicação, possivelmente baseado no contexto, enquanto outros sistemas automaticamente migram entidades. O conceito de migração está relacionado ao conceito de adaptação a seguir.

- Adaptação: quando uma informação de contexto está disponível, tanto a aplicação como o *middleware* podem se adaptar à mudança do contexto. Se a adaptação ocorrer na aplicação, fica a cargo dos desenvolvedores gerenciarem as informações contextuais e seus impactos. Caso a adaptação ocorra no *middleware*, existem ações que ele pode tomar através de regras para sinalizar às aplicações das mudanças ocorridas. Essas adaptações podem ocorrer de forma transparante, ou seja, o *middleware* reagiria as mudanças sem que as aplicações clientes tomem conhecimento. Outra forma de adaptação seria a aplicação cliente solicitar um tipo de serviço que ela necessita ao *middleware* e ele se adaptar de forma a prover tal serviço da melhor maneira possível.

2.8 Considerações Finais

Neste capítulo apresentamos uma visão geral sobre sistemas inteligentes de transporte, além de conceitos e exemplos relacionados à área de sistemas distribuídos e *middleware* sensíveis ao contexto, que são a base para o entendimento deste trabalho. No próximo capítulo serão mostrados alguns dos *middleware* sensíveis ao contexto mais conhecidos para a partir de uma análise de suas vantagens e desvantagens, propor a arquitetura do Ubimid.

3

Middleware sensíveis ao contexto

Neste capítulo serão apresentados alguns dos principais *middleware* sensíveis ao contexto, bem como suas características e particularidades, mostrando o que já existe na literatura quanto aos trabalhos relacionados. Alguns dos *middleware* apresentados são bastante recentes como o ConProVa (SILVA et al., 2013), outros já mais consolidados como o WASP (COSTA, 2003) e o GAIA (RANGANATHAN; CAMPBELL, 2003).

Ao final do capítulo, serão apresentadas algumas considerações finais e serão mostradas as principais diferenças entre os *middleware* existentes e o *middleware* proposto neste trabalho.

3.1 SOCAM

O SOCAM (*Service-Oriented Context-Aware Middleware*) (GU; PUNG; ZHANG, 2005) é um *middleware* orientado a serviços sensível ao contexto, cuja arquitetura é composta por componentes independentes que atuam como serviços (Figura 3.1), são eles: serviços provedores de contexto, interpretador de contexto, banco de dados contextual, localizador de serviços e serviços sensíveis ao contexto.

Os provedores de contexto são serviços responsáveis pela aquisição das informações contextuais, que podem ser externas ou internas. Os provedores de contexto externos obtêm informações contextuais através de fontes de terceiros, como pode ser o caso da temperatura e condições climáticas, já o contexto interno ou local se refere às informações obtidas através de sensores localizados em um sub domínio específico como, por exemplo, informações obtidas através de sensores em um sistema de automação residencial. Ambas as informações são convertidas para OWL (*Web Ontology Language*) para compartilhamento e reuso.

O interpretador de contexto é o componente responsável pelo processamento das informações contextuais. Após o processamento, é possível se extrair informações contextuais de mais alto nível, através de um processo de dedução ou inferência a partir de informações contextuais isoladas. Dessa forma, o interpretador também atua como provedor de contexto. Ao fim do processo de inferência, o conhecimento obtido é armazenado em um banco de dados contextual ou disponibilizado ao serviço solicitante.

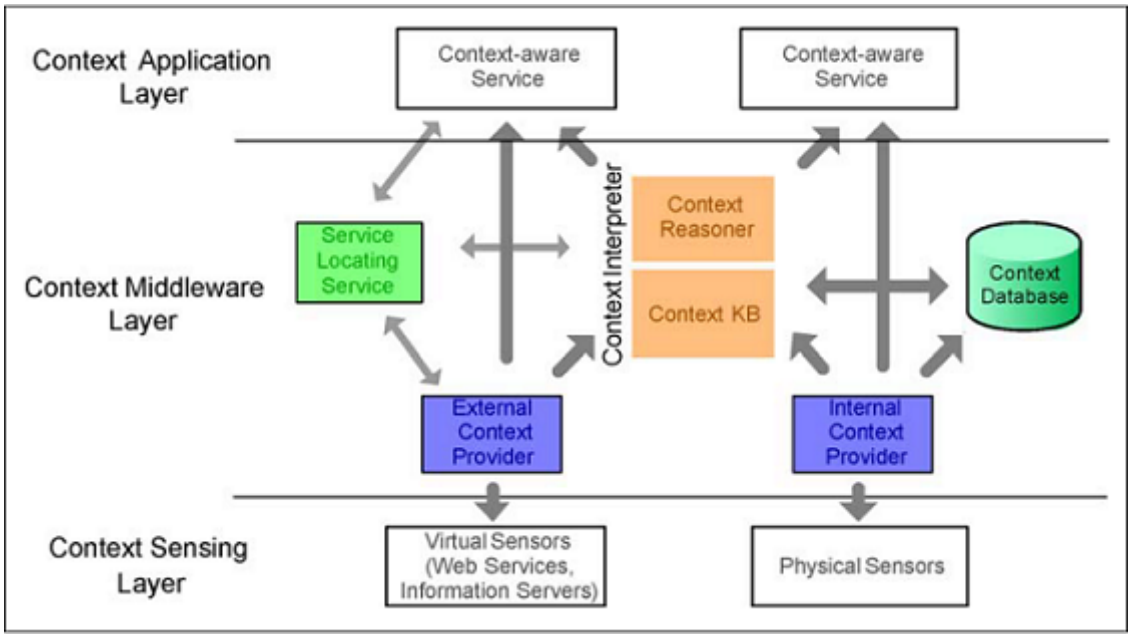

Figura 3.1: SOCAM - Visão geral da arquitetura. Extraído de (GU; PUNG; ZHANG, 2005)

O componente responsável pela localização de serviços permite ao usuário ou aplicação descobrir e localizar provedores e interpretadores de serviços. Caso uma fonte de contexto interna, por exemplo, sofra mudanças como adição ou remoção de sensores, o serviço de localização é capaz de rastrear e se adaptar a mudança dinamicamente.

Os serviços sensíveis ao contexto são agentes, aplicações ou serviços que adaptam seu comportamento de acordo com o contexto atual. Para isso, são especificadas ações e definidas regras que devem ser tomadas ou obedecidas caso determinado evento ocorra ou determinada condição seja satisfeita.

A arquitetura do SOCAM foi concebida para prover um suporte eficiente à construção de serviços sensíveis ao contexto em um ambiente de computação pervasiva através do uso de OWL para representar, manipular e acessar informações contextuais.

3.2 GAIA

Gaia é uma plataforma para espaços inteligentes, cujo objetivo é auxiliar os seres humanos na execução de tarefas dentro de espaços físicos como: salas, casas, prédios e aeroportos (RANGANATHAN; CAMPBELL, 2003).

O *middleware* presente no Gaia possui diferentes tipos de agentes que desempenham diferentes funções. Os tipos de agentes presentes na infraestrutura de contexto do GAIA estão representados na Figura 3.2 e serão descritos com mais detalhes a seguir.

- Provedores de contexto: são sensores ou qualquer outra fonte de dados com informações contextuais. Eles permitem que outros agentes consultem informações

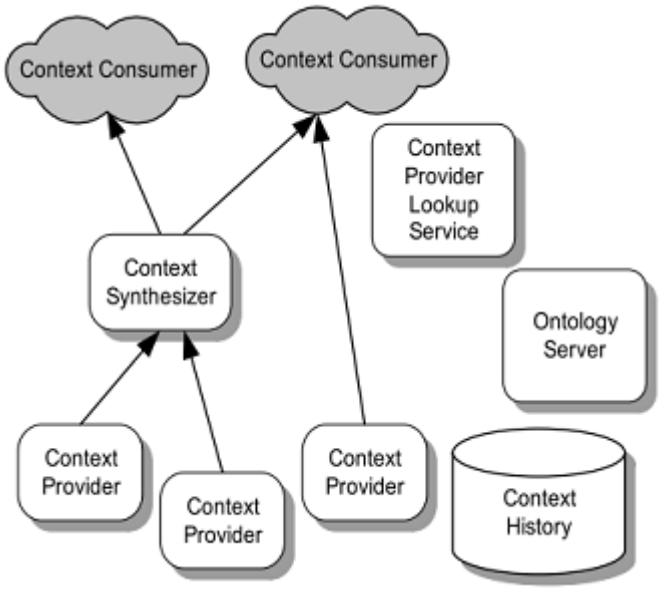

Figura 3.2: Gaia - Infraestrutura de contexto. Extraído de (RANGANATHAN; CAMPBELL, 2003)

contextuais, como é o caso dos consumidores de contexto. Além disso, alguns provedores de contexto ficam disparando eventos com informações contextuais para que outros agentes possam ouvir e fazer uso dessas informações de forma mais eficaz.

- Sintetizadores de contexto: eles são capazes de inferir informações de alto nível a partir de informações básicas adquiridas pelos provedores de contexto. Um sintetizador de contexto pode inferir que uma determinada atividade está ocorrendo em uma sala, por exemplo, baseado no número de pessoas e na aplicação que está sendo executada no momento.

- Consumidores de contexto: são agentes que extraem informações tanto dos provedores quanto dos sintetizadores de contexto. Dessa forma, eles podem raciocinar sobre o contexto atual e então se adaptar da melhor forma.

- Serviço de localização de provedores de contexto: esses serviços permitem aos agentes encontrarem provedores de contexto adequados.

- Serviço de histórico de contexto: contextos passados são armazenados numa base de dados, permitindo que sejam consultados por outros agentes.

- Servidor de ontologias: o servidor de ontologias mantém ontologias que descrevem diferentes tipos de informações contextuais. O uso de ontologias no Gaia, assegura que os diferentes agentes presentes no ambiente tenham o mesmo entendimento semântico dos diferentes tipos de informações contextuais, permitindo também uma melhor interoperabilidade entre diferentes tipos de agentes.

No Gaia, o contexto é baseado num modelo de predicados estruturados. Os predicados são formas de representar contexto sem necessariamente estar ligado a uma linguagem de programação. O modelo de predicados permite aos agentes desenvolverem mecanismos de inferência usando regras ou aprendizagem de máquina, por exemplo, para decidir qual atitude tomar baseado no contexto corrente. Um predicado pode ser representado da seguinte forma: *Tipo de contexto (Informações contextuais)*. Por exemplo: luz(sala, 50%), sala(A706, 20 pessoas), localização(Gilson, indo para, 52021-180).

Existem inferências de natureza mais concreta, como é o caso da luz e da sala, onde a probabilidade de acerto é maior. Por outro lado, existem os predicados incertos, como é o caso da localização, que possui uma característica mais abstrata e uma natureza incerta. No caso, Gilson pode estar indo para o CEP 52021-180 para casa ou para o dentista uma vez que ambos possuem o mesmo CEP.

Dentre outras aplicações, o GAIA foi utilizado em uma aplicação de gerenciamento de apresentações, onde a apresentação era mostrada em várias telas simultaneamente e poderia ser controlada por diferentes tipos de dispositivos (ROMÁN et al., 2002).

3.3 ConProVA

O ConProVA (*Context Provisioning for Vehicular Applications*) é um *middleware* para o provisionamento de contexto para aplicações de redes veiculares e que tem como características a modularidade, inferência de modelos lógicos, tratamento de conflitos de interesse e compartilhamento de informações entre os veículos (SILVA et al., 2013).

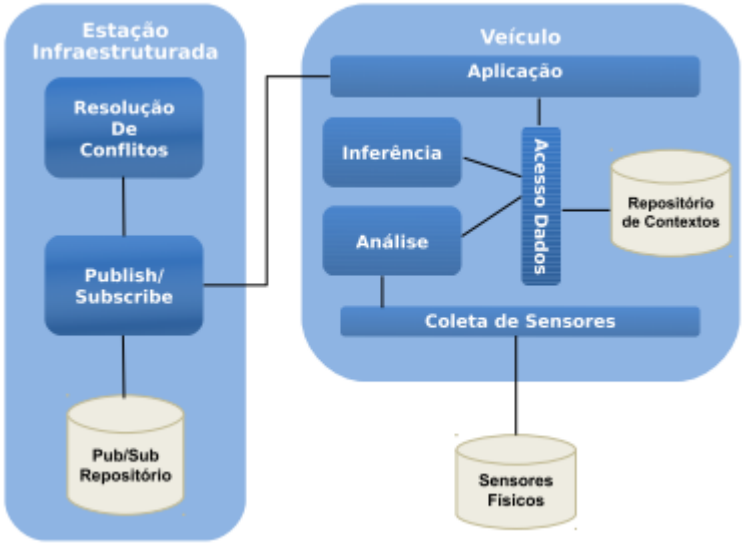

Figura 3.3: Arquitetura de componentes do ConProVA. Extraído de (SILVA et al., 2013)

O ConProVA é baseado numa arquitetura cliente/servidor (Figura 3.3). Como a maioria dos *middleware*, ele possui um componente responsável pela coleta de informações dos sensores

e outro componente para a análise e agregação dos dados relacionados. Além dos componentes já citados, o *middleware* em questão possui uma camada de acesso aos dados contextuais responsável por controlar o acesso aos dados do repositório de contexto.

O componente de inferência utiliza a técnica de aprendizagem chamada de *Naive Bayes* (HAN; KAMBER; PEI, 2006), baseada no teorema probabilístico de Bayes, justamente por ser uma técnica bastante simples, na maioria dos casos, e onde os dados contextuais considerados contribuem de forma independente no resultado do algoritmo. Similarmente ao componente de inferência de contexto lógico, existe um componente para resolução de conflitos que utiliza uma função de utilidade, onde o valor da utilidade é calculado para todos os veículos envolvidos no conflito e, ao final, são selecionados os veículos com a maior utilidade correspondente ao número de recursos disponíveis. O componente de inferência utiliza a técnica de aprendizagem chamada de *Naive Bayes* (HAN; KAMBER; PEI, 2006), baseada no teorema probabilístico de Bayes, justamente por ser uma técnica bastante simples e onde os dados contextuais considerados contribuem de forma independente no resultado do algoritmo.

O componente de *publish/subscribe* é responsável pelo compartilhamento de informações entre os veículos. Quando um veículo tem interesse em algum tipo de informação ele faz uma subscrição ao servidor. Caso exista alguma informação relevante para o veículo em questão, o mesmo é notificado, caso contrário ele deve ficar enviando mensagens de subscrição atualizadas periodicamente para confirmar seu interesse.

A publicação ocorre quando um evento acontece. Dessa forma, o veículo envia uma mensagem de publicação com informações contendo sua localização, o contexto que originou o evento, o tempo de expiração do evento, dentre outras. Ao chegar ao servidor, a mensagem é compartilhada e caso haja alguma subscrição relacionada ao evento lançado, o veículo que fez a subscrição recebe uma notificação. Esse mecanismo de publicação e subscrição é bastante importante em uma rede veicular, pois facilita a tomada de decisão através do consumo de informações providas por outros veículos.

3.4 WASP

O WASP, do inglês *Web Applications Service Platform* (COSTA, 2003), é uma plataforma baseada na tecnologia de *Web Services* que dá suporte ao desenvolvimento de aplicações sensíveis ao contexto. A plataforma pode se relacionar com o mundo exterior de três formas distintas.

Os provedores de contexto são os atores responsáveis por adquirir informações contextuais, estas podem ser obtidas através de sensores ou provedores de contexto de terceiros. Os provedores de serviços são entidades responsáveis pela disponibilização de serviços junto à plataforma. Por fim, os clientes das aplicações WASP interagem com a plataforma através dos serviços e, dessa forma, podem se adaptar ao contexto, caso o mesmo mude. Os provedores de contexto são os atores responsáveis por adquirir informações contextuais, estas podem ser obtidas através de sensores ou provedores de contexto de terceiros. Os provedores de serviços

são entidades responsáveis pela disponibilização de serviços junto à plataforma. Por fim, os clientes das aplicações WASP interagem com a plataforma através dos serviços e, dessa forma, podem se adaptar ao contexto, caso o mesmo mude.

Expandindo o componente da plataforma WASP (Figura 3.4) é possível ter uma visão da sua arquitetura, que é composta por três componentes principais: o interpretador de contexto, os repositórios e o monitor, cada um deles com funções específicas.

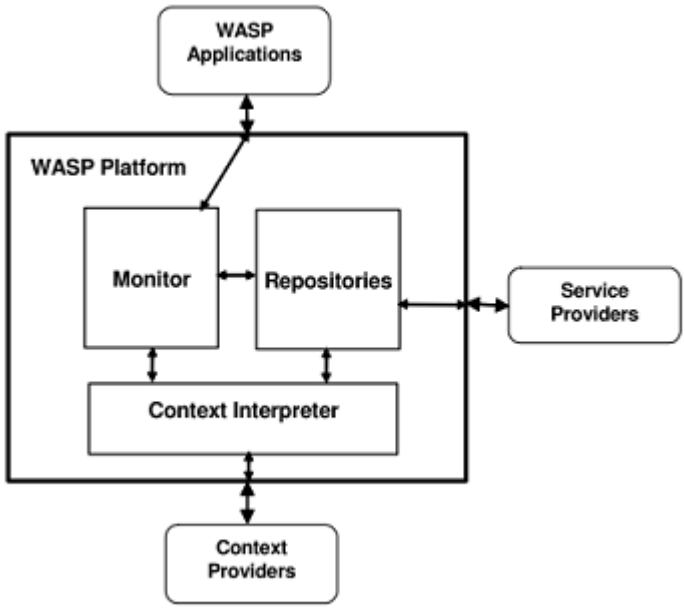

Figura 3.4: Arquitetura de componentes da plataforma WASP. Extraído de (COSTA, 2003)

O interpretador de contexto recebe as informações capturadas pelos provedores de contexto e as manipula, tornando-as disponíveis de forma mais uniforme para os demais componentes da plataforma. Essa camada é bastante importante, uma vez que o WASP faz uso de provedores de contexto externos e é necessário essa uniformização das informações contextuais.

Os repositórios são componentes responsáveis por armazenar e manter as informações providas pelos interpretadores de contexto e informações relativas aos provedores de serviços. O monitor é responsável por interpretar e gerenciar as assinaturas e os pedidos feitos pelas aplicações clientes à plataforma e é considerado o coração da plataforma.

Posteriormente, o modelo da Figura 3.4 foi estendido com o objetivo de modelar e manipular as informações contextuais usando ontologias. Dessa forma o WASP pode prover uma anotação semântica padronizada dos termos utilizados pelos componentes e demais atores da plataforma, além de definir uma descrição uniforme de serviços e contexto (PEREIRA FILHO et al., 2006).

3.5 Infraware

O Infraware (Figura 3.5) é um *middleware* baseado em *Web Services* que busca prover suporte ao desenvolvimento, construção e execução de aplicações móveis sensíveis ao contexto. Sua arquitetura foi estendida, em vários aspectos, a partir da plataforma WASP, um projeto desenvolvido pela *University of Twente, Telematica Instituut e Ericsson* (PEREIRA FILHO et al., 2006).

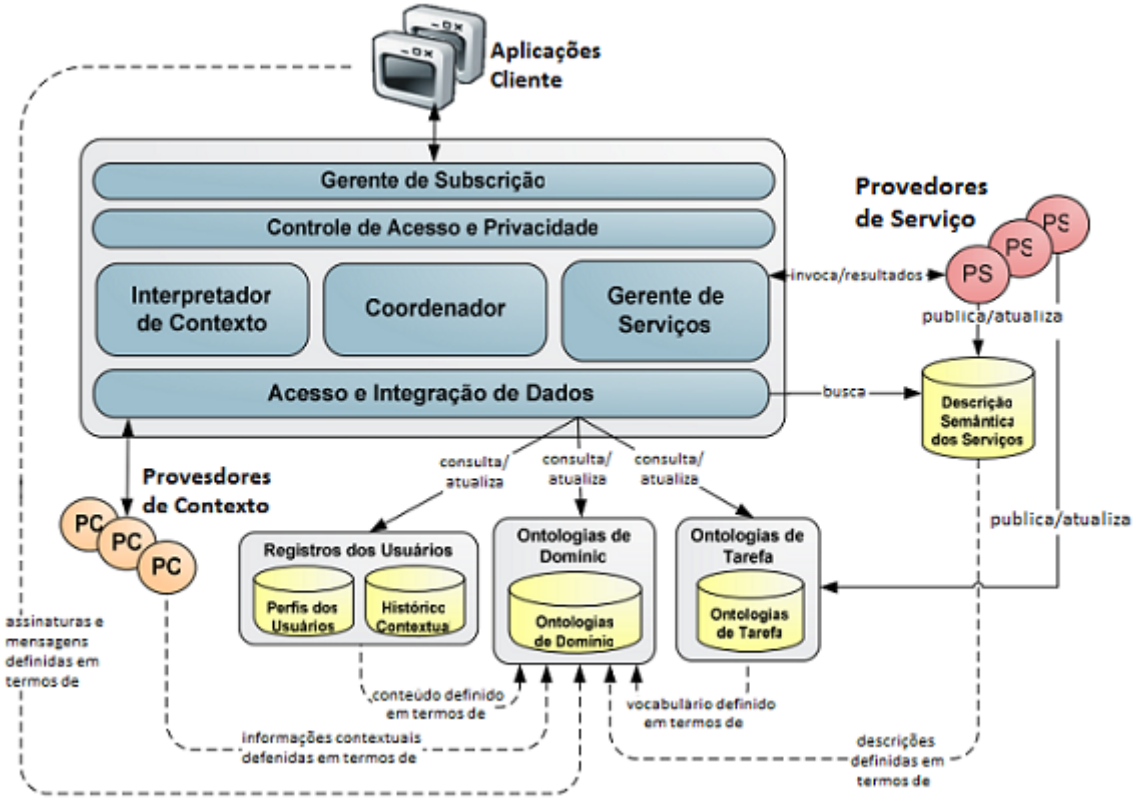

Figura 3.5: Arquitetura da plataforma Infraware. Adaptado de (PEREIRA FILHO et al., 2006)

O Infraware tem como objetivo prover suporte arquitetural ao desenvolvimento de aplicações sensíveis ao contexto, fornecendo serviços, mecanismos e interfaces que abstraiam a complexidade da aquisição, manipulação e uso das informações contextuais por parte das aplicações (PEREIRA FILHO et al., 2006). Exemplificando, atualmente, ele tem sido usado como base em aplicações móveis na área de saúde, mais especificamente, em sistemas de telemonitoramento de pacientes com problemas cardíacos crônicos (PEREIRA FILHO et al., 2006). E, assim como outras plataformas, o Infraware também faz uso de ontologias. Elas são utilizadas para especificar modelos formais extensíveis para o domínio da aplicação, bem como para seus serviços. Os principais componentes da plataforma serão descritos com mais detalhes a seguir.

O gerente de subscrição é o componente responsável por fornecer uma interface de co-

municação que permite às aplicações externas se comunicarem com a plataforma e, dessa forma, conseguir adicionar, remover ou atualizar pedidos de subscrições. O controle de privacidade atua como um filtro sobre os dados que trafegam entre a plataforma e as aplicações com base em políticas de privacidade e preferências do usuário e da aplicação, além de estabelecer limites de visibilidade com relação aos dados coletados.

O interpretador de contexto é responsável pela manipulação e inferência de contexto mais elaborado a partir de informações contextuais primitivas. Este componente faz uso de ontologias para manipulação e inferência de novas informações, garantindo uma representação compartilhada e reutilizável dos dados (PEREIRA FILHO et al., 2006).

O componente de acesso e integração de dados estende o CoDIMS (*COnfigurable Data Integration Middleware System*) (BARBOSA, 2001), um *middleware* usado para integração de dados baseado em *frameworks* e componentes. Dessa forma, é possível manipular e tratar informações oriundas de diversas fontes de contexto.

O gerente de serviços é responsável por publicar, descobrir e selecionar os serviços que serão disponibilizados. Por fim, existe um componente chamado coordenador, responsável pelo gerenciamento do *middleware*. Suas principais funções estão relacionadas ao monitoramento e controle do estado geral da plataforma.

O Infraware foi projetado para o tratamento de questões relacionadas à manipulação de informações contextuais, diferentemente das aplicações tradicionais. A plataforma apresenta definições de contextos genéricos, aquisição de dados heterogêneos e um conjunto de outras características que dão suporte ao desenvolvimento de aplicações móveis ubíquas sensíveis ao contexto.

3.6 Aura

Aura (*Archictetural Framework for User Mobility in Ubiquitous Computing Environment*) (SOUSA; GARLAN, 2002) é uma arquitetura de *framework* usada para o desenvolvimento de aplicações ubíquas criada pela Universidade de Carnegie Mellon. Seu objetivo é maximizar o uso de recursos disponíveis no ambiente, assim como evitar que o usuário se distraia durante a execução de uma tarefa (SOUSA; GARLAN, 2002). O *framework* também provê infra-estrutura para permitir aplicações se moverem facilmente de um dispositivo para o outro, suportando de maneira robusta a heterogeneidade e variabilidade dinâmica de recursos do ambiente (PEREIRA FILHO et al., 2006). No momento em que o usuário se move de um ambiente para outro, o Aura procura adaptar as tarefas que estão sendo executadas.

A Figura 3.6 mostra uma visão panorâmica da arquitetura do *framework* composta por quatro componentes: o gerenciador de tarefas, chamado de *Prism*, o observador de contexto (*Context Observer*), o gerenciador de ambiente (*Environment Manager*) e, por fim, os fornecedores de serviços (*Service suppliers*).

Cada um dos componentes serão descritos com mais detalhes a seguir:

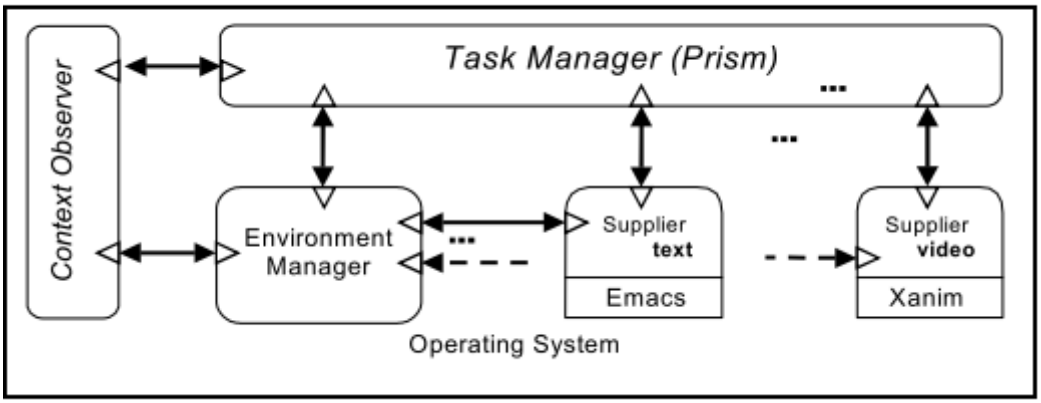

Figura 3.6: Visão panorâmica da arquitetura do *framework* Aura. Extraído de (SOUSA; GARLAN, 2002)

- *Task Manager (Prism)*: este componente é responsável por coordenar a migração das informações relacionadas às tarefas durante as mudanças de ambiente, bem como monitorar a qualidade das informações providas pelos *Suppliers* e gerenciar a execução das tarefas.
- *Suppliers*: proveem serviços abstratos para uma determinada tarefa como, por exemplo: edição de texto, reprodução de video, entre outros. Na prática, esses serviços são providos através do encapsulamento de aplicações e outros serviços já existentes, de forma que eles entrem em conformidade com a API do Aura para que possam ser utilizados.
- *Context Observer*: responsável por prover informações contextuais, além de reportar a ocorrência de eventos para os demais componentes.
- *Environment Variable*: gerencia os serviços disponíveis no ambiente, bem como a descoberta de serviços, sendo responsável por avaliar os serviços e escolher o que melhor se adéqua às preferências do usuário.

O Aura atua como um *proxy* para o usuário, assim que ele entra num ambiente novo, o *proxy* identifica os recursos disponíveis mais apropriados para a realização de uma determinada tarefa. Quando os requisitos da tarefa não estão sendo contemplados, o Aura permite a busca por configurações alternativas para que a atividade em execução seja concluída, sendo dessa forma, bastante útil.

3.7 CASS

CASS (*Context-Awareness Sub-Structure*) é um *middleware* baseado em servidor, cujo objetivo é dar um melhor suporte a aplicações móveis sensíveis ao contexto. Segundo FAHY;

CLARKE (2004), um *middleware* deve ser suficientemente flexível para prover informações e serviços fazendo uso das informações contextuais disponíveis. Para isso, o CASS tem como características suportar a aquisição de informações contextuais de uma grande quantidade de sensores, dentre outras fontes de dados distribuídas, de forma transparente.

A Figura 3.7 mostra a arquitetura do *middleware*. Os nós de sensores são responsáveis por fornecerem informações contextuais relevantes. Devido ao *middleware* ser baseado em servidor, ele não sofre com problemas relativos a limitações de processamento (permitindo o uso de componentes com Inteligência Artificial) e memória, além de permitir uma maior facilidade quanto ao armazenamento de uma maior quantidade de dados. Entretanto, existe uma dependência de comunicação entre as aplicações e o servidor do *middleware*. Para diminuir essa dependência o CASS suporta a utilização de *cache* local.

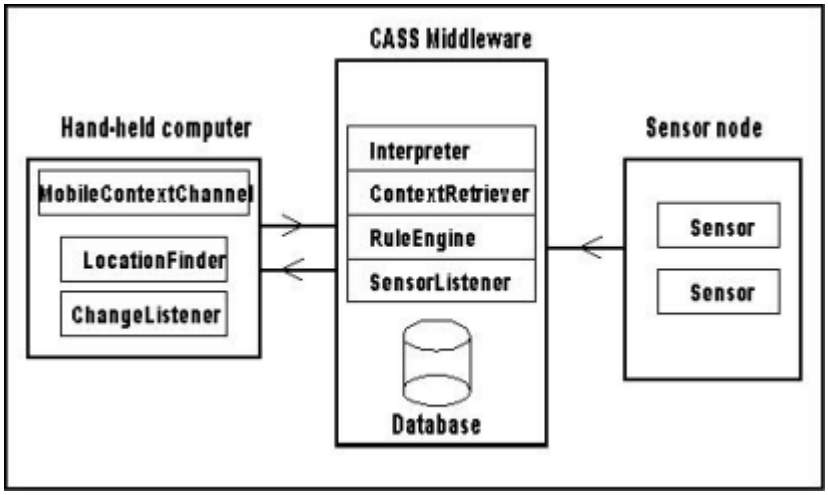

Figura 3.7: Arquitetura do CASS. Extraído de (FAHY; CLARKE, 2004)

No CASS, o motor de inferência trabalha junto à base de conhecimento, respeitando um grupo de regras para resolver os problemas. Por exemplo, dada uma situação, que ocorreu devido a um conjunto de condições e parâmetros, a base de conhecimento é consultada em busca de de situações semelhantes, ou seja, que ocorreram em condições semelhantes, com o objetivo de encontrar e inferir um solução para o problema baseado em situações parecidas que já ocorreram. A técnica de inferência utilizada utiliza encadeamento progressivo (*forward chaining*), onde os fatos já conhecidos são usados para inferir novos fatos. Essa técnica de busca é bastante útil para situações onde o espaço de busca possui várias saídas possíveis, o que é o caso de um sistema sensível ao contexto.

A inferência de contexto ocorre de forma independente do código da aplicação, tornando desnecessária mudança no código e a recompilação, permitindo dessa forma que os usuários fiquem menos dependentes dos programadores.

3.8 Características analisadas

Para análise dos *middleware* pesquisados, este trabalho selecionou algumas características (Tabela 3.1) consideradas relevantes no âmbito deste projeto e fez um levantamento, dentre os *middleware*, quais apresentam tais características.

Tabela 3.1: Comparação entre os trabalhos relacionados

Middlewares	Garantia de entrega e qualidade de serviço	Utiliza mecanismo de *time-driven* ou *event-driven*	Baseado em *web services*	Utilização de informações históricas
Infraware	Não	Sim	Sim	Sim
ConProVa	Não	Sim	Não	Não
GAIA	Não	Sim	Não	Sim
WASP	Não	Sim	Sim	Sim
Aura	Não	Sim	Não	Não
CASS	Não	Não	Não	Sim
SOCAM	Não	Não	Não	Sim

É importante a arquitetura de um *middleware* preveja algum mecanismo de entrega de serviços de forma assíncrona, principalmente para serviços que exigem um maior processamento, independentemente do seu tipo de assincronicidade, podendo ser tanto baseada em eventos, quanto baseada em tempo. Dessa forma, as aplicações clientes não precisam ficar esperando uma resposta para continuidade do seu processamento ou mesmo ficar fazendo a mesma solicitação várias vezes, simplesmente quando algo ocorrer, ou mesmo periodicamente, ela poderá ser notificada sobre mudanças ou a ocorrência de algum evento.

A garantia de entrega de serviço é importante, porque ela tem como objetivo garantir que o serviço uma vez requisitado seja entregue de alguma forma, podendo ser pela redundância de serviços, caso um serviço esteja indisponível outro serviço assume seu lugar ou seja através de mecanismos de confirmação de recebimento de resposta, caso a confirmação não seja recebida o serviço pode reenviar a resposta ao cliente, por exemplo. No que diz respeito aos *middleware* pesquisados, não é bem especificado o que acontece quando se necessita de informações de serviços externos e os mesmos não estão disponíveis. Nesse caso, o serviço que necessita da informação pára de funcionar? Existe algum mecanismo de garantia de entrega dos serviços? Essas questões não são abordadas ou explicitadas durante as especificações das arquiteturas relacionadas.

Web Services é uma tecnologia bastante madura, difundida e utilizada, características que facilitam no seu desenvolvimento e na sua utilização. O WASP e o Infraware, fazem uso de *Web Services* cujo objetivo é proporcionar a interação entre aplicações e serviços, além de facilitar e promover o compartilhamento e reuso de serviços e informações contextuais.

O uso de informações históricas é considerado relevante para inferência de informações

contextuais de qualidade, pois além de agregar mais informações que podem ser relevantes ao contexto atual do usuário, permite uma análise a posteriori dessas informações para extração de conhecimento.

3.9 Considerações finais

Nas seções anteriores foram mostrados alguns dos *middleware* sensíveis ao contexto mais conhecidos e suas respectivas arquiteturas. Todos eles estão relacionados à aplicações ubíquas e pervasivas, áreas onde o contexto vem ganhando bastante destaque nos últimos anos.

Muitas das arquiteturas citadas apresentam características semelhantes como, a presença de provedores de contexto que fazem aquisição de informações contextuais de fontes externas e internas, além de analisadores ou interpretadores de contexto, cujo objetivo é fazer inferências a partir das informações contextuais adquiridas dinamicamente ou através de informações históricas armazenadas em um banco de dados contextual. Porém, cada *middleware* possui aspectos específicos como, serviços próprios, diferentes tipos de modelagem de contexto, diferentes formas de se comunicar. O principal objetivo é atender os requisitos de negócio para os quais foi concebido. A plataforma mais apropriada é aquela que melhor se adéqua às exigências da aplicação para qual será usada.

Um fator importante para a escolha da arquitetura é a identificação da abordagem de desenvolvimento que será utilizada. A maioria dos *middleware* como, por exemplo, SOCAM e Aura, faz uso de um abordagem de desenvolvimento de software chamada *top-down*, onde os requisitos são elicitados, descritos e detalhados repetidamente, até que estejam refinados o suficiente para que possa começar a etapa de desenvolvimento. A abordagem *bottom-up* funciona de forma inversa, a partir de componentes existentes são extraídos os requisitos e são detalhadas as funcionalidades do sistema. Algumas das arquiteturas pesquisadas, como o Infraware, são tão robustas e complexas que dificultam sua adaptação, tornando difícil sua utilização em projetos já iniciados.

A próxima seção tem como objetivo apresentar a arquitetura do *middleware* proposto, mostrando as principais funções dos seus componentes e o objetivo para o qual eles foram propostos. Ao final, serão apresentados os serviços sensíveis ao contexto que foram trabalhados no UbiMid, bem como a sua integração e interação com os componentes da arquitetura.

É importante salientar que o UbiMid utiliza uma abordagem de desenvolvimento híbrida, tanto *top-down*, quanto *bottom-up*, pois além de terem sido elicitados novos requisitos, como já existiam projetos sendo desenvolvidos em paralelo, um dos objetivos deste trabalho também é fazer a extração dos serviços já existentes para que os mesmos também possam ser compartilhados e reusados por outras aplicações. Essas característica serão mostradas no capítulo seguinte.

4

UbiMid: um *middleware* sensível ao contexto voltado para aplicações na área de sistemas de transporte público e urbano

A construção de novas aplicações e serviços é uma atividade que pode ser de natureza bastante complexa, uma vez que pode envolver a utilização de uma grande diversidade de outros serviços. Dessa forma, é necessário ter em mente quais são os potenciais problemas que podem ocorrer durante a criação dessas aplicações ou serviços como, indisponibilidade dos serviços utilizados, complexidade de utilização dos serviços no que diz respeito a necessidade de aquisição de novas informações que são necessárias para execução do serviço, bem como divergência e complexidade de transformação do formato das informações utilizadas pela aplicação e pelo serviço.

O *middleware* proposto, chamado UbiMid, foi desenvolvido para prover qualidade de serviço e facilitar a comunicação e a coordenação de componentes de softwares distribuídos através de um barramento de serviços (Figura 4.1), cujo objetivo é prover modularidade, desacoplamento de sistemas, reuso de código (evitando retrabalho), inferência de contexto, aquisição, armazenamento, enriquecimento, transformação e compartilhamento de informações contextuais entre os diferentes tipos de aplicações e sistemas na área de sistemas inteligentes de transporte.

Tendo em mente esses requisitos, este capítulo aborda os conceitos pertinentes à arquitetura do *middleware* proposto.

4.1 Visão geral da arquitetura do UbiMid

A arquitetura apresentada neste capítulo (Figura 4.2) busca prover suporte à disponibilização de serviços sensíveis ao contexto, comunicação e integração entre aplicações de diferentes plataformas através do uso de um ESB, cuja definição e características já foram citadas anteriormente.

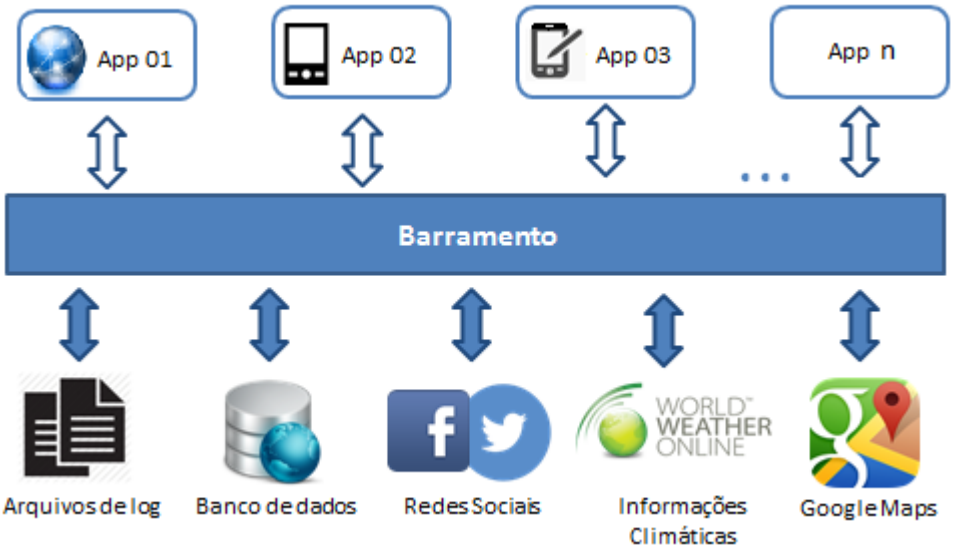

Figura 4.1: Abstração da arquitetura do UbiMid provida através de um barramento de serviços.

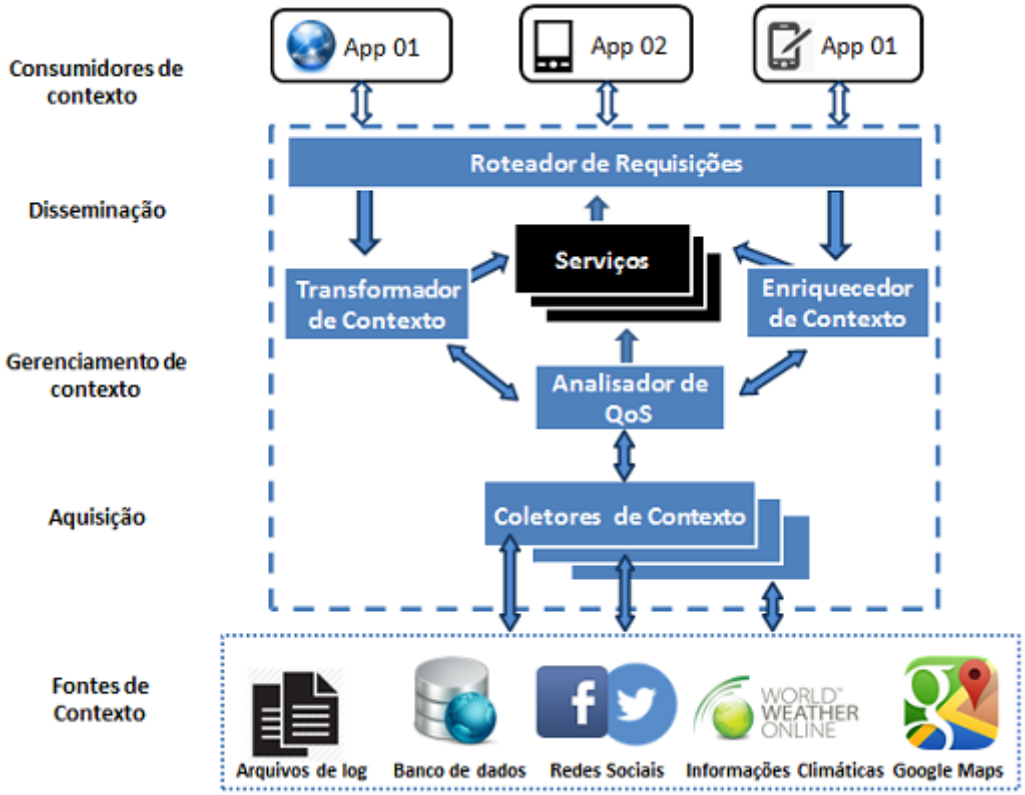

Figura 4.2: Arquitetura do *middleware* proposto.

Tendo em mente o conceito de barramento de serviços, a arquitetura do UbiMid tem como um de seus principais objetivos proporcionar um maior desacoplamento entre os sistemas, também conhecido como *loose coupling*. Assim, a comunicação pode ser feita com baixo grau de dependência entre os sistemas e de forma transparente. Essa característica será bastante evidenciada pelo uso dos componentes de roteamento, enriquecimento e transformação de contexto, que serão vistos com mais detalhes nas próximas seções. Além disso, o UbiMid também apresenta um serviço de mensageria e serviços sensíveis ao contexto disponibilizados por meio de uma arquitetura SOA.

Os componentes do UbiMid são descritos a seguir:

- Roteador de requisições: a utilização de roteadores busca proporcionar um maior desacoplamento entre a mensagem fonte e o seu destino. Os roteadores de requisições têm como objetivo rotear as mensagens recebidas pelo *middleware* baseado em uma análise contextual do seu conteúdo. Dessa forma, é possível inferir seu destino baseado em um conjunto de regras (Figura 4.3).

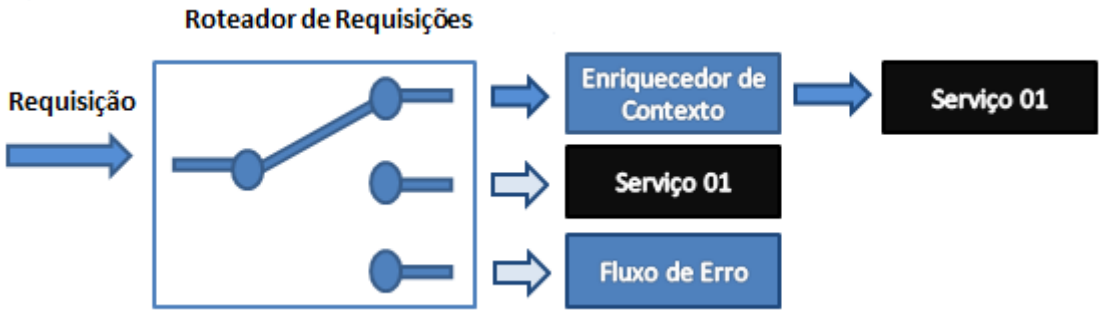

Figura 4.3: Roteador de requisições - A mensagem recebida é analisada e redirecionada para o serviço correspondente.

- Transformador de contexto: como a maioria das aplicações e serviços de terceiros possuem seus próprios modelos estruturais, é preciso ter um padrão responsável por modificar e transformar todas as informações manipuladas pelo *middleware* em um formato compreensível entre as partes envolvidas. Uma forma de fazer isso seria modificar todas as aplicações para usarem o mesmo modelo, o que é inviável tanto do ponto de vista de implementação, quanto do ponto de vista de negócio. Esse componente tem como objetivo permitir uma maior flexibilidade na comunicação entre as diversas aplicações e o *middleware*. Dado que um serviço foi requisitado, a requisição é analisada e baseado no contexto dos parâmetros recebidos, pode ser transformada para se adequar ao formato dos parâmetros esperados pelo serviço. No exemplo da Figura 4.4, o Transformador transforma os dados enviados pelas aplicações A e B no formato esperado pelo serviço (double) para que então o serviço possa ser executado corretamente. Assim, diferentes aplicações podem fazer a mesma

requisição de diferentes formas, cabe ao transformador de contexto interpretar as informações recebidas e transformá-las para o formato requerido pelo serviço.

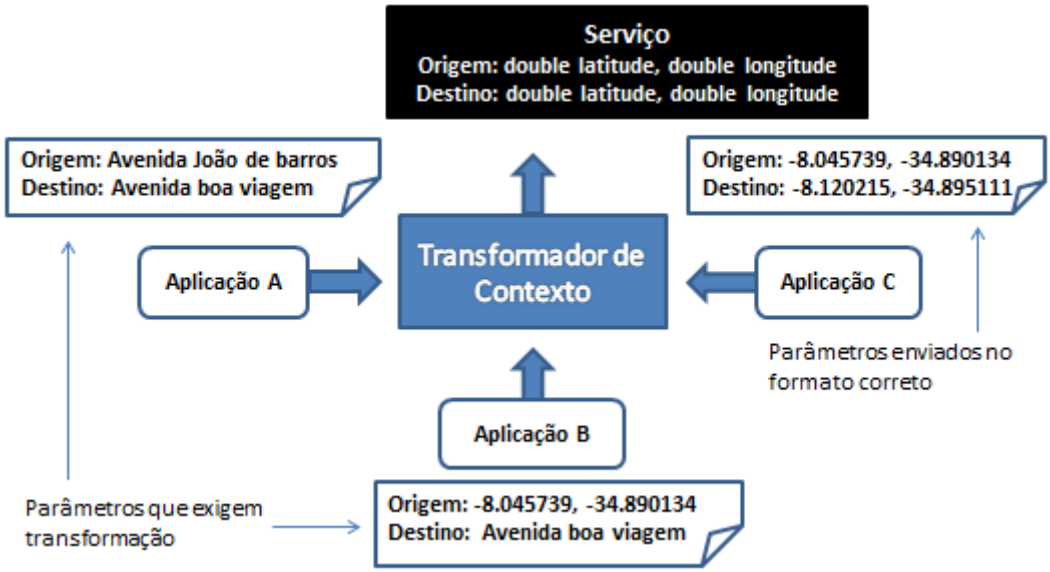

Figura 4.4: Transformador de contexto

- Enriquecedor de contexto: quando uma mensagem é enviada a outro sistema, é comum que o sistema receptor precise de mais informações do que as enviadas pelo remetente. Dessa forma, o UbiMid com o mínimo de informações necessárias é capaz de inferir informações contextuais do usuário, a partir de informações brutas e isoladas. A Figura 4.5 mostra um exemplo da utilização do componente de enriquecimento, onde a aplicação cliente manda apenas o CEP e o componente deriva as demais informações, ou seja, a partir do CEP é possível se extrair o endereço e a cidade onde o usuário se encontra, e a partir da cidade é possível se extrair o clima e a temperatura. A hora corrente é extraída do servidor no momento da requisição.

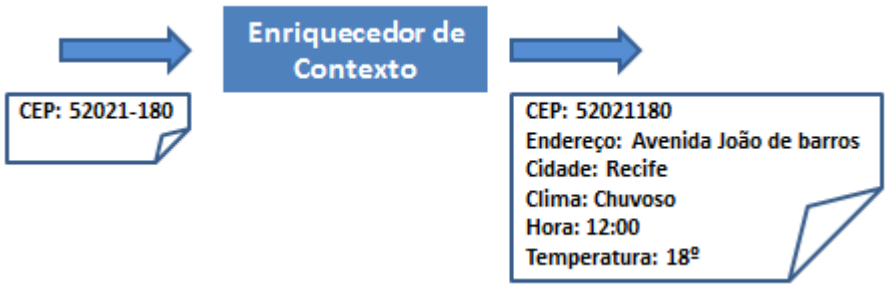

Figura 4.5: Enriquecedor de contexto

- Coletores de contexto: são componentes responsáveis pela comunicação entre o *middleware* e as fontes de contexto, bem como por fazer a aquisição e armazenamento dessas informações, caso necessário. Essas informações podem ser mensagens coletadas através das redes sociais, informações do clima, dentre outras informações

que o serviço julgue necessário e não sejam informadas de forma explícita pelo usuário. A Figura 4.6 apresenta dois exemplos da utilização dos componentes responsáveis pela coleta de informações contextuais onde em (a) ocorre a aquisição de informações relativas ao clima corrente em uma dada cidade e em (b) ocorre a captura e a inserção no banco das informações do Twitter, relacionadas ao trânsito de Recife, para serem analisadas posteriormente pelos serviços de inferência.

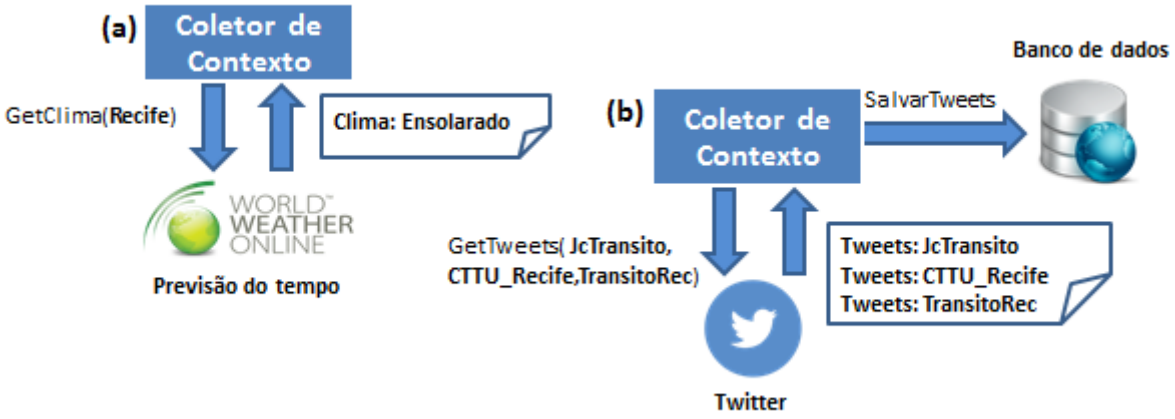

Figura 4.6: Coletores de contexto

- Serviços de inferência: são serviços capazes de inferir contexto lógico por meio de informações contextuais que podem ser adquiridas dinamicamente, disponibilizadas explicitamente pelo usuário ou mesmo através de informações armazenadas no banco.
- Analisador de QoS : o UbiMid também apresenta um componente QoS (*Quality Of Service*) ou análise de qualidade de serviço (Figura 4.7), que leva em consideração o tempo de resposta entre o servidor de destino e o *middleware* em questão. Dado que a arquitetura do UbiMid foi desenvolvida para facilitar o acoplamento de novos componentes ou serviços, é possível que existam serviços de coleta ou serviços de inferência semelhantes, ou seja, que desempenham a mesma função. Quando um serviço é solicitado ao *middleware* e existem várias formas de coletar ou processar uma determinada informação, o componente de análise verifica qual dos serviços existentes está disponível no momento e qual tem o menor tempo de resposta. Essa informação é enviada ao componente de roteamento que faz a análise dessas informações e encaminha a requisição para o serviço eleito.

Como visto no capítulo anterior, existem elementos básicos e fundamentais que se encontram presentes na maioria dos *middleware* sensíveis ao contexto que também foram levados em consideração na concepção e na elaboração desta arquitetura, como os coletores e provedores de contexto, uma camada de gerenciamento e distribuição e a existência de serviços ou módulos de inferência.

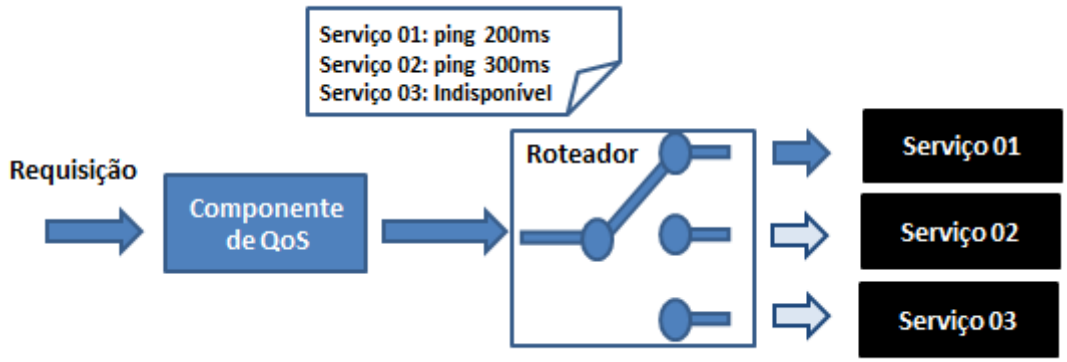

Figura 4.7: Componente de QoS - Análise do melhor serviço.

Nas próximas seções serão descritas com mais detalhes cada uma das camadas do *middleware* e, por fim, falaremos do funcionamento dos principais serviços disponibilizados pelo *middleware*.

4.2 Fontes e aquisição de contexto

A aquisição do contexto é realizada pelos componentes de coleta chamados de coletores de contexto, cujo objetivo é monitorar e/ou capturar informações contextuais relevantes de diversas fontes dados. Por sua vez, as fontes de dados são os locais de onde as informações são capturadas e usadas durante o processo de inferência de contexto.

No UbiMid, essa aquisição de informações pode ocorrer de forma explícita, onde o próprio usuário pode fornecer informações relevantes para execução de uma determinada tarefa. Por exemplo, as preferências do usuário podem ser explicitadas antes da execução da tarefa de consultar as melhores rotas para um dado usuário. Assim, ele pode informar se precisa de acessibilidade (cadeirante), se deseja pegar um ônibus mais caro, porém que chega mais rápido ao destino ou à parada, dentre outras coisas, sendo possível, dessa forma, sugerir o melhor ônibus que atenda suas necessidades.

Por outro lado, a aquisição de contexto também pode ser feita através de fontes de dados persistentes, cujas informações não mudam frequentemente como, por exemplo, as linhas e horários dos ônibus, informações que foram extraídas de arquivos de logs enviados pelo Grande Recife Consórcio de Transporte. Por fim, existem as informações obtidas por meio de sensores lógicos, no caso, serviços web que são utilizados para coletar informações contextuais do ambiente em que o usuário está inserido como, por exemplo, informações climáticas, informações do trânsito, dentre outras.

Os coletores de contexto podem adquirir informações contextuais de dois tipos de fontes de contexto que, quanto a sua natureza, podem ser classificadas como: internas ou externas. Com relação às fontes externas, o UbiMid faz uso de serviços web providos por fontes como:

Google Maps, *Twitter* e *World Weather Online*, além de fazer uso de arquivos de log providos pelo Grande Recife Consórcio de Transporte, para provisionamento, enriquecimento e inferência de informações contextuais atuais baseadas ou não no histórico dos dados. As fontes internas são compostas pelos sensores dos dispositivos móveis como *smartphones*, PDAs, dentre outros.

Como forma constante de aquisição de contexto, o UbiMi possui um mecanismo de agendamento, que funciona como uma rotina para captura de informações periodicamente em algumas fontes de dados externas como é o caso do Twitter, por exemplo.

Dentre as principais fontes de contexto está o repositório ou base de dados contextual. Ela é responsável pelo armazenamento de informações contextuais adquiridas pelos componentes de aquisição. As informações são manipuladas e armazenadas em um formato comum e padronizado, uma vez que são capturadas de diferentes fontes, podendo ser usadas tanto para consultas simples (recuperação de dados), como para manipulação e inferência de informações contextuais por meio de um conjunto de dados históricos armazenados. Quando necessário, o repositório pode ser acessado pelo componente de processamento e inferência para recuperação e análise de informações contextuais.

4.3 Gerenciamento de contexto: análise, processamento e inferência

A camada de gerenciamento de contexto provê um conjunto de métodos e processos que realizam inferência de contexto através de raciocínio lógico, transformação e enriquecimento de contexto, de modo a obter informações mais refinadas e relevantes a partir de informações contextuais de baixo nível, brutas e isoladas, que foram adquiridas através de diferentes fontes de dados.

Inicialmente, as informações obtidas pelos componentes de aquisição são processadas e tratadas para que então possam ser armazenadas no banco. Assim, quando requisitado, o *middleware* faz uso dessas informações, ou de informações obtidas dinamicamente, para inferir o contexto do usuário naquele dado momento.

Antes da requisição ser enviada para os serviços de inferência, ela pode passar por componentes de enriquecimento de contexto, cujo objetivo é a complementação dos parâmetros baseada nas informações faltantes (informações contextuais que podem ser adquiridas ou inferidas). Dessa forma, os serviços podem ser requisitados com uma tolerância de ausência de parâmetros, os quais são notados baseado no contexto da requisição e preenchidos pelo *middleware*. Por exemplo, o cenário onde o usuário está inserido (considerado um elemento contextual) é relevante no processo de ordenação de rotas. Caso a requisição seja feita sem ele, ele é montado pelo *middleware* com as informações contextuais necessárias e enviado ao serviço de ordenação.

Após a passagem pelo componente de enriquecimento, a requisição também pode passar

por um componente de transformação, onde é feita a manipulação dos dados, caso necessário. Por exemplo, uma determinada função recebe como um de seus parâmetros a localização do usuário, esta informação pode ser enviada em dois formatos diferentes, através de coordenadas geográficas (latitude e longitude) ou através de um endereço, dependendo do contexto do parâmetro, ele pode ser adaptado ou não, para então ser enviado à função original com os parâmetros bem definidos.

O *workflow* do processo descrito anteriormente pode ser evidenciado com mais detalhes e melhor compreendido conforme a Figura 4.8.

Tanto o componente de enriquecimento quanto o componente de transformação proporcionam uma grande flexibilidade e transparência em relação ao tipo e à quantidade de parâmetros enviados pelas aplicações consumidoras aos serviços providos pelo *middleware*. Essa flexibilidade proporciona uma baixa dependência entre os serviços e a aplicação consumidora, caso os serviços precisem adicionar novos parâmetros para melhorar a resposta ao usuário, o *middleware* pode enriquecer a requisição para se adequar às novas necessidades do serviço, sem a necessidade de comunicar a aplicação o surgimento de um novo parâmetro.

4.4 Distribuição e roteamento de requisições

A camada de disseminação funciona como uma API de comunicação entre o *middleware* e as aplicações consumidoras. Essa comunicação é feita através de protocolos de comunicação que são usados para fazer a requisição de um serviço (representados por setas não preenchidas no desenho da arquitetura). Após a requisição ser feita, ela passa por um roteador de requisições sensível ao contexto que, baseado em uma análise da composição dos parâmetros recebidos (quais informações foram fornecidas), pode encaminhar a requisição para o serviço correspondente de diferentes formas. Se todos os parâmetros estiverem presentes na requisição, o serviço é chamado diretamente, caso contrário, a requisição pode sofrer uma mudança de trajetória, em relação ao fluxo lógico de execução e pode ser enviada para o componente de enriquecimento e/ou transformação, antes da execução do serviço propriamente dito.

A comunicação também pode ser feita através de mensagens. Para isso, o *middleware* utiliza um mecanismo de *publish/subscribe* (pub/sub), onde uma aplicação qualquer se inscreve em um determinado canal de comunicação, mostrando interesse por uma determinada informação ou determinado assunto. Sinalizado esse interesse, a aplicação ficará recebendo mensagens caso um determinado evento aconteça. Tanto o envio de mensagens quanto o recebimento podem ocorrer de forma assíncrona, ou seja, o emissor envia as mensagens, caso o receptor não possa receber, essas mensagens são mantidas em uma fila e serão enviadas assim que o receptor estiver disponível.

Um exemplo de uso do serviço de mensagens seria: a aplicação cliente faz uma subscrição indicando o interesse do usuário pelo status de uma determinada rua (engarrafada, com acidente, em obras, trânsito fluindo), visto que esse usuário passa por essa rua todo dia ao voltar para casa. Dessa forma, a aplicação cliente ficaria recebendo informações contextuais relativas ao status

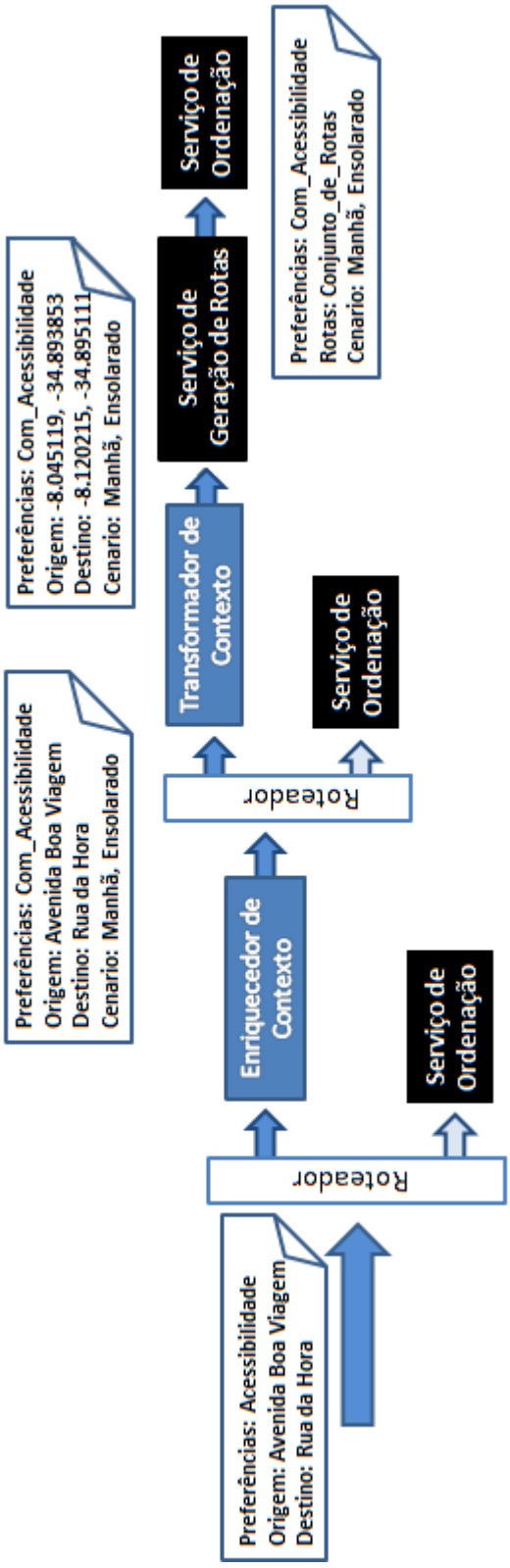

Figura 4.8: Gerenciamento de contexto - análise, processamento e inferência.

da via, assincronamente, por um período de tempo, baseadas em análises feitas pelo serviço de análise de mensagens que leva em consideração informações obtidas por meio das redes sociais. Essas informações auxiliam diretamente o processo de tomada de decisão do usuário e, dessa forma, ele pode optar por mudar o caminho para casa.

4.5 Serviços do UbiMid para sistemas inteligentes de público e urbano

Alguns dos serviços providos pelo UbiMid foram originados a partir da extração de módulos julgados úteis em outros trabalhos desenvolvidos no âmbito do projeto UbiBus, como é o caso do algoritmo de classificação e ordenação de melhor rota baseado nas preferências do usuário do *RecRoute* (OLIVEIRA TITO, 2013), o analisador de mensagens de texto obtidas por meio das redes sociais (Twitter) do *UbiBus Analysis* (LIMA, 2012) e o Gerador de Rotas do *Pilgrim* (BORGIANI, 2013). A seguir é dada uma breve descrição desses trabalhos, assim como é citado o seu relacionamento com o *middleware*.

- O *RecRoute* é um projeto que tem como objetivo facilitar o dia a dia das pessoas que utilizam transporte público, recomendando rotas aos passageiros em tempo real, baseadas em informações estáticas e dinâmicas de contexto relacionadas aos usuários e aos próprios meios de transporte. Tendo em mente sua principal contribuição, o UbiMid adaptou o módulo de ordenação/classificação de rotas para disponibilizá-lo como um serviço para outras aplicações que também têm interesse de utilizar ou prover essa informação.

- O *UbiBus Analysis* é uma ferramenta que utiliza a Análise de Sentimentos (AS) para análise de mensagens obtidas através das redes sociais. No caso, o UbiMid faz, periodicamente, a extração das mensagens de redes sociais como o Twitter (poderiam ser outras redes sociais), as quais são inseridas no banco para depois serem analisadas pelo módulo de inferência. Além disso, o UbiMid adaptou o serviço de análise para ele ser mais simples e flexível, fazendo com que o usuário utilize-o passando o mínimo de parâmetros possíveis.

- O *Pilgrim* é um projeto cujo objetivo é gerar e classificar rotas de ônibus em tempo real que sejam adaptadas ao contexto que permeia o sistema de transporte público rodoviário, em especial o trânsito. Para tanto, ele propõe duas abordagens utilizando técnicas de otimização da inteligência artificial, Algoritmos Genéticos e *Hill-Climbing*, para o desenvolvimento do sistema. No UbiMid, o Pilgrim, devido a sua funcionalidade, é chamado de Gerador de Rotas.

Nesta seção serão apresentados com mais detalhes os serviços providos pelo *middleware* citados nas seções anteriores durante a descrição dos componentes da arquitetura. Após uma

explanação detalhada sobre o funcionamento de cada serviço, aspectos de modelagem dos seus componentes serão representados através de diagramas de sequência utilizando UML (*Unified Modelling Language*).

4.5.1 Serviços de geração e ordenação de rotas

O *middleware* tem o papel de prover a comunicação entre o Gerador de Rotas e o ordenador do RecRoute, pois foram utilizadas diferentes plataformas para elaboração de cada uma das soluções, fato que resulta em problemas relacionados à heterogeneidade de características e incompatibilidade de linguagens. Esse problema demanda que as aplicações se comuniquem através de um protocolo de comunicação, onde as interfaces de comunicação sejam conhecidas.

A utilização desse serviço evita a necessidade do conhecimento das diferentes aplicações envolvidas, abstraindo suas heterogeneidades, proporcionando uma maior transparência e simplicidade, facilitando o entendimento e propiciando o reuso de código para as aplicações clientes. Além disso, como o objetivo é prover serviços para usuários de transporte público que estejam, principalmente, nas ruas acessando as aplicações clientes através de um dispositivo móvel, esse serviço tem como vantagem o processamento de informações sendo feito no servidor, uma vez que dispositivos móveis possuem restrições de memória e processamento, poupando, dessa forma, a bateria.

O serviço de ordenação tem início quando a aplicação cliente requisita ao *middleware* informações relativas às melhores rotas para um dado usuário de acordo com suas preferências. Caso sejam fornecidos todos os parâmetros necessários para execução do serviço, o roteador, baseado no contexto dos parâmetros, identifica que a requisição está completa e logo a encaminha ao serviço de ordenação. Caso a requisição esteja incompleta (ver diagrama de sequência na Figura 4.9), o roteador analisa quais elementos contextuais estão faltando e muda o trajeto da chamada do serviço de ordenação para que a requisição possa ser enriquecida, para, por fim, o serviço de ordenação ser invocado.

Para o ordenador de rotas, o “cenario”, composto por “turno” e “clima”, é considerado um parâmetro relevante. Da mesma forma, o *middleware* o considera como um elemento contextual, que mesmo não sendo informado, pode ser obtido dinamicamente. Ou seja, se esta informação estiver ausente na requisição, o roteador encaminhará a requisição ao componente de enriquecimento responsável por adquirir, processar e inferir esta informação.

Já para ordenar as melhores rotas, é necessário ter as informações relativas a cada rota, para que então elas possam ser classificadas em relação às preferências do usuário. Dessa forma, se as rotas não forem informadas, o roteador vai sinalizar sua ausência mudando novamente o fluxo da chamada ao serviço de ordenação. Assim, para geração das rotas, o *middleware* considera a *origem* e o *destino* do usuário como elementos contextuais e, baseado no contexto da requisição, ela é enviada para o componente de transformação que identifica em que formato foram passadas essas informações. Se elas foram passadas como coordenadas geográficas ou se a

especificação foi feita através de um endereço, é possível se fazer a adaptação dessa informação de acordo com as necessidades do serviço consumidor, no caso, o Gerador de Rotas.

Este serviço também leva em consideração o número máximo de trocas de ônibus que o usuário aceita e o horário da viagem. Caso esses elementos contextuais não tenham sido informados, o roteador de requisições encaminha a requisição ao componente de enriquecimento correspondente, que faz o processamento dessas informações, considerando a hora como a hora corrente e, baseado nas preferências, infere a aceitação ou não de trocas de ônibus pelo usuário.

Por fim, com todas as informações disponíveis (cenário, rotas e preferências) em mãos, o *middleware* as envia ao serviço de ordenação de rotas. Sua resposta é processada e enviada de volta à aplicação solicitante juntamente com as informações das rotas, caso estas não tenham sido fornecidas a priori.

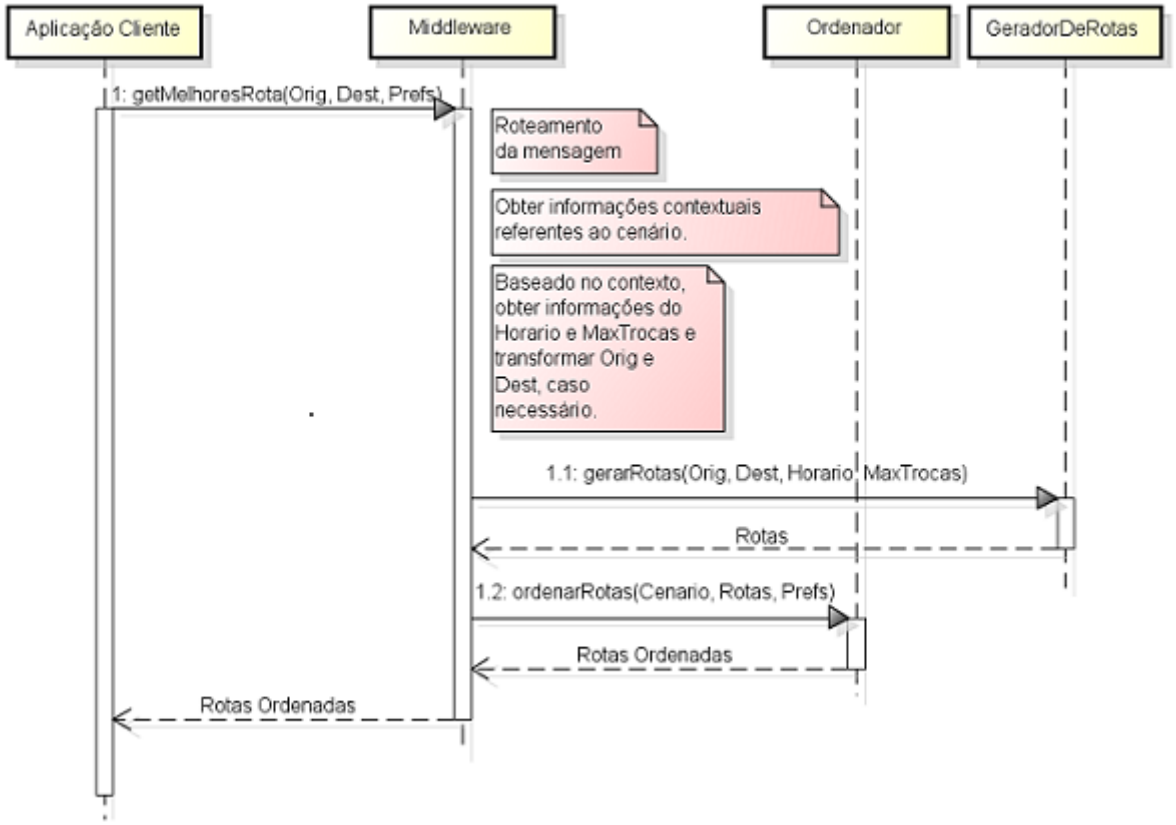

Figura 4.9: Diagrama de sequência - Integração entre o RecRoute e o Gerador de Rotas.

4.5.2 Serviços de análise de mensagens do Twitter e geração de ocorrências

O serviço de análise de mensagens e geração de ocorrências é um serviço provido pelo UbiMid em conjunto com o *Ubibus Analysis*, onde este é responsável pelo algoritmo de análise de mensagens e geração de ocorrências e aquele é responsável pela extração, transformação e

armazenamento dos *tweets*, além de prover o enriquecimento da requisição baseado no contexto do usuário.

O diagrama de sequência ilustrado na Figura 4.10 descreve o fluxo de execução do serviço de geração de ocorrências. O *middleware* recebe a mensagem e a envia ao componente de roteamento que a analisa e a encaminha para o serviço correspondente. Caso a mensagem esteja completa, com todos os parâmetros preenchidos (*horário, data, localização e intervalo*), ela é enviada diretamente para o componente de transformação, que verifica o formato da localização, se foi fornecida como coordenadas geográficas ou se foi informada como o endereço, para em seguida enviar ao serviço de inferência. Caso esteja faltando algum parâmetro (*horário, data ou intervalo*), ela é encaminhada ao componente de enriquecimento correspondente, que se encarrega de obter essas informações baseadas no contexto do usuário.

Se a requisição for solicitada enviando-se apenas a localização, por exemplo, "Avenida João de Barros", a mensagem é enviada para o componente de enriquecimento, que trata de preencher o *horário* com a hora corrente, a *data* com o dia atual e o *intervalo* de acordo com o horário, levando em consideração os horários de pico, pre-estabelecidos, que pela manhã seria entre 7h e 9h, à tarde entre 11:30 e 13:30 e à noite entre 17:30 e 19:30. Baseado nesses parâmetros o *intervalo* varia entre 30 e 60 minutos. Após todo esse processo de enriquecimento dos dados, a requisição está pronta para ser enviada para o serviço de inferência que analisa e processa a mensagem, responde ao *middleware*, e este a encaminha para a aplicação consumidora.

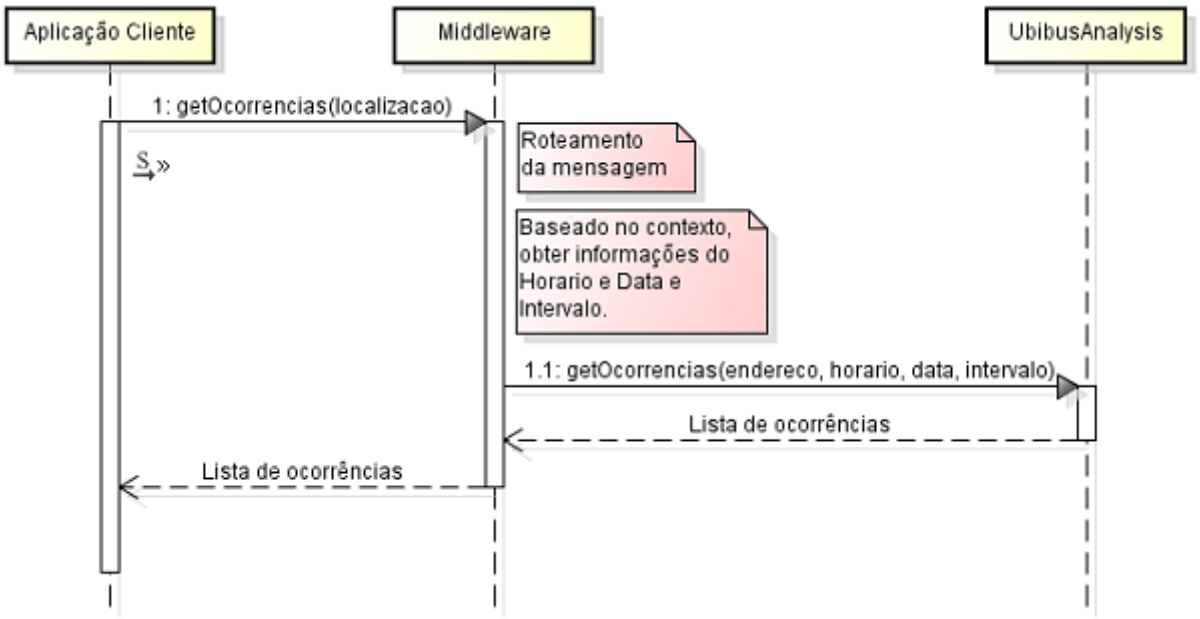

Figura 4.10: Diagrama de sequência - Integração serviço de ocorrências.

4.5.3 Serviço de *publish/subscribe* relativos às condições das vias

O paradigma de comunicação *publish/subscribe* (EUGSTER et al., 2003) foi utilizado por ser uma solução interessante quando aplicado a um cenário de mudanças constantes, como é o caso das condições de trânsito das vias urbanas. Estas mudanças podem ser ocasionadas por diferentes fontes como: acidentes, mal funcionamento dos semáforos, alagamento, passeatas, buracos, obras ou bloqueios na via, ou mesmo engarrafamento em horários de pico, sendo necessário, dessa forma, mecanismos informativos baseados em eventos. Abaixo será definido como cada funcionalidade desse módulo foi definida.

Subscrição: quando o usuário de uma aplicação tem interesse em informações relacionadas ao status de uma avenida ou rua, ele cria uma mensagem de subscrição informando o nome da via e a envia ao *middleware*. Essa mensagem contendo informações relacionadas aos eventos de interesse devem ser atualizadas constantemente para indicar que a aplicação ainda tem interesse por aquela informação. A falta de mensagens de ratificação indicarão que a aplicação não necessita mais da notificação daquele evento. Dessa forma, não precisa haver uma mensagem para cancelar a inscrição, dado um certo tempo, ela é cancelada automaticamente.

Recepção de subscrição: quando o *middleware* recebe a mensagem de subscrição, ele verifica se essa mensagem é uma mensagem válida, ou seja, se podem existir eventos relacionados àquela mensagem. Se a mensagem não for uma mensagem válida (se o endereço da via estiver incorreto ou se ele não estiver registrado no repositório), o *middleware* responde à aplicação que a subscrição não pode ser realizada. Se já houver uma subscrição feita com informações daquela via, é dado um *reset* no tempo de expiração da mensagem e a contagem recomeça. Se for uma nova mensagem o *middleware* responde ao cliente que a inscrição pode ser feita e uma nova inscrição é inserida no repositório de pub/sub.

Publicação: quando um evento ocorre, o *middleware* verifica no repositório de subscrições se existe alguma interessada naquele dado evento. Caso seja encontrada alguma subscrição interessada, um evento é lançado com uma mensagem com o nome da via e o seu status (engarrafada, em obras, acidente, semáforo defeituoso). O objetivo dessa mensagem é auxiliar o usuário ou aplicação cliente no seu processo decisório. Por exemplo, dado que o cliente fez duas subscrições informando o nome de duas possíveis vias (A e B) que ele pode optar quando volta de casa para o trabalho, e ele recebeu notificações informando que ocorreu um acidente na via A, naturalmente ele poderá optar por seguir pela via B, que está, provavelmente, com um trânsito melhor.

Recepção da notificação: quando a aplicação cliente recebe uma notificação, ela verifica se ainda tem interesse por aquela informação, e se é uma informação válida. Caso a informação não seja uma informação de interesse da aplicação cliente ela é descartada. Caso seja, ela pode fazer uso da notificação, mas não precisa confirmar ao *middleware* o recebimento da mensagem.

4.6 Considerações finais

Neste capítulo apresentamos a arquitetura do UbiMid, bem com o detalhamento dos seus componentes, cujo objetivo é prover transparência, flexibilidade e sensibilidade ao contexto tanto aos serviços quanto as aplicações consumidoras. Dessa forma, ele deixa de ser apenas uma camada de abstração para a comunicação entre aplicações heterogêneas e o processamento de informações, passando a fazer parte do processo de tomada de decisão no ciclo de vida da chamada e execução do serviço. A aplicação não precisa mais requisitar um serviço específico com todos os parâmetros para atender suas necessidades. Ela pode chamar um determinado tipo de serviço e o *middleware* se encarrega de redirecionar a requisição para o melhor serviço em termos de QoS (o mais rápido e disponível no momento) de acordo com os parâmetros enviados pela aplicação requerente.

No próximo capítulo serão abordados detalhes de implementação como tecnologias utilizadas, bem como resultados alcançadas com o desenvolvimento do protótipo.

5

Implementação e experimentos

Neste capítulo entraremos em mais detalhes sobre o processo implementação do *middleware*. Nas seções a seguir serão apresentadas as tecnologias e bibliotecas que foram utilizadas e como foi feita a escolha delas. Após falarmos da implementação, serão mostrados os experimentos realizados e os resultados obtidos.

5.1 Tecnologias utilizadas

Para a implementação do UbiMid foi utilizada a linguagem de programação Java e o Mule ESB juntamente com algumas bibliotecas extras para execução de tarefas específicas.

Para fazer o *deploy* da aplicação desenvolvida no servidor de produção do projeto utilizando o Mule ESB, foi necessário instalar o *Mule ESB Enterprise Runtime*. Existe também a possibilidade de colocar a aplicação Mule na nuvem através do *CloudHub*[1] que é uma plataforma para hospedagem de aplicações desenvolvida utilizando o Mule ESB, que tem como características permitir o crescimento e escalonamento da aplicação sobre demanda.

Também foi utilizado o servidor de aplicação web Apache Tomcat 7 [2] para hospedagem de aplicações complementares desenvolvidas em Java que foram utilizadas para a chamada de serviços mais simples e realização de tarefas secundárias.

A principal tecnologia utilizada para comunicação entre as aplicações clientes e o servidor foram os *web services*, apesar de também terem sido usados serviços de mensagens assíncronos baseados no tempo (*time-driven*). Os *web services*, por sua vez, foram implementados utilizando a tecnologia RESTful ou REST (*Representational State Transfer*) (FIELDING, 2000) por ser uma tecnologia mais simples, escalável e performática (MENG; MEI; YAN, 2009) se comparada ao SOAP (*Simple Object Access Protocol*), por exemplo. Ainda segundo MENG; MEI; YAN (2009), grandes empresas como *Google*, *Amazon*, *Yahoo* e *eBay* já divulgaram suas APIs REST e, segundo estatísticas do site mais popular de APIs e *Mashups* (ProgrammableWeb.com), os serviços web RESTful correspondem a 66% do total das publicações efetuadas. Tornou-se uma

[1] http://www.mulesoft.org/documentation/display/current/CloudHub
[2] http://tomcat.apache.org/

tendência utilizar esse tipo de serviço para o desenvolvimento de sistemas distribuídos.

O UbiMid utiliza o Sistema de Gerenciamento de Banco de Dados (SGBD) PostgreSQL[3] com a extensão PostGIS[4]. O PostgreSQL é um dos líderes mundiais em sistemas de banco de dados relacionais de código aberto. Ele tem mais de 15 anos de desenvolvimento ativo, já encontra-se consolidado no mercado, possui uma forte reputação de confiabilidade, integridade de dados e conformidade a padrões e roda em todos os grandes sistemas operacionais (GNU/Linux, Unix e Miscrosoft Windows). Já o PostGIS é uma extensão do PostgreSQL cujo objetivo é prover suporte à utilização de objetos geográficos, permitindo que consultas de localização sejam feitas usando SQL. A Figura 5.1 mostra as principais entidades do modelo ER (Entidade Relacionamento) do banco de dados do projeto Ubibus.

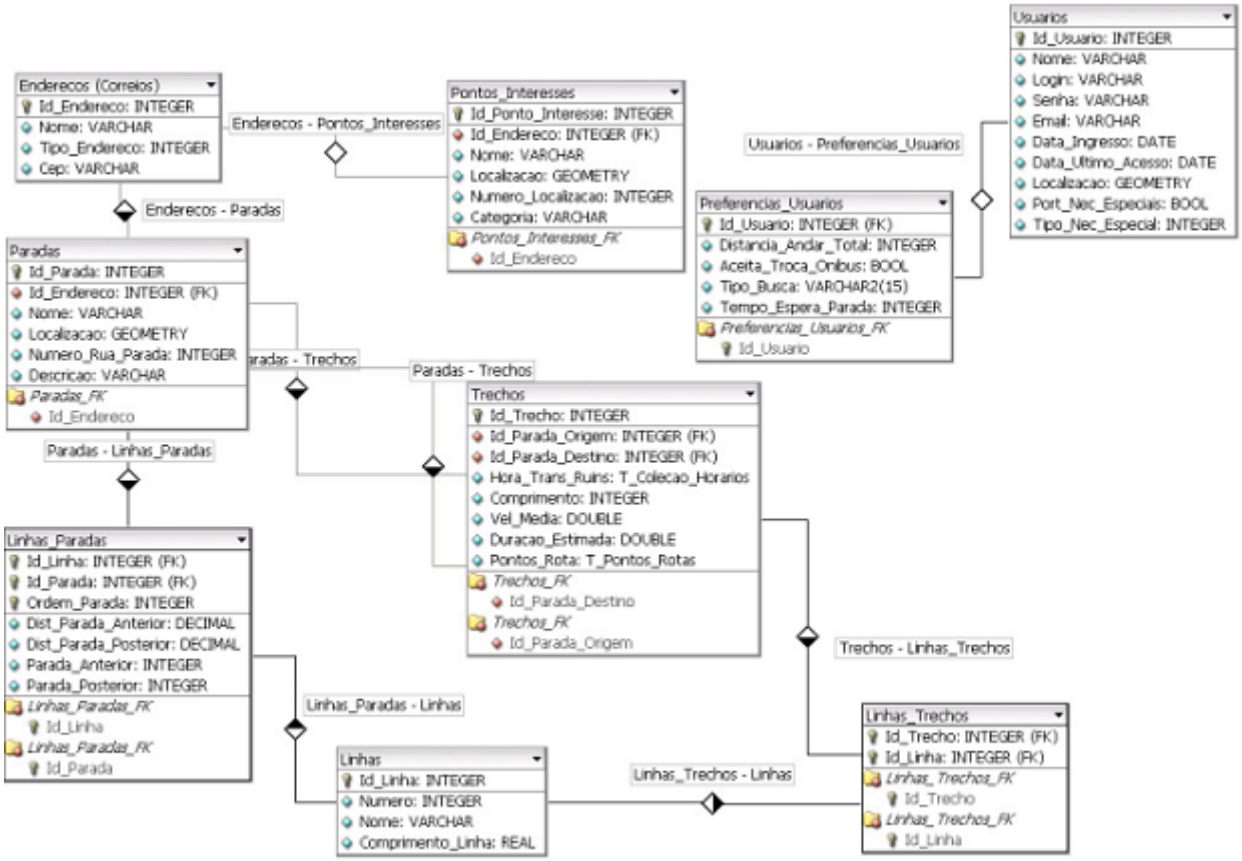

Figura 5.1: Modelo entidade relacionamento das principais tabelas do banco do projeto Ubibus.

A base de dados do projeto foi alimentada com dados reais tanto da cidade de João Pessoa, quanto da cidade de Recife. Para o povoamento do banco com os dados de Recife entrou-se em contato com o Grande Recife Consórcio de Transporte e para os dados de João Pessoa foi feita uma parceria com o SEMOB (Superintendência de Mobilidade Urbana). Dessa forma, foram disponibilizadas informações como a localização das paradas de ônibus, o trajeto

[3] www.postgresql.org
[4] www.postgis.net

dos ônibus, bem como informação relativas às empresas fornecedoras do serviço de transporte público. Informações relacionadas à localização dos pontos de interesse para associação com as paradas mais próximas que seriam utilizadas pelo serviço de geração de rotas foram adquiridas através do *Google Places*[5] juntamente com o *Google Geocoding*[6] que tinham como objetivo fazer a devida associação com os seus respectivos endereços.

Para o desenvolvimento do *middleware* propriamente dito utilizou-se o Mule ESB criado pela empresa MuleSoft[7] e disponibilizado através de uma *framework* chamada Mule Studio desenvolvido sobre a linguagem de programação Java. Dentre seus principais objetivos estão prover rapidez e facilidade de uso durante o processo de desenvolvimento e integração entre aplicações. Essa rapidez e facilidade são consequência da disponibilização de uma gama de componentes gráficos que facilitam a configuração e a execução de um conjunto de tarefas recorrentes durante o processo de criação de um *middleware*. A seguir serão descritos os principais componentes do Mule ESB utilizados na construção do UbiMid, bem como suas responsabilidades.

Para a comunicação entre as aplicações e o UbiMid foi utilizado um conjunto de *endpoints* que representam as extremidades do *middleware*, que por sua vez podem ser tanto componentes de entrada quanto de saída. A seguir serão descritos os *endpoints* utilizados e suas respectivas funcionalidades no UbiMid:

- HTTP - Esse componente pode ser usado tanto para se fazer uma requisição à serviços de terceiros quanto para receber as requisições das aplicações clientes. Ele representa o principal ponto de comunicação entre o UbiMid e as aplicações consumidoras de contexto. A requisição deve ser feita utilizando o método POST e é do tipo *request/reply*, dessa forma, o cliente faz a requisição e fica aguardando a resposta do serviço. Assim que a requisição é feita, o componente HTTP a recebe e a encaminha a um componente de transformação para tradução, tornando a requisição legível para os componentes do *middleware*. Além disso, o componente HTTP é responsável pela especificação da porta e do nome do serviço que será requisitado.

- Database - Componente utilizado para fazer a conexão com os mais diversos tipos de banco, que podem ser: MySQL, PostgreSQL, Oracle, DB2, Micrsoft SQL, mais alguns outros, além de permitir realizar operações no banco de dados como: consultas, inserções, atualizações, dentre outras.

- JMS - Esse componente encapsula uma API para desenvolvimento de um MOM. Ele permite a comunicação entre os diferentes componentes de uma aplicação distribuída para ser flexível, confiável e assíncrono. Esse componente foi utilizado na implementação do mecanismo de *publish/subscribe*, permitindo que as aplicações inscritas em

[5] https://developers.google.com/places/documentation/?hl=pt-br
[6] https://developers.google.com/maps/documentation/geocoding/?hl=pt-br
[7] http://www.mulesoft.org/

um determinado tópico fiquem recebendo as mensagens que as interessa de forma assíncrona.

- Quartz - É um componente que permite o estabelecimento de rotinas. Através dele é possível a criação de milhares de tarefas que podem ser executadas em intervalos determinados, periodicamente ou indefinidamente (QUARTZ, 2013). No UbiMid, ele fui utilizado para o agendamento da coleta de mensagens das redes sociais.

Além dos componentes citados acima, existem os *cloud connectors* que são componentes que fazem a conexão com serviços disponíveis na nuvem como, por exemplo, *Amazon Amazon Simple Queue Service*[8] (SQS), *Google Calendar*, *Google Tasks*, *Salesforce*, ou redes sociais como Facebook e Twitter. No caso, o UbiMid utilizou o componente de comunicação com o Twitter para a coleta de tweets. Dessa forma, não é necessário buscar ou importar nenhuma API no Google, pois ela já se encontra encapsulada no componente do Mule, nesse caso, basta apenas configurá-lo e utilizá-lo. Se essa funcionalidade tivesse sido desenvolvida em Java sem o auxilio de um ESB, seria necessário buscar a API que fizesse essa comunicação como, por exemplo, a twitter4j que foi usada em um dos protótipos iniciais do *middleware*.

Também foram utilizados componentes para transformação dos dados, estes são componentes mais simples que fazem a conversão de um tipo em outro, por exemplo, é possível converter um objeto JSON [9] em um objeto Java, um objeto Java em XML (BRAY et al., 1997), um Array de Bytes em uma String e a recíproca é verdadeira para todos os casos citados. Foi utilizado também o componente para tratamento de erro chamado *Catch Exception Strategy*, cujo objetivo é fazer o tratamento de requisições inválidas. Por fim, é importante salientar que o UbiMid utilizou componentes da categoria dos controladores de fluxo para fazer o roteamento das requisições baseado em um conjunto de regras estabelecidas.

Na próxima subseção serão abordadas algumas bibliotecas e *drivers* que precisaram ser incorporadas ao *middleware*, pois apesar do Mule prover as interfaces para fácil configuração e utilização, os *drivers* não se encontram embutidos nos componentes do ESB, deixando o usuário livre para importar a versão desejada. Por exemplo, existe o componente para conexão com o banco de dados, que pode ser facilmente configurado utilizando o Mule, porém é preciso importar o seu *driver* no projeto para que o componente funcione corretamente.

5.1.1 Bibliotecas

As bibliotecas citadas a seguir foram utilizadas para facilitar o desenvolvimento de tarefas específicas, como o Gson, por exemplo, bem como para servir como meio de comunicação entre os componentes utilizados pelo *middleware* e os *drivers* utilizados. Tantos as bibliotecas quanto suas funcionalidades serão descritas a seguir:

[8] http://aws.amazon.com/pt/sqs/
[9] http://www.json.org/

- Gson[10]- É uma biblioteca desenvolvida em Java que pode ser usada para converter strings no formato JSON (*JavaScript Object Notation*) em um objecto Java, bem como converter um objeto Java na sua equivalência em JSON. Além disso, ela também auxilia no processo de manipulação das informações possibilitando o fácil acesso, adição e exclusão de novos campos. A versão utilizada nessa dissertação foi a 2.2.4.

- PostgresSQL JDBC (*Java Database Connectivity*) - Biblioteca utilizada para fazer a conexão com o banco de dados do projeto permitindo a recuperação e inserção de novos dados. A versão utilizada foi a 9.2.

- ActiveMQ - Biblioteca utilizada para estabelecer a comunicação entre o UbiMid e o servidor de mensagens. O Apache ActiveMQ é um dos servidores de mensagens mais populares e poderosos disponíveis na internet. A versão utilizada nesta dissertação foi a versão 5.8.0, contudo, já foi anunciado o lançamento de uma nova versão no final de 2013.

5.1.2 Formato de representação das informações providas pelo *middleware*

Foi feita uma pesquisa para se escolher qual a melhor forma de representar as informações providas pelo *middleware* para as demais aplicações. Inicialmente, a ideia era disponibilizar a resposta das requisições apenas em XML (*Extensible Markup Language*), formato de representação universal criado pelo consórcio técnico internacional chamado W3C (*World Wide Web Consortium*). Contudo, apesar do XML ser uma tecnologia bastante madura e consolidada, além de possuir uma grande quantidade de tutoriais e informações disponíveis na internet, o JSON, outra tecnologia para representação de objetos para transmissão de informações que vem crescendo bastante.

Ambos, tanto o XML quanto o JSON, foram criados com o objetivo de serem fáceis de usar, legíveis e de fácil compreensão tanto para o homem quanto para o computador. Porém, o JSON apresenta algumas vantagens quando a comunicação é feita do servidor para um browser, pois por ser suportado diretamente pelo *JavaScript*, a análise das informações transmitidas pode ser feita até 100 vezes mais rápida se comparado ao XML que faz uso de uma biblioteca extra para leitura dessas informações (*DOM - Document Object Model*) (NURSEITOV et al., 2009), essa análise na linguagem técnica é chamada de *parse*.

Devido ao formato mais enxuto das informações que trafegam pela rede no formato JSON, sua velocidade de transmissão é bem superior e o consumo de memória e CPU para leitura e análise dessas informações é bem menor, se comparado com o XML (NURSEITOV et al., 2009).

[10]https://code.google.com/p/google-gson/

Tanto para aplicações web, que irão processar a informação diretamente no navegador, quanto para aplicações web e, principalmente, móveis cuja largura de banda é pequena e bastante limitada (no caso de redes 3G), se sugere o uso do formato JSON. E é justamente esse o foco do UbiMid , prover informações dinâmicas e em tempo real, de forma rápida e eficiente, para o usuário de aplicações móveis, com seus *smartphones*, a respeito das vias, das paradas de ônibus, ou mesmo do próprio ônibus, dentre várias outras, baseadas no contexto e nas preferências do usuário. Levando em consideração essas informações e os requisitos do *middleware*, foi escolhido o formato JSON como sendo o formato de comunicação tanto de entrada como saída.

5.2 Métodos para avaliação de arquiteturas de *software*

Dentre os objetivos deste trabalho estão a proposição e o desenvolvimento de uma arquitetura de *middleware* sensível ao contexto que, apesar ter sido validada por meio do desenvolvimento de um *middleware* na área de sistemas inteligentes de transporte, ela pode adaptada para qualquer outro domínio, da mesma forma, a escolha da tecnologia utilizada é irrelevante.

É importante frisar que a escolha de uma arquitetura de software adequada reduz o custo e o esforço do desenvolvimento e manutenção, melhora a qualidade do software desenvolvido e evidencia os potencias riscos e gargalos. Além disso, não existem arquiteturas inerentemente ruins ou boas, a qualidade da arquitetura vai depender do tipo de negócio ou atividade em que ela será empregada.

Existem pelo menos 10 métodos e técnicas para a avaliação de arquiteturas de software no seu estado inicial que visam avaliar atributos de qualidade (ROY; GRAHAM, 2008). A Figura 5.2 mostra o relacionamento entre atributos de qualidade e arquiteturas de *software* de um modo geral. Os atributos podem ser: manutenibilidade, performance, segurança, confiabilidade, usabilidade, portabilidade, reusabilidade, dentre outros. Entre essas técnicas, há algumas voltadas para a validação de atributos específicos como, por exemplo, o SALUTA (*Scenario-based Architecture Level UsabiliTy Analysis*) (FOLMER; VAN GURP; BOSCH, 2005) que visa a validação da qualidade do atributo usabilidade. Contudo, há também técnicas mais genéricas que podem avaliar múltiplos atributos.

Esses métodos de avaliação da arquitetura baseados em cenário são aplicados antes da arquitetura ser implementada. A arquitetura do UbiMid foi evoluindo utilizando essas técnicas e tendo em mente as arquiteturas propostas no estado da arte. Inicialmente, tinha-se uma ideia dos componentes necessários, alguns cenários eram montados por meio de *brainstorms*, e a arquitetura foi se transformando até a versão final apresentada.

Foi utilizado o método de avaliação SAAM (*Scenario-based Software Architecture Analysis Method*) (KAZMAN et al., 1994), cujo objetivo é verificar os princípios da arquitetura de acordo com os documentos que a descrevem, no caso, as especificações feitas no capítulo anterior. Esse método possui seis atividades para elicitação dos cenários que foram seguidas neste

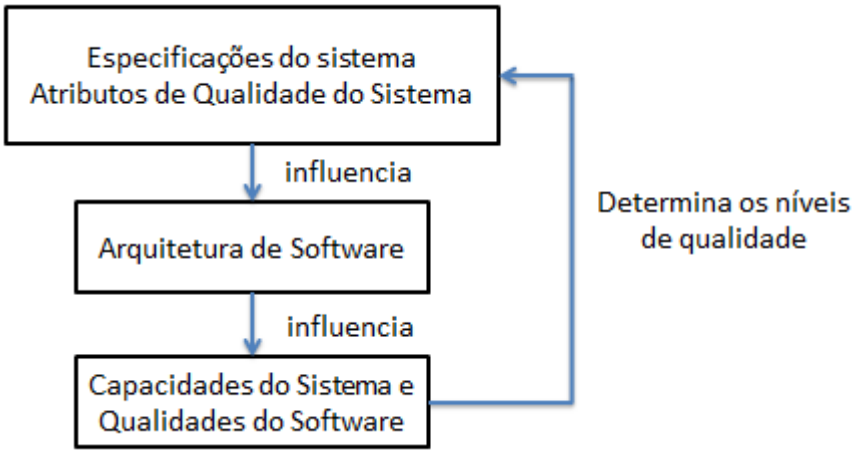

Figura 5.2: Relacionamento entre a arquitetura de um software e os atributos de qualidade. Adaptado de (ROY; GRAHAM, 2008).

trabalho. Contudo, o SAAM não considera as interações entre os atributos de qualidade. Para isso, foi utilizado o ATAM (*Architecture-based Tradeoff Analysis Method*) (ROY; GRAHAM, 2008) que segue uma linha de raciocínio da seguinte forma: em um *middleware*, a performance e a disponibilidade em uma arquitetura cliente/servidor podem ser melhoradas aumentando-se o número de servidores, entretanto, essa decisão arquitetural pode causar um impacto negativo na segurança do sistema, aumentando o número de potenciais pontos de ataque ou falha (ROY; GRAHAM, 2008).

Tendo em mente os atributos prioritários e elicitando cenários que atendem esses atributos, é possível fazer a validação da arquitetura através dos cenários mais prioritários e analisar o resultado para ver se ele é o esperado. No caso do UbiMid, o provisionamento de contexto era o principal foco do *middleware*, dessa forma foram priorizados os cenários com a utilização de contexto, tendo em mente que esses cenários afetam negativamente o desempenho, uma vez que informações contextuais precisam ser processadas para a derivação e inferência de novas informações.

A seguir será mostrado um cenário de aplicação sem o uso do *middleware* e depois com o uso do *middleware*. Esse foi um dos cenários utilizados para a elaboração da arquitetura e foi escolhido por ser um cenário prioritário e mais completo, que envolve a utilização de todos os componentes. Também serão mostrados alguns dos atributos de qualidade que ele atende, bem como os *tradeoffs* entre os atributos.

5.3 Cenário de Aplicação

Como prova de conceito, essa seção ilustra um cenário de aplicação da utilização do protótipo do *middleware* pela aplicação móvel desenvolvida em Android, chamada RecRoute. Poderiam ser descritos outros cenários nesta seção, contudo, o trabalho ficaria muito extenso. Além disso, o cenário escolhido é bastante completo no que diz respeito ao desenvolvimento e

utilização dos componentes sugeridos.

Inicialmente será mostrado como o serviço de Ordenação de Rotas funcionava para depois mostrar como ele passou a ser usado com a utilização do UbiMid. Ao final serão mostradas as vantagens incorporadas com a utilização de um *middleware* sensível ao contexto se comparado a serviços comuns.

A critério de teste no RecRoute, como o foco do trabalho era a avaliação dos resultados obtidos através da execução do algoritmo de ordenação das rotas, os parâmetros relativos ao "cenario" e às "rotas" durante o processo de experimentação foram simulados com dados reais, contudo, não houve de fato a integração entre os sistemas. Esses fatores dificultaram a comparação com relação a uma análise quantitativa de atributos como desempenho, por exemplo. No entanto, foi dado um foco maior na comparação com característica como: reusabilidade, portabilidade, extensibilidade, dentre outras características que serão vistas no decorrer deste capítulo.

5.3.1 Cenário de utilização do serviço de recomendação de rotas para usuários do RecRoute sem a utilização do *middleware*

Para a recomendação ordenada das melhores rotas para um determinado usuário, o serviço de ordenação precisa das preferências do usuário (se o usuário precisa de ônibus com acessibilidade, se ele prefere pegar um ônibus mais barato, ou um ônibus mais caro, mas que chegue mais rápido, se o usuário aceita troca de ônibus, dentre outras), informações quanto ao cenário, ou contexto em que o usuário se encontra, composto pelas condições climáticas (dia chuvoso, ensolarado ou nublado) e temporais (manhã, tarde ou noite), e um conjunto de rotas de ônibus, para dentre elas, eleger as melhores.

A Figura 5.3 mostra a transformação das informações citadas no parágrafo anterior em uma chamada de método passando os parâmetros no formato JSON (essa representação foi utilizada para critério de comparação na seção posterior) simulando a chamada ao serviço de ordenação de rotas do RecRoute.

Tendo em mente que a aplicação forneceu as informações relativas às rotas ao serviço, a resposta à função de ordenação é o número correspondente a cada uma das rotas de forma ordenada. Por exemplo, caso a resposta seja: 4, 2, 3, 1, indica que a rota de número 4 é a melhor rota para o usuário, a rota de número 2 é a segunda melhor, e assim por diante. Os números correspondentes às rotas são relativos à ordem na qual elas foram passadas para execução do algoritmo de classificação.

{"cenario":{"clima":"ensolarado","horario":"manha"},"rotas":[{"andar_total":"170","tempo_espera":"20","tem_troca":"nao","tempo_total":"45","distancia_total":"4800","tem_acessibilidade":"sim","preco_total":"2.3"},{"andar_total":"520","tempo_espera":"20","tem_troca":"sim","tempo_total":"60","distancia_total":"6000","tem_acessibilidade":"sim","preco_total":"2.3"},{"andar_total":"520","tempo_espera":"30","tem_troca":"nao","tempo_total":"55","distancia_total":"5000","tem_acessibilidade":"nao","preco_total":"2.3"},{"andar_total":"480","tempo_espera":"5","tem_troca":"nao","tempo_total":"30","distancia_total":"5000","tem_acessibilidade":"nao","preco_total":"2.3"}],"preferencias":{"aceita_troca_onibus":"sim","qual_deficiencia":"nenhuma","tipo_busca":"tempo","espera_parada":"10","quanto_andar":"30"}}

Figura 5.3: Em destaque estão os nomes do principais atributos - JSON composto por um cenário, quatro rotas de ônibus e as preferências de um determinado usuário para a execução do algoritmo de ordenação.

5.3.2 Cenário de utilização do serviço de recomendação de rotas para usuários do RecRoute com o *middleware*

Com a utilização do UbiMid buscou-se extrair o serviço de ordenação de rotas do RecRoute, para que ele pudesse ser disponibilizado para outras aplicações e, diferentemente do que acontecia antes, onde as informações eram simuladas, foi realizada a integração com o Gerador de Rotas (desenvolvido em .NET) e passou-se a adquirir as informações do cenário através de fontes de contexto externas, dando origem aos componentes de enriquecimento de contexto presentes na arquitetura, proporcionando uma maior simplicidade, flexibilidade e transparência quanto à aquisição de novas informações pelo *middleware*.

Antes podia-se dizer que um serviço estava associado a um único recurso com um fluxo bem definido. Com a utilização do *middleware* um serviço pode estar associado a vários recursos (semelhantes, de preferência, para manter uma semântica no processo de requisição) ou mesmo a um único recurso, porém com várias alternativas de fluxos de execução internos. Cabe ao roteador de requisições, baseado no contexto, rotear a mensagem para o fluxo lógico ou recurso correspondente.

As informações contextuais, que são específicas do ambiente ao qual usuário está inserido, passaram a ser obtidas dinamicamente e em tempo real pelo *middleware* sem precisar que a aplicação interfira no processo, seja perguntando ao usuário (forma mais rápida e fácil), seja ela indo em busca da informação. O clima, um dos componentes do cenário, além de ser obtido em tempo real pelo componente de enriquecimento de contexto, passou a fornecer informações mais precisas, ou seja, além de nublado, ensolarado e chuvoso, passou a fornecer mais uma possibilidade de valor que seria “céu limpo”, pois à noite pode não estar nublado, nem chuvoso, muito menos ensolarado. Um novo valor também foi disponibilizado para as condições temporais (antes só existiam os valores manhã, tarde ou noite). Dessa forma, foi incorporado o valor “madrugada”. Ainda mais valores podem ser facilmente considerados, tanto para as

condições climáticas quanto para as condições temporais, para que a execução da ordenação fique ainda mais precisa. Basta apenas refazer o treinamento do algoritmo de classificação para que ele passe a aceitar novos valores para esses atributos.

A descrição do funcionamento deste serviço é feita detalhadamente na Seção 4.5.1. Nessa mesma seção é mostrado o diagrama de sequência do serviço na Figura 4.9 que é equivalente à interface visual do serviço implementado utilizando o Mule Studio representado na Figura 5.4.

A Figura 5.5 mostra algumas das diferentes formas de passagem dos parâmetros que podem ser enviados ao *middleware* em substituição aos parâmetros mostrados na Figura 5.3. Com a utilização do *middleware*, o cenário passou a ser facultativo, uma vez que é considerado um elemento contextual e caso não seja informado, o *middleware* faz a aquisição dessa informação de forma dinâmica e em tempo real. Contudo, informações relativas às rotas continuam sendo necessárias. Elas podem ser informadas (Figura 5.5 (a)), ou podem ser geradas a partir das coordenadas geográficas (Figura 5.5 (b)) ou através do endereço (Figura 5.5 (c)) de origem de destino. O resultado dessas requisições são as rotas obtidas a partir da localização informada, bem como sua ordenação. Caso a aplicação cliente já detenha as informações relativas as rotas, o *middleware* responde apenas a lista com a classificação das rotas.

Antes a requisição era feita passando-se 801 caracteres (Figura 5.3), agora a mesma requisição pode ser feita passando-se apenas 201 (Figura 5.5 (b)) ou 208 (Figura 5.5 (c)) caracteres, uma redução de cerca 75%. Essa redução ocorreu devido à transferência de responsabilidade da aplicação para o *middleware*. Antes as informações relativas ao cenário e às características das rotas teriam que ser adquiridas e processadas pela aplicação móvel, para então serem passadas como parâmetro para o serviço. Agora, quem faz essa tarefa de aquisição e processamento é o próprio *middleware*, diminuindo a complexidade de utilização dos serviços. É importante lembrar que, agora, quem faz a aquisição das informações contextuais é um servidor (onde o *middleware* está hospedado) que utiliza Internet de banda larga e que possui uma elevada capacidade de processamento, se comparado ao processamento feito em um dispositivo móvel e utilizando a Internet de uma rede 3G, por exemplo.

Outra melhoria e diferencial da utilização do *middleware* é a presença de um componente de qualidade e entrega de serviço. No caso do serviço de ordenação de rotas, esse componente é responsável por verificar a disponibilidade e medir o tempo de resposta dentre uma coleção de serviços responsável por fornecer as condições climáticas para a composição do cenário contextual do usuário. Após a verificação dos serviços disponíveis, a requisição é roteada para o serviço com o menor tempo de resposta, podendo variar de requisição para requisição. Essa redundância de serviços é muito importante, porque se um serviço estiver indisponível, a informação necessária para execução do algoritmo pode ser adquirida por um outro serviço semelhante ou equivalente, garantindo o funcionamento correto e a resposta para a aplicação consumidora.

Como resultado da avaliação da arquitetura, tem-se uma lista de requisitos não funcionais em forma de atributos de qualidade conforme a norma NBR ISO/IEC 9126-1 (algumas das

definições dadas para os itens abaixo foram extraídas de BASS; CLEMENTS; KAZMAN (1998)) que são atendidos, são eles:

- Reusabilidade: é a capacidade de um programa ou partes de um programa poderem ser reusadas por outras aplicações. Essa característica é evidente, possibilitada através da criação de *web services* pelo *middleware*. Antes o serviço ou funcionalidade era provido apenas por uma e para uma aplicação, agora ele é disponibilizado para qualquer aplicação conectada à Internet, independentemente da plataforma em que a aplicação foi desenvolvida.

- Funcionalidade: é a capacidade do sistema fazer o trabalho para o qual foi criado, ou seja, para execução de um serviço é necessário que todos ou a maioria dos componentes estejam funcionando para que a tarefa possa ser concluída. Para auxiliar essa característica, o UbiMid provê redundância de serviços internos através do componente de qualidade, que verifica quais serviços de aquisição estão funcionando e seleciona aquele com o menor tempo de resposta. No caso do serviço de ordenação, para aquisição de informações relacionadas ao cenário, caso elas não tenham sido informadas, como as informações climáticas, o UbiMid pode buscar essa informação de diversos serviços de clima disponíveis na Internet como, por exemplo, o *World Weather Online* [11], e a encaminha para o serviço de ordenação que requer essa informação para execução do serviço.

- Manutenibilidade ou modificabilidade: é a capacidade ou a facilidade de um software ser modificado de forma rápida e eficaz, ou seja, com baixo custo. Modificações podem ser correções, melhorias, adaptações, mudanças de ambientes e requisitos funcionais ou não funcionais. A arquitetura do UbiMid facilita a manutenção por causa da sua natureza componentizada. Caso seja adicionado um novo parâmetro e ele possa ser tratado como um elemento contextual, basta adicionar um novo componente de enriquecimento de contexto responsável por tratar daquele elemento específico e não precisa mexer em mais nada no fluxo do programa como mostra a Figura 5.4.

- Desempenho: refere-se à capacidade de resposta de um sistema, dessa forma, pode ser medida pelo tempo necessário para a resposta de um evento ou um conjunto de eventos. No caso, com a utilização do UbiMid, o processamento e o gerenciamento do contexto passou a ser todo feito no servidor. Este, por sua vez, possui uma maior capacidade de processamento e memória que, de forma geral, permite a manutenção ou mesmo diminuição no tempo de resposta dos serviços. Estima-se que o tempo extra gasto pelos serviços que utilizam contexto por meio da utilização de componentes, que verificam serviços externos de onde as informações complementares serão buscadas,

[11] http://www.worldweatheronline.com/

acaba sendo compensado com a aquisição e o processamento das informações feito no dispositivo móvel.

- Portabilidade: é a capacidade do sistema rodar em diferentes ambientes computacionais. Esses ambientes computacionais dizem respeito tanto a software quanto a hardware ou uma combinação de ambos. Se a portabilidade de um sistema requerer mudanças, então a portabilidade se torna um tipo especial de modificabilidade. No caso do UbiMid, como ele disponibiliza seus serviços por meio da internet, qualquer aplicação conectada à mesma pode acessá-los. Além disso, ele pode ser hospedado em diferentes sistemas operacionais como: Linux, Mac ou Windows.

- Variabilidade ou extensibilidade: é a capacidade da arquitetura poder ser expandida ou modificada de forma a produzir novas arquiteturas. Essa modificação da arquitetura pode ser feita em tempo de compilação (modificando parâmetros de compilação), em tempo de execução (negociação de protocolos em tempo real), em tempo de codificação (codificar um drive de um novo dispositivo) ou em tempo de construção (incluindo e excluindo componentes). O UbiMid possui uma arquitetura de componentes onde novos componentes podem ser adicionados ou removidos facilmente. Como os componentes possuem um fraco acoplamento, a exclusão de um componente ou a inserção de um novo não interfere de forma significativa no processo.

Foram apresentadas algumas características que o *middleware* proposto possui em forma de atributos de qualidade e, na próxima seção, serão apresentadas algumas considerações sobre os assuntos discutidos.

5.4 Considerações finais

Neste capítulo foram apresentadas as tecnologias utilizadas para o desenvolvimento do UbiMid, bem como um cenário real de utilização do mesmo por outros trabalhos de mestrado no âmbito do projeto UbiBus. A utilização de ESBs para o desenvolvimento de *middleware* ainda é uma atividade recente. A escolha de uma boa tecnologia de desenvolvimento ajuda bastante na concepção de ideias, além de facilitar a implementação da arquitetura proposta.

A utilização de um *middleware* por si só já influencia em alguns atributos relacionadas à distribuição de sistemas. Contudo, o foco deste trabalho foi mostrar como a utilização de contexto pode ajudar na flexibilidade, no desacoplamento e na diminuição da complexidade de comunicação e consumo dos serviços entre as aplicações clientes e os serviços desejados.

A presença de componentes arquiteturais sensíveis ao contexto, como o roteador de requisições e o componente de qualidade de serviço ajudam a provê a garantia de entrega durante a busca de serviços disponíveis bem como o balanceamento de carga direcionando as requisições para os serviços mais apropriados.

A Tabela 5.1 resume alguns atributos e características que podem ser evidenciadas no cenário de aplicação citado anteriormente sem a utilização do *middleware* e depois com a utilização do *middleware*.

Tabela 5.1: Comparação entre a utilização e não utilização do *middleware*.

Atributos e características	Serviço de ordenação	
	Sem o *middleware*	Com o *middleware*
Reusabilidade	Não	Sim
Provisionamento de contexto	Não	Sim
Tamanho médio da mensagem	850 caracteres	215 caracteres
Manutenibilidade/Modificabilidade	Baixa	Alta
Variabilidade/Extensibilidade	Baixa	Alta
Complexidade de utilização	Alta	Baixa
Flexibilidade	Não	Sim

No próximo capítulo apresentaremos as conclusões deste trabalho, bem como sugestões de trabalhos futuros.

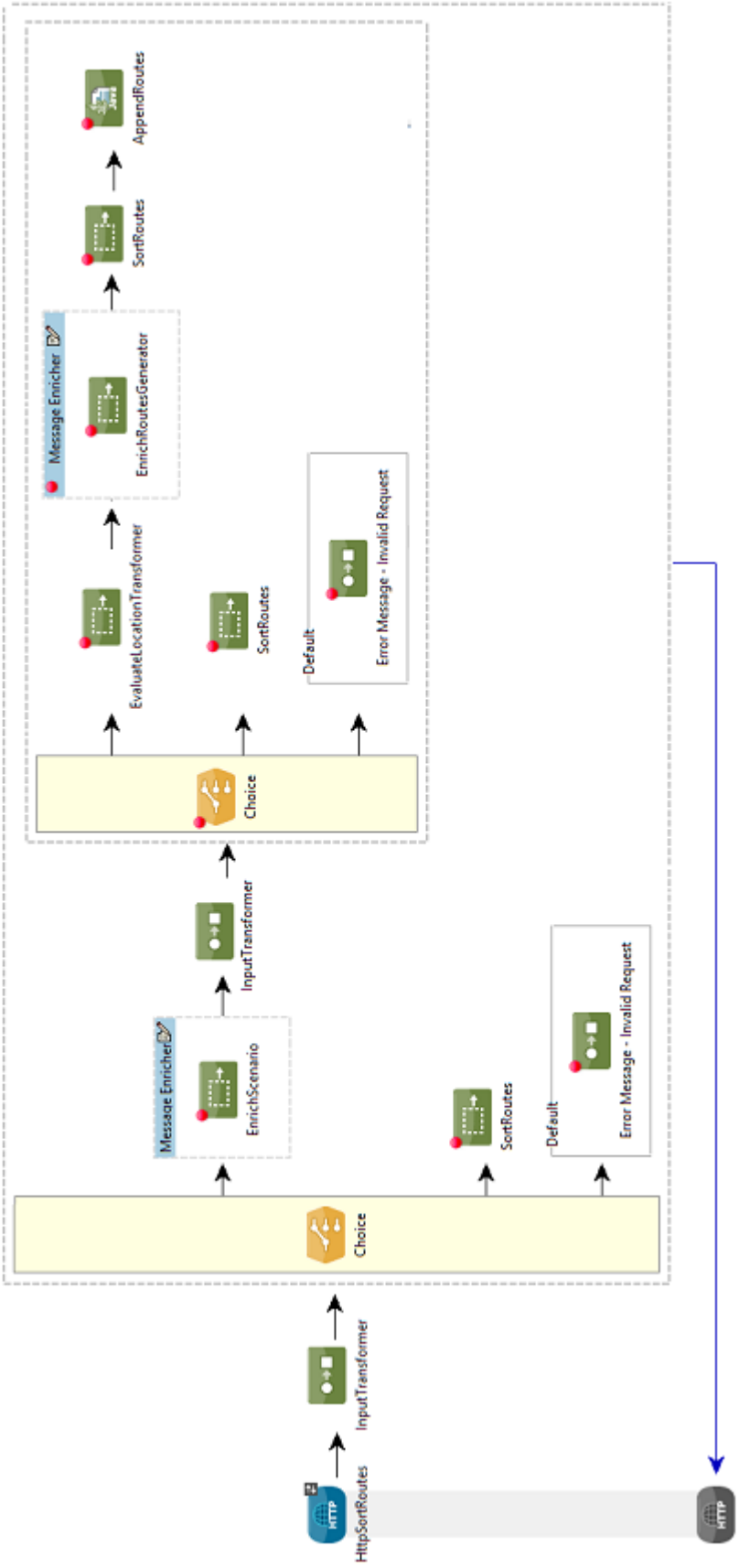

Figura 5.4: Interface gráfica da implementação do serviço de Geração e Ordenação de Rotas do RecRoute.

a) {"rotas":[{"andar_total":"170","tempo_espera":"20","tem_troca":"nao","tempo_total":"45","distancia_total":"4800","tem_acessibilidade":"sim","preco_total":"2.3"},{"andar_total":"520","tempo_espera":"20","tem_troca":"sim","tempo_total":"60","distancia_total":"6000","tem_acessibilidade":"sim","preco_total":"2.3"},{"andar_total":"520","tempo_espera":"30","tem_troca":"nao","tempo_total":"55","distancia_total":"5000","tem_acessibilidade":"nao","preco_total":"2.3"},{"andar_total":"480","tempo_espera":"5","tem_troca":"nao","tempo_total":"30","distancia_total":"5000","tem_acessibilidade":"nao","preco_total":"2.3"}],"preferencias":{"aceita_troca_onibus":"sim","qual_deficiencia":"nenhuma","tipo_busca":"tempo","espera_parada":"10","quanto_andar":"30"}}

b) {"origem":"-7.133787,-34.88181","destino":"-7.118395,-34.878325","preferencias":{"aceita_troca_onibus":"sim","qual_deficiencia":"nenhuma","tipo_busca":"tempo","espera_parada":"10","quanto_andar":"30"}}

c) {"origem":"avenida doutor João da mata","destino":"almeida, joão pessoa","preferencias":{"aceita_troca_onibus":"sim","qual_deficiencia":"nenhuma","tipo_busca":"tempo","espera_parada":"10","quanto_andar":"30"}}

Figura 5.5: Diferentes formas de chamar o mesmo serviço utilizando o *middleware*

6

Conclusão e trabalhos futuros

O número aplicações sendo desenvolvidas na área de sistemas inteligentes de transporte vem crescendo bastante nos últimos anos, devido, principalmente, ao acelerado crescimento da Tecnologia da Informação e Comunicação. No Brasil, a aproximação da realização de eventos internacionais como a Copa do Mundo de 2014 e as Olimpíadas de 2016 vem contribuindo bastante com esse processo. Já mundialmente, aliado à área de sistemas inteligentes de transporte está o conceito de cidades inteligentes que está bastante em alta devido ao elevado crescimento populacional urbano, ocasionado graças a fatores relacionados a oportunidades de emprego, qualidade de vida, dentre outros. Dessa forma, percebe-se que há uma grande demanda para o desenvolvimento de soluções nessa área.

Uma alternativa para acelerar o desenvolvimento de aplicações nessa área é a implementação/utilização de um *middleware*. Além disso, cada vez mais aplicações e sistemas têm levado em consideração o contexto do usuário para a inferência dos mais diversos tipos de informações, tanto para auxiliar na execução de tarefas pelos usuários como para fazer proveito próprio dessas informações. Dessa forma surgem os *middleware* sensíveis ao contexto. Esses *middleware* podem ser compostos tanto por serviços sensíveis ao contexto como possuir um comportamento diferenciado, dependendo do contexto da requisição.

É importante lembrar que o desenvolvimento de um *middleware* é uma atividade complexa, pois envolve um conjunto de tecnologias diferentes e é bastante propensa a ocorrência erros arquiteturais. Este trabalho teve por objetivo apresentar uma arquitetura de *middleware* sensível ao contexto extensível que permite que diversas aplicações usem seus serviços com baixa complexidade, de forma flexível, transparente e com baixo acoplamento, diminuindo complexidade a de desenvolvimento e integração por parte das aplicações consumidoras.

A solução proposta difere dos outros trabalhos relacionados principalmente na forma de utilização de contexto. Em vez de ter apenas serviços de inferência sensíveis ao contexto, que é o mais comum, o *middleware* também possui componentes internos sensíveis ao contexto. Esses componentes são compostos pelos roteadores, enriquecedores, transformadores e pelos componentes de qualidade. E além da utilização do contexto, o UbiMid foi desenvolvido com uma tecnologia bastante inovadora que facilita o desenvolvimento e o entendimento do que foi

desenvolvido.

6.1 Contribuições

A principal contribuição deste trabalho foi a proposição de uma arquitetura de desenvolvimento de um *middleware* sensível ao contexto voltado para aplicações na área de sistemas inteligentes de transporte. O UbiMid mostrou-se uma arquitetura interessante, abstraindo os desenvolvedores de boa parte das complexidades existentes na implementação desse tipo de aplicativo. Essa diminuição quanto à complexidade ocorreu graças à utilização dos componentes que compõem a sua arquitetura. Outras contribuições do UbiMid:

- Roteamento de mensagens sensível ao contexto, onde o usuário faz a requisição a um determinado tipo de recurso e o *middleware* encaminha a requisição ao serviço correspondente.
- Suporte à execução dos serviços de forma distribuída e paralela. Garantindo que o sistema seja construído em uma infraestrutura escalar.
- Suporte à extensão e manutenibilidade dos componentes, graças à sua natureza modular.
- Suporte à garantia de entrega dos serviços, graças à existência de componentes de qualidade que proveem redundância durante o processo de aquisição e enriquecimento de mensagens na busca de informações contextuais em fontes de dados externas.
- Suporte a flexibilidade de passagem de parâmetros por meio da utilização de componentes de transformação sensíveis ao contexto, que tem o objetivo de adaptar os dados recebidos, caso necessário, de forma que os serviços requisitados possam utilizá-los.

6.2 Dificuldades encontradas

Dentre as dificuldades encontradas durante a realização deste trabalho, as mais relevantes foram:

- Apesar do *middleware* possuir serviços básicos, que tinham como resposta dados armazenados em um banco de dados, os serviços mais relevantes e mais complexos são os serviços com a utilização de contexto que foram desenvolvidos por vários alunos relacionados ao projeto UbiBus. A integração e manipulação desses serviços foi bastante difícil e demandou muito tempo, pois eles foram desenvolvidos por terceiros

de forma independente, gerando problemas, principalmente, relacionados à comunicação. Além disso, foram desenvolvidas utilizando diversos tipos de tecnologias como: Python, Java e .NET, sem nenhuma documentação.

- A tecnologia utilizada apesar de ter bastante potencial, ainda é pouco utilizada. Por isso, ainda possui poucas informações disponíveis na Internet, tornando difícil o início do desenvolvimento e a sua melhor utilização.

6.3 Limitações do trabalho

Conforme dito anteriormente, como os parâmetros para execução do algoritmo de ordenação das rotas do RecRoute foram todos simulados, não foi possível realizar a comparação do desempenho entre a execução do algoritmo pelo *middleware* com a utilização do contexto e a execução do algoritmo pelo RecRoute.

Apesar de existirem várias outras aplicações no projeto na mesma área que poderiam fazer proveito dos serviços disponibilizados para a derivação e criação de novos serviços, apenas o RecRoute utiliza o *middleware*.

6.4 Trabalhos futuros

Os trabalhos futuros propostos nessa dissertação têm como objetivo tornar o *middleware* mais robusto, com mais funcionalidades e serviços para que ele deixe de ser um protótipo e possa, realmente, ser utilizado por aplicações reais na área de transporte público, para podermos avaliar outras características como disponibilidade e confiabilidade, além de tentar estimular o desenvolvimento de novos *middleware* com essa mesma arquitetura em outras áreas do conhecimento.

Alguns pontos a serem considerados:

a) Estudar a possibilidade de extrair novos serviços providos por outras aplicações desenvolvidas no âmbito do projeto UbiBus com intuito de disponibilizá-los para que outras aplicações também possam utilizá-los, promovendo o reuso de código e estimulando a comunicação entre as aplicações.

b) Melhorar os serviços existentes e propor novos serviços a partir deles. Para melhorar, por exemplo, no caso do serviço de análise de mensagens, além de informações do Twitter, poderia ser feita a extração de novas mensagens a partir de outras redes sociais, como Facebook e Google+, diversificando, dessa forma, as fontes de dados e aumentando o volume de mensagens e informações que possam vir a ser relevantes para o usuário. Com o objetivo da criação de novos serviços a partir dos serviços existentes, o gerador de rotas poderia levar em consideração informações analisadas

em tempo real a partir de informações das redes sociais para propor rotas para os usuários do transporte público.

c) Implementar mecanismos de segurança para que o contexto de um usuário não possa ser acessado por outros usuários indevidamente, ou mesmo que as informações que trafegam na rede não sejam interceptadas ou traduzidas por outras aplicações indevidamente. Dessa forma, o *middleware* deve ter a capacidade de não autorizar o acesso a serviços ou encriptar as informações que estejam trafegando pela rede.

d) Prover novos componentes genéricos que possam ser facilmente acoplados a novos serviços ou mesmo exportados para que outros *middleware* de outros domínios também possam utilizá-los.

e) Apesar do *middleware* estar hospedado no servidor da UFPE, a ideia é que ele seja disponibilizado num servidor mais robusto e estável na nuvem, de forma que o mantenedor tenha mais autonomia sobre o servidor e possa manipulá-lo de forma independente da instituição.

Referências

AMATO, F. **Governo prorroga IPI menor para carros, linha branca e móveis**. Último acesso em: 06/12/2013, http://g1.globo.com/economia/noticia/2012/08/mantega-anuncia-prorrogacao-de-reducao-de-ipi-para-linha-branca.html.

ASTLEY, M.; STURMAN, D.; AGHA, G. **Customizable middleware for modular distributed software**. Communications of the ACM, v. 44, n. 5, p. 99-107, 2001.

BAI, X. et al. **Dresr**: dynamic routing in enterprise service bus. In: e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on. IEEE, 2007. p. 528-531.

BAKKEN, D. **Middleware**. In Encyclopedia of Distributed Computing, Kluwer Academic Press, 2003.

BARBOSA, A. C. P. **Middleware para Integração de Dados Heterogêneos Baseado em Composição de Frameworks**. 2001. Tese (Doutorado em Ciência da Computação) — Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Practice**. Addison-Wesley Professional, Reading, Mass, 1998.

BAZIRE, M.; BRÉZILLON, P. **Understanding context before using it**. In: 5th International and Interdisciplinary Conference, CONTEXT-05, v. LNAI 3554, p. 29-40, Springer Verlag, Paris, France, 2005.

BERNSTEIN, P. A. **Middleware**: a model for distributed system services. ACM, 1996, p. 86–98.

BORGIANI, F. S. M. **Pilgrim**: um sistema para geração e classificação de rotas de Ônibus sensível ao contexto. 2013. Dissertação de mestrado — Universidade Federal de Pernambuco - Centro de Informática (CIn).

BRAY, T. et al. **Extensible markup language (XML)**. World Wide Web Journal, v. 2, p. 27-66, 1997.

BRUNEO, D.; PULIAFITO, A.; SCARPA, M. **Mobile Middleware**: definition and motivations. The Handbook of Mobile Middleware, Bellavista, P. Corradi, A., Eds Auerbach, Boca Raton, 145-167, 2007.

CALDAS, L. R. R.; VIEIRA, V. **Desenvolvimento de Uma Solução Sensível ao Contexto Como Suporte a Um Sistema de Transporte Público Inteligente**. Trabalho de Conclusão de Curso. Graduação em Ciência da Computação - Universidade Federal da Bahia. Salvador/BA, 2010.

COSTA, P. D. **Towards a services platform for context-aware applications**. Telematics, University of Twente, Enschede. Netherlands, August 2003.

DEY, A. K. **Understanding and using context**. Personal and ubiquitous computing, v. 5, p. 4-7, 2001.

EMMERICH, W. **Software engineering and middleware**: a roadmap. Proceedings of the Conference on The future of Software engineering. ACM, 2000.

EUGSTER, P. T. et al. **The many faces of publish/subscribe**. ACM Computing Surveys (CSUR), v. 35, n. 2, p. 114-131, 2003.

FAHY, P.; CLARKE, S. **CASS–a middleware for mobile context-aware applications**. Workshop on Context Awareness, MobiSys. 2004.

FIELDING, R. T. **Architectural styles and the design of network-based software architectures**. 2000. Tese (Doutorado em Ciência da Computação) — University of California.

FOLMER, E.; VAN GURP, J.; BOSCH, J. **Software architecture analysis of usability**. Springer, 2005.

GU, T.; PUNG, H. K.; ZHANG, D. Q. **A service-oriented middleware for building context-aware services**. Journal of Network and computer applications, v. 28, n. 1, p. 1-18, 2005.

HAN, J.; KAMBER, M.; PEI, J. **Data mining**: concepts and techniques. Morgan kaufmann, 2006.

KAZMAN, R. et al. **SAAM**: a method for analyzing the properties of software architectures. Proceedings of the 16th international conference on Software engineering. IEEE Computer Society Press, 1994.

KJÆR, K. E. **A survey of context-aware middleware**. In: Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering. ACTA Press, 2007. p. 148-155.

LIMA, V. G. de. **UbibusAnalysis**: uma ferramenta de interpretação de mensagens de trânsito com análise de sentimentos. Trabalho de Conclusão de Curso. Graduação em Ciência da Computação - Universidade Federal de Pernambuco - Centro de Informática (CIn). Recife/PE.

MARÉCHAUX, J.-L. **Combining service-oriented architecture and event-driven architecture using an enterprise service bus**. IBM Developer Works, p. 1269-1275, 2006.

MENG, J.; MEI, S.; YAN, Z. **Restful web services**: a solution for distributed data integration. Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on. IEEE, 2009.

MENGE, F. **Enterprise service bus**. In: Free and open source software conference. 2007.

MURALIDHARAN, K. et al. **mConnect**: a context aware mobile transaction middleware. Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on. IEEE, 2008.

NURSEITOV, N. et al. **Comparison of JSON and XML data interchange formats**: a case study. Computer Applications in Industry and Engineering (CAINE), v. 9, p. 157-162, 2009.

OLIVEIRA, M. **Frota de veículos cresce até 240% em oito anos nas maiores cidades do país @ONLINE**. Último acesso em: 18/02/2013, `http://g1.globo.com/Noticias/Carros/0,,MUL1352939-9658,00-FROTA+DE+VEICULOS+CRESCE+ATE+EM+OITO+ANOS+NAS+MAIORES+CIDADES+DO+PAIS.html`.

OLIVEIRA TITO, A. de. **RecRoute**: um sistema de recomendação de rotas de Ônibus baseado em informações contextuais dos usuários. 2013. Dissertação de mestrado — Universidade Federal de Pernambuco - Centro de Informática (CIn).

PAPAZAGLOU, M.; HEUVEL, W.-J. van-den. **Service Oriented Computing**: state-of-the-art and open research issues. IEEE Computer. v40 i11, 2003.

PEREIRA FILHO, J. G. et al. **Infraware**: um middleware de suporte à aplicações móveis sensíveis ao contexto. Infraware: Middleware Support for Context-Aware Mobile Applications). Proceedings of the: 24º 24th Brazilian Symposium on Computer Networks, Curitiba-PR, Brasil, 2006.

QUARTZ. **Quartz Enterprise Job Schedule @ONLINE**. Último acesso em: 09/04/2013, http://quartz-scheduler.org/overview.

RANGANATHAN, A.; CAMPBELL, R. H. **A middleware for context-aware agents in ubiquitous computing environments**. In: Middleware 2003. Springer Berlin Heidelberg, 2003. p. 143-161.

ROMÁN, M. et al. **A middleware infrastructure for active spaces**. IEEE pervasive computing, v. 1, n. 4, p. 74-83, 2002.

ROY, B.; GRAHAM, T. N. **Methods for evaluating software architecture**: a survey. School of Computing TR, v. 545, p. 82, 2008.

SALGADO, A. C. B. **UbiBus**: um sistema de transporte público inteligente, ubíquo e sensível ao contexto. [S.l.]: Universidade Federal de Pernambuco - Centro de Informática (CIn), 2010. Relatório Técnico.

SCHILIT, B. N.; THEIMER, M. M. **Disseminating active map information to mobile hosts**. Network, IEEE, v. 8, n. 5, p. 22-32, 1994.

SILVA, D. M. d. **Sistemas inteligentes no transporte público coletivo por ônibus**. 2000. Dissertação de mestrado — Universidade Federal do Rio Grande do Sul.

SILVA, F. A. et al. **Um Middleware para Provisionamento de Contextos para Redes Veiculares**. 31º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos SBRC 2013.

SOUSA, J. P.; GARLAN, D. **Aura**: an architectural framework for user mobility in ubiquitous computing environments. In: Software Architecture. Springer US, 2002. p. 29-43.

TANENBAUM, A. S.; VAN STEEN, M. **Distributed systems**: principles and paradigms. Upper Saddle River: Prentice Hall, 2002.

VIEIRA, V.; TEDESCO, P.; SALGADO, A. C. **Modelos e Processos para o desenvolvimento de Sistemas Sensíveis ao Contexto**. André Ponce de Leon F. de Carvalho, Tomasz Kowaltowski.(Org.). Jornadas de Atualização em Informática, p. 381-431, 2009.