UNILASALLE
CENTRO UNIVERSITÁRIO LA SALLE

RODRIGO BARCELOS DA SILVA

UMA ARQUITETURA PARA SISTEMAS DE NOTIFICAÇÃO

CANOAS, 2012

RODRIGO BARCELOS DA SILVA

UMA ARQUITETURA PARA SISTEMAS DE NOTIFICAÇÃO

Trabalho de conclusão do Curso de Ciência da Computação do Centro Universitário La Salle – Unilasalle. Canoas - RS

Orientador: Prof º. Me. : Abraham Lincoln Rabelo de Sousa

CANOAS, 2012

RODRIGO BARCELOS DA SILVA

UMA ARQUITETURA PARA SISTEMAS DE NOTIFICAÇÃO

Trabalho de conclusão aprovado como requisito parcial para obtenção do grau de bacharel em Ciência da Computação pelo Centro Universitário La Salle – Unilasalle.

Aprovado pela banca examinadora em 13 de Dezembro de 2012.

BANCA EXAMINADORA:

_____

Profº Me. Abraham Lincoln Rabelo de Sousa
Unilasalle

_____

Profº Me. Roberto Petry
Unilasalle

_____

Profº Me. Gustavo Passos Tourinho
Unilasalle

Dedico este trabalho de conclusão aos meus pais, cuja ajuda e incentivo tornaram possível alcançar esta conquista e aos amigos que estiveram ao meu lado durante o trajeto.

AGRADECIMENTOS

Agradeço em primeiro lugar a meus pais cujo apoio foi fundamental para realização desta conquista, ao meu orientador Abraham Lincoln Rabelo de Sousa por todos esses anos dividindo conhecimento e amizade dentro e fora de sala de aula, demonstrando sempre muita cordialidade e paciência.

Aos meus amigos de batalha, Pablo Marcel Parada por ajudado no desenvolvimento da arquitetura, ao Ricardo Arnoud por ter configurado as ferramentas de backup e rede. E a todos demais colegas que propiciaram momentos inesquecíveis e também compartilharam seu conhecimento durante esses anos de graduação.

Muito obrigado a todos.

“O verdadeiro homem mede a sua força, quando se defronta com o obstáculo.”
(Antoine de Saint-Exupéry)

RESUMO

Este trabalho tem como objetivo apresentar o conceito de cidades inteligentes, tendo um foco os ambientes sócio cultural. Neste conceito, temos uma investigação sobre os recursos requeridos para o desenvolvimento de uma infraestrutura arquitetural que apóie o desenvolvimento de aplicações, com o objetivo de também contextualizar uma aplicação para ser usada como caso de teste.

Palavras-chave: Cidades inteligentes. Colaboração. Problemas urbanos. Inovação urbana. Patrimônio cultural.

ABSTRACT

The objective of this paper is to introduce the concept of smart cities, focusing on social culture environments. With this concept, we're going to investigate the required resources to develop an arquitectural infrastructure which help the application development, in order to contextualize an application to be used as a test case.

Keywords: Smart cities. Collaboration. Urban problems. Urban innovation. Cultural heritage.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

TIC	Tecnologia da informação e comunicação
SOA	*Service-oriented architecture*
GPS	*Global Positioning System*
WiFi	Wireless Internet Access
iOS	Sistema operacional da Apple
ESB	Enterprise Service Bus
RFID	*Radio-frequency identification*
OSB	*Oracle Service Bus*
BPM	*Business process management*
BPEL	*Business Process Execution Language*

SUMÁRIO

1 INTRODUÇÃO

O conceito cidades inteligentes surgiu na década de 90 com o objetivo de definir novas políticas de planejamento urbano, devido aos problemas gerados pelo crescimento populacional como poluição do ar, preocupação com a saúde humana, trânsito elevado nas estradas e entre outros problemas.

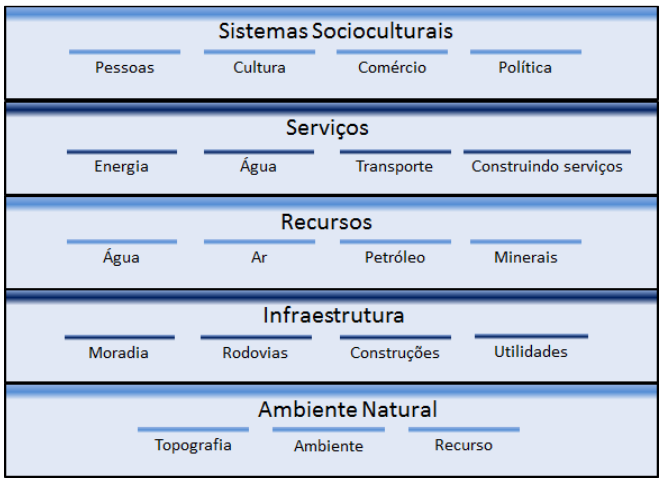
Uma cidade inteligente tem o foco em economia com envolvimento de pessoas e do governo, também possui o objetivo de auxiliar a preservação dos recursos naturais e construir um ambiente colaborativo dentro da cidade, com cidadãos mais independentes e conscientes. Um ambiente capaz de monitorar e integrar a sua infraestrutura critica como estradas, metros, aeroportos, comunicações, água, energia e até construções (CHOURABI, et al., 2012). Cidade Inteligente é um conceito emergente que possui várias definições na literatura científica especializada. Entretanto, todas as definições convergem para o sentido do uso da Tecnologia da Informação e Comunicação (TIC) no tratamento de problemas urbanos. A idéia geral é usar toda a inovação tecnológica existente em prol da sociedade via produtos e serviços que agreguem valor ao cidadão e proporcionem melhorias na qualidade de vida

Com os avanços no domínio da TIC, atualmente dois bilhões de pessoas utilizam a internet e mais de cinco bilhões são assinantes de telefonia móvel. Temos 30 bilhões de etiquetas RFID incorporadas em nosso mundo e um bilhão de transistores por pessoa. Essa convergência de sensoriamento difuso, o acesso redes e a mobilidade, com computadores menores e mais rápidos ficaram mais fácil acessar os sistemas de forma inteligente e capacitando as pessoas no acesso da informação. (NAPHADE et al., 2011).

Entre 2002 a 2005, grandes empresas de tecnologia se interessaram por cidades inteligentes, sendo algumas delas CaldWell[i] , Cisco[ii] , IBM[iii] e Simens[iv] . Através do uso da informação as cidades esperam criar uma visão, prover inovação, gerar oportunidades de emprego e aumentar a qualidade de vida da população. (HARRISON; DONNELLY, 2011).

O conceito cidade inteligente pode ser dividido em subáreas conforme ilustrado na figura abaixo.

Figura 1. Uma visão simplificada do modelo urbano



Fonte: Theory of Smart Cities adaptado de (Harrison e Donnelly, 2011))

No ambiente natural temos o foco na topografia, recursos naturais, geologia e entre outros recursos que envolvem o meio ambiente.

A infraestrutura tem como foco auxiliar em construções de rodovias, utilização da terra, construções entre outros recursos que envolvem a infraestrutura.

Os recursos representam materiais que se originam e depois retornam ao ambiente natural, após passar por alguns processos de refinação e recomposição.

A subárea serviços possui os subitens relacionados a abastecimento de energia, abastecimento de água, transporte público e entre outros relacionados a esse tipo subárea.

Os sistemas socioculturais possui o foco relacionado a pessoas, cultura, comercio, política e entre outros contextos relacionados a essa subárea. (HARRISON; DONNELLY, 2011)

Atualmente existem cidades que estão realizando pesquisa dentro deste conceito de cidades inteligentes, temos como exemplo o Rio de Janeiro no Brasil, Dubuque que está localizado em Lowa nos Estados Unidos da América, Bornholm na Dinamarca, Porto Alegre no Brasil e Seul na capital da Corea do Sul, junto á empresas incluindo as citadas no parágrafo anterior. (NAPHADE et al., 2011)

Segundo (DIGITAL, 2012), Porto Alegre cidade localizada estado do Rio grande do Sul, foi uma das cidades selecionadas pela IBM em um projeto chamado *Executive Service Corps* (ESC). A empresa vai fornecer um time de executivos e especialistas para auxiliar no desenvolvimento de projetos, análises e estudos dos principais problemas indicados pelo município, tornando a cidade de Porto Alegre um lugar ainda melhor para se viver e trabalhar.

No caso do Rio de janeiro existe um foco na questão dos desastres naturais, gerenciar o recurso de emergência na cidade e eventos planejados de importância nacional. Tendo em vista a copa do mundo em 2014 e as olimpíadas em 2016, surgi à necessidade de um melhor gerenciamento no transporte publico, conforme for avançando as pesquisas e do desenvolvimento dos sistemas para essas prioridades, trás consigo uma oportunidade de sistemas para grandes construções e possivelmente envolvendo energia, água e outros recursos. (NAPHADE et al., 2011)

Em Bornholm na Dinamarca existe um desenvolvendo sistemas para integrar veículos com grandes fornecedores de energia, tendo em vista o crescimento dos carros elétricos, com isso a emissão de carbono vai ter uma redução e um aumento na gestão de energia. (NAPHADE et al., 2011).

Em Seul, na capital da Corea do Sul a banda larga de alta velocidade está disponível em quase todos os limites da cidade e com baixo custo, com isso acelerou o crescimento econômico do país e se tornou o mais eficiente governo eletrônico. Os maiores benefícios é a transparência da informação na divulgação das propostas do governo e em virtude disto à corrupção teve um declínio significativo. (NAPHADE et al., 2011).

Dubuque é a pioneira em colocar a civilização em meio à transformação de uma cidade inteligente, após uma crise econômica seu foco de crescimento foi em TIC, criou um modelo de sistema sustentável em 2006 com três temas relacionados às cidades inteligentes, como propósito econômico, oscilação sociocultural e integridade ambiental. No ano de 2009 em parceria com a IBM tornou-se um laboratório vivo para gestão da sustentabilidade, onde há um monitoramento e o racionamento de água e energia elétrica envolvendo residências, utilizando métodos inteligentes que oferece orientações e incentivos para os moradores com objetivo de aperfeiçoar o consumo individual. (NAPHADE et al., 2011).

Dentro deste conceito de cidades inteligentes, focando no ambiente sociocultural, temos as pessoas dos centros urbanos colaborando e trocando informações entre elas. Essa informação pode ser processada por outros sistemas e de certa forma auxiliar na tomada de decisão de uma determinada situação dentro da cidade. (NAPHADE et al., 2011).

O foco deste trabalho esta relacionado aos sistemas socioculturais apresentados na Figura 1, desta forma, temos que abranger sistemas relacionados a pessoas, cultura, comercio, política e entre outros contextos relacionados, bem como auxiliar outras camadas do conceito de cidade inteligente, onde podemos ter diferentes aplicações colaborando informações de diferentes camadas.

1.1 Problema

O problema de pesquisa explorado por esse estudo está relacionado à necessidade de investigação sobre os recursos requeridos para o desenvolvimento de uma infraestrutura arquitetural que apóie o desenvolvimento de aplicações a esse conceito de cidades inteligentes com foco na camada sociocultural apresentada na Figura 1. Assim, vislumbram-se algumas questões de pesquisa:

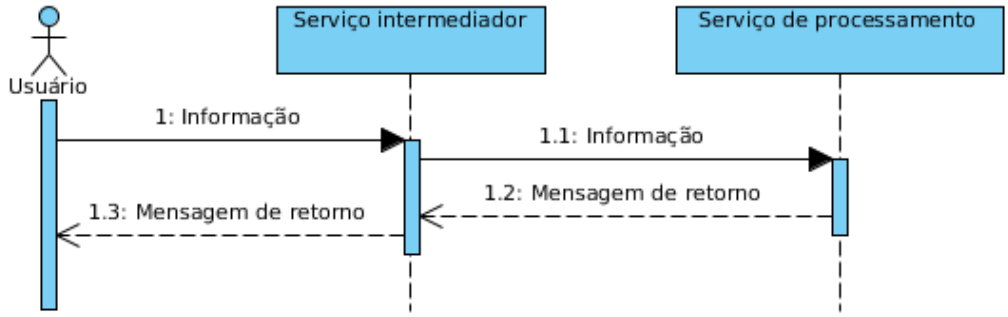
a) Quais os requisitos básicos?
b) Quais os componentes de software podem ser utilizados?
c) Como esses componentes se relacionam/cooperam para a execução da aplicação?
d) Qual estilo arquitetural o sistema deve ser utilizado?

1.2 Objetivos

O objetivo central deste trabalho é desenvolver uma infraestrutura arquitetural para servir como ambiente laboratorial de desenvolvimento de aplicações móveis, com foco no aspecto sociocultural da área de cidades Inteligentes. Em seguida, aplicar essa arquitetura no desenvolvimento de uma aplicação piloto, um sistema de notificações, visando à avaliação da viabilidade técnica e a observação das suas potencialidades e limitações.

Como objetivo especifico desta arquitetura, destacam-se: agregar diferentes serviços que serão responsáveis pelo processamento dos dados; e a possibilidade de gerenciar informações. Dentro deste contexto aplicação do cliente vai requisitar determinada informação, onde vai ser encaminhada para um serviço intermediador e ela vai requisitar para um determinado serviço processar, após o processo o serviço retorna a informação solicitada para o controle e ele envia para o cliente. Abaixo uma figura ilustrando esse processo.

Figura 2. Processo de execução



Fonte: Autoria própria

1.3 Justificativa

Criar uma arquitetura de controle de serviços possibilitando a centralização da informação, o acesso facilitado à informação, a agregação de serviços conforme a necessidade e que seja uma ferramenta rápida e confiável. Utilizando um modelo de arquitetura como essa, é possível remover à responsabilidade da aplicação cliente processar os dados e solicitando para algum serviço realizar esta tarefa, como um *Web Service*[v] ou até enviar a informação para alguma base dados aonde um determinado sistema, usado por um departamento responsável pela analise, acesse as informações para serem analisadas.

Levando em consideração que o estudo de caso será uma aplicação para dispositivos móveis, uma arquitetura capaz de remover a responsabilidade do processamento e armazenamento da informação se torna um estudo importante a ser realizado.

2 BASES TEÓRICAS

Segundo (NAPHADE et al., 2011), atualmente, as cidades enfrentam muitos problemas infraestruturais resultantes do crescimento urbano, que se refletem com um aumento da procura por recursos como água, energia, transporte, saúde, educação e segurança. As cidades vêem buscando cada vez mais tecnologias de informação e comunicação para apoiar estratégias sustentáveis, de forma que melhorem a gestão de sua infraestrutura e recursos para atender melhor a população.

Além de existir essas áreas que foram descritas anteriormente dentro do conceito de cidades inteligentes temos uma iniciativa voltada para indústrias, segundo (Bakici, Almirall, & Wareham, 2012) em um estudo de caso na cidade de Barcelona na Espanha foi definido uma subárea chamada *Smart Disctrict*, que possui o objetivo de incentivar a colaboração entre as indústrias e o desenvolvimento urbano.

Porém, quando podemos dizer que uma cidade é inteligente? Conforme (CARAGLIU, BO; NIJKAMP, 2009), uma cidade será inteligente a partir do momento que a população que nela vive aprender a adaptar e a inovar. Elas precisam ser capazes de usar a tecnologia em seu benéfico, deixar a mão de obra do trabalhador mais qualificada.

De acordo com (SIE, HOL.; CHIA, 2011), existem duas maneiras de prover uma cidade inteligente: as cidades emergentes; e as cidades remodeladas. As cidades emergentes sãos as que têm o foco na sustentabilidade, não havendo desperdícios. Já o conceito de cidades remodeladas é mais aplicado a países desenvolvidos que focam a infraestrutura urbana por meio da utilização de sensores e tecnologias inteligentes, visando aperfeiçoar o consumo de recursos naturais e melhorar a qualidade de vida da população.

Levando em consideração o envolvimento da população dentro do conceito em estudo, existe a importância da colaboração da informação que deve ser estimulada, envolve permitir que o cidadão participe ativamente de todo o processo realizado na cidade seja pela notificação de eventos ou correção/atualização de informações. Conforme (KIRWAN, 2011), com os problemas urbanos que estão ocorrendo atualmente, há a possibilidade de fazer com que a haja colaboração entre a

sociedade para auxiliar na solução dos problemas que estão ocorrendo e também abrir novos caminhos para o conhecimento.

Ainda sobre a importância da participação dos cidadãos, (NAPHADE et al. 2011) relata que em Dubuque, 50% dos custos de consertar vazamentos de água em domicílios são feitos por usuários, que relatam os vazamentos e acompanham os reparos. Seu feedback perspicaz sobre o consumo de água aumentou a conservação em 10%. Da mesma forma, Seul está usando sua conectividade com a internet em toda a cidade com objetivo de incentivar o uso do transporte público, fornecendo informações on-line para os condutores sobre os descontos de seguro, custo reduzido de estacionamento e redução de impostos para aqueles que deixam seus carros em casa um dia útil por semana.

No desenvolvimento de sistemas dentro das cidades inteligentes Harrison e Donnelly (2011) defendem que é importante envolver e se preocupar com as pessoas que se beneficiarão do novo serviço, focando nos aspectos relacionados à atratividade e o acesso, visando que a população desperte o interesse em utilizar determinada ferramenta o serviço tecnológico a ser disponibilizado. Além disso, Rotti e Torensend (2011) enfatizam que adotar uma visão de baixo para cima, uma visão partindo do público alvo, onde o contexto são as pessoas, seria fundamental para o desenvolvimento de uma cidade inteligente.

2.1 Trabalhos Relacionados

Analisando a arquitetura que está sendo proposta neste trabalho, vai ser utilizado o conceito SOA como base, que segundo (ALI, 2012) é um conjunto flexível de princípios de projeto usados durante as fases de desenvolvimento de sistemas. A implantação de uma arquitetura baseada em SOA irá fornecer um conjunto integrado de serviços que podem ser usados dentro de áreas negócios que tem sues domínios diferentes.

Já em relação à aplicação piloto que vai ser desenvolvida, temos como base da o sistema *Cambridge iReports*[vi]. Ela tem como objetivo notificar defeitos nas estradas, através de um dispositivo móvel você é capaz de reportar ao um setor responsável uma cavidade que você localizou na rodovia e que esta prejudicando o trânsito dos automóveis. Como segundo trabalho relacionado tem uma aplicação de

origem Italiana o *Eureka!*[vii], ela foi desenvolvida para ser apresentada no evento chamado *Apps4Italy*[viii], a aplicação tem como objetivo apresentar dados da qualidade de vida em uma determinada posição geográfica. Trata-se de um aplicativo gratuito e aberto para ser usados em dispositivos móveis, e utiliza o sistema operacional *Android*[ix].

Como a aplicação vai utilizar posicionamento geográfico, (MAO, 2011) informa a seguinte preocupação em relação ao uso de redes sem fio como GPS e Wi-fi. O GPS é um bom sistema para localização geografia, porem pode ter um mau funcionamento em ambientes fechados, neste caso temos a localização por Wi-Fi, que pode fornecer uma localização razoável, porém pode ser menos efetivos em alguns locais.

3 DESENVOLVIMENTO/MODELO

Esta seção apresenta uma descrição do problema, o cenário, a solução onde vai envolver a arquitetura e o estudo de caso.

Como descrito na introdução, este estudo tem como objetivo investigar arquiteturas computacionais para identificar aquela mais apropriada ao desenvolvimento de aplicações móveis com foco na área de cidades inteligentes. Essa arquitetura deve funcionar como um serviço, onde podemos ter a informação centralizada facilitando á utilização da mesma.

3.1 Cenário

Em primeiro momento vamos utilizar a universidade como ambiente de experimentação do sistema de notificação, onde os usuários realizem suas sugestões ou criticas de um determinado local.

Como exemplo de utilização, podemos ter uma situação conforme ilustrado na figura abaixo. Considere um dia de intenso calor:

Figura 3. Exemplo de utilização da aplicação piloto

Fonte: Autoria própria, 2012.

No momento que o aluno realizou a notificação, o departamento responsável recebeu e desta forma foi possível ser avaliado o ambiente, onde foi optado por colocar um equipamento de ventilação melhor.

3.3 Solução

Criar uma aplicação piloto análoga ao *Cambridge iReports,* que tem o objetivo de reportar defeitos nas estradas, entretanto a aplicação de notificação desde trabalho vai possibilitar ao frequentador da universidade reportar sugestões e defeitos nos recursos oferecidos pela instituição de ensino.

Desta forma o aluno ou professor, pode sugerir melhorias para o ambiente acadêmico, ou informar a falha de algum recurso oferecido como um ar-condicionado com defeito. Com esse tipo de sistema pode ser visto as necessidades dos freqüentadores da universidade e possibilita melhorar o ambiente.

3.3.1 Arquitetura

Em primeiro momento, foi questionado o que vamos precisar para criar uma arquitetura para ser utilizada pelo sistema de notificação. Desta forma, foram levantados alguns questionamentos:

a) Como vai ser gerenciada a informação fornecida pelo usuário?
b) Será necessário disponibilizar para diferentes plataformas, como *iOS*^{x}?
c) Quais os recursos podem ser utilizados?
d) Aplicações que não estão relacionadas à plataforma *mobile* utilizarão o recurso da arquitetura?

Através destes questionamentos, surgiu a necessidade de haver um local para gerenciar o tráfego da informação e que fosse possível agregar diferentes serviços, sem impacto na aplicação cliente. Com isso iniciou-se o estudo de uma arquitetura capaz de fornecer este tipo comunicação entre serviços. Ou seja, uma arquitetura que utilize o mínimo possível dos recursos dos dispositivos móveis usados como cliente.

Para criar essa arquitetura, é utilizado um estilo de arquitetura chamada SOA. Uma plataforma deste tipo possui uma composição de diferentes tecnologias. Assim, podemos ter três tipos de serviços dentro deste conceito, um serviço solicitante, um serviço provedor e um intermediador. A Figura 4 ilustra esses três conceitos onde:

a) Serviço solicitante: O solicitante do serviço é a entidade que basicamente exige algum trabalho operacional realizado por outro serviço, onde é realizada uma busca através de listas de descrições dos serviços prestados pelo intermediador de serviço. Também é responsável pela ligação dos serviços após a sua descoberta.
b) Serviço provedor: O provedor de serviços é a entidade que tem acesso a outros serviços, também é responsável pela criação de serviços e de suas publicações no corretor de serviços.
c) Serviço intermediador: Têm a informação sobre todos os serviços registrados dentro do ESB, os mesmos responsáveis pelo redirecionamento de requisições para o solicitante do serviço correspondente e provedor.

Figura 4. Uma visão dos serviços SOA

Fonte: Adaptado de (Ali, 2012)

No presente projeto foi direcionado o foco para um ESB que por sua vez é um *middleware*[xi], um mediador entre software e outras aplicações e aplicações chaves. (ALI, 2012)

3.3.1.1 Serviço intermediador

Segundo o site da *Red Hat*[xii], o ESB nos possibilita a implementação de uma arquitetura orientada a serviços, como fornecedor desta plataforma temos um servidor de código aberto chamado *jBoss* do Java EE, hoje pertencente a *Red Hat*. Essa arquitetura de software fornece serviços fundamentais para arquiteturas complexas através de mecanismos de mensagens orientados a eventos e baseada em padrões. (ALI, 2012)

Atualmente temos outras empresas que oferecem o serviço intermediador sendo duas delas, a IBM com o *WebShere* ESB (IBM, 2012) e a Oracle com o OSB (ORACLE, 2012). No presente trabalho é utilizada a disponibilização da Red Hat por ter uma versão *Open Source*[xiii].

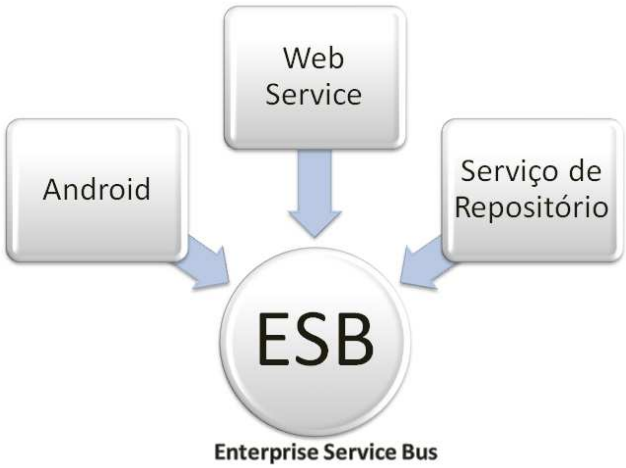
Com a utilização do *Enterprise service bus* conforme Ali e Red Hat (2012) temos os seguintes benefícios para aplicações que utilizaram a arquitetura:

a) Fornecer conectividade de sistemas;

b) Encaminhamento ou roteamento inteligente. Isto pode ser encaminhamento conteúdo de base via uma mensagem de um sistema para o outro com base no seu conteúdo;
c) Segurança e confiabilidade;
d) Monitoramento de serviços de gestão e registros de *logs*;
e) Conectividade com outros modelos de serviços como BPEL e BPM;

Segundo Ali (2012), o objetivo principal deste sistema, é a interoperabilidade, ou seja, é a capacidade de um sistema se comunicar de forma transparente. Usado para integrar diferentes plataformas e linguagens, abaixo uma figura ilustrativa da estrutura típica de um ESB em um ambiente SOA.

Figura 5. Uma visão sobre ESB



Fonte: Adaptado de (ALI, 2012)

Temos definido um serviço intermediador, mas no presente projeto ele não possui a responsabilidade de realizar regras de negocio, como se trata de um intermediador o papel deste componente da arquitetura é somente realizar o intermédio entre plataformas. Com isso surge à necessidade de criar um serviço para realizar as regras de negócio estabelecidas pelas funcionalidades que serão criadas, devido a isso foi analisada a possibilidade de criar serviços *web*.

3.3.1.2 Serviço web

Analisando a possibilidade de criar um ou mais serviços *web*, que segundo (CASTILLO et al., 2011) temos duas abordagens principais REST e SOAP. Em sua pesquisa o autor comenta que o ideal para quando temos baixa largura de banda e pouco recurso é usar a abordagem REST.

Envolvendo a utilização da tecnologia para dispositivos moveis, onde (HAMAD; SAAD; ABED, 2010) relata em sua pesquisa que o serviço *web* para ser utilizado por este tipo de dispositivo o melhor seria o REST*ful*, que por sua vez são aplicações *web* construídas sobre a abordagem REST. Em sua pesquisa o autor realizou testes usando as duas abordagens descritas, onde foi coletado o tamanho da mensagem e o tempo de resposta, foi realizado operações de concatenação de s*trings* e adicionado números do tipo *float* em arrays. Abaixo duas tabelas ilustrando o resultado dos testes realizados.

Tabela 1. Teste com tamanho da mensagem em serviços web REST e SOAP

Nº de arrays	Tamanho da mensagem em byte			
	SOAP/HTTP		REST/HTTP	
	Concatenação *strings*	Inclusão de números do tipo *float*	Concatenação de *strings*	Inclusão de números do tipo *float*
2	351	357	39	32
3	371	383	48	36
4	395	409	63	35
5	418	435	76	39
6	443	461	93	43
7	465	487	104	47
8	493	513	127	51

Fonte: Adaptado de (Hamad, Saad, & Abed, 2010)

Tabela 2. Teste com tempo de execução em serviços web REST e SOAP

Nº de arrays	Tempo em milisegundos			
	SOAP/HTTP		REST/HTTP	
	Concatenação *strings*	Inclusão de números do tipo *float*	Concatenação de *strings*	Inclusão de números do tipo *float*
2	781	781	359	359
3	828	781	344	407
4	828	922	359	375
5	969	1016	360	359
6	875	953	359	359
7	875	875	469	360
8	984	875	437	344

Fonte: Adaptado de (Hamad, Saad, & Abed, 2010)

Segundo (DIMAGGIO et al., 2011) o modelo SOAP também é suportado pelo servidor de aplicação *jBoss*, logo se for necessário criar um serviço web com esse modelo será suportado.

Portanto, tomando como base esses dados, neste trabalho será usado o modelo REST.

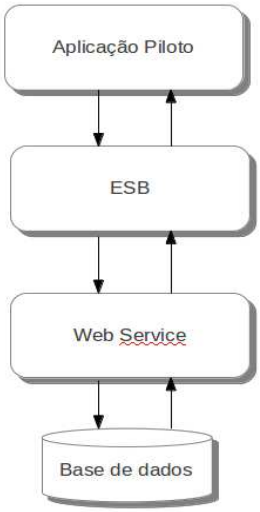
3.3.1.3 Repositório de dados

No desenvolvendo da aplicação piloto temos a necessidade de criar um local para armazenar os dados das movimentações realizadas no sistema, nesse caso, foi estudada a possibilidade de usar o servidor de banco de dados *MySQL, que* segundo (Site sobre Mysql, 2012) possui uma licença livre e atende as necessidade da aplicação piloto*.* Em primeiro momento não existe a necessidade de trabalhar com um servidor de banco de dados mais robusto que o *MySQL,* pois não vai existir muitos serviços solicitando transações para o banco de dados. Caso precise de troca ou adicionar algum outro servidor de repositório de dados o *jBoss,* servidor de aplicação, oferece a funcionalidade de criar *datasources*[xiv] para realizar as conexões. Desta forma em caso de uma troca é só alterar os parâmetros de conexão, quando o serviço precisar utilizar o banco de dados vai utilizar o que está configurado no *datasource* (Site sobre Jboss, 2012)*.*

3.3.2 Ambiente de desenvolvimento

Nesta seção, é falado sobre o ambiente de desenvolvendo envolvendo a aplicação piloto, o ESB, o serviço *web* e o banco de dados. Abaixo uma figura ilustrando o ambiente de desenvolvimento.

Figura 6. Estrutura de ambientes



Fonte. Autoria própria, 2012

Com a estrutura definida é possível separar as responsabilidades. Temos a aplicação piloto que é a interface do cliente ou usuário final, onde serão inseridas as informações de notificação, como: um vazamento de água no prédio 8. Logo abaixo está ESB como já foi falado anteriormente ele é um serviço intermediador, ele vai ficar entre o terminal do usuário e o serviço *web,* sendo assim ele fica responsável por enviar a mensagem do usuário ao destino certo e retornar um uma mensagem para o usuário conforme já visto na Figura 2. Na Figura 6 também temos os serviços *web* que ficaram responsáveis pela regra de negócio e realização da persistência dos dados. E por fim temos a base de dados, ela possui a responsabilidade de armazenar os dados da aplicação piloto.

No desenvolvimento da arquitetura foram utilizados dois servidores de aplicação, um responsável pela interpretação do ESB o *jboss* 6.x e temos o *jboss* 7.1 para realizar a interpretação do serviço *web.* Foi preciso usar servidores

diferentes devido ao *jBoss 7.1* não oferecer suporte, conforme (Site sobre Jboss, 2012) e a última versão capaz de interpretar o ESB é a 6. Contudo seria de boa utilidade os serviços *web* serem interpretado pelo *jboss* 7.1 devido a isto foi mantido os dois servidores.

Em relação à comunicação entre as plataformas, foi preciso definir de que forma a mensagem, neste caso está sendo usado o XML por decisão de projeto. Com o uso do XML foi estabelecido um contrato entre as plataformas, o desenvolvedor que for utilizar a arquitetura deve seguir esse padrão estabelecido e considerar os tipos de dados que estão sendo enviados e recebidos. Abaixo uma figura ilustrando um modelo de contrato.

Figura 7. Modelo de contrato

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.smartcity.com.br/register/userInfoLoginRequest"
 xmlns:tns="http://www.smartcity.com.br/register/userInfoLoginRequest"
 elementFormDefault="qualified">

    <element name="userInfoLoginRequest" type="tns:UserInfoLoginRequest" />

    <complexType name="UserInfoLoginRequest">
        <sequence>
            <element name="username" type="string" maxOccurs="1" minOccurs="1" nillable="false"  />
            <element name="password" type="string" maxOccurs="1" minOccurs="1" nillable="false"  />
        </sequence>
    </complexType>

</schema>
```

Fonte. Autoria própria, 2012

No desenvolvimento dos ambientes construído utilizando a linguagem de programação *Java*, com essa linguagem é possível utilizar algumas ferramentas para facilitar o desenvolvimento, além de suportar o desenvolvimento de todos seguimentos ilustrados na Figura 6.

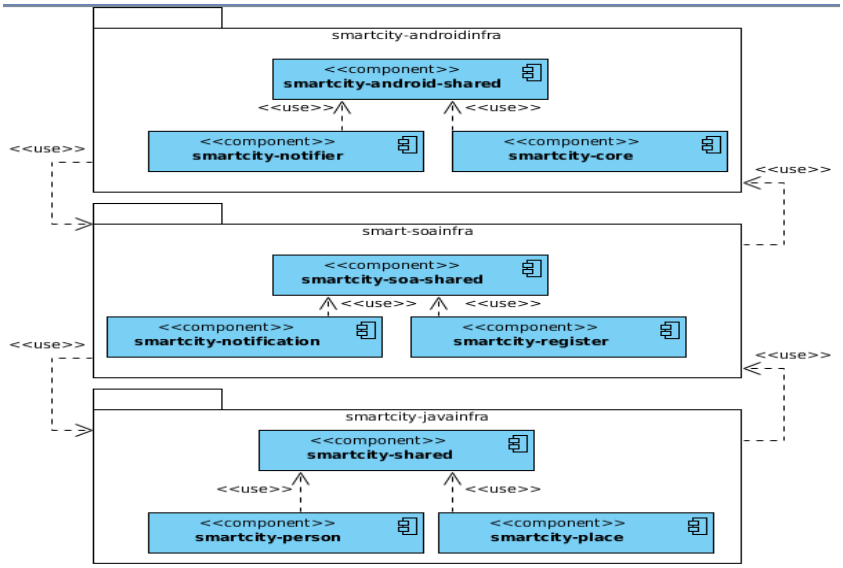
Uma destas ferramentas importantes é o *maven* que segundo (Site sobre Maven, 2012), ela surgiu com o objetivo de padronizar os projetos, com uma maneira clara de publicar informações e compartilhar dependências. Abaixo algumas vantagens na utilização do *maven.*

a) Possui um bom gerenciamento de dependências;
b) Construção e gerenciamento do ciclo de vida do projeto;
c) Fornecer informações sobre o projeto com qualidade;
d) Possui um vasto repositório de dependências;

A outra ferramenta importante que foi definida é o *hibernate.* Ela é utilizada pelos serviços *web.* Segundo (JBOSS, 2012), ele é um framework que faz o mapeamento objeto/relacional para ambiente *Java*.

Envolvendo a estrutura dos ambientes como visto na Figura 6, foi possível realizar uma estrutura de componentes para criar uma infraestrutura seguindo o modelo ilustrado. Conforme diagrama abaixo.

Figura 8. Diagrama de componentes



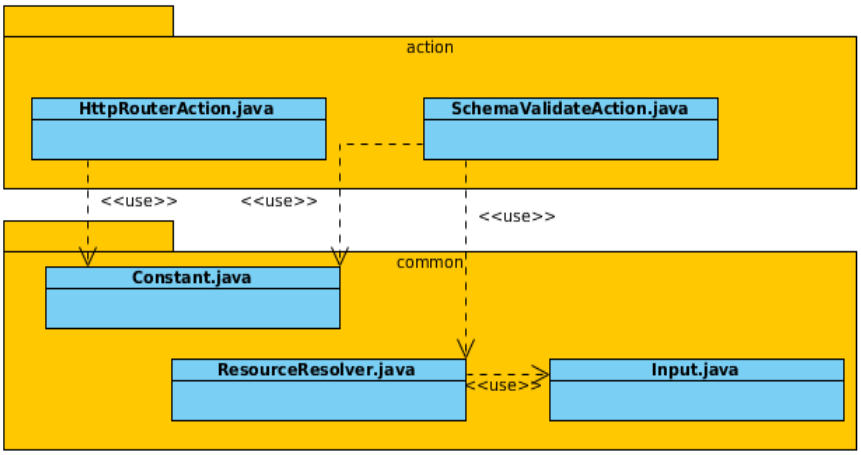
Fonte. Autoria própria, 2012

3.3.2.1 Serviço intermediador

Nesta seção, é mostrada de que forma foi realizado o desenvolvimento do serviço intermediador, além deste serviço realizar a comunicação entre as aplicações, ele ficou responsável por realizar a validação dos dados enviados por XML e transformações usando XSLT. Os componentes para criar este serviço foram definidos conforme o diagrama ilustrado na Figura 8 onde temos um componente definido como *smart-soainfra*.

Olhando para estrutura dos componentes temos o *smartcity-soa-shared, que* contém objetos mapeados que podem ser utilizados pelos outros componentes *smartcity-register* e *smartcity-notification.* Abaixo temos uma figura ilustrando como ficou estruturado o diagrama classes.

Figura 9. Diagrama de classes smartcity-soa-shared



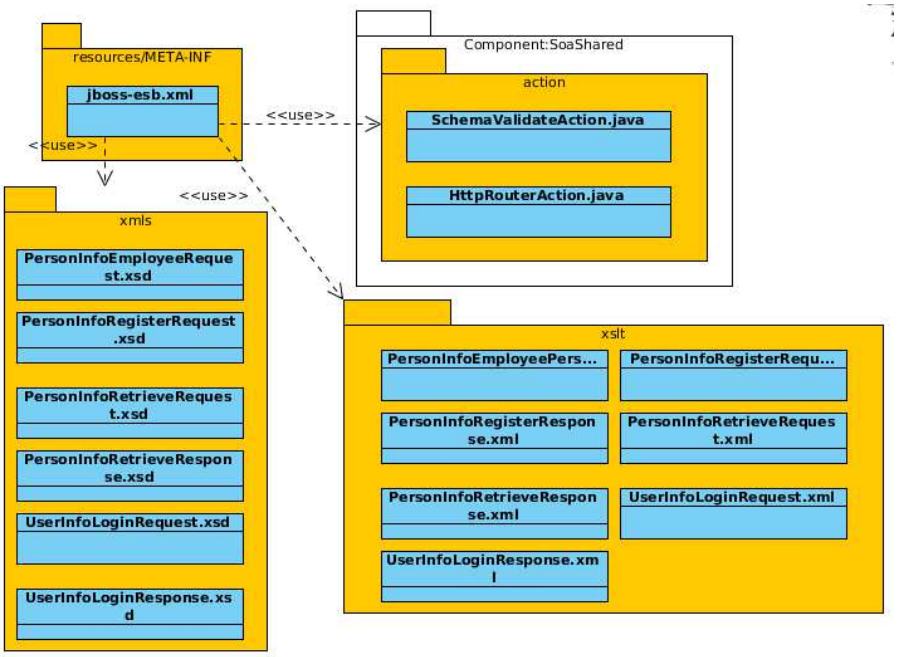
Fonte. Autoria própria, 2012

Neste diagrama ilustrado da Figura 9, temos a classe *HttpRouterAction* para realizar a função de roteamento da informação entre as aplicações. Outra classe presente é o *SchemaValidateAction,* ela coleta o validador do XML que é usado nos componentes *smartcity-register* e *smartcity-notification*. As outras classes como *Input, ResourceResolver* e *Constant* são usadas pelas classes pertencentes ao pacote *action.*

Os componentes *smartcity-register* e *smartcity-notification* são os dois ESB que estão sendo utilizados pela aplicação piloto e pelo serviço *web*. O *smartcity-register* é responsável pelo roteamento de informações relacionadas a pessoas, logo temos que o *login* e o cadastro da pessoa são de responsabilidade deste ESB. Já o *smartcity-notification* ficou com a responsabilidade de realizar o roteamento das informações referente a notificações criadas com a aplicação piloto e do cadastro de locais.

Nestes componentes são estabelecidos os contratos, conforme Figura 7, para realizar as operações usando a arquitetura e as transformações do XML, ambos com uma estrutura parecida. Na figura abaixo é possível visualizar o diagrama de classe *smartcity-register.*

Figura 10. Diagrama de classes smartcity-register



Fonte. Autoria própria, 2012

Na figura acima podemos visualizar a estrutura definida como padrão dos componentes ESB, o pacote *xmls* que contem os XML *schemas* eles são arquivos usados para validar os XML enviados pela aplicação cliente ou qualquer outra aplicação. O pacote *xslt,* possui os arquivos responsáveis pela transformação do XML que for recebido ou enviado.

Um arquivo importante presente dentro destes componentes *smartcity-notification* e *smartcity-register* é o *jboss-esb* nele fica descrito o processo de execução a ser realizado, de que maneira o serviço vai ser solicitado, a maneira que vai ser enviado os dados e o método utilizado. Neste caso foi utilizado o POST, tendo a possibilidade de usar o GET ambos utilizados em requisições HTTP.

Analisando esse arquivo *jboss-esb* ele possui uma *tag* chamada <services>, onde dentro dela vamos ter os serviços realizados pelo ESB. Conforme a figura abaixo tem um serviço de autenticação do usuário, utilizado pela aplicação piloto.

Figura 11. Mapeamento do serviço de autenticação

```
<service category="retrieve" name="user-info-login" description="Retrieve class user information of login" invmScope="GLOBAL">
    <listeners>
        <http-gateway name="HttpUserInfoLogin" busidref="http-bus" payloadAs="STRING" />
    </listeners>

    <actions mep="RequestResponse">
        <action name="validateRequest"  class="br.com.smartcity.soashared.action.SchemaValidateAction" process="validateSchema">
            <property name="schema" value="/br/com/smartcity/register/xmls/UserInfoLoginRequest.xsd"/>
        </action>

        <action name="transformRequest" class="org.jboss.soa.esb.smooks.SmooksAction">
            <property name="smooksConfig" value="/br/com/smartcity/register/xslt/UserInfoLoginRequest.xml" />
            <property name="set-payload-location" value="user-request" />
            <property name="resultType" value="STRING" />
        </action>

        <action name="httpRouter" class="br.com.smartcity.soashared.action.HttpRouterAction" process="invoke">
            <property name="input_location" value="user-request" />
            <property name="output_location" value="user-response" />
            <property name="method" value="POST" />
            <property name="Content-Type" value="application/xml" />
            <property name="uri" value="http://localhost:8080/smartcity-person/person-info/login" />
        </action>

        <action name="transform" class="org.jboss.soa.esb.smooks.SmooksAction">
            <property name="smooksConfig" value="/br/com/smartcity/register/xslt/UserInfoLoginResponse.xml" />
            <property name="get-payload-location" value="user-response" />
            <property name="set-payload-location" value="org.jboss.soa.esb.message.defaultEntry" />
            <property name="resultType" value="STRING" />
        </action>

        <action name="validateRequestResponse"  class="br.com.smartcity.soashared.action.SchemaValidateAction" process="validateSchema">
            <property name="schema" value="/br/com/smartcity/register/xmls/UserInfoLoginResponse.xsd"/>
        </action>
    </actions>
</service>
```

Fonte. Autoria própria, 2012

Como ilustrado na figura o serviço tem em seu cabeçalho uma categoria, que neste caso foi chamada de *retrive,* um nome *user-info-login,* uma descrição onde é possível visualizar na no atributo *description* do XML e o escopo que no caso esta como global.

Temos após o cabeçalho a *tag listeners* onde é definido o nome do *http-gatway,* para cada serviço criado é preciso definir um nome para que não exista conflito. Com essas informações do cabeçalho e criando a *tag* do *http-gatway* já definidas é possível você montar o caminho HTTP para solicitar o serviço, no caso deste exemplo temos o seguinte "*http://seudominio/register/http/retrive/user-info-login*".

Logo após a *tag listeners,* temos as ações a serem realizadas quando esse serviço for requisitado, elas estão localizadas dentro da *tag actions.* Seguindo o processo descrito na Figura 11, à primeira ação que temos é a de validação onde é

utilizado um XML *Shema,* conforme foi visto na Figura 8, ele vai validar o XML que esta sendo enviado antes realizar as próximas ações.

Se o XML estiver de acordo conforme foi estabelecido, a próxima ação a ser realizada é a de transformação do XML enviado pela aplicação piloto, para que ele seja enviado de acordo como esperado pelo serviço *web.*

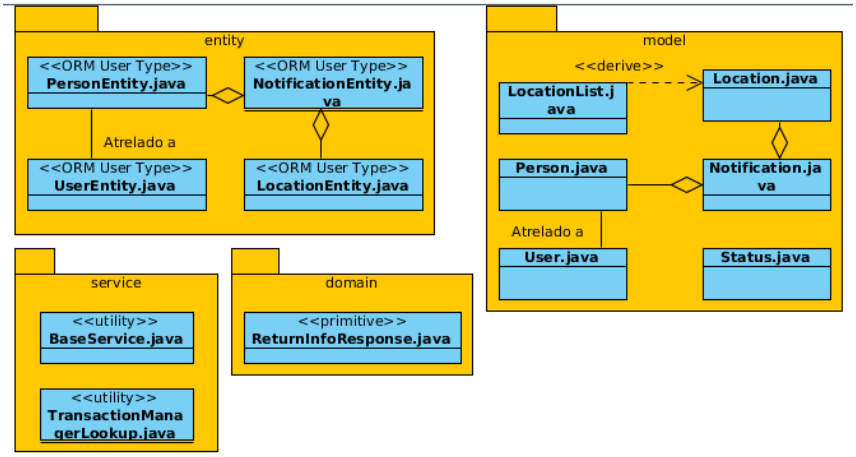
Após transformação do XML é realizado o envio para o serviço *web,* nesta ação é possível visualizar a utilização as variáveis que estão declaradas no *smartcity-shared,* conforme Figura 9, na classe *Cosntant* as variáveis estáticas *input_location,* output_location, *method, content_type* e *uri* onde os seus valores foram mapeados nesta ação*.* Na propriedade *input_location* é inserido os valores a ser enviado para o serviço *web*, na *output_location* o valor que vai ser retornado, na *method* têm o método de envio que no caso é POST, no *content_type* temos o *content-type* do XML e no valor *uri* temos o HTTP para envio. Essa ação realiza o envio da informação para o serviço *web,* vai carregar a resposta na propriedade de *output_location,* com a resposta do serviço *web* é realizado o processo de transformação do XML e depois de validado novamente para que seja enviada a resposta para aplicação piloto. Os outros serviços presentes seguem o mesmo padrão que foi descrito ESB

3.3.2.3 Serviço *web*

Nesta seção vamos abordar a camada de serviço, ela seguiu o mesmo padrão dos componentes do ESB, contento um principal chamado *smartcity-javainfra* ele contem as dependências comuns entre os componentes, que neste caso temos *smartcity-person, smartcity-place* e *smartcity-shared.*

O *smartcity-shared* possui o mapeamento dos objetos que podem ser utilizados pelos demais, segue abaixo uma figura ilustrando o diagrama de classe.

Figura 12. Diagrama de classe smartcity-shared



Fonte. Autoria própria, 2012

Como ilustrado na figura acima, temos o componente divido por pacotes, o pacote *entity* possui as classes que foram mapeadas com o uso do *hibernate,* onde serão utilizadas quando existir a necessidade de comunicação com o banco de dados.

O pacote *model* ficou responsável pelas regras de negocio e nele foi mapeado o XML usando anotações sobre os atributos, com a utilização das seguintes dependências *Jax-RS*[xv] e *Jax-B*[xvi].

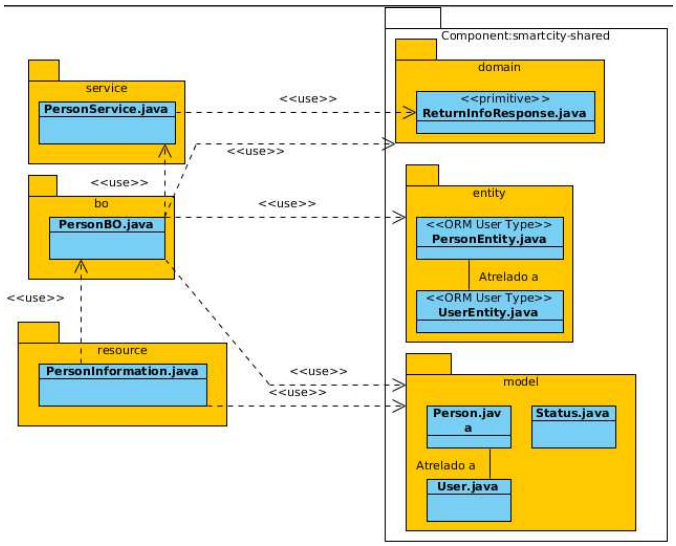
Conforme ilustrado na figura, temos o pacote *domain,* nele foi criada uma classe que possui parâmetros de retorno estáticos, como *success* e *failed*. Onde os outros componentes podem utilizar se existir a necessidade

Outro pacote ilustrado na figura é o *service* onde temos a classe *BaseService* ela ficou responsável pelas funções de banco de dados, tais como realizar persistência, atualizações, remoções e pesquisas. Além disso, essa classe ficou responsável por realizar o *serializer*[xvii] do objeto conforme o mapeamento realizado com *hibernate* usado nas classes do pacote *entity*. A outra classe chamada de *TransactionManagerLookup* é responsável por realizar controle de transações utilizando ferramentas do servidor de aplicação, ela foi criada para exemplificar uma eventual necessidade, no projeto atual não foi preciso utilizar.

Da mesma forma que no ESB foi divida a responsabilidade entre os componentes *smartcity-person* e *smartcity-place*. O *smartcity-person* ficou

responsável pelas atividades relacionadas à pessoa, como *login* e cadastro. A figura abaixo mostra o diagrama de classe deste componente.

Figura 13. Diagrama de classe smartcity-person

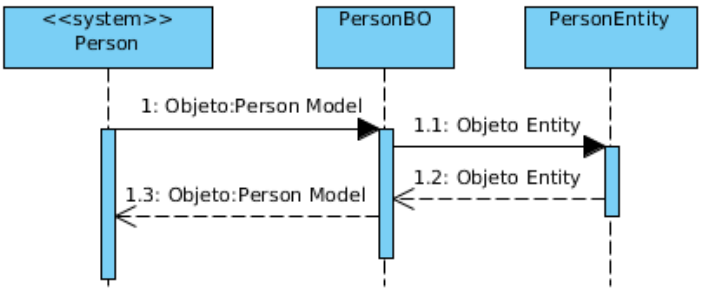


Fonte. Autoria própria, 2012

Como ilustrado na figura temos um pacote chamado *service*, o mesmo possui uma classe *PersonService* que herda as funcionalidades da *BaseService* criada no componente *smartcity-shared.* A classe *PersonService* possui as funcionalidades para realizar as funções relacionadas a banco de dados.

O pacote *resource* temos uma classe com métodos que realizam operações de entrada e saída com XML, assim como no componente *smartcity-shared* é utilizado anotações sobre os métodos. No pacote BO há classes para realizar o intermédio entre as classes de recursos e entidade, para exemplificar melhor foi criado um diagrama de sequência usando a classe *PersonBO* conforme a figura abaixo.

Figura 14. Funcionalidade da classe PersonBO

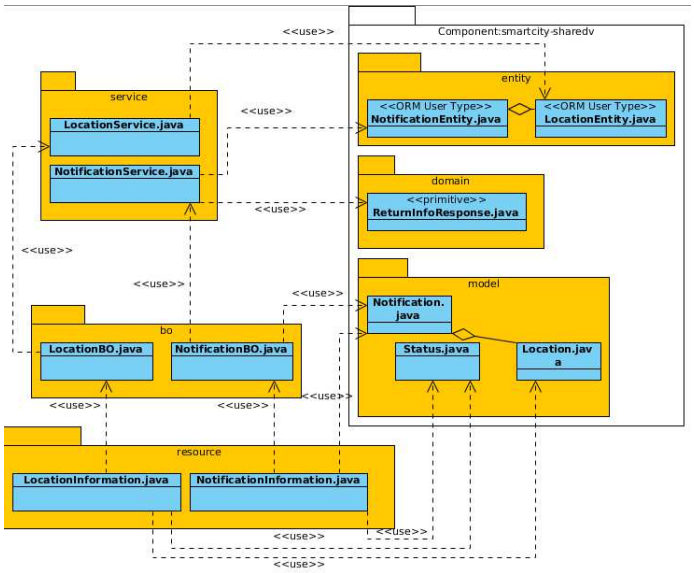


Fonte. Autoria própria, 2012

Como pode ser visto no diagrama de exemplo acima temos a classe *PersonBO* que esta entre a classe de recursos e entidade, com isso ela recebe um objeto de modelo e envia um objeto de entidade para ser realizadas as operações do banco de dados, após o objeto retornado da classe de entidade a classe BO envia um objeto modelo para classe recursos para ser retornado em forma de XML para o ESB.

O componente *smartcity-place* possui uma hierarquia de pacotes idêntica ao do *smartcity-person,* porem realiza operações referentes ao local, operações como registro de um novo local, cadastro de uma notificação e pesquisa de locais. Abaixo uma figura ilustrando o diagrama de classes do componente *smartcity-place.*

Figura 15. Diagrama de classes smartcity-place



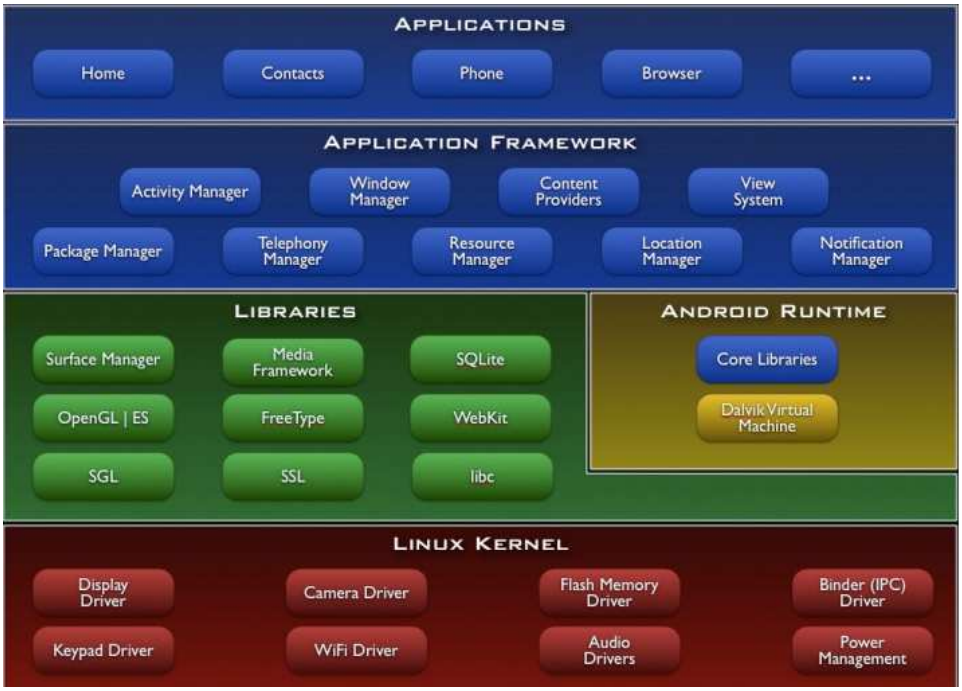
Fonte. Autoria própria, 2012

3.3.2.2 Aplicação piloto

Para montar um sistema que servirá como um estudo de caso para teste da arquitetura descrita anteriormente, vai ser utilizado uma tecnologia baseada em *Android*[xviii] a mesma trabalha como um sistema operacional, *middleware* e aplicações chaves para tecnologia móvel, atualmente a empresa responsável pelo *Android* é a *Google Inc*[xix].

Sua arquitetura pode ser dividida em cinco grupos *Aplications*, *Aplication Framework*, *Libraries*, *Android Runtime* e *Linux Kernel*, segue abaixo a figura ilustrando essas cinco camadas. (Sobre Developer Android)

Figura 16. Uma visão sobre arquitetura Android



Fonte. Arquitetura Andriod de (Sobre Developer Android)

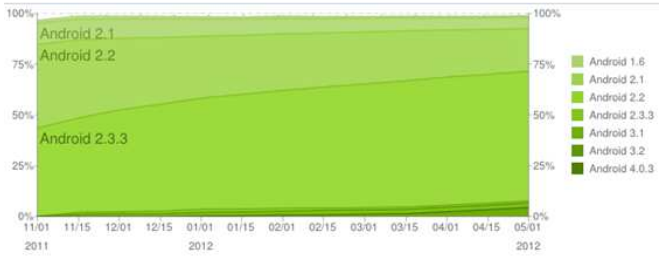
Cada camada possui uma responsabilidade dentro do sistema Android, segue abaixo as funcionalidades:

a) Aplications: Essa camada contem aplicações de mais alto nível utilizadas pelo usuário, como ferramenta de e-mail, mapa, calendário e também as ferramentas vistas nos *Smarts Phones*, como os contatos do telefone. (SPECKMANN, 2008)
b) Aplication Framework: Essa camada encontra-se aplicações para nível de sistema operacional, dentro destas camadas estão incluídas as funcionalidades *standard*[xx]. (SPECKMANN, 2008)
c) Libraries: Nesta camada estão incluídas as bibliotecas para o uso do sistema Android, como por exemplo, *SQLite*[xxi] que esta ilustrada na imagem acima, usada para conexão com o banco de dados *OpenGL*[xxii] para design gráfico. (SPECKMANN, 2008)
d) Runtime: Contém duas funcionalidades nesta camada, a primeira como visto na ilustração acima é a *Core Libraries*, que tem o objetivo de fornecer às principais bibliotecas da linguagem de programação Java, a segunda *Dalvik Virtual Machine*[xxiii] ela fica presente entre a aplicação e o sistema operacional do dispositivo móvel, como o sistema operacional não realiza a interpretação do código escrito em *Java*, onde a maquina virtual realiza essa tarefa. (SPECKMANN, 2008)
e) Linux Kernel: O nível mais baixo tem como funcionalidades o gerenciamento dos drives responsáveis para comunicação do entre o *hardware* e a aplicação. Neste nível temos a questão do gerenciamento de memória, gerenciamento de processos e entre outras funções. (SPECKMANN, 2008)

A plataforma *Android* possui algumas versões, sendo elas 1.5, 2.1, 2.2, 2.3 - 2.3.2, 2.3.3 - 2.3.7, 3.0, 3.1, 3.2, 4.0 - 4.0.2 e 4.0.3 - 4.0.4. (Sobre Developer Android)

Segundo a pesquisa realizada com o *Google Play*[xxiv], com base nas versões de plataformas que acessaram o ambiente em busca de aplicativos, temos o seguinte gráfico.

Figura 17. Versões que realizam busca de aplicativos no Google Play



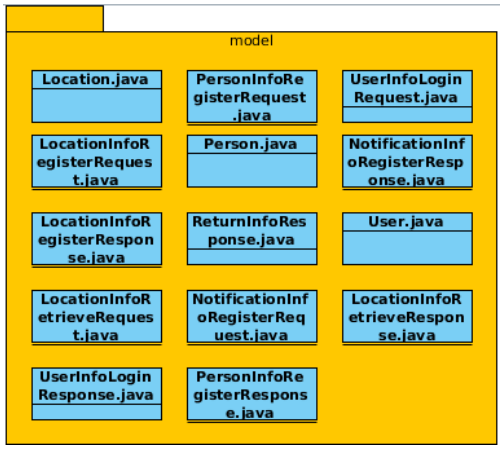
Fonte. Dados de utilização do *Andriod* (Sobre Developer Android)

Na aplicação piloto utiliza a plataforma *android* utilizaremos a versão 4.0, devido a existir uma grande chance dos novos *smarts phones* e *tablets* serem fabricados com essa versão.

A aplicação piloto foi estruturada no mesmo padrão que outros componentes descritos nas seções anteriores utilizando o *maven*, desta forma nos possibilitou o gerenciamento de dependências e dos componentes. Como pode ser visto na Figura 8 o componente *smartcity-androidinfra* possui três sub-componentes *smartcity-android-shared, smartcity-core* e *smartcity-notifier*.

Cada componente possui a sua responsabilidade, desta forma temos que o *smartcity-shared* é responsável por mapear os objetos que serão utilizados entre o *smartcity-core* e *smartcity-notifier*. No caso da aplicação piloto ele mapeou os objetos de requisição e de resposta para arquitetura, conforme pode ser visualizado no diagrama abaixo.

Figura 18. Visão do componente smartcity-share pacote model



Fonte. Autoria própria, 2012

Devido a existir o contrato de envio de informações, foi necessário realizar um mapeamento de elementos para criar o XML. Abaixo uma figura ilustrando como foi realizado o mapeamento dos atributos.

Figura 19. Requisição de login

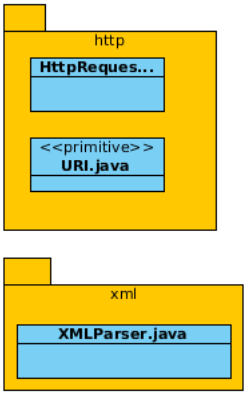
```
package br.com.smartcity.shared.model;

import org.simpleframework.xml.Element;
import org.simpleframework.xml.Namespace;
import org.simpleframework.xml.Order;
import org.simpleframework.xml.Root;

@Root(name = "userInfoLoginRequest")
@Namespace(reference = "http://www.smartcity.com.br/register/userInfoLoginRequest")
@Order(elements={"username", "password"})
public class UserInfoLoginRequest
{
    @Element(name = "username")
    private String username;

    @Element(name = "password")
    private String password;

    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

Fonte. Autoria própria, 2012

Essa classe ilustrada acima foi mapeado a requisição de *login* que é realizado pela aplicação cliente, usando o *framework simpleframework*[xxv] foi possível realizar o mapeamento dos elementos XML.

O componente *smartcity-core,* ficou responsável por executar a requisição HTTP para o ESB e realizar o *serializer* onde é convertido o objeto *Java* em XML e o XML para objeto *Java*. Conforme ilustrado na figura abaixo.

Figura 20. Visão do componente smartcity-core

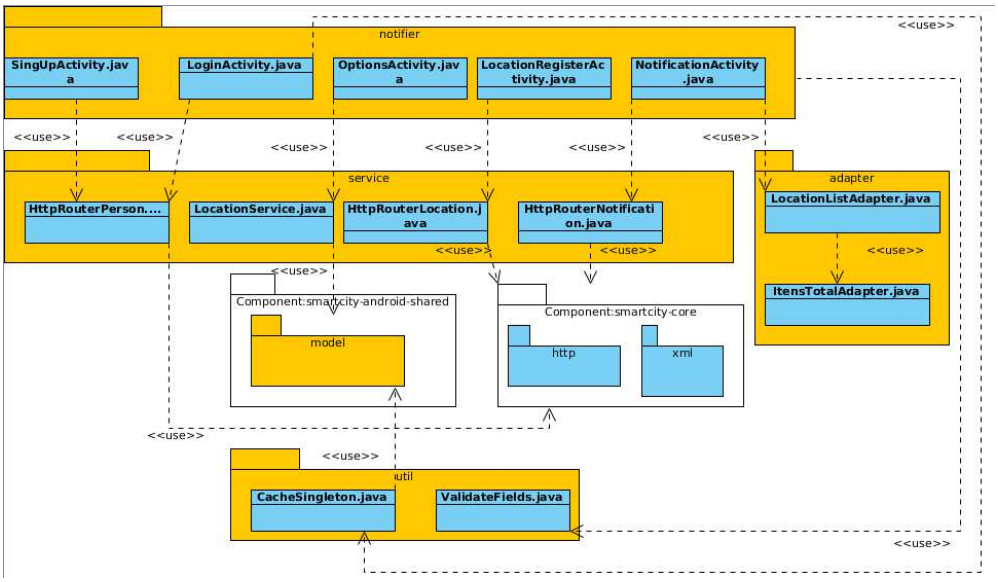


Fonte. Autoria própria, 2012

Como pode ser visualizado acima, tempos a classe *HttpRequest* como responsável por realizar a conexão com o ESB, a classe *URI* que contem os links para realizar a conexão e a classe *XMLParser* onde é realizado o processo de *serializer.*

No *smartcity-notifer* ele ficou responsável pela parte da comunicação com o usuário, *design* de telas, alguns serviços a serem executados em tempo de execução como validação de campos. Abaixo o diagrama de classes deste componente.

Figura 21. Visão do componente smartcity-notifer



Fonte. Autoria própria, 2012

No pacote *adapters* quando necessário criar uma lista de itens personalizada tem que ser implementado um *adapter* (Site Android Developer, 2012), nesse caso foi preciso para criar a tela de notificações, onde temos uma lista de locais.

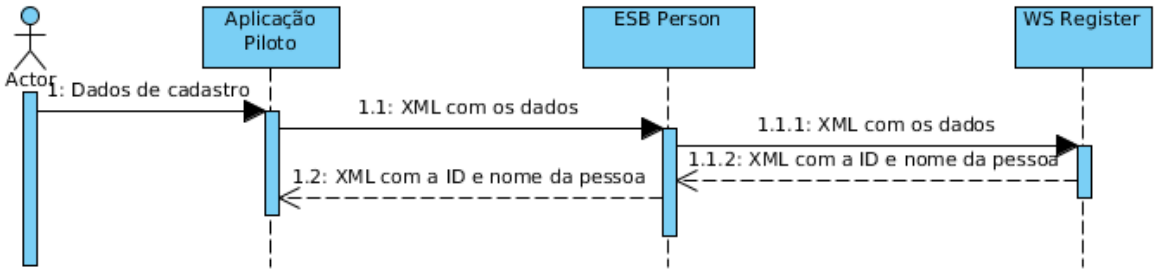
Logo abaixo do pacote *adapter* temos um pacote chamado *service* onde foi criado classes que utilizaram a classe de roteamento presente no componente *smartcity-core,* logo foi criado uma classe para cada atividade relacionada*,* por exemplo, se o usuário realizar o *login* no sistema o roteamento desta atividade vai ser utilizado à classe *HttpRouterPerson,* pois ela é responsável por realizar as requisições relacionadas à atividade do usuário/pessoa. Outra classe presente

nesse pacote é o *LocationService,* ela ficou responsável por providenciar o acesso do GPS ou acesso para internet com objetivo de buscar a posição geográfica.

Conforme ilustrado no diagrama foi criado um pacote chamado de *utils* temos duas classes dentro dele a *CacheSingleton* que foi desenvolvida no padrão *Singleton*[xxvi]*.* Ao usuário realizar *login* na aplicação ela carrega um objeto da classe *Person* com o identificador *id* e o nome *name*, desta forma possibilita que a seja usado esse objeto em outras telas da aplicação. Ela foi usada para latitude e longitude em um objeto de *Location* e garantiu as mesmas possibilidades de utilização do objeto *Person.*

Conforme ilustrado na Figura 19, temos as *activitys,* conforme (PEREIRA e SILVA, 2009) é um conjunto de *views* que formão uma interface com usuário, possuindo eventos previamente programados. Como pode ser visto foi definido cinco telas no sistema, a *SingUpActivity, LoginActivity, OptonsActivity, NotificationActivity e LocationRegisterActivity.* A *activity SingUpActivity* possui operações de cadastro do usuário, toda pessoa que deseja utilizar o sistema de notificação precisa criar um usuário e senha. Abaixo uma figura ilustrando o processo de cadastro envolvendo a utilização da arquitetura.

Figura 22. Diagrama de seqüência do envio da informação

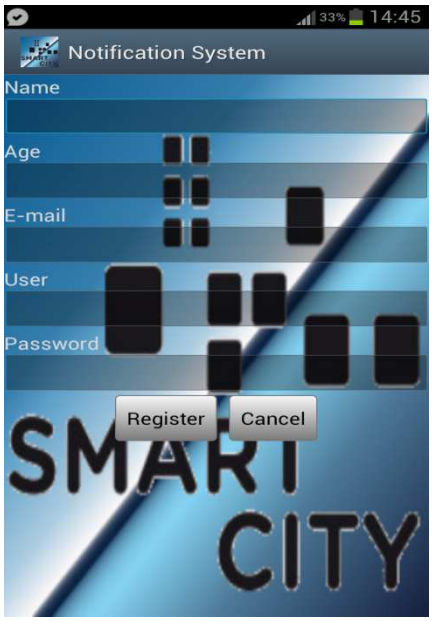


Fonte. Autoria própria, 2012

No digrama acima é possível visualizar a realização do cadastro de usuário envolvendo a arquitetura, a pessoa entra com os dados para o cadastro na aplicação piloto, que por sua vez envia o XML para o ESB *Person* onde realizado os processos de sua responsabilidade e envia o XML com os dados para o serviço *web* que neste contexto é o *Register,* onde é feito o tratamento da informação para cadastrar os dados no banco de dados. Se tudo for bem sucedido o retorno é um

XML com a com um valor *success,* caso a operação não seja realizada com êxito o retorno vai ser um *failed*. Logo abaixo é possível visualizar a tela de cadastro definida na aplicação piloto.

Figura 23. Visão da tela de cadastro

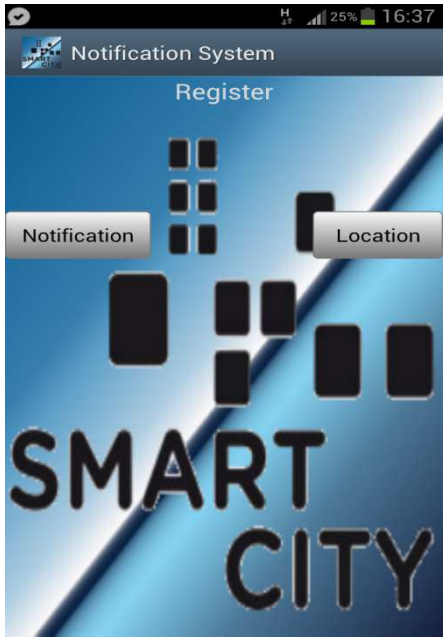


Fonte. Autoria própria, 2012

Como é possível ver na figura temos os campos necessários para realizar o cadastro na aplicação piloto informando os seguintes dados, nome, idade, e-mail, usuário e senha. Destes campos descritos somente o campo idade não precisa ser preenchido pelo usuário.

Após ser realizado o cadastro é possível entrar na aplicação com o usuário e senha informados, quando a autenticação for bem sucedida o usuário é encaminhado para tela de seleção de opções onde está sendo executada pela *activity OptonsActivity.* Ela possui as duas opções que podem ser escolhidas pelo usuário, o cadastro de um novo local ou realizar uma nova notificação, ela também é responsável por utilizar o serviço responsável pela busca da posição geográfica e carrega o objeto *Location* usando a classe *CacheSingleton*. Abaixo uma figura ilustrando a tela de opções.

Figura 24. Visão da tela de opções



Fonte. Autoria própria, 2012

Em caso do usuário selecionar a opção de *Location* o sistema vai abrir a tela de cadastro de locais, onde vai ser possível descrever o nome do local. Como ilustrado na figura abaixo.

Figura 25. Visão da tela de cadastro de locais

Fonte. Autoria própria, 2012

Após o cadastro do local o usuário já possui a possibilidade de realizar uma notificação clicando no botão *Go Notification* ilustrado na Figura 25. Na figura abaixo temos ilustrado a tela de notificação de ocorrências.

Figura 26. Visão da tela de cadastro da notificação

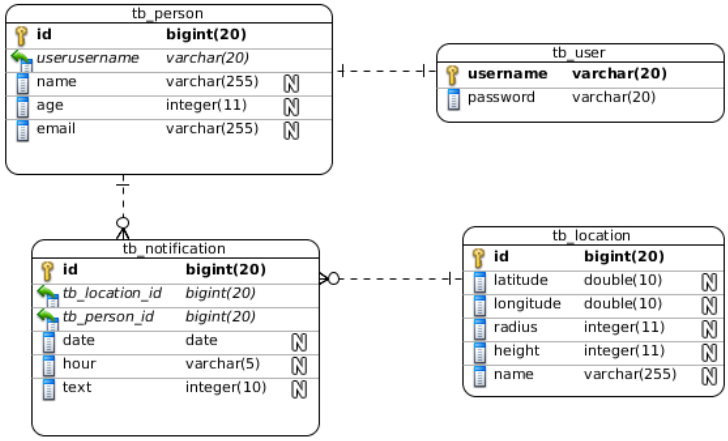
Fonte. Autoria própria, 2012

Nesta tela o usuário pode descrever a sua ocorrência e enviar clicando no local, em caso do lugar onde a pessoa está não esteja listado para realizara a operação. É possível cadastrar o lugar clicando no botão *Register Location,* onde será redirecionado para tela de registro de locais conforme Figura 25.

3.3.2.4 Repositório de dados

Nesta seção vamos falar de como foi modelado à estrutura de tabelas no banco de dados. Como já foi informado anteriormente, o servidor do banco de dados utilizado é o *MySQL*. Segue a baixo o diagrama ER utilizado.

Figura 27. Diagrama ER



Fonte. Autoria própria, 2012

4 CONCLUSÃO

Em relação á área cidades inteligentes, podemos dizer que possibilita muitos trabalhos e pesquisas dentro deste conceito e que há uma grande possibilidade destes trabalhos e pesquisas agregarem valor dentro das cidades, resolvendo problemas que antes não eram vistos ou não existiam.

Na construção da arquitetura foi utilizado um *notebook* Dell modelo D630, com 4Gb de memória e um processador *core* 2 *duo* em um sistema operacional Ubuntu 12.04. Quando realizado a inicialização dos servidores de aplicação foi possível notar que eles utilizam bastante os recursos de *hardware*, caso a arquitetura for utilizada em ambiente real, por exemplo, em uma cidade seria de boa utilidade usar *clusters*[xxvii] para ter um melhor desempenho.

Envolvendo a utilização do ESB, temos algumas empresas que já utilizam esse modelo de serviço e foram considerados casos de sucesso, conforme (REDHAT, 2012) a Ampersand[xxviii] no México utiliza o ESB oferecido pela *Red Hat.* Segundo (ORACLE, 2012) a Telenet[xxix] da Bélgica ela utiliza uma solução da Oracle o OSB e temos também a empresa Humana S.A[xxx] do Equador que utiliza a solução disponibilizada pela IBM o *WebSphere* ESB (IBM, 2012).

A arquitetura desenvolvida é escalável, desta forma é possível agregar outros serviços conforme a necessidade e abre a possibilidade de desenvolvimentos de trabalhos futuros. Com a utilização do ESB podemos utilizar outros modelos de serviços conforme citado em subtitulo 3.3.1.1.

Levando em consideração a aplicação piloto, a mesma teve um bom funcionamento com arquitetura, executou todas as operações descritas no subtítulo 3.3.2.2. Em relação da utilização da posição geográfica fornecida pelo dispositivo *mobile*, temos que cada dispositivo vai ter a sua margem de erro na coleta da longitude e latitude, vale ressaltar que em ambientes fechados não foi coletada a posição geográfica pelo GPS, o dispositivo acabou utilizando a conexão 3G do celular. Considerado isso, possuir uma boa conexão do celular com a internet via 3G, pode garantir uma comunicação estável e a posição pode ser mais precisa.

5 TRABALHOS FUTUROS

Como um dos trabalhos futuros, é inserir o envio de fotos junto com a notificação, desta forma haveria mais detalhes do problema ou sugestão do que esta sendo enviado. Além deste trabalho, é ainda tem a necessidade de construir um sistema que realizar a análise dos dados que foram enviados.

Este sistema poderia ser alocado em um departamento responsável pelas notificações realizadas pelas pessoas, parecido com que temos no Centro de Operações do Rio de Janeiro, usado para o monitoramento do transito (RIO Prefeitura, 2012). Ao invés de existir o monitoramento das câmeras, pode ser realizado a analise das notificações. Abaixo uma figura ilustrando como poderia ser analisada a informação e levando em consideração a possibilidade de envio de fotos pela aplicação piloto.

Figura 28. Visão da tela do analisador de notificação

Fonte. Autoria própria, 2012

Como visto na figura acima, a aplicação piloto grava as posições geográficas de onde foi realizada a notificação, seria possível criar esses pontos em mapa facilitando a localização do local.

A arquitetura construída pode ser utilizada por outros sistemas envolvendo cidades inteligentes e seus seguimentos Figura 1, conforme for à regra de negocio do novo sistema é possível criar novos serviços intermediadores e serviços *web*. Como já existe o cadastro de pessoas, outros sistemas a serem criados que utilizam este tipo de funcionalidade, pode reutilizar. Desta forma pode ser utilizado o mesmo usuário para outros sistemas que utilizam a arquitetura.

REFERENCIAS

ALI, S. **Analyzing the Usage of Open Source Products for Soa**. [S.l.]: LAP Lambert Academic Publishing, 2012. ISBN ISBN: 9783848432271. Disponivel em: <http://books.google.com.br/books?id=uSq5tgAACAAJ>.

BAKICI, T.; ALMIRALL, E.; WAREHAM, J. A Smart City Initiative: the Case of Barcelona. **Journal of the Knowledge Economy**, p. 1-14, 2012.

CARAGLIU, A.; BO, C. D.; NIJKAMP, P. **Smart cities in Europe**. VU University Amsterdam, Faculty of Economics, Business Administration and Econometrics. [S.l.]. 2009.

CASTILLO, P. A. et al. SOAP vs REST: Comparing a master-slave GA implementation. **CoRR**, p. -1--1, 2011.

CHOURABI, H. et al. **Understanding Smart Cities:** An Integrative Framework. [S.l.]: [s.n.]. 2012. p. 2289-2297.

DIGITAL, C. Site Convergência Digital. **UOL**, 2012. Disponivel em: <http://convergenciadigital.uol.com.br/cgi/cgilua.exe/sys/start.htm?tpl=home>. Acesso em: 20 nov. 2012.

DIMAGGIO, K. C. M. K. B. T. C. L. et al. **Jboss Esb Beginner's Guide**. [S.l.]: Packt Publishing, Limited, 2011. ISBN ISBN: 9781849516587. Disponivel em: <http://books.google.com.br/books?id=xjCLHCeZmxYC>.

HAMAD, H.; SAAD, M.; ABED, R. Performance evaluation of restful web services for mobile devices. **International Arab Journal of e-Technology**, v. 1, n. 3, p. 72-78, 2010.

HARRISON, C.; DONNELLY, I. A. **A Theory of Smart Cities**. [S.l.]: [s.n.]. 2011.

HSIEH, H. N.; HOU, C. Y.; CHIA, P. C. **A study of smart town development strategies**. IEEE. [S.l.]: [s.n.]. 2011. p. 6684-6689.

IBM. Site IBM ESB. **WebSphere Enterprise Service Bus**, 2012. Disponivel em: <http://www-01.ibm.com/software/integration/wsesb/about/>. Acesso em: 15 dez. 2012.

JBOSS. Site sobre Hibernate. **Jboss Community**, 2012. Disponivel em: <http://www.hibernate.org/>. Acesso em: 02 jul. 2012.

KIRWAN, C. G. **Urban media:** a design process for the development of sustainable applications for ubiquitous computing for livable cities. New York, NY, USA: ACM. 2011. p. 1-2.
MAO, Y.

MCNERNEY, P. J.; ZHANG, N. Smarter Cities: Making societies smarter. **XRDS**, New York, NY, USA, v. 18, n. 2, p. 48-48, #dec# 2011. ISSN ISSN: 1528-4972 DOI: 10.1145/2043236.2071895. Disponivel em: <http://doi.acm.org/10.1145/2043236.2071895>.

NAPHADE, M. et al. Smarter Cities and Their Innovation Challenges. **Computer**, v. 44, n. 6, p. 32-39, june 2011. ISSN ISSN: 0018-9162 DOI: 10.1109/MC.2011.187.

ORACLE. Oracle. **Oracle Brazil**, 2012. Disponivel em: <http://www.oracle.com/br/index.html>. Acesso em: 15 dez. 2012.

PEREIRA, L. C. O.; SILVA, M. L. D. **Android para desenvolvedores**. [S.l.]: Brasport, 2009.

REDHAT. RedHat Brazil. **RedHat**, 2012. Disponivel em: <https://br.redhat.com>. Acesso em: 16 dez. 2012.

RIO DE JANEIRO. Prefeitura do Rio de Janeiro. **Site da Prefeitura do Rio de Janeiro**, 2012. Disponivel em: <http://www.rio.rj.gov.br/web/corio>. Acesso em: 26 nov. 2012.

SITE Android Developer. **Android Developer**, 2012. Disponivel em: <http://developer.android.com/guide/basics/what-is-android.html>. Acesso em: 04 maio 2012.

SITE sobre Jboss. **JBoss Community team**, 2012. Disponivel em: <http://www.jboss.org/>. Acesso em: 22 jul. 2012.

SITE sobre Maven. **Apache Maven**, 2012. Disponivel em: <http://maven.apache.org/>. Acesso em: 15 out. 2012.

SITE sobre Mysql. **Mysql Developer Zone**, 2012. Disponivel em: <http://dev.mysql.com/doc/refman/5.6/en/introduction.html>. Acesso em: 20 nov. 2012.

SPECKMANN, B. The Android mobile platform. **Access**, 2008. Disponivel em: <http://www.emich.edu/compsci/projects/Master_Thesis_-_Benjamin_Speckmann.pdf>.

---

[i] CadWell http://www.cadwell.com/
[ii] Cisco http://www.cisco.com/web/strategy/smart_connected_communities.html
[iii] IBM http://www-03.ibm.com/innovation/us/thesmartercity/index_flash.html
[iv] Simens http://www.siemens.com/sustainable-city/sustainable-city.html
[v] http://www.w3schools.com/webservices/ws_intro.asp
[vi] Cambridge iReports: Sistema utilizado a universidade de Cambridge. http://www.cambridgema.gov/iReport
[vii] Aplicação Eureka!: http://eurekatheapp.wordpress.com/progetto/
[viii] É uma competição que envolve cidades da União Europeia http://www.appsforitaly.org/en/
[ix] Sistema operacional utilizado em dispositivos móveis http://developer.android.com/guide/basics/what-is-android.html

[x] iOS: Plataforma utilizada pela Apple. https://developer.apple.com/technologies/ios/
[xi] Middleware: É o neologismo criado para designar camadas de software que não constituem diretamente aplicações, mas que facilitam o uso de ambientes ricos em tecnologia da informação. A camada de middleware concentra serviços como identificação, autenticação, autorização, diretórios, certificados digitais e outras ferramentas para segurança - http://www.rnp.br/noticias/2006/not-060926.html

[xii] http://www.redhat.com/products/jbossenterprisemiddleware/application-platform
[xiii] Open Source: Não possui fins lucrativos
[xiv] Datasource: http://docs.oracle.com/javase/6/docs/api/javax/sql/DataSource.html
[xv]Jax-RS: http://jax-rs-spec.java.net/
[xvi] Jax-B: http://jaxb.java.net/
[xvii] Serializer:http://docs.oracle.com/javaee/1.4/api/javax/xml/rpc/encoding/Serializer.html
[xviii] Android sistema muito utilizado em tecnologia móvel.
[xix] https://www.google.com.br/intl/pt-BR/about/
[xx] Standard: Neste contexto a palavra esta ligado a padrão.
[xxi] http://www.sqlite.org/
[xxii] http://www.opengl.org/
[xxiii] Dalvik Virtual Machine: Maquina Virtual utilizada em aplicações Android
[xxiv] https://play.google.com/store?hl=en&tab=m8
[xxv] Simpleframework http://www.simpleframework.org/
[xxvi] Singleton: http://www.ibm.com/developerworks/library/co-single/index.html
[xxvii] Cluster: http://technet.microsoft.com/en-us/library/cc785197(v=ws.10).aspx
[xxviii] Ampersand: http://www.ampersand.com/
[xxix] Telenet: www.telenet.be
[xxx] Humane S.A: www.**humana**.com.ec