UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

COMUNICAÇÃO DIRETA ENTRE DISPOSITIVOS USANDO O MODELO CENTRADO EM CONTEÚDO

IGOR MALDONADO FLOÔR

ORIENTADOR: PROF. DR. HÉLIO CRESTANA GUARDIA

São Carlos – SP

Outubro/2015

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

COMUNICAÇÃO DIRETA ENTRE DISPOSITIVOS USANDO O MODELO CENTRADO EM CONTEÚDO

IGOR MALDONADO FLOÔR

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Sistemas Distribuídos e Redes de Computadores
Orientador: Prof. Dr. Hélio Crestana Guardia

São Carlos – SP

Outubro/2015

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária UFSCar
Processamento Técnico
com os dados fornecidos pelo(a) autor(a)

Floôr, Igor Maldonado
F631c Comunicação direta entre dispositivos usando o modelo centrado em conteúdo / Igor Maldonado Floôr. -- São Carlos : UFSCar, 2016.
91 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2015.

1. Arquitetura orientada a serviços. 2. SOA. 3. SOAP. 4. REST. 5. Computação ubíqua. I. Título.

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato Igor Maldonado Floôr, realizada em 13/11/2015:

Prof. Dr. Helio Crestana Guardia
UFSCar

Prof. Dr. Cesar Augusto Cavalheiro Marcondes
UFSCar

Prof. Dr. Alfredo Goldman Vel Lejbman
USP

AGRADECIMENTOS

Agradeço pelo apoio de minha família, meu pai Inácio, minha irmã Thays e minha doce namorada Ingrid. Agradeço aos meus amigos Fernando Molina e Denis Oliveira que considero irmãos que nunca tive e que me ajudaram no desenvolvimento deste trabalho. Agradeço ao meu orientador Prof. Dr. Hélio Guardia pelas suas contribuições e direcionamentos importantes durante o desenvolvimento deste trabalho. Agradeço ao Ministério da Educação, que através da CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) financiou o desenvolvimento deste trabalho. Por último, e não menos importante, agradeço a Deus por ter me dado saúde e sabedoria, além de ser peça fundamental em todas as etapas da minha vida.

"God grant me the serenity to accept the things I cannot change, the corage to change the things I can, and wisdow to know the difference"

Reinhold Niebuhr

RESUMO

A popularização de dispositivos móveis dotados de capacidade de comunicação sem fio possibilita a criação de ambientes onde estes dispositivos interagem diretamente entre si. Essas comunicações ocorrem no modelo P2P, de forma que cada dispositivo pode implementar simultaneamente papéis de cliente e de servidor. Contudo, para que ocorram interações diretas entre dispositivos através de aplicações, é preciso que estes dispositivos implementem algum mecanismo de descoberta. Atualmente, a maioria das aplicações que se comunicam diretamente utilizam informações pré-configuradas para identificação de dispositivos e serviços. Uma forma utilizada para interação entre dispositivos é através da oferta e consumo de serviços utilizando a arquitetura orientada a serviços (SOA), baseada em requisições HTTP utilizando os padrões REST ou SOAP. Um problema recorrente para consumidores de serviços é a identificação de serviços disponíveis. A identificação utilizada em protocolos de descoberta existentes baseia-se apenas no nome do serviço, salvo em comunicações pré-configuradas, o que não apresenta flexibilidade para descobrir novos serviços. De forma a facilitar a troca de informações entre dispositivos, este trabalho propõe um modelo em que interações diretas entre dispositivos sejam centradas no conteúdo envolvido na interação e nas ações que se deseja realizar sobre eles. Para tanto, uma identificação de serviço pode ser baseada em metadados que são adicionados às descrições de serviços existentes, ou em informações obtidas com protocolos de descoberta de serviço existentes. Para avaliar o modelo proposto, esse trabalho apresenta um estudo sobre i) a viabilidade de interações diretas entre dispositivos, considerando suas mobilidades; ii) o uso de um modelo de interação centrado em conteúdo e ação; iii) o desenvolvimento de um *Middleware* para simplificar o desenvolvimento de aplicações que usem o modelo de serviço proposto. Os resultados de simulação obtidos mostram que o modelo é viável. Além disso, uma versão preliminar do *Middleware* proposto foi avaliada e mostra que a interação direta entre dispositivos pode ocorrer de forma oportunística e espontânea.

Palavras-chave: Arquitetura orientada a serviços, SOA, SOAP, REST, descoberta de dispositivos e serviços, P2P, Middleware, Comunicações centradas no conteúdo, Computação Ubíqua.

ABSTRACT

The popularization of mobile devices capable of communicating via wireless network technologies allows us to consider different scenarios in which these devices may autonomously interact with each other. The envisioned communications would occur in a P2P fashion, as each device could simultaneously provide and consume services. A mechanism for dynamically discovering nearby devices and the available services would be necessary. Although a few existing applications already provide the direct interaction among devices they are purpose-specific and rely on pre-configured information for identifying other devices. A service-oriented architecture (SOA), based on HTTP requests and the REST or SOAP protocols, is commonly used in this type of communication. However, automatically finding available known services is still challenging. Service discovery is usually based exclusively on service name, which is not very flexible. This work proposes a new model for the direct interaction between computing devices. In an attempt to facilitate service discovery and selection we propose a content centric model in which interactions are defined according to an object's type and the action to be applied to it. The proposed approach can work atop of existing discovery protocols, based on extensible metadata fields and on existing service data. Our proposal is evaluated according to i) the viability of direct communication between nearby devices, even when carried by users or associated to vehicles; ii) the proposed service discovery and matching using the content centric approach; iii) the effectiveness of a middleware to support the development of generic applications for direct device communication. Simulation results show our proposed model is viable. A preliminary implementation of the middleware was also evaluated and the results show that spontaneous, opportunistic, service-based interactions among devices can be achieved for different types of services

Keywords: Service Oriented Architecture, SOA, SOAP, REST, device discovery, service discovery, P2P, Middleware, Content Centric Communication, Ubiquitous Computing

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

Capítulo 1

INTRODUÇÃO

A computação está cada vez mais inserida no cotidiano, de forma ubíqua e pervasiva. Computação ubíqua está relacionada à presença e à possibilidade de uso de dispositivos computacionais em todos os lugares (POSLAD, 2009). De maneira relacionada, a computação pervasiva está associada à incorporação de capacidade de processamento e comunicação em objetos e em ambientes. Para tanto, tem sido fundamental a evolução dos processadores, que proporcionam capacidade de execução cada vez mais significativa em função de seus custos e consumos de energia. Também as tecnologias de transmissão sem fio têm evoluído e proporcionado capacidade de comunicação em taxas elevadas e mantendo a preocupação com gastos de energia.

Como resultado, novas formas de aplicação da computação têm surgido, como a Internet das Coisas (IoT) (HOLLER, 2014), em que diferentes tipos de objetos podem incorporar capacidade de processamento e de comunicação em rede. Diferentes tipos de dispositivos são utilizados para coletar e processar dados de objetos ou ambientes, como sensores, microprocessadores, dispositivos de entrada e saída de dados ou mesmo smartphones. Como consequência da incorporação destes dispositivos, os ambientes se tornam mais inteligentes (SAHA; MUKHERJEE, 2003). Em cenários urbanos, por exemplo, a coleta e o uso de diferentes tipos de informações associadas aos serviços públicos deu origem ao conceito de Cidades Inteligentes (ROSCIA; LONGO; LAZAROIU, 2013). Aplicações das cidades inteligentes têm evoluído em função das inúmeras aplicações acessíveis via dispositivos computacionais móveis pelos cidadãos.

Para que seja viável e eficiente, contudo, a incorporação de inteligência e capacidade de comunicação em objetos e nos ambientes deve ocorrer sem causar rupturas no modo de ação de seus usuários. Observa-se que as tecnologias mais profundas são aquelas que desaparecem ao entrelaçar-se no cotidiano, até se tornarem imperceptíveis (WEISER, 2002). Para tanto, novas tecnologias comumente passam por uma fase de associação entre o uso e os resultados obtidos, até que não seja preciso agir conscientemente para utilizá-las. Do ponto de vista de suas utiliza-

ções, tecnologias calmas se comportam de maneira intuitiva e sem chamar a atenção do usuário para a forma de acesso (WEISER; SEELY BROWN, 1995).

Uma forma de acesso a informações e dados utilizada em ambientes com dispositivos e nas cidades inteligentes são os serviços Web (W3C, 2004). Utilizando requisições do protocolo HTTP (FIELDING, 1999) no modelo REST (FIELDING, 2000), diferentes aplicações encontram nessa tecnologia uma forma padronizada para o acesso a dados e serviços providos por outros dispositivos computacionais, sejam eles outros computadores ou pequenos processadores incorporados a objetos ou ao ambiente.

O modelo de comunicação baseada em serviços é particularmente útil em cenários em que a proximidade física dos dispositivos comunicantes é relevante. Por exemplo, pode-se pensar na coleta de informações de sensores próximos, no uso de dispositivos computacionais no local onde o usuário se encontra, ou mesmo na interação direta entre dispositivos de usuários. A comunicação através da oferta de serviços é útil para a troca de informações e para o consumo de serviços entre eles. Estas informações podem ser, por exemplo, dados de condições climáticas, situação do trânsito, mensagens com conteúdo privado, e vídeos, ou qualquer conteúdo que se deseja manipular.

A interação direta entre dispositivos começa com a descoberta (*Discovery*) de outros dispositivos que estejam próximos e que tenham capacidade de comunicação em rede. Esta descoberta pode ser realizada por protocolos como UPnP (FORUM, 2008), Bonjour (BONJOUR, 2008) e SLP (ANDREWS, 1998). Após a localização de um dispositivo próximo, é preciso descobrir quais serviços este dispositivo oferece. Alguns dos protocolos que implementam o mecanismo de *Discovery* implementam também o anúncio e a descoberta de serviços, identificando serviços em função de seus nomes. Comumente, a identificação só é possível quando o serviço foi previamente configurado para identificar nomes específicos de serviços.

A identificação de serviços de interesse disponíveis deve ser realizada a seguir, podendo levar em consideração mecanismos de composição de serviços, bem como ontologias para nomeações e descrições (MADANI, 2009). Questões de formatação das solicitações e das respostas também são relevantes na comunicação efetiva entre os dispositivos, assim como aspectos de segurança

Como consequência da evolução das tecnologias que possibilitam a incorporação de capacidade de processamento e comunicação em objetos e ambientes e do modelo de comunicação baseado em serviços, prevê-se cenários em que dispositivos poderão interagir diretamente, em qualquer lugar. Contudo, para que estas interações ocorram de forma calma e profunda, é preciso de um mecanismo que auxilie na identificação de dispositivos e serviços. Além disto, é útil

que este mecanismo saiba interpretar e identificar serviços descobertos, associando demandas e disponibilidades de serviços de forma flexível.

Embora esse modelo de interação oportunística baseado em serviços possa ser implementado com tecnologias conhecidas, essa comunicação ainda não ocorre naturalmente, salvo em cenários previamente configurados.

Nesse sentido, esse trabalho desenvolve uma análise sobre a interação direta entre dispositivos. Acreditando que o modelo de serviços é adequado para comunicações casuísticas, espontâneas e sob demanda, este trabalho tem o objetivo de identificar fatores críticos ao seu funcionamento e criar meios que simplifiquem essa forma de comunicação.

Para ampliar a interação direta entre dispositivos usando serviços, este trabalho apresenta um modelo genérico para essa interação. Para tanto, propõe mecanismos para que serviços sejam identificados através de atributos relacionados com suas funcionalidades e oferece uma plataforma de apoio para a criação de aplicações genéricas para interação via serviço.

De maneira geral, considera-se um modelo centrado no tipo do conteúdo envolvido em uma interação e na ação associada a esse conteúdo. Como resultado, uma única aplicação para interação poderia ser implementada, que seria capaz de manipular diferentes tipos de dados e de diferentes formas.

Para apoiar o desenvolvimento de variações desta aplicação, este trabalho apresenta também um *middleware* que realiza procedimentos básicos para descoberta de dispositivos e serviços de acordo com o modelo proposto. Executado em cada dispositivo móvel, este *middleware* visa a oferecer a descoberta de dispositivos e serviços utilizando diferentes protocolos existentes. Além disso, cabe ao *middleware* intermediar a associação entre serviços demandados e disponíveis, deixando, contudo, que a comunicação ocorra de forma direta entre as aplicações.

Com a utilização desta simplificação para definição de serviços em conjunto com o *middleware* proposto, aplicações podem se comunicar diretamente e facilmente utilizar requisições HTTP para obter diferentes tipos de dados e manuseá-los da forma como for conveniente para estes serviços.

Inicialmente, para avaliar a viabilidade do modelo de comunicação investigado, este trabalho apresenta resultados de um estudo de simulação para a movimentação de usuários e veículos em um cenário urbano. Para tanto, o simulador Omnet++[1] é usado para a avaliação da capacidade de comunicação dos dispositivos, segundo o modelo de serviço, enquanto os dispositivos se movimentam por um cenário urbano. Esse trabalho calcula estimativas de tempos de con-

[1] https://omnetpp.org/

tatos envolvendo os dispositivos e posteriormente obtém e análisa os tempos de contatos dos dispositivos nas simulações realizadas.

A seguir, foca-se nas interações entre dispositivos com mobilidade e, observando os seus comportamentos, tem-se por objetivo aprimorar o desempenho das comunicações entre dispositivos. A melhoria do desempenho se dá principalmente através da economia de requisições durante as trocas de mensagens e consequente economia de energia dos dispositivos móveis, através da taxa de descoberta de dispositivos e do número de consumo de serviços realizados.

Para viabilizar a adoção do modelo de interação oportunística usando um modelo centrado em conteúdo, este trabalho apresenta então, um *middleware* desenvolvido para esta função. Desenvolvido utilizando a linguagem de programação C++, este *middleware* funciona como uma interface para o acesso uniforme a diferentes tecnologias de descoberta de dispositivos e serviços, contando com uma simplificação de definição de serviços em função de conteúdos e ações sobre eles. Este *middleware* possui uma interface que permite a aplicações oferecerem e solicitarem serviços de maneira genérica utilizando alguns protocolos de descoberta estudados.

A utilização do *middleware* foi avaliada em ambiente simulado correspondente a uma cidade com a presença de pedestres e veículos se deslocando enquanto equipados com dispositivos móveis capazes de se comunicar utilizando tecnologias sem fio. Estes dispositivos contam com a presença do *middleware* para realização de descobertas de serviços e contam também com uma aplicação móvel que oferece alguns serviços implementados e que utilizam o *middleware* para oferta e consumo destes serviços.

Os resultados obtidos mostram que essa forma de comunicação oportunística entre dispositivos é viável, mesmo para dispositivos com elevada mobilidade, como veículos, e que a utilização do *middleware* proposto permite a descoberta de dispositivos e serviços utilizando diferentes protocolos.

O restante deste trabalho está organizado da seguinte forma: no capítulo 2 são abordados conhecimentos teóricos e trabalhos relacionados com este trabalho. No capítulo 3 é apresentado o trabalho desenvolvido para obtenção dos resultados. No capítulo 4 são mostrados os resultados obtidos e bem como discussão dos resultados. Finalmente, no capítulo 5 são apresentadas conclusões e propostas de trabalhos futuros relacionados.

Capítulo 2

COMUNICAÇÕES OPORTUNÍSTICAS

Embora mecanismos de endereçamento na Internet possibilitem a comunicação entre quaisquer dispositivos conectados, há certos tipos de comunicação pertinentes apenas entre dispositivos fisicamente próximos. Isso ocorre em situações como na troca de informações diretamente entre os nós, na impressão de documentos, na apresentação de arquivos de múltiplos tipos de mídia.

Dispositivos móveis, por exemplo, podem trocar informações entre si através de comunicação sem fio e podem oferecer serviços para outros dispositivos. Para tal, é preciso que um dispositivo descubra outros dispositivos próximos e quais serviços estão sendo oferecidos por esses dispositivos. Para realizar estas funções de descoberta existem diferentes protocolos, que devem ser executados pelos dispositivos comunicantes.

Cada protocolo utilizado para descoberta de dispositivos e serviços utiliza portas específicas da camada de transporte, tipicamente o protocolo UDP. Alguns protocolos de descoberta utilizam também o protocolo TCP quando o número de mensagens é grande, pois como UDP não é orientado a conexões, chances de ocorrerem falhas na transmissão crescem consideravelmente. Alguns dos protocolos de descoberta trocam informações através de arquivos em formatos padronizados, como o formato XML, enquanto um deles funciona de maneira semelhante a um servidor DNS, de forma que cada serviço é representado por um registro DNS.

Nas comunicações diretas entre dispositivos, considerando que estes possam localizar outros dispositivos e serviços, é preciso ainda identificar os serviços disponíveis e as funcionalidades que oferecem. Alguns serviços são conhecidos através de protocolos específicos, como, por exemplo, a reprodução de mídias em displays através do protocolo UPnP. Este reconhecimento de que o serviço é capaz de reproduzir mídias é possível através da descrição do serviço via um arquivo *XML* descritor do serviço que é disponibilizado pelo serviço. Este identificador é

bem conhecido pois é um padrão utilizado para prestação deste tipo serviço. Porém, embora pesquisados não foram encontrados padrões semelhantes para outros tipos de serviço.

2.1 Descoberta de dispositivos e serviços

Tendo como base a conectividade provida pela arquitetura TCP/IP, protocolos como UPnP, Bonjour e SLP, entre outros, permitem a descoberta de nós ativos no mesmo segmento de rede e de serviços que esses oferecem. Neste sentido, é útil utilizar protocolos que já existem para estas funções, e como estes protocolos já estão estabelecidos é interessante conhecer seus comportamentos para avaliar suas vantagens e desvantagens. O protocolo ZeroConf foi a base para o desenvolvimento de alguns protocolos incluindo alguns dos citados anteriormente.

2.1.1 ZeroConf - Zero Configuration Networking

Zeroconf é um protocolo utilizado para estabelecer automaticamente os parâmetros de uma rede TCP/IP entre computadores em um mesmo segmento de rede. Este protocolo permite, por exemplo, que dois dispositivos sejam capazes de configurar parâmetros de endereçamento de rede a fim de que possam se comunicar. Parte importante da tecnologia está descrita na RFC 3927 (CHESHIRE; ABOBA; GUTTMAN, 2005), que determina a forma para que os dispositivos presentes na rede determinem seus endereços IP. As funções mais importantes do protocolo são: atribuição de endereços para os dispositivos presentes na rede e resolução de nomes.

2.1.2 SSDP - Simple Service Discovery Protocol

O procolo SSDP é um protocolo para descoberta de dispositivos e serviços que trabalha utilizando HTTPU (chamadas HTTP sobre datagramas UDP (*User Datagram Protocol*)). As comunicações de descoberta utilizam *multicast* e os dispositivos anunciam as informações através da porta UDP 1900. SSDP é capaz de estabelecer a comunicação entre dispositivos presentes em uma rede com ou sem serviços auxiliares como por exemplo, um servidor DHCP ou um servidor DNS. O protocolo foi formalmente definido em uma proposta na IETF (Internet Engineering Task Force) no ano de 1999 pelas empresas Microsoft e Hewlett-Packard. Embora o rascunho proposto tenha expirado e não tenha sido padronizado, o SSDP foi incorporado ao UPNP e a implementação final do SSDP está descrita na documentação do UPNP.

2.1.3 UPnP - Universal Plug and Play

UPnP é um conjunto de protocolos cujas especificações são mantidas pelo UPnP Forum, um órgão constituído por mais de 800 empresas. Esses protocolos permitem a descoberta de dispositivos em um mesmo segmento de rede, assim como a descoberta de serviços providos por estes dispositivos. Tanto as portas UDP e TCP quanto o formato das mensagens do protocolo são as mesmas do protocolo de descoberta SSDP, do qual UPnP é derivado. O protocolo consiste em uma sequência de etapas para funcionamento. Estas etapas são mostradas na figura 2.1.

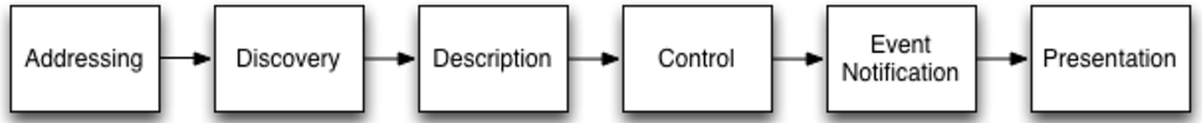


Figura 2.1: Etapas do protocolo UPnP.

A fase *Addressing* obtém um endereço para o dispositivo, o que pode ser feito via os protocolos DHCP (*Dynamic Host Configuration Protocol*), ou ZeroConf. Em seguida, na etapa de descoberta de dispositivos (*Discovery*) é utilizado o protocolo SSDP (*Simple Service Discovery Protocol*). Posteriormente, são obtidas informações dos dispositivos (*Description*) através de arquivos de configuração em formato XML (*Extension Markup Language*). A figura 2.2 mostra um exemplo da estrutura de um arquivo XML utilizado nesse protocolo. Com as informações de um dispositivo é possível conhecer funcionalidades oferecidas e controlá-lo através de funções específicas (*Control*). Durante o controle de dispositivos é possível que existam variáveis relacionadas com a função executada. O protocolo permite que seja realizada uma subscrição aos dados do dispositivo relacionados com a função sendo executada. A partir da subscrição, qualquer mudança nos valores será notificada para os dispositivos subscritos (*Event Notification*). Como última etapa do protocolo, o dispositivo pode disponibilizar uma URL contendo uma página que poderá ser acessada através de algum navegador Web. Através desta página o dispositivo cliente poderá controlar e/ou visualizar o status do dispositivo e de seus serviços (*Presentation*). Devido às restrições de processamento que os dispositivos podem ter, esta fase é opcional.

Na figura 2.2 é mostrado um exemplo de um arquivo descrevendo um dispositivo segundo o protocolo UPnP. Nesse arquivo é possível definir a versão mínima e máxima aceitável para definição do dispositivo utilizando o valor <*specVersion*>. No local designado para descrição do dispositivo (<*device*>) existe um campo dedicado para informação dos serviços oferecidos (<*serviceList*>). Cada serviço especificado (<*service*>) possui um conjunto de valores que o descreve. Na tabela 2.1 são demonstrados os valores que identificam um serviço e seus respectivos significados. Ainda no local para descrição do dispositivo, é possível identificar outros

```
<xml version="1.0"?>
    <root xmlns="urn:schemas-upnp-org:device-1-0">
        <specVersion>
            <major>1</major>
            <minor>0</minor>
        </specVersion>
        <device>
            <serviceList>
                <service>
                    <serviceType>urn:schemas-upnp-org:service:serviceType:v</serviceType>
                    <serviceId>urn:upnp-org:serviceId:serviceID</serviceId>
                    <SCPDURL>URL to service description</SCPDURL>
                    <controlURL> URL for control </controlURL>
                    <eventSubURL> URL for eventing </eventSubURL>
                </service>
                Declarations for other services defined by a UPnP Forum (if any) go here.
                Declarations for other services added by UPnP vendor (if any) go here
            </serviceList>
            <deviceList> Description of embedded devices defined by a UPnP Forum (if any) go here.
                Description of embedded devices added by UPnP vendor (if any) go here
            </deviceList>
            <presentationURL>URL for presentation</presentationURL>
        </device>
    </root>
```

Figura 2.2: Exemplo de arquivo XML de descrição de dispositivo utilizado no protocolo UPnP.

dispositivos que estejam embarcados no dispositivo principal utilizando o valor <*deviceList*>. E caso exista alguma URL para apresentação do dispositivo, esta deve ser definida utilizando o valor <*presentationURL*>.

Valor	**Uso**.
<serviceType>	Tipo de serviço oferecido. Exemplos: reprodução de mídias de audio, vídeo, imagens, entre outros.
<serviceId>	Identificador de um serviço. Deve ser único em um mesmo dispositivo, porém, serviços em dispositivos diferentes podem ter o mesmo identificador.
<SCPDURL>	URL através da qual pode-se obter informações detalhadas sobre o serviço.
<controlURL>	URL através da qual podem-se submeter requisições para controlar um serviço oferecido pelo dispositivo.
<eventSubURL>	URL através da qual são realizadas as subscrições para o status do serviço.

Tabela 2.1: Valores utilizados no UPnP para identificação e especificação de um serviço.

Após o recebimento das informações do dispositivo contida no arquivo XML é possível ter conhecimento de quais serviços o dispositivo oferece. Junto do nome do serviço é fornecida uma URL (*Uniform Resource Locator*) identificada pelo valor <*SCPDURL*>, através da qual é possível realizar uma requisição e saber todas as informações do serviço (que estão serializadas no formato XML).

Na figura 2.3 é mostrado um exemplo de um arquivo XML que descreve um serviço. Neste arquivo XML está presente uma lista de ações (<*actionList*>), onde cada ação é uma função

que o dispositivo que oferece o serviço executa. Na descrição de cada ação são informados os argumentos necessários para execução da ação, sendo cada argumento composto pelo nome do parâmetro, pela direção do parâmetro (entrada ou saída) e pelas possíveis variáveis de estado relacionadas. No arquivo XML também estão descritas as variáveis de estado, às quais os dispositivos podem se subscrever. Quando um dispositivo faz a subscrição à uma variável de estado de outro dispositivo, qualquer alteração no valor da variável é notificada ao dispositivo subscrito.

```
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
    <actionList>
        <action>
            <name> actionName </name>
            <argumentList>
                <argument>
                    <name>formalParameterName </name>
                    <direction> in or out </direction>
                    <relatedStateVariable> stateVariableName </relatedStateVariable>
                </argument>
            </argumentList>
        </action>
    </actionList>
    <serviceStateTable>
        <stateVariable sendEvents="yes">
            <name>variableName </name>
            <dataType> variable data type </dataType>
            <defaultValue>default value </defaultValue>
            <allowedValueList>
                <allowedValue> enumerated value </allowedValue>
            </allowedValueList>
        </stateVariable>
    </serviceStateTable>
</scpd>
```

Figura 2.3: Exemplo de arquivo XML contendo a descrição de um serviço no protocolo UPnP.

Na imagem 2.4 é possível ver o fluxo de mensagens entre dois dispositivos durante a interação de acordo com o protocolo UPnP.

2.1.4 Apple Bonjour

Bonjour é a implementação do conjunto de tecnologias ZeroConf, realizada pela empresa Apple Inc. Em sua implementação são utilizados registros de mDNS (multicast Domain Name System) para descoberta e utilização de serviços. Na ausência de um servidor DHCP o protocolo mDNS utiliza como base o protocolo ZeroConf para atribuição dos parâmetros de configuração. mDNS realiza a tradução de nomes para endereços IP de maneira distribuída. Quando um nó da rede precisa traduzir um *hostname* para um endereço IP, este envia uma mensagem *multicast* para o grupo local pedindo que o nó responsável por aquele *hostname* se identifique. Então, caso um dos nós esteja associado àquele *hostname*, este enviará de volta uma mensagem

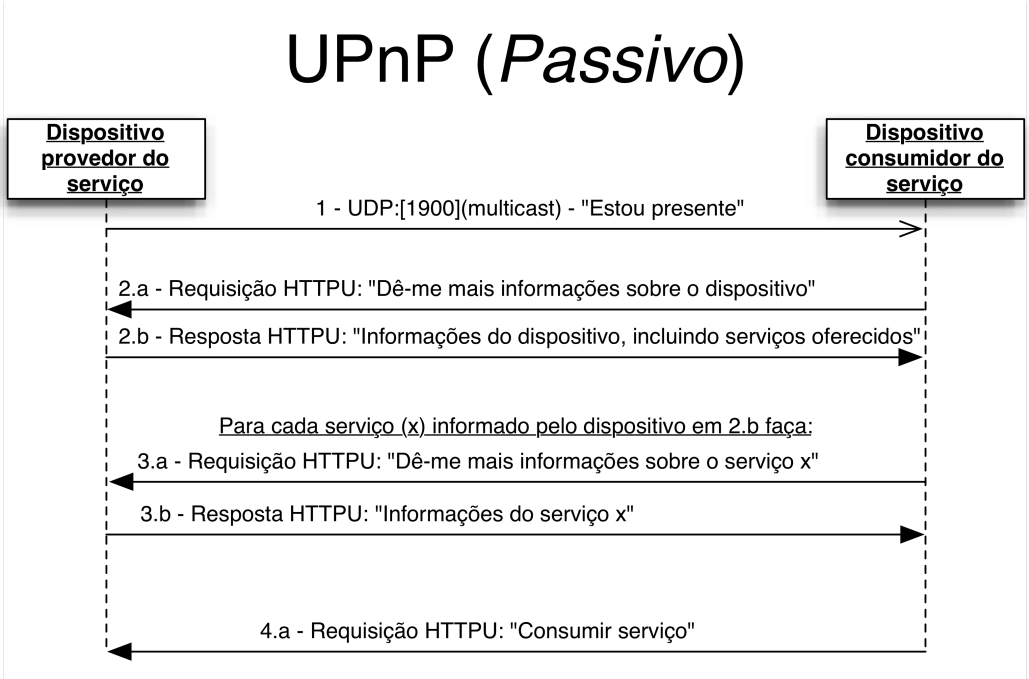


Figura 2.4: Ilustração do funcionamento do protocolo UPnP (as informações entre aspas não estão presentes no protocolo, elas servem apenas para explicitar o sentido de cada mensagem/requisição).

multicast contendo seu endereço IP. Os serviços são registrados em um nó, e as informações necessárias para utilização do serviço estão presentes em registros. Um exemplo de registros em um nó pode ser visto na figura 2.5. Assim, para a descoberta de serviços de um dispositivo (nó) basta requisitar deste nó quais são seus serviços (registros). Esta tecnologia é ativa por padrão em todos os dispositivos vendidos pela empresa Apple e pode ser instalada em outros sistemas operacionais.

Apple LaserWriter 8500._printer._tcp.local	A	192.168.0.68
Apple LaserWriter 8500._ipp._tcp.local	TXT	Port 631
Apple LaserWriter 8500._pdl._tcp.local	TXT	Port 9100

Figura 2.5: Exemplos de serviços armazenados como registros sob o protocolo mDNS.

Na figura 2.5 são ilustrados alguns registros mDNS. Estes registros são compostos pelo *hostname* (**Apple LaserWriter 8500._printer._tcp.local.**), pelo tipo de registro DNS (**A**) e pelo valor do registro (**192.168.0.68**).

Na imagem 2.6 é possível ver o fluxo de mensagens entre dois dispositivos durante a interação segundo o protocolo Bonjour. Normalmente, a abordagem passiva é utilizada para que um nó propague para os outros dispositivos quais serviços oferece. Quando o protocolo troca mensagens de maneira passiva, os dispositivos informam periodicamente aos outros dispositi-

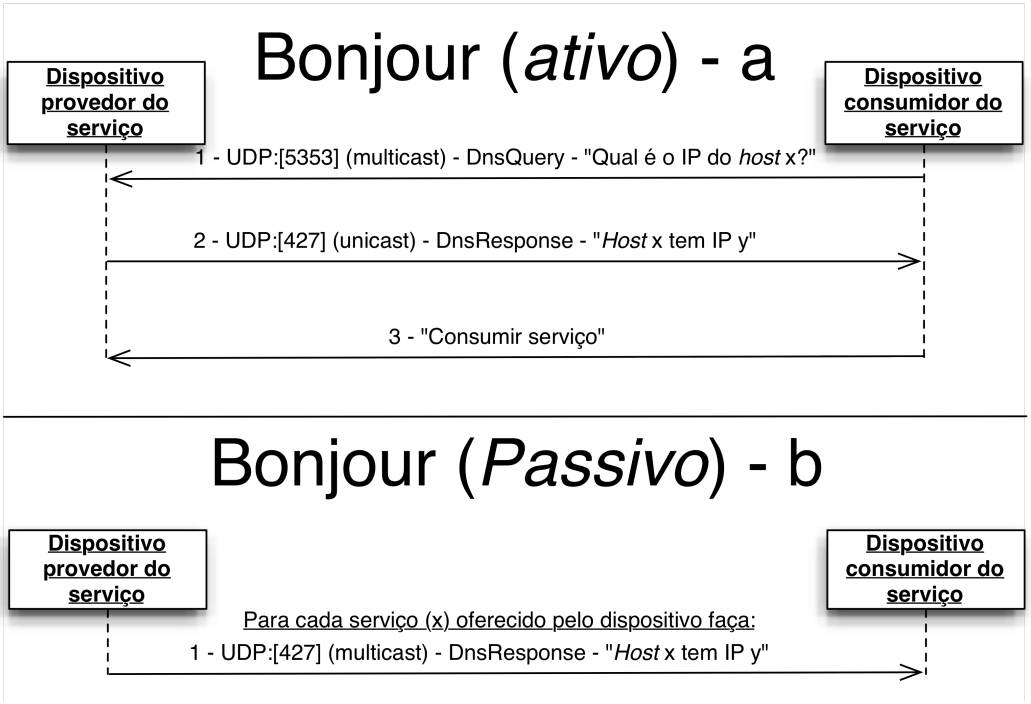


Figura 2.6: Ilustração do funcionamento passivo do protocolo. As informações entre aspas explicitam o significado de cada mensagem enviada.

vos quais registros possuem. Porém, em alguns casos um nó pode explicitamente solicitar um serviço. Quando agindo de maneira ativa o dispositivo envia para os outros nós uma requisição questionando quais os valores para o *host* especificado, e o nó que tiver algum registro responde à requisição.

2.1.5 SLP - Service Location Protocol

O protocolo SLP implementa funções de descoberta de dispositivos e serviços. SLP foi definido na RFC 2608 (GUTTMAN, 1999) e posteriormente extendido pela RFC 3224 (GUTTMAN, 2002). No protocolo SLP são previstos 3 tipos de entidades, a primeiro é a UA (*User Agent*), que é a entidade que busca e consome serviços. O segundo agente é o SA (*Service Agent*) que oferece serviços para os UA's, e a terceira entidade presente no protocolo são os chamados DA's (*Directory Agent*). DA's são responsáveis por registrar os serviços providos pelos SA's, armazenando estes registros. Quando um UA realiza uma requisição de busca por serviços, o DA enviará uma requisição com a localização do SA apropriado. Os DA's foram desenvolvidos com o enfoque em escalabilidade, pois eles agrupam os serviços e permitem maior gerenciamento de requisições e respostas.

O funcionamento do protocolo SLP se dá da seguinte forma: quando um dispositivo UA se junta à rede pela primeira vez, ele envia uma requisição em busca dos DA's existentes. Se

não houver nenhuma resposta, o UA assume que não existe nenhum DA. Quando algum DA está presente, o UA envia requisições diretamente para o DA através de TCP ou UDP, e como todos os serviços oferecidos pelos SA's estão registrados no DA, o DA é capaz de responder essa requisição. Porém, caso nenhum DA esteja presente na rede, o UA irá enviar a requisição em busca do serviço através de um pacote UDP *multicast*, e todos os SA's que possuírem um serviço cuja descrição solicitada for compatível com o serviço oferecido deverão responder à requisição informando que oferecem tal serviço. Um serviço é compatível quando o nome buscado é idêntico ao nome do serviço encontrado e os atributos buscados também são idênticos aos atributos do serviço encontrado.

O protocolo SLP se comunica através de pacotes UDP utilizando a porta 427 e as entidades SA e DA devem ouvir também a porta 427 do protocolo TCP. Esta necessidade existe pois as requisições por serviços podem ser grandes, e a probabilidade de falhas aumenta consideravelmente conforme o tamanho excede o MTU (*Maximum Transmission Unit*) da rede. Quando este for o caso, o UA realizará a requisição através do protocolo TCP.

Na figura 2.7 é mostrado um exemplo da URL de identificação de um serviço. A palavra "service"não é obrigatória, porém com ela é possível agrupar os serviços. Como mostrado no exemplo, é possível encontrar todos os serviços "printer"e não apenas o serviço que implementa o protocolo "lpr". O terceiro parâmetro ("lpr") é chamado de tipo de serviço, e os outros dois parâmetros anteriores ("service"e "printer") são chamados de tipo de serviços abstratos. Na URL de identificação consta o endereço do serviço, que corresponde ao valor "printerAddress"e no caso de um serviço de impressão, o nome da fila de impressão "myQueue".

```
service:printer:lpr://printerAddress/myQueue
```

Figura 2.7: Exemplo de uma URL de um serviço no protocolo SLP.

Na figura 2.8 é mostrado um exemplo da descrição das informações de um serviço de impressão oferecido através do protocolo SLP. As informações são organizadas em uma estrutura de pares chave-valor onde cada par é informado sob parêntesis e separado por vírgula dos demais pares. Dessa forma, o nome do atributo é especificado antes do símbolo = e o valor deste atributo vem posteriormente. No exemplo mostrado o atributo **printer-name** possui valor **Main-printer**.

Na imagem 2.9 é possível ver o fluxo de mensagens entre dois dispositivos durante a interação segundo o protocolo SLP. A abordagem passiva é utilizada para que um nó propague para os outros dispositivos quais serviços oferece e, para tal, é preciso da presença de um DA, poise sem um DA dispositivos não propagam seus serviços para outros nós. Porém, em alguns casos

```
(printer-name=Main-printer),
(printer-natural-language-configured=en-us),
(printer-location=Laboratory printer),
(printer-document-format-supported=application/postscript),
(printer-color-supported=true),
(printer-compression-supported=deflate, gzip)
```

Figura 2.8: Exemplo de uma descrição dos dados de um serviço no protocolo SLP.

o nó pode explicitamente solicitar um serviço, utilizando a abordagem ativa.

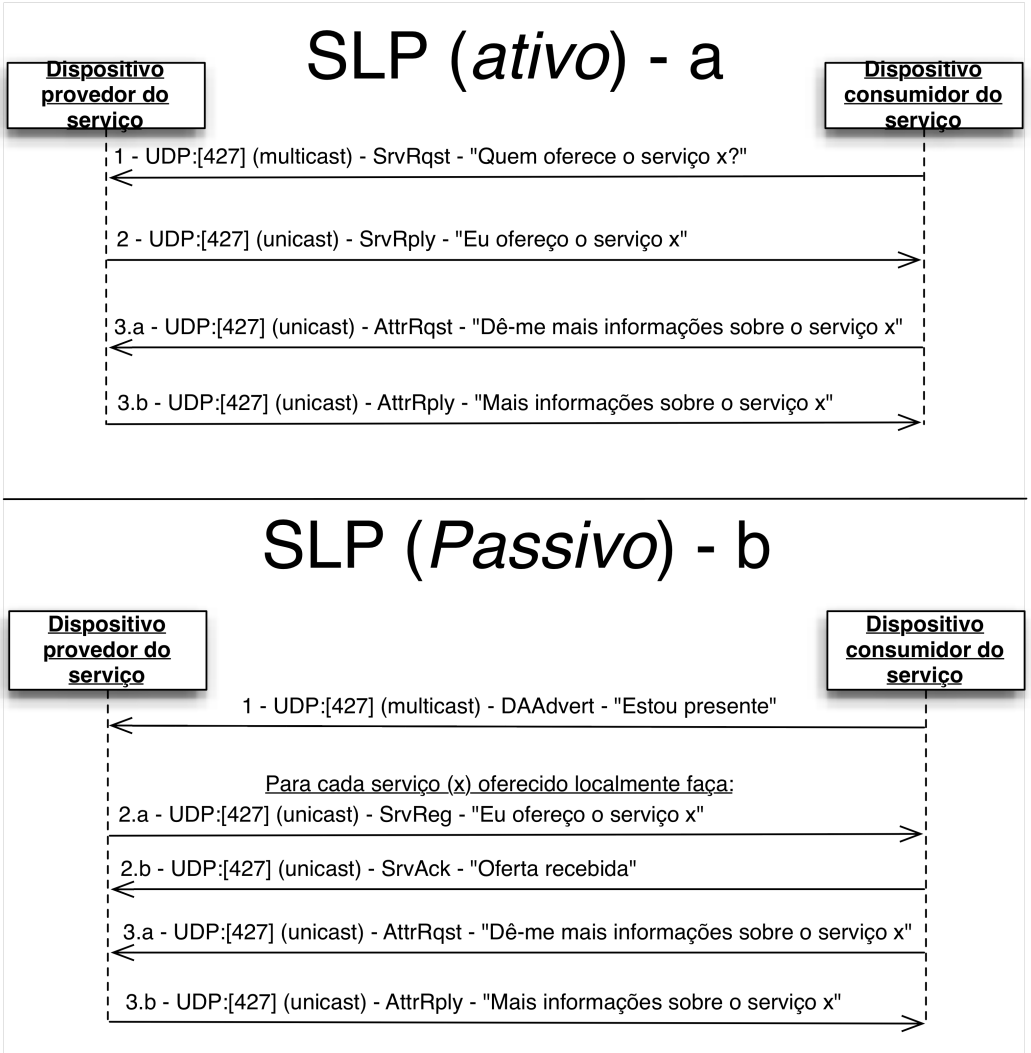


Figura 2.9: Ilustração do funcionamento do protocolo SLP (as informações entre aspas não estão presentes no protocolo, elas servem apenas para explicitar o sentido de cada mensagem).

O protocolo SLP é frequentemente utilizado e suportado em servidores de impressão, como por exemplo o servidor de impressão CUPS (Common Unix Printing System). O protocolo SLP possui suporte para segurança através de criptografia por chaves públicas. Porém, este recurso faz com seja necessário que cada UA tenha a chave pública do SA com que for se comunicar,

o que é um problema para o protocolo, pois o objetivo do protocolo é permitir que dispositivos interajam sem prévia configuração.

2.2 *Middlewares* para interação entre dispositivos

Interações diretas entre dispositivos permitem que dispositivos cooperem e troquem informações entre si através da prestação e consumo de serviços. Para auxiliar esta forma de interação é possível utilizar uma camada de software que realize as funções de descoberta de dispositivos e serviços. Esta camada normalmente está situada entre os serviços básicos de comunicação providos pelo SO e as aplicações, essa camada de software é denominada *Middleware*. Com a utilização de um *Middleware* o desenvolvimento de serviços envolve menos atividades, o que diminui a dificuldade de desenvolvimento.

Um *Middleware* pode abordar diferentes protocolos de descoberta. Além de tratar de protocolos distintos, é possível que o *Middleware* seja capaz de abstrair detalhes da interface sendo utilizada, como por exemplo *WiFi* ou *Bluetooth*. Um *Middleware* pode oferecer também informações de contexto para os serviços, permitindo assim serviços personalizados e mais inteligentes.

Diferentes tipos de *middleware* existem para diferentes sistemas distribuídos. Estes *middlewares* são responsáveis pela descoberta de dispositivos e pelo gerenciamento de informações de contexto, entre outras funções.

Em (Ahmed Surobhi; JAMALIPOUR, 2014) é proposto um *Middleware* que troca informações com outros *Middleware's* em outros dispositivos através de redes *AdHoc* sobre WiFi. O *Middleware* oferece funções para os serviços, como por exemplo disponibilizar os serviços encontrados ordenados pelo número de *hops* necessários para alcançar o serviço, partindo do princípio que quanto menos *hops*, melhor. O mecanismo de descoberta de serviços é composto por requisições (chamadas no trabalho de SREQ) que o *Middleware* envia para todos os dispositivos da rede. Quando um dispositivo que oferece o serviço requisitado recebe a requisição, este dispositivo então envia uma requisição de resposta (chamada naquele trabalho de RREQ) para o dispositivo que enviou a primeira requisição, e posteriormente inicia-se o procedimento de consumo do serviço. O *Middleware* foi desenvolvido para cenários emergenciais, pois nestes ambientes a infraestrutura de comunicação costuma estar danificada e pode não ser utilizável. Logo, permitir que os dispositivos se comuniquem diretamente e encaminhem mensagens um dos outros representa uma alternativa de rede para comunicação e informação dos dispositivos. O *Middleware* desenvolvido no trabalho inicializa automaticamente a comunicação direta

entre dispositivos a partir do momento que é identificada a ausência de infraestrutura. A funcionalidade do *middleware* é avaliada através de simulação, porém o simulador empregado na simulação não é exibido no trabalho. Para a realização da simulação é especificado um cenário retangular de 120 metros por 120 metros. Neste cenário, as pessoas utilizam mobilidade no modelo *Random Way Point* (RWP). Este modelo de simulação de locomoção é conhecido por fazer com que os nós se movimentem em zigue-zague na simulação, o que difere de uma movimentação totalmente aleatória. A simulação ocorre de forma que os nós (pessoas) se locomovem enquanto utilizam a infraestrutura. A partir de um determinado momento ocorre um acidente e a infraestrutura que agora se encontra danificada não mais responde e o *Middleware* presente em cada dispositivo inicia seu modo emergencial. A avaliação do *Middleware* é realizada através da análise da taxa de descoberta de dispositivos e taxa de encaminhamento de mensagens.

Em (LOUREIRO, 2006) é apresentado um *Middleware* para interações em ambientes ubíquos, chamado *Wings*. *Wings* é desenvolvido em Java CDC, que é uma plataforma para dispositivos embarcados. Para realização da descoberta de novos dispositivos é utilizado um *plugin* chamado *Host Discovery Plugin* (HDP), e para descoberta de serviços é utilizado o *Service Provision plugin* (SPP). Wings contém uma interface para adição de componentes em tempo de execução e estes plugins são inseridos desta forma no *Middleware*. Para esta implementação é utilizado um modelo de gerenciamento de componentes chamado *Compour Component Model*. No trabalho, é definido que um serviço é composto por um nome, uma descrição, uma lista de parâmetros e um tipo de retorno. É dito no trabalho que o *Middleware* permite o estabelecimento de chamadas de *callbacks* para eventos relacionados com descoberta de dispositivos e serviços. O *Middleware* também oferece informações de contexto para as aplicações através de requisições. Embora esteja escrito no trabalho que o código esteja disponível, na presente data o endereço *URL* informado não funciona. Além disso, não foi apresentada no trabalho nenhuma métrica de eficiência no serviço ou qualquer dado similar.

Em (CHA; DU, 2011) é apresentada uma proposta de um *Middleware* que auxilia dispositivos móveis em ocasiões de falhas abruptas na comunicação. Para o adequado funcionamento deste *Middleware* é preciso ter um dispositivo com um servidor definido como Mobile Inteligent Server (MIS) instalado neste dispositivo. Nos dispositivos que podem enfrentar falhas na comunicação é preciso instalar o *Middleware* que ao identificar falhas na comunicação tenta se comunicar com o MIS para conseguir uma alternativa para comunicar-se. É dito no trabalho que a comunicação dispositivo-MIS é confiável, enquanto a comunicação com outros dispositivos e serviços pode falhar. O MIS é responsável por ter conhecimento que o dispositivo móvel está saindo do alcance da comunicação e toma medidas para que a falha da comunicação seja informada e o dispositivo possa lidar com a falha nas comunicações. Como restrição para uso

do MIS toda comunicação entre os dispositivos e serviços precisa realizar chamadas especiais ao *Middleware* para poder consumir serviços com sucesso. No trabalho é demonstrado um cenário real onde existem dispositivos com o *Middleware* instalado e possuem comunicação com o servidor MIS. Neste cenário são simuladas situações onde há a falha na comunicação e os dispositivos trocam informações para se adaptar melhor a essas falhas. É exibido no trabalho, que o *Middleware* consegue adiar certas comunicações e restaurá-las no momento que as comunicações forem restauradas.

2.3 Composição de serviços

Composição de serviço é uma técnica utilizada para criação de serviços mais complexos utilizando serviços mais simples. A composição de serviços é útil tendo em vista que combinar serviços e utilizá-los em conjunto dá novas utilidades e funções para estes serviços menores, como mostrado, por exemplo, na figura 2.10. Nesta figura é possível ver um exemplo de composição de serviços. Neste exemplo um serviço maior oferece a impressão de documentos de vários formatos. O serviço conhecido e que de fato realiza a impressão aceita apenas arquivos em um formato específico. Através da composição de serviços e utilizando um outro serviço que realiza a tradução de arquivos entre diferentes formatos, é possível oferecer um serviço de impressão que aceita vários formatos de arquivo para impressão.

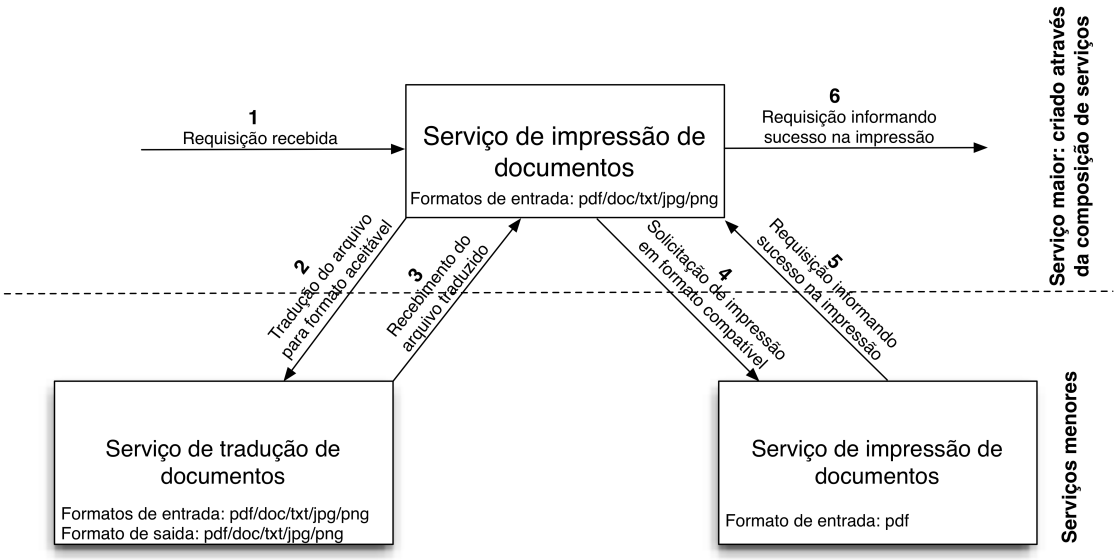


Figura 2.10: Composição de serviços: Um serviço maior é criado através da utilização de serviços menores.

Para compor serviços duas abordagens são comuns: coreografia e orquestração.

A orquestração descreve as interações entre serviços de forma que um serviço é o responsá-

vel pelo controle das interações de todos os serviços participantes da composição. A linguagem mais comum utilizada para orquestração de serviços é WS-BPEL (OASIS, 2007).

Na composição de serviços através da coreografia, serviços cooperam para alcançar um objetivo maior sem que haja um nó central controlando as interações dos serviços. Para realização de coreografias entre serviços a linguagem WS-CDL (W3C, 2005) é comumente utilizada.

Ambas formas de composição de serviços possuem linguagens especificas para especificação dos serviços e das interaçãos que envolverão os serviços. Assim, existe a flebilidade de utilizar o método mais adequado, composição ou orquestração, para cada serviço composto sendo criado.

Através da composição de serviços é possível combinar serviços e prover novas funcionalidades sem a necessidade de realizar alterações nos serviços já existentes. Dessa forma, em ambientes onde cada vez mais funcionalidades computacionais são providas através da arquitetura de serviços, a composição de serviços é útil e permite a disponibilização de serviços com maior complexidade ou mais funcionalidades utilizando serviços menores e mais simples.

2.4 Comunicação sem fio

Utilizar comunicação sem fio é imprescindível para não restringir a mobilidade dos usuários e seus dispositivos. Isto pode ser obtido com protocolos de comunicação sem fio comoWi-Fi (Wireless Fidelity - http://www.wi-fi.org) e Bluetooth (http://www.bluetooth.org). Estes dois protocolos são definidos e mantidos por entidades independentes, Wi-Fi Alliance e *Bluetooth Special Interest Group*, respectivamente. Cada entidade é mantida por representantes de empresas e organizações.

O protocolo *Bluetooth*, padrão IEEE 802.15 é um protocolo para transmissões de curto alcance, que pode variar entre 1 e 100 metros, dependendo da classe do dispositivo (ver tabela 2.2). Dispositivos móveis geralmente estão equipados com *hardwares* de classe 2, que possui um alcance de comunicação de aproximadamente 10 metros. As velocidade de transferência para as diferentes versões do protocolo podem ser vistas na tabela 2.3.

O protocolo *Bluetooth* foi desenvolvido com o objetivo de ser utilizado em *Personal Area Networks* (PAN). PAN's são redes de curto alcance que têm como objetivo conectar dispositivos pessoais de um usuário. O protocolo *Bluetooth* é utilizado, por exemplo, para permitir reprodução de áudio, a transferência de arquivos e controle de periféricos. Por ser desenvolvido com o foco principal de ser utilizado em dispositivos móveis, o protocolo possui otimizações

para economia de energia. Além disto, o protocolo foi desenvolvido para interações diretas entre dispositivos. *Bluetooth* foi desenvolvido com o objetivo principal de ter um baixo consumo energético. Além disto, novas versões do protocolo apresentam ganhos ainda maiores na economia de energia durante transmissões, como por exemplo a tecnologia Low Energy, introduzida na versão 4.0 do protocolo.

Classe	**Alcance típico**.
1	100 metros.
2	10 metros.
3	1 metro.

Tabela 2.2: Classes de dispositivos *Bluetooth*'s.

Versão	**Taxa de transferência**.
1.2	1 Mbit/s.
2.0	3 Mbit/s.
3.0	24 Mbit/s.
4.0	24 Mbit/s + BLE.

Tabela 2.3: Taxas de transferências para as diferentes versões do protocolo *Bluetooth*.

Dispositivos utilizando *Bluetooth* podem se comunicar diretamente. Para que os dispositivos se comuniquem, é preciso que antes de se comunicarem, realizem um procedimento de pareamento. Durante esse procedimento de pareamento (*pairing*, em inglês), um dos dispositivos se torna o mestre (*master*), enquanto o outro dispositivo se torna o escravo (*slave*). Após o pareamento dos dispositivos, ambos os dispositivos podem se comunicar, para tal, são enviados de um dispositivo para o outro conjuntos de bytes. As comunicações utilizando Bluetooth podem envolver mais de 2 dispositivos, contudo, apenas 1 dispositivo será o dispositivo mestre e todos os outros dispositivos serão escravos. Com isso dispositivos escravos podem se comunicar com outros dispositivos escravos, porém, os dados devem ser enviados para o mestre, que encaminhará para o dispositivo escravo de destino, de forma que dois dispositivos escravos nunca se comunicam diretamente.

Outro protocolo bastante difundido e utilizado em dispositivos móveis é o *WiFi*, padrão IEEE 802.11. Este protocolo possui algumas variantes, como por exemplo 802.11.n e 802.11.g, com tecnologias que aumentam o alcance de transmissão e a velocidade de transmissão. A especificação mais popular de *WiFi* tem um alcance típico entre 50 e 100 metros e taxa de trasmissão entre 6 e 54 Mb/s em um cenário ideal.

Usando um ponto de acesso, a tecnologia *WiFi* permite o estabelecimento de uma rede lógica, através da qual dispositivos conectados podem interagir. Também é possível realizar a

comunicação diretamente entre dois dispositivos sem um *access point*. Para tanto, é preciso primeiramente que um dos dispositivos crie a rede sem fio e realize o papel de nó central, sendo o responsável pelo gerenciamento da rede. Uma outra forma de comunicação utilizando *WiFi* é estabelecendo redes *AdHoc* onde o gerenciamento e encaminhamento dos quadros é distribuído entre os nós e não há um nó central. Porém, para a criação de uma rede *AdHoc* que possua encriptação dos dados e um nível mínimo de segurança nos dados transmitidos é preciso que os dispositivos conheçam informações pré configuradas, como por exemplo, a senha da rede.

Para permitir mais facilidade na interação direta entre dispositivos foi proposto o padrão *WiFi Direct* (WIFI-ALLIANCE, 2014), que já possui estabelecido um mecanismo para negociação de parâmetros de segurança da comunicação bem como para a organização dos dispositivos. Desta forma, *WiFi Direct* é uma tecnologia que permite a interação direta entre dispositivos.

Embora diferentes tecnologias das camadas de enlace e rede permitam a identificação de nós no mesmo segmento de rede e a troca de pacotes entre eles, ainda não há um *Middleware* que auxilie nas funcionalidades de descoberta de serviços e dispositivos independente do protocolo de descoberta. Também pode ser útil um *middleware* que ofereça um mecanismo de identificação de serviços mais flexível e intuitivo para oferta/consumo de serviços entre dispositivos próximos. A identificação de serviços de outros dispositivos através de atributos dos serviços também pode ser facilitada pelo *Middleware*.

Este trabalho visa auxiliar na integração dos dispositivos móveis, de forma que dispositivos possam descobrir e oferecer/consumir serviços de outros dispositivos. Propõe-se que estas funcionalidades se deem através da utilização de um *Middleware* apresentado neste trabalho. Além de viabilizar este *Middleware*, este projeto analisa o desempenho desta forma de comunicação, levando em consideração questões como performance de descoberta de dispositivos e serviços, como também a troca de mensagens e os custos que envolvem tal atividade.

Capítulo 3

MIDDLEWARE PARA COMUNICAÇÕES OPORTUNÍSTICAS

Este capítulo apresenta um estudo sobre o modelo de interação oportunística entre dispositivos. Para tanto, propõe a criação de um *Middleware* que se encarregue de detalhes da comunicação direta entre os nós. Este *Middleware* possui uma API para permitir que os dispositivos tenham acesso às funcionalidades previstas. Com a utilização do *Middleware*, um dispositivo pode anunciar serviços oferecidos e também buscar por serviços através de requisições HTTP. O encontro entre dois dispositivos representa uma oportunidade para que ambos se descubram e venham a interagir de maneira cooperativa trazendo benefícios mútuos.

Para validar este modelo de comunicação, foram feitos estudos sobre a viabilidade da execução de serviços entre nós móveis e foi criado um *Middleware* que utiliza este modelo e testa o desempenho das comunicações através de simulações.

3.1 Interação via serviços

Uma forma de interação entre dispositivos é a chamada de serviços, em que um dispositivo oferece recursos através de serviços e outros dispositivos os utilizam.

A identificação de serviços é um problema recorrente para prestadores de serviços computacionais. Uma forma de identificar serviços é observando o número da porta que este dispositivo utiliza para oferecer o serviço, como por exemplo a 80 para HTTP, 22 para SSH, 5900 para VNC, 5432 para o banco de dados PostgreSQL, entre muitas outras. Porém, esta técnica é ineficaz quando os serviços são oferecidos através da arquitetura orientada a serviços, pois todos os serviços realizam a comunicação sobre o protocolo HTTP, que utiliza a porta 80, ou 443 para

HTTPS.

Como tratado em (HUERGO, 2014), algumas técnicas são comumente utilizadas para identificação de *Web services*: comparações de padrões, análise de requisitos, encapsulamento, decomposição de serviços, implementações orientadas a modelos, extração de código fonte, e mapeamento de ontologias. Essas abordagens apresentam algumas soluções para identificação de serviços, porém criam algumas restrições quanto à nomenclatura e à funcionalidade desses serviços.

Visando a desenvolver uma forma de identificação de serviços mais simples para a oferta de serviços, este trabalho propõe uma abordagem que se baseia em duas informações para definição do serviço. A primeira informação é o que é manipulado pelo serviço, como por exemplo arquivos de texto, imagens, áudio e etc. A segunda informação é o tipo de ação que será realizada sobre os dados, definida pela primeira informação, como por exemplo execução, impressão, salvar os dados para posterior obtenção entre outros.

Utilizando estes dois dados, tipo e ação, é possível simplificar a identificação dos serviços pois estes podem ser classificados de acordo com a ação que realizam e com o tipo de dados que manipulam.

Um dado importante é o tipo de dado que o serviço consome/oferece. Para classificar os dados foi identificado que já existem classificações de dados *Mime-type* utilizadas em Web-Services para descrever o conteúdo dos dados enviados/recebidos. Esta forma de classificar os dados já é utilizada há mais de 10 anos e tem se mostrado adequada para classificação dos dados que são enviados e recebidos. Porém, apenas classificar os dados não é suficiente para identificar um serviço, é preciso levar em consideração a ação que o serviço irá realizar sobre estes dados. Desta forma é proposto utilizar valores *(Action)* que determinam quais as ações que um serviço pode realizar sobre um tipo de dado. Com estes dois valores é possível extrair informação para conhecimento razoável de um serviço, compreender o funcionamento deste serviço e utilizá-lo.

Este modelo de definição de serviços não é capaz de atender todos os tipos de serviços que podem ser oferecidos por todos os tipos de dispositivos computacionais existentes, mas é destinado para serviços que estão relacionados com dispositivos pessoais como por exemplo *smartphones*, *tablets* e afins. E ao tratar de interações entre dispositivos próximos, esta simplificação proposta neste trabalho é flexível para atender estes tipos de serviço. A utilização desta simplificação para prestação de serviços é útil pois permite interações entre os dispositivos na forma de serviços com a participação do *Middleware* desenvolvido.

O tipo de dado que o serviço manipula é definido da mesma forma que WebServices definem os dados enviados nas requisições. Os tipos de dados são definidos como *Mime* (Multipurpose Internet Mail Extensions) *types* (FREED; BORENSTEIN, 1996), que também são conhecidos como *Content-Types*. Alguns exemplos de *MIME-types* são *text/plain* para dados textuais sem formatação, *video/** para vídeos em geral, *application/json* para dados serializados no formato JSon, entre muitos outros existentes. O modelo desenvolvido utiliza também qual é ação que o serviço realizará sobre o objeto enviado para o serviço, e esta ação é aqui chamada de *ACTION*.

A definição de serviços baseada no tipo de dado e na ação torna desnecessário ter conhecimento sobre aspectos de implementação do serviço, pois a forma como é implementado um serviço que imprime um arquivo, por exemplo, não faz diferença desde que o arquivo seja adequadamente impresso. É importante ressaltar que embora os serviços sejam desacoplados da implementação é possível disponibilizar para o cliente alguns parâmetros que tornam o consumo do serviço personalizável, como por exemplo se o cliente deseja que a impressão seja realizada frente e verso. Estes parâmetros são valores atributos chave/valor que podem ser informados junto do serviço através do *Middleware*.

Na tabela 3.1 é possível visualizar alguns valores de *MIME-types* disponíveis para utilização no *middleware*.

Valor	**Descrição**
MIME_IMAGE	Este valor representa objetos que sejam imagens (.png, .bmp, .jpg).
MIME_AUDIO	Este valor representa objetos que sejam áudio (.mp3, .aac).
MIME_VIDEO	Este valor representa objetos que sejam vídeos (.mp4, .avi).
MIME_PLAIN_TEXT	Este valor representa objetos textuais.
MIME_JSON	Este valor representa objetos de texto serializados no formato JSon.
MIME_XML	Este valor representa objetos de texto serializados no formato XML.
MIME_FILE	Este valor representa qualquer arquivo.
MIME_AUDIO_STREAM	Este valor representa objetos de áudio que serão enviados através de *Stream*.
MIME_VIDEO_STREAM	Este valor representa objetos de vídeo que serão enviados através de *Stream.*
MIME_ASTERISK	Este valor representa qualquer tipo de objeto.

Tabela 3.1: Exemplo de valores *MIME-type* e seus respectivos significados.

Estes valores foram definidos com o intuito de abranger os tipos de dados estimados de serem utilizados em interações oportunísticas entre dispositivos. Nas simulações realizadas estes valores atenderam as necessidades do serviços implementados e permitiram que os serviços

fossem oferecidos e consumidos com sucesso.

Na tabela 3.2 são listados alguns valores de *ACTION* estabelecidos, bem como a descrição do significado destes valores.

Valor	**Descrição**
ACTION_PRINT	O serviço imprime os dados.
ACTION_PLAY_VIDEO	O serviço reproduz dados que devem estar em um formato de vídeo.
ACTION_PLAY_AUDIO	O serviço reproduz dados que devem estar em um formato de áudio.
ACTION_PLAY_ANY	O serviço reproduz dados que podem ser tanto vídeo quanto áudio.
ACTION_STORE	O serviço armazena em disco dados que forem enviados para ele.
ACTION_RETRIEVE	O serviço disponibiliza informações através de requisições.
ACTION_STORE_RETRIEVE	O serviço armazena em disco dados que forem enviados para ele, assim como também disponibiliza estes dados através de requisições.
ACTION_EXCHANGE	O serviço realiza a troca de informações através de requisições HTTP *POST* para envio de informações, e de requisições HTTP *GET* para busca e obtenção.

Tabela 3.2: Exemplo de valores *ACTION* e seus respectivos significados.

Estes valores de ACTION, extraídos de exemplos de aplicações com significado atual, são ilustrativos de como se poderia definir um serviço desejado. Qualquer valor significativo para uma aplicação poderia ser usado, contudo, de forma a permitir a seleção de um serviço desejado entre aqueles disponíveis. Dessa forma, aplicações podem utilizar valores específicos de ACTION que seja adequado para seus serviços.

Com os atributos *Mime-type* e *ACTION* explicados é possível definir quais são esses atributos para os serviços implementados para realização das simulações. Para o serviço FileTransfer, o atributo *Mime-type* possui valor *MIME_FILE* e o atributo *ACTION* tem o valor *ACTION_STORAGE_RETRIEVE* e *ACTION_PRINT*, o que significa dizer que o serviço armazena e disponibiliza para clientes qualquer tipo de arquivo, assim como também imprime arquivos. O serviço InformationExchanger possui o atributo *Mime-type* com o valor *MIME_PLAIN_TEXT* e o atributo *ACTION* possui valor *ACTION_EXCHANGE*, o que representa que o serviço aceita e disponibiliza apenas informações puramente textuais. Estes valores estão ilustrados na tabela 3.3.

Além dos atributos Mime-type e ACTION previstos na simplificação, os serviços podem conter metadados relacionados. Todos os protocolos de descoberta abordados possuem suporte

Serviço	***Mime-type***	***ACTION***
FileTransfer	*MIME_FILE*	*ACTION_STORAGE_RETRIEVE*, *ACTION_PRINT*
InformationExchanger	*MIME_PLAIN_TEXT*	*ACTION_EXCHANGE*

Tabela 3.3: Síntese dos atributos *Mime-type* e *ACTION* para os serviços presentes na simulação.

para adição de metadados junto da descrição dos dispositivos. Dessa forma, junto dos atributos MIME-type e ACTION, os mecanismos de descoberta irão divulgar os metadados relacionados. Os metadados relacionados com serviços não são utilizados pelo *Middleware* no procedimento de identificação de um serviço, de forma que apenas os atributos Mime-type e ACTION são considerados. Contudo, os metadados podem ser obtidos pelas aplicações e podem oferecer mais informações sobre os serviços disponibilizados. Como exemplo de metadados reais, são considerados: versão do serviço, reputação do serviço ou dispositivo, classificações do serviço, *tags* (palavras-chave), entre outros.

3.1.1 Integração com protocolos de descoberta existentes

Na interação entre dispositivos, o *Middleware* tem o papel de oferecer os serviços registrados utilizando os protocolos de descoberta que estiverem disponíveis. Para tal, é preciso que o *Middleware* saiba representar nestes protocolos os metadados propostos para permitir a identificação dos serviços de acordo com a simplificação proposta. Na figura 3.1 é possível visualizar a representação dos metadados de simplificação de serviços utilizando o protocolo UPnP. No documento XML em que é descrito o serviço sendo oferecido, existe um campo específico para a adição de metadados, no qual são descritos os valores *Action* e *Mime-type*, sendo desta forma que outros dispositivos podem verificar estes atributos ao descobrir serviços.

A representação dos valores *Action* e *Mime-type* no protocolo SLP é realizada inserindo os atributos junto da definição de metadados sobre o serviço sendo oferecido, que é recebido após a requisição dos atributos do serviço. Dessa forma, junto de outros valores que possam estar presentes no serviço estarão os valores *Action* e *Mime-type* que representam os valores do serviço de acordo com a identificação de serviços proposta. É possível visualizar um exemplo destes valores sendo expressados no protocolo SLP na da figura 3.2.

Na implementação utilizada neste trabalho não são utilizados DA's. Todo dispositivo que deseja oferecer um serviço é um SA e todo dispositivo que deseja consumir um serviço é um UA, e sendo assim, os UA's interagem diretamente com os SA's e não há necessidade da presença de DA's.

```
<service>
  <serviceType>urn:schemas-upnp-org:service:InformationExchanger:1</serviceType>
  <serviceId>urn:upnp-org:serviceId:InformationExchanger:1</serviceId>
  <SCPDURL>/InformationExchanger.xml</SCPDURL>
  <controlURL>/InformationExchanger/Control</controlURL>
  <eventSubURL>/InformationExchanger/Event</eventSubURL>
  <metadataList>
    <metadata>
      <key>
        Mime-type
      </key>
      <value>
        MIME_PLAIN_TEXT
      </value>
    </metadata>
    <metadata>
      <key>
        Action
      </key>
      <value>
        ACTION_EXCHANGE
      </value>
    </metadata>
  </metadataList>
</service>
```

Figura 3.1: Ilustração da representação dos valores *Action* e *Mime-type* no protocolo UPnP em um arquivo XML.

```
(InformationExchanger-device-name=IgorsDevice),
(InformationExchanger-Mime-type=MIME_PLAIN_TEXT),
(InformationExchanger-Action=ACTION_EXCHANGE)
```

Figura 3.2: Ilustração da representação dos valores *Action* e *Mime-type* no protocolo SLP, representados por uma lista de atributos.

No protocolo Bonjour, todos os dados referentes à identificação de dispositivos e serviços são representados como registros DNS. Dessa forma, a única possibilidade de adicionar metadados aos serviços é utilizando registros do tipo TXT. Na imagem 3.3 é possível visualizar uma tabela contendo registros referentes ao serviço *InformationExchange*, onde os registros A e AAAA representam respectivamente os endereços IPv4 e IPv6 do dispositivo que oferece o serviço, e registros TXT representam informações adicionais sobre o serviço. Nessas informações adicionais estão representados os valores de *Mime-type* e *Action*

Domínio	Tipo	Valor do registro DNS
igorsDevice_MIME_PLAIN_TEXT_ACTION_EXCHANGE.local	A	x.x.x.x (IPv4 que oferece o serviço)
igorsDevice_MIME_PLAIN_TEXT_ACTION_EXCHANGE.local	AAAA	*:*:*:*:*:*:*:* (IPv6 que oferece o serviço)
igorsDevice_MIME_PLAIN_TEXT_ACTION_EXCHANGE.local	TXT	Mime-type=MIME_PLAIN_TEXT,Action=ACTION_EXCHANGE

Figura 3.3: Ilustração da representação dos valores *Action* e *Mime-type* no protocolo Bonjour através de registros DNS.

Esta abordagem simplificada para identificação de serviços pode não ser suficiente em alguns casos onde os serviços sejam muito específicos e mesmo atributos *Mime-type* e *Action* únicos podem representar mais de um serviço. Para estes cenários seria possível adicionar no

Middleware uma camada de processamento baseada em ontologias para definição de serviço. Essa camada utilizaria dados e metadados dos serviços descobertos, realizaria processamentos sobre essas informações e extrairia novas informações que poderiam ser utilizadas para identificação e escolha de serviços. Esta camada citada não foi implementada neste trabalho, porém, se desenvolvida pode ser acoplada ao *Middleware* e prover uma interface para adição de novas ontologias, aumentando a capacidade de identificação de serviços do *Middleware* e tornando possível a identificação de serviços mais complexos.

Esta solução ideal ainda não foi encontrada. Porém, com a simplificação apresentada neste trabalho serviços que são oferecidos possuem um mecanismo que auxilia na identificação destes serviços, mesmo não sendo o mecanismo ideal.

3.2 O Middleware e a API para serviços

Para realizar a descoberta de dispositivos e serviços, os dispositivos que estão aptos a interagir devem implementar algum mecanismo de comunicação que permita a troca de mensagens de algum protocolo de descoberta. Há alguns protocolos que implementam estas funcionalidades. Os protocolos utilizam mecanismos diferentes, portas da camada de transporte diferentes e formatos de mensagens diferentes. Mesmo utilizando a tecnologia *WiFi Direct*, protocolos de descoberta continuam sendo necessários pois, embora *WiFi Direct* possibilite o estabelecimento de um canal de comunicação, este protocolo não oferece nenhum recurso relacionado com a descoberta de dispositivos e serviços.

Embora existam alguns protocolos que tratem da descoberta de dispositivos e serviços, estes protocolos não interagem entre si, de forma que um dispositivo só consegue descobrir outro dispositivo que implemente o mesmo protocolo. Desta forma, quando um serviço é oferecido através de um protocolo específico, a única forma de descobri-lo é utilizando o mesmo protocolo.

O *Middleware* desenvolvido é uma camada de software entre o sistema operacional e aplicações baseadas em serviços. Para tanto, oferece as funções de descoberta de dispositivos e serviços, que podem utilizar qualquer protocolo de descoberta de dispositivos e serviços existente. Neste trabalho, o *Middleware* implementa os protocolos UPnP, SLP e Bonjour, sendo capaz de descobrir dispositivos e serviços que estejam próximos e sejam oferecidos através desses protocolos. Dispositivos e serviços descobertos utilizando os protocolos implementados e podem ser obtidos através da API disponibilizada.

Uma vantagem da utilização do *Middleware* é que este permite aos desenvolvedores de

serviços focarem-se no desenvolvimento das funcionalidades de seus serviços e utilizar as funções já implementadas de descoberta de dispositivos do *Middleware* que, por sua vez, utilizará diversos protocolos para propagação de seus serviços oferecidos.

Na seção de trabalhos relacionados no capítulo 2 foram mostrados alguns *Middlewares* que auxiliam na oferta de serviços entre dispositivos. Eles têm abordagens que tratam da descoberta de dispositivos e serviços, porém, não oferecem mecanismos para identificação de serviços além do nome do serviço. O *Middleware* aqui apresentado possui um mecanismo que auxilia na identificação de dispositivos e serviços.

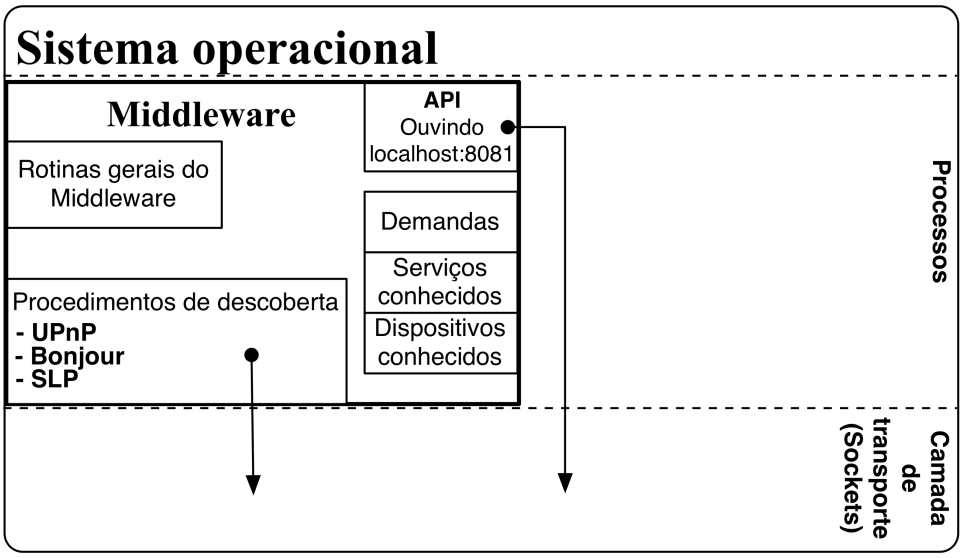


Figura 3.4: Localização do *Middleware* no dispositivo alvo.

Na imagem 3.4 é possível ver a esquematização em alto nível do *Middleware* e como estão organizados logicamente os seus componentes internos. O *Middleware* armazena informações dos dispositivos e serviços conhecidos e demandas registradas. Também estão presentes no *Middleware* códigos que executam rotinas para verificação da conectividade de dispositivos e serviços, como por exemplo quando um dispositivo ou serviço não é mais válido. Essas rotinas também verificam se novos dispositivos ou serviços descobertos são compatíveis com as demandas registradas. Para verificação das demandas, quando são descobertos novos serviços, são realizadas comparações entre os atributos dos serviços buscados e os atributos presentes em cada serviço. Quando é identificado que um serviço possui os atributos buscados, é dito que este serviço é compatível com o serviço buscado e então é enviada uma notificação para o serviço que solicitou o serviço. Também estão presentes no *Middleware*, rotinas que realizam o anúncio dos dispositivos e serviços sobre os protocolos de descoberta disponíveis.

Para oferecer as funcionalidades previstas na API, um servidor local de WebServices embutido no *Middleware*, aceita requisições HTTP locais na porta 80. Como ilustrado na imagem

3.4, o *Middleware* é executado como um processo no dispositivo e utiliza as funções de rede providas pelo sistema operacional do dispositivo para se comunicar.

Cabe ao *Middleware* descobrir dispositivos próximos, no mesmo segmento de rede local, e serviços disponíveis. Seu uso pelas aplicações se dá na forma de chamadas às funções de sua API.

O *Middleware* proposto não tem como objetivo intermediar a prestação de serviços, pois isto criaria restrições na forma como prestadores e consumidores de serviços utilizam o *Middleware*. Qualquer restrição que o *Middleware* criar no encaminhamento de mensagens deve ser tratada pelos serviços, o que cria um acréscimo na complexidade de seu uso. Além do aumento da complexidade, tal ação tornaria obrigatória a utilização do *Middleware* entre os envolvidos nas interações, pois não seria possível que um cliente consumisse determinado serviço sem o uso do *Middleware*. Dessa forma, o *Middleware* atua apenas no anúncio e na descoberta de dispositivos e serviços, deixando os aspectos da forma de interação para os serviços.

Para que um serviço seja anunciado pelo *Middleware*, é preciso que ele seja informado ao *Middleware*, que passa a informar aos dispositivos próximos este serviço oferecido utilizando os protocolos disponíveis. Dessa forma, o *Middleware* pode funcionar apenas como um cache sobre a localização de dispositivos e serviços. Para que o serviço seja registrado, o *Middleware* disponibiliza uma API através de chamadas WebService - Rest.

Para consumo de serviços através do *Middleware*, uma aplicação pode consultá-lo se um determinado serviço está sendo oferecido por algum dispositivo próximo ou pode requisitar quais serviços estão sendo oferecidos na mesma rede. Caso o serviço desejado não esteja disponível, o requisitante tem duas alternativas. A primeira é questionar o *Middleware* novamente depois de esperar um determinado intervalo de tempo, pois é possível que o serviço tenha sido encontrado. A segunda é registrar interesse a determinado serviço no *Middleware*. Este registro de interesse causa a criação de uma demanda (*demand*), pois é registrado que existe a demanda por um serviço ou dispositivo e o requisitante deverá estar atento a qualquer notificação que o *Middleware* enviar sobre a descoberta do serviço buscado. Na figura 3.5, é possível visualizar um diagrama de estados representando a busca de determinado serviço utilizando o *Middleware*.

Para que uma aplicação seja alertada sobre a disponibilidade de um serviço através de uma demanda, esta aplicação precisa implementar uma função que permita ao *Middleware* enviar notificações para as aplicações. Para isso, no registro da demanda o serviço requisitante deve informar em qual porta disponibiliza o seu container Web para envio de requisições. Sempre que o *Middleware* possuir notificações, ele enviará requisições do tipo REST POST para o serviço

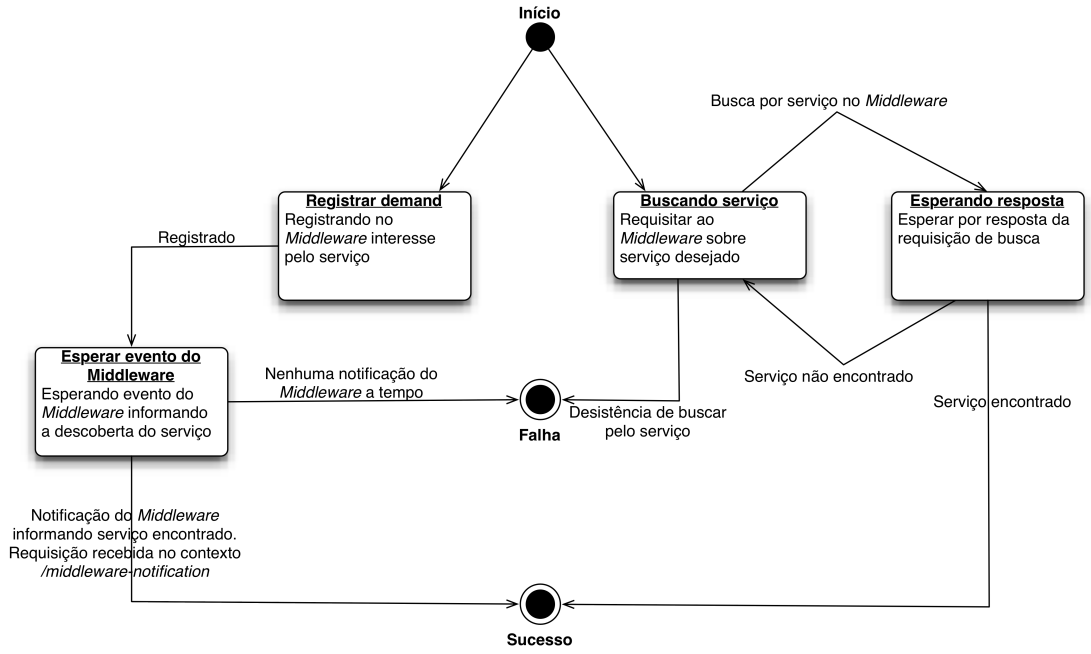


Figura 3.5: Ilustração da busca de determinado serviço utilizando o *Middleware* para descoberta de serviços.

no seguinte endereço: *http://127.0.0.1:PORTA/middleware-notifications*. Para tanto, é necessário que o serviço trate requisições no contexto */middleware-notifications* para poder receber as *callbacks* do *Middleware*. Caso a aplicação que busca por um serviço não possa disponibilizar chamadas REST para aviso da disponibilidade de um serviço, o uso de demandas não é possível. Nesses casos, o recomendado é que a aplicação consulte o *Middleware* periodicamente para verificar se o serviço está disponível.

Através da demanda, sempre que um serviço compatível for encontrado, será enviada uma requisição para a aplicação avisando que o serviço requisitado foi encontrado e agora está disponível. Nesta mensagem enviada para o serviço estão incluídos todos os dados que permitem a localização do serviço. Na figura 3.6 é possível visualizar um documento JSon contendo um exemplo de uma notificação enviada pelo *Middleware*. Também é possível notar que a notificação contém o tipo de demanda criada, o nome da demanda criada, e uma lista de serviços buscados. Para cada serviço buscado, são descritos o endereço do dispositivo, a porta do servidor Web que oferece o serviço bem como o contexto para as requisições, o nome do serviço, uma possível descrição sobre o serviço, uma lista contendo os MIME-types e outra lista contendo os ACTIONS relacionados com o serviço.

Na figura 3.7 é ilustrado o *Middleware* em uma composição lógica mais detalhada. Nesta imagem, são exibidos os módulos e as mensagens que estes trocam entre si. Para oferta das funcionalidades é prevista uma API, oferecida através de um módulo Web, que recebe as re-

```
{
    notification-type : "SERVICE_DISCOVERY",
    demand-name: "searching-for-weather",
    services :
        [
            {
                device-address : "192.0.2.17",
                port : 80,
                context : "/app",
                service-name : "temperature-measurer",
                service-description : "",
                mime-types: [{ mime-type: "MIME_PLAIN_TEXT"}],
                actions: [{ action: "ACTION_EXCHANGE"}]
            }
        ]

}
```

Figura 3.6: Exemplificação de uma notificação que o *Middleware* envia como resposta para uma demanda criada.

quisições HTTP externas, realiza os processamentos relacionados e devolve respostas para os solicitantes. O processamento realizado pelo módulo Web está relacionado com a verificação dos dispositivos e serviços que o módulo de Ciência tem conhecimento. Além disto, o módulo Web pode se comunicar com o módulo de Demanda para buscar um serviço e, quando um serviço sendo buscado não estiver presente, o módulo de demanda deixa registrado o interesse por tal serviço. Esta comunicação através de registros de interesses ocorre seguindo o modelo de comunicação de publicação e subscrição (*publish/subscribe*).

O módulo de demanda tem registros de todos os dispositivos e serviços que estão sendo requisitados localmente. Estes registros podem ser especificados através do nome do dispositivo e/ou do serviço e possíveis metadados relacionados com o dispositivo e/ou serviço. O módulo de demanda pode utilizar mecanismos de identificação para classificação e identificação de dispositivos e serviços utilizando os metadados relacionados. Como descrito neste trabalho, o *Middleware* identifica serviços baseados nos valores *Mime-type* e *Action* presentes nos serviços ofertados. Além desta simplificação descrita, o módulo de demanda pode incluir outras ontologias para identificação de serviços que tornariam a identificação de serviços mais precisa e inteligente.

O módulo de demanda pode trabalhar com requisições de serviços compostas por mais de um serviço. Em alguns casos, a utilização de um serviço está diretamente vinculada com a utilização de outro serviço (composição de serviços). Desta forma, a notificação será enviada para o serviço solicitante apenas quando ambos os serviços estiverem disponíveis, na notificação enviada para o serviço estão presentes dados suficientes para localização de todos os dispositivos que oferecem os serviços requisitados.

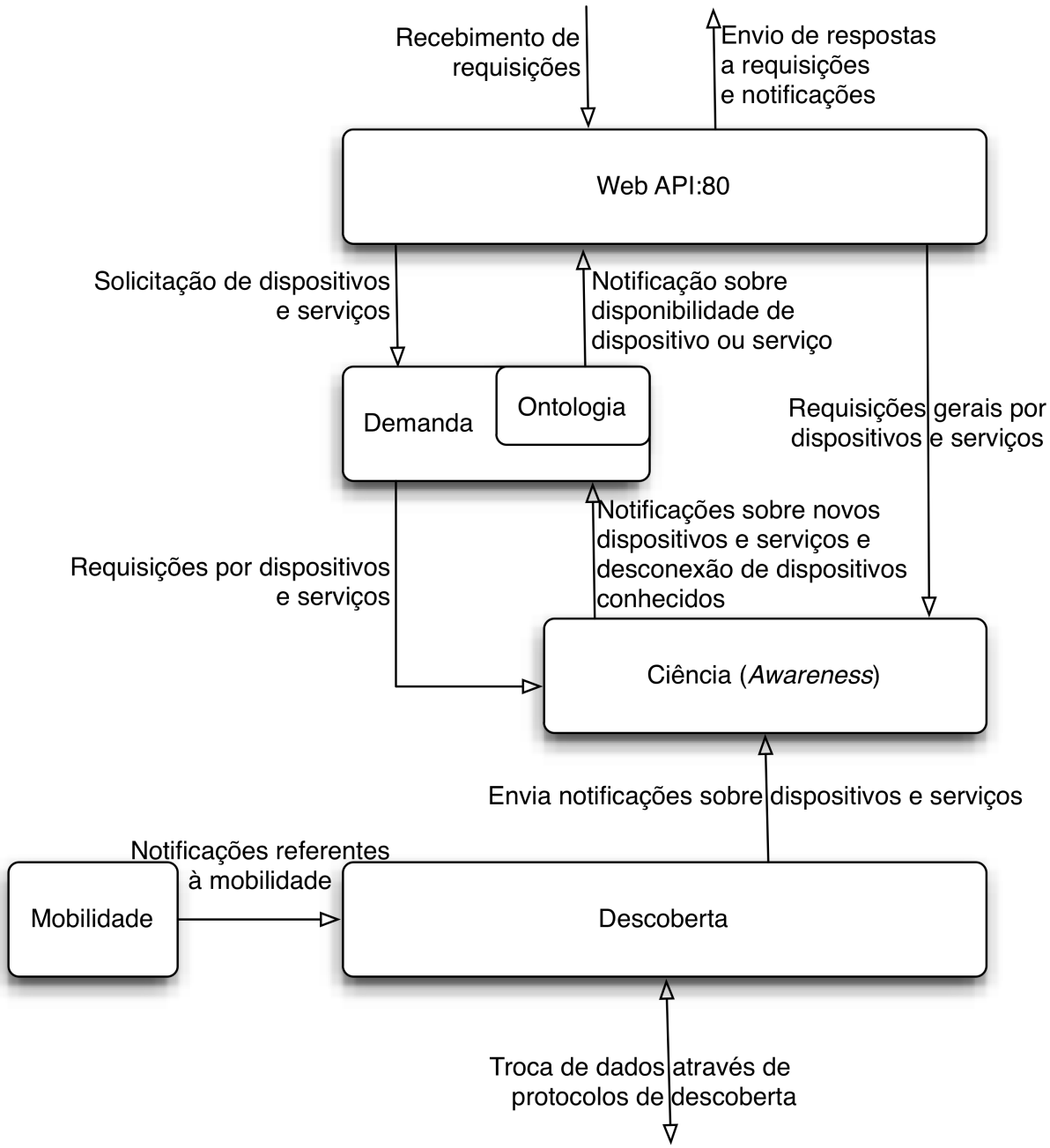


Figura 3.7: Módulos presentes para definição do *Middleware* proposto.

Para que o módulo de demanda possa ter conhecimento dos dispositivos que estão disponíveis, o módulo Ciência o notifica toda vez que um novo dispositivo ou serviço é descoberto, assim como o notifica quando um dispositivo ou serviço não está mais disponível. Um dispositivo é definido como não mais disponível quando não mais envia mensagens de descoberta por um período de tempo superior a um valor limite pré-estabelecido. Desta forma, o módulo de demanda registra os serviços requisitados e, quando for notificado sobre a descoberta de um serviço, analisa utilizando os mecanismos e ontologias disponíveis para definir se o novo serviço é um serviço requisitado. Em caso afirmativo, o módulo envia uma notificação para o módulo

Web do *Middleware*, informando sobre a descoberta de um ou mais serviços solicitados. A notificação recebida será traduzida para o modelo Web e enviada para a aplicação solicitante.

O módulo de ciência envia notificações de atualização apenas para o módulo de demanda, e responde às requisições sobre os dispositivos e serviços existentes que podem ser efetuadas pelos módulos Web e de demanda. O módulo de ciência mantém uma lista de todos os dispositivos disponíveis e possui uma lista de serviços que cada dispositivo oferece. Para os dispositivos conectados e disponíveis, o módulo de ciência possui *timers* para alertar sobre a ausência de dispositivos anteriormente descobertos mas não mais disponíveis. O valor de tempo que os *timers* utilizam é um valor relacionado com a velocidade do dispositivo, de forma que dispositivos mais lentos possuem *timers* maiores, pois o intuito do *timer* é realizar processamento quando o outro dispositivo possivelmente saiu do alcance deste dispositivo. Para que o módulo de ciência possa ter conhecimento sobre os serviços disponíveis, o módulo de descoberta envia notificações para o módulo de ciência toda vez que uma mensagem de descoberta é recebida. Desta forma, o módulo de ciência tem conhecimento da frequência em que um dispositivo envia mensagens e é capaz de determinar se um dispositivo está disponível ou não.

O módulo de descoberta implementa o suporte a alguns protocolos de descoberta e através destes envia mensagens contendo informações sobre o dispositivo e os serviços oferecidos. Simultaneamente com o envio de mensagens, o módulo processa todas as mensagens de descoberta recebidas e extrai informações relacionadas com os dispositivos e serviços ofertados. Durante o processamento das mensagens recebidas, o módulo interpreta documentos no formato XML, pares chave/valor e registros DNS, dependendo de qual protocolo foi utilizado para enviar os dados recebidos. Desta forma, o *Middleware* compreende e descobre os serviços oferecidos por outros dispositivos que utilizam o *Middleware* para anúncio e oferta de serviços. Além disso, compreende e pode descobrir dispositivos e serviços de outros dispositivos que não utilizam o *Middleware*, mas se comunicam através dos protocolos conhecidos pelo *Middleware*.

O módulo de descoberta envia mensagens para anunciar a sua presença para outros dispositivos em intervalos regulares de tempo. Este intervalo é configurável e pode mudar dependendo das circunstâncias que envolvem o dispositivo. É possível, por exemplo, que a mobilidade do dispositivo influencie o intervalo em que são enviadas mensagens de descoberta.

Dessa forma, o módulo de mobilidade previsto possui informações sobre a mobilidade do dispositivo, como a velocidade de deslocamento e direção do dispositivo. Com estas informações o módulo de mobilidade estima um valor ideal para envio de mensagens de descoberta e notifica o módulo de descoberta sobre o intervalo que deve ser utilizado. É útil que este intervalo seja flexível, pois quanto mais rápido um dispositivo se locomover, mais vezes por minuto

devem ser enviadas mensagens de descoberta para descobrir os dispositivos próximos. Caso o intervalo seja maior do que o tempo em que os dispositivos estiveram próximos, dispositivos próximos não serão descobertos.

Na tabela 3.4 é descrita a API através da qual os serviços se comunicam com o *Middleware* e utilizam suas funcionalidades.

Contexto da requisição HTTP	**Tipo**	**Descrição**
/service/register/{service-name}	POST	É registrado um serviço que será anunciado pelo *Middleware*. Deve ser informado o nome do serviço que será registrado. Caso este nome já esteja sendo disponibilizado pelo *Middleware*, será devolvida uma resposta para o solicitante informando que o nome de serviço não está disponível. Neste registro deve ser passado como parâmetro pelo menos um *Mime* e pelo menos um *Action* que descrevam o serviço.
/service/unregister/{service-name}	POST	Remove um serviço que tenha sido registrado no *Middleware*, dessa forma o *Middleware* interrompe o anúncio deste serviço.
/service/search/	GET	Solicita do *Middleware* os serviços que estão disponíveis. Para tal, pode-se passar como parâmetro valores que irão filtrar as buscas por *Mime* e/ou *Action* e/ou nome do serviço.
/service/lookup/{demand-name}	POST	Registra uma demanda no *Middleware*. Na requisição de criação da demanda é possível informar parâmetros que definem o serviço buscado, como por exemplo *MIME-types*, *Actions* ou o nome do serviço. A demanda irá analisar todos os novos serviços que o *Middleware* descobrir e assim que for encontrado um serviço que corresponda aos valores informados, será enviada uma mensagem para o serviço.
/service/lookup-stop/{demand-name}	POST	Remove a demanda (definida através do parâmetro **demand-name**) da lista de demandas existente no *Middleware*.
/device/search/	GET	Solicita do *Middleware* os dispositivos conhecidos. Para tal é possível passar como parâmetro um nome que será utilizado como filtro.
/device/lookup/{demand-name}	POST	Registra uma demanda no *Middleware*. Essa função é útil quando busca-se por um dispositivo específico.
/device/lookup-stop/{demand-name}	POST	Remove uma demanda de dispositivos existente no *Middleware*.

Tabela 3.4: API do *Middleware* para utilização.

Geralmente, a identificação de serviços é realizada pela comparação do nome do serviço encontrado com o nome esperado. Essa forma de identificação não é flexível e não permite nenhum tipo de agrupamento de serviços ou tipos de serviços. Serviços pré-configurados podem adicionar dados ao nome do serviço ou interpretar o nome do serviços encontrados de forma específica. Contudo, estas interpretações específicas são isoladas e utilizadas por aplicações pré-configuradas. Visando possibilitar uma identificação de serviços mais flexível, no momento de comparar serviços buscados com serviços encontrados são utilizados valores que vão além do nome do serviço. Nada impede, contudo, que mecanismos de ontologias sejam incorporados ao *Middleware*. Também podem ser incorporados mecanismos para descoberta de grupos de serviços e composições.

Durante o processo de comparação entre um serviço buscado por um dispositivo e um serviço oferecido por algum dispositivo, este trabalho propõe que sejam analisados 3 grupos de valores dos serviços: *Mime-types*, *Actions* e o nome do serviço. Um serviço buscado é compatível com o oferecido quando o serviço oferecido possui todos valores de *Mime-types* requisitados. Para o serviço ser compatível é preciso também que os valores *Action* do serviço buscado também estejam presentes em um serviço descoberto. Quando for informado um nome no serviço buscado, o nome do serviço oferecido tem que ser idêntico ao serviço que está sendo buscado, caso nenhum nome seja informado, serão utilizados para comparação apenas os valores *Mime-type* e *Action*. Na especificação de um serviço, é possível informar mais de um valor *MIME-type* ou *Action*. Essa característica permite que um mesmo serviço trate de múltiplos tipos de mídia e possa realizar múltiplas ações sobre os dados.

Quando um serviço realiza o registro, o *Middleware* passa a anunciar a disponibilidade deste serviço, e o *Middleware* reserva automaticamente o contexto para requisições HTTP no *Middleware* idêntico ao nome do serviço. Por exemplo, quando um serviço chamado *chat-system* faz o registro no *Middleware*, as requisições HTTP que forem enviadas para o endereço *device_address:8081/chat-system* serão encaminhadas automaticamente para o serviço *chat-system*. Estes encaminhamentos são realizados através de repasses das requisições HTTP para os serviços específicos, realizando um procedimento análogo ao realizado pela biblioteca "mod_proxy", utilizada pelo servidor Web Apache.

Como o *Middleware* leva em consideração questões acerca da identificação de serviços e mobilidade dos dispositivos, têm-se um *Middleware* genérico e flexível que se adapta ao cenário no qual está o dispositivo, oferecendo estes recursos e funcionalidades sem alterar o funcionamento típico de aplicações prestadoras ou consumidoras de serviços.

3.3 Implementação

Para que fosse possível analisar seu funcionamento, desempenho esperado e aspectos relacionados com a prestação de serviços através de dispositivos, o *Middleware* foi implementado em um simulador. Desta forma, o *Middleware* foi implementado em C++ por ser a linguagem compatível com o simulador OMNET++. Alguns serviços para avaliação também foram implementados na linguagem C++. Este simulador possui disponível para utilização vários protocolos de comunicação, como por exemplo UDP e TCP, entre outros.

A arquitetura proposta para o *Middleware* foi implementada em C++, contudo poderia também ser implementada utilizando a linguagem de programação Java. Utilizando esta linguagem, os módulos se comunicariam através de chamadas efetuadas sobre interfaces do próprio módulo. Para utilização dos protocolos de descoberta citados podem ser utilizadas bibliotecas já implementadas e disponibizadas por terceiros através de códigos no formato "jar". Os serviços implementados na simulação também foram implementados utilizando a linguagem C++, contudo estes serviços poderiam ser implementados em qualquer linguagem de programação que tratasse requisições HTTP.

Uma aplicação real que poderia utilizar o *Middleware* poderia ser uma aplicação que distribui arquivos para dispositivos próximos. Esta aplicação deve implementar um container Web para que possa receber notificações do *Middleware*, assim como prestar serviços. A aplicação deve realizar uma requisição HTTP POST para o *Middleware* no seguinte endereço: *http://127.0.0.1:8081/register/file-sharer*, passando como parâmetros dados como Mime-type e Action relacionados com a oferta deste serviço, deve informar também através do parâmetro *"port"* qual a porta que está disponibilizando seu container Web e através do parâmetro *context* informar em qual contexto oferece o serviço. Na figura 3.8 é possível ver estes parâmetros presentes em um documento JSon enviado junto da requisição ao *Middleware*. Em caso de uma resposta positiva, a aplicação tem conhecimento que o *Middleware* está anunciando sua oferta e então aguardará por requisições para consumo do serviço. Em caso negativo, a aplicação pode solicitar um outro contexto e esperar uma resposta positiva.

Para essa aplicação processar as requisições de outros dispositivos que buscam por arquivos, requisições devem conter parâmetros para especificar o nome do arquivo desejado e se a operação é de armazenamento ou obtenção do arquivo. Este arquivo será selecionado para o sistema de arquivos local, ou do sistema de arquivos local, dependendo da operação. A resposta para o serviço que enviou a requisição conterá o arquivo selecionado, ou uma resposta informando sobre o sucesso no armazenamento do arquivo, dependendo da operação solicitada.

```
{
    port : 80,
    context : "/application",
    service-name : "file-distributor",
    service-description : "Service intended to distribute files across devices",
    mime-types: [{ mime-type: "MIME_FILE"}],
    actions: [{ action: "ACTION_STORE"}, {action: "ACTION_RETRIEVE"}]
}

```

Figura 3.8: Conteúdo da requisição de registro de uma aplicação de exemplo enviada ao *Middleware*.

As aplicações são livres para enviarem e processarem quantos parâmetros desejarem, de forma que estes parâmetros tratem de questões como metadados relacionados com os arquivos, filtros, estruturas de pastas, autenticação entre outros parâmetros que auxiliem no funcionamento do serviço oferecido.

Para que este mesmo serviço possa usufruir de outro serviço sendo oferecido, a aplicação deve enviar uma requisição para o *Middleware* no seguinte endereço: *http://127.0.0.1:8081/search*, passando como parâmetro o nome do serviço (*file-sharer*) ou informando os atributos Action e Mime-type correspondentes a um serviço de compartilhamento de arquivos. O *Middleware* recebe esta requisição e busca no módulo de ciência se possui algum serviço compatível com o solicitado e responde a requisição. Como resposta desta solicitação, a aplicação pode receber uma notificação positiva, listando quais os dispositivos oferecem o serviço desejado. A aplicação também pode receber uma resposta negativa, indicando que não existe nenhum dispositivo próximo oferecendo um serviço de compartilhamento de arquivos. O serviço pode esperar um período de tempo e realizar uma nova busca no *Middleware*, ou pode realizar uma requisição no *Middleware* no endereço *http://127.0.0.1:8081/lookup/file-sharer* e passar como parâmetros os dados Mime-type e Action que descrevam o serviço desejado. A partir desta requisição, o serviço espera até receber uma notificação de serviço disponível, informando que foi encontrado um serviço que atende aos parâmetros informados. A aplicação extrai os parâmetros de identificação do dispositivo que oferece tal serviço e envia uma requisição para o serviço com o objetivo de consumi-lo. Após o consumo, o serviço pode enviar uma requisição para o *Middleware* informando não ter mais interesse nos serviços solicitados. Para tal, deve realizar uma requisição para o endereço *http://127.0.0.1:8081/lookup-stop/file-sharer*, informando ao *Middleware* que pare de verificar se os novos serviços descobertos são compatíveis com esta demanda criada. A partir deste momento, a aplicação não receberá nenhuma notificação sobre novos serviços descobertos.

Na imagem 3.9 é exibido um cenário onde dispositivos interagem e utilizam o *Middleware* para buscar por serviços.

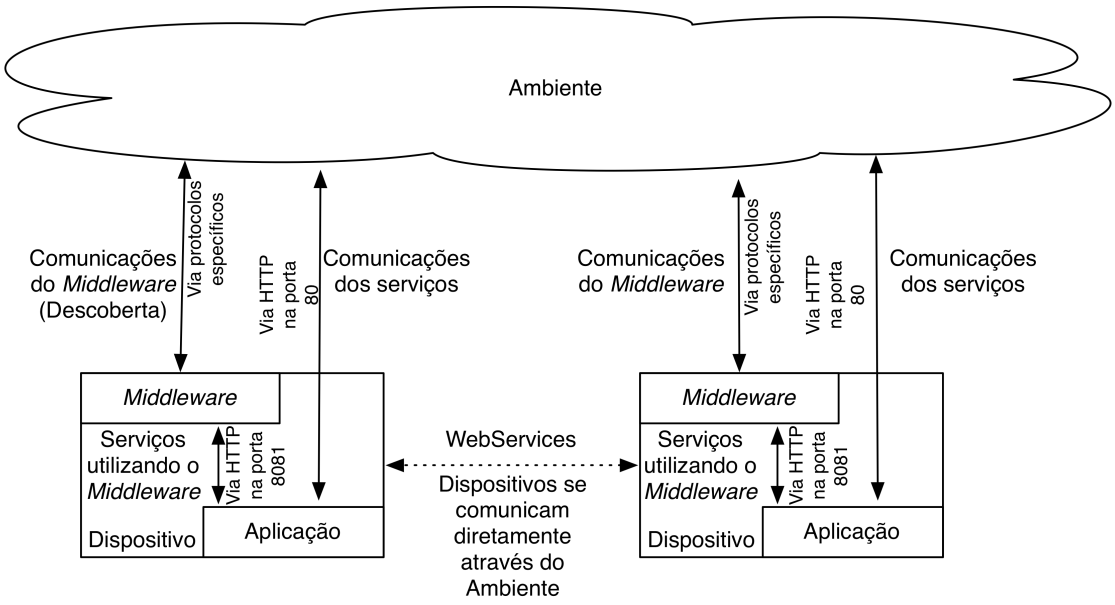


Figura 3.9: Relacionamento de diferentes dispositivos entre si e com o ambiente através de aplicações disponíveis.

Uma aplicação semelhante poderia coletar dados de sensores, utilizar o *Middleware* para oferta e descoberta de serviços e poderia disponibilizar tais dados focando apenas nos aspectos de coleta, distribuição e de processamento de tais dados. Esta aplicação teria que tratar requisições Web e os parâmetros para definir seu funcionamento. Cabe ao *Middleware* realizar tarefas relacionadas com a descoberta e identificação de serviços, permitindo que o serviço desenvolva apenas a parte diretamente relacionada com a manipulação e interpretação dos dados.

Embora seja objetivo deste trabalho estudar a viabilidade do modelo apresentado e o impacto da mobilidade na descoberta de dispositivos e serviços, não é objetivo deste trabalho avaliar a probabilidade de dispositivos encontrarem serviços específicos desejados no ambiente. Portanto, o objetivo deste trabalho aborda apenas identificar a viabilidade do modelo de interações diretas.

Capítulo 4

RESULTADOS E DISCUSSÕES

Este capítulo apresenta uma análise do modelo de comunicação oportunística entre dispositivos via descoberta e uso de serviços. Essa análise é feita analisando simulações, avaliando a viabilidade do modelo. Apresenta também uma avaliação de serviços para nós que se comunicam interagindo com o *Middleware*.

4.1 Estudo de caso

Para avaliar a viabilidade do modelo de comunicação baseado em serviços utilizando dispositivos móveis, estabeleceu-se um cenário urbano para os testes com o simulador. Nestas simulações estão presentes dispositivos que oferecem/consomem serviços e que utilizam a API proposta. Dada a disponibilidade de um modelo da cidade de São Carlos (SP) na base do sistema OpenStreet maps, optou-se por sua utilização nos testes, já que conhecimentos prévios sobre a estrutura desta cidade poderiam prover testes e resultados mais conclusivos.

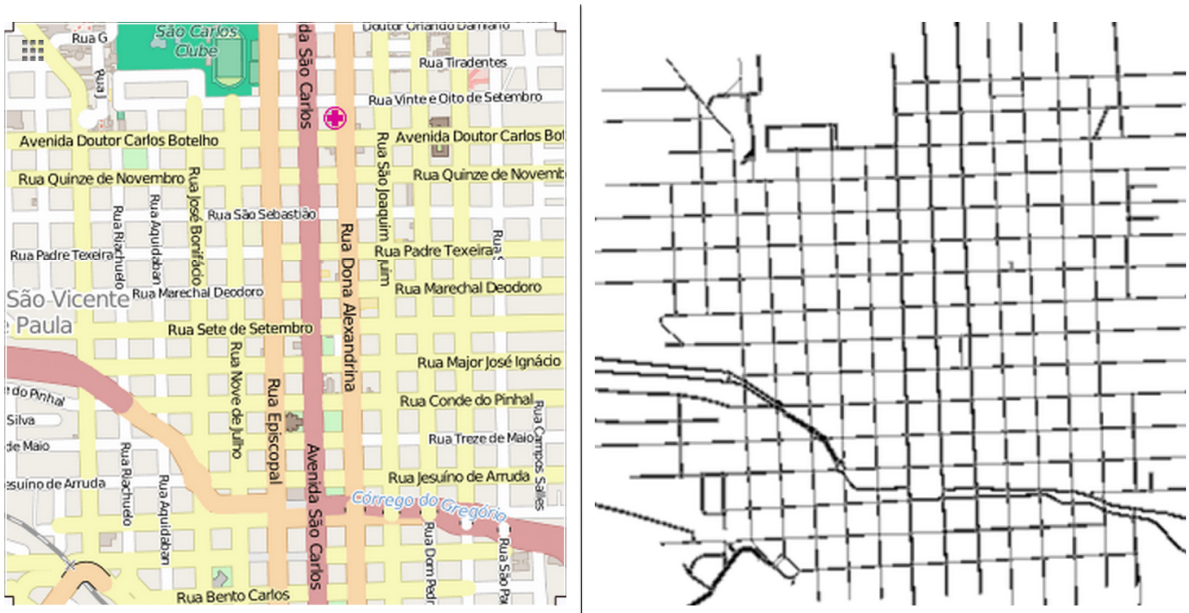
Nos testes, pedestres equipados com *smartphones* e veículos equipados com *tablets* deslocam-se pelo cenário escolhido para simulação. Estes dispositivos utilizam funcionalidades previstas pelo *middleware*. Os serviços presentes nestes dispositivos são serviços que realizam a troca de dados e de arquivos entre os dispositivos.

Para realização da simulação, foi utilizado o simulador de redes OMNET++ em conjunto com o simulador de mobilidade de veículos e pedestres SUMO. Os resultados das simulações permitiram avaliar o número de mensagens enviadas/recebidas, o número de dispositivos descobertos, o número de serviços consumidos, entre outros dados.

De acordo com o IBGE (Instituto Brasileiro de Geografia e Estatística) (disponível em: http://www.ibge.gov.br/home/) a população da cidade de São Carlos no ano de 2014 é estimada

em 238.958 habitantes. O município possui 1.141 Km^2 de área e sua densidade populacional é estimada em 195 hab/Km^2. A área urbana é estimada em 6% da área total de acordo com um estudo realizado pelo IBGE em 2006, o que equivale a 67,25Km^2. Do total populacional do município de São Carlos, estima-se que 95,1% esteja na área urbana, o que corresponde à 227.274 habitantes e gera uma densidade demográfica urbana de 3.379 hab/Km^2. Também é estimado que a população universitária flutuante seja de 16.400 estudantes. De acordo com uma análise de 2013 do IBGE, a frota municipal de veículos é composta de 99.907 automóveis, 10.337 caminhonetes, 3.757 caminhões, 26.816 motocicletas e 782 utilitários.

Para a realização da simulação, foi escolhido o centro da cidade. A área escolhida engloba a maioria das quadras do centro da cidade, região que possui várias lojas e estabelecimentos. O centro da cidade foi considerado para estudos sobre o deslocamento de pedestres e veículos pois representa o deslocamento em um ambiente repleto de dispositivos, onde alguns destes podem consumir serviços oferecidos por dispositivos vizinhos. A região simulada pode ser observada na figura 4.1. É estimado para este ambiente que muitos pedestres se desloquem pelo centro da cidade, e consequentemente ocorram interações entre dispositivos. É estimado também que muitos veículos percorram o centro da cidade e tenham oportunidade de interagir com outros veículos e pedestres.



A - Mapa obtido da organização OpenStreetMaps **B - Mapa obtido em "A", e traduzido para o SUMO**

Figura 4.1: Ambiente utilizado para simulação.

4.2 Simulação de viabilidade e comunicação com Omnet++

O simulador Omnet++ (https://omnetpp.org/) é um simulador de redes orientado a eventos, desenvolvido em linguagem de programação C++. Suas funcionalidades incluem implementações dos principais protocolos de enlace sobre diferentes tecnologias de transmissão como WiFi, Ethernet e Bluetooth. Também há implementação da pilha de protocolo TCP/IP. Assim, Omnet++ permite modelar e simular o funcionamento de diferentes cenários de rede e foi escolhido para as análises deste trabalho.

A definição de módulos e funcionalidades adicionais ao simulador Omnet++ é feita com a linguagem NED (*Network Descritor Language*) criada especificamente para descrição dos módulos que estarão presentes na simulação. NED é utilizada para descrever parâmetros e submódulos, para declarar interfaces de rede, conectar interfaces utilizando *links* de rede, entre outras funções. Os comportamentos dos módulos são programados usando a linguagem de programação C++.

Omnet++ pode ser utilizado nas plataformas Linux, MacOs e Windows. O simulador possui implementação dos protocolos UDP, TCP e seus principais algoritmos de controle de fluxo. O simulador também possui implementação de simulação das tecnologias Ethernet e WiFi(802.11). Parte destes protocolos estão implementados em um projeto externo chamado INET. No desenvolvimento deste trabalho foi utilizada a versão 4.6 do simulador Omnet++.

Na figura 4.2 são apresentados os ambientes de desenvolvimento do Omnet++. Na parte superior da imagem é apresentada a interface que é utilizada para programação e edição dos módulos e outros arquivos de configuração da simulação. Na parte inferior da imagem é possível visualizar a interface de simulação realizada nessas avaliações, os módulos na simulação, o tempo da simulação, o fluxo de mensagens entre os nós (módulos) e a localização dos módulos, entre outras possibilidades. Também nesta interface apresentada vê-se comandos para iniciar/interromper a simulação entre diversos outros comandos de controle sobre a simulação.

Para tornar mais realista a mobilidade das pessoas e dos veículos nas simulações, o simulador SUMO (*Simulation of Urban MObility*) (disponível em: http://sumo.dlr.de/wiki/) foi integrado ao ambiente de teste. SUMO é um simulador de mobilidade que pode ser usado em conjunto com outros simuladores, como o Omnet++. Ao realizar a simulação da mobilidade dos nós, o simulador leva em consideração regras de locomoção, como por exemplo, fazer com que os veículos respeitem as faixas de pistas e semáforos e, no caso de nós que são pedestres, estes se locomovem apenas utilizando calçadas e em faixas destinadas para os mesmos. Para o funcionamento do SUMO é preciso de um mapa que será o ambiente onde os nós irão se deslocar.

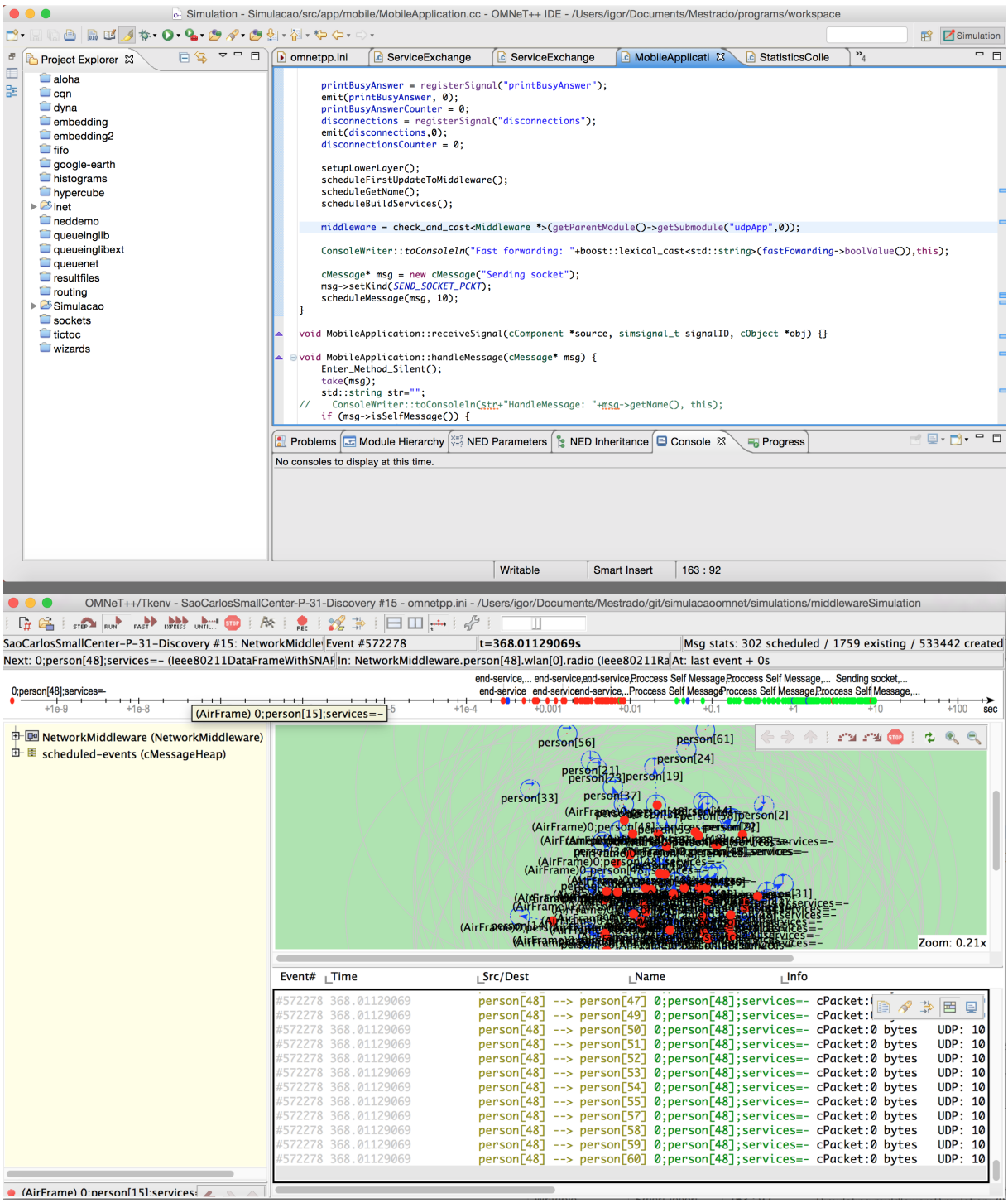


Figura 4.2: Exemplo das interfaces do simulador Omnet++.

Este mapa é descrito na linguagem XML, e como fornecido por http://www.openstreetmap.org, por exemplo. Para que SUMO possa trabalhar integrado com o Omnet++ foi desenvolvido um protocolo chamado TRACI (*TRAfic Control Interface*). Esta comunicação permite que o Omnet++ seja responsável pelo comportamento de rede dos nós e o Sumo seja responsável apenas pela mobilidade destes nós. A junção destes dois simuladores faz parte do *Veins Framework* (http://veins.car2x.org). A interação entre os dois simuladores pode ser visualizada na figura

4.3.

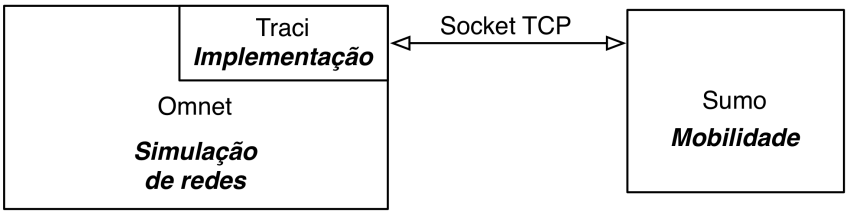


Figura 4.3: Implementação do protocolo Traci para conexão entre simuladores.

4.2.1 Quantidade de dispositivos para cenários com diferentes densidades de dispositivos

Considerando a região para análise da mobilidade, vista na imagem 4.1, foram realizadas 3 configurações de simulações para serem executadas variando a densidade de dispositivos por área. No cenário com densidade pequena (P) tem-se como objetivo analisar o comportamento de dispositivos que se encontram esporadicamente com outros dispositivos e raramente interagem com muitos dispositivos simultaneamente. Para o cenário médio (M) é esperado que os dispositivos interajam com outros dispositivos com frequência e que ocorram interações simultâneas entre diversos dispositivos com frequência elevada. E na simulação com densidade grande (G) é esperado que os dispositivos sempre estejam em contato com outros dispositivos, pertencendo a grandes grupos de interação, de forma que ocorra com frequência a troca de grupo e consequentemente interagindo com grupos diferentes de dispositivos. Para simular estes perfis de interações, o número de pedestre e veículos escolhido para cada caso é mostrado na tabela 4.1.

Cenário	Pedestres	Veículos
P	150	400
M	300	800
G	600	1.600

Tabela 4.1: Densidade dos cenários de simulação.

Estes valores foram escolhidos pois o número de dispositivos em um cenário P representa poucos dispositivos por quadra no ambiente simulado. O ambiente M, por sua vez, possui o dobro de dispositivos por quadra e cada quadra possui em média 10 dispositivos. Já no cenário G cada quadra possui em média 20 dispositivos, o que caracteriza um ambiente com alta presença de dispositivos interagindo diretamente.

Para evitar que os resultados e conclusões obtidas neste trabalho fossem influenciados por alguma anormalidade presente em uma simulação específica, foram realizadas várias simula-

ções para análise dos dados. Cada simulação executada é constituída da quantidade de dispositivos descrita anteriormente, com a diferença que em cada simulação os dispositivos estão em pontos diferentes e possuem destinos diferentes, o que ocasionará rotas diferentes e consequentemente encontros diferentes entre os dispositivos.

4.2.2 Entrada de dispositivos na simulação

Na simulação, os dispositivos são adicionados conforme a simulação se desenvolve, ao invés de adicionar todos os dispositivos no início da simulação. Esta abordagem foi utilizada para simular um cenário onde os dispositivos vão sendo inseridos conforme o tempo passa, semelhantemente a um ambiente dinâmico real.

Para gerar uma taxa de entrada de dispositivos compatível com taxas reais, foi utilizada a distribuição de Poisson, de acordo com a implementação proposta por Donald Knuth (KNUTH, 1973).

Com a utilização de taxas de entradas compatíveis com a distribuição de Poisson, os dispositivos tendem a entrar acompanhados de outros dispositivos na simulação.

Para implementar a inserção de dispositivos na simulação não basta apenas ter a taxa de entrada de dispositivos, é preciso definir também o intervalo entre os eventos de entrada de dispositivos. Para geração dos intervalos foi utilizada uma distribuição exponencial. Ela se mostrou adequada pois apresenta um ápice no número de dispositivos inseridos na simulação e diminui a ocorrência de inserções conforme a simulação avança. O algoritmo em Java utilizado gerar a distribuição exponencial pode ser visualizado a seguir. O parâmetro L determina a dispersão dos valores gerados pela distribuição.

Através da distribuição exponencial, as entradas de dispositivos na simulação ocorrem com muito mais frequência no início da simulação, e conforme a simulação avança, cada vez menos dispositivos são inseridos.

Estes algoritmos foram utilizados em conjunto, de forma que após a geração dos intervalos de inserção utilizando a distribuição exponencial, foi utilizada a distribuição de Poisson para geração do número de dispositivos inseridos em cada evento. Na figura 4.4 é possivel visualizar um exemplo do total de dispositivos que foram adicionados à simulação utilizando estas distribuições, onde o eixo X representa o tempo de simulação em segundos e o eixo Y representa o número de dispositivos inseridos no total. Nesta figura é possível observar que a taxa de entrada de dispositivos diminui conforme a simulação avança, até um determinado momento a partir do qual nenhum novo dispositivo é inserido na simulação. Isso pode ser visualizado na figura 4.4.

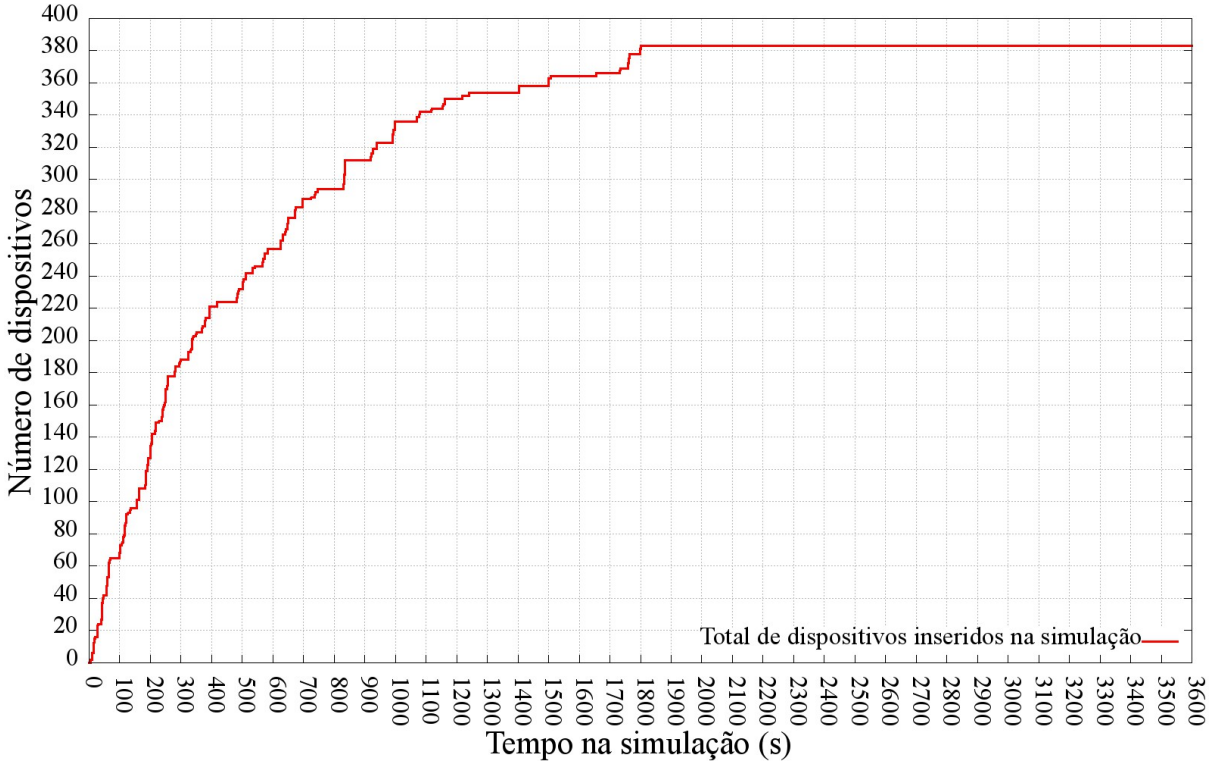


Figura 4.4: Número de dispositivos nas simulações avaliadas dada a entrada de dispositivos de acordo com uma distribuição de Poisson.

4.2.3 Serviços na simulação

Foram implementados dois serviços na simulação com o objetivo de avaliar a viabilidade de suas ofertas e de seus consumos por dispositivos em um cenário típico urbano. Estes serviços utilizam o *Middleware* proposto para serem descobertos e consequentemente consumidos. Estes serviços permitiram obter informações sobre as descobertas e o número de vezes que o serviço de um dispositivo foi consumido.

4.2.4 FileTransfer

Um dos serviços implementados, chamado de FileTransfer, é uma simulação de uma aplicação que realiza transferências de arquivos utilizando *WebSevices* REST. As chamadas do WebService podem ser visualizadas na figura 4.5.

Existem duas funções oferecidas por este serviço, a primeira pode ser utilizada através de uma chamada HTTP do tipo *GET* no contexto */check*. Esta chamada pode ser utilizada para checar a disponibilidade em aceitar novos conteúdos, pois o provedor do serviço pode estar ocupado ou sem espaço para armazenamento e não aceitará nenhuma nova requisição. Como resposta para esta chamada, o requisitante pode receber uma resposta código 200-OK, ou pode

receber uma resposta código 503-BUSY, caso o servidor esteja ocupado ou não tenha espaço suficiente para armazenamento do arquivo a ser enviado. A criação dessa funcionalidade tem como objetivo otimizar o número de transferências de arquivos, pois apenas o cliente que tiver requisitado o consumo do serviço e tiver recebido uma confirmação irá enviar uma requisição com o arquivo. Essa otimização se dá devido à forma de funcionamento das requisições. Em requisições HTTP, para realizar o envio de arquivo não é possível saber previamente o tamanho do arquivo ou se a requisição será bem sucedida, devendo portanto realizar o envio (ocupando o meio físico) para, na resposta da requisição, ter conhecimento se a requisição foi bem sucedida ou não.

A outra funcionalidade do serviço está disponível através de requisições no contexto */consume*. Esta chamada realiza o envio do arquivo para o dispositivo prestador do serviço onde ele será consumido. É importante notar que qualquer uma das ações pode ser realizada livremente no serviço. Porém, é indicado o uso da chamada que testa se o serviço está ocupado para evitar um possível congestionamento do meio físico com dados que podem ser perdidos, conforme explicado no parágrafo anterior.

Este serviço é útil pois através dele é possível estudar a frequência com que dispositivos se encontram e tentam enviar grandes quantidades de dados uns para os outros. Um exemplo de serviço real que pode ser implementado desta forma é um cliente direto que permite o envio de arquivo entre os dispositivos.

Na figura 4.5 são mostradas as possiveis requisições que um cliente pode enviar para o serviço *FileTransfer*. A requisição **a**) é uma requisição que é utilizada para verificar a disponibilidade do serviço, e neste exemplo o cliente recebe uma resposta "OK", que significa que o servidor está aceitando requisições. A requisição **b**) é semelhante à requisição **a**), a única diferença é que o serviço devolve uma resposta para o cliente informando que o serviço encontra-se ocupado. A requisição **c**) é a requisição utilizada para consumir o serviço sendo oferecido por outro dispositivo e consequentemente enviar arquivos para o serviço. Neste caso, o cliente recebe uma resposta 'OK' informando que arquivo foi recebido com sucesso. A requisição **d**) é semelhante à requisição **c**), com a única diferença sendo que o arquivo enviado para o servidor não foi recebido. As requisições presentes no diagrama **e**) representam o fluxo ideal para o serviço, onde o cliente verifica se o servidor está ocupado, recebe uma resposta positiva e então envia uma requisição para consumir o serviço e consequentemente recebe uma resposta informando o sucesso no consumo do serviço.

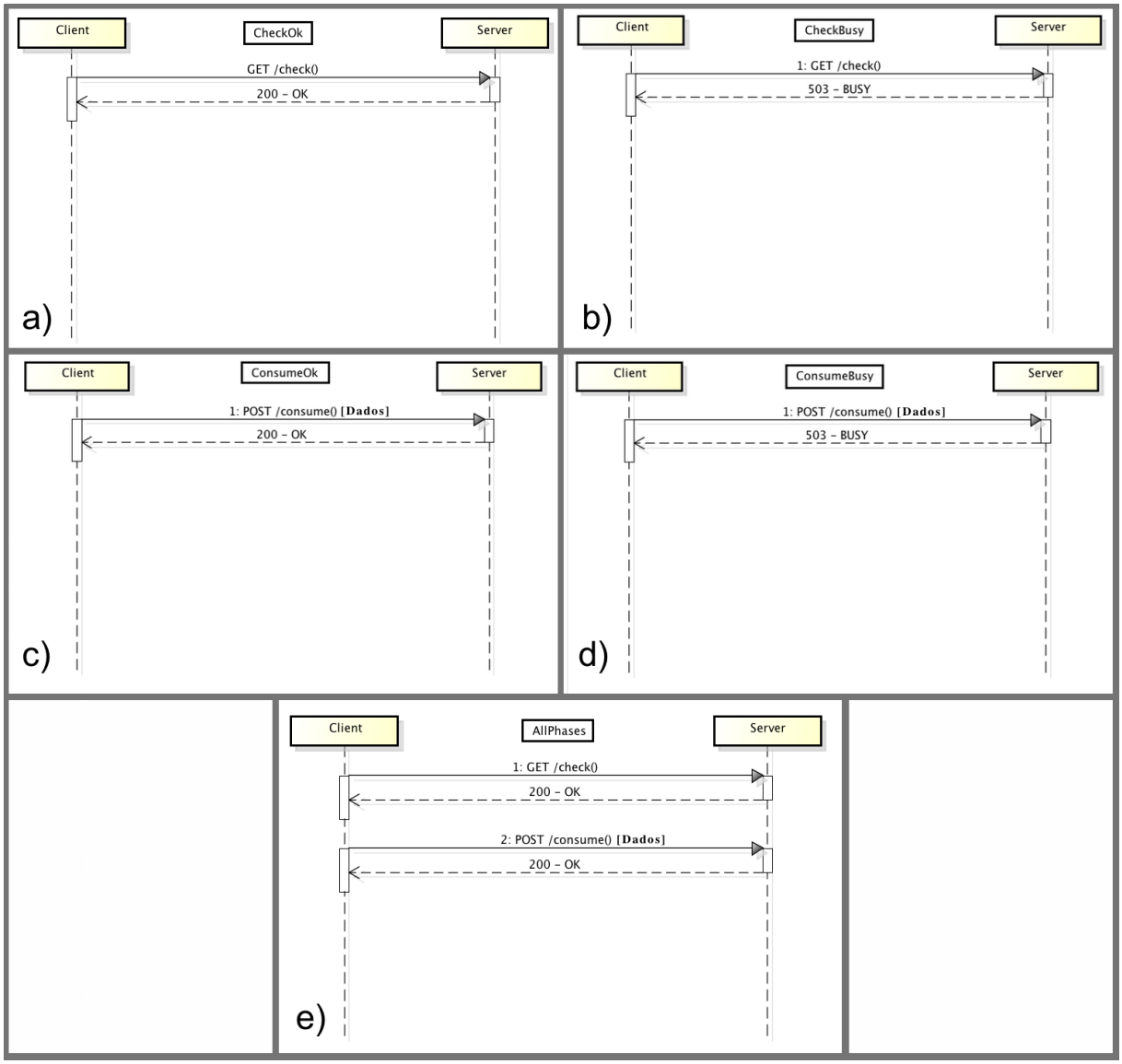


Figura 4.5: Demonstração da API do serviço FileTransfer.

4.2.5 InformationExchanger

Um outro serviço desenvolvido para a avaliação do *Middleware* é uma aplicação que realiza trocas de informações entre dispositivos. Este serviço é chamado de *InformationExchanger* e funciona através de requisições HTTP compatíveis com *WebServices - REST*. O serviço pode trocar qualquer tipo de informação textual na forma de uma sequência de caracteres, porém, para simplificar, as informações geradas estão contidas em três grupos de informações: informações climáticas (*weather*), informações relacionadas ao trânsito (*traffic*), e notícias (*news*).

As funcionalidades deste serviço são oferecidas através de requisições HTTP no dispositivo que oferece o serviço e no contexto do serviço, por exemplo *http://device:80/function*, onde o valor *function* é a funcionalidade do serviço sendo utilizada.

Para descobrir quais informações estão disponíveis em um dispositivo, dois tipos de requisições podem ser utilizadas. A primeira é uma requisição do tipo GET que deve ser realizada sobre o contexto */query*, e que pode enviar como parâmetro da requisição um valor de texto que servirá de filtro nos resultados obtidos. Esta requisição retorna para o cliente uma lista com o ID do registro, o título, a data do registro e o grupo ao qual pertence tal registro. Como exemplo, poderia ser realizada uma requisição para buscar por toda e qualquer informação que estiver relacionada com a palavra chave "ônibus", para tal, deve realizar uma requisição HTTP do tipo GET para o dispositivo no seguinte endereço *http://device:80/query/onibus*. Na figura 4.6 é mostrado um exemplo de como seria o retorno de uma informação do serviço utilizando o formato JSon para serialização. Também é possível requisitar as informações do serviço pes-

```
{
  "id" : "aqflpq02l123lk",
  "title" : "car jam in the 5th avenue",
  "date" : 1430870644686,
  "group" : "NEWS"
}
```

Figura 4.6: Exemplo de retorno de informações do serviço *InformationExchanger*.

quisando através do grupo de informações. Para fazer buscas desta forma deve-se utilizar o contexto */queryg* (abreviação de *query group*) que, com o parâmetro textual informado seleciona um grupo de registros e devolve para o cliente uma lista de registros pertencentes a este grupo. Estes registros são ordenados pela data dos registros e estão estruturados de maneira semelhante ao registros retornados no contexto */query* (visível na figura 4.6). Uma possível requisição para obtenção de todas informações relacionadas com o tráfego seria uma requisição HTTP do tipo GET realizada sobre o contexto *http://device:80/queryg/traffic*. Estas características descritas estão presentes nos serviços implementados na simulação para avaliação do modelo de comunicação apresentado.

Após a etapa de descoberta de quais registros um dispositivo possui, registros específicos podem ser obtidos através de uma requisição do tipo GET no contexto */detail*, passando como parâmetro o ID do registro que se deseja obter. A mensagem de resposta desta requisição é uma lista de registros contendo o ID, o título e o detalhe de cada informação pertencente ao grupo solicitado. Na figura 4.7 é mostrado um exemplo deste tipo de resposta.

Também é possível enviar um parâmetro de requisição chamado *with-details* que recebe valores lógicos de verdadeiro ou falso quando enviando requisições aos contextos */query* e */queryg*. Este valor permite que, ao realizar uma busca, sejam obtidos os valores *details* (ilustrado na figura 4.7) junto dos valores mostrados na figura 4.6. Esta funcionalidade permite que

```
{
  "id" : "aqflpq021123lk",
  "title" : "car jam in the 5th avenue",
  "details" : "Two cars have crashed in the
5th avenue at about 05:15 pm. Creating a
traffic jam of some miles."
}
```

Figura 4.7: Exemplo de retorno de detalhes de informações do serviço *InformationExchanger*.

todos os registros que serão recebidos terão o campo *details* preenchido. Essa funcionalidade pode ser útil, pois como o campo *details* pode ter uma grande quantidade de dados, o recebimento da resposta pode resultar em grandes transferências de dados entre dispositivos, portanto detalhes sobre a notícia devem ser requisitados apenas quando necessário.

Estes dois serviços descritos têm comportamentos bem diferentes. O serviço *FileTransfer* tem comportamento estático, se comparado com o *InformationExchanger*, pois ele atende apenas um cliente por vez. Portanto, embora muitos clientes próximos podem desejar consumir este serviço, apenas 1 irá consumir por vez. Esse comportamento é inserido na implementação do serviço pois o serviço pode necessitar realizar algum processamento relacionado com o arquivo recebido, e que por alguma restrição física ou lógica não pode processar arquivos simultaneamente, o que o impede de aceitar novas requisições até o término do processamento do arquivo. Esta restrição pode ser justificada, por exemplo, caso o serviço realize a impressão do arquivo enviado para o serviço, não podendo assim, imprimir múltiplos arquivos simultaneamente. Portanto, este serviço é importante para o desenvolvimento deste trabalho pois ele analisa dados estáticos das interações de consumo de serviços e avalia o funcionamento do *Middleware* nesses cenários.

O serviço *InformationExchanger* possui um funcionamento diferente do serviço *FileTransfer*, pois é capaz de tanto enviar quanto receber múltiplas requisições simultaneamente. Dessa forma, procurou-se avaliar um serviço dinâmico que trata todas as requisições que recebe, o que parece ser plausível quando muitos dispositivos estão próximos e tentam interagir simultaneamente. Informações relacionadas com o desempenho destes dois serviços avaliam o *Middleware* em cenários com grandes números de requisições, tanto dinâmicos quanto estáticos.

Para garantir que os comportamentos desempenhados pelos dispositivos na simulação correspondam ao comportamento de dispositivos reais, alguns aspectos da implementação da simulação foram estudados em profundidade. Como um exemplo, os serviços descritos acima foram implementados com requisições HTTP, utilizando a porta 80 sobre TCP. Além disso, os protocolos de descoberta de dispositivos e serviços implementados na simulação foram estudados e implementados de acordo com o protocolo real, dessa forma esses protocolos se comportam da

mesma forma que em dispositivos reais. Os protocolos de descoberta presentes no *Middleware* foram implementados de acordo com a especificação, como mostrado no capítulo 2 através das figuras 2.4, 2.6 e 2.9.

4.3 Processamento dos resultados

O simulador Omnet++ possui uma API bem detalhada para coleta de dados da simulação e geração de estatísticas. No entanto, para poder realizar este trabalho e utilizar estes dados foi preciso desenvolver um interpretador para o formato em que o simulador Omnet++ os exporta. Estes dados da simulação são obtidos através de um arquivo de texto. Um exemplo de como esses dados são exportados pode ser visto na figura 4.8. Estes valores representam respectivamente: um ID da origem deste dado, o número do evento que é responsável pelo registro, o tempo em segundos (da simulação) do registro do valor e por último, o valor que foi registrado.

```
1023    483559  345.527996274145    12
1023    489569  347.025243816377    13
1023    495687  348.530241986582    14
1023    501970  350.027669804401    15
```

Figura 4.8: Ilustração dos dados brutos exportados pelo simulador Omnet++.

O interpretador desenvolvido obtém os dados da simulação e, dependendo dos dados que se deseja, realiza diferentes processamentos, como por exemplo a soma dos valores de todos os dispositivos descobertos com o intuito de obter o total de dispositivos descobertos na simulação. Na figura 4.9 é possível ver um exemplo dos dados disponibilizados pelo interpretador.

# numFile	Discovery	person-meanSpeed-max	person-meanSpeed-min	person-totalDiscoveredDevices-sum	person-totalDiscoveredServices-sum
1	0.5	4.514197826	0.154575005	11040	2252
2	0.6	4.488195419	0.163000196	10612	2219

Figura 4.9: Ilustração dos dados obtidos do interpretador.

Após os dados gerais terem sido criados pelo interpretador, é utilizada a ferramenta Gnuplot (disponível em: http://www.gnuplot.info/) para criação de todos os gráficos ilustrados nesta dissertação.

4.4 Implementação da simulação

Para a coleta de dados, o *Middleware* definido neste trabalho foi implementado nos moldes do simulador, sendo o responsável pela descoberta de dispositivos e serviços. Foram implemen-

tados também serviços que simulam a transferência de informações em aplicações modelo, de acordo com as descrições da seção 4.2.

Omnet++ é um simulador discreto, baseado em eventos. Sua operação considera uma linha de tempo, na qual são inseridos eventos associados a cada dispositivo simulado.

Assim, para simular o comportamento de dispositivos e suas interfaces de rede é preciso criar módulos que determinem quando transmissões irão ocorrer. Quando uma transmissão é solicitada no simulador, pacotes são gerados de acordo com a pilha de protocolos selecionada (UDP/IP/WiFi, e.g.). Um evento correspondente ao recebimento desses pacotes é então inserido na fila, levando em considerações os tempos típicos considerados pelo simulador.

Num cenário com vários dispositivos, cabe ao simulador levar em consideração as localizações dos dispositivos e as capacidades de sinalização das interfaces virtuais utilizadas para determinar quais delas receberão os pacotes apropriados. É nesse aspecto que o modelo de mobilidade provido pelo simulador SUMO é relevante nos testes.

Dado um cenário urbano selecionado, SUMO consegue simular o deslocamento de pessoas e de veículos, de forma a respeitar questões dos espaços disponíveis para isso, como calçadas e ruas. Dados da movimentação dos dispositivos simulados gerados pelo SUMO são repassados para que o Omnet++ determine as comunicações.

Para cada pacote recebido por um dispositivo simulado é preciso tratar então das operações do *Middleware* especificado. Para isso, novos módulos do simulador foram implementados, para tratar o comportamento das operações de descoberta de dispositivos e serviços. Todas as comunicações recebidas pelos dispositivos, endereçadas às portas selecionadas dos protocolos de transporte, associadas aos protocolos de descoberta são tratadas pelo módulo implementado.

No simulador Omnet++, todos os objetos que participam da simulação devem ser implementados como módulos. Um módulo é uma classe que define alguns comportamentos necessários pelo simulador. Para um objeto ser um módulo é preciso estender algumas classes específicas definidas pelo simulador. Na figura 4.10 é possível observar uma imagem retirada do próprio ambiente de desenvolvimento do Omnet++. Este módulo representa o dispositivo presente na simulação realizada. Neste módulo exibido é possível visualizar outro módulo interno chamado *middleware*, que é o responsável por realizar o a descoberta de dispositivos e de serviços. Neste diagrama, as linhas representam conexões entre os módulos, e módulos só podem se comunicar através destas conexões. Outro módulo interno é o *mobileApplication*, que representa uma aplicação que simula a existência de dois serviços, descritos na seção 4.2. Mais detalhes sobre os outros componentes que pertencem ao dispositivo podem ser obtidos na tabela

4.2.

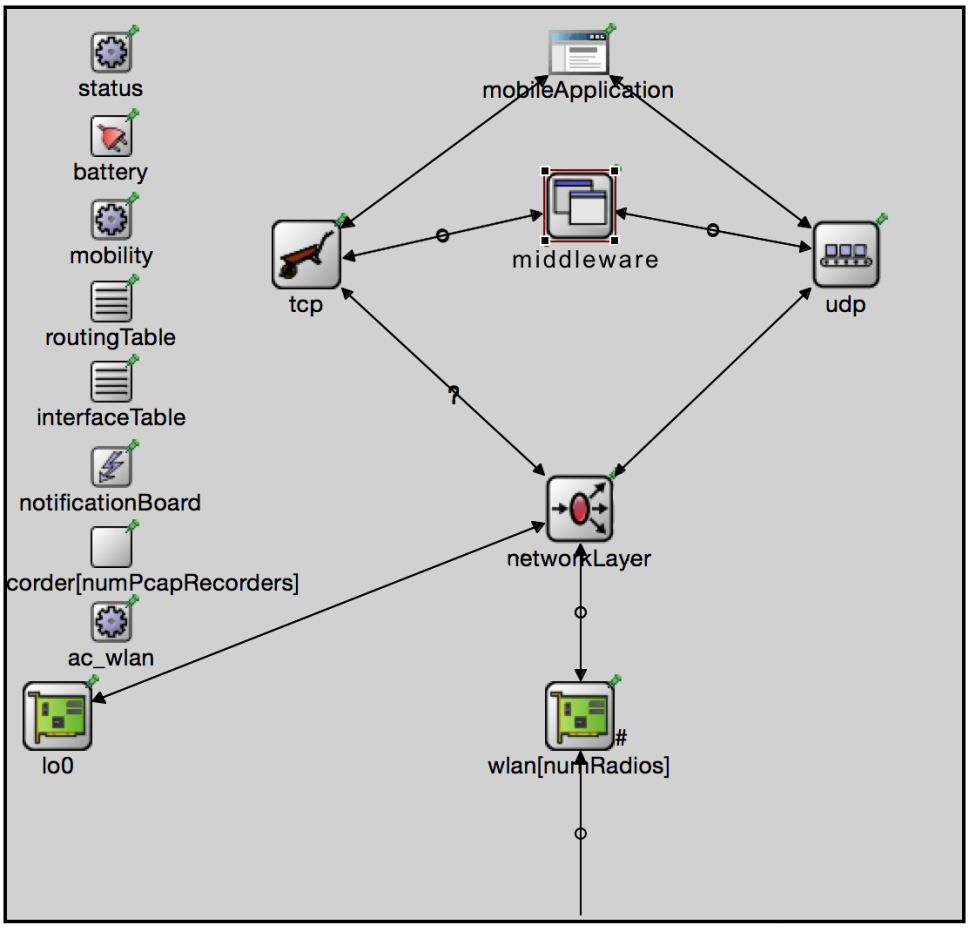


Figura 4.10: Imagem referente ao módulo responsável pela simulação de um dispositivo. Imagem extraída do simulador Omnet++.

Na figura 4.11 (A) é possível visualizar o cenário onde os dispositivos se deslocam representado pelo simulador Omnet++, assim como visualizar os submódulos que compõem o cenário da simulação. O módulo *channelControl* é responsável por gerenciar os canais para comunicação através de redes sem fio. O módulo *manager* é o responsável pela integração do simulador Omnet com o simulador SUMO. Este módulo se comunica com o simulador SUMO e acompanha o deslocamento dos veículos e dos pedestres na simulação do simulador e adiciona e remove dispositivos da simulação do Omnet, conforme esses eventos ocorrem no simulador de mobilidade. O módulo *statisticsCollector* é o responsável por coletar estatísticas da simulação e salvá-los em arquivos. Em B) é possível visualizar o simulador SUMO tratando a mobilidade de veículos, representados por triângulos amarelos na imagem. Os veículos exibidos em A) são tratados pelo SUMO em B), e os simuladores se comunicam através do protocolo TraCI implementado atavés do módulo *manager*. Na figura 4.11 nota-se que a parte B) é área de A) selecionada por um retângulo.

Módulo	Descrição
Middleware	Módulo que representa o *Middleware* na simulação. Responsável pela implementação dos protocolos de descoberta e todas as funcionalidades descritas.
mobileApplication	Aplicação móvel que implementa os serviços *FileTransfer* e *Information-Exchanger* e se conecta ao *Middleware* para obtenção dos dispositivos e serviços encontrados para consumo.
status	Representa o status do dispositivo, como por exemplo se ele está ligado, ou operacional, entre outros.
battery	Representa informações sobre a bateria que o dispositivo possui. Armazena a carga, vida útil entre outros.
ac_wlan	Módulo responsável pela configuração dos parâmetros da interface wlan (wireless), como por exemplo o canal de transmissão, a potência da antena, entre outros.
mobility	Módulo responsável pela mobilidade do dispositivo.
NotificationBoard	Este módulo recebe as notificações que o dispositivo recebe de outros dispositivos e encaminha para os submódulos correspondentes.
routingTable	Possui a tabela de roteamento das mensagens.
interfaceTable	Possui uma tabela com todas as interfaces que o dispositivo possui.
order	Módulo responsável por armazenar e ordenar os quadros recebidos através das interfaces.
lo0	Representa a interface de loopback.
wlan	Representa a interface wireless.
tcp	Módulo que representa o protocolo orientado a conexões TCP da camada de transporte.
udp	Módulo que representa o protocolo orientado a datagramas UDP da camada de transporte.
Network Layer	Módulo que representa a camada de rede, responsável pelo encaminhamento dos pacotes recebidos pelas camadas físicas para as camadas superiores.

Tabela 4.2: Descrição dos submódulos presentes em um dispositivo na simulação.

Na figura 4.12 é possível observar a organização criada para armazenar as classes e os métodos utilizados para implementação das simulações. Esta estrutura de grupos foi criada para facilitar o desenvolvimento de todos os aspectos das simulações apresentadas neste trabalho. Na imagem 4.12, cada grupo existente é representado pela imagem de uma pasta seguida do nome do grupo. Dentre os grupos que podem ser visualizados na imagem, o grupo **app** engloba códigos responsáveis pelo comportamento da aplicação móvel que se comunica com o *Middleware*, assim como código dos serviços implementados. O grupo **communication** engloba códigos implementados para comunicação, como por exemplo, mensagens HTTP, mensagens TCP, entre outros. O grupo **devices** abrange códigos implementados para tratar os dispositivos que executam o *Middleware*, como, por exemplo, os dispositivos móveis e os dispositivos estacionários. O grupo **dns** trata códigos que implementam classes que auxiliam no funcionamento

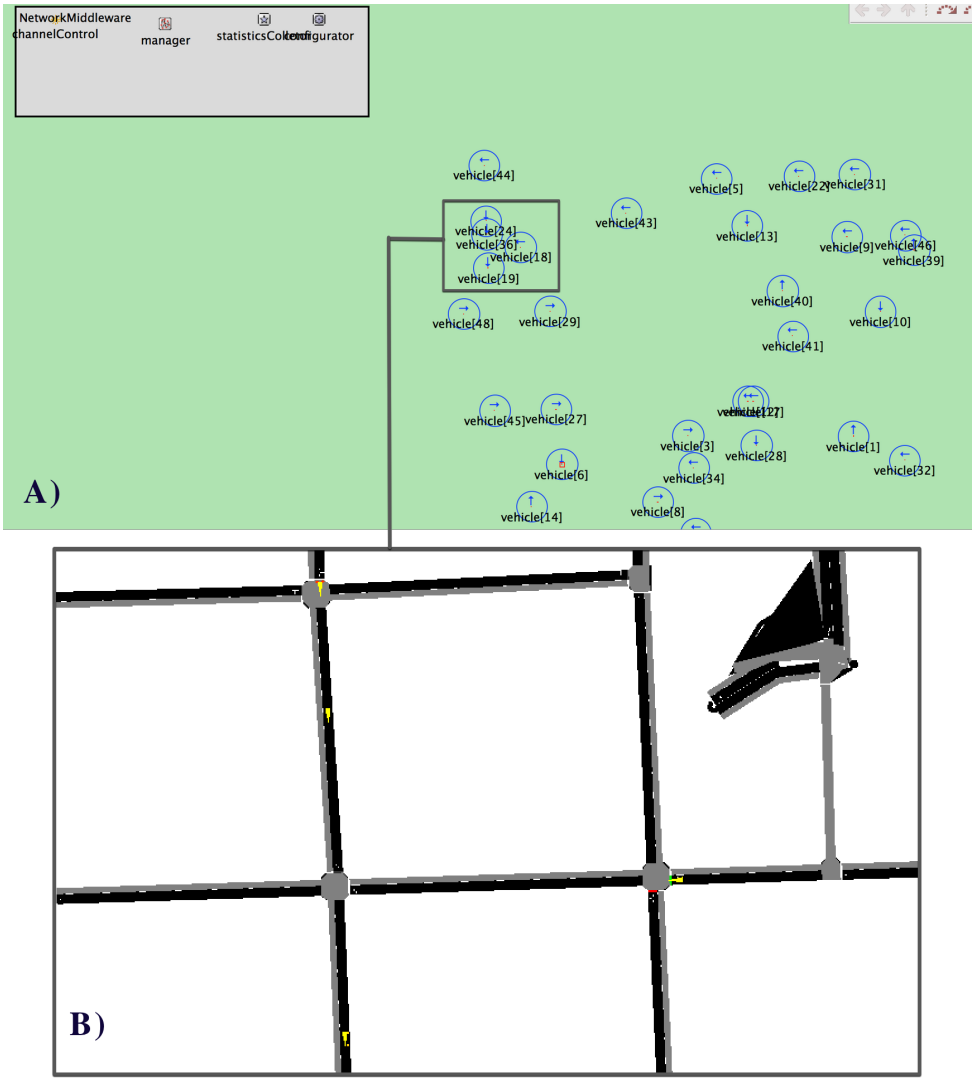


Figura 4.11: Exibição dos simuladores utilizados. Na parte A) é visível a interface do simulador OMNET tratando dispositivos presentes na simulação. Na parte B) é possível visualizar a interface exibida pelo simulador SUMO na qual vê-se alguns dispositivos no mapa. Os mesmos dispositivos estão presentes nos dois simuladores simultaneamente.

do Bonjour (mDNS), como por exemplo registros DNS, servidores DNS, entre outros. O grupo **interfaces** é responsável por algumas interfaces utilizadas por outras classes. O grupo **json** contém bibliotecas para serialização e desserialização de objetos seguindo o padrão JSon. O grupo **message** contém implementações de mensagens que são utilizadas pelo simulador para funções internas do simulador, como por exemplo, *callbacks*, *timers*, entre outros. O grupo **middleware** trata da implementação do *Middleware* e suas funcionalidades. O grupo **mobility** está relacionado com implementações de classes responsáveis por controlar a mobilidade dos dispositivos. O grupo **scenarioManager** é relacionado com implementações do protocolo TraCi para comunicação com o SUMO, representado pelo módulo *manager* na figura 4.11. O grupo **socket** trata de implementações de classes que auxiliam na comunicação utilizando soc-

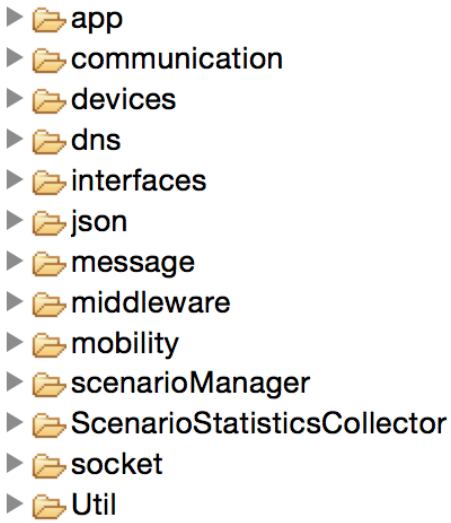


Figura 4.12: Ilustração da estruta de pastas contendo os códigos implementados para execução da simulação.

kets. Por fim, o grupo **Util** engloba classes que disponibilizam funções utilitárias para facilitar o trabalho de outras classes.

Nas simulações executadas, todo dispositivo está equipado com uma implementação do *Middleware* descrito. Todo dispositivo possui também uma aplicação móvel que oferece uma implementação dos serviços descritos, tanto para oferta quanto para consumo de serviços de outros dispositivos.

No simulador Omnet++, o tratamento de mensagens recebidas por um nó se dá utilizando *callbacks* no código. Quando um nó envia uma mensagem para outro nó, o pacote enviado é transmitido para todos os nós que estiverem dentro do alcance de comunicação da interface de comunicação sem fio. Todo nó que estiver ao alcance de uma transmissão receberá os pacotes de dados recebido em uma fila própria. Processando a fila de eventos, cada dispositivo trata a lista de pacotes registrados, disparando assim a *callback* que foi registrada anteriormente.

A aplicação móvel na simulação utiliza *sockets* compatíveis com o simulador, e utiliza tanto o protocolo UDP quanto o protocolo TCP para transporte de pacotes entre os nós. Os protocolos da camada de transporte implementados pelo simulador operam de maneira semelhante aos protocolos reais.

Cada simulação está relacionada a um cenário que é descrito em um documento XML. SUMO interpreta este documento e tem conhecimento de quais dispositivos devem estar presente na simulação. Cada dispositivo tem um tempo para ser inserido na simulação. Quando é o momento de inserir um dispositivo na simulação, SUMO se comunica com Omnet++, para que este realize os procedimentos necessários para criação de um dispositivo. Periodicamente

o Omnet++ se comunica com o simulador SUMO e utilizando o protocolo TRaCI consulta quais dispositivos estão presentes na simulação de mobilidade e suas posições. Quando um dispositivo chega ao seu destino, o SUMO envia uma mensagem para o simulador Omnet++ informando este evento e Omnet++ realiza o procedimento para remoção do dispositivo simulado.

O simulador possui métodos específicos pra tratar a criação de um novo dispositivo. Na inserção de um novo dispositivo no simulador, é criado um objeto que representa este dispositivo e o simulador mantém uma lista com todos os dispositivos simulados. No dispositivo implementado na simulação, o primeiro objeto a ser criado é o *Middleware*, que trata dos mecanismos de comunicação com outros dispositivos. Esse procedimento envolve estabelecer *sockets* UDP e TCP para comunicação através de protocolos de descoberta com outros dispositivos. Além disso, é instanciado o módulo Web do *Middleware* que permite que aplicações requisitem funcionalidades através da API definida. Posteriormente, é criada a aplicação móvel do dispositivo, responsável por implementar serviços que utilizam o *Middleware*. A aplicação móvel, por sua vez, cria *sockets* TCP para realização de chamadas HTTP destinadas ao *Middleware*. Nestas requisições, a aplicação consulta o *Middleware* em busca de serviços desejados e, caso não haja nenhum serviço conhecido, é criada uma demanda no *Middleware* informando os serviços desejados. Em conjunto com a criação de *sockets* para requisição, também são instanciados *sockets* para tratar requisições que desejam consumir serviços oferecidos por outros nós. Também através deste *socket* a aplicação disponibiliza uma URL para recebimento de *callbacks* originadas do *Middleware*.

Quando a aplicação móvel recebe uma notificação, é processado primeiramente o contexto ao qual a requisição é destinada, e logo depois é verificado qual o método HTTP que a requisição possui. Baseando-se nesses atributos da requisição, o container Web da aplicação móvel trata a requisição de acordo com o solicitado. Dependendo da requisição, notificações podem ser recebidas, serviços podem ser oferecidos ou consumidos. Posteriormente ao processamento da requisição, é elaborada uma resposta compatível com o protocolo HTTP, que é enviada ao dispositivo que enviou a requisição.

O *Middleware* que está presente em todos os dispositivos possui rotinas programadas para realizar o procedimento de descoberta de dispositivos e serviços. De tempos em tempos, cabe a ele enviar mensagens de descoberta para os dispositivos que estiverem ao alcance. O *Middleware* também possui rotinas para tratar um novo dispositivo descoberto e verificar se este dispositivo possui serviços, e em caso afirmativo, verificar se possui algum serviço compatível com alguma demanda local. Além de realizar estar verificações, o *Middleware* mantém regis-

trados os dispositivos e serviços conhecidos. O *Middleware* também implementa rotinas para manter o monitoramento de quais dispositivos estão em contato. A verificação se um serviço é compatível com alguma demanda é feita comparando-se os atributos Mime-type e ACTION dos serviços buscados com cada serviço descoberto.

Durante a simulação, Omnet++ monitora todos os dados configurados e registra em arquivos de log os parâmetros previamente configurados. Estes logs são posteriormente interpretados e os dados resultantes são usados nas análises deste trabalho.

4.5 Resultados

Os primeiros testes realizados com o simulador procuraram identificar se o modelo de comunicação oportunística sendo considerado é viável em um cenário urbano de pedestres e veículos. Para tanto, buscou-se identificar intervalos apropriados para as operações de descoberta, maximizando o número de dispositivos encontrados sem onerar de forma desnecessária a carga da bateria dos dispositivos e sem que essas operações impedissem trocas de informações efetivas das aplicações.

As simulações foram planejadas com diferentes quantidades de veículos e pessoas no cenário alvo, porém, por restrições de tempo, nem todos os cenários foram simulados. Também foram utilizados diferentes protocolos de descoberta de dispositivos e serviços implementados no *Middleware*, incluindo UPnP, Bonjour e SLP. Com os resultados obtidos, foi constatado que o uso de um protocolo de descoberta específico não acarretou alterações nos resultados das simulações. As taxas de descobertas para os diferentes protocolos podem ser vistas na figura 4.13. Simulações utilizando os mesmos parâmetros e protocolos de descoberta diferentes produziram resultados semelhantes, como se vê pelas linhas sobrepostas que representam as descobertas de dispositivos com cada protocolo. Este fato indica que qualquer um dos protocolos utilizados permite a descoberta de dispositivos de maneira equivalente, já que o procedimento de troca de mensagens para descoberta através de qualquer protocolo é inferior a 100 milissegundos. O fator que impacta as descobertas na simulação é o intervalo no qual são realizadas tentativas de descoberta de dispositivos, e este aspecto é analisado e avaliado neste trabalho.

Nas simulações, o *Middleware* se mostrou funcional e permitiu que dispositivos e serviços fossem localizados. Porém, para funcionamento do *Middleware* é importante definir um intervalo adequado para o envio de mensagens de descoberta/anúncio de dispositivos e serviços. Tentando determinar um intervalo de envio de mensagens de descoberta adequado para ser utilizado pelo *Middleware*, que permitisse a descoberta eficiente de dispositivos e não ge-

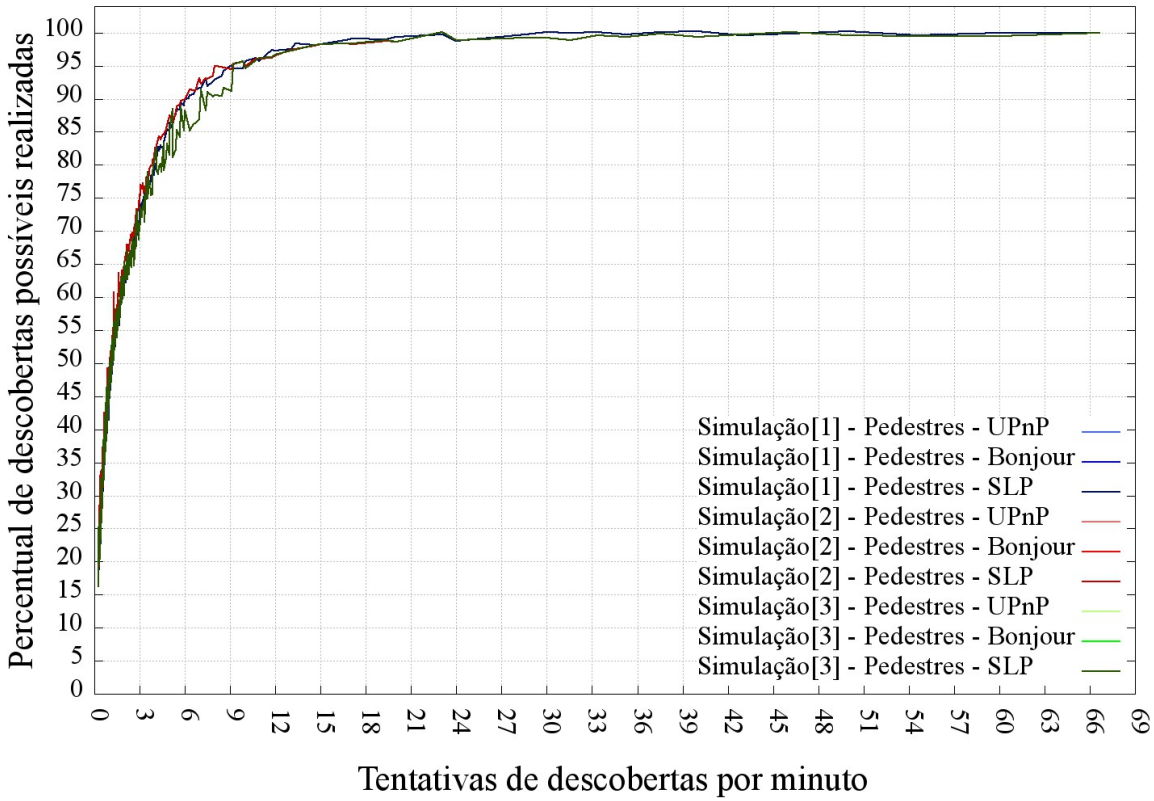


Figura 4.13: Taxas de descobertas para diferentes protocolos de descoberta.

rasse impacto significativo no gasto energético, diferentes simulações foram realizadas, com diferentes frequências de envio de mensagens.

Na figura 4.14 é possível ver um gráfico exibindo o impacto da frequência de envio de mensagens no total de descobertas realizadas **pelos dispositivos**. No eixo X do gráfico é exibido o número de tentativas de descobertas que foram realizadas por minuto, enquanto no eixo Y é possível observar a porcentagem do total de descobertas possíveis de fato realizadas. É possível notar que, com requisições de descobertas enviadas entre intervalos de tempo menores, maiores taxas de descobertas de dispositivos são obtidas. Assim, seguindo-se o eixo X, vê-se , por exemplo, que com 60 requisições por minuto, 1 por segundo, todas as possibilidades de descobertas foram efetivas. Isso significa que sempre que 2 dispositivos estiveram no raio de alcance um do outro, o software de descoberta proporcionou suas identificações mútuas. Quando o intervalo de envio de mensagens é pequeno contudo, além de aumentar o número de dispositivos descobertos, os dispositivos enviam mais requisições para descobertas, o que consequentemente aumenta a utilização do dispositivo de comunicação sem fio e o gasto energético.

Neste gráfico são exibidos os resultados de 3 simulações, mostradas em 3 curvas que representam as taxas de descobertas de pedestres, e em 3 curvas que representam as de veículos, conforme lê-se na legenda no canto inferior direito.

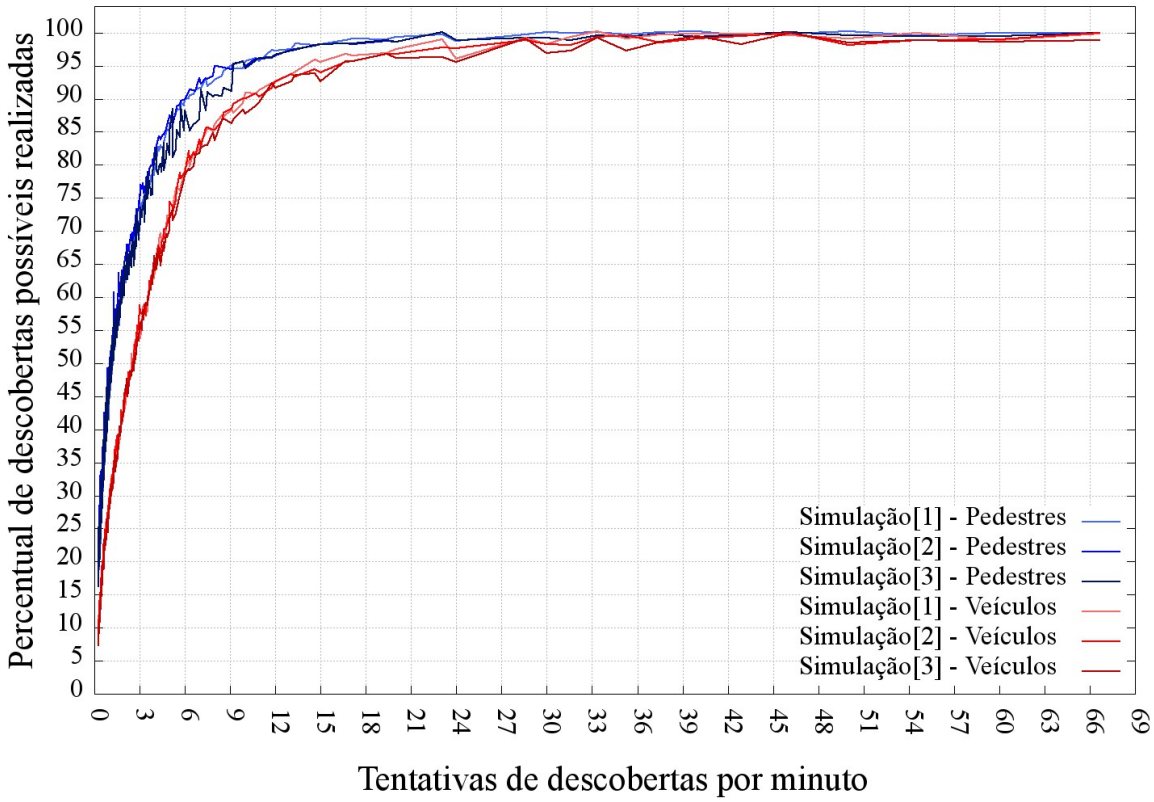


Figura 4.14: Impacto da frequência de envio de mensagens de descoberta na efetiva descoberta de outros dispositivos.

Como pode ser visto na figura 4.14, os veículos precisam enviar mais mensagens por minuto do que os pedestres para terem a mesma taxa de descoberta. Isto é explicado pelo fato de veículos se deslocarem mais velozmente do que pedestres. Quando um dispositivo se desloca com maior velocidade, este precisa enviar mensagens de descoberta com mais frequência pois o tempo de contato com outros dispositivos é menor.

Na imagem 4.14 é possível ver que pedestres podem descobrir 90 % dos dispositivos existentes no ambiente se enviarem 6 mensagens de descoberta por minuto, enquanto para ter a mesma taxa de descoberta, veículos precisam enviar 12 mensagens por minuto. Esta proporção continua se olharmos a quantidade de mensagens que devem ser enviadas para alcançar 95 % de descoberta, pedestres enviam 9 mensagens por minuto, enquanto veículos enviam 18 mensagens por minuto. Também é possível notar que a partir de 33 mensagens de descoberta por minuto a taxa de descoberta tanto de pedestres quanto veículos alcança aproximadamente a totalidade de dispositivos possíveis de serem descobertos. A partir desta informação conclui-se que enviar 30 mensagens por minuto (1 mensagem a cada 2 segundos aproximadamente) é suficiente para descobrir aproximadamente todos os dispositivos e vê-se também que utilizar um intervalo de envio menor que 2 segundos não proporciona ganhos significativos na descoberta de dispositivos.

Nos testes, o valor de referência para as descobertas é determinado analisando a proximidade entre os dispositivos, de forma que dispositivos que se encontraram no alcance de outros dispositivos durante a simulação são considerados dispositivos que poderiam ser descobertos. Durante a simulação, é normal que alguns dispositivos nunca se encontrem, pois dependendo de suas rotas eles nunca estarão dentro do alcance de comunicação de outros dispositivos específicos.

Na figura 4.15 é exibido um gráfico no qual são mostradas as taxas de descobertas **de serviços** nas simulações executadas. São exibidas 6 curvas, das quais 3 são relacionadas com pedestres e 3 com veículos. Cada curva exibe o percentual de serviços descobertos em relação ao total de serviços passíveis de descoberta.

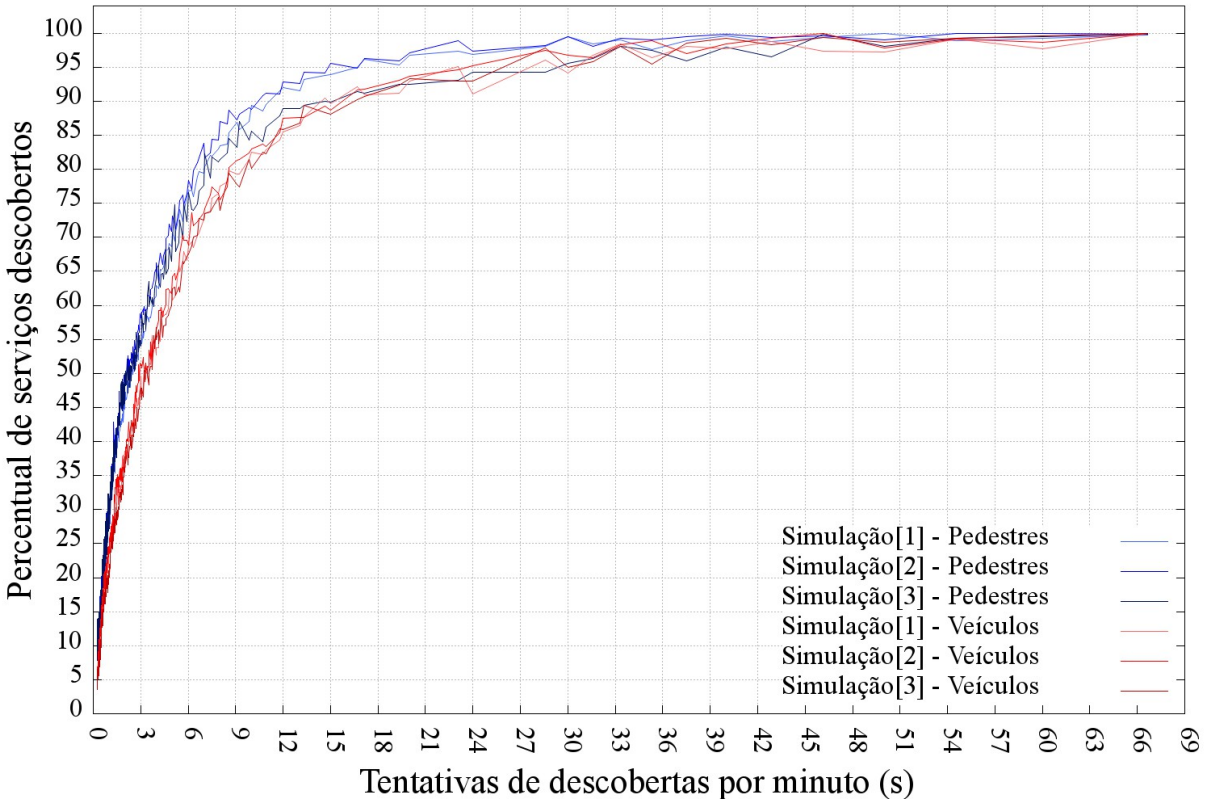


Figura 4.15: Taxas de descoberta de serviços presentes na simulação (eixo Y) de acordo com o intervalo de envio de mensagens de descoberta por minuto (eixo X).

Comparando as taxas de descoberta **de dispositivos** (figura 4.14) e as taxas de descoberta **de serviços** (figura 4.15) é possível notar que as curvas entre serviços e dispositivos possuem os mesmos comportamentos, conforme figura 4.16. A proporção de dispositivos e serviços descobertos é aproximadamente a mesma e isso é justificado pelo fato de que quando um dispositivo descobre outro dispositivo, descobre também os serviços deste. Contudo, é notável na figura 4.16 que a quantidade de serviços descobertos é inferior à quantidade de dispositivos. Essa diferença é justificada pelo fato de que nem sempre quando dois dispositivos se encontram o

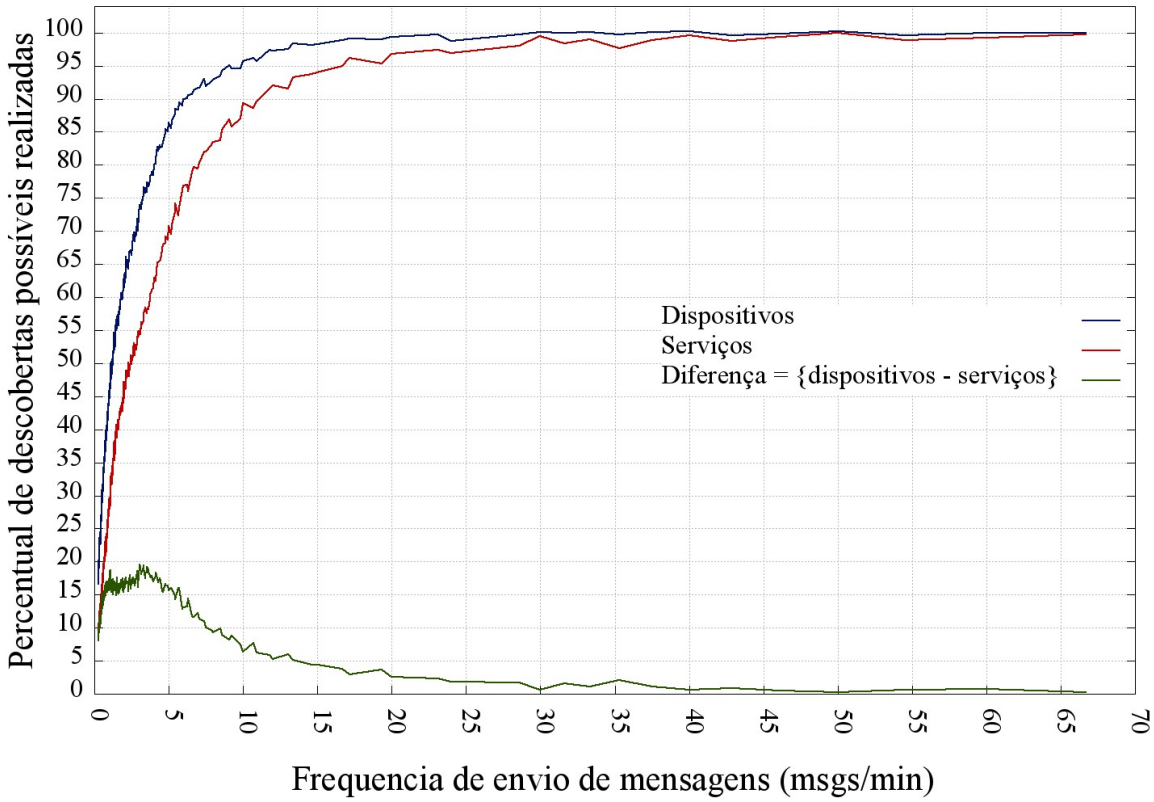


Figura 4.16: Comparação entre descoberta de dispositivos e serviços.

tempo disponível para interação é suficiente para descoberta do dispositivo e posteriormente descoberta dos serviços do dispositivo.

Com os resultados obtidos na simulação é notável que todos os serviços descobertos foram sempre consumidos com sucesso. Esta conclusão é apoiada pelos números de serviços **descobertos** e os números de serviços **consumidos**. Estes números são sempre iguais, o que indica que todo serviço descoberto sempre foi consumido com sucesso pelo dispositivo consumidor. Isto é possível compreender se analisarmos o tempo necessário para ser realizado o consumo de um serviço na simulação, que é a da ordem de décimos de segundos. O consumo de serviços é rápido pois depende apenas do envio e recebimento de algumas requisições HTTP entre dispositivos que estão próximos e se comunicam diretamente.

4.5.1 Tempo de contato entre dispositivos

As simulações realizadas procuraram refletir as oportunidades de encontro entre dispositivos de acordo com modelos reais de mobilidade. Nessas avaliações, o tipo de encontro que ocorreu com mais frequência foi quando os dispositivos se deslocam em direções opostas, o que é ilustrado na figura 4.17. Esses dados foram obtidos pelo simulador considerando a distância máxima para contato de 50 metros, que é o alcance máximo da transmissões via interface de

comunicação sem fio utilizada pelos dispositivos simulados.

Analisando também a movimentação dos nós nas simulações, os veículos tiveram uma velocidade média de 40km/h durante seus deslocamentos, e os pedestres tiveram uma velocidade média de 4km/h. Dessa forma, é estimado que o pior caso de encontro no cenário avaliado, ilustrado na figura 4.17, quando estes se movem a uma velocidade constante de 40km/h para veículos, em sentidos opostos tenha a duração de 4,5 segundos. Já o tempo de encontro estimado para pedestres que se deslocam em sentidos opostos é de 45 segundos.

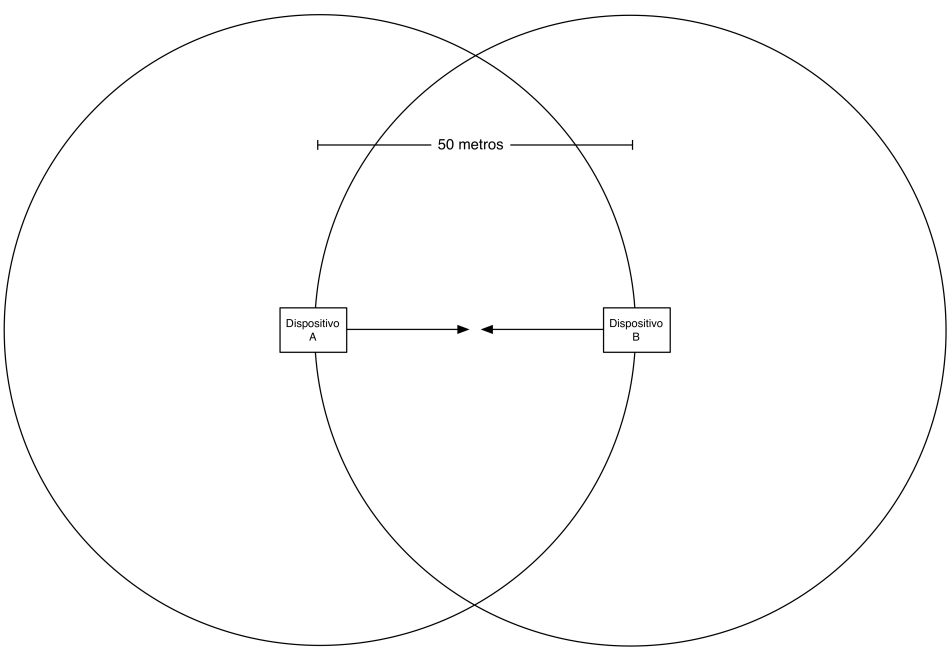


Figura 4.17: Estimativa de encontro de dispositivos.

Como resultado, é esperado que os contatos entre os dispositivos tenham duração superior aos valores estimados. Porém, é sabido que podem ocorrer encontros nos quais os dispositivos não estão sob os mesmos valores no eixo de coordenadas y, o que implica em tempos de contato ainda menores que o pior caso comum. O esquema da figura 4.18 mostra o pior caso comum pois, devido ao cenário de simulação ser composto de quadras e ruas, os dispositivos frequentemente se encontram quando cruzando um ao lado do outro já que o tamanho de uma quadra impede a comunicação entre dispositivos em quadras vizinhas.

Na figura 4.18 a distância x é resultado da equação 4.1.

$$x = 50 \times cos(\theta) \tag{4.1}$$

Como cos(0°) possui valor 1, o melhor caso seria x valendo 50 metros e como cos(90°) possui valor 0, o pior caso seria x valendo 0.

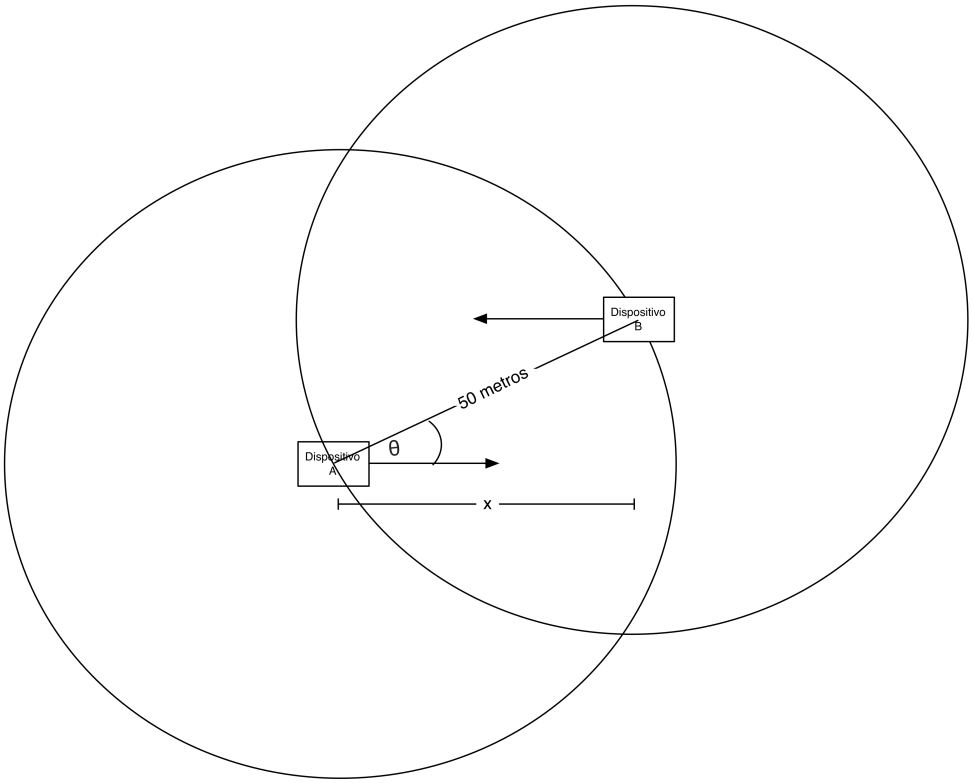


Figura 4.18: Pior caso de encontro entre os dispositivos.

Nas simulações realizadas foram coletados os tempos de contato entre os dispositivos. Esses tempos de contato foram calculados através da observação do recebimento de mensagens de descoberta, de forma que enquanto um dispositivo recebe mensagens de um outro dispositivo, o dispositivo receptor tem conhecimento da proximidade do outro dispositivo. A partir do momento que não se recebe mais nenhuma mensagem de descoberta, é notável que o dispositivo não é mais comunicável e é contabilizado o tempo que se esteve em contato com o outro dispositivo.

Como apresentado na figura 4.19, com dados obtidos das simulações executadas, é possível observar que os tempos de contatos estão relacionados diretamente com a mobilidade dos nós.

Na figura 4.19, no eixo 'x' é exibido o tempo de duração dos contatos e no eixo 'y' é exibido um valor que representa os valores acumulados de contatos que ocorreram entre os dispositivos. As curvas de contatos entre os dispositivos exibem os tempos em que os dispositivos estiveram próximos e foi possível que eles se comunicassem. Analisando os tempos que esses dispositivos estiveram próximos vê-se quanto tempo estes dispositivos possuem para comunicação em relação às suas mobilidades.

Nas simulações, o tempo de contato entre veículos é curto, a maioria dos contatos tem

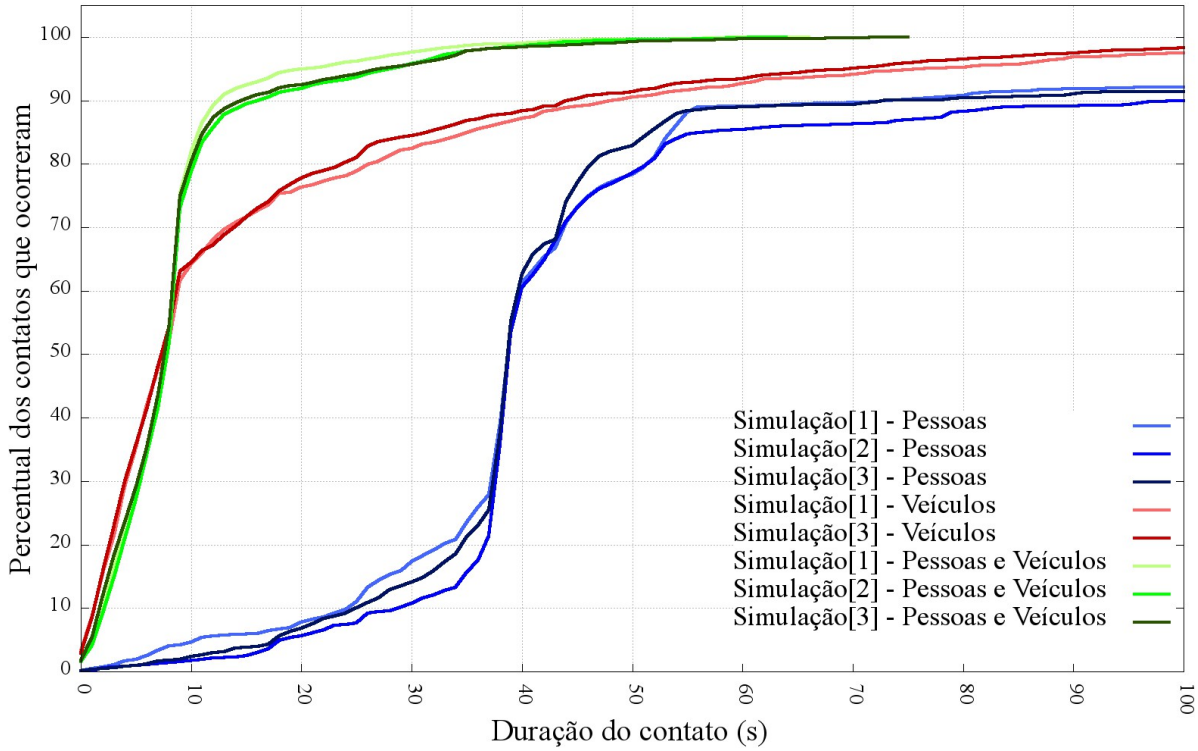


Figura 4.19: Tempos de contatos dos dispositivos exibidos de maneira cumulativa.

duração de até 20 segundos. Os contatos que envolvem veículos e pedestres tem predominantemente duração de até 20 segundos. Já os tempos de contatos dos pedestres em sua grande maioria ocorre entre 30 e 40 segundos. Também pode ser observado que existem contatos, embora em pequena quantidade, que duram vários minutos, quando seguem caminhos comuns.

Na figura 4.20 é possível ver com mais detalhes os contatos mais longos que envolvem pedestres. Enquanto pedestres têm uma maior frequência de contato, entre 30 e 50 segundos, para os veículos a maioria dos contatos se concentra entre intervalos maiores do que 0 segundos e menores do que 12 segundos. Os contatos entre pedestres e veículos ocorrem predominantemente em até 10 segundos; porém, diferentemente dos contatos apenas entre veículos, esses contatos continuam ocorrendo até 20 segundos. É visível também que o número de contatos entre pedestres e veículos é maior que o número de contato entre veículos apenas. Este fato é justificado pela maior velocidade de deslocamento dos veículos, que rapidamente entram e saem do raio de alcance de comunicação dos dispositivos de pedestres.

Na figura 4.21 o eixo 'y' apresenta a porcentagem do número total de contatos que ocorreram.

Nesta imagem, fica claro que os tempos de contatos envolvendo veículos são maiores do que os que envolvem pedestres e veículos. Além disto, quanto tratam-se de contatos entre pedestres e veículos, muitos ocorrem com duração de até 10 segundos, havendo contatos que duram até

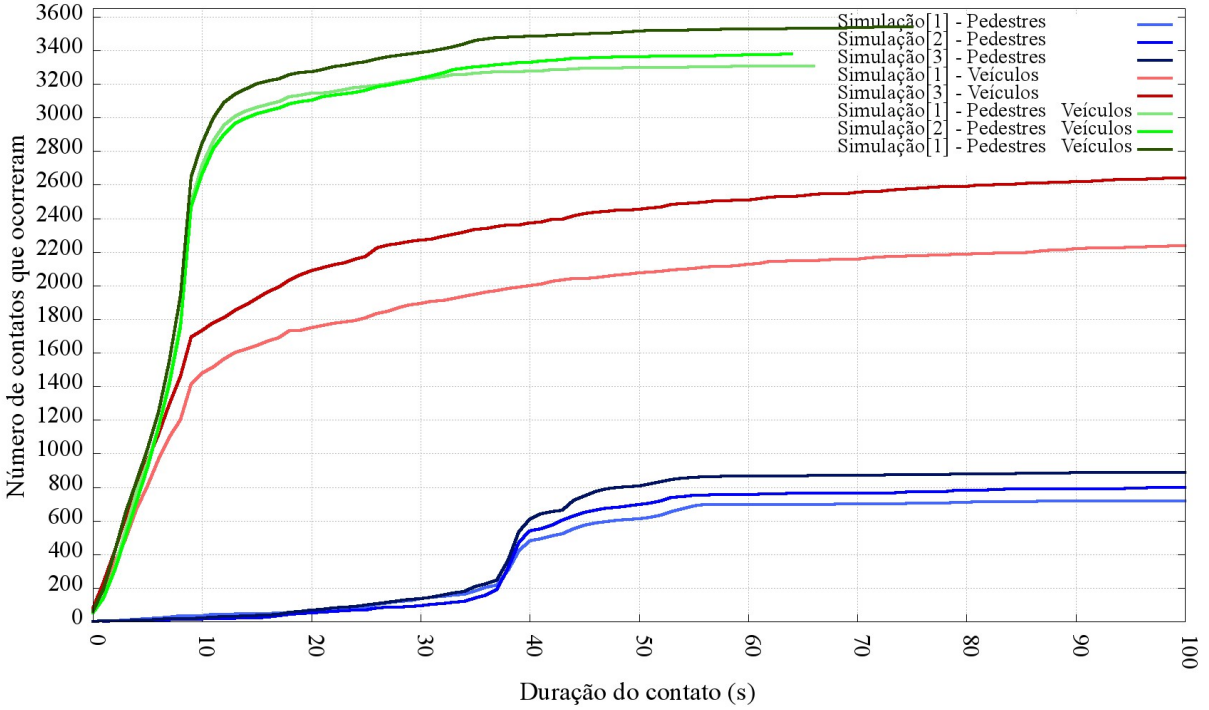


Figura 4.20: Tempos de contatos dos dispositivos exibido de maneira acumulativa, exibindo as interações que duraram até 100 segundos.

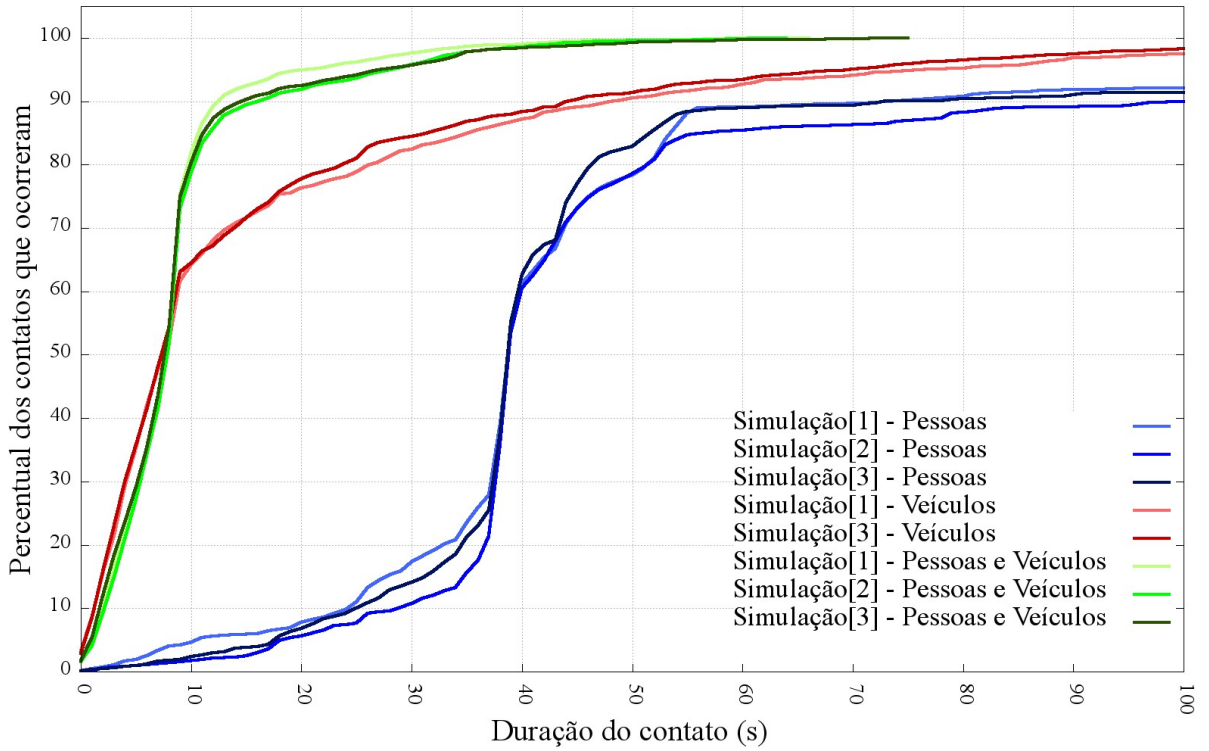


Figura 4.21: Estudo dos tempos de contatos. Os tempos de contatos são exibidos através do percentual do total de contatos que ocorreram e são exibidos de acordo com uma CDF (Cumulative Distribution Function).

30 segundos, embora em uma proporção menor se comparado até 10 segundos.

Além de identificar a predominância das durações dos contatos envolvendo os dispositivos é possível notar que não existem contatos entre veículos e pedestres superiores a 80 segundos, sendo este então o tempo máximo para comunicação entre veículos e pedestres. Já entre veículos, não existem contatos superiores a 260 segundos (4 minutos e 20 segundos), enquanto que entre pedestres existem contatos, embora sejam poucos, de até 2.500 segundos (41 minutos e 40 segundos).

Conhecendo as características das durações de contato entre dispositivos de pedestres e de veículos, é possível estimar frequências apropriadas para as operações de descoberta. Como resultado das simulações, vê-se que, para dispositivos de pedestres, aproximadamente 70% dos encontros têm duração entre 20 e 50 segundos, sendo que cerca de 50% duram cerca de 37 a 42 segundos. Ou seja, a pequena mobilidade de pedestres faz com que seus dispositivos permaneçam próximos de outros, em geral, por cerca de 20 segundos ou mais.

Para dispositivos associados a veículos, os tempos de proximidade são menores, com cerca dd 60% deles menores que 10 segundos. Por extensão, encontros entre dispositivos de veículos e de pedestres também têm curta duração, com cerca de 60% com valores de 5 a 10 segundos. 30% desses encontros, contudo, têm duração de até 5 segundos, o que pode inviabilizar comunicações oportunísticas.

4.5.2 Avaliação de funcionalidade do *Middleware*

Para verificação do funcionamento do *Middleware*, foram propostos alguns casos de testes de aplicações que podem ser desenvolvidas utilizando suas funcionalidades. Estas aplicações são descritas em detalhes com aspectos relevantes para justificar seus funcionamentos e demonstrar como interagem com o *Middleware*.

1. Aplicação 1: Aplicação móvel de compartilhamento de arquivos. Aplicação se registra no Middleware com o nome *file-sharer* e informa o Mime-type=MIME_FILE e as ACTION's=ACTION_STORE, ACTION_RETRIEVE. Oferece o serviço de armazenamento de arquivos no contexto */store* e o serviço de obtenção de arquivos no contexto */retrieve*. Utiliza a porta 8080 para implementação do container Web.

2. Aplicação 2: Aplicação de impressão de arquivos. Aplicação se registra no *Middleware* com o nome de *printer-service* e informa o Mime-type= MIME_FILE e a ACTION= ACTION_PRINT. Oferece o serviço de impressão no contexto */print* e implementa o

container Web na porta 8082.

3. Aplicação 3: Aplicação móvel de compartilhamento de informações. Aplicação se registra no *Middleware* com o nome de *information-exchanger* e informa os Mime-types= MIME_PLAIN_TEXT, MIME_JSON e a ACTION=ACTION_EXCHANGE. Oferece a obtenção de informações no contexto */information* e implementa o container Web na porta 8083.

4. Aplicação 4: Aplicação móvel para prestação de serviço de táxi com definição automática de itinerário. Aplicação se registra no *Middleware* com o nome *smart-cab* e informa os Mime-types=MIME_PLAIN_TEXT, MIME_JSON e a ACTION= ACTION_EXCHANGE. A aplicação troca informações relacionadas com a identidade do usuário e o destino da corrida utilizando o contexto */app* e implementa o container Web na porta 8084. Neste caso, observa-se que a ação definida não permite, por si só, realizar a seleção de serviços disponíveis. Uma solução para este caso poderia ser repassar todos os dados disponíveis sobre serviços com essa definição de tipo e ação, ou seja, os metadados, deixando à aplicação realizar uma busca mais fina. Outra opção seria definir, tanto na aplicação de quem busca o serviço quanto de quem o provê, uma ação mais específica, como ACTION_TAXI_DRIVE_TO. Neste caso, a seleção do serviço poderia ser realizada pelo próprio middleware.

5. Aplicação 5: Aplicação móvel para identificação de usuários. A aplicação se registra no Middleware utilizando o nome *user-identification* e informa os Mime-types=MIME_IMAGE MIME_PLAIN_TEXT, MIME_JSON e a ACTION=ACTION_EXCHANGE. A aplicação recebe de outros dispositivos uma consulta textual sobre o perfil do usuário detectado. A identificação do usuário pode ser através do seu nome, de um ID, ou de outra informação relacionada com o usuário. O serviço também pode receber arquivos de imagem que serão processadas e tentarão identificar o usuário e retornar as informações relevantes. Este serviço é oferecido através do contexto */identify* e implementa o container Web utilizando a porta 8085. Novamente, a possibilidade de uso de uma ação mais específica poderia prover a identificação inequívoca do serviço buscado.

No funcionamento das aplicações consideradas, todos os serviços descritos devem inicialmente se registrar no *Middleware* local, para que ele tenha conhecimento de suas ofertas e possa anunciá-los para dispositivos próximos. Estes serviços também podem requisitar do *Middleware* informações sobre quais dispositivos estão próximos e quais serviços oferecem, prosseguindo posteriormente para a identificação e o eventual consumo daqueles desejados.

Todas estas aplicações podem estar presentes simultaneamente em um dispositivo qualquer equipado com o *Middleware*. O *Middleware* realizará seus procedimentos e oferecerá as funcionalidades previstas para todos os serviços do dispositivo.

Considerando a viabilidade da seleção de serviços através de seus parâmetros Mime-type e ACTION, vê-se que as aplicações ilustradas poderiam funcionar desta maneira. Tendo como referência o modelo centrado em conteúdo, especificado pelo campo tipo, percebe-se que toda interação envolve algum tipo de transferência de um objeto do tipo especificado. Desta forma, o campo ação pode representar não apenas um sentido de transferência, mas um nome de serviço, ou qualquer outro parâmetro conhecido pelos nós que interagem, que permita a seleção de um provedor para uma demanda existente.

O *Middleware* descobre outros dispositivos próximos através dos protocolos de descoberta utilizados, podendo ter comportamentos diferentes dependendo do protocolo utilizado. O protocolo UPnP, por exemplo, define o comportamento dos nós de forma que o provedor do serviço deve agir ativamente enviando mensagens sobre sua presença, enquanto o nó cliente deve agir passivamente e aguardar mensagens. Já o protocolo Bonjour possui um comportamento inverso ao UPnP, pois o nó provedor não envia mensagens informando sua presença e quem tem o papel de requisitar quais dispositivos e serviços estão próximos é o cliente. No protocolo SLP, tanto o nó provedor quanto o nó cliente podem enviar mensagem informando/requisitando serviços. Na tabela 4.3 são apresentados os comportamentos dos protocolos de descoberta utilizados.

Protocolo	**Provedor**	**Cliente**
UPnP	Ativo	Passivo
Bonjour	Passivo	Ativo
SLP	Ativo	Ativo

Tabela 4.3: Comportamentos dos protocolos utilizados.

Para que o *Middleware* possa usar os protocolos de descoberta é preciso definir como será o comportamento quando camadas superiores do *Middleware* requisitarem dispositivos e serviços. Dependendo do estado do dispositivo, o *Middleware* pode se comportar ativa ou passivamente nos papéis de provedor e cliente.

Na figura 4.22 é exibido o diagrama de estados do comportamento do *Middleware* quando oferecendo serviços passivamente. Neste diagrama, é visível que o *Middleware* inicialmente não realiza nenhuma ação quando recebe uma requisição em busca de serviços. Posteriormente, quando for informado que o dispositivo oferece algum serviço (utilizando a função *register()* da API) o *Middleware* adiciona o serviço indicado à lista de serviços oferecidos, altera seu estado e irá responder quando receber requisições em busca de serviços. O *Middleware* continuará neste

estado até que não existam mais serviços oferecidos localmente, cancelados através da função *unregister()* da API.

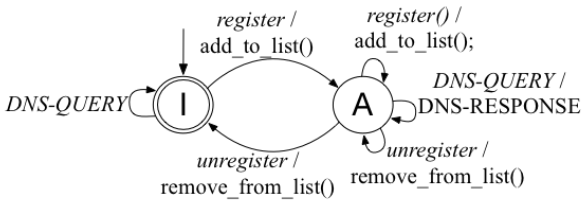


Figura 4.22: Diagrama de estados do provedor passivo.

Na tabela 4.5 são exibidos os estados, as ações e os novos estados em decorrência de eventos no diagrama da figura 4.22.

Estado atual	Evento	Ação	Novo estado
Iddle	*DNS-QUERY*[from_net]		Iddle
Iddle	*register()*[local_api]	add_to_list()	Answering
Answering	*DNS-QUERY*[from_net]	DNS-RESPONSE[to_net]	Answering
Answering	*register()*[local_api]	add_to_list()	Answering
Answering	*unregister()*[local_api]	remove_from_list()	Answering
Answering	*unregister()*[local_api]	remove_from_list()	Iddle

Tabela 4.4: Estados do provedor passivo.

Nos diagramas apresentados são exibidas as mensagens (PDU - Protocol Data Unit) dos protocolos utilizados pelo *Middleware*. O protocolo Bonjour, por exemplo, utiliza a mensagem *DNS-Query* para buscar por serviços, enquanto a mensagem *DNS-RESPONSE* é utilizada para responder uma requisição. Já o protocolo UPnP utiliza a mensagem *NOTIFY* para anunciar a oferta de serviços. Como no protocolo SLP podem existir tanto provedores como clientes ativos, o protocolo prevê as mensagens *SrvRqst* para requisição de serviços, *SrvRply* para resposta de requisições e a mensagem *SrvReg* para que provedores anunciem serviços oferecidos.

Protocolo	Oferta de serviços	Requisição de serviços	Resposta para requisições
Iddle	*DNS-QUERY*[from_net]		Iddle

Tabela 4.5: Estados do provedor passivo.

Quando o provedor de serviços é passivo é necessário que o cliente seja ativo em busca de dispositivos e serviços. Na figura 4.23 é exibido o diagrama de estados de comportamento do *Middleware* sendo cliente e agindo ativamente. Neste comportamento, o *Middleware* permanece em espera até ser informado que estão sendo buscados dispositivos e serviços através da função de busca *lookup()* provida pela API, causando a chamada da função *start_searching()*

que configura *timers* para periodicamente buscar por serviços. Quando em busca de serviços, periodicamente são enviadas mensagens em *broadcast* para outros dispositivos próximos. Quando o *Middleware* recebe alguma mensagem informando a oferta de serviços é verificado se algum dos serviços ofertados é um dos buscados, e caso seja, é enviada uma notificação de *callback* para a aplicação que requisitou determinado serviço. O *Middleware* continua neste estado até que seja chamada a função *stop_lookup()* fazendo com que o *Middleware* pare de buscar por serviços.

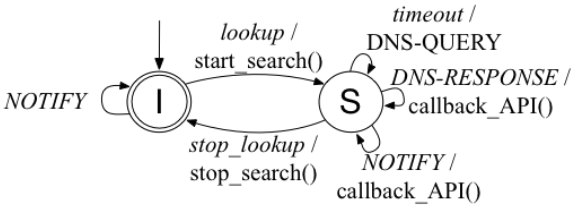


Figura 4.23: Diagrama de estados do cliente ativo.

Os eventos que causam alterações no diagrama presente na figura 4.23 são exibidos na tabela 4.6.

Estado atual	Evento	Ação	Novo estado
Iddle	*NOTIFY*[from_net]		Iddle
Iddle	*lookup*[local_api]	start_searching()	Searching
Searching	*timeout*	DNS-QUERY[to_net]	Searching
Searching	*DNS-RESPONSE*[from_net]	*callback_API()*	Searching
Searching	*NOTIFY*[from_net]	*callback_API()*	Searching
Searching	*stop_lookup*[local_api]	stop_searching()	Iddle

Tabela 4.6: Estados do cliente ativo.

Dependendo do protocolo utilizado na descoberta de serviços é possível que o *Middleware* quando ofertando serviços realize o envio de mensagens ativamente. Quando no modo ativo, o *Middleware* inicialmente está em um estado de espera, no qual não realiza nenhum envio de mensagens e também não responde nenhuma requisição por serviços. O *Middleware* apenas altera seu estado quando é informado do registro de um serviço sendo oferecido através da chamada *register()* da API local. Após essa chamada, o *Middleware* altera seu estado para o estado de oferta e invoca a função *start_announce()* que configura *timers* para periodicamente enviar ofertas de serviços em *broadcast*. Neste estado, o *Middleware* também responde requisições em busca de serviços geradas por outros dispositivos na rede. Enquanto neste estado, o *Middleware* pode receber outras chamadas *register()* informando mais serviços ofertados e o estado de oferta será mantido. Neste estado também é possível o recebimento de chamadas *unregister()* removendo serviços registrados anteriormente, e quando não existirem mais servi-

ços sendo ofertados, o *Middleware* retorna para o estado de espera Na figura 4.24 é exibido o diagrama de estados do *Middleware* quando sendo um provedor e agindo ativamente.

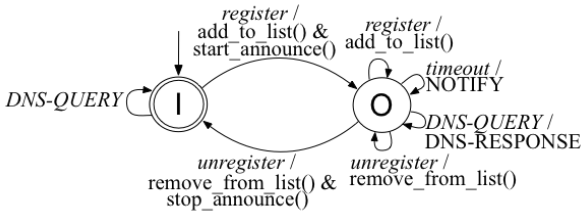


Figura 4.24: Diagrama de estados do provedor ativo.

Na tabela 4.7 são descritos os eventos e o funcionamento do *Middleware* correspondente ao diagrama da figura 4.24.

Estado atual	Evento	Ação	Novo estado
Iddle	*DNS-QUERY*[from_net]		Iddle
Iddle	*register()*[local_api]	add_to_list() & start_announce()	Offering
Offering	*register()*[local_api]	add_to_list()	Offering
Offering	*DNS-QUERY*[from_net]	DNS-RESPONSE[to_net]	Offering
Offering	*timeout*	NOTIFY	Offering
Offering	*unregister()*[local_api]	remove_from_list()	Offering
Offering	*unregister()*[local_api]	remove_from_list() & stop_announce()	Iddle

Tabela 4.7: Estados do provedor ativo.

Quando o *Middleware* em um dispositivo é um provedor ativo, é possível que outros dispositivos sejam clientes passivos. Quando atuando como um cliente passivo, o *Middleware* inicialmente está em um estado de espera no qual não processa requisições de ofertas de serviços. Após receber uma notificação *lookup()* da API local, o *Middleware* altera seu estado e passa a processar todas as requisições de ofertas recebidas. No recebimento destas requisições, o *Middleware* processa e verifica se o serviço ofertado é o mesmo serviço buscado, e caso seja, é enviada uma notificação de *callback* para a aplicação solicitante. Podem ser realizadas novas chamadas *lookup()* e *stop_lookup()* que o estado de funcionamento permanece aceitando requisições. O estado só retorna para em espera quando realizada uma chamada *stop_lookup()* na API local e não existirem mais demandas sendo buscadas. Esse funcionamento é exibido no diagrama de estados presente na figura 4.25.

Os eventos tratados no diagrama da figura 4.25 são exibidos na tabela 4.8.

Dessa forma, conforme descrito, o *Middleware* e as aplicações terão algumas rotinas básicas bem definidas. Estas rotinas estão relacionadas com as interações envolvendo os agentes descritos, e estas rotinas previstas serão utilizadas pelas aplicações através da API disponível. Estas funcionalidades são:

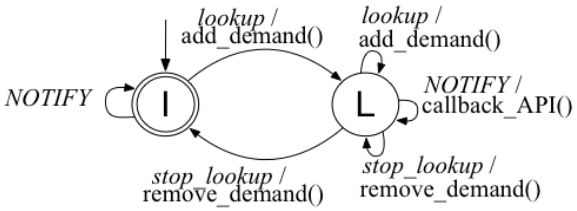


Figura 4.25: Diagrama de estados do cliente passivo.

Estado atual	Evento	Ação	Novo estado
Iddle	*NOTIFY*[from_net]		Iddle
Iddle	*lookup()*[local_api]	add_demand()	Listening
Listening	*lookup()*[local_api]	add_demand()	Accepting
Listening	*NOTIFY*[from_net]	callback_API()	Listening
Listening	*stop_lookup()*[local_api]	remove_demand()	Listening
Listening	*stop_lookup()*[local_api]	remove_demand	Iddle

Tabela 4.8: Estados do cliente passivo.

- Registro de serviço.
- Destruição de registro de um serviço.
- Busca por um serviço.
- Criação de uma demanda em busca de um serviço.
- Destruição de uma demanda criada previamente.

Estes eventos são criados pelas aplicações em busca de funcionalidades e envolvem uma comunicação local no dispositivo. Portanto o *Middleware* provê as funcionalidades e os recursos necessários descritos.

Todas comunicações originadas do *Middleware* com destino às aplicações são locais e ocorrem na forma de *callbacks* através de requisições HTTP no contexto */middleware-notification*.

Todas as comunicações que vêm de outros dispositivos com destino ao *Middleware* são comunicações referentes aos mecanismos de descoberta de dispositivos implementados pelo *Middleware*. O *Middleware* não aceita nenhuma requisição externa de nenhuma outra forma de comunicação.

As comunicações originadas por outros dispositivos com destino às aplicações locais são realizadas sempre na forma de requisições HTTP. Os serviços oferecidos por tais aplicações são disponibilizados por requisições HTTP em contextos específicos definidos pelas próprias aplicações.

Dessa forma, é exibido o *Middleware* auxiliando o modelo de comunicação proposto e são mostradas as possíveis interações que envolvem os agentes presentes nas condições esperadas. Com as interações testadas, é possível concluir que a API proposta é válida e oferece as funcionalidades do *Middleware* descritas. O *Middleware* conforme descrito, se mostrou funcional nas simulações realizadas e também atendeu aos requisitos de outras aplicações móveis supostas.

Capítulo 5

CONCLUSÕES

Como pode ser visto na seção destinada aos resultados, o *Middleware* realizou as funcionalidades de descoberta de dispositivos e serviços como era esperado. A simulação utilizou uma implementação da API apresentada neste trabalho para permitir que os dispositivos se comunicassem. Nestas simulações, dispositivos possuem uma aplicação móvel que contém dois serviços, *file-transfer* e *information-exchanger*, conforme descritos neste trabalho. Estes serviços são oferecidos através do *Middleware* que utiliza a simplificação para identificação de serviços descrita.

Os dados e conclusões aqui apresentados foram resultados de simulações executadas com densidade de 120 pedestres e 400 veículos por Km^2. Ainda é necessário realizar estudos com maiores números de dispositivos nos cenários, o que requer longos tempos de simulação. Enquanto cada simulação realizada precisa de 7 dias para conclusão, simulações maiores levaram em média 21 dias.

Foi observado que as interações entre os dispositivos ocorreram como o esperado. Os valores para referência foram estimados através de cálculos baseados nas velocidades de deslocamento dos dispositivos nas simulações. Esses valores representam um tempo para comunicação de 4,5 segundos para dispositivos associados a veículos utilizando tecnologia de transmissão sem fio WiFi. Para dispositivos de pedestres a maior parte dos tempos de contato tem duração de 45 segundos. Além das estimativas obtidas, foi observado que os tempos de contato entre veículos ocorrem com duração de até 12 segundos, neste caso, para veículos que se deslocam na mesma direção na mesma via.

Foi observado que, embora os tempos de contato entre dispositivos móveis sejam pequenos, por volta de 5 segundos no caso de veículos, estes poucos segundos foram suficientes para que os dispositivos fossem descobertos utilizando o *Middleware* para descoberta de dispositivos e

serviços. Como esperado, os protocolos foram capazes de trocar as mensagens necessárias para seus funcionamentos e descobrir os dispositivos e serviços em fração de décimos de segundos, o que é plausível dado que os dispositivos se comunicaram diretamente através de comunicação sem fio. Não houve diferenças no funcionamento do *Middleware* por decorrência do protocolo de descoberta utilizado.

Durante a execução das simulações, foi constatado que se forem utilizados intervalos muito pequenos entre os envios de mensagens de descoberta a comunicação entre os nós é prejudicada. Intervalos menores do que 1 segundo trouxeram perdas nas trocas de mensagens entre os dispositivos, o que é justificado pelo fato que se muitos dispositivos desejarem enviar muitas mensagens simultaneamente, nenhum dispositivo utilizará o meio sem fio de maneira eficaz. Intervalos inferiores a meio segundo ocasionaram dispositivos sendo praticamente incapazes de descobrir e consumir serviços de outros dispositivos.

Nas simulações executadas, o envio de mensagens de descoberta com frequência de 12 vezes por minuto (uma mensagem a cada 5 segundos) é suficiente para permitir que um dispositivo portado por um pedestre descubra 90% dos outros dispositivos. Por outro lado, veículos precisam enviar mensagens de descoberta 20 vezes por minuto (uma mensagem a cada 3 segundos) para descobrir 90% dos dispositivos próximos. Dispositivos portados por pedestres enviando mensagens com uma frequência de 30 mensagens por minuto são capazes de descobrir aproximadamente 98% dos dispositivos, enquanto dispositivos portados por veículos para obterem esta mesma eficácia precisam enviar mensagens com uma frequência de 45 mensagens por minuto.

O tempo de contato disponível para os dispositivos foi suficiente para que os dispositivos realizassem as chamadas de serviços para consumo dos serviços desejados. Isto é observado pois um pequeno tempo de contato entre dispositivos, de por exemplo 1 segundo, foi suficiente para trocas de mensagens entre dispositivos para consumo de serviços que não demandem muito tempo de processamento. Como esperado também, o modelo de simplificação de serviços utilizado neste trabalho não demandou processamento exagerado, o que permite que o *Middleware* identifique serviços com tempos na ordem de décimos de segundos.

Portanto, este trabalho apresentou um *Middleware* que auxiliou a implementação de serviços, provendo para estes a funcionalidade de descoberta de dispositivos e serviços através da API disponível. Este *Middleware* conseguiu descobrir serviços e informar os serviços a tempo de que estes pudessem consumir os serviços desejados. Tentando enviar mensagens de descoberta com a menor frequência possível, ocupando assim o mínimo possível a interface de comunicação sem fio, foram constatados intervalos que oferecem adequada taxa de descoberta

de dispositivos. Além disto, como seria esperado, este trabalho observou que dispositivos que se deslocam mais velozmente devem enviar mensagens de descoberta com mais frequência para obterem a mesma taxa de descoberta que dispositivos mais lentos.

Novos serviços e aplicações que forem desenvolvidos podem utilizar campos explícitos de Mime-type e ACTION para definição de serviços, consequentemente simplificando o processo de *matching* do *Middleware*.

Atualmente, os serviços disponíveis não possuem metadados associados, dessa forma é mais difícil realizar o procedimento de identificação de serviços. Isso se deve ao fato de que sem metadados, a única forma de identificação de serviços é através do nome do serviço, onde o nome do serviço já deve pressupor um ou mais tipos de dados e ações pré-definidas. Para auxiliar na identificação de servicos, o *Middleware* disponibiliza serviços com metadados associados, apesar disto, o *Middleware* é capaz de lidar com serviços sem qualquer metadado associado. Além dos metadados inseridos pelo *Middleware* que utiliza a simplificação proposta, é possível a incorporação de ontologias que podem tratar outros metadados, tornando assim a identificação de serviços mais flexível.

5.1 Trabalhos futuros

Está em desenvolvimento a implementação de uma versão em ambiente real do *Middleware* que possua suporte dos protocolos de descoberta de dispositivos apresentados. Para utilizar as implementações destes protocolos serão utilizados códigos já implementados por terceiros através de bibliotecas compartilhadas. Para o protocolo Bonjour, a empresa Apple disponibiliza a implementação de registros mDNS compatíveis com o protocolo[1]. Para o protocolo UPnP pode-se utilizar a biblioteca[2]. Para utilização do protocolo SLP, é prevista a utilização da biblioteca OpenSLP[3].

É esperado realizar em trabalhos futuros, a simulação da prestação de serviços com tamanhos diferentes de arquivos transferidos entre serviços. Essa simulação servirá para análise do impacto do tamanho dos dados transferidos durante o consumo de um serviço, pois é sabido que esta variável influencia na taxa de sucesso de consumo de serviços.

Em trabalhos futuros, também é esperado estudar e implementar mecanismos de segurança que envolvem a comunicação do *Middleware* com as aplicações. Neste sentido, alguns proto-

[1] http://www.opensource.apple.com/source/mDNSResponder/mDNSResponder-333.10.
[2] http://pupnp.sourceforge.net/
[3] http://openslp.org/doc/html/IntroductionToSLP/index.html

colos oferecem a possibilidade de utilização de mecanismos que protejam os serviços sendo oferecidos, como por exemplo a utilização de chaves assimétricas para controle de acesso no protocolo SLP e a definição de usuários com identificação e senha de acesso para consumo de serviços no protocolo UPnP. Porém, o protocolo Bonjour não possui nenhum mecanismo para controle de acesso das informações dos nós. Para identificação e oferta de serviços um mecanismo que pode ser adequado é a utilização de reputação sobre os dispositivos, de forma que esta reputação seja baseada na qualidade dos serviços prestados. Com estes aspectos é esperado que o *Middleware* possua autenticação e confidencialidade nas operações realizadas.

É esperado também adicionar ao *Middleware* mecanismos para obtenção e disponibilização para as aplicações de informações de contexto.

O estudo da probabilidade de que dispositivos específicos encontrem serviços específicos pode ser útil para o modelo apresentado. Desta forma, estudar meios para determinar a probabilidade de serviços serem encontrados no ambiente é importante e é considerado para trabalhos futuros.

REFERÊNCIAS

Ahmed Surobhi, N.; JAMALIPOUR, A. A Context-Aware M2M-Based Middleware for Service Selection in Mobile Ad-Hoc Networks. *IEEE Transactions on Parallel and Distributed Systems*, v. 9219, n. c, p. 1–1, 2014. ISSN 1045-9219. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6747333>.

ANDREWS, M. *Negative Caching of DNS Queries (DNS NCACHE)*. IETF, mar. 1998. RFC 2308 (Proposed Standard). (Request for Comments, 2308). Updated by RFCs 4035, 4033, 4034, 6604. Disponível em: <http://www.ietf.org/rfc/rfc2308.txt>.

BONJOUR, A. *Bonjour implementation by Apple Inc*. 2008. Disponível em: $<http : //www.apple.com/support/bonjour/>$.

CHA, S.; DU, W. Context Aware Middleware Services for Disconnection Tolerant Mobile Applications. *2011 Ninth Annual Communication Networks and Services Research Conference*, p. 145–152, 2011. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5771204>.

CHESHIRE, S.; ABOBA, B.; GUTTMAN, E. *Dynamic Configuration of IPv4 Link-Local Addresses*. IETF, maio 2005. RFC 3927 (Proposed Standard). (Request for Comments, 3927). Disponível em: <http://www.ietf.org/rfc/rfc3927.txt>.

FIELDING, R. et al. *Hypertext Transfer Protocol – HTTP/1.1*. IETF, jun. 1999. RFC 2616 (Draft Standard). (Request for Comments, 2616). Updated by RFCs 2817, 5785, 6266, 6585. Disponível em: <http://www.ietf.org/rfc/rfc2616.txt>.

FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. Disponível em: $<https : //www.ics.uci.edu/ fielding/pubs/dissertation/rest_arch_style.htm>$.

FORUM, U. *UPnP Device Architecture*. 2008. Disponível em: $<http : //www.w3.org/TR/2004/NOTE-ws-gloss-20040211/\#webservice>$.

FREED, N.; BORENSTEIN, N. *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. IETF, nov. 1996. RFC 2046 (Draft Standard). (Request for Comments, 2046). Updated by RFCs 2646, 3798, 5147, 6657. Disponível em: <http://www.ietf.org/rfc/rfc2046.txt>.

GUTTMAN, E. *Vendor Extensions for Service Location Protocol, Version 2*. IETF, jan. 2002. RFC 3224 (Proposed Standard). (Request for Comments, 3224). Disponível em: <http://www.ietf.org/rfc/rfc3224.txt>.

GUTTMAN, E. et al. *Service Location Protocol, Version 2*. IETF, jun. 1999. RFC 2608 (Proposed Standard). (Request for Comments, 2608). Updated by RFC 3224. Disponível em: <http://www.ietf.org/rfc/rfc2608.txt>.

HOLLER, J. et al. *From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence*. [S.l.]: Academic Press, 2014.

HUERGO, R. S. et al. *A systematic survey of service identification methods*. 2014.

KNUTH, D. E. *The Art of Computer Programming*. [S.l.]: Addison-Wesley Professional, 1973.

LOUREIRO, E. et al. A flexible middleware for service provision over heterogeneous pervasive networks. *Proceedings - WoWMoM 2006: 2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks*, v. 2006, p. 609–614, 2006.

MADANI, Z. et al. A logical representation and verification of web service choreography. In: *3rd International Symposium on Intelligent Information Technology Application, IITA 2009*. [S.l.: s.n.], 2009. v. 2, p. 404–407. ISBN 9780769538594.

OASIS. *OASIS Web Services Business Process Execution Language (WSBPEL) TC*. 2007. Disponível em: <https://www.oasis-open.org/committees/wsbpel/>.

POSLAD, S. *Ubiquitous Computing - Smart devices, Enviroments and Interactions*. [S.l.]: WILEY, 2009.

ROSCIA, M.; LONGO, M.; LAZAROIU, G. Smart City by multi-agent systems. *2013 International Conference on Renewable Energy Research and Applications IEEE*, n. October, p. 20–23, 2013. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6749783>.

SAHA, D.; MUKHERJEE, A. Pervasive computing: A paradigm for the 21st century. *Computer*, v. 36, n. March, p. 25–31+4, 2003. ISSN 00189162.

W3C. *Web Services Glossary*. 2004. Disponível em: <http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice>.

W3C. *Web Services Choreography Description Language Version 1.0*. 2005. Disponível em: <http://www.w3.org/TR/ws-cdl-10/>.

WEISER, M. The computer for the 21st Century. *IEEE Pervasive Computing*, v. 1, 2002. ISSN 1536-1268.

WEISER, M.; SEELY BROWN, J. *Designing Calm Technology*. 1995. Disponível em: <http://www.ubiq.com/hypertext/weiser/calmtech/calmtech.htm>.

WIFI-ALLIANCE. *Wi-Fi Peer-to-Peer (P2P) Technical Specification Version 1.5*. 2014. Disponível em: <https://www.wi-fi.org/download.php?file=/sites/default/files/private/Wi-Fi_P2P_Technical_Specification_v1.5.pdf>.

WOHLSTADTER, E. et al. A service-oriented middleware for runtime Web services interoperability. *Proceedings - ICWS 2006: 2006 IEEE International Conference on Web Services*, p. 393–400, 2006.

GLOSSÁRIO

CUPS – *Common Unix Printing System*

DA – *Directory Agent*

DHCP – *Dynamic Host Configuration Protocol*

HDP – *Host Discovery Plugin*

HTTP – *HyperText Transfer Protocol*

IBGE – *Instituto Brasileiro de Geografia e Estatística*

IETF – *Internet Engineering Task Force*

IP – *Internet Protocol*

MTU – *Maximum Transmission Unit*

OSI – *Open Systems Interconnection*

PAN – *Personal Area Network*

RFC – *Request For Comment*

RWP – *Random Way Point*

SA – *Service Agent*

SLP – *Service Location Protocol*

SPP – *Service Provision Plugin*

SSDP – *Simple Service Discovery Protocol*

SSH – *Secure Shell*

UA – *User Agent*

UPnP – *Universal Plug and Play*

URL – *Uniform Resource Locator*

VNC – *Virtual Networking Computing*

XML – *Extension Markup Language*

mDNS – *multicast Domain Name System*