UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS ARARANGUÁ

Janquiel Pereira

INTEGRAÇÃO DE DISPOSITIVOS HETEROGÊNEOS VIA TECNOLOGIA DLNA

Araranguá, Dezembro de 2014.

Janquiel Pereira

INTEGRAÇÃO DE DISPOSITIVOS HETEROGÊNEOS VIA TECNOLOGIA DLNA

Trabalho de Curso submetido à Universidade Federal de Santa Catarina, como parte dos requisitos necessários para a obtenção do Grau de Bacharel em Tecnologias da Informação e Comunicação.
Orientador: Gustavo Medeiros de Araujo.

.

Araranguá, Dezembro de 2014.

Janquiel Pereira

INTEGRAÇÃO DE DISPOSITIVOS HETEROGÊNEOS
VIA TECNOLOGIA DLNA

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de Bacharel em Tecnologias da Informação e Comunicação, e aprovado em sua forma final pelo Curso de Graduação em Tecnologias da Informação e Comunicação.

Araranguá, Dezembro de 2014.

Prof. Vilson Gruber, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Gustavo Medeiros de Araujo Dr.
Orientador
Universidade Federal de Santa Catarina

Prof.ª Luciana Bolan Frigo Dr.ª
Universidade Federal de Santa Catarina

Prof.ª Analucia Schiaffino, Dr.ª
Universidade Federal de Santa Catarina

Este trabalho é dedicado aos meus colegas de classe, meus queridos pais e a todos que contribuíram para minha formação.

AGRADECIMENTOS

Com grande prazer presto meus agradecimentos as pessoas que contribuíram para o sucesso de minha formação.

Agradeço primeiramente Deus pela minha vida.

Aos meus colegas de classe que me deram força para continuar nesta jornada.

Agradeço a minha família pelo incentivo e carinho nas horas difíceis.

Ao orientador Prof. Gustavo Medeiros de Araujo que teve papel importante no desenvolvimento do presente trabalho.

"É muito melhor lançar-se em busca de conquistas grandiosas, mesmo expondo-se ao fracasso,
do que alinhar-se com os pobres de espírito, que nem gozam muito nem sofrem muito, porque vivem numa penumbra cinzenta, onde não conhecem em vitória, nem derrota"

(Theodore Roosevelt)

RESUMO

Em 2003, foi proposto um acordo entre os fabricantes de eletrônicos, que teve como objetivo trazer a interoperabilidade entre diferentes equipamentos fabricados. Através deste acordo, o DLNA (Digital Living Network Alliance) surge. Esta tecnologia utiliza vários protocolos como UPnP (Universal Plug and Play), para promover a integração entre os dispositivos.

Ao longo dos anos, esta tecnologia tem sido disseminada, presente em muitos dispositivos atualmente vendidos, mas ainda há dispositivos que não possuem essa tecnologia.

Dispositivos, como o Arduino, não possuem a tecnologia DLNA, e exigem a construção de um *middleware* para comunicação entre o Arduino com outros dispositivos remotos na rede.

Este trabalho propõe uma integração entre dispositivos com comunicações heterogêneos através da tecnologia DLNA, e esses dispositivos são extremamente necessários para a automação residencial.

A automação residencial é uma realidade hoje. Assim, usando uma tecnologia como DLNA, para integrar todos os dispositivos presentes em uma casa, pode proporcionar inteligência para a casa, a fim de trazer mais conforto e segurança para os seus habitantes. Controles de iluminação, ventilação, alarmes etc, podem ser centralizados em um único dispositivo capaz de gerenciar os outros e proporcionar a integração através da tecnologia DLNA através de um roteador simples.

Palavras-chave: Arduíno. DLNA. UPNP. Domótica

ABSTRACT

In 2003, it was proposed an agreement between electronics manufacturers which aimed to bring interoperability between different equipment manufactured. Through this agreement, the DLNA (Digital Living Network Alliance) arises technology using various protocols such as UPnP (Universal Plug and Play), to promote integration between devices.

Over the years, this technology has been disseminated, present in many devices currently sold, but there are still devices that do not have this technology.

Devices such as the Arduíno, lack of DLNA technology, requiring the construction of a middleware for communication between the Arduíno with other remote devices on the network.

This paper proposes an integration between devices with heterogeneous communications via DLNA technology, and such devices are extremely necessary for home automation.

The home automation is a reality today. Thus, using a technology such as DLNA, to integrate all devices present in a house, can provide intelligence to the house in order to bring more comfort and safety for its inhabitants. Lighting controls, ventilation, alarms etc. can be centralized in a single device capable of managing others and provide their integration via DLNA technology through a simple router.

Keywords: Arduíno. DLNA. UPNP. Home Automation.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

DLNA – Digital Living Network Aliance
UPnP – Universal Plug and Play
USB – Universal Serial Bus
UDN – Unique Device Name
URN – Uniform Resourse Name
IDE - Integrated Development Environment

Sumário

1. Introdução

A cada dia diversos dispositivos computacionais vêm sendo criados, *Smartphones*, *Tablets*, consoles para games, sensores, entre outros.

O desenvolvimento de microcontroladores também vem crescendo exponencialmente devido à vasta aplicabilidade.

O *Arduíno* é uma plataforma que conjuntamente com sensores, e outros dispositivos podem proporcionar o desenvolvimento de uma *SmartHouse* ou casa inteligente.

As casas inteligentes podem ser vistas como uma plataforma que pode ser implementada para diversas aplicações. Dentre estas aplicações, a casa inteligente pode realizar o monitoramento de idosos ou pessoas com deficiências, com intuito de tornar o cotidiano mais seguro e confortável.

> "Na era da tecnologia da informação, os idosos e deficientes podem ser monitorados com inúmeros dispositivos inteligentes. Os sensores podem ser implantados em sua casa fornecendo assistência à mobilidade contínua e prevenção de doenças não intrusivas. Casas incorporadas por sensores modernos, ou casas inteligentes, não só pode ajudar as pessoas com reduzidas funções físicas, mas ajudar a resolver o isolamento social que enfrentam. Eles são capazes de fornecer assistência sem limitar ou perturbar o cotidiano do morador, dando-lhe maior conforto, prazer e bem-estar." (CHAN, Marie, et al. 2008).

Uma barreira para a automação residencial é a dificuldade em fornecer a interoperabilidade entre os diferentes dispositivos.

Uma das soluções para tal barreira foi criada em 2003 pela Sony juntamente com outras 23 empresas. O consórcio entre estas empresas desenvolveu uma tecnologia baseada

em padrões para facilitar a vida dos consumidores. Esta aliança entre as empresas de tecnologia resultou na tecnologia hoje conhecida como DLNA. Hoje, são mais de 240 membros associados ao DLNA trabalhando para deixar as pessoas cada vez mais conectadas (HAGEDORN, 2012).

Apesar de o DLNA ter sido criado para a distribuição multimídia, há uma grande tendência em utilizá-lo para integrar dispositivos, cujos serviços não se restringem a distribuição de multimídia e que não possuem a certificação DLNA. É neste contexto que se insere o presente projeto.

Com a gama de dispositivos diversos de diferentes plataformas, faz-se necessário a criação de uma camada de software que provenha à integração destes dispositivos com o intuito de aumentar a disponibilização de serviços em um ambiente ubíquo, como por exemplo uma Smart House onde existem diversos dispositivos fornecendo serviços distintos .

Esta camada de software é também conhecida como *middleware*, que no sentido estrito é um software de transporte usado para mover informação de um ou mais programas para outro (Talarian, 2000).

A principal função do *middleware* no projeto é trazer independência da aplicação com o sistema de transmissão DLNA e fornecer o canal necessário para a troca de dados entre os dispositivos espalhados pela casa.

Com o auxilio do *middleware* desenvolvido, os dispositivos heterogêneos utilizados pela Smart House poderão comunicar-se entre si sem a necessidade de configuração externa. Quando um dispositivo se conectar a rede ele se autoconfigura sem a necessidade do usuário informar portas, IPs e outras configurações até então necessárias. A única preocupação de um programador será a implementação do serviço para o qual o dispositivo foi disponibilizado.

Pode-se utilizar o *middleware* proposto através de um *service broker*, que consiste em fazer a tradução entre requisições nativas DLNA para requisições especificas da plataforma adotada.

No contexto das casas inteligentes, o *Arduíno* pode ser uma das plataformas que torna possível a automatização de tarefas. Ele pode agir como atuador, sendo alimentado por dispositivos sensores, podendo controlar a climatização de uma casa ou abrir o portão quando o dono da casa chegar por exemplo.

O *Arduíno* é uma plataforma de prototipagem eletrônica de baixo custo que é capaz de comunicar-se com um *PC, notebook, RaspBerrypi* entre outros e conjuntamente ou individualmente executar tarefas de automação (WADDINGTON, 2007).

Para a automação das tarefas foi criado um servidor DLNA genérico que fará conexões com as placas de prototipagem através do *middleware.* As tarefas serão executadas em uma rede local, e o servidor também será capaz de receber informações das placas ou sensores, podendo assim tomar decisões de acordo com o fim para qual foi programado.

Neste projeto, serão adotadas as métricas de teste de exaustão, e teste de velocidade de resposta com a finalidade de descobrir qual a melhor implementação possível.

1.1. Objetivos

1.1.1. Objetivo Geral

Propor uma arquitetura computacional baseado no protocolo UPNP, por meio da tecnologia DLNA, para fazer a integração de dispositivos heterogêneos.

1.1.2. Objetivos Específicos

Seguem os objetivos específicos para desenvolvimento do projeto:

- Estudar a tecnologia DLNA.
- Estudar a plataforma *Arduíno.*
- Desenvolver um servidor DLNA baseado na aplicação *open source Cling*.

- Efetuar a comunicação do servidor DLNA com o dispositivo Arduíno.
- Realizar testes de exaustão.
- Realizar testes tempo de resposta.

1.2. Metodologia

Para o desenvolvimento do projeto foi adotado uma pesquisa aplicada, utilizando-se do dispositivo Arduíno para o sucesso do mesmo.

Primeiramente, foi feito uma pesquisa sobre a tecnologia DLNA, e a utilização da plataforma de prototipagem Arduíno.

A pesquisa está focada no desenvolvimento do *Middleware* baseada na tecnologia DLNA.

1.3. Organização do texto

O documento esta dividido em seis capítulos, onde o primeiro trás uma breve introdução do tema a ser tratado durante o decorrer do documento.

O capítulo dois descreve todas as tecnologias que foram estudadas e utilizadas para o desenvolvimento do trabalho.

No terceiro capítulo é apresentada uma descrição mais detalhada do que foi desenvolvido mostrando como o protótipo funciona e quais são seus principais componentes.

O quarto capítulo traz informações sobre a proposta do trabalho, ilustrando através de diagramas de sequência o funcionamento da arquitetura e suas diferentes formas de comunicação.

Os testes realizados estão dispostos no quinto capítulo onde foi feita uma analise dos dados e apresentados os resultados obtidos.

O sexto capítulo contém as considerações finais e indicações para projetos futuros, seguido posteriormente das referências.

2. Descrição das Tecnologias

2.1. DLNA (Digital Living Network Alliance)

Segundo o site oficial do DLNA, a fim de alcançar interoperabilidade em casa, a DLNA lançou um conjunto de diretrizes de design que permitem que a indústria participe do crescente mercado de dispositivos de rede (DLNA, 2014).

O DLNA foi proposto em 2003 pela Sony que juntamente com outras 23 empresas concordaram em fazer uma tecnologia baseada em padrões onde seus produtos pudessem se comunicar em uma rede mesmo sendo fornecidos por fabricantes diferentes (HAGEDORN, 2012).

Com o DLNA, surge uma solução para facilitar o acesso a arquivos multimídia em diversos dispositivos eletrônicos. Um padrão de conectividade foi estabelecido e, através dele, é possível compartilhar músicas, vídeos e fotos entre dispositivos de marcas e modelos diferentes (FRANCO, 2012).

Esta tecnologia trabalha com três tipos principais de conexão, Wirelles, Ethernet e MoCa, e os dispositivos e aplicativos são classificados de acordo com a função proposta.

- **Digital Media Servidor (DMS):** Dispositivos capazes de fornecer aos demais o poder de receber, armazenar, e também gravar conteúdo de mídia (DLNA, 2014).
- **Digital Media Player (DMP):** Dispositivos capazes de encontrar DMS na rede e assim, reproduzir conteúdos de multimídia (DLNA, 2014).
- **Digital Media Renderer (DMR):** Dispositivos com o poder de reproduzir conteúdo multimídia recebido de um DMS vindo de outro dispositivo da rede como um DMC (DLNA, 2014).
- **Digital Media Controller (DMC):** Dispositivos capazes de controlar o conteúdo multimídia

> proveniente de um DMS que está sendo reproduzido por um DMP, uma espécie de controle remoto (DLNA, 2014).

Apesar de o DLNA ter sido criado para o compartilhamento de arquivos multimídia, observa-se uma grade tendência em utilizar esta tecnologia para outros fins, possibilitando novos rumos para ela, e tornando cada dia a vida das pessoas mais interligada.

A tecnologia DLNA trabalha com protocolos UPNP, fornecendo interatividade direta com outras aplicações DLNA que estiverem na rede. Ao se conectar em uma rede, o dispositivo ou aplicação faz um teste e se auto configura para poder trabalhar naquela rede. Em seguida, o dispositivo faz uma busca por outros dispositivos que podem ser compatíveis com sua finalidade.

Se um dispositivo é capaz de emitir e receber conteúdo multimídia, ele vai se identificar na rede, ficando disponível para outros dispositivos poderem se conectar a ele para receber ou enviar dados, assim um *SmartPhone* com DLNA pode transmitir fotos armazenadas diretamente para a tela de uma TV, sem a necessidade de cabos, utilizando apenas a rede *Wi-Fi* da residência.

Com o aperfeiçoamento da tecnologia e a sua grande disseminação, surgiram milhares de produtos certificados e aplicações dos mais variados tipos. Hoje no mercado, existem várias TVs que fornecem aplicativos para *SmartPhones* que possibilitam a execução de tarefas antes executadas pelo controle remoto.

2.2. UPnP (Universal Plugin and Play)

Universal Plug and Play é um conjunto de protocolos da rede de computadores lançados em 1999 pelo fórum UPnP. Uma das principais metas desta iniciativa é permitir que aparelhos conectem-se de uma maneira simplificada e rápida possibilitando maior conectividade em redes domésticas e em escritórios (Fórum UPnP, 2014).

A arquitetura dos protocolos trabalha de maneira aberta e distribuída, utilizando o TCP/IP para fornecer uma conectividade perfeita (Fórum UPnP, 2014).

Existem diversos tipos de dispositivos e aplicações que utilizam os protocolos UPnP, pois seus benefícios são inúmeros. Dispositivos como roteadores, Smart Phones, Smart TV´s são apenas alguns dos inúmeros dispositivos que utilizam este protocolo.

Os protocolos podem ser executados em qualquer tipo de rede, incluindo Wi-Fi, coaxial, linha telefônica, linha de alimentação, Ethernet. O UPnP é uma tecnologia independente de plataforma, e independente de linguagem de programação (Fórum UPnP, 2014).

O UPnP fornece um serviço de "*Zero Configuration and Automatic Discovery*", ou seja os dispositivos conectados a rede podem automaticamente, aderir uma rede, obter um endereço IP, anunciar seu nome e suas capacidades. Além disso, os dispositivos podem ficar cientes da existência de outros dispositivos e de suas capacidades (Fórum UPnP,2014).

2.3. Plataformas de Prototipagem Arduíno

O Arduíno é uma placa de prototipagem eletrônica utilizada para simplificar e tornar mais acessível á robótica.

Segundo SOUZA (2011), "O Arduíno é uma placa de hardware *open source* de fácil utilização e ideal para criação de dispositivos que permitam interação com o ambiente".

No mercado existe uma série de tipos de placa Arduíno, diferenciando entre si em quantidade de pinos, velocidade de *clock* e memória.

O Arduíno mais simples é o Arduíno Uno que marca o inicio de uma nova geração de placas. Ele se difere dos modelos anteriores, pois possui uma placa controladora baseada no ATmega 328 trabalhando a 16MHz. Ele possui 14 pinos digitais de entrada/saída, seis entradas analógicas,

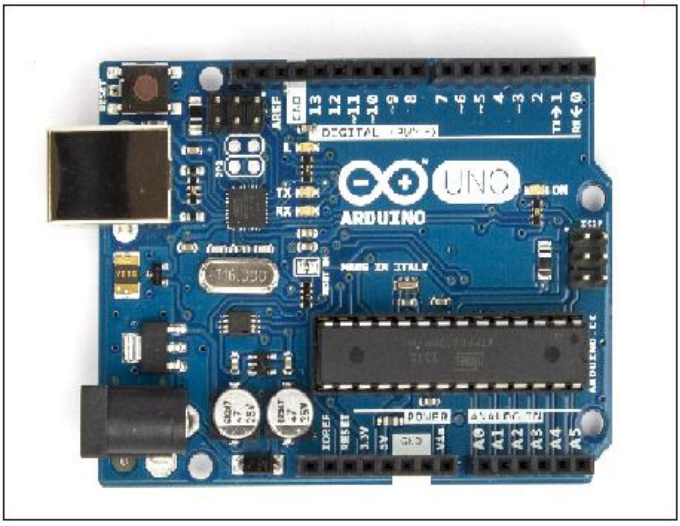
conexão USB, e uma entrada para fonte externa (Arduíno, 2014).

Apesar de ele ser simples e barato, suas aplicabilidades são inúmeras. Sua utilidade se torna maior ainda com a utilização das placas *Shields*, que possibilitam novas funções, como por exemplo, conexão via GPRS e conexão Ethernet.

O Arduíno possui uma IDE muito amigável feita em Java, e utiliza programação baseada em C/C++. AS funções da IDE permitem o desenvolvimento de software que possa ser executado pelo dispositivo (Arduíno, 2014).

Na figura 1 pode-se observar a ilustração de um Arduíno Uno, que é um dos mais simples Arduínos existentes.

Figura 1: Arduíno Uno



Fonte: Arduíno (2014).

2.4. Domótica

Com o grande avanço das tecnologias surge a cada dia a necessidade de estar mais conectado.

Os celulares tornaram-se computadores que se carrega no bolso, relógios não servem mais apenas para ver as horas, televisores são verdadeiras centrais multimédia, entre outras evoluções que facilitam muito a vida de todos.

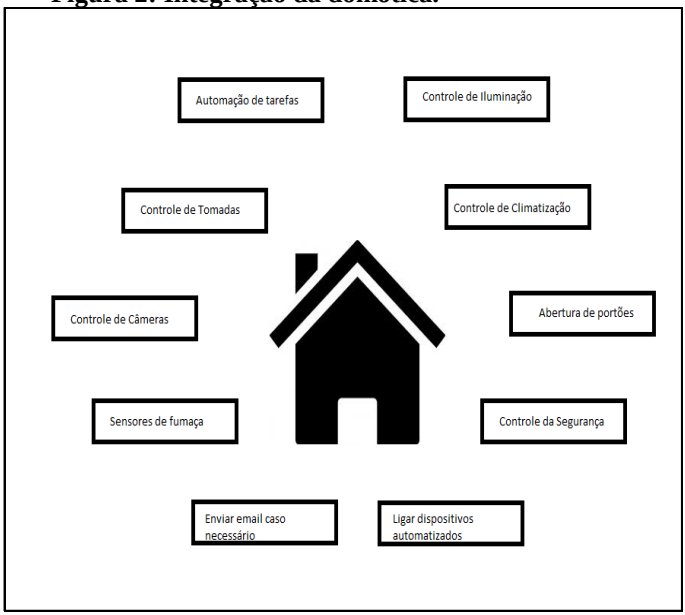
Uma realidade que não esta tão distante são as casas inteligentes ou *Domótica*. *A domótica* pode fornecer ainda mais conectividade e segurança para as pessoas automatizando tarefas do dia a dia.

O termo domótica é uma fusão da palavra "*Domus*" (casa) com "Telemática" (eletrônica + informática). A domótica também e referenciada por "*Smart house*", ou casas inteligentes. Quando a domótica surgiu a ideia principal era controlar iluminação, condições climáticas e segurança. Hoje em dia, a ideia é praticamente a mesma só que existem muito mais dispositivos que podem ser integrados a domótica (PALMA, 2012).

Algum tempo atrás, era visto em filmes, casas onde apenas com gestos ou com comandos de voz executava tarefas antes realizadas apenas por humanos. Com a criação da tecnologia DLNA, a domótica vem deixando de ser apenas uma peça de ficção, e passa a se tornar uma realidade.

A figura 2 ilustra algumas das muitas automações que a domótica pode possuir.

Figura 2: Integração da domótica.



2.5. Cling

Cling é uma biblioteca JAVA/Adroid que procura implementar os protocolos UPnP mais fielmente possível, o objetivo do projeto é disponibilizar uma biblioteca de fácil personalização e que atenda as especificações UPnP (Cling, 2014).

O projeto Cling apresenta diversos modelos com manuais de fácil entendimento para usuários que não participam da iniciativa possam entender e modelar os códigos conforme for necessário.

3. Integração de dispositivos heterogêneos

O projeto consiste em uma pesquisa sobre a tecnologia DLNA, e a aplicação da mesma em dispositivos heterogêneos que podem ser utilizados em projetos de automação industrial ou residencial.

O presente projeto é uma implementação baseada na aplicação *Cling* disponível online de forma *OpenSource* (Cling, 2014).

Para o sucesso do mesmo foi adotado o Arduíno Uno para a elaboração dos testes de funcionalidades e desempenho.

A interação entre o DLNA com a plataforma de prototipagem, os dispositivos sensores e os dispositivos atuadores não pode ser realizada diretamente. Neste caso, foi desenvolvido um servidor DLNA que age diretamente junto ao Arduíno. Este Servidor é responsável por capturar as informações obtidas pelo Arduíno e enviá-las para o *Service Broker* ou para outro servidor disponível na rede para serem tratadas e redirecionadas para os devidos dispositivos responsáveis por executarem as tarefas de automação.

Esta abordagem pode levar ao desenvolvimento combinatório de diversas implementações entre os diferentes dispositivos e diversas tecnologias.

Um exemplo da utilização do presente projeto é na automação de um portão eletrônico, onde um usuário com um SmartPhone pode abrir o portão através da utilização de um serviço DLNA em conjunto com um Arduíno, após abrir o portão o serviço DLNA fica aguardando as informações de um sensor de presença para poder fechar o portão.

Este é um exemplo simples da utilização da tecnologia, mas pode-se destacar a segurança e a praticidade que ela proporciona, pois somente quem possuir a senha da rede *wireless* poderá entrar na residência e após a senha ser inserida a primeira vez não

será mais necessário configurações para a utilização do serviço DLNA ali presente.

3.1. Servidor DLNA

O servidor DLNA será o ponto chave da aplicação. O servidor quando executado publicará na rede suas credenciais, ou seja, ele vai publicar na rede que esta disponível para conexões, publicará também dados sobre suas funcionalidades e a sua URN (*Uniform Resource Name*). O servidor DLNA ficará disponível na rede até que alguma requisição seja feita a ele. Quando isto acontecer, ele captura os dados e os envia para o adaptador serial. Após o recebimento da requisição, o adaptador serial repassa para o dispositivo os dados provenientes do servidor DLNA.

O servidor DLNA utiliza o serviço do UPnP para executar as tarefas. Ele utiliza a classe do serviço encapsulando os metadados da mesma, e faz uso de seus métodos para poder executar alguma função da qual seja necessária.

A implementação é simples, porém de muita importância. Ele implementa a interface *Runnable* para poder executar concorrentemente com outros processos.

Na figura 3 pode-se observar o método main do servidor onde é criado uma *Thread* para a execução poder ocorrer concorrentemente com outros processos.

Figura 3: Método main do Servidor

```
public class Server implements Runnable {

    public static void main(String[] args) throws Exception {
        // Start a user thread that runs the UPnP stack
        Thread serverThread = new Thread(new Server());
        serverThread.setDaemon(false);
        serverThread.start();
    }
```

(Fonte: Cling, 2014)

No método run() o serviço UPnP é instanciado através da interface UpnpServiceImpl(), e logo depois ativa o registro do dispositivo na rede após criar o dispositivo através do método createDevice.

```
upnpService.getRegistry().addDevice(createDevice());
```

Para criação de um dispositivo, o servidor utiliza o método "createDevice". Este método é o ponto chave para criação de um dispositivo, ele fornece a identidade, o tipo e os detalhes do dispositivo. O metadado de um dispositivo pode ser visto na figura **4.**

Figura 4: Método "createDevice" do Servidor

```
LocalDevice createDevice()
        throws ValidationException, LocalServiceBindingException, IOException {
    DeviceIdentity identity =
            new DeviceIdentity(UDN.uniqueSystemIdentifier("Arduino"));

    DeviceType type =
            new UDADeviceType("arduinoSensor", 1);

    DeviceDetails details =
            new DeviceDetails("SensorTemperatura",
                    new ManufacturerDetails("SensorAT"),
                    new ModelDetails("SensorAT"));

    LocalService<DlnaClass> SensorService =new AnnotationLocalServiceBinder().read(DlnaClass.class);
    SensorService.setManager( new DefaultServiceManager(SensorService, DlnaClass.class)
    );

    return new LocalDevice(identity, type, details, SensorService);


}
```

Em relação ao código mostrado na figura 4, pode-se perceber que todos os argumentos que compõe o dispositivo tem que ser passados através de construtores, pois esses dados precisam ser imutáveis.

Todo dispositivo precisa ter uma UDN (*Unique Device Name*), ou seja, identificação única para que ele possa ser identificado na rede. A UDN é criada pelo "DeviceIdentity" e precisa ser imutável, não podendo ser alterada quando o dispositivo é reiniciado.

A propriedade "*DeviceType*" descreve o tipo de dispositivo e sua versão de acordo com a padronização UDA (Arquitetura UPnp de Dispositivo).

Os serviços precisam ter nomes amigáveis que serão mostrados para os usuários da rede, esta é a função da propriedade "*DeviceDetails*".

O método, o "LocalService" é responsável pelo encapsulamento dos metadados do serviço, que consiste nas ações do serviço e suas variáveis.

Os metadados são disponibilizados por anotações na classe do serviço. As anotações UPnP são responsáveis não só por disponibilizar os metadados, mas também são elas que descrevem quais métodos serão expostos como ações UPnP (Cling, 2014).

3.2. Serviço DLNA

O servidor e o serviço disponibilizado trabalham juntos, se algum dos dois não existir não há comunicação DLNA na rede.

O serviço é uma das partes principais da aplicação, pois é através dele que o dispositivo heterogêneo vai receber as informações enviadas pela rede. O servidor manipula as anotações UPnP para acessar os métodos e os estados das variáveis.

Figura 5: Variáveis de estado UPnP do Serviço

```
@UpnpService(serviceId =
@UpnpServiceId("Arduino"),
serviceType =
@UpnpServiceType(value = "Arduino", version = 1))
@UpnpStateVariables({
    @UpnpStateVariable(
                        name = "Target",
                        defaultValue = "0",
                        sendEvents = false),
    @UpnpStateVariable(
                        name = "Msg",
                        defaultValue = "",
                        sendEvents = false)
})
```

A implementação do serviço começa com a definição das variáveis de estado, elas precisam ser declaradas com anotações UPnP para que o servidor possa acessá-las e modificá-las, caso uma variável não seja declarada no cabeçalho como variável de estado ela poderá ser acessada apenas pelos métodos internos da classe do serviço.

No exemplo mostrado acima, figura 5, pode-se ver a variável de estado "Target" e também a variável "Msg".

Figura 6: Método setTarget do Serviço

```
@UpnpAction
public void setTarget(@UpnpInputArgument(name = "NewTargetValue") boolean newPort) {
    port = newPort;
    System.out.println("O valor da variável eh: " + port);
}
```

Como mostrado na figura 6, à anotação "@UpnpAction" marca quais métodos serão expostos

como ações UPnP. Na mesma figura ainda pode-se observar a anotação "@UpnpInputArgument". Essa anotação é responsável por marcar a maneira como os dados serão passados para o método do lado cliente.

Para um cliente utilizar o método "setTarget", ele precisa apenas do nome do *InputArgument* para enviar os dados. A figura 7 mostra a maneira que um Cliente acessa os métodos UPnP do Servidor. O acesso é realizado através do nome do método descrito na anotação @UpnpInputArgument. Dessa forma, o Cliente apenas precisa fornecer o nome do método e o novo valor. Por meio das anotações o *framework Cling* se encarrega de identificar qual método será utilizado para inserir o novo valor no Servidor.

Figura 7: Cliente acessando o método "setTarget"

```
setInput("NewTargetValue", port);
```

3.3. Cliente

Cada dispositivo possuirá um cliente e um servidor. O servidor recebe as requisições dos clientes externos e as encaminha para a o dispositivo conectado a porta serial. Um cliente recebe os dados da porta serial através da classe adaptador e as envia para o *Service Broker*. O *Service Broker* é responsável por interpretar as informações enviadas pelo dispositivo e encaminhar para um servidor disponível na rede que esteja de acordo com as informações contidas na mensagem.

Tanto o cliente como o servidor implementam a interface *Runnable* para poder executar paralelamente com outros processos.

Figura 8: Método "run()" do Cliente.

```
public void run() {
    try {

        UpnpService upnpService = new UpnpServiceImpl();

        // Add a listener for device registration events
        upnpService.getRegistry().addListener(createRegistryListener(upnpService));

        // Broadcast a search message for all devices
        upnpService.getControlPoint().search(new STAllHeader());

    } catch (Exception ex) {
        System.err.println("Excessão ocorrida: " + ex);
        System.exit(1);
    }

}
```

O método run() do cliente, ilustrado na figura 8, adiciona uma escuta de registro através do "*upnpService.getRegistry*()". Cada vez que um dispositivo ingressar o cliente ficará sabendo qual dispositivo ingressou e qual serviços o mesmo disponibiliza.

O cliente também é capaz de fazer uma busca na rede por todos os dispositivos disponíveis especificando o nome amigável ou o tipo de dispositivo (propriedades que descrever o dispositivo e seus serviços). Caso o cliente possua a UDN (Unique Device Name) de um determinado dispositivo, ele pode buscar diretamente o dispositivo que ele deseja se comunicar não sendo necessário aguardar o anúncio periódico dos dispositivos.

Figura 9: Método "createRegistryListener"

```
RegistryListener createRegistryListener(final UpnpService upnpService) {
    return new DefaultRegistryListener() {
        ServiceId serviceId = new UDAServiceId(nameService);

        @Override
        public void remoteDeviceAdded(Registry registry, RemoteDevice device) {

            Service service;
            if ((service = device.findService(serviceId)) != null) {

                System.out.println("SERVICO DESCOBERTO: " + service.getServiceId());

                executeAction(upnpService, service);
            }
        }

        @Override
        public void remoteDeviceRemoved(Registry registry, RemoteDevice device) {
            Service service;
            if ((service = device.findService(serviceId)) != null) {
                System.out.println("SERVICO DESCONECTADO: " + service);
            }
        }
    };
}
```

Para o cliente poder se comunicar com um determinado dispositivo da rede, ele utiliza o método "*createRegistryListener*" (figura 9), que utiliza a UDN (*Unique Device Name*) do dispositivo para encontrá-lo e criar um registro do mesmo.

Como em um ambiente ubíquo os serviços são voláteis, ou seja, podem entrar e sair da rede. O cliente reconhece a saída do dispositivo através da escuta de dispositivos na rede e o remove da sua lista de dispositivos disponíveis.

Quando o dispositivo é removido corretamente da rede, desligado, o mesmo envia uma mensagem *broadcast* comunicando a saída da rede. Entretanto, caso isso não aconteça, o Cliente perceberá a saída do dispositivo somente quando ele parar de receber os anúncios periódicos do dispositivo. A remoção dos dispositivos que se desconectaram da rede acontece através do método "*remoteDeviceRemoved*()" mostrado na figura 9.

Caso o dispositivo pelo qual o cliente procura seja encontrado, uma mensagem de serviço descoberto é enviada e o Cliente passa utilizar o serviço.

Figura 10: Método "setTargetActionInvocation"

```
class SetTargetActionInvocation extends ActionInvocation {

    SetTargetActionInvocation(Service service) {
        super(service.getAction("SetTarget"));
        try {
            // Throws InvalidValueException if the value is of wrong type
            setInput("NewTargetValue", port);


        } catch (InvalidValueException ex) {
            System.err.println(ex.getMessage());
            System.exit(1);
        }

    }
}
```

A figura 10 demonstra como o Cliente utiliza os métodos UPnP dos dispositivos. O método "*setTargetActionInvocation*", é responsável por fazer a invocação da ação do serviço. Esta invocação é assíncrona, quando um Cliente executa o método *actionCallback*, ele aguarda a resposta do servidor. Ao executar o método *actionCallback*, o cliente aguardará a resposta do servidor para saber se a chamada foi bem sucedida ou foi falha.

No presente projeto, após fazer o envio dos dados, o cliente é interrompido e só é iniciado novamente quando algum dispositivo conectado a porta serial fizer uma requisição para algum dispositivo externo que esteja disponível na rede.

3.4. Service Broker

O *Service Broker* é um servidor central que receberá informações de todos os clientes e as interpretará. De acordo com as informações contidas nas mensagens, o *ServiceBroker* inicia um cliente especifico a cada requisição para enviar as informações do dispositivo para um servidor disponível na rede que possa utilizar as informações da mensagem.

O projeto atual poderia funcionar perfeitamente sem a necessidade de um *ServiceBroker*, fazendo uma comunicação *peer to peer* entre os dispositivos da rede. Mas como a rede pode possuir diversos tipos de dispositivos com tecnologias diferentes optou-se por fazer um *ServiceBroker* que pode ser útil em projetos futuros para a integração de diferentes tecnologias, como o *WiFi-Direct*, Miracast ou DPWS(HOFFMAN, 2013). Também pode-se implementar controle de concorrência, geração de relatórios de Log entre outras propostas de novos projetos que podem dar sequência ao presente projeto.

3.5. Adaptador serial

Para podermos interpretar as informações recebidas da porta serial, e também poder enviar informações pela porta foi necessária à construção de uma classe adaptadora.

Esta classe possui uma *thread* que “ouve” e captura todos os dados vindos da porta serial para poderem ser utilizados pelo cliente na comunicação com o *ServiceBroker* ou com qualquer outro servidor encontrado na rede.

Figura 11: Método ouvinte da porta Serial.

```
public synchronized void serialEvent(SerialPortEvent ev) {
    if (ev.getEventType() == SerialPortEvent.DATA_AVAILABLE) {
        try {
            String inputLine = input.readLine();
            System.out.println(inputLine);
            // Envia a mensagem do arduino para um servidor
            Client cliente = new Client("ServiceBroker",input.readLine());

        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }
}
```

O método "*serialEvent*" captura os dados da porta serial e inicia um cliente para comunicar-se com o *ServiceBroker*. Como o cliente implementa a interface Runnable não haverá problema se o método *serialEvent* criar vários clientes.

4. Proposta

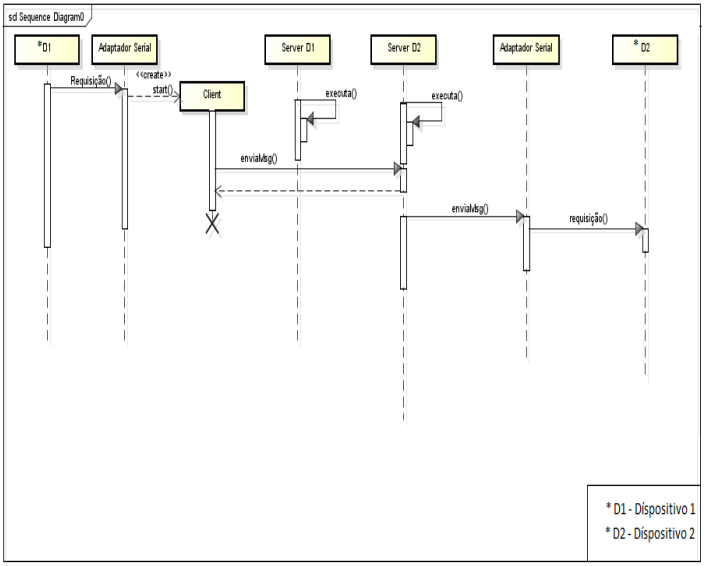
A proposta é constituída por duas formas de funcionamento. A primeira forma é com a utilização de um *ServiceBroker* que será responsável por interpretar e encaminhar as mensagens para o dispositivo correto, outra maneira é a comunicação simples, onde cada dispositivo possui um Cliente e um Servidor para comunicação.

Na abordagem simples quando um dispositivo heterogêneo precisar utilizar algum serviço da rede, a classe adaptadora iniciará um serviço Cliente e a comunicação ocorrerá diretamente com o dispositivo responsável pelo serviço. Essa abordagem é mais rápida em relação à comunicação utilizando o *ServiceBroker*, mas ela pode sobrecarregar um dispositivo caso sejam feitas muitas requisições.

Caso haja uma sobrecarga de requisições em um dispositivo, algumas dessas podem deixar de ser executadas, ou ainda podem ser executadas de forma desordenada por causa do escalonador do sistema, que neste caso, pode ser dada prioridade a uma requisição mais nova. Caso uma requisição dependa da resposta de uma requisição anterior, a execução da aplicação pode ser prejudicada.

Para o perfeito entendimento do funcionamento da arquitetura foram desenvolvidos diagramas de sequência que procuram ilustrar como ocorre o envio e o recebimento dos dados na rede, os seguintes diagramas também podem ser encontrados nos anexos para uma melhor visualização.

Figura 12: Comunicação unidirecional sem SB

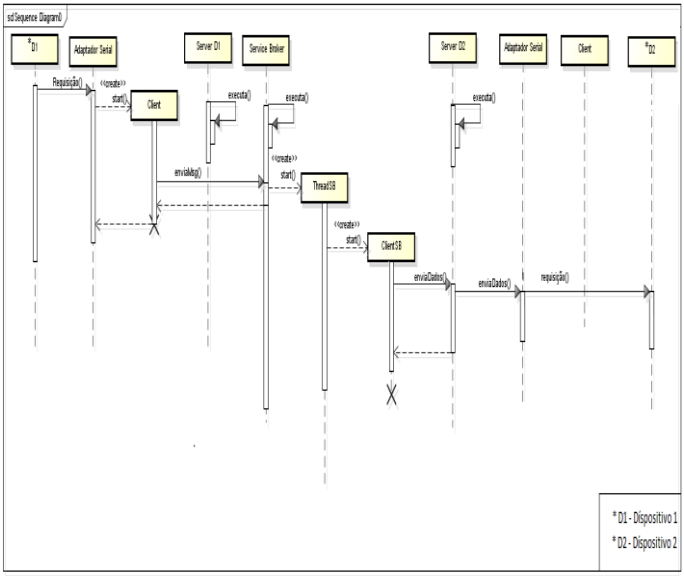


No diagrama de sequência da figura 12 pode-se observar a ilustração de uma comunicação simples sem a utilização do *ServiceBroker* .

O “dispositivo 1” precisa enviar os dados adquiridos para o “dispositivo 2”. Para conseguir realizar esta tarefa ele

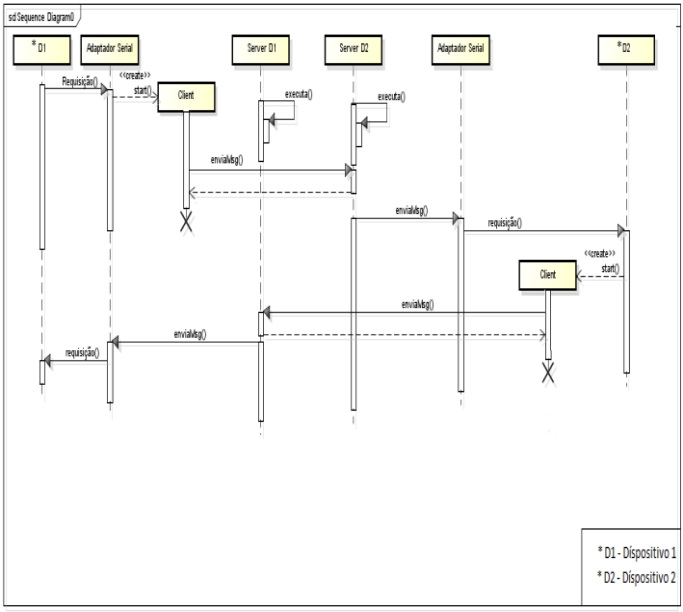
se comunica com a classe adaptadora que captura os dados da porta serial e cria um cliente que será capaz de se comunicar com o servidor do “dispositivo 2”, após enviar os dados o cliente automaticamente é fechado depois de receber a confirmação que o servidor recebeu os dados. Já em posse dos dados o servidor faz a comunicação com a classe adaptadora do “dispositivo 2” que faz o envio dos dados pela porta serial. Caso na rede existam mais de um servidor que esteja disponibilizando o mesmo serviço os dados serão enviados para ambos.

Figura 13: Comunicação unidirecional com SB



A figura 13 representa a mesma comunicação citada anteriormente, mas com um *ServiceBroker* para direcionar as mensagens. Os dispositivos da rede conhecem apenas o *ServiceBroker*, não sendo possível a comunicação peer to peer. Entretanto, o *ServiceBroker* é capaz de utilizar qualquer serviço disponível na rede, mesmo se os dispositivos não utilizarem a mesma tecnologia.

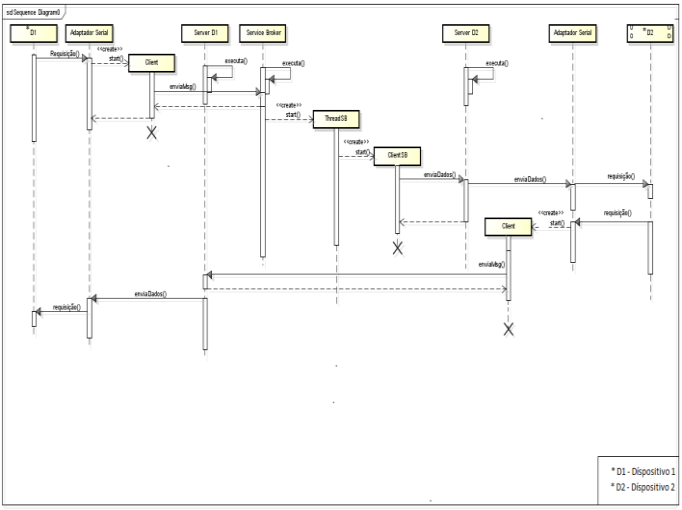
Figura 14: Comunicação bidirecional sem SB



A comunicação bidirecional, ilustrada na figura 14, pode ser utilizada para comunicação entre sensores e atuadores. O “dispositivo 1” ativa o “dispositivo 2” através de uma requisição. Quando o “dispositivo 2” aceita a requisição, este retorna a resposta da requisição ao “dispositivo 1”.

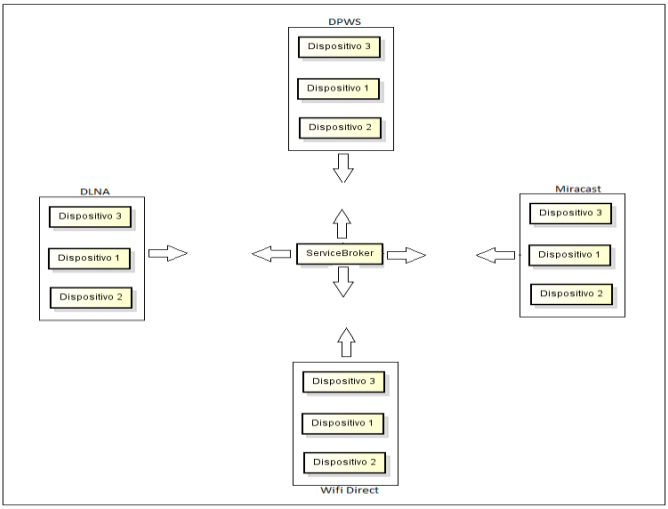
Além disso, o “dispositivo 2” pode também enviar uma chamada para qualquer outro dispositivo que esteja na rede. Com esta comunicação pode-se fazer facilmente a automação de uma residência apenas com a utilização de alguns sensores e atuadores ligados á uma infraestrutura de rede. Outro fator importante na comunicação DLNA é a segurança, pois os dispositivos para se comunicarem precisam estar na mesma rede doméstica, ou seja, nenhum usuário externo a rede poderá utilizar os serviços ali disponibilizados.

Figura 15: Comunicação bidirecional com SB



Na figura 15, pode-se observar uma comunicação bidirecional com a utilização de um *ServiceBroker*. Esta implementação fornece muito mais possibilidades relacionada às comunicações anteriores, pois com um *ServiceBroker* a gama de dispositivos pode ser bem maior sem a sobrecarga dos dispositivos da rede.

Figura 16: Integração de tecnologias distintas através do ServiceBroker.



É ilustrado na figura 16 a utilização do *ServiceBroker*. O *ServiceBroker* pode ser crucial para comunicação entre protocolos diferentes. Cada conjunto de dispositivos pode estar associado a um protocolo diferente. Sendo assim, a comunicação entre eles não seria trivial. O papel do *ServiceBroker*, neste caso, seria realizar a tradução entre os protocolos. Dessa forma, pode-se aumentar a gama de serviços disponíveis para os diferentes dispositivos possibilitando a comunicação entre os mesmos.

Enfim, existem várias possibilidades de comunicação via tecnologia DLNA basta analisar qual se encaixa melhor com a necessidade de utilização. No presente projeto, foi desenvolvido um *ServiceBroker* simples sem comunicação com diferentes protocolos, deixando a possibilidade de continuação para futuros projetos.

5. Avaliação

Para a avaliação da aplicação foi necessário o desenvolvimento de uma classe responsável por armazenar o tempo e a identificação dos clientes que fizeram requisições para o servidor.

Assim como o servidor, o cliente, e o adaptador, a classe de experimento foi desenvolvida em JAVA, diferenciando-se apenas pelo fato do cliente e o servidor serem baseados no *framework Cling* e o adaptador utilizar a biblioteca RxTx.

Pela dificuldade em se obter vários dispositivos reais, foram utilizado clientes virtuais (*threads*), com intuito de mensurar a confiabilidade da tecnologia e da proposta em suportar várias requisições simultâneas.

5.1 Implementação do experimento

Para a implementação dos experimentos foram utilizados:

Um Notebook: Marca Itautec, processador duocore P6200 2.13GHz, memória de 3 GB, disco rígido de 320 GB e sistema operacional Windows 8.1.

Um PC desktop: Marca Qbex, processador core i5 3330 3.00GHz, memória de 4 Gb, disco rígido de 500 GB e sistema operacional Windows 8.1.

Um roteador wireless:Marca Tp-Link, com 4 portas RJ45, velocidade de 150Mbps, com padrão Ethernet, 802.11n/g/b, e uma antena de 5Dbi;

Um Arduíno: Marca Arduíno, modelo Uno, microcontrolador ATmega328, 14 pinos digitais de entrada e saída, 6 pinos analógicos de entrada e saída, memória de 32 Kb e velocidade de *clock* 16 MHz.

5.2 Resultados

A aplicação DLNA desenvolvida pode ser utilizada para várias finalidades. Contudo, é necessário mensurar a confiabilidade da tecnologia para aplicação em automação residencial.

O grau de confiabilidade foi determinado por duas métricas: i) tempo de resposta entre a requisição de um cliente e a resposta de um servidor e ii) capacidade de responder às requisições.

Todos os testes aqui descritos foram realizados 5 (cinco) vezes para a obtenção de dados mais confiáveis e precisos.

As medições foram realizadas em duas partes, a primeira com teste de requisições unidirecional (testes de 1 a 4) e a segunda com requisições bidirecionais (testes de 5 a 9).

O primeiro foi realizado com uma sequência de 50 clientes virtuais fazendo requisições para um servidor remoto de forma unidirecional.

Na forma unidirecional somente foram enviados os dados do cliente para o servidor remoto sem a necessidade de uma resposta do dispositivo conectado a porta serial. Sabe-se apenas que o servidor recebeu a requisição e a enviou para o dispositivo, o que aconteceu após o envio para o dispositivo não foi considerado no teste unidirecional e será abordado mais adiante no teste bidirecional.

Apenas foram contabilizados o tempo que o cliente levou para descobrir o serviço na rede, e o tempo necessário para o servidor executar o serviço. Perdas de envio de requisições pela porta serial e *delay* na comunicação da classe adaptadora não foram contabilizados.

As requisições foram composta de uma *String* contendo o nome do cliente e número sequencial gerado no momento da criação da *Thread* (Cliente).

Tabela 1: Teste unidirecional com 50 clientes.

Teste	*QCA	Média Descoberta (ms)	Desvio Padrão (%)	Média Execução (ms)	Desvio Padrão (%)
1	50	2504	38,18%	2556	37,83%
2	50	2358	40,03%	2415	39,38%
3	50	2076	35,93%	2127	35,40%
4	50	2068	41,88%	2119	40,59%
5	50	2399	36,10%	2449	35,20%
Total	**50**	**2281**	**38,40%**	**2333**	**37,68%**

*QCA- Quantidade de Clientes Atendidos

Observando a tabela 1, pode-se notar que 100% das requisições foram atendidas (cinquenta requisições). Além disso, o tempo médio de descoberta foi de 2281 milissegundos e o desvio padrão médio foi de 38,40% ou seja, 876 milissegundos. Também foi mensurado o tempo médio de execução que retornou um tempo de 2333 milissegundos com um desvio padrão médio de 879 totalizando um desvio de 37,68%.

A tabela 2 apresenta o resultado dos testes com 100 (cem) clientes virtuais fazendo requisições ao servidor remoto.

Tabela 2: Teste unidirecional com 100 clientes.

Teste	*QCA	Média Descoberta (ms)	Desvio Padrão (%)	Média Execução (ms)	Desvio Padrão (%)
1	100	3371	40,84%	3452	41,04%
2	100	3516	37,82%	3595	37,52%
3	100	3322	38,62%	3371	38,27%
4	100	3363	39,83%	3412	39,24%
5	100	3402	39,83%	3458	39,55%
Total	**100**	**3395**	**39,38%**	**3457**	**39,12%**

*QCA- Quantidade de Clientes Atendidos

Pode-se observar na tabela 2 que 100% das requisições foram atendidas em todas as repetições.

O tempo de descoberta de serviço alcançou uma média de 3395 milissegundos com um desvio padrão de 1337 (39,38%), e o tempo de execução totalizou uma média de 3457 milissegundos com um desvio padrão de 1352 (39,12%).

Após a realização do teste com 100 (cem) requisições, o servidor foi testado novamente com 150 (cento e cinquenta) requisições a fim de aferir a capacidade de resposta sem a perda de dados.

Tabela 3: Teste unidirecional com 150 clientes.

Teste	*QCA	Média Descoberta (ms)	Desvio Padrão (%)	Média Execução (ms)	Desvio Padrão (%)
1	150	4243	45,64%	4326	45%
2	150	4594	43,87%	4701	44%
3	150	4137	47,46%	4228	47%
4	150	4384	45,61%	4477	45%
5	150	4353	43,25%	4433	44%
Total	**150**	**4342**	**45,13%**	**4433**	**45%**

*QCA- Quantidade de Clientes Atendidos

Neste novo teste 100% das requisições foram atendidas. É apresentado na tabela 3 o tempo médio de média de resposta para descoberta, que foi de 4342 milissegundos e um desvio padrão de 1959 (45,13%). O tempo médio de execução que ficou em 4433 milissegundos com um desvio padrão de 1995, ou seja, 45%.

Devido a uma limitação de hardware o teste só pode ser realizado com 200 (duzentas) requisições simultâneas, pois após essa quantidade houve uma falha devido ao auto consumo de memória RAM.

Tabela 4: Teste unidirecional com 200 clientes.

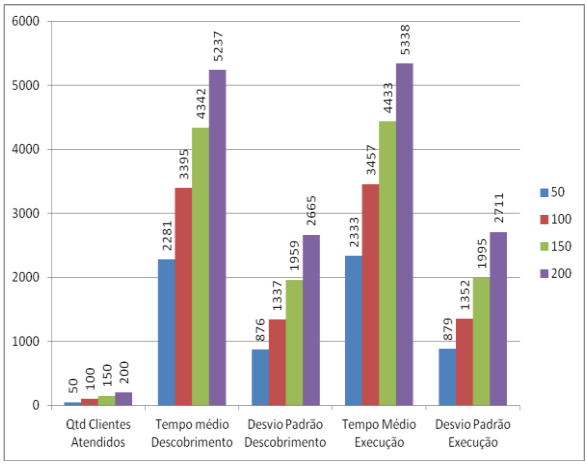
Teste	*QCA	Média Descoberta (ms)	Desvio Padrão (%)	Média Execução (ms)	Desvio Padrão (%)
1	200	5184	46%	5212	45%
2	200	5223	48%	5329	48%
3	200	5206	48%	5336	48%
4	200	5163	48%	5295	48%
5	200	5410	64%	5520	63%
Total	**200**	**5237**	**51%**	**5338**	**51%**

*QCA- Quantidade de Clientes Atendidos

É demonstrado na tabela 4, o último teste feito com uma comunicação unidirecional. Constatou-se, que mesmo com várias requisições simultâneas o

servidor remoto conseguiu atender 100% das requisições efetuadas. O tempo médio de descoberta foi de 5237 milissegundos, com um desvio padrão de 2265 (51%). Ademais, o tempo de execução ficou em 5338 milissegundos com um desvio padrão de 2711 (51%).

Gráfico 1: Testes unidirecionais



No gráfico 1 pode-se analisar a evolução dos testes, onde se teve um aumento gradativo no tempo de descoberta e no tempo de execução. O desvio padrão também aumentou, pois quanto mais requisições eram feitas mais tempo elas aguardavam na fila de execução do escalonador de processos do sistema. Como as requisições foram realizadas por clientes virtuais em forma de *Threads* elas ganhavam o processador de forma aleatória, o que acarretou na desordenação de execuções e por consequência na desordenação do atendimento dos pedidos.

Foram realizados também alguns experimentos utilizando uma comunicação bidirecional. Nesta comunicação, os clientes virtuais enviam dados

para um servidor remoto e os repassa para um Arduíno conectado a porta serial do dispositivo. O Arduíno processa os dados e os retorna para o dispositivo de origem.

O primeiro teste se realizou com uma sequência de 50 clientes virtuais realizando requisições para um servidor remoto.

Tabela 5: Teste bidirecional com 50 clientes.

Teste	*QCA	Média Descoberta (ms)	Desvio Padrão (%)	Média Execução (ms)	Desvio Padrão (%)
1	46	4832	27,51%	5549	24,44%
2	49	4588	23,85%	5413	20,60%
3	48	4679	26,00%	5423	23,07%
4	48	4823	24,26%	5562	21,77%
5	47	4814	22,59%	5561	19,84%
Total	**48**	**4747**	**24,85%**	**5502**	**21,94%**

*QCA- Quantidade de Clientes Atendidos

A tabela 5 demonstra que após o envio de 50 requisições realizadas pelos clientes virtuais, o Arduíno ligado ao servidor remoto respondeu apenas 96% das requisições efetuadas. O tempo médio de descoberta foi de 4747 milissegundos, com um desvio padrão de 1179 (24,85%). O tempo de execução foi de 5502 milissegundos com um desvio padrão de 1207 (21,94%).

Depois de constatado que o Arduíno não processou 100% das requisições foram realizados testes com 40 (quarenta) clientes virtuais.

Tabela 6: Teste bidirecional com 40 Clientes.

Teste	*QCA	Média Descoberta (ms)	Desvio Padrão (%)	Média Execução (ms)	Desvio Padrão (%)
1	38	4694	26,82%	5474	23,21%
2	36	4295	23,64%	5010	20,37%
3	36	4888	18,14%	5706	15,05%
4	35	4458	30,81%	5167	27,05%
5	39	4427	27,39%	5238	22,41%
Total	**37**	**4553**	**25,25%**	**5319**	**21,51%**

*QCA- Quantidade de Clientes Atendidos

Os resultados dos testes com 40 (quarenta) clientes virtuais estão ilustrados na tabela 6. Pode-se constatar que o Arduíno respondeu apenas 92,5% das requisições.

Os resultados ainda apresentam um tempo de descoberta médio de 4553 milissegundos com um desvio padrão de 1149 (25,25%), e um tempo de execução de 5319 milissegundos com um desvio padrão de 1144 (21,51%).

Como o Arduíno ainda não estava atendendo um numero satisfatório de requisições foi reduzido mais uma vez o número de clientes virtuais para 30.

Tabela 7: Teste bidirecional com 30 Clientes.

Teste	*QCA	Média Descoberta (ms)	Desvio Padrão (%)	Média Execução (ms)	Desvio Padrão (%)
1	25	4868	21,05%	5542	18,79%
2	28	4620	31,64%	5370	26,39%
3	28	4846	19,44%	5655	18,12%
4	27	4632	28,72%	5507	25,03%
5	29	6529	27,22%	7581	24,99%
Total	**27**	**5099**	**25,64%**	**5931**	**22,78%**

*QCA- Quantidade de Clientes Atendidos

É demonstrado na tabela 7 que no teste realizado com 30 (trinta) clientes, Arduíno não foi capaz de responder 100% dos clientes, chegando a um total de 90% das requisições atendidas. O tempo médio de descoberta para este teste foi de 5099 milissegundos com um desvio padrão de 1307 (25,64%). O tempo médio de execução foi de 5931 milissegundos com um desvio padrão de 1351 (22,78%).

Tabela 8: Teste bidirecional com 20 Clientes.

Teste	*QCA	Média Descoberta (ms)	Desvio Padrão (%)	Média Execução (ms)	Desvio Padrão (%)
1	16	4818	31,88%	5564	28,08%
2	17	4353	23,97%	5117	19,09%
3	18	3732	29,78%	4473	24,84%
4	18	4003	32,08%	4764	28,71%
5	17	4460	24,28%	5269	20,61%
Total	**17**	**4273**	**28,35%**	**5037**	**24,23%**

*QCA- Quantidade de Clientes Atendidos

Ainda a fim de aferir a confiabilidade do projeto desenvolvido foi efetuado um teste com 20 clientes, como é mostrado na tabela 8.

Sendo submetido a 20 requisições simultâneas o Arduíno respondeu com sucesso a 17 requisições, ou seja, 85% tendo um percentual de perda de 15%. Os tempos de descoberta e de execução tiveram uma média de 4273 milissegundos com desvio padrão de 1212 (28,35%), e 5037 milissegundos com um desvio padrão de 1221 (24,23%), respectivamente.

Nos testes com 10 Clientes virtuais o Arduíno respondeu a 80% das requisições. Sendo que o tempo médio de descoberta foi de 4476 milissegundos com desvio padrão de 1355 (30,28%), e o tempo de execução de 5218 milissegundos com um desvio padrão de 1417 (27,16%), como é mostrado na tabela 9.

Tabela 9: Teste bidirecional com 10 Clientes.

Teste	*QCA	Média Descoberta (ms)	Desvio Padrão (%)	Média Execução (ms)	Desvio Padrão (%)
1	7	4533	18,24%	5263	16,76%
2	9	5538	32,64%	6351	30,56%
3	8	4490	26,79%	5286	22,91%
4	8	4192	26,45%	4842	24,49%
5	9	3628	50,48%	4348	42,94%
Total	**8**	**4476**	**30,28%**	**5218**	**27,16%**

*QCA- Quantidade de Clientes Atendidos

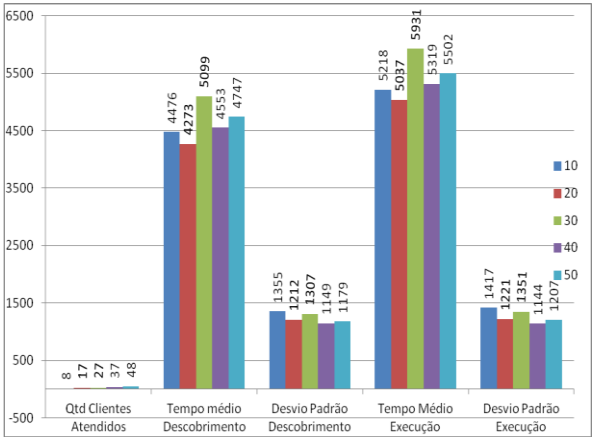
No gráfico 2 pode-se observar mais detalhadamente a evolução dos testes.

Os testes bidirecionais foram bem mais desuniformes que os testes unidirecionais, pois envolvem dois dispositivos diferentes com poder de processamento diferente elevando os tempos de descoberta e execução.

Em relação à perda de informações os testes bidirecionais obtiveram em torno de 88,7% de sucesso nas requisições devido ao fato de o dispositivo Arduíno necessitar de duas a três requisições para poder iniciar a comunicação. Fato que necessita de um estudo mais aprofundado da comunicação serial com o Arduíno para poder chegar a níveis satisfatórios de resposta.

Então considerando a comunicação com o servidor obtivemos 100% de aproveitamento das requisições em ambos os testes, mas devido à necessidade de pelo menos duas requisições para o Arduíno começar a responder aos clientes que originaram a mensagem houve uma pequena perda de desempenho relacionada à quantidade de requisições atendidas.

Gráfico 2: Testes bidirecionais



Apesar do desvio padrão alto, tanto para a descoberta e quanto para a execução, o servidor conseguiu responder a todas as requisições com até 200 clientes nos testes unidirecionais, e nos testes bidirecionais se obteve até 88,7% de atendimento das requisições, número que pode ser elevado com o devido estudo da comunicação serial com o Arduíno.

Os resultados de desvio padrão podem ser explicados pelo alto tráfego gerado e pela falta de controle na ordenação das requisições. A falta de controle na ordenação das requisições nos aponta para a oportunidade de construção de um mecanismo de escalonamento de requisições na camada da aplicação.

6 **Considerações Finais**

O presente projeto propôs uma arquitetura distribuída que permita fazer a interoperabilidade entre dispositivos heterogêneos por meio da tecnologia DLNA.

Com base nesta proposição foram realizadas pesquisas a fim de encontrar maneiras de se realizar a interoperabilidade. Depois de um estudo inicial, optou-se pelo *framework open source Cling.*

Após o desenvolvimento da aplicação foram elaborados testes com o propósito de mensurar o desempenho da mesma, fazendo uma correlação com testes unidirecionais e bidirecionais. Os resultados foram satisfatórios em relação ao propósito em que se trata o presente projeto, porém a arquitetura ainda precisa de melhorias na comunicação bidirecional, necessitando um estudo mais aprofundado da comunicação serial.

Ao longo do trabalho surgiram novas possibilidades de desenvolvimento que não foram abordadas pois tornariam o projeto muito extenso. Dentre as novas

possibilidade, está a necessidade do desenvolvimento mais aprofundado do *ServiceBroker*, um serviço do *middleware* responsável por gerenciar as requisições, fazer integração com diferentes tecnologias e ainda fazer o controle de acesso ao dispositivo impedindo a postergação indefinida.

Por fim, o uso da tecnologia DLNA ainda é muito recente não possuindo muitos estudos na área, ficando limitada a dispositivos com certificação DLNA. Com a ajuda do presente projeto, dispositivos que até então não possuíam a tecnologia poderão se comunicar utilizando o DLNA aumentando a gama de dispositivos que podem ser utilizados para uma automação residencial ou industrial.

REFERÊNCIAS

CHAN, Marie, et al. **"A review of smart homes—Present state and future challenges."** Computer methods and programs in biomedicine 91.1 (2008): 55-81.

HAGEDORN, Sebastian. Interchangeability and Interconnectivity of Current Home Media Streaming Technologies. 2012. Disponível em: <http://www.hagi-dd.de/docs/StreamingTechnologies.pdf.> Acesso em: 15 out. 2014.

NIRMALYA, Roy; Brooks, K.; Julien, C., **"Usability of Semantic Web for Enhancing Digital Living Experience,"** Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE , vol., no., pp.1,6, 9-12 Jan. 2010.

TALARIAN. "**Everything You Need To Know About Middleware",** Los Altos: Talarian Corporate Headquarters. 2000.

SOUZA, Anderson. R. de; et al. **"A placa Arduíno: uma opção de baixo custo para experiências de física assistidas pelo PC"**, Revista Brasileira de Ensino de Física, São Paulo, v. 33, n. 1, 1702, 2011. Disponível em: < http://www.sbfisica.org.br/rbef/pdf/331702.pdf>. Acesso em: 26 set. 2014.

ARDUÍNO. **"Arduíno Uno",** Disponível em: <http://arduino.cc/en/Main/ArduinoBoardUno>. Acesso em: 26 set. 2014.

FRANCO, Carolina. "**O que é e como funciona DLNA".** Disponível em: <http://adrenaline.uol.com.br/tecnologia/artigos/225/o-que-e-e-como-funciona-dlna.html>. Acesso em: 21 set. 2014.

UPnP, Fórum. **“What is UPnP”.** Disponível em: < http://upnp.org/about/what-is-upnp/>. Acesso em: 25 set. 2014.

WADDINGTON, Nathan; TAYLOR**, Russell. “Arduíno & Open Source Design”.** 2007. Disponível em: <http://anri.go.id/assets/doc/press_release/arduino-and-open-source-design.pdf.> Acesso em: 15 out 2014.

Cling. **“Java/Android UPnP library and tools”.** Disponível em: <http://4thline.org/projects/cling/> Acessado em: 26 jul. 2014.

PALMA, Diana Sobreiro da Costa et al. **FEUP KNX: domótica KNX/EIB de baixo custo.** 2012. Disponível em: <http://repositorio-aberto.up.pt/bitstream/10216/59047/2/Texto%20integral.pdf>. Acesso em: 01 out 2014.

DLNA. **DLNA Device Classes.** Disponível em: <http://www.dlna.org/dlna-for-industry/certification/dlna-device-classes/digital-media-Servidor>. Acesso em: 25 set. 2014

SISLITE. **O que é a domótica?** Disponível em: <http://www.sislite.pt/domus.htm.>. Acesso em: 05 out. 2014.

VENKITARAMAN, Narayanan. **Wide-area media sharing with UPnP/DLNA.** In: Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE. IEEE, 2008. p. 294-298.

HOFFMAN, Chiris. “**Wireless Display Standards Explained: AirPlay, Miracast, WiDi, Chromecast, and DLNA”.** Disponível em : <http://www.howtogeek.com/177145/wireless-display-standards-explained-airplay-miracast-widi-chromecast-and-dlna/ >. Acesso em: 26 de Nov. 2014.

ANEXOS

Figura 13: Comunicação unidirecional sem *serviceBroker.*

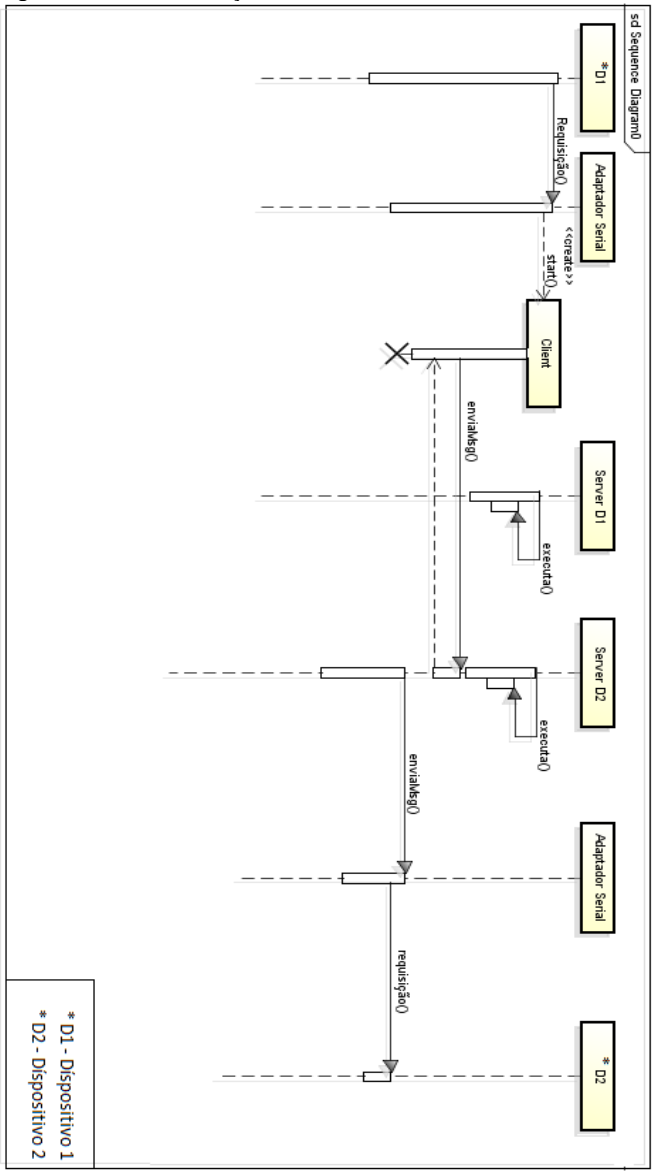


Figura 14: Comunicação unidirecional com *serviceBroker.*

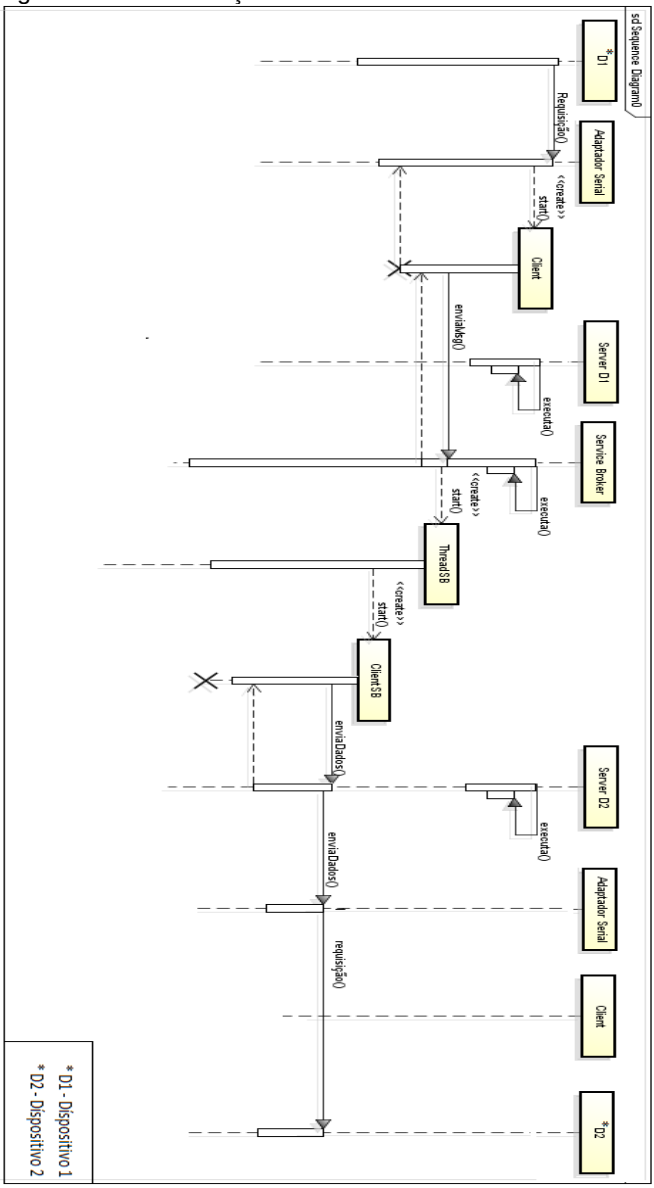


Figura 15: Comunicação bidirecional sem *serviceBroker.*

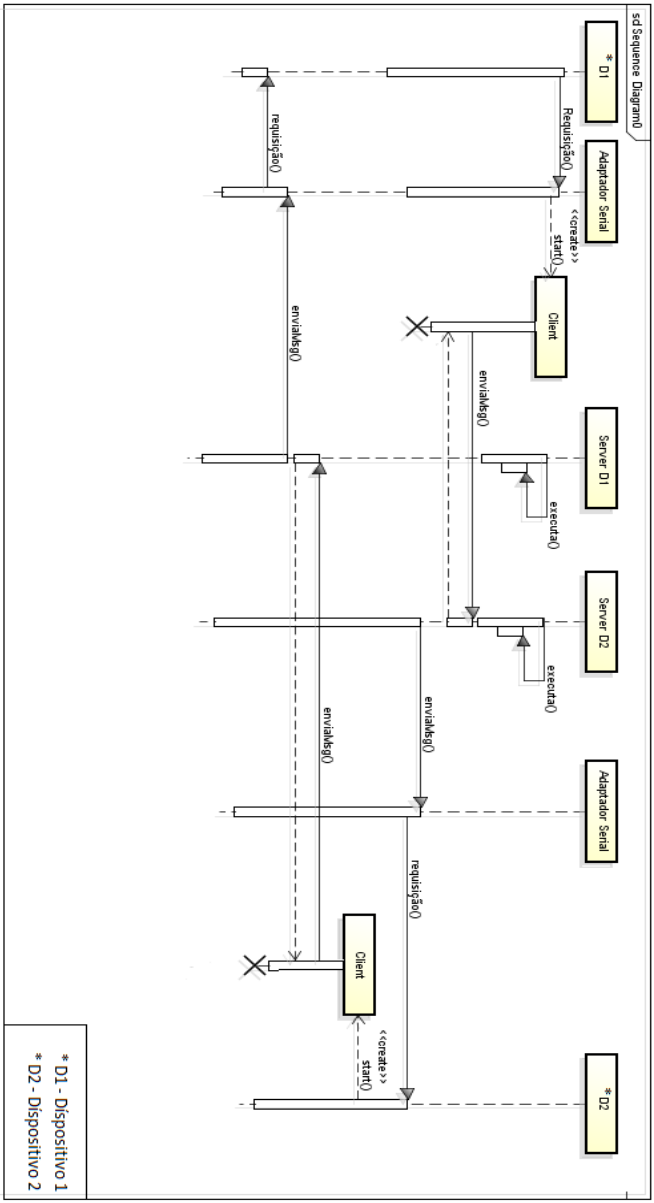


Figura 16: Comunicação bidirecional com *serviceBroker.*

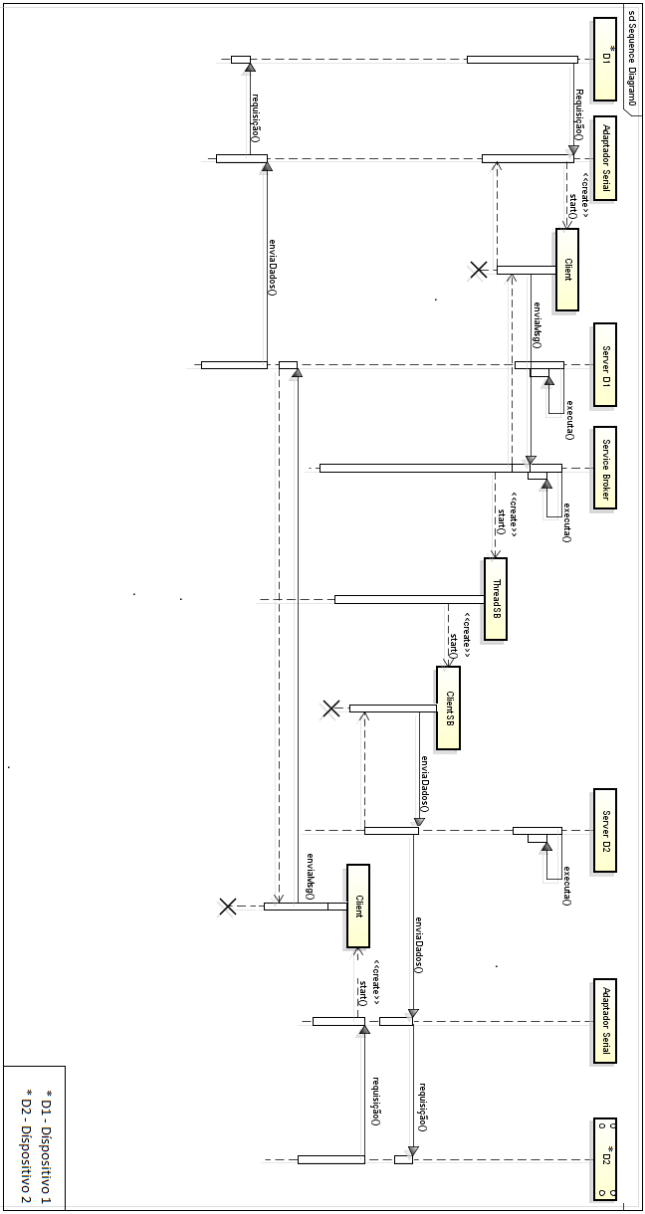


Figura 17: Integração de tecnologias distintas através do *serviceBroker.*