Marcos Henrique de Morais Golinelli

ARQUITETURA DE DISPOSITIVOS INTELIGENTES APLICADA EM LABORATÓRIOS DE EXPERIMENTAÇÃO REMOTA

Dissertação submetida ao Programa de Pós-Graduação em Tecnologias da Informação e Comunicação da Universidade Federal de Santa Catarina para a obtenção do Grau de Mestre em Tecnologias da Informação e Comunicação.
Orientadora: Prof.ª Olga Yevseyeva, Dr.ª
Coorientador: Prof. Juarez Bento da Silva, Dr.

Araranguá

2018

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Golinelli, Marcos Henrique de Morais Golinelli
Arquitetura de dispositivos inteligentes aplicada em laboratórios remotos / Marcos Henrique de Morais Golinelli Golinelli ; orientador, Olga Yevseyeva Yevseyeva, coorientador, Juarez Bento da Silva Silva, 2018.
149 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Campus Araranguá, Programa de Pós Graduação em Tecnologias da Informação e Comunicação, Araranguá, 2018.

Inclui referências.

1. Tecnologias da Informação e Comunicação. 2. Padronização de Laboratórios Remotos. 3. Arquitetura de Dispositivos Inteligentes. 4. Laboratórios Remotos. I. Yevseyeva, Olga Yevseyeva. II. Silva, Juarez Bento da Silva. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Tecnologias da Informação e Comunicação. IV. Título.

Marcos Henrique de Morais Golinelli

ARQUITETURA DE DISPOSITIVOS INTELIGENTES APLICADA EM LABORATÓRIOS DE EXPERIMENTAÇÃO REMOTA

Esta Dissertação foi julgada adequada para a obtenção do Título de "Mestre em Tecnologias da Informação e Comunicação", e aprovada em sua forma final pelo Programa de Pós-Graduação em Tecnologias da Informação e Comunicação.

Araranguá, 19 de setembro de 2018.

_____
Prof.ª Andréa Cristina Trierweiller, Dr.ª
Coordenadora do Curso

_____
Prof. ª Olga Yevseyeva, Dr. ª
Orientadora
Universidade Federal de Santa Catarina

Banca Examinadora:

_____
Prof. Juarez Bento da Silva, Dr.
Coorientador
Universidade Federal de Santa Catarina

_____
Prof. Hélio Aisenberg Ferenhof, Dr.
Universidade Federal de Santa Catarina

_____
Prof. Vanderlei Freitas Junior, Dr.
Instituto Federal Catarinense

_____
Prof. Paulo Manoel Mafra, Dr.
Universidade Federal de Santa Catarina

Dedico este trabalho aos meus pais, que com o esforço, educação e carinho me incentivaram e ajudaram a chegar até aqui.
A minha amada filha Amanda.

AGRADECIMENTOS

Em primeiro lugar agradeço a Deus, que sempre esteve comigo nas horas fáceis e difíceis da minha vida.

Agradeço a orientadora Prof.ª Dr.ª Olga Yevseyeva, ao coorientador Prof. Dr. Juarez Bento da Silva, que com paciência e sabedoria, abriu minha mente para enxergar os desafios do contexto educacional, como também pelo acompanhamento pontual e competente no desenvolvimento deste trabalho.

Aos professores do Programa de Pós-Graduação em Tecnologias da Informação e Comunicação pela colaboração efetiva.

À prof.ª Msc. Valdirene da Rosa Rocho, pelo incentivo ao desenvolvimento deste trabalho.

A todos os meus amigos e amigas, que estiveram presentes em minha vida durante minha formação, e todos que diretamente ou indiretamente contribuíram para a realização deste trabalho.

Aos membros da banca avaliadora, pela contribuição de suas considerações.

Os nossos pais amam-nos porque somos seus filhos, é um fato inalterável. Nos momentos de sucesso, isso pode parecer irrelevante, mas nas ocasiões de fracasso, oferecem um consolo e uma segurança que não se encontram em qualquer outro lugar.

Bertrand Russell

RESUMO

A experimentação dos conhecimentos teóricos em laboratórios tende a melhorar o processo de ensino-aprendizagem, no entanto, para suprir a falta destes, surgem alternativas como os laboratórios virtuais e remotos. Destaca-se que na manipulação dos laboratórios remotos, o resultado obtido em cada experiência é real, sendo influenciado por variáveis do ambiente. Os benefícios de seu uso vão além do pedagógico, como a disponibilidade de acesso pela Internet, compartilhamento de infraestrutura e possível economia de recursos financeiros. De modo geral, tais laboratórios são desenvolvidos de maneira fortemente acoplada e sem padronização, para atender necessidades específicas. Esta falta de padrão dificulta seu compartilhamento, diminuindo a possibilidade de ampliação dos benefícios gerados por seu uso, partindo desta dificuldade, este trabalho propõe a utilização da arquitetura de dispositivos inteligentes como alternativa a esta lacuna, desenvolvendo um protótipo para comunicação de laboratório remoto, para que possa ser utilizado como referencial no desenvolvimento de novos experimentos ou na readequação de existentes, visando a padronização dos experimentos. Este trabalho conta com uma revisão da literatura a cerca da arquitetura de dispositivos inteligentes aplicada em laboratórios e o desenvolvimento de protótipos. Utilizou-se o plano inclinado do laboratório RExLab como ponto de partida, pois possui sensores e atuadores manipulados pelos usuários, possibilitando uma implementação mais ampla. As tecnologias computacionais utilizadas foram o Node.js no lado do servidor, o WebSockets como meio de comunicação, por sua característica assíncrona, e o formato JSON como padrão para intercâmbio de dados, já os clientes, para plataforma Web em linguagem HTML e JavaScript, nas linguagens C# e Java para desktop. Os protótipos mostraram-se funcionais, não interferindo nos requisitos de funcionamento do experimento. A adoção deste modelo como padrão mostrou-se viável, pois é independente de plataforma de hardware e linguagem de programação, possuindo ferramentas compatíveis para as plataformas mais utilizadas no desenvolvimento de experimentos remotos.

Palavras-chave: Padronização de Laboratórios Remotos, Laboratórios Online, Laboratórios Remotos, Arquitetura de Dispositivos Inteligentes

ABSTRACT

The experimentation of theoretical knowledge in laboratories tends to improve the teaching-learning processes, however, to overcome the lack of these, there are alternatives such as virtual and remote laboratories. It is noteworthy that in the manipulation of the remote laboratories, the result obtained in each experiment is real, being influenced by environmental variables. The benefits of its use go beyond pedagogical, such as the availability of Internet access, sharing of infrastructure and possible savings of financial resources. In general, such laboratories are developed in a strongly coupled and non-standardized way to meet specific needs. This lack of standard makes it difficult to share, reducing the possibility of increasing the benefits generated by its use, starting from this difficulty, this work proposes the use of the architecture of smart devices as an alternative to this gap, developing a prototype for remote laboratory communication, to which can be used as a reference in the development of new experiments or in the re-adaptation of existing ones, aiming at the standardization of the experiments. This work relies on a review of the literature on the architecture of intelligent devices applied in laboratories and the development of prototypes. The inclined plane of the RExLab laboratory was used as a starting point, since it has sensors and actuators manipulated by the users, allowing for a broader implementation. The computational technologies used were Node.js on the server side, WebSockets as communication medium, asynchronous feature, and JSON format as standard for data exchange, as well as clients, for Web platform in HTML and JavaScript language, in the C# and Java desktop languages. The prototypes proved to be functional, not interfering with the operation requirements of the experiment. The adoption of this model as a standard has proven to be feasible, since it is independent of hardware platform and programming language, having compatible tools for the most used platforms in the development of remote experiments..

Keywords: Standardization of Remote Laboratories, Online Laboratories, Remote Laboratories, Smart Device Architecture

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Ajax - *Asynchronous JavaScript and XML*
API - *Application Programming Interface*
CSV - *Comma-Separated Values*
DSRM - *Design Science Research Methodology*
E/S - Entrada e Saída
HTML - *HyperText Markup Language*
HTTP - *Hypertext Transfer Protocol*
INEP - Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira
IoT - *Internet of Things*
JSON - *JavaScript Object Notation*
LR - *Laboratórios Remotos*
MEC - Ministério da Educação
MEMS - *Micro Electro Mechanical Systems*
NPM - *Node Package Manager*
POO - Programação Orientada a Objetos
RLMS - *Remote Laboratory Management System*
SBC - *Single-board Computers*
SOA - *Service-Oriented Architecture*
STEM - *Science, Technology, Engineering, and Mathematics*
TLS - *Transport Layer Security*
UI - *User Interface*
URI - *Uniform Resource Identifier*
URL - *Uniform Resource Location*
XML - *Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Os ambientes de aprendizagem mais tradicionais são pautados pela difusão teórica, onde o professor transmite conteúdos por meios teóricos aos alunos, no entanto, a realização de experimentos possui grande importância para a comprovação dos conhecimentos teóricos adquiridos e como modo de aprender (CAMPOS; CRUZ, 2013).

Desta forma, Freeman et al. (2014) afirmam que o uso de laboratórios, permite que o estudante tenha um papel de protagonista em seu processo de aprendizagem.

Além dos laboratórios tradicionais, existem alternativas para que os alunos realizem atividades experimentais, por meio de laboratórios virtuais e remotos (NEDIC; MACHOTKA; NAFALSKI, 2003). Os laboratórios virtuais geralmente se apresentam como simulações e animações, já os remotos, são laboratórios reais acessados a distância por meio da Internet.

Com o aumento da disponibilidade de infraestruturas de telecomunicações e condições de acesso à Internet, Lowe et al. (2009) apontam um aumento recente no desenvolvimento de experimentos remotos, que são baseados na utilização de dispositivos tais como desktops, notebooks, *tablets* ou *smartphones*, bem como também na Internet das Coisas - IoT (do inglês: *Internet of Things*) .

O acesso e manipulação de experimentos remotamente se diferencia de outras tecnologias empregadas no ensino como simulações, pois o resultado obtido em cada experiência é real, sendo influenciado por uma série de variáveis de ambiente, que normalmente não são incluídas pelas simulações (SILVA, 2007).

A utilização de laboratórios de experimentação remota pode proporcionar a redução de custos e compartilhamento de uso com outras instituições, ampliando o alcance dos benefícios de sua utilização, uma vez que um mesmo experimento pode ser acessado por diversas escolas, mesmo que geograficamente distantes. Além disso, Silva (2007) cita que as configurações de equipamento necessárias para o acesso aos experimentos geralmente são mínimas, sendo a conexão à Internet o requisito mais importante.

Diante disto, o foco desta dissertação centra-se nos laboratórios remotos, pois desafios significantes são impostos na forma como os experimentos são desenvolvidos, disponibilizados e compartilhados, principalmente na padronização da comunicação entre o software cliente e o laboratório remoto, de modo que por meio da arquitetura cliente-servidor, esta comunicação ocorra de forma independente.

1.1 IDENTIFICAÇÃO DO PROBLEMA DA PESQUISA

De acordo com estudo realizado pelo projeto Go-Lab, Tsourlidaki e Zervas (2014) apontam que dentre os principais repositórios de atividades para aprendizagem, existem em torno de 1.565 laboratórios virtuais e remotos no mundo em 2014. Porém, a falta de padronização no desenvolvimento destes laboratórios pode dificultar o compartilhamento de recursos entre instituições de ensino.

Para Govaerts e Salzmann (2014), a integração de diferentes modelos de experimentos com softwares clientes e/ou sistemas de gerenciamento de laboratórios é trabalhosa, poderia ser facilitada com a adoção de um padrão a ser obedecido, pois estes estabelecem especificações e procedimentos, projetados para maximizar a compatibilidade, funcionalidade e facilitar a interoperabilidade.

O melhor aproveitamento dos recursos laboratoriais é uma das vantagens da utilização dos laboratórios remotos, (SILVA, 2007), que são muitas vezes caros e ocupam espaço físico, no entanto, a falta de padronização dificulta seu compartilhamento, deixando-os ociosos por grande parte do tempo.

A criação de federações com instituições que compartilham experimentos remotos entre si, pode ser uma saída para o desenvolvimento e popularização deste modelo de laboratório.

Portanto, identifica-se a seguinte pergunta de pesquisa: *"Como padronizar a comunicação dos laboratórios remotos?"*.

1.1.1 Objetivo Geral

Desenvolver um protótipo de comunicação para laboratórios remotos utilizando a arquitetura de dispositivos inteligentes, visando a padronização dos experimentos.

1.1.2 Objetivos Específicos

O levantamento dos objetivos específicos deu-se em função do caminho a ser percorrido para alcançar o proposto no objetivo geral, são eles:

- Pesquisar na literatura especificações técnicas e arquiteturas de laboratórios remotos;
- Verificar se a adoção do arquitetura proposta, compromete funcionalidades do laboratório;

- Analisar a utilização deste modelo de comunicação e a potencialidade de proporcionar a expansão da quantidade de laboratórios e compartilhamento de experimentos.

1.2 JUSTIFICATIVA

Os laboratórios de experimentação remota, segundo Govaerts e Salzmann (2014), de maneira geral, são desenvolvidos de maneira fortemente acoplada, para atender as necessidades específicas de cada experimento ou instituição, não sendo adotado uma padronização no modelo de comunicação entre os experimentos e servidores.

Esta falta de padronização cria dificuldades em compartilhar estes experimentos com terceiros, como também na adaptação de experimentos externos para serem utilizados em suas próprias plataformas RLMS (do inglês: *Remote Laboratory Management System*).

A adoção do paradigma de dispositivos inteligentes, juntamente com a padronização aplicada aos laboratórios remotos, pode permitir a criação ou ampliação de federações para compartilhamento de laboratórios online.

Geralmente, laboratórios possuem alto custo, seja de aquisição ou manutenção, o compartilhamento de experimentos ou laboratórios entre diferentes instituições pode ser benéfico no quesito financeiro como também pedagógico, ampliando a oferta de recursos educacionais à seus estudantes (SILVA, 2007).

1.3 ADERÊNCIA AO PROGRAMA DE PÓS-GRADUAÇÃO EM TECNOLOGIA DA INFORMAÇÃO E COMUNICAÇÃO

O programa de Pós-Graduação em Tecnologias da Informação e Comunicação (PPGTIC) da Universidade Federal de Santa Catarina (UFSC) - campus Araranguá, é um programa interdisciplinar, que está estruturado na área de concentração "Tecnologia e Inovação", tendo três linhas de pesquisa: "Tecnologia, Gestão e Inovação", "Tecnologia Educacional" e "Tecnologia Computacional".

A linha de pesquisa de tecnologia computacional, tem como objetivo desenvolver modelos, técnicas e ferramentas computacionais que auxiliem na resolução de problemas de natureza interdisciplinar, especificamente, uma de suas vertentes, procura desenvolver novas tecnologias para aplicação na área de educação.

Neste contexto, este trabalho converge com a linha de pesquisa em

tecnologia computacional, pois propõe um modelo de comunicação para laboratórios remotos baseado na arquitetura de dispositivos inteligentes.

1.4 ESTRUTURA DO TRABALHO

Esta dissertação, está organizada em 5 capítulos, além da introdução, os demais abordam os seguintes conteúdos:

No capítulo 2, apresenta-se o referencial teórico sobre os conceitos elementares ligados ao uso de tecnologia na educação, laboratórios remotos e tecnologias computacionais utilizadas no desenvolvimento do protótipo.

O capítulo 3 descreve as fases da pesquisa e os procedimentos metodológicos utilizados em seu desenvolvimento.

O capítulo 4 apresenta os materiais utilizados no desenvolvimento dos protótipos e o capítulo 5 os testes realizados pelo método descritivo por meio de cenários.

O capítulo 6 apresenta as considerações finais, verificando se os objetivos propostos foram alcançados, recomendações para trabalhos futuros, seguido do referencial bibliográfico e apêndices.

2 REVISÃO DA LITERATURA

Neste capítulo, são apresentadas as tecnologias computacionais inerentes ao desenvolvimento do protótipo de comunicação para laboratórios remotos, utilizando a arquitetura dos dispositivos inteligentes, ressalta-se que a intenção não é aprofundar-se na literatura sobre o assunto, mas sim abordá-las de maneira sucinta.

Um dos objetivos dos laboratórios remotos é estarem acessíveis por meio da Internet, o uso de tecnologias e padrões abertos evita a necessidade de *plugins* adicionais nos clientes, assim, o protocolo HTTP normalmente é utilizado.

No entanto, um problema dos serviços da Web baseados no HTTP é seu esquema cliente/servidor síncrono, em que o servidor não pode iniciar um "envio" de informações para os clientes, porém, o uso de WebSockets fornece um canal de comunicação bidirecional, *full-duplex*, suportada por todos os navegadores modernos (SALZMANN et al., 2015).

Nesse sentido são apresentados as seguintes seções: tecnologias na educação, laboratórios remotos, as tecnologias computacionais utilizadas neste trabalho, que possui as subseções: plataforma Node.js, o formato para troca de dados JSON (*JavaScript Object Notation*) e o protocolo e API[1] WebSocket, na seção seguinte é abordo o paradigma dos Dispositivos Inteligentes e sua arquitetura para utilização nos laboratórios remotos.

2.1 TECNOLOGIAS NA EDUCAÇÃO

Para compreendermos o papel das Tecnologias da Informação e Comunicação na educação, é necessária uma breve abordagem que revele princípios historicamente construídos, que regem a dinâmica das relações entre as duas ao longo da história (ROCHA, 2009). O processo de educação utilizando novas mídias, vêm se desenvolvendo desde 1970 no mundo inteiro, com o objetivo de formar usuários ativos, criativos, críticos de todas as tecnologias de informação e comunicação (DORIGONI; SILVA, 2008).

Segundo Almeida (2008) foi a partir da década de 70 que deu-se os primeiros passos para inserção da tecnologia digital no sistema brasileiro de ensino, sendo que esta iniciativa representava uma grande inovação ao criar um espaço de diálogo entre pesquisadores e educadores, que se dedicavam a estudos sobre o uso dos computadores na educação, buscando viabilizar a

[1] API é o acrônimo de *Application Programming Interface*, ou em português, Interface de Programação de Aplicativos.

articulação entre pesquisa e ensino, sendo que posteriormente isto se concretizaria como uma atividade comum na área da educação.

No entanto, Nascimento (2009) ressalta que devido a posição autoritária que predominava na época, foi uma atitude ousada e diferente pensar-se em novas ferramentas para auxiliar no processo de ensino-aprendizagem. Uma das estratégias para a discussão do referido tema foi a realização de seminários nacionais com a participação da comunidade científica, no qual decidiu-se que seria necessário a criação de referências para elaboração de projetos pilotos que visavam a utilização dos computadores, antes de uma disseminação massiva.

Diante destas recomendações, Nascimento (2009) afirma que no ano de 1984, o MEC implantou o projeto EDUCON – Educação com Computador - em cinco universidades públicas brasileiras (Universidades Federais do Rio Grande do Sul, Pernambuco, Minas Gerais, Rio de Janeiro e Universidade Estadual de Campinas), tendo como objetivo promover a criação de centros pilotos com a finalidade de contribuir para o desenvolvimento de pesquisas sobre o uso do computador no processo de ensino-aprendizagem, na formação de professores da rede pública e a produção de software educativo.

Diversos documentos, seminários e projetos referentes a integração das TIC na educação foram sendo publicados no decorrer do tempo, nesta linha, o último projeto foi criado em abril de 1997, o Programa Nacional de Informática na Educação (ProInfo), que a partir de 12 de dezembro de 2007, foi reestruturado mediante a criação do Decreto n.º 6.300, e passou a ter o objetivo de promover o uso pedagógico das tecnologias de informação e comunicação nas redes públicas de educação básica (NASCIMENTO, 2009).

No entanto, apesar dos programas governamentais, a quantidade de equipamentos e as condições de infraestrutura para seu funcionamento não alcançaram todas as escolas, o Censo Escolar MEC/INEP de 2017, apresenta que 39% das escolas públicas possuem laboratórios de informática, também segundo o censo, apenas 10% dispõem de laboratórios de ciências.

As carências de infraestrutura nas escolas é um dos fatores que dificultam a integração de tecnologia na educação, e a presença dos equipamentos não é igualitária no Brasil, dentre as que possuem laboratórios, a maioria estão nas regiões Sul, Sudeste e Centro-Oeste (CHITUNGO, 2018).

Conforme Becker et al. (2016), muitos países compreendem a importância das TIC no processo de ensino-aprendizagem e estão utilizando-as como apoio na elaboração de estratégias e programas educacionais.

Mas para isso é essencial que os docentes, independente da sua área de formação, possam conhecer as potencialidades e limitações pedagógicas envolvidas nas diferentes tecnologias que podem ser utilizadas: seja por meio de vídeos, computador, dispositivos móveis, entre outros (CHITUNGO, 2018).

Conforme Barreto (2004), o uso de novas tecnologias educativas leva à extinção dos limites entre as disciplinas, redefinindo ao mesmo tempo, a função, formação e o aperfeiçoamento dos docentes.

Como já citado, as escolas públicas são deficitárias tanto na quantidade de equipamentos de TIC como na infraestrutura física para para suportá-las. Com isso, são necessárias alternativas que permitam o acesso a "ferramentas" educacionais, seja dentro ou fora do ambiente escolar por meio da Internet, como os laboratórios virtuais e remotos.

2.2 LABORATÓRIOS REMOTOS

Antes de descrever os conceitos a cerca dos laboratórios, é necessário relembrar que os ambientes de aprendizagem mais tradicionais são aqueles pautados pela difusão teórica, ou seja, são as salas de aula onde o professor transmite conteúdos por meios teóricos aos alunos, formando geralmente indivíduos que adquiriram conhecimento das diversas teorias transmitidas por professores (CAMPOS; CRUZ, 2013).

No entanto, é importante entender que a realização de experimentos possui grande importância para a comprovação dos conhecimentos teóricos e como modo de aprender, desta forma considera-se que as atividades práticas realizadas muitas vezes em laboratórios, reafirmam os conhecimentos teóricos.

Freeman et al. (2014) citam que diversos estudos apontam que as atividades experimentais utilizadas no modelo de aprendizagem ativa, possuem um papel importante no processo de ensino-aprendizagem, principalmente nas disciplinas de ciências, tecnologia, engenharia e matemática, conhecido como STEM (do inglês: *Science, Technology, Engineering, and Mathematics*).

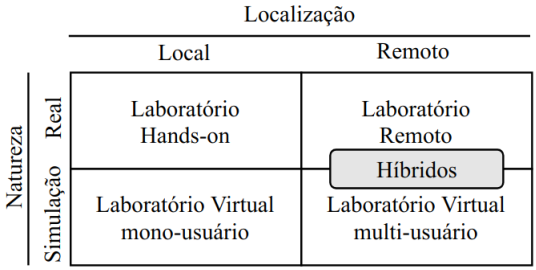
Neste sentido, o trabalho experimental não somente possibilita a comprovação de leis e teorias, como também permite que o estudante tenha um papel de protagonista em seu processo de aprendizagem junto ao docente e os recursos disponibilizados. Além disso, é possível argumentar que as atividades práticas, principalmente nas disciplinas de ciências, podem se constituir de uma estratégia didática para a construção de competências procedimentais, que é o aprender a fazer (CHITUNGO, 2018).

Deste modo, além dos laboratórios tradicionais, existem alternativas para que os alunos realizem atividades experimentais, por meio de laboratórios virtuais e remotos. Nedic, Machotka e Nafalski (2003) classificam os laboratórios como: real, virtual e remoto, contudo Silva (2007), classifica os laboratórios como: presencial e virtual, dividindo os virtuais em laboratórios

baseados em simulação e laboratórios de acesso remoto. Muito embora, com os avanços das tecnologias computacionais, Gomes e Bogosyan (2009), Zutin et al. (2010) e Rodriguez-Gil et al. (2017) já descrevem uma nova categoria de laboratórios, que são os laboratórios remotos híbridos, na figura 1 ilustra-se um diagrama quanto a classificação dos laboratórios.

Com diversas nomenclaturas na literatura, a denominação utilizada neste trabalho é composta por: (a) Laboratório *Hands-on*, que são os presenciais, (b) Laboratório Virtual, que são os laboratórios on-line baseados em simulação, (c) Laboratório Remoto, que são laboratórios *hands-on* utilizados a distância, e (d) Laboratório Híbrido, que são os laboratórios que misturam elementos dos laboratórios virtuais e remotos.

Figura 1 – Classificação dos Laboratórios.



Fonte: Traduzido de Rodriguez-Gil et al. (2017).

Na sequência são descritas as principais características de cada tipo de laboratório.

(a) Laboratórios *Hands-on*:

São a forma tradicional de apoiar a experimentação, sendo um local de trabalho e muita concentração, no entanto, pode tornar-se um local perigoso se for usado de forma inadequada por causa dos materiais e dos equipamentos existentes. A maioria dos acidentes ocorre por desconhecimento das regras básicas de segurança ou por falhas no preparo prévio dos alunos (CAMPOS; CRUZ, 2013).

Segundo Gomes e Bogosyan (2009), muitas vezes estão associados a altos custos, infelizmente alguns laboratórios presenciais são muito caros em relação ao equipamento necessário, além do espaço e as questões de pessoal e manutenção. Em algumas especialidades de engenharia esses custos aumentam ainda mais.

Para Chitungo (2018), outra limitação significativa nos experimentos *hands-on*, ocorre em algumas situações onde existe um elevado número de

estudantes, dificultando a utilização desses laboratórios, deste modo é uma razão para considerar algumas alternativas como a simulação e laboratórios remotos.

(b) Laboratórios Virtuais:

Estes são baseados em simulação e não requerem espaço físico, são altamente escalonáveis e podem adaptar-se a realidade para atender às necessidades de ensino, simplificando-a ou exibindo fenômenos inobserváveis.

Estes laboratórios, geralmente se apresentam como simulações e animações de experimentos científicos, geralmente disponíveis na Web, no entanto, em algumas ocasiões a simulação não leva em conta aspectos do mundo real que poderiam ser importantes na hora de realizar experimentos (SILVA, 2007).

Como exemplo de laboratórios virtuais, estão os presentes no projeto "PhET Interactive Simulations" da Universidade do Colorado Boulder, que disponibiliza quase 150 simulações, fundado em 2002 pelo ganhador do prêmio Nobel Carl Wieman.

> Os PhET Sims, criam simulações interativas gratuitas de matemática e ciências, são baseados em extensa pesquisa educacional e envolvem os alunos através de um ambiente intuitivo, semelhante a um jogo, onde os alunos aprendem através da exploração e descoberta. (PhET, 2018, s.p.)

A figura 2 ilustra a tela de acesso ao simulador de ondas, na qual é possível realizar diversas alterações nos parâmetros da simulação, permitindo observar os diferentes resultados.

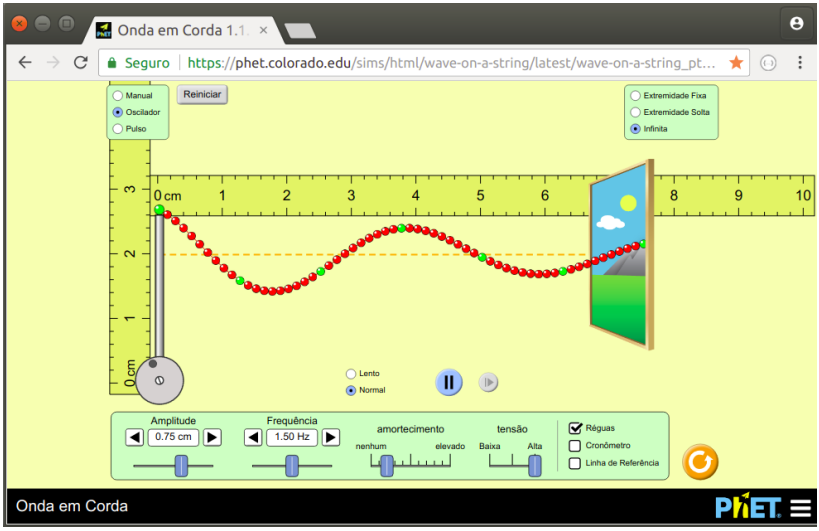
(c) Laboratórios Remotos:

Os laboratórios virtuais remotos, ou simplesmente Laboratórios Remotos - LR, permitem o acesso à experimentos que possuem equipamentos reais de forma remota, ou seja, pode ser controlado remotamente pelo usuário (ZUTIN et al., 2010), ainda como citado por Nicolete (2016), os elementos e as experiências são reais, apesar do acesso virtual.

Os laboratórios remotos tradicionais contam apenas com equipamentos reais, de modo que fornecem dados reais e incluem atrasos autênticos e eventos imprevistos, como imprecisões nas medições, através dos quais os alunos podem aprender sobre as complexidades da ciência (RODRIGUEZ-GIL et al., 2017).

O acesso e manipulação de experimentos remotamente se diferencia de outras tecnologias empregadas no ensino como simulações, pois o resultado obtido em cada experiência é real, sendo influenciado por uma série de variáveis de ambiente que normalmente não são incluídas pelas simulações.

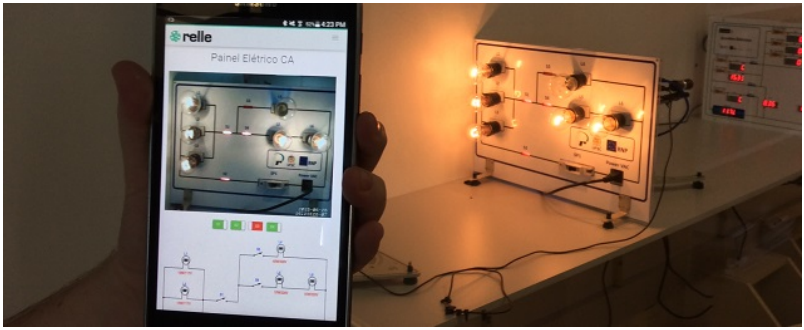
Figura 2 – Laboratório Virtual de simulação de ondas.



Fonte: PhET (2018).

O usuário geralmente observa o laboratório através de um fluxo de vídeo ao vivo (figura 3), em alguns experimentos pode-se estipular o valor de determinados parâmetros permitidos e visualizar os dados ou medição em tempo real. O servidor do laboratório tem de assegurar a interação remota do usuário, partindo de um aplicativo autônomo ou como um componente em um ambiente da Web.

Figura 3 – Laboratório Remoto.



Fonte: http://relle.ufsc.br (2018).

(d) Laboratórios Híbridos:

Segundo Rodriguez-Gil et al. (2017), este tipo de laboratório mistura componentes virtuais e reais. O laboratório híbrido padrão, no entanto, precisará representar ou estender a realidade da experimentação por meio de gráficos e, às vezes, de mídias adicionais.

Os laboratórios virtuais e remotos são uma inovação promissora para melhorar o ensino e a aprendizagem nas disciplinas de ciências em todos os níveis da educação, a utilização destes recursos pode permitir a construção de conhecimentos em contextos reais de trabalho. Os trabalhos de Nicolete (2016), Heck (2017), Chitungo (2018) apresentam resultados positivos no processo de ensino-aprendizagem utilizando os laboratórios remotos.

Silva (2007), cita algumas vantagens no uso de laboratórios remotos tais como:

- Maior utilização dos equipamentos do laboratório - ao estarem disponíveis 24 horas por dia, 365 dias ao ano;
- Organização de laboratórios - não é necessário manter abertos o tempo todo, basta que estejam operacionais;
- Organização do trabalho dos alunos e professores - podem organizar melhor seu tempo, de maneira similar aos horários de aulas;
- Aprendizagem autônoma - os laboratórios remotos fomentam o trabalho autônomo, que é fundamental no modelo atual de educação superior;
- Inserção dos usuários em um contexto real - uma vez que elementos do experimento passam a ser controlados através de um computador os usuários são inseridos em um contexto real de aprendizagem.

Além disso, as configurações necessárias dos equipamentos para o acesso aos experimentos geralmente são mínimas, sendo a conexão à Internet o requisito mais importante.

Nicolete (2016) relata a importância dos laboratórios de experimentação remota para complementar as aulas expositivas nas áreas das ciências, tecnologia e engenharias, tanto para o Ensino Superior como para a Educação Básica, em disciplinas como Física, Química e Biologia.

Simão et al. (2013) afirmam que esses laboratórios se apresentam como meio alternativo de suprir ou amenizar a falta e a complexidade em relação ao uso de experimentos nas aulas de ciências.

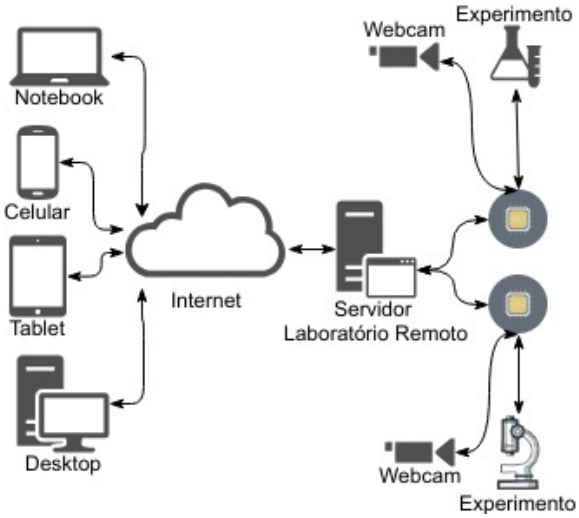
Com o aumento da disponibilidade de infraestruturas de telecomunicações e de condições de acesso a rede, Lowe et al. (2009) apontam um aumento recente no desenvolvimento de experimentos remotos, que são baseados na utilização de dispositivos tais como desktops, notebooks, tablets ou smartphones bem como na Internet das Coisas.

Os laboratórios remotos, geralmente dependem de uma arquitetura cliente/servidor, permitindo por meio da Internet a comunicação com o equipamento real. O servidor de um laboratório pode ser um microcontrolador, um computador ou um sistema de gerenciamento de laboratório remoto.

Como ilustrado na figura 4, em muitas implementações, os LR são disponibilizados através de interfaces baseadas na Web, ou seja, uma página HTML[2] com um fluxo de vídeo em tempo real do experimento. Esta interface geralmente permite que o estudante altere o valor de uma ou mais variáveis[3], deste modo, pode visualizar o resultado e obter valores resultante.

Ao utilizar este tipo de tecnologia no desenvolvimento da interface, não existe necessidade de instalação de softwares adicionais para utilização do LR.

Figura 4 – Estrutura típica dos Laboratórios Remotos.



Fonte: Elaborado pelo autor.

Estes laboratórios, de maneira geral, são desenvolvidos de modo for-

[2]HyperText Markup Language – Linguagem de Marcação de HiperTexto.
[3]Depende do tipo de experimento e configurações possíveis.

temente acoplado, para atender as necessidades específicas de cada experimento ou instituição, não sendo adotada uma padronização no modelo de comunicação entre os software cliente e o servidor do experimento.

Com foco nos laboratórios remotos, significantes desafios são impostos na forma como os experimentos são disponibilizados e compartilhados. A possível implementação do paradigma dos dispositivos inteligentes (*Smart Device*) permite que a arquitetura cliente/servidor ocorra de forma independente.

Devido a grande variedade de experimentos, a maneira como os laboratórios remotos funcionam pode variar, a figura 5, apresenta uma arquitetura típica de um laboratório remoto.

O software cliente é responsável por realizar a interação com o usuário na parte (A) da comunicação, o desenvolvimento desta interface pode prezar pela estética, usabilidade e responsividade[4].

Na comunicação entre o software cliente e os serviços providos pelo laboratório (B), a comunicação não obedece padrões, o que dificulta o compartilhamento ou desenvolvimento de novos softwares clientes, deste modo, essa comunicação pode ser padronizada utilizando o paradigma dos dispositivos inteligentes.

Já na comunicação entre o servidor e o experimento (C), cada implementação demanda de necessidades específicas na comunicação, pois com a utilização de servo motores, acionamento de interruptores, regulagem de tensão entre outros acionamentos, cada projeto deve ser elaborado de acordo com os requisitos do proprietário do laboratório.

Na sequência, faz-se referências às tecnologias computacionais utilizadas no desenvolvimento do trabalho.

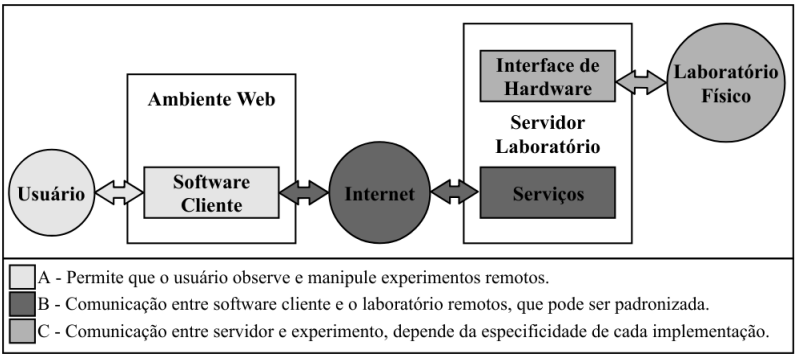
2.3 TECNOLOGIAS COMPUTACIONAIS

As tecnologias computacionais estão integradas ao cotidiano de forma transparente. Realizar atividades por meio de recursos tecnológicos já se transformou em hábito para grande parte dos usuários. No entanto, para cada tipo de recurso ou serviço disponível, diversos mecanismos e protocolos são empregados em conjunto, propiciando serviços complexos.

Neste contexto, para que seja possível a disponibilização de laboratórios remotos, um amplo conjunto de tecnologias podem ser adotadas, no entanto, serão apresentadas as que são defendidas por Govaerts e Salzmann (2014) e utilizadas neste trabalho para implementar a arquitetura dos *Smart*

[4]Ajuste automático do *layout* conforme o tamanho da tela do equipamento utilizado pelo usuário (ex:computador, tablet e celular).

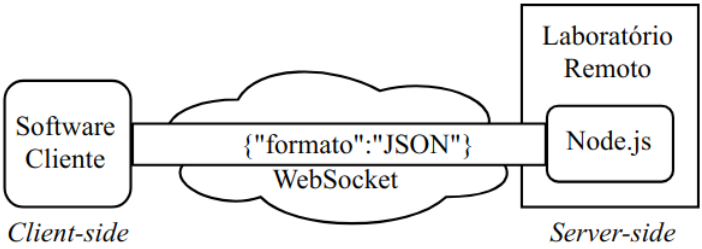
Figura 5 – Arquitetura típica de Laboratórios Remotos.



Fonte: Adaptado de Govaerts e Salzmann (2014).

Devices em Laboratórios Remotos na figura 6.

Figura 6 – Tecnologias e suas aplicações.



Fonte: Elaborado pelo autor.

O Node.js trabalha como uma espécie de servidor Web e permite fornecer as interfaces dos serviços do laboratório, já o WebSocket permite a comunicação *full-duplex* entre o software cliente e o laboratório remoto em tempo real, e o JSON é um formato para serialização de dados, possibilitando a troca de informações entre sistemas distintos.

2.3.1 Node.js

No acesso aos conteúdos disponíveis na Web, os dados devem estar em um servidor que conta com um software específico para disponibilizar conteúdo na World Wide Web - WWW, que são conhecidos como servido-

res Web ou servidores HTTP [5]. Atualmente os servidores mais conhecidos e utilizados são: Apache, NGINX e IIS da Microsoft (NETCRAFT, 2018),

Cada software possui pontos fortes e fracos, para algumas aplicações específicas, estes softwares mais utilizados podem não representar a melhor solução, como por exemplo, a implementação de um servidor Web em minicomputadores, como por exemplo o Raspberry PI, que é muito utilizado em laboratórios remotos.

Neste sentido, destaca-se que o Node.js, de fato não se apresenta como um servidor Web, mas como um ambiente JavaScript em tempo de execução[6] do lado do servidor, construído com a *engine* V8 do Google Chrome.

Esta *engine* é responsável por permitir que o navegador Web consiga interpretar os códigos JavaScript no computador do cliente, por outro lado, o Node.js é utilizado para interpretar os códigos no servidor. Em outras palavras, ele funciona como um ambiente no lado do servidor, que traduz códigos escritos em JavaScript para linguagem de "máquina", ele pode ser utilizado no Windows 7+, MacOS 10.12+ e no Linux.

A linguagem de programação Javascript[7] surgiu em 1995, e foi padronizada em 1997 sendo inicialmente utilizada para Web, atualmente desenvolvedores utilizam Javascript na Web com HTML5, que geralmente possui alto desempenho, como pode ser visto no desenvolvimento de jogos 3D para navegadores Web.

As principais vantagens da utilização do Node.js são:

- Modelo não bloqueante por eventos de E/S (Entrada e Saída);
- Compatibilidade com minicomputadores e sistemas embarcados;
- Sistema de pacotes NPM[8] (Node Package Manager);
- Amplo número de bibliotecas para comunicação (rede, USB, GPIO[9]).

O modelo não bloqueante por eventos de E/S, o torna leve e eficiente (MULDER; BRESEMAN, 2016), para diferenciar uma operação bloqueante e não bloqueante, Teixeira (2012) explica que uma execução bloqueante ocorre

[5]HTTP - Protocolo de Transferência de Hipertexto (do inglês: *Hypertext Transfer Protocol*) é um protocolo de comunicação utilizado na Web.

[6]O mecanismo de tempo de execução deve estar sendo executado no computador para que o aplicativo seja executado. Ele fornece rotinas e funções comuns que os aplicativos exigem e normalmente converte o programa, que está em uma linguagem temporária intermediária, em linguagem de máquina, fonte: <https://www.pcmag.com/encyclopedia/term/56079/runtime-engine>.

[7]JavaScript e Java são linguagens completamente diferentes, tanto em conceito como design.

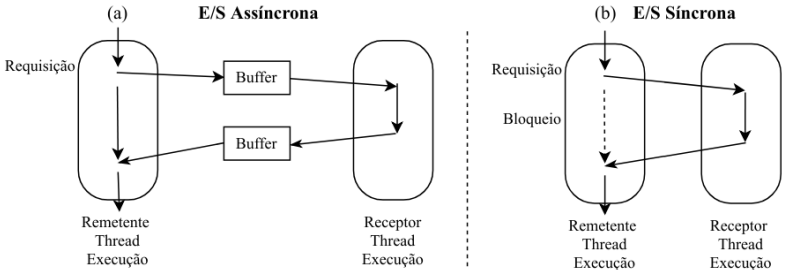
[8]Pode ser acessado em: <https://www.npmjs.com/>.

[9]Do inglês: *general-purpose input/output*, são basicamente portas programáveis de entrada e saída de dados.

quando a execução do código JavaScript adicional no processo Node.js deve aguardar até que uma operação E/S seja concluída (leitura de disco ou rede por exemplo) ilustrado na parte (b) da figura 7.

Isso ocorre porque o *loop* de eventos não consegue continuar executando o JavaScript enquanto uma operação de bloqueio está ocorrendo.

Figura 7 – Modelo assíncrono e síncrono.



Fonte: Traduzido de Teixeira (2012).

Deste modo, a característica de destaque do Node.js está em seu modo não bloqueante, ilustrado na figura 7 parte (a), Teixeira (2012) afirma que este estilo de programação - em que, ao invés de usar um valor de retorno, define-se funções chamadas pelo sistema quando ocorrem eventos interessantes - é chamada de programação assíncrona ou baseada em eventos.

Chaniotis, Kyriakou e Tselikas (2015) descrevem que no modelo bloqueante, o desempenho pode apresentar-se fraco devido ao uso intensivo de CPU para desempenho de operações que não são JavaScript, no entanto, todos os métodos de E/S na biblioteca padrão Node.js fornecem versões assíncronas, que não são bloqueadoras e aceitam funções de retorno de chamada.

Neste estilo de programação, significa que o processo atual não será bloqueado quando estiver fazendo operações de E/S, portanto várias E/S podem ocorrer em paralelo, e cada função de retorno de chamada respectiva será invocada quando a operação for encontrada.

Para ilustrar a diferença entre programação bloqueante (síncrona) e não bloqueante (assíncrona), a figura 8 ilustra ambos tipos de códigos na linguagem JavaScript.

O código apresenta duas formas de realizar uma consulta em um banco de dados, as linhas 1 e 5 são comentários, significa que não serão interpretadas. As linhas 2 e 3 consistem numa programação típica de bloqueio de E/S, em que a consulta em um banco de dados requer que a $thread$ ou processo atual espere até que a camada do banco de dados termine de processá-la.

Na linha 9, o código está definindo o que ocorrerá quando a consulta

Figura 8 – Código JavaScript bloqueante e não bloqueante.

```
1 // codigo bloqueante
2 result = query('SELECT * FROM posts WHERE id = 1');
3 do_something_with(result);
4
5 // codigo nao bloqueante
6 query_finished = function(result) {
7   do_something_with(result);
8 }
9 query('SELECT * FROM posts WHERE id = 1', query_finished);
```

Fonte: Adaptado de Teixeira (2012).

for concluída, armazenando-a em uma função chamada $query\_finished$, então estará passando essa função como um argumento para a consulta, que ao terminar, chamará a função $query\_finished$ na linha 6 ao invés de simplesmente retornar o resultado.

Outra vantagem que se apresenta é sua compatibilidade com minicomputadores, o Node.js pode ser utilizado em SBCs (do inglês: *single-board computers*) tais como: Raspberry Pi, Arduino, BeagleBone e Intel Galileo, estes são alguns citados por Mulder e Breseman (2016).

Além das características já citadas, a linguagem JavaScript também é adequada para a programação de dispositivos embarcados. Algumas bibliotecas[10], como a para porta serial por exemplo, foram desenvolvidas pela equipe do *Chromium* com uma abordagem *wrapper* para o *gyp* compilar códigos na linguagem C/C++ nativo que possuem dependências de hardware específicas em bibliotecas JavaScript .

Isso permite que os projetos do Node.js façam interface com bibliotecas de hardware de nível muito baixo em diferentes plataformas.

O desempenho em tempo de execução da *engine* JavaScript V8 em um dispositivo embarcado ou SBCs é muito bom em comparação ao código compilado e a outras linguagens de alto nível. Além disso possui bibliotecas para comunicação em rede, sistemas de arquivos, *drivers* e bancos de dados (MULDER; BRESEMAN, 2016).

2.3.2 JSON - *JavaScript Object Notation*

Sistemas desenvolvidos para diversas finalidades podem ser implementados utilizando uma ampla gama de linguagens de programação, tecno-

[10] Biblioteca é uma coleção de subprogramas utilizados no desenvolvimento de software.

logias e arquiteturas, que são adotadas para suprir necessidades específicas, no entanto, a capacidade de comunicação entre sistemas distintos é essencial.

Deste modo é necessário um formato padronizado para troca de dados, que pode ser um formato de texto usado na transição de dados entre plataformas, segundo Bassett (2015), os formatos mais utilizados atualmente são o XML (*Extensible Markup Language*), CSV (*Comma-Separated Values*) e JSON (*JavaScript Object Notation*).

Um exemplo prático da necessidade de troca de informações entre sistemas distintos, ocorre diariamente na emissão de nota fiscal eletrônica (NFe) na venda de mercadorias. É necessário que um arquivo de texto contendo as informações da venda seja enviado para receita estadual através da Internet para ser validado. Neste processo, o formato para troca de dados adotado é o XML.

JSON e XML são parecidos em alguns aspectos, tais como: auto descrição, que facilita a leitura humana, utilizam hierarquia para armazenamento de valores e podem ser analisadas por muitas linguagens, no entanto, o JSON se diferencia por ser mais rápida para ser lida e escrita e permite a utilização de *arrays* [11].

Bray (2014) descreve que JSON é um formato de texto para serialização de dados estruturados, sendo um formato para troca de dados leve, baseado em texto, e independente de linguagem.

Reforçando, Severance (2012) afirma que os programadores utilizam JSON extensivamente para codificar dados para transferência entre um servidor e um aplicativo Ajax (*Asynchronous JavaScript and XML*), para conectar dois servidores que se comunicam por meio de serviços da Web e em muitos outros cenários semelhantes.

Apesar da palavra JavaScript estar presente na sigla JSON, ao analisar "Notação de Objetos", Bassett (2015) esclarece que "Objeto" é um conceito comum em programação, em particular na programação orientada à objetos (POO). A sintaxe JSON é derivada da notação de objeto JavaScript, em que os dados são pares de $chave$ e $valor$ separados por dois pontos ":".

A palavra "notação" implica num sistema de caracteres para representar dados como números, palavras ou elementos, deste modo pode representar os seguintes tipos de dados:

(a) Caracteres e *strings*[12] (ex:"a"ou "RExLab")

Exemplo: { "nome":"Amanda"}

(b) Números (ex: 289, 22.59, 47.606209)

[11] *Array* também é conhecido como lista ou vetor, é uma estrutura de dados que armazena uma coleção de elementos de forma que cada um possa ser identificado por uma chave ou índice.

[12] Conjunto de caracteres utilizadas para representar palavras.

Exemplo: { "idade":31 }

(c) Boleanos[13] (ex: *true* ou *false*)

Exemplo: { "venda":true }

(d) Nulo (null)

Exemplo: { "carro":null }

(e) *Arrays*

Exemplo: { "funcionarios":["Joao", "Ana", "Peter"] }

(f) Objetos

Exemplo: { "livro": { "titulo":"A", "paginas":30, "autor":"Roberto"} }

Ao representar uma *string* qualquer, como no exemplo (a), o valor correspondente a chave "nome", deve ser grafado entre aspas dupla, no caso, "Amanda", já no caso de dados do tipo boleano (b), numérico (c) e nulo (d) não é necessário utilização de aspas no valor.

Para a estrutura de um *array* (e) utiliza-se o chave grafada entre aspas duplas seguida por : (dois pontos) , o [(colchete de abertura) indica o início dos dados e] (colchete de fechamento) indica o fim dos dados do *array*, os valores constantes são separados por , (vírgula) e devem obedecer as regras quanto ao uso de aspas duplas quando for do tipo caractere ou *string*.

Para a estrutura de um objeto como no exemplo (f), utiliza-se a chave grafada entre aspas duplas seguido por : (dois pontos), e { (chave de abertura) para indicar o início dos dados do objeto e } (chave de fechamento) para indicar o fim, já os dados pertencentes ao objeto compostos por "chaves":"valores" devem seguir a mesma regras de notação, sendo separados por , (vírgula) (BRAY, 2017).

Os dados em formato JSON podem ser armazenados como um documento no sistema de arquivos, para isso basta utilizar a extensão ".json", também é reconhecido como um tipo de mídia da Internet (MIME Type), quando dados são enviados ou recebidos através do protocolo HTTP por exemplo, a requisição deve apresentar o nome do cabeçalho e o seu valor como: "*Content-type*", "*application/json*" (BASSETT, 2015; W3SCHOOLS, 2018).

Quando dados são recebidos de um servidor da Web, são sempre uma *string*, para trabalhar com os dados, é necessário torná-los um objeto (no conceito de programação) realizando o *parsing*[14] do texto que está em formato

[13] Tipo primitivo de dados que possui valores 0 ou 1 (0 = *false*, 1 = *true*)

[14] Processo de analisar um texto para determinar sua estrutura lógica, o texto pode descrever, por exemplo, um programa de computador, um banco de dados ou uma estrutura de dados.

JSON. O procedimento inverso também é necessário quando um objeto deve ser transformado em texto no formato JSON para ser enviado.

A maioria das linguagens possuem estas funcionalidades de conversão de objetos para texto em formato JSON e vice-versa, na linguagem JavaScript as funções são respectivamente $JSON.stringify()$ e $JSON.parse()$.

2.3.3 WebSocket

Antes de abordar a tecnologia Websocket, faz-se necessário realizar um resumo sobre a evolução dos mecanismos e protocolos utilizados no armazenamento e transmissão de dados na World Wide Web.

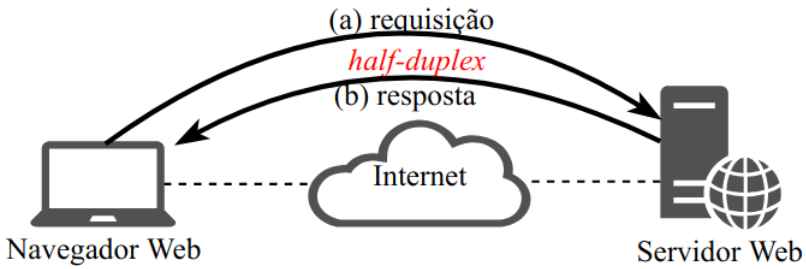
O modelo foi concebido em 1989 por Tim Berners-Lee, que em 1990 havia escrito a definição das três tecnologias que são a fundação da Web como conhecemos hoje: HTML, URI e HTTP (FOUNDATION, 2018).

O HTML *(HyperText Markup Language)* que é uma linguagem de marcação para formatação de documentos na Web, o URI *(Uniform Resource Identifier)* que é uma espécie de endereço único utilizado para identificar um recurso na Web, comumente chamado de URL *(Unifor Resource Location)* e o HTTP *(Hypertext Transfer Protocol)*, que permite o envio e recebimento de informações.

Este modelo, é baseado no paradigma cliente/servidor, como ilustrado na figura 9, em que o cliente geralmente utiliza um navegador Web, e o servidor possui um serviço de hospedagem de páginas Web, o *Web server*.

Por natureza, o protocolo HTTP é *half duplex*, o que significa que o tráfego flui em uma única direção por vez como na figura 9: (a) o cliente envia uma solicitação ao servidor (uma direção), e (b) o servidor então responde ao pedido (uma direção).

Figura 9 – Modelo cliente/servidor *half duplex*.



Fonte: Adaptado de Wang, Salim e Moskovits (2013).

Embora existam benefícios nesta forma de comunicação, ela nem sempre é a mais eficiente ou adequada, segundo Wang, Salim e Moskovits (2013), engenheiros têm trabalhado em torno desta limitação há anos com uma variedade de métodos como: *polling, long polling e HTTP streaming*.

Quando um cliente realiza uma requisição ao servidor, normalmente envia uma página Web como resposta. Em muitos casos ela pode conter tabelas de preços, informações climática, tráfego entre outras, se um usuário precisar de informações em tempo real, seria necessário atualizar a página constantemente de forma manual.

Wang, Salim e Moskovits (2013) apresentam que tentativas atuais de fornecer aplicativos da Web em tempo real giram em torno de um técnica chamada *polling*, que consiste numa chamada síncrona com horário regular, em que o cliente faz solicitação ao servidor para verificar se existe alguma informação disponível para ele, cada pedido é respondido, independente de existirem novas informações.

O *Polling*, envia dados ou checa eventos periodicamente (WANG; SALIM; MOSKOVITS, 2013), é uma boa solução se o intervalo de entrega das informações for conhecido, no entanto, dados em tempo real não possuem intervalo de alterações previsíveis, dessa forma existe uma quantidade de requisições desnecessárias.

O *Long Polling* é outro método, em que o cliente solicita informações do servidor e abre uma conexão durante um período de tempo definido, este método carece de implementações padrão e não fornece melhorias significativas de desempenho (WANG; SALIM; MOSKOVITS, 2013).

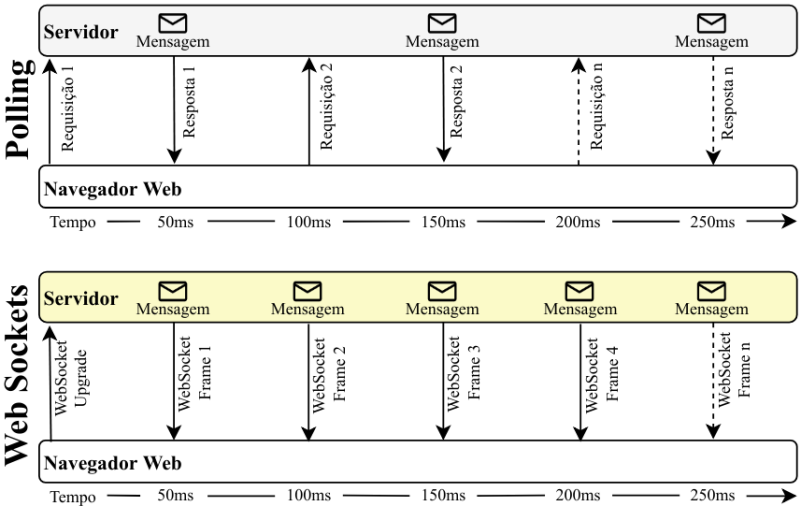
Para suprir esta limitação, o WebSocket, definido em Fette e Melnikov (2011), utiliza conexões HTTP para criar e manter comunicação bi-direcional entre servidor e cliente, desta forma, ambos podem enviar novos dados a qualquer momento em que a conexão estiver estabelecida (figura 10).

Ele foi projetado para funcionar com a infraestrutura da Web existente, como parte deste princípio, a especificação do protocolo define que a conexão WebSocket inicie com uma conexão HTTP, garantindo total compatibilidade com o mundo pré-WebSocket.

O seu desenvolvimento é parte de esforços para tornar o HTML5 capaz de competir com outras plataformas, uma vez que tende a tornar os navegadores Web capazes de executarem aplicações cada vez mais completas (WEBSOCKET.ORG, 2018).

O protocolo suporta a transmissão de dados em forma binária ou texto, já em termos de segurança, assim como no HTTP os dados são transmitidos em forma de texto, e no HTTPS são transportados criptografados, o WebSocket permite transmitir os dados em texto, utilizando o prefixo "ws://" ou criptografado por intermédio do WebSocket sobre TLS (do inglês: *Transport*

Figura 10 – Comparação entre HTTP Polling e WebSocket



Fonte: Adaptado de Wang, Salim e Moskovits (2013).

Layer Security) utilizando o prefixo "wss://".

A API WebSocket é uma interface que permite que os aplicativos usem o protocolo WebSocket, desta forma pode-se controlar um canal de comunicação *full-duplex* por meio de um aplicativo que pode enviar e receber mensagens simultaneamente.

A programação segue o modelo assíncrono: enquanto a conexão estiver aberta, o aplicativo escutará eventos: *open, message, error e close* e possui os métodos: *send*() e *close*(), que servem respectivamente para enviar dados e encerrar a conexão do WebSocket.

Para ilustrar sua utilização, é realizado uma breve descrição do fragmento de código JavaScript na figura 11, que apresenta a implementação de um cliente que realizará uma conexão ao servidor "echo.websocket.org" utilizando WebSocket.

Na linha 1, a variável ws receberá um novo objeto *WebSocket*, que se conectará ao endereço *echo.websocket.org/echo*, utilizando conexão sem criptografia, representado pelo prefixo *ws://*.

Na linha 3, quando o evento *onopen* do WebSocket *ws* ocorre, a função relacionada é acionada, e por meio da linha 4 escreve a mensagem *conectado* no console[15], em seguida na linha 5, executa a função *sendMessage()* com

[15]Console terminal ou console do navegador Web, de acordo com o local em que é executado.

Figura 11 – Exemplo aplicação cliente WebSocket.

```
1  ws = new WebSocket("ws://echo.websocket.org/echo");
2
3  ws.onopen = function(e) {
4    console.log("Conectado");
5    sendMessage("Ola WebSocket!");
6  }
7
8  ws.onclose = function(e) {
9    console.log("Disconectado: " + e.reason);
10 }
11
12 ws.onerror = function(e) {
13   console.log("Erro ");
14 }
15
16 ws.onmessage = function(e) {
17   console.log("Messagem recebida: " + e.data);
18   ws.close();
19 }
20
21 function sendMessage(msg){
22   ws.send(msg);
23   console.log("Messagem enviada");
24 }
```

Fonte: Traduzido e adaptado de WebSocket.org (2018).

a mensagem de texto *Ola WebSocket!* como argumento, que será enviada através do WebSocket estabelecido.

Na linha 8, quando o evento *onclose* do WebSocket *ws* ocorre, a função relacionada é acionada, e na linha 9 escreve a mensagem *Desconectado:* mais a informação sobre encerramento contidas em *e.reason* no console.

O código na linha 12 é executado quando o evento *onerror* acontece, assim a função relacionada exibe a mensagem *Erro* realizado pela linha 13.

Na linha 16, evento esperado é o *onmessage*, que significa que uma mensagem foi recebida através do WebSocket, a função relacionada recebe a mensagem como argumento, e na linha 17 apresenta na tela do console a mensagem *Menssagem recebida:* mais o conteúdo que está contido em *e.data*. Na linha 18, a conexão WebSocket contida em ws é encerrada pelo método *close()*.

Por fim, a função *sendMessage(msg)* na linha 21, recebe uma mensagem como argumento, e na linha 22 utiliza o método *send()* para enviar a mensagem através do WebSocket estabelecido em *ws*, na sequência, o código da linha 23 é responsável por apresentar a mensagem *Mensagem enviada* no

console.

Destaca-se que o evento *onclose* ocorre quando o WebSocket é encerrado pelo outro dispositivo conectado, neste caso o servidor, já o método *close()* realiza o fechamento da conexão por iniciativa do cliente.

Com esta flexibilidade na comunicação *full-duplex* fornecida pelo WebSocket, é possível enviar e receber informações assim que eventos ocorram, atualizando os valores na interface gráfica do usuário sem que tenham necessidade de atualizar a página para realizar nova requisição de dados.

Esta característica é útil nos laboratórios remotos, pois alguns experimentos ao serem acionados/iniciados, podem levar um tempo pré determinado para fornecer uma resposta, ao utilizar Websockets, no momento que os dados contendo os resultados estiverem disponíveis, serão enviados para o usuário.

2.4 DISPOSITIVOS INTELIGENTES - *SMART DEVICES*

Habitamos em um mundo digital em pleno crescimento, com uma profusão de serviços online de assistência e automação das atividades humanas. O ambiente físico está sendo incrementado por equipamentos digitais com capacidades de controle baseados em sensores incorporados, os circuitos eletrônicos e dispositivos estão sendo fabricados em tamanhos e valores menores, e cada vez mais confiáveis (POSLAD, 2009).

Os dispositivos inteligentes, como por exemplo: computador pessoal, telefone celular, tendem a ser dispositivos de TIC com múltiplos propósitos, contento uma tela para saída de dados e um teclado para entrada de dados (ou tela *touch screen* como dispositivo de entrada e saída de dados), no entanto, estes podem ser miniaturizados, e não utilizar saídas visuais, por exemplo, sistemas micro electro-mecânicos (MEMS - do inglês: *Micro Electro Mechanical Systems*).

Mesmo com seu tamanho cada vez mais diminuto, alguns aparelhos podem combinar vários pequenos componentes mecânicos e eletrônicos, permitindo que um crescente conjunto de funções sejam incorporadas nestes dispositivos.

Atualmente, MEMS são incorporados em muitos dispositivos, um exemplo é o acelerômetro em notebooks, para detectar quedas e parar componentes móveis, como os braços de discos rígidos, tais componentes estão sendo cada vez mais incorporados em sistemas amplamente acessados. Eles também são usados em muitos dispositivos para oferecer suporte à interação baseada em gestos. A miniaturização acompanhada de fabricação barata é um facilitador essencial para a visão da computação onipresente (POSLAD, 2009).

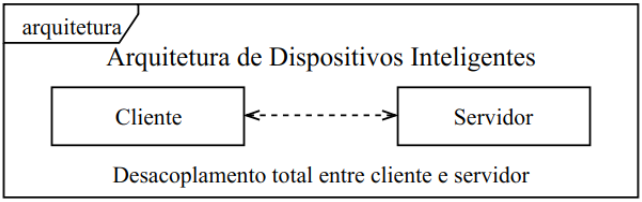
O conjunto destes dispositivos, formam um sistema distribuído, em que possui como princípio o modelo cliente/servidor. Ressalta-se que esta interação pela rede depende da infra-estrutura disponível, como o desempenho e a confiabilidade dos links.

Com relação aos recursos e à direção das interações, como já ilustrado na figura 9, o processo do cliente inicia a comunicação, fazendo solicitações ao processo no servidor, que aguardam solicitações dos clientes.

Uma arquitetura orientada a serviços - SOA (do inglês: *Service- Oriented Architecture*) cliente/servidor tradicional, geralmente utiliza uma especificação que muitas vezes é exclusiva de cada sistema, o paradigma dos Dispositivos Inteligentes define uma especificação comum para todos (POSLAD, 2009).

Este paradigma revisita a abordagem cliente/servidor tradicional, da qual dependem muitas implementações de laboratórios remotos. As principais diferenças entre as implementações existentes e os *Smart Devices* são o desacoplamento completo do servidor e do cliente (GOVAERTS; SALZMANN, 2014).

Figura 12 – Arquitetura dos dispositivos inteligentes.



Fonte: Elaborado pelo autor.

A especificação de um dispositivo inteligente deve descrever de forma bem definida a forma de comunicação e suas interfaces, provendo informações suficientes pelo servidor que permitam gerar aplicativos clientes ou reutilizar aplicativos existentes com base nesta especificação. Como a especificação é comum a muitos *Smart Devices*, os aplicativos do cliente não estão acoplados a um único servidor, o que incentiva a interoperabilidade e sua reutilização.

Segundo Thompson (2005), os dispositivos inteligentes fornecem suporte para interação com o mundo real através de atuadores e sensores, estes são um tipo de transdutor que converte alguns fenômenos físicos, ou seja, calor, luz, som, temperatura e umidade, entre outros, em sinais elétricos. Frequentemente atuam como facilitadores, fornecendo informações que representam o comportamento do sistema para que ele possa se adaptar de

forma propícia, por exemplo, um sensor de temperatura ligado a um sistema de aquecimento.

Govaerts e Salzmann (2014) esclarecem que os sensores podem ser usados para diversas tarefas, tais como: monitorar ambientes; rastrear ativos através do tempo e espaço com relação a algum fluxo de trabalho ou processo; detectar mudanças definidas como significativas no ambiente, que os seres humanos são incapazes ou estão em risco para perceber.

Os sensores podem atuar como geradores de dados, registrando e armazenando tais valores. Estes representam a saída de um transdutor que converte fenômenos físicos diversos em sinais variáveis.

Os atuadores trabalham de forma inversa aos sensores, convertendo um sinal elétrico em fenômenos físicos para mover ou controlar mecanismos como motores, atuadores pneumáticos, pistões hidráulicos, relés, entre outros. Ao utilizá-los pode-se controlar um sistema com relação ao ambiente dentro de um intervalo definido de mudanças, adaptar serviços para melhorar sua utilidade.

O dispositivo inteligente não necessita fornecer uma interface do usuário - UI (do inglês: *User Interface*) , mas pode propor uma interface mínima a ser processada no lado do cliente. Isso significa que o dispositivo cliente pode renderizar interfaces de usuário de diferentes provedores, sendo os navegadores da Web os ambientes preferidos para esta renderização.

Para Govaerts e Salzmann (2014) um *Smart Device* fornece serviços e funcionalidades. Um serviço representa, por exemplo, um sensor ou um atuador que é disponibilizado por meio de uma API. Os serviços são totalmente descritos e documentados para que um cliente possa usá-los sem maiores explicações.

Uma funcionalidade é um comportamento interno fornecido pelo dispositivo inteligente, por exemplo, a existência de um serviço de um atuador que permite ao aplicativo cliente definir a tensão de um motor e uma funcionalidade que verifique se a tensão máxima não é excedida (e a corrige se necessário).

O serviço do atuador é descrito pelos metadados do *Smart Device*, por outro lado, um mecanismo interno de validação deve ser implementado para garantir que os aplicativos clientes utilizem parâmetros aceitáveis.

2.4.1 Revisão exploratória da literatura acerca do uso do paradigma de dispositivos inteligentes em Laboratórios Remotos

Em busca de trabalhos que abordam a padronização de laboratórios remotos utilizando o paradigma dos dispositivos inteligentes, uma revisão

exploratória da literatura foi realizada para essa dissertação com o intuito de buscar dados relevantes.

Deste modo, foi formulada a seguinte questão de pesquisa: *Como padronizar a comunicação de laboratórios remotos?* É necessária a definição de uma questão de pesquisa, pois segundo Freire (2013) isso permite a delimitação das fronteiras do estudo proporcionando maior exatidão nas buscas.

Com a definição da questão que norteou esta pesquisa, tomou-se os procedimentos cabíveis para identificação das variáveis a serem utilizadas no levantamento bibliográfico, identificadas tais como: "Laboratórios Remotos" e "Arquitetura Dispositivos Inteligentes", sendo utilizada a base de dados Scopus, as palavras foram traduzidas para o idioma inglês, e incluídos termos similares, ainda utilizou-se do operador "OR" para resultar o maior número de resultados possíveis, conforme figura 13.

Figura 13 – Consulta realizada na base de dados Scopus.

(TITLE-ABS-KEY (remote laboratory) OR TITLE-ABS-KEY (distributed remote laboratory) OR TITLE-ABS-KEY (online laboratory architecture) OR TITLE-ABS-KEY (learning object repository) OR TITLE-ABS-KEY (metadata for learning resources) OR TITLE-ABS-KEY (networked learning environments) OR TITLE-ABS-KEY (open educational resources)) AND (TITLE-ABS-KEY (smart device) OR TITLE-ABS-KEY (smart device architecture))

Fonte: Elaborado pelo autor.

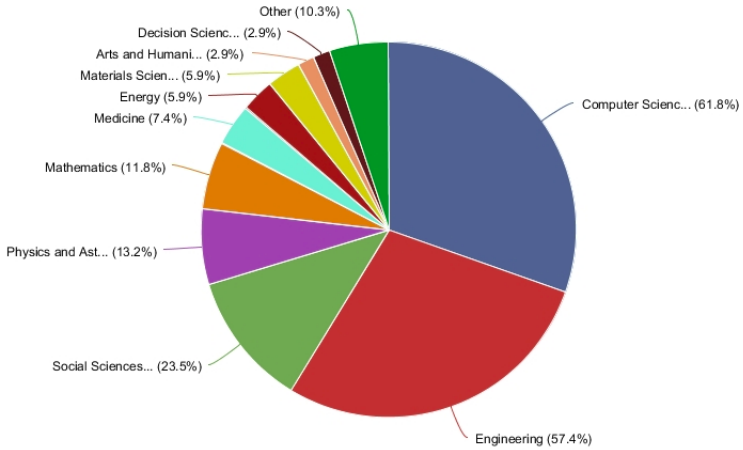
A consulta com os termos em questão, retornou 110 documentos, que deu início ao processo de seleção dos artigos utilizados na elaboração deste trabalho.

Posteriormente, delimitou-se o período de publicação, no qual foi aplicado um filtro limitando as publicações do ano 2010 ou posteriores, resultando em 78 documentos.

O filtro aplicado na sequência, foi a limitação de documentos com assuntos relacionados às áreas de: ciência da computação, engenharias ou ciências sociais, resultando em 68 publicações, ressalta-se que um documento pode estar classificado em mais de uma área de conhecimento, fato que explica a distribuição dos documentos em áreas diferentes das utilizadas nos filtros de consultas na base de dados, conforme figura 14 que apresenta a distribuição dos documentos nas áreas de conhecimento. Contudo, em ciência da computação obteve-se 42 documentos, com maior número de trabalhos, seguido por engenharias com 39 e ciências sociais com 16.

Destaca-se que, os países com maior quantidade de publicações neste conjunto de documentos são: Suíça com 9, Espanha com 8, Estados Unidos também com 8, seguidos pelo Canadá com 5 e a China com 4, o Brasil conta

Figura 14 – Áreas de conhecimento.



Fonte: Elaborado pelo autor.

com apenas 1 documento publicado.

Dos artigos, 8 autores são filiados a escola “Ecole Polytechnique Federale de Lausanne” da Suíça, 5 autores são da “Universidad Nacional de Educacion a Distancia” e 4 autores filiados a “Universidad de Murcia”, ambas da Espanha.

Após seleção dos documentos, fez-se a leitura do resumo dos artigos afim de analisar quais apresentavam o mesmo tema da pesquisa, sendo que dentre os 68 documentos foram selecionados 15 para leitura completa.

Utilizou-se os objetivos e resultados como critério para esta etapa de seleção, verificando o foco dado ao paradigma “*Smart Device*” no desenvolvimento, no entanto, após a realização deste estudo, dos 15 trabalhos, apenas 7 publicações atendiam o critério, as quais foram utilizadas como base.

Apresenta-se na tabela 1 a relação dos artigos selecionados na ordem cronológica de publicação.

Tabela 1 – Trabalhos Selecionados

Ano	Autor(es)	Título	Tipo
2012	Rekik, W.a; Mhiri, M.b; Khemakhem, M.b	A smart cloud repository for online instrument	Anais de Eventos
2013	Deaky, B.-A.	Contribution to online laboratory implementation and standardization	Anais de Eventos
2013	García, M.L.; Fernandez, G.C.; Ruiz, E.S.; Martín, A.P.; Gil, M.C.	Rethinking remote laboratories: Widgets and smart devices	Anais de Eventos
2015	Halimi, W.; Salzmann, C.; Gillet, D.	The smart wind turbine lab	Anais de Eventos
2015	Orduna, P.a; Zutin, D.G.b; Govaerts, S.c; Zorrozua, I.L.d; Bailey, P.H.e; Sancristobal, E.d; Salzmann, C.c; Rodriguez-Gil, L.a; DeLong, K.e; Gillet, D.c; Castro, M.d; Lopez-De-Ipina, D.a; Garcia-Zubia, J.a	An Extensible Architecture for the Integration of Remote and Virtual Laboratories in Public Learning Tools	Artigo
2015	Salzmann, C.; Govaerts, S.; Halimi, W.; Gillet, D.	The smart device specification for remote labs	Artigo
2016	Halimi, W.; Salzmann, C.; Gillet, D.	The Mach-Zehnder interferometer - A smart remote experiment based on a software template	Anais de Evento

Fonte: Elaborado pelo Autor.

Nesse contexto, é apresentada uma análise descritiva dos trabalhos, iniciando por Rekik, Mhiri e Khemakhem (2012), os autores tratam da criação de um repositório online inteligente para controle de acesso a experimentos remotos, utilizando o conceito da Web semântica por meio de ontologias, que em outras palavras, significa uma representação formal de um domínio. As propriedades dos instrumentos devem ser representadas em três classes: usuários, provedores de instrumentos e repositório inteligente na nuvem, responsável por manipular os acessos aos experimentos. As propriedades mínimas sobre um instrumento no modelo dos autores devem conter: URI (identificação de onde o dispositivo é acessível online), nome, proprietário, localização do serviço, tipo, status, custo de acesso e qualidade de serviço.

No trabalho intitulado "Contribution to online laboratory implementation and standardization", Deaky (2013) aborda o desafio no desenvolvimento de aplicações clientes na plataforma Android para acesso a experimentos baseados no ISA (iLab Shared Architecture), aponta dificuldades de autenticação em clientes que não são desenvolvidos com base na plataforma de navegadores Web, sendo necessário emular a interação do protocolo HTTP na aplicação, também descreve o problema de agendamento de uso de experimentos, que para serem realizados, necessitariam de um complexo processo no desenvolvimento das aplicações clientes para permitir a compatibilidade com esta função.

Deaky (2013) conclui o artigo considerando a necessidade de padronização da arquitetura e da possibilidade da utilização de métodos não baseados em navegadores, permitindo o desenvolvimento de aplicações móveis para tal fim, o autor sugere a criação de padronização como a utilizada com o formato XML pelo fato de ser simples de entendimento.

Para o autor, o formato XML não é uma obrigatoriedade, mas um modelo para discussão inicial, e o modo para envio e recebimento de requisições, pode ocorrer por meio de protocolos como: SOAP, REST ou HTTP POST.

García et al. (2013) argumentam que a maioria dos laboratórios de experimentação remota possui uma arquitetura monolítica, e são gerenciados por diversos sistemas que implementam controles de forma customizada. Os autores defendem que os laboratórios devem ter sua arquitetura repensada, e uma opção é a utilização de uma arquitetura como a SOA, que permite entradas e saídas de dados de forma padronizada, pois um grande desafio na integração de sistemas de laboratórios remotos encontra-se na forma particular em que cada sistema troca informações. A adoção do paradigma "*Smart Device*" pode eliminar este problema crítico de comunicação entre dispositivos e interfaces clientes.

Segundo García et al. (2013), no modo de operação dos experimentos, os elementos fundamentais são: entradas (comandos, valores, informação de

usuário, seleção de componentes) e saídas (medições e resultados).

Um dos elementos fundamentais para os autores é a decomposição dos laboratórios em serviços divididos em: *Authentication and authorization, Discover, Learning analytics, Metadata, Scheduler and reservation e Storage* que possuem os seguintes propósitos:

- *Authentication and authorization*: devem controlar se um usuário é quem afirma ser e verificar quais funcionalidades está autorizado a utilizar;

- *Discover*: deve ser utilizado para que anuncie sua existência e encontre outros laboratórios;

- *Metadata*: os metadados devem descrever o laboratório, seu funcionamento e seus recursos, isto facilita sua inclusão em repositórios de experimentos remotos, relacionar seus recursos à pesquisas realizadas facilitando que usuários encontrem o laboratório;

- *Learning analytics*: funcionalidade que permite analisar as informações acerca dos passos realizados pelo usuário no decorrer da utilização do laboratório;

- *Schedule and reservation*: deve limitar o número de usuários acessando o experimento conforme sua capacidade, e organizar filas virtuais dos usuários excedentes para acesso ao laboratório assim que disponível;

- *Storage*: este serviço deve proporcionar um espaço de armazenamento para dados estáticos tais como tutoriais, diagramas e figuras, afim de facilitar a integração com RLMS e portais Web;

Nem todos os laboratórios existentes podem ser decompostos desta maneira. Os autores concluem que a arquitetura de dispositivos inteligentes fornece diretrizes chaves para comunicação de dados nos laboratórios.

O trabalho de Halimi, Salzmann e Gillet (2015) descreve o desenvolvimento de um experimento remoto para estudo da geração de energia eólica, já desenvolvido obedecendo ao paradigma "*Smart Device*", que permite sua utilização independente do sistema de gerenciamento de laboratório ou aplicativo cliente específico. O acesso aos dados pode ser realizado por meio de objetos JSON. A comunicação entre o software cliente e o software do servidor é realizada por meio de WebSockets.[16]

[16]Os metadados utilizados neste experimento podem ser encontrados em `https://github.com/go-lab/smart-device/tree/master/myRIO/wind-turbine-interplay/metadata`.

Orduña et al. (2015) abordam uma arquitetura para integração de laboratórios e defendem a utilização de uma federação para interoperabilidade entre os RLMS. O Gateway4labs é um *middleware* [17] que promove a integração de múltiplos laboratórios, permitindo que outros sistemas utilizem o Go-Lab *(Global Online Science Labs for Inquiry Learning at School)*, no qual está utilizando o paradigma "*Smart Device*". Atualmente permite a integração com laboratórios que ainda não utilizam este conceito utilizando o *plugin* gateway4labs[18] denominando como integração leve, e também a integração completa utilizando um protocolo de tradução entre o laboratório e o *Smart Device*. Os autores consideram que é possível a integração dos diversos RLMS em uma federação e que enquanto não houver a padronização dos laboratórios no paradigma *Smart Device* a migração dependerá de protocolos de tradução.

No artigo "The smart device specification for remote labs" Salzmann et al. (2015) apresentam uma visão geral sobre a arquitetura dos "*Smart Devices*" para laboratórios remotos, discutem a interoperabilidade proporcionada com a adoção deste modelo, e ainda descrevem as especificações em detalhes e exemplos para sua utilização. Os autores ainda destacam que a transferência dos metadados pode ser realizada utilizando o protocolo HTTP por meio do método GET, já os demais dados devem ser transmitidos e recebidos por intermédio do protocolo WebSocket.

Reforçando direcionamento do trabalho, Orduña et al. (2015) informam que as principais plataformas utilizadas na construção de experimentos como LabVIEW, e JavaScript (Node.js e Socket.IO) permitem a implantação deste conceito, em outras situações um *smart gateway* pode ser utilizado como tradutor.

Halimi, Salzmann e Gillet (2016) apresentam uma proposta de utilização de "Software *Template*" para desenvolvimento de laboratórios utilizando o paradigma "*Smart Devices*". De forma simplista, o *template* (modelo) é dividido em dois módulos: API de serviços e módulo controle de processo.

O uso destes *templetes*, proporcionam a elaboração de um "esqueleto" modelo da aplicação no servidor do laboratório remoto obedecendo esta arquitetura, os metadados podem ser adaptados pelo proprietário do laboratório. Fica fora da responsabilidade deste *template* a implementação da interface de controle do hardware em questão. Halimi, Salzmann e Gillet (2016) argumentam em suas considerações que o uso de *templates* gera economia de tempo e esforço no desenvolvimento de laboratórios na arquitetura *Smart Device*.

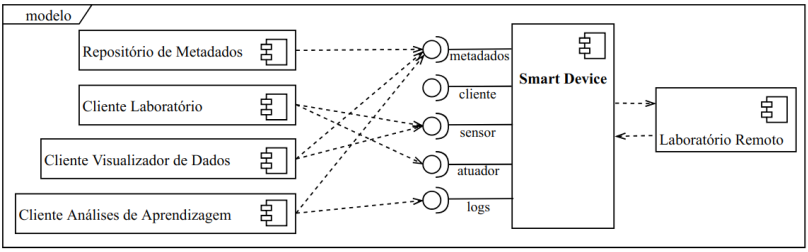
[17] Camada de software intermediária em sistemas distribuídos.

[18] Projeto que visa a integração de múltiplos laboratórios remotos em diferentes ambientes de aprendizagem digital.

2.4.2 Arquitetura de dispositivos inteligentes para laboratórios remotos

Salzmann et al. (2015) descrevem que o *Smart Device* deve prover um conjunto de interfaces bem definidas, que permitam a comunicação entre o laboratório remoto, serviços e aplicações externas. A figura 15 ilustra um diagrama básico da arquitetura com cenários de interação, no entanto a especificação não define como deve ser a comunicação entre o servidor e o equipamento físico do laboratório remoto.

Figura 15 – Diagrama de componentes dos serviços mais comuns de dispositivos inteligentes.



Fonte: Adaptado de Salzmann et al. (2015).

As interfaces (metadados, cliente, sensor, atuador e logs) devem fornecer especificações detalhadas sobre formatos de dados e os protocolos necessários para comunicação com os serviços, desta forma permite um desacoplamento entre o cliente e servidor.

No que se refere ao conceito dos termos *sensores* e *atuadores*, eles refletem a direção do deslocamento da informação em relação ao dispositivo inteligente. Por exemplo, um sensor permite a leitura de um termômetro, já um atuador permite a configuração de um valor, por exemplo, definindo uma tensão de um motor (SALZMANN et al., 2015).

Os sensores e atuadores podem ser físicos (temperatura), virtuais (cálculo de posição ou velocidade) ou complexos (agregação de sensor/atuador, ex. acelerômetro 3D), deste modo, os sensores (alguns) e atuadores podem ser configurados.

As próximas subseções discorrem sobre as especificações dos serviços e funcionalidades.

2.4.2.1 Serviço de Metadados

O serviço de metadados é obrigatório e está no coração da interoperabilidade fornecida pela especificação dos dispositivos inteligentes, ele fornece informações a respeito do dispositivo, Salzmann et al. (2015) descrevem quais dados e como devem ser providos:

- Descrição do laboratório, contendo seus objetivos e proprietário, essas informações podem ser utilizadas na indexação automática em ferramentas de pesquisa;
- Descrição da integração com serviços externos tais como autenticação e agendamento;
- Descrição do mecanismo de concorrência (por exemplo, pode ser baseado em fila ou perfil do usuário, ainda informação sobre a possibilidade de visualização do experimento com o perfil de observador enquanto outros usuários estão manipulando o experimento);
- Descrição e definição dos serviços providos (especificação sobre requisição de serviços e formatos de respostas).

Os metadados podem ser obtidos por meio do método GET do protocolo HTTP no diretório raiz do experimento (Apêndice A - Seção getMetadata).

2.4.2.2 Serviço de Aplicação Cliente - $getClients$

É um serviço opcional que provê links para aplicações clientes caso existam, permitindo operar o *Smart Device*. Sua requisição deve ser realizada por intermédio do método $getClients$ (figura 16).

Cada elemento da lista *clients* contém um *type* e uma URL, o tipo correspondente a tecnologia do aplicativo, que pode ser: OpenSocial Gadget, W3C widget, Web page, Java WebStart e Desktop application, e a URL informa o local para seu respectivo acesso.

2.4.2.3 Serviço de Metadados de Sensores - $getSensorMetadata$

A chamada do serviço de metadados dos sensores pode ser realizada pelo método $getSensorMetadata$ em um objeto JSON, por exemplo {"method":

Figura 16 – Exemplo requisição e resposta do serviço *getClients*.

```
{
  "method": "getClients"
}
```

```
{
  "method": "getClients",
  "clients":
  {
    "type": "Web page",
    "url": "http://relle.ufsc.br/labs/7/moodle"
  }
}
```

Fonte: Adaptado de Govaerts e Salzmann (2014).

"getSensorMetadata"}, pode ser necessário a utilização de um *token* de autenticação em alguns casos. Este serviço retorna um *array* descrevendo cada sensor disponível, como especificado por Govaerts e Salzmann (2014) contendo os seguintes itens:

- *sensorId*: para identificar o sensor, por exemplo: “3d-acc”;
- *fullName*: nome completo, por exemplo: “3D acceleration”;
- *description*: descrição, por exemplo: “The robot arm 3d acceleration”;
- *WebSocketType*: tipo do WebSocket, pode ser do tipo *text* que é o padrão, ou *binary* para vídeo por exemplo;
- *produces*: tipo de mídia que o serviço do sensor produz, tipicamente é *application/json* para uma resposta JSON, no entanto, pode ser um tipo de mídia de Internet como *image/jpeg* por exemplo;
- *values[]*: poderá conter um único valor para um sensor simples como um termômetro, mas para um sensor que necessita de mais valores para representar seu estado, como um acelerômetro, o *array* contém, por exemplo, 3 elementos para a aceleração X-Y-Z, sendo então denominado sensor complexo. Os valores são descritos com um nome e uma unidade, opcionalmente, o tempo do último registro, um intervalo mínimo e máximo de valores produzidos pelo sensor. Além disso, para valores medidos continuamente, a frequência com que é atualizada pode ser fornecida em *Hertz*;

- *configuration[]*: pode ser usado para ajustar um sensor, cada parâmetro tem um nome e tipo de dados, pode ser um esquema de dados primitivo JSON, um *array* ou modelo de dados para parâmetros complexos, a qualidade de vídeo e resolução de uma câmera é um exemplo de sensor que pode ser configurado;
- *accessMode*: descreve como o sensor pode ser acessado, por exemplo, alguns sensores podem realizar medições uma vez, enquanto outros fornecem um fluxo contínuo de dados.

O exemplo ilustrado na figura 17 apresenta uma resposta parcial do serviço $getSensorMetadata$, de um *Smart Device* que possui um $value$ complexo contendo 3 valores, foram suprimidos os itens $configuration[]$ e $accessMode$.

Figura 17 – Exemplo de resposta do serviço *getSensorMetadata*.

```
{
  "method": "getSensorMetadata",
  "sensors": [
  {
    "sensorId": "3Dacc",
    "fullName": "3D acceleration",
    "description": "the 3D acceleration of the robot handle
        ",
    "webSocketType": "text",
    "produces": "application/json",
    "values": [
      {
      "name": "X",
      "unit": "m*s^-2",
      "lastMeasured": "2014-06-23T18:25:43.511Z",
      "rangeMinimum": -100.00,
      "rangeMaximum": 100.00,
      "rangeStep": 0.10,
      "updateFrequency": 10 /* in Hertz */
      },
      /* Repeat for 'Y' and 'Z' acceleration */
      {
      "name": "Y",
      ...
      },
      {
      "name": "Z",
      }
    ]
```

Fonte: Govaerts e Salzmann (2014).

2.4.2.4 Serviço de Sensores - $getSensorData$

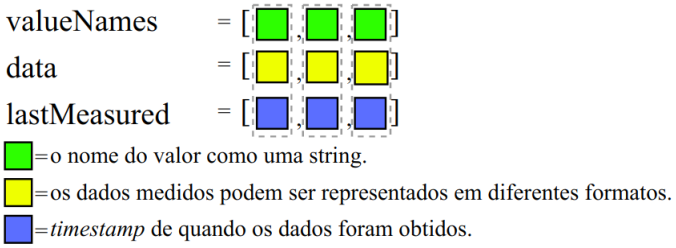
Salzmann et al. (2015) salientam que os serviços $getSensorData$ e $sendActuatorData$ estão no centro da interação dos dispositivos inteligentes e ambos são semelhantes. Os serviços em combinação com os metadados permitem que os desenvolvedores promovam a reutilização de aplicativos e interoperabilidade. Da mesma forma, diferentes aplicativos poderiam ser desenvolvidos para um dispositivo inteligente.

Existem diferentes tipos de sensores e atuadores:

- Real: representa um sensor físico, por exemplo, um termômetro;
- Virtual: representa um sensor que apresenta um valor calculado, por exemplo, uma medição de velocidade derivada da medição de uma posição;
- Complexo: representa um conjunto de sensor/atuadores, por exemplo, botões na frente do painel de um osciloscópico.

A estrutura de dados retornada por um sensor ou enviada a um atuador pode variar dependendo do número de valores e das estruturas de medição de dados, pode ser um valor, um *array*, um *array* de valores agregados ou ainda uma estrutura complexa de dados (figura 18). Sua forma é semelhante a figura 17.

Figura 18 – Estrutura de dados dos sensores e atuadores.



Fonte: Traduzido de Salzmann et al. (2015).

A figura 19 apresenta um exemplo de requisição e resposta do serviço $getSensorData$. Na linha 2, a chave $authToken$ é utilizada quando é requerido validar o *token* para envio de configurações aos sensores, a chave $accessRole$ na linha 4, possui valor $controller$, indicando que podem ser realizadas modificações, outro valor possível para esta chave é $observer$, que

pode ser usado por exemplo, na requisição de um sensor como uma câmera para visualizar o experimento enquanto outro usuário está utilizando.

Na linha 5 consta a identificação do sensor, na 6, a frequência com que o *Smart Device* deve realizar novas leituras do sensor e enviar para o cliente, no *array* $configuration$, o parâmetro $precision$ indica para o servidor que a leitura dos valores do sensor deve utilizar precisão com 2 casas decimais.

Figura 19 – Exemplo requisição e resposta do serviço *getSensorData*.

```
1  {
2    "authToken": "dskds909ds8a76as675sa54",
3    "method": "getSensorData",
4    "accessRole": "controller",
5    "sensorId": "3D-pos",
6    "updateFrequency": 20,
7    "configuration": [
8      {
9      "parameter": "precision",
10     "value": 2
11     }
12   ]
13 }
```

```
1  {
2    "method": "getSensorData",
3    "sensorId": "3D-pos",
4    "accessRole": "controller",
5    "responseData": {
6      "valueNames": ["X", "Y", "Z"],
7      "data": [12.37, 23.51, 43.18],
8      "lastMeasured": [
9        "2014-06-23T18:28:43.511Z",
10       "2014-06-23T18:28:43.511Z",
11       "2014-06-23T18:28:43.511Z"
12     ]
13   }
14 }
```

Fonte: Govaerts e Salzmann (2014).

O exemplo de resposta, ainda na figura 19, retorna os dados na forma de matriz, portanto, o dispositivo inteligente colocará o nome dos sensores ou atuadores no *array* $valueName$, os valores de medição no *array* $data$ e opcionalmente a data/hora de última medição no *array* $lastMeasured$, no entanto, sempre no mesmo índice do *array* $valueNames$.

O entendimento da reposta da requisição fica mais claro ao comparar a estrutura de dados representada na figura 18 com a resposta da figura 19,

dessa forma tem-se:

indice 0 "X" tem valor 12.37 no momento "2014-06-23T18:28:43.511Z";

indice 1 "Y" tem valor 23.51 no momento "2014-06-23T18:28:43.511Z";

indice 2 "Z" tem valor 43.18 no momento "2014-06-23T18:28:43.511Z".

A requisição $getSensorData$ é mais complexa de ser administrada pelo servidor, pois, pode ser necessário utilizar um mecanismo de autenticação, controle de concorrência, validação de parâmetros de configuração ou ainda de acordo com o papel do usuário (observador ou controlador), alguns destes itens são abordados na subseção funcionalidades.

2.4.2.5 Serviço de Metadados de Atuadores - $getActuatorMetadata$

Este serviço também pode ser obtido especificando o valor do *method* em um objeto JSON {"method":"getActuatorMetadata"}, sendo este um serviço muito similar ao $getSensorMetadata$.

Salzmann et al. (2015) explicam que nestes metadados contém informações sobre especificações de configuração que um usuário pode enviar para um atuador, tais como: unidade de medida, valor mínimo, máximo, passo de intervalo e precisão da medida, o JSON é o formato para envio, mas pode ser definido outro tipo de mídia da Internet.

2.4.2.6 Serviço de Atuadores - $sendActuatorData$

Este serviço é muito parecido com o de sensores, a maioria dos campos são similares, a principal diferença é o fato do método $sendActuatorData$ permitir que o usuário realmente defina um valor de configuração para o atuador desejado (figura 20).

Desta forma, a linha 5 do exemplo informa à qual atuador a mensagem é destinada $motor$, na linha 6 o comando $left$ e na linha 7 o valor 17.90 que representa quantos graus o motor deve girar.

Funcionalidades internas devem ser implementadas para validação dos dados recebidos, realizar o controle de concorrência e o controle de interrupções dos atuadores.

Figura 20 – Exemplo de envio de configuração pelo método *sendActuatorData*.

```
{
  "authToken": "dskds909ds8a76as675sa54",
  "method": "sendActuatorData",
  "accessRole": "controller",
  "actuatorId": "motor",
  "valueNames": ["left"],
  "data": [17.90]
}
```

Fonte: Govaerts e Salzmann (2014).

2.4.2.7 Serviços opcionais

- Serviço de Logs de Atividades do Usuário - $getLoggingInfo$

Salzmann et al. (2015) descrevem que o serviço $getLoggingInfo$ deve retornar o registro das ações do usuário no laboratório. O formato dos registros deve descrever a sequência de ações do usuário com um registro de data e hora no formato ActivityStream[19].

A chamada desse serviço, opcionalmente, pode solicitar a utilização de um *token* de autenticação (para proteção dos dados ou diminuir o número de requisições indevidas).

- Serviço de Modelos - $getModels$

Pode fornecer vários modelos do laboratório físico (ou seja, a instrumentação) e seu histórico teórico. Por exemplo, um modelo gráfico 3D da instrumentação de laboratório pode permitir que um aplicativo cliente gere uma GUI (Interface Gráfica do Usuário) com um objeto de escala 3D que os alunos possam manipular.

Com um modelo matemático do experimento, um aplicativo cliente pode ser construído com uma simulação local. Isso pode fornecer uma versão simulada interativa de um laboratório remoto quando estiver sendo utilizado por outro usuário.

Devido à grande variedade de formatos existentes para expressar modelos práticos e teóricos (por exemplo, VRML, X3D e MathML), os autores não limitam a especificação e deixam a escolha do idioma do modelo para o proprietário do laboratório (SALZMANN et al., 2015).

[19] A especificação do ActivityStream está disponível em: http://activitystrea.ms/specs/json/1.0/

2.4.2.8 Funcionalidades

Quando o laboratório remoto está disponível pela Internet, os autores Salzmann et al. (2015) recomendam fortemente que funcionalidades internas sejam implementadas para um funcionamento adequado e para proteção do experimento.

(a) Autenticação.

O dispositivo inteligente não precisa conter um sistema de reserva/autenticação, pode utilizar um sistema de reservas externo, como o do laboratório Go-Lab, o dispositivo inteligente entra em contato com o sistema de reservas para verificar se o usuário está atualmente autorizado a acessá-lo, deste modo, a integração do serviço de reservas requer pouco esforço em comparação ao fornecimento de seus próprios mecanismos de autenticação e reserva.

(b) Estado automático e conhecido.

A implementação desta funcionalidade é recomendada, no entanto, é deixada ao critério dos proprietários dos laboratórios. Ela busca garantir que o laboratório remoto seja redefinido para um estado adequado após a conclusão de uma sessão ou de uma interrupção do sistema, de modo que o próximo usuário possa utilizá-lo (SALZMANN et al., 2015).

Assim, o sistema deve ser o mais autônomo possível, sendo capaz de se adaptar a qualquer situação, Salzmann et al. (2015) sugerem implementar os seguintes procedimentos: (1) inicialização automática ao ligar, (2) redefinição para um estado conhecido após o último cliente desconectar, e (3) potencial para calibração automática de hardware.

(c) Segurança e controle local.

Funcionalidade a critério dos proprietários, no entanto, em todos os momentos, a segurança do servidor e equipamentos conectados deve ser assegurada. Todos os comandos devem ser validados antes de serem encaminhados.

Os usuários geralmente tentam levar o laboratório aos limites, ou seja, ao limite físico de um determinado sensor/atuador como na intensidade de utilização. Como os atuadores podem estar conectados à Internet, Govaerts e Salzmann (2014) recomendam: (1) validação dos valores antes de aplicá-los aos atuadores, e (2) validação do estado do atuador para verificar se o comando a ser aplicado é seguro.

(d) Logs e alarmes.

Essas funcionalidades devem registrar informações sobre a sessão e interações do usuário e do estado do laboratório. Em caso de problemas, alarmes podem ser acionados automaticamente. Como um dispositivo inteligente normalmente ficará sem supervisionamento físico por um longo período de tempo, é essencial monitorá-lo e ter um método para realizar a análise após algum evento que causou problemas.

As ações dos usuários devem ser registradas, e podem ser acessíveis por meio do serviço de registro de atividades. Informações extras também devem ser registradas, ou seja, o estado do sistema e do ambiente (por exemplo, temperatura ambiente).

Alguns sensores podem estar disponíveis internamente ao dispositivo inteligente, mas não necessariamente acessíveis através do serviço de sensores. As seguintes informações devem ser registradas: (1) ações do usuário, (2) o estado completo do sistema e (3) seu status ambiental.

A necessidade de padronização no modo de comunicação dos laboratórios é discutido de forma crescente no decorrer do tempo, como apontado pela revisão exploratória, este trabalho realiza a implementação detalhada da arquitetura de dispositivos inteligentes, apresentando-a detalhadamente.

Os trabalhos que apresentam tais propostas, não focam nos detalhes de implementação nem à compatibilidade do padrão com plataformas que não sejam Web.

Outro ponto está na utilização da biblioteca Socket.IO para fornecer um servidor WebSocket. Ela apresenta diversas vantagens como compatibilidade com navegadores antigos, facilidade no envio e recebimento de mensagem podendo identificá-las, facilitando o tratamento dos dados e direcionando diretamente para funções específicas.

No entanto, ela não permite que seja utilizada a API padrão dos navegadores Web, sendo necessário utilizar bibliotecas fornecidas pela Socket.IO na aplicação cliente, isto dificulta ou inviabiliza o desenvolvimento de softwares clientes em linguagens que não são Web.

3 PROCEDIMENTOS METODOLÓGICOS

Neste capítulo são descritos os procedimentos estabelecidos para a elaboração e desenvolvimento deste estudo, assim como a classificação desta pesquisa.

3.1 CLASSIFICAÇÃO DA PESQUISA

A realização de pesquisa torna-se necessária quando um problema é apresentado, para Gil (2008) pesquisas são realizadas com o objetivo de buscar respostas quando não há dados suficientes para responder aos questionamentos apresentados.

O presente trabalho classifica-se como pesquisa tecnológica, que de acordo com Cupani (2006) objetiva a "solução de problemas específicos e pontuais, tendo foco no artefato a ser desenvolvido", que pode ser um produto, uma modificação de um sistema natural por meio artificial ou mesmo social.

Os desenvolvimento científico e tecnológico devem seguir caminhos paralelos, pois o sucesso de um possibilita o aperfeiçoamento do outro.

Gil (2008) aponta quatro aspectos indispensáveis no qual uma pesquisa precisa ser classificada que são: (a) natureza; (b) abordagem; (c) objetivos; e (d) procedimentos técnicos, assim, a classificação deste estudo é ilustrada na tabela 2.

Tabela 2 – Classificação da pesquisa

Classificação	**Metodologia utilizada**
Natureza	Pesquisa aplicada
Abordagem	Qualitativa
Objetivos	Exploratória
Procedimentos	*Design Science Research*

Fonte: Elaborado pelo autor.

Quanto sua natureza, apesar deste estudo possuir elementos da pesquisa básica, que objetiva gerar novos conhecimentos, sendo úteis para o avanço da ciência, sua característica é predominantemente aplicada, que nas palavras de Filho e Filho (2013, p. 62) "tem seus resultados voltados à aplicação prática".

Tratando da abordagem, Gerhardt (2009, p. 32) diz que "a pesquisa

qualitativa preocupa-se, portanto, com aspectos da realidade que não podem ser quantificados, centrando-se na compreensão e explicação da dinâmica das relações sociais".

Assim, mesmo que este estudo apresente dados como a quantidade de publicações no tema pesquisado, que poderia caracterizar uma abordagem quantitativa, seu enfoque é qualitativo, na qual fez-se uma análise sobre os principais apontamentos das obras, ou seja, são dados que não podem ser quantificados.

Já quanto aos objetivos, a pesquisa exploratória possui como objetivo tornar o problema mais familiar, tornando-o mais explícito ao pesquisador, Gil (2008) e Freire (2013) explicam que esse tipo de pesquisa procura deixar mais claro os fatos e fenômenos relacionados.

Deste modo, quanto aos seus objetivos, este trabalho é classificado como exploratória, pois torna o problema de pesquisa mais explícito.

Quanto aos procedimentos, utilizou-se a *Design Science Research Methodology* (DSRM) que é composta por seis etapas, cada uma compreendendo uma fase do desenvolvimento do trabalho.

Conforme Freitas Junior et al. (2017) a DSRM possui seu foco central nas pesquisas tecnológicas, amparando todas as etapas de seu desenvolvimento, desde a sua concepção até o processo de comunicação de seus resultados.

3.2 ETAPAS DA PESQUISA

No que tange os delineamentos metodológicos empregados no desenvolvimento deste trabalho, a figura 21 ilustra a correlação entre as etapas da DSRM e os capítulos.

Na primeira etapa, foi identificado o problema de pesquisa e sua motivação, que são apresentados nos capítulos um e dois.

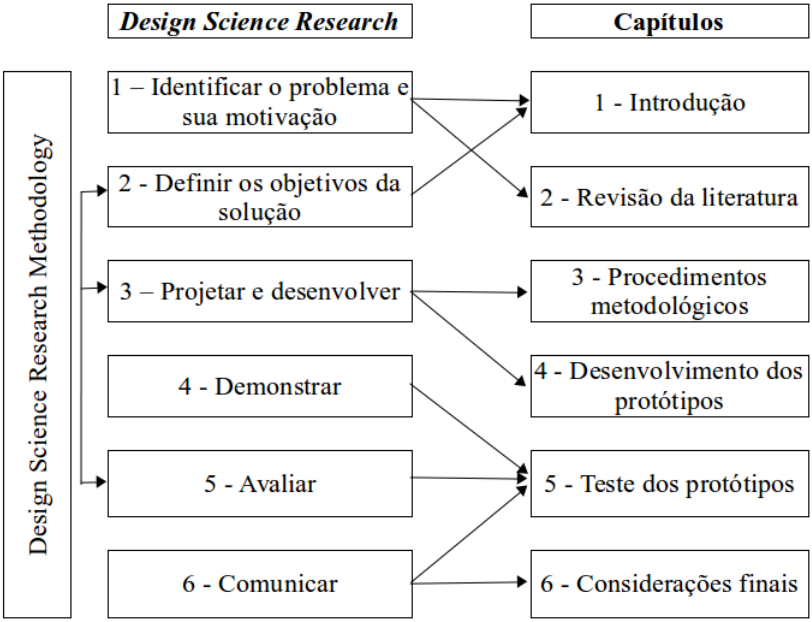
Buscou-se na literatura fundamentos acerca do uso de tecnologia e laboratórios na educação, em sequência, trabalhos relacionados a padronização da comunicação dos laboratórios e tecnologias computacionais empregadas nesta tarefa.

Para a segunda etapa, a definição do objetivo é apresentada por meio dos objetivos gerais e específicos.

A etapa número três, apresenta-se por meio dos capítulos três e quatro, que após o levantamento das tecnologias e padrões, iniciou-se o desenvolvimento dos protótipos utilizando a arquitetura de dispositivos inteligentes.

O capítulo cinco da dissertação, compreende as etapas quatro, cinco e parte seis da metodologia.

Figura 21 – Procedimentos metodológicos e relação com capítulos.



Fonte: Elaborado pelo autor.

A etapa de avaliação pode ser realizada de cinco forma diferentes: observacional, analítica, experimental, teste ou descritivo.

Neste trabalho foi utilizado o método descritivo por meio de cenários, que constitui na criação de cenários detalhados em torno do artefato, para demonstrar sua utilidade.

Ainda na etapa de avaliação, deve-se observar e mensurar como os protótipos atendem à solução do problema, comparando-o aos objetivos propostos, pode-se definir pela recursividade da metodologia, isto é, o retorno às etapas 3 ou 4, de modo a aprimorar os protótipos.

Como parte final, a comunicação é realizada com a apresentação dos resultados obtidos e as considerações finais.

4 DESENVOLVIMENTO DOS PROTÓTIPOS

Para elaboração do protótipo foi utilizado um experimento já existente e em funcionamento no Laboratório de Experimentação Remota - RExLab[1], que surgiu em 1997, na Universidade Federal de Santa Catarina (UFSC), em que um de seus objetivos é atender a necessidade de apropriação social da ciência e da tecnologia, popularizando conhecimentos científicos e tecnológicos, estimulando os jovens a inserirem-se nas carreiras científico-tecnológicas e buscar iniciativas que integrem a educação científica ao processo educacional.

Atualmente são disponibilizados 17 experimentos remotos, que podem ser acessados gratuitamente por meio da Internet[2], além dos experimentos, cadernos com sequências didáticas são disponibilizados para apoio ao uso de cada experimento.

Dentre os experimentos, o plano inclinado tem como objetivo auxiliar os estudantes do Ensino Médio e do Ensino Superior a efetuar práticas relacionadas ao MRU (movimento retilíneo uniforme), MRUV (movimento retilíneo uniformemente variado), velocidade, aceleração e queda livre (ao inclinar o plano à 90 graus, a esfera, ao ser solta, realiza uma queda livre).

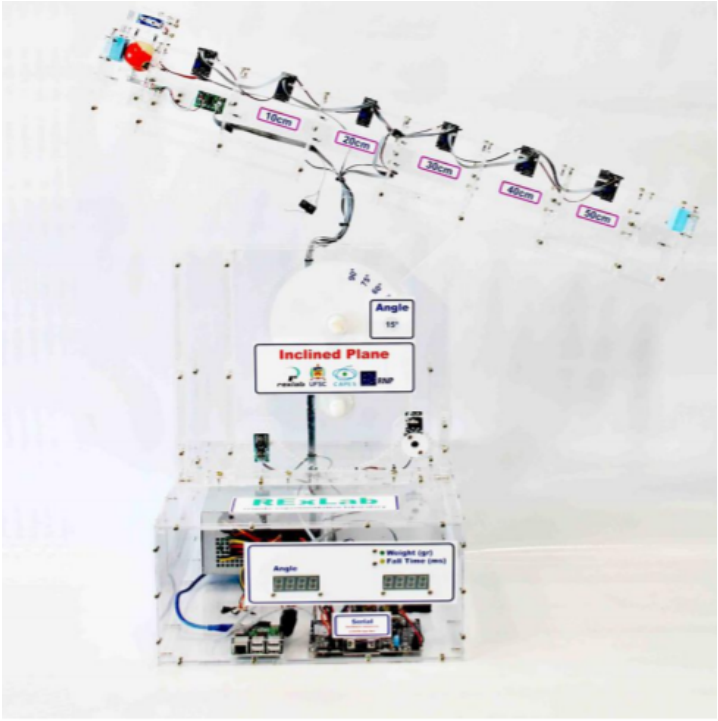
O plano inclinado, (figura 22) consiste de uma gangorra de acrílico, uma esfera, uma trava para esfera, sensores, um *display* que mostra o ângulo de inclinação da gangorra e um *display* que informa o peso da esfera ou o tempo total gasto para percorrer a extensão da gangorra. O ângulo de inclinação da gangorra pode ser ajustado pelos usuários, deste modo pode-se visualizar a movimentação da esfera a partir do ângulo desejado.

A escolha deste experimento para implementação do paradigma de dispositivos inteligentes, ocorreu pelo fato deste contar com diversos sensores e atuadores. Alguns dos experimentos disponíveis, como por exemplo, os painéis elétricos CC (corrente contínua) e CA (corrente alternada), possuem como atuadores chaves que alteram a configuração do circuito elétrico, no entanto, como sensor, apresentam apenas o fluxo de vídeo para observação dos *displays* com valores resultantes. Deste modo, a escolha de um experimento que possua diversos sensores e atuadores, demanda uma implementação mais ampla, com maior número de variáveis.

[1] Sigla oriunda da expressão em inglês - *Remote Experimentation Lab*, pode acessado em: <www.rexlab.ufsc.br>

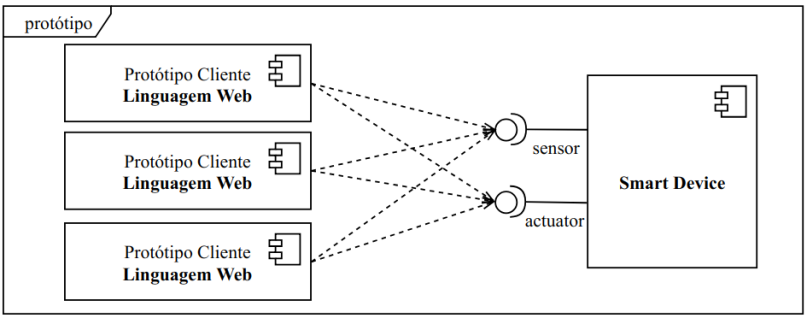
[2] http://relle.ufsc.br/labs

Figura 22 – Plano Inclinado



Fonte: Silva et al. (2016) .

Figura 23 – Protótipo.

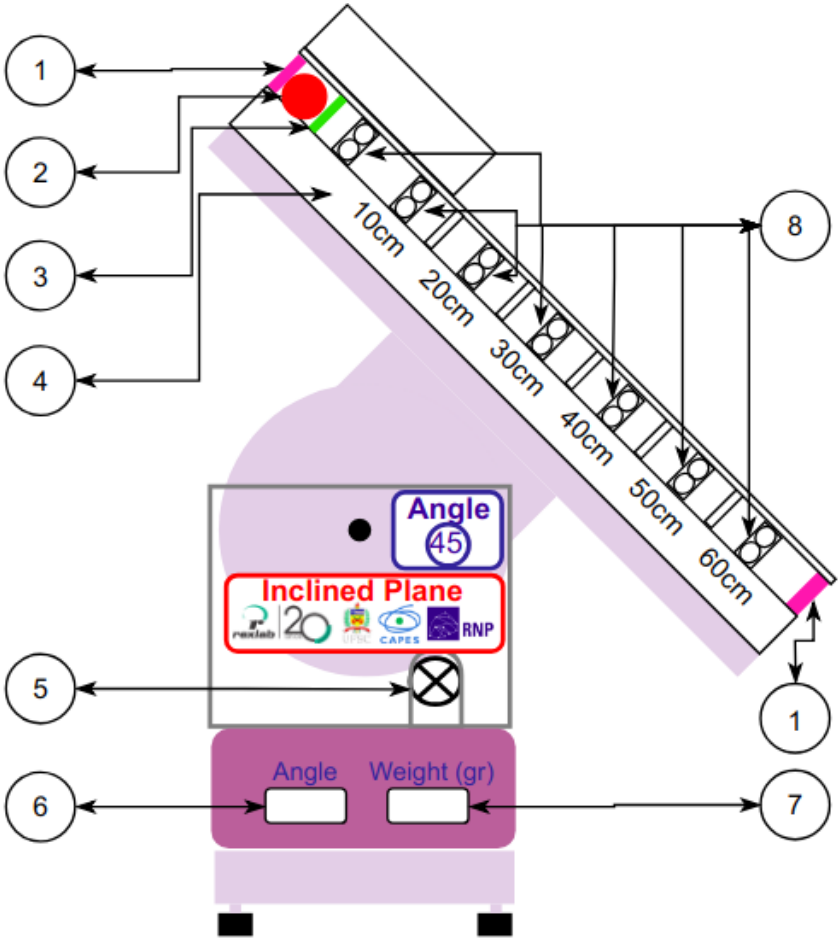


Fonte: Elaborado pelo autor.

Pela especificação, um *Smart Device* possui serviços providos pelos sensores e atuadores e funcionalidades, no entanto, alguns sensores ou atuadores podem ser utilizados apenas pelo próprio experimento para funcionalidades internas, assim, não ficam disponíveis como um serviço acessível externamente.

O diagrama apresentado na figura 24 indica os principais componentes do experimento.

Figura 24 – Diagrama do Plano Inclinado.



Fonte: Elaborado pelo autor.

Cabe destacar que fora do diagrama existe uma câmera que transmite o fluxo de vídeo para observar o experimento, deste modo pode-se listar os seguintes itens:

1. Anteparos, para amortecer o impacto da esfera;
2. Esfera com diâmetro de 38mm e 33 gramas de peso.
3. Sensor de peso e trava da esfera (servo motor);
4. Canaleta de acrílico;
5. Mecanismo de movimentação da gangorra (servo motor e sensor de inclinação);
6. Visor indicador de inclinação do plano;
7. Visor indicador do peso (em gramas) ou tempo da queda (ms) da esfera;
8. Sensores infravermelhos que registram o tempo de passagem da esfera em 0, 10, 20, 30, 40, 50 e 60cm, denominados: d0, d1, d2, d3, d4, d5 e d6.

No decorrer da elaboração do protótipo, o primeiro passo foi a criação dos metadados do laboratório, sua estrutura foi construída baseada no modelo apresentado no trabalho de Govaerts e Salzmann (2014) e possui: (a) versão, (b) *basePath* (URL ou endereço IP) para acesso ao laboratório, (c) informações, (d) mecanismos de autorização, (e) mecanismos de concorrência, e (f) APIs, que possuem informações sobre as interfaces clientes, sensores, atuadores e logs do dispositivo inteligente.

Atualmente, a API clientes fornece o endereço atual do laboratório, com a versão cliente desenvolvida para o trabalho, já a API Logs não foi implementada.

A interface de sensores fornece dois mecanismos de interação, que são utilizados por meio dos métodos $getSensorMetadata$ e $getSensorData$. O método $getSensorMetadata$ fornece informações a respeito dos sensores disponíveis, $time$, $angle$ e $video$, já o método $getSensorData$ possibilita que o cliente receba o valor resultante da medição dos sensores, no entanto, na requisição deve ser indicado qual sensor deseja-se obter as medições na requisição realizada.

A interface dos atuadores possui os métodos $getActuatorMetadata$ e $sendActuatorData$, o primeiro, resgata informações sobre os atuadores disponíveis, e o segundo possibilita o envio de dados para configuração dos atuadores, que neste experimento são: (a) $angle$ - atuador que realiza a inclinação do plano, (b) $drop$ - atuador que prende e solta a esfera e (c) $setup$.

O tipo de dado e o intervalo permitido são fornecidos por meio dos metadados, no entanto, recomenda-se que sejam validados no servidor para garantir a segurança do *Smart Device*.

Para que a esfera fique presa no compartimento na extremidade esquerda do plano, é necessário inclinar a gangorra à -15 graus, de forma que fique parada e o servo motor posicione a trava na posição fechada. No intuito de facilitar a operação do experimento por parte dos usuários, uma funcionalidade interna no plano faz o procedimento, sendo apresentado como o atuador $setup$, ao ser acionado, ele inclina a gangorra para o ângulo inicial -15 e prende a esfera, à partir disso o usuário pode inserir um valor para o ângulo de inclinação desejado e enviar para o atuador $angle$.

Ao finalizar a inclinação da gangorra no ângulo solicitado, o experimento está pronto para soltar a esfera e registrar o tempo de queda, assim, o usuário aciona o atuador $drop$ para soltar a esfera, o dispositivo verifica se a inclinação esta dentro do permitido (se o sensor $angle$ for maior que 0 e menor que 90) e abre a trava. Os sensores registram o tempo de passagem da esfera e disponibiliza ao usuário por meio do método $getSensorData$ solicitando o sensor $time$.

4.1 INTERAÇÃO POR MEIO DE WEBSOCKETS

A interação com o experimento para envio e recebimento de requisições por meio de WebSockets, pode ser realizada com uma ou várias conexões, a figura 25 ilustra o uso de uma única conexão.

A explicação detalhada do processo de comunicação apresenta-se da seguinte forma:

1 O usuário acessa a interface do software cliente;

1.1 O Software cliente cria uma conexão WebSocket com o *Smart Device*;

1.2 O Software cliente prepara a mensagem e envia por meio do WebSocket uma requisição solicitando informações do sensor $angle$;

1.2.1 O WebSocket transporta a mensagem ao *Smart Device*, que recebe, processa e responde a requisição enviando-a pelo WebSocket ao software cliente;

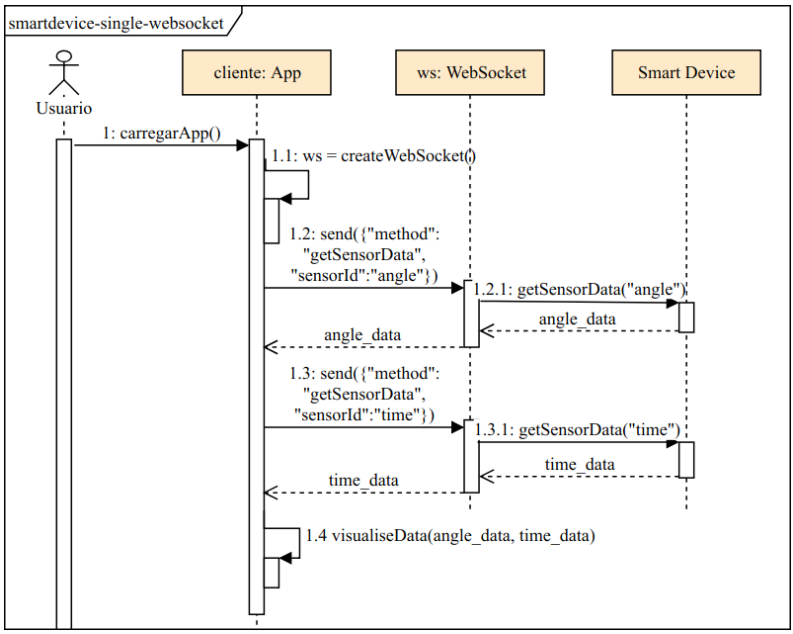
1.3 O Software cliente envia uma nova requisição em busca dos dados referente ao tempo de queda da esfera fornecido pelo sensor $time$;

1.3.1 O WebSocket transporta a mensagem ao *Smart Device*, que recebe, processa e responde a requisição enviando-a pelo WebSocket ao software

cliente;

1.4 O Software cliente processa os dados recebidos e apresenta ao usuário.

Figura 25 – Interação com uma conexão WebSocket.

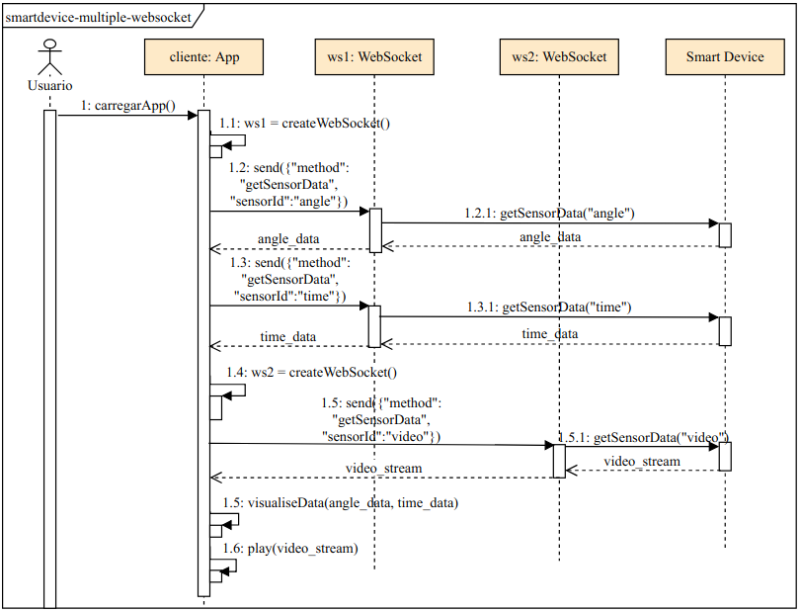


Fonte: Adaptado de Govaerts e Salzmann (2014).

Para alguns experimentos, a utilização de duas ou mais conexões pode facilitar a interação, por exemplo, utilizando uma conexão para o recebimento de um fluxo de vídeo, e outra para interação com os sensores e atuadores.

A figura 26 ilustra a utilização de duas conexões WS na interação com o *Smart Device*, a primeira instância do WebSocket (ws1) realiza a interação com os sensores, e a segunda instância (ws2) é utilizada para receber o fluxo de vídeo no software cliente.

Figura 26 – Interação com dois WebSockets.



Fonte: Adaptado de Govaerts e Salzmann (2014).

4.2 CASO DE USO COM CLIENTE WEB

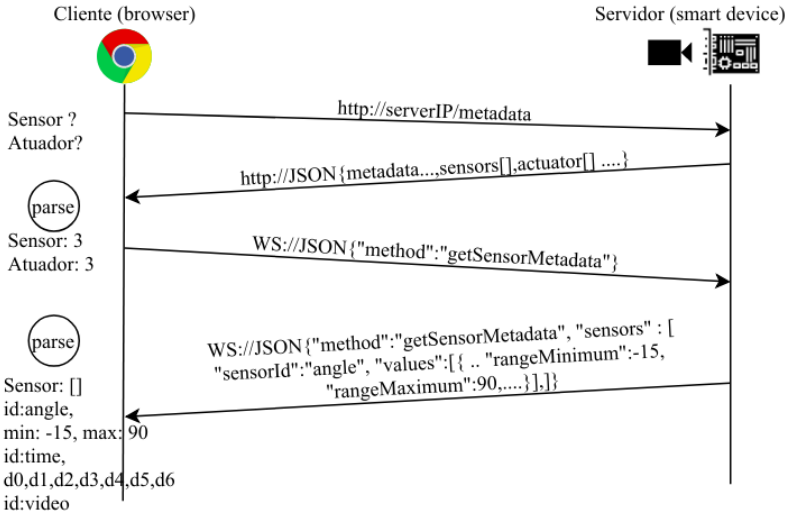
Esta seção ilustra detalhadamente como um cliente Web interage com o laboratório remoto baseado no modelo de dispositivos inteligentes.

O primeiro passo, consiste na consulta ao *Smart Device* sobre suas capacidades gerais por meio dos metadados, esta consulta pode ser realizada com uma requisição HTTP GET ao endereço do servidor, por exemplo: *"http://serverIP/metadata"* (figura 27).

O dispositivo inteligente envia a resposta em formato JSON contendo os metadados do experimento. O cliente Web realiza o *parse* e obtém as informações gerais.

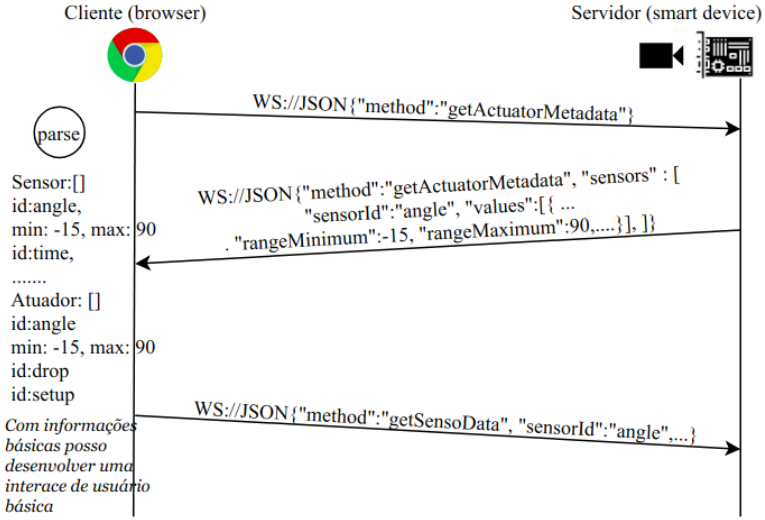
Na sequência o cliente consulta as informações sobre os sensores disponíveis. Esta requisição é realizada por meio de WebSockets, enviando um objeto JSON contendo {"method":"getSensorMetadata"} para o servidor, que responde com outro objeto JSON contendo um *array* com os sensores disponíveis.

Figura 27 – Cliente interagindo com o *Smart Device*.



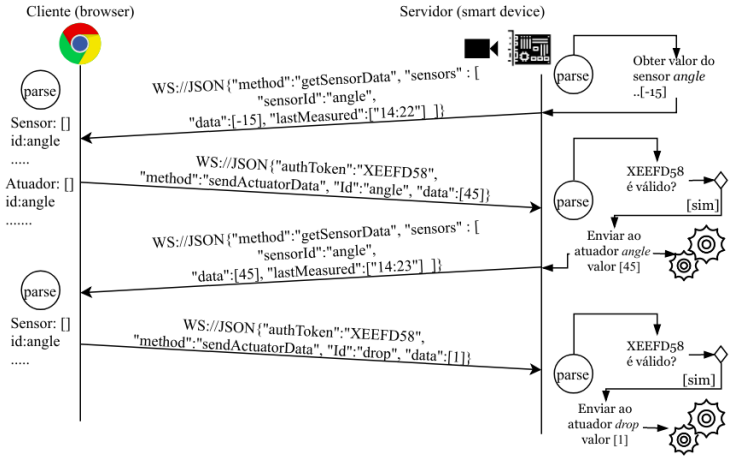
Fonte: Elaborado pelo autor.

Figura 28 – Cliente consultando atuadores do *SmartDevice*.



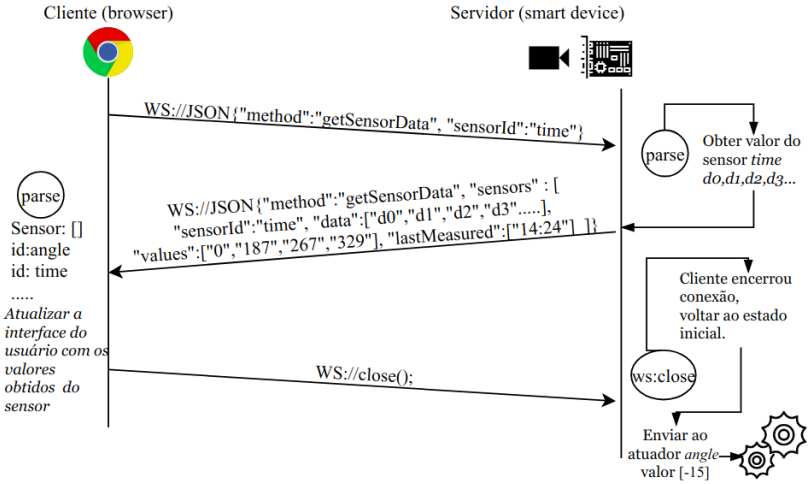
Fonte: Elaborado pelo autor.

Figura 29 – Cliente enviando valores para atuadores do *SmartDevice*.



Fonte: Elaborado pelo autor.

Figura 30 – Cliente enviando valores para atuadores do *SmartDevice*.



Fonte: Elaborado pelo autor.

O próximo passo consiste no cliente requisitar informações acerca dos atuadores (figura 28). O experimento responde com os dados, e a partir disso, o cliente possui informações para construção de uma interface gráfica básica

para o usuário.

Quando o cliente quer enviar um valor para um atuador (figura 29), deve realizar por meio de WebSocket, informando o novo valor em um objeto JSON como por exemplo {"authToken":"XEEFD58", "method": "sendActuatorData", "Id":"angle", "data":[45]}.

A requisição carrega um $token$ de autenticação, que será utilizado pelo *Smart Device* para garantir que a aplicação cliente possui autorização para acesso ao atuador (por exemplo, baseado em um serviço de agendamento). Se o $token$ for inválido, ou o valor enviado estiver fora do limite aceitável, não será aplicado ao atuador.

Ao encerrar o uso do experimento por parte do cliente, a conexão WebSocket será encerrada. Internamente o *Smart Device* irá para um estado conhecido (neste caso -15 graus) e aguardará por novos usuários.

Este capítulo descreve as ferramentas utilizadas no desenvolvimento e apresenta a utilização do protótipo servidor e cliente.

4.3 MATERIAIS UTILIZADOS

O desenvolvimento do protótipo foi dividido em duas etapas, a primeira, está relacionada ao desenvolvimento do servidor, que é responsável por fornecer as interfaces de comunicação no modelo de dispositivos inteligentes, os softwares e bibliotecas utilizadas estão descritos na tabela 3.

Tabela 3 – Softwares e bibliotecas utilizadas no servidor.

Software	**Funcionalidade**
Node.js	*framework* JavaScript responsável por interpretar os códigos no lado do servidor.
Express	biblioteca que fornece um mini servidor HTTP para o Node.js.
Cors	biblioteca que permite receber requisições de origem externa no servidor.
WebSocket	biblioteca que permite a criação de servidor e cliente WebSockets.

Fonte: Elaborado pelo autor.

A segunda etapa, refere-se aos clientes, em que cada software faz uso dos serviços providos pelo dispositivo inteligente e interação com o usuário.

Neste desenvolvimento, optou-se pela elaboração de softwares em mais de uma linguagem e plataforma computacional, desta forma, demonstrando que o modelo de dispositivos inteligentes pode ser adotado independente da linguagem adotada.

O ranking contendo as linguagens de programação mais populares em Junho de 2018, apresentado por O'Grady (2018) conta com dezenas de linguagens, no entanto, as cinco mais populares são: em primeiro lugar JavaS-

cript, em segundo a linguagem Java, em terceiro Python, quarto PHP e quinto lugar C#.

A linguagem JavaScript é popular no desenvolvimento de sistemas para Web, no ano de 2012, encontrava-se na segunda posição em popularidade (O'GRADY, 2018), e veio alternando a liderança em com o Java até 2015, onde tomou a ponta e continua como a mais popular até o momento.

O Java é popular por ser uma linguagem multiplataforma, o código é compilado para *bytecodes* que são interpretados por uma máquina virtual JVM (*Java Virtual Machine*), desta forma, o mesmo software pode ser executado em sistemas operacionais distintos.

Na terceira posição, a linguagem Python é utilizada em vários ambientes seja aplicações desktop ou Web, em quarto, a linguagem PHP é predominantemente utilizada em sistemas Web sendo utilizado no lado do servidor.

Na quinta posição, o C# é uma linguagem utilizada principalmente para desenvolvimento de aplicações para desktop Windows.

Tendo em vista estas linguagem mais populares, foram utilizadas as seguintes linguagens e bibliotecas listadas na tabela 4.

Tabela 4 – Linguagens e bibliotecas utilizadas nos softwares cliente.

Plataforma	**Linguagem e bibliotecas**
Web	HTML5, JavaScript e API WebSockets.
Desktop	C#, GTK2, biblioteca WebSocket-sharp e Newtonsoft.Json.
Desktop	Java, biblioteca Java-WebSocket e JSON.

Fonte: Elaborado pelo autor.

Para o desenvolvimento do software na linguagem C#, foi utilizado o ambiente integrado de desenvolvimento - IDE MonoDevelop, já na linguagem HTML e JavaScript o editor Sublime Text e na linguagem Java a IDE NetBeans.

5 TESTES DOS PROTÓTIPOS

Este capítulo apresenta os testes realizados com os protótipos, o primeiro teste foi realizado com o servidor, na sequência com os clientes Web, C# e Java.

5.1 SERVIDOR UTILIZANDO NODE.JS

O servidor baseado em dispositivos inteligentes deve fornecer interfaces bem definidas que permitam a comunicação entre ele e aplicações externas.

Os dados gerais devem ser especificados nos metadados do laboratório, o qual deve estar acessível pelo endereço IP do servidor (figura 31). Os dados completos estão disponíveis no apêndice A.

Figura 31 – Acesso aos metadados por meio do navegador Web.

Fonte: Elaborado pelo autor.

Ao instanciar o servidor WebSocket, ele aguarda por conexões, que quando ocorrem, disparam eventos relacionados.

A figura 32 apresenta um fragmento de código para explicar como o funciona a interação assíncrona. Quando o cliente envia um objeto JSON, os dados serializados são enviados em modo texto, o servidor recebe por meio da função $onmessage$ (linha 1), a partir disto, tenta realizar o $parse$ (linha 3) transformando-o num objeto, dessa forma pode ler os dados e com base nisto fornecer uma resposta ao cliente ou realizar uma ação no experimento,

Figura 32 – Fragmento de código do servidor.

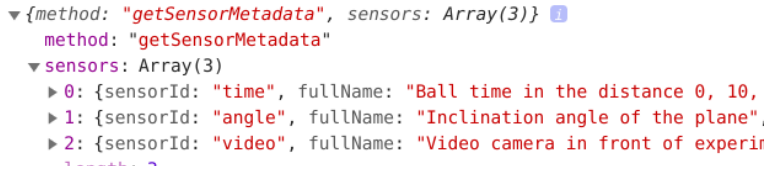
```
1  connection.on('message', function(message){
2    try {
3      var obj = JSON.parse(message.utf8Data);
4      switch(obj["method"]) { // start of switch method
5        case "getSensorMetadata":
6          .... // implementation of the methods / interfaces
7        }
8      }catch (e){
9        // handle parse errors
10     }
11 });
12 connection.on('close', function(e){
13   // handle close connection
14 });
15 connection.on('disconnect', function(e){
16   //handle disconnection
17 });
```

Fonte: Elaborado pelo autor.

Os metadados dos sensores e atuadores devem especificar todas as funcionalidades disponíveis tais como: tipo de dados, formatos, valores limites de aplicação aos atuadores entre outros.

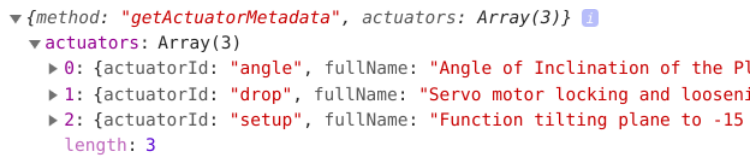
A figura 33, ilustra parcialmente a captura das informações recebidas pelo cliente obtidas como respostas das requisições $getSensorMetadata$ e a figura 34 $getActuatorMetadata$, o conteúdo completo das respostas são apresentadas no apêndice A.

Figura 33 – Inspeção no console do navegador Web $sensorMetadata$.



Fonte: Elaborado pelo autor.

Figura 34 – Inspeção no console do navegador Web $actuatorMetadata$.



Fonte: Elaborado pelo autor.

As trocas de mensagens das interfaces atuadores e sensores serão apresentadas detalhadamente na seção do cliente Web.

5.2 CLIENTE NA PLATAFORMA WEB

Uma das formas mais flexíveis de utilizar um software é por meio da plataforma Web, pois não é necessário a instalação de softwares adicionais no dispositivo do usuário (computador, notebook, *smartphone, tablet*), basta acessá-lo através do navegador Web.

Desta forma, o cliente Web torna-se a escolha mais adotada no desenvolvimento dos laboratórios remotos. A figura 35 ilustra o protótipo cliente sendo acessado por um *smartphone*, já a figura 36 ilustra a tela inicial acessada pelo navegador de um desktop.

Como pode-se observar na figura 36, no lado esquerdo, a área com imagem/vídeo é utilizada para recepção do fluxo de vídeo, para acompanhar os movimentos do experimento em tempo real. Do lado direito, estão os elementos para interação com o usuário.

O círculo cinza, em seu centro, a representação numérica indica o ângulo de inclinação da gangorra que o usuário deseja enviar ao experimento. Para enviar dados, utiliza-se os botões “Enviar”, “Setup” e “Soltar”. Abaixo, os campos de texto em branco, são utilizados para apresentar ao usuário o resultado do tempo de passagem da esfera em cada sensor do experimento.

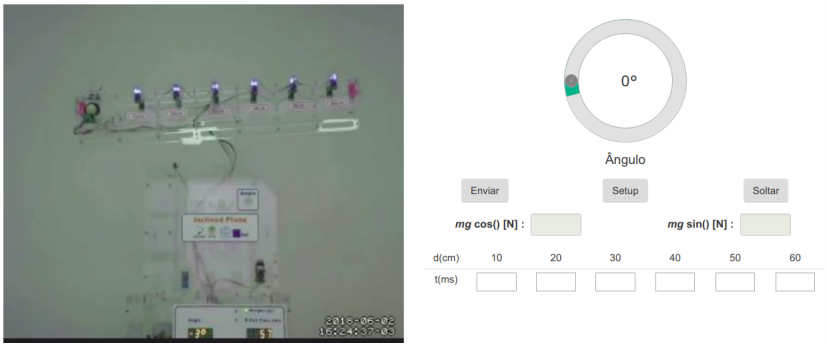
Para iniciar a utilização do experimento, o usuário deve clicar no botão “Setup” para que a gangorra seja posicionada à -15 graus, e prenda a esfera. Ao clicar, o software cliente gera um objeto JSON e o envia por meio de WebSockets ao servidor (figura 37).

Neste JSON, a linha 2 apresenta a chave $authToken$, que contém um cadeia de caracteres que é utilizada como uma chave de autenticação, que permite ao servidor verificar a autenticidade da requisição, na linha 3 campo $accessRole$ o tipo de acesso, que pode ser $controller$ ou $observer$, o valor $controller$ indica que ações de manipulação do experimento podem ser

Figura 35 – Tela inicial do protótipo Web acessado por celular.

Fonte: Elaborado pelo autor.

Figura 36 – Tela inicial do protótipo Web.

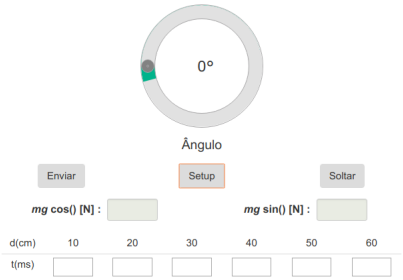


Fonte: Elaborado pelo autor.

executadas, já quando o valor é *observer*, as requisições apenas solicitam dados dos sensores/atuadores, também pode ser utilizada para solicitar a visualização do fluxo de vídeo do laboratório enquanto outro usuário utiliza o

Figura 37 – Objeto JSON para atuador $setup$ e resultado.

```
1  {
2  "authToken":"kGTKLsdpg6XkYxWiYFv5tIArsuUUOdJ2SaX4UVCQ",
3  "accessRole":"controller",
4  "method":"sendActuatorData",
5  "actuatorId":"setup",
6  "valueNames":["setup"],
7  "data":[1]
8  }
```



Fonte: Elaborado pelo autor.

experimento.

Na linha 4, a chave $method$, informa para qual interface e serviço a requisição é destinada, nesta ação o destino é o serviço $sendActuatorData$.

As linhas 5, 6 e 7 são relacionadas ao atuador no qual o valor é destinado, a chave $actuatorId$ indica ao servidor que o atuador alvo é o $setup$, a chave $valueNames$ é utilizada para identificar qual atuador específico referese, pois em alguns casos, um atuador pode ser complexo, isto é, composto por mais de um atuado, por isso, ele está entre [ ] (abertura e fechamento de colchetes) que indica um vetor de valores, assim pode conter apenas um valor ou uma coleção de valores separados por “,” (vírgula). A chave $data$ envia o valor “1”, que conforme documentação disponível por meio dos metadados, indica o valor para indicar a execução do atuador.

O resultado da ação do atuador é o posicionamento da gangorra na inclinação inicial.

Neste ponto, o usuário deve selecionar o ângulo de inclinação desejado para o experimento, selecionando o valor por meio do círculo que representa o ângulo, e clicar no botão “Enviar”.

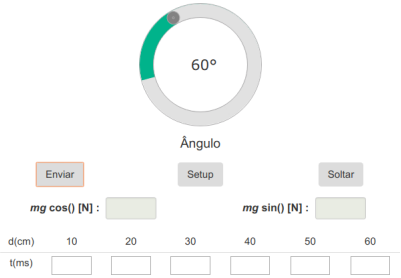
O fluxo de vídeo permite que o usuário acompanhe o movimento da

gangorra, o JSON que o software cliente envia para o servidor e a imagem do resultado da ação do atuador é ilustrado na figura 38.

A função das linhas 2 e 3 já foram explicadas na figura anterior, na linha 4, a chave $method$ também possui valor $sendActuatorData$, no entanto, a chave $actuatorId$ possui valor $Angle$, a chave $valueNames$, valor A, e a chave $data$, valor 60, que indica o ângulo que o plano inclinado deve posicionar a gangorra.

Figura 38 – Objeto JSON para o atuador $angle$ com valor 60 e resultado.

```
1 {
2   "authToken":"kGTKLsdpg6XkYxWiYFv5tIArsuUUOdJ2SaX4UVCQ",
3   "accessRole":"controller",
4   "method":"sendActuatorData",
5   "actuatorId":"Angle",
6   "valueNames":["A"],
7   "data":[60]
8 }
```



Fonte: Elaborado pelo autor.

Ao concluir a movimentação da gangorra para o grau selecionado, o usuário então, deve clicar no botão “Soltar”, o experimento abre a trava que prende a esfera, que desce pelo plano. O JSON enviado ao servidor é ilustrado na figura 39.

As chaves $actuatorId$ e $valueNames$ nas linhas 5 e 6 possuem valor $drop$, e a chave $data$ na linha 7, valor 1, utilizado para realizar a abertura da trava.

No momento que a esfera desce, os sensores infravermelho registram o tempo de passagem em cada um dos pontos: 10cm, 20cm, 30cm, 40cm, 50cm e 60cm.

Após alguns instantes o *Smart Device* envia os dados do sensor $time$ para o software cliente por meio de um objeto JSON, este, por sua vez, realiza

Figura 39 – Objeto JSON para atuador $drop$.

```
{
  "authToken":"kGTKLsdpg6XkYxWiYFv5tIArsuUUOdJ2SaX4UVCQ",
  "method":"sendActuatorData",
  "accessRole":"controller",
  "actuatorId":"drop",
  "valueNames":["drop"],
  "data":[1]
}
```

Fonte: Elaborado pelo autor.

o *parse* da mensagem e exibe os valores na interface do usuário.

O objeto JSON enviado pelo servidor com os dados do sensor $time$, o plano inclinado e a interface exibindo os dados ao usuário é exibido na figura 40.

Dentro deste objeto JSON, a chave $responseData$ é um novo objeto composto por 3 *arrays*, sendo eles $valueNames$, $data$ e $lastMeasured$, pode-se representá-los em forma de tabela para facilitar a visualização de como os dados estão dispostos (tabela 5), deste modo, a primeira coluna representa o nome do *array* e as demais representam os índices (posição dos elementos) e seus valores.

Tabela 5 – Tabulação do objeto $responseData$.

chaves	**índices**						
	0	1	2	3	4	5	6
valueNames	d0	d1	d2	d3	d4	d5	d6
data	0	153	221	275	319	358	394
lastMeasured[1]	16:10	16:10	16:10	16:10	16:10	16:10	16:10

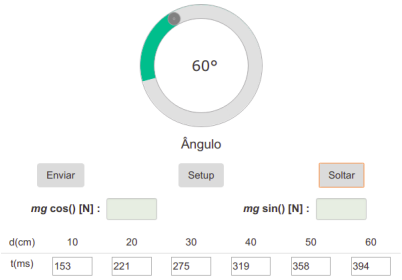
Fonte: Elaborado pelo autor.

A linhas contêm seus respectivos valores nos índices, assim, a leitura dos resultados no índice 0, é composto por $valueNames$ = $d0$, $data$ = 0 e $lastMeasured$ = 16:10, o índice 1 é composto por $valueNames$ = $d1$, $data$ = 153 e $lastMeasured$ = 16:10, e assim os resultados são formados sucessivamente, de modo que o software cliente possa apresentar os resultados ao usuário.

A partir disso, a gangorra volta para posição inicial e o usuário pode selecionar um novo valor para inclinação do plano e continuar realizando experimentos, ou encerrar a conexão.

Figura 40 – Objeto JSON do sensor $time$ e resultado.

```
{
  "method":"getSensorData",
  "lastMeasured":"2018-07-01T16:10:59.288Z",
  "accessRole":"controller",
  "responseData":{
    "valueNames":["d0","d1","d2","d3","d4","d5","d6"],
    "data":[0,153,221,275,319,358,394],
    "lastMeasured":[
     "2018-07-01T16:10:59.288Z","2018-07-01T16:10:59.288Z",
     "2018-07-01T16:10:59.288Z","2018-07-01T16:10:59.288Z",
     "2018-07-01T16:10:59.288Z","2018-07-01T16:10:59.288Z",
     "2018-07-01T16:10:59.288Z"
    ]
  }
}
```



Fonte: Elaborado pelo autor.

5.3 CLIENTE NA LINGUAGEM C#

O C# é uma linguagem orientada a objeto e fortemente tipada[2], sua sintaxe simplifica muitas complexidades do C++ e fornece recursos poderosos que não existem no Java.

O processo de compilação é simples comparado ao C e C++ e mais flexível do que em Java. Programas em C# são executados no .NET Framework, um componente integral do Microsoft Windows.

O Mono é uma plataforma de software projetada para permitir que os desenvolvedores criem rápido e facilmente aplicativos C# para área de trabalho no Linux, Windows e macOS (*cross plataform*) como parte do .NET

[2]Todas as variáveis devem ter seu tipo definido (int, float e varchar por exemplo) para serem utilizadas.

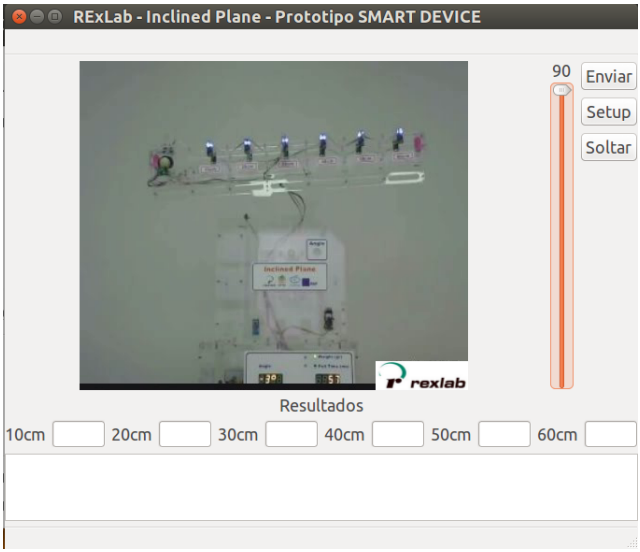
Foundation.

Já o Gtk#, é um kit de ferramentas para desenvolvimento de interface gráfica do usuário para Mono e .Net. Este projeto vincula o Gtk + *toolkit* e as bibliotecas do Gnome[3], permitindo o desenvolvimento de aplicações Gnome totalmente nativas usando os *frameworks* de desenvolvimento Mono e .Net.

O visual da janela do software cliente em C# e Gtk (figura 41) é semelhante ao cliente Web, no entanto, o fluxo de vídeo do experimento ocupa uma parte central da janela.

Do lado direito, estão posicionados a barra de escala para selecionar o ângulo desejado e os botões “Enviar”, “Setup” e “Soltar”. As caixas de texto abaixo da palavra “Resultados” são utilizadas para apresentar o tempo de passagem da esfera nos respectivos sensores, e por fim, a caixa de texto na parte inferior da janela, serve como uma espécie de visualização de registros de logs, onde é possível verificar o conteúdo dos objetos JSON enviados e recebidos entre software cliente e o dispositivo inteligente.

Figura 41 – Tela inicial do protótipo em C#.



Fonte: Elaborado pelo autor.

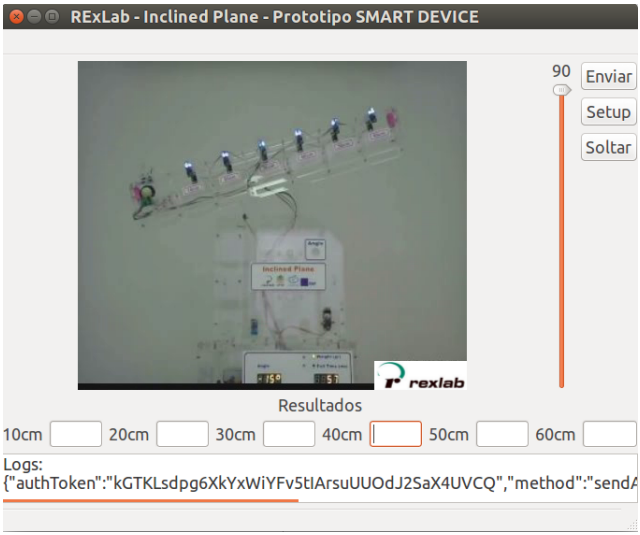
A utilização do experimento e sua interação com o servidor é apresentada por meio das imagens seguintes. O primeiro procedimento é clicar no botão “Setup”, para posicionar o experimento na posição inicial e prender a

[3]Gerenciador de janelas compatível com a maioria das distribuições GNU/Linux.

esfera, como ilustrado na figura 42.

Afim de validar o experimento, optou-se por selecionar a inclinação de 49 graus, que possui o resultado ilustrado na figura 43.

Figura 42 – Resultado do atuador *setup*.



Fonte: Elaborado pelo autor.

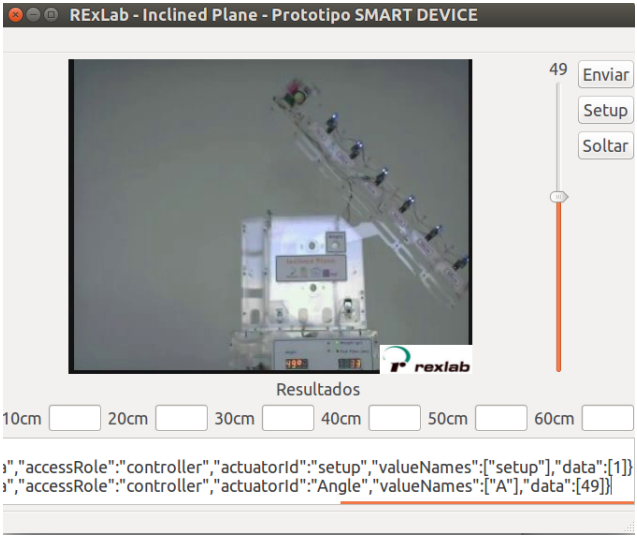
Ao clicar no botão “Soltar” a esfera desce pela gangorra e os sensores registram o tempo, em seguida os resultados são exibidos para o usuário (figura 44).

A caixa de texto que exibe os objetos JSON, possui finalidade de apresentar a troca de mensagens e observação de possíveis erros, na versão deste software cliente para uso de estudantes, esta parte do programa poderia ser omitida, assim como melhorias na parte visual da janela também podem ser implementadas.

5.4 CLIENTE NA LINGUAGEM JAVA

Dentre as principais características da linguagem de programação Java estão a portabilidade e a orientação a objetos. Sobre portabilidade, nas palavras de Furgeri (2012, p. 19) “uma mesma aplicação pode ser executada em diferentes tipos de plataforma sem a necessidade de adaptação de código”, sendo assim um programador não se preocupa se sua aplicação irá ou não

Figura 43 – Plano inclinado na posição 49 graus.



Fonte: Elaborado pelo autor.

executar em determinado sistema operacional.

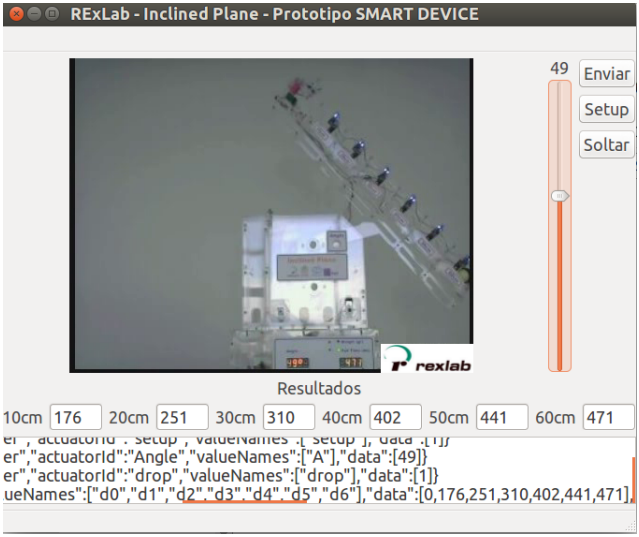
Esta característica do Java, pode torná-lo interessante para o desenvolvimento de clientes para laboratórios remotos em alguns casos, continuando com a validação deste software, o visual da janela (figura 45) também é semelhante ao cliente Web e quase idêntico ao desenvolvido em C#, em que o fluxo de vídeo do experimento ocupa a maior parte da janela.

Do lado direito,estão posicionados a barra de escala para selecionar o ângulo desejado e os botões “Enviar”, “Setup” e “Soltar”. As caixas de texto abaixo da palavra “Resultados” são utilizadas para apresentar os tempos de passagem da esfera nos respectivos sensores, e por fim, a caixa de texto na parte inferior da janela, serve como uma espécie de visualização de registros de logs, onde é possível verificar o conteúdo dos objetos JSON enviados e recebidos entre software cliente e o dispositivo inteligente.

Apresenta-se a utilização do experimento e sua interação com o servidor por meio das imagens seguintes. O primeiro procedimento é clicar no botão “Setup”, para posicionar o experimento na posição inicial e prender a esfera, como ilustrado na figura 46

Neste caso, optou-se por selecionar a inclinação de 90 graus, que fará com que a esfera realize uma queda livre, seu resultado ilustrado na figura 47. Ao clicar no botão “Soltar” a esfera desce pela gangorra e os sensores regis-

Figura 44 – Plano inclinado após descida da esfera e resultados exibidos.



Fonte: Elaborado pelo autor.

tram o tempo, em seguida os resultados são exibidos para o usuário (figura 48).

Salienta-se que a caixa de texto que exibe os objetos JSON, possui finalidade de demonstração da troca de mensagens e observação de possíveis erros, esta parte do programa poderia ser omitida no uso em ambiente de produção[4], assim como melhorias no visual podem ser implementadas.

No processo de desenvolvimento, nota-se que no ambiente Web, as ferramentas necessárias para comunicação e manipulação dos dados (API WebSockets e suporte da linguagem para realizar o $parse$ para manipulação do JSON) já são integrada aos navegadores, facilitando o desenvolvimento.

Já no desenvolvimento para desktop, as dificuldades são maiores, necessitando de bibliotecas externas para comunicação por meio de WebSockets e para realização do tratamento das informações contidas nos objetos JSON.

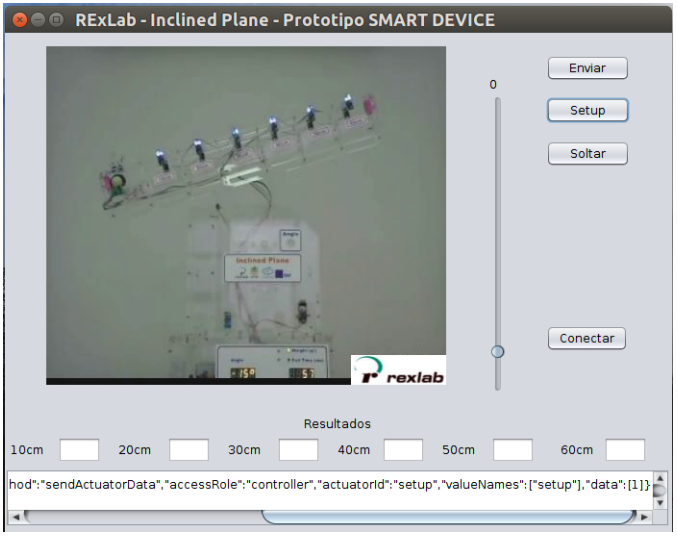
A utilização do da biblioteca Socket.IO no servidor, pode comprometer a compatibilidade dos experimentos com outras plataformas além da Web, porém neste trabalho, foi apresentado a implementação da biblioteca Websocket, apresentando que a viabilidade do padrão em termos de compatibilidade com as demais plataformas.

[4]Quando o software já passou por testes e está disponível para uso regular.

Figura 45 – Tela inicial do protótipo em Java.

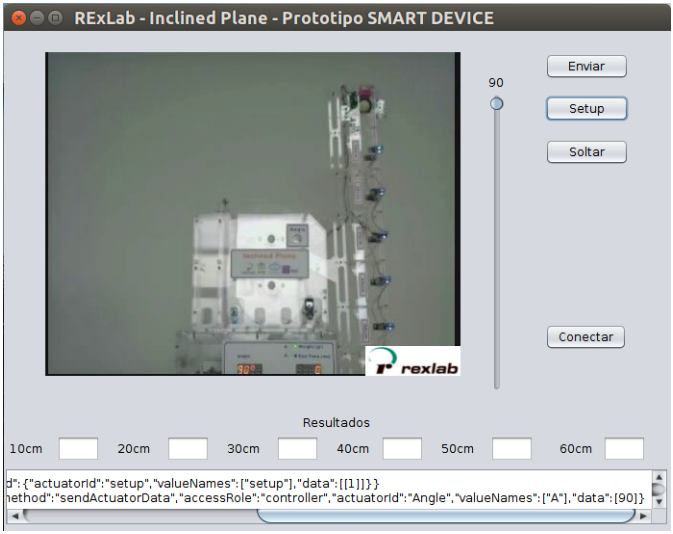
Fonte: Elaborado pelo autor.

Figura 46 – Resultado do atuador *setup*.



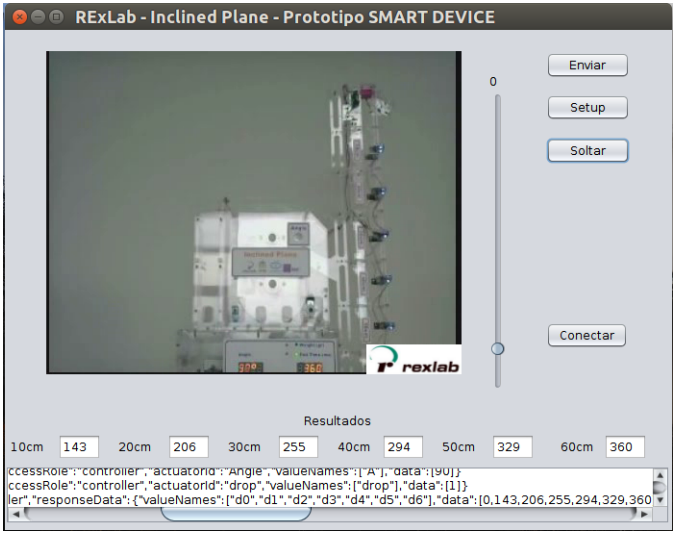
Fonte: Elaborado pelo autor.

Figura 47 – Plano inclinado na posição 90 graus.



Fonte: Elaborado pelo autor.

Figura 48 – Plano inclinado após descida da esfera e resultados exibidos.



Fonte: Elaborado pelo autor.

6 CONSIDERAÇÕES FINAIS

Os laboratórios virtuais e remotos tornaram-se alternativas para que, na falta de laboratórios tradicionais, estudantes possam realizar atividades experimentais, tanto para comprovação de conhecimentos teóricos como também para que tenham um papel de protagonista em seu processo de aprendizagem.

A utilização dos laboratórios de experimentação remota, pode proporcionar a redução de custos e compartilhamento com outras instituições, no entanto, existem desafios relacionados ao seu compartilhamento, haja visto que não existe padronização na forma que são desenvolvidos.

A revisão exploratória da literatura apresentou que, ao passar dos anos, houve uma preocupação crescente com a padronização da comunicação dos laboratórios remotos, no qual a arquitetura dos dispositivos inteligentes é o padrão defendido.

Destaca-se que nos trabalhos já publicados, há uma impressão da obrigatoriedade em utilizar a plataforma Web nesta padronização, no entanto, é possível utilizar linguagens de programação e plataformas computacionais diversas, como as três apresentadas neste trabalho, porém, não restringindo-se somente a estas.

Os resultados deste trabalho respondem a pergunta de pesquisa, por meio da aplicação do modelo de dispositivos inteligentes, em que o cliente e o servidor do laboratório estão totalmente desvinculados.

Esta abordagem pode proporcionar um compartilhamento de experimentos de forma mais fácil, e a elaboração de interfaces de softwares clientes que atenda requisitos específicos de cada aplicação, pois as interfaces de comunicação são bem definidas.

Este estudo apresenta a implementação do modelo de comunicação baseado nos dispositivos inteligentes de forma ampla e detalhada, suprindo lacunas a cerca da implementação presente em trabalhos já publicados.

Conclui-se que este modelo de comunicação desacoplando as funcionalidades servidor sendo acessadas por qualquer cliente utilizando as interfaces especificadas, pode ser implantado independente da plataforma de hardware ou da linguagem de programação adotada, neste trabalho, o servidor foi implementado com o Node.js e WebSocket, como clientes, foram desenvolvidos três softwares em linguagens distintas.

Foi apresentado detalhadamente a implementação do servidor e dos clientes Web, C# e Java, dessa forma, podem ser utilizados como modelo para outros desenvolvedores.

Na plataforma Web utilizando HTML e JavaScript, apresenta-se como

implementação mais flexível, pois proporciona fácil acesso pela Internet sem a necessidade de instalação de softwares adicionais. Em C# por ser uma linguagem muito utilizada no desenvolvimento de softwares para desktop, principalmente no sistema operacional Windows. E para finalizar, na linguagem Java, que é multiplataforma e uma das mais utilizadas na atualidade.

Na realização deste trabalho, não foram encontrados elementos que comprometessem as funcionalidades deste experimento remoto com a implementação do modelo de comunicação utilizando a arquitetura de dispositivos inteligentes.

Outras linguagens de programação como PHP, Python entre outras, também possuem suporte para WebSockets e JSON no lado cliente. Já no lado do servidor, o WebSocket pode ser implementado em outros softwares como Jetty em Java, EventMachine em Ruby, C++ por meio da libwebsocket como também os populares servidores Web Apache e IIS.

Outro benefício deste modelo, está na autonomia que proporciona ao desenvolvedor para integrar outras funcionalidades no software cliente, seja para ampliar a interação do usuário com o experimento, ou como forma de analisar o comportamento durante seu uso.

Trabalhos relacionados, propõem o registro de dados referente à experiencia de usuários com os laboratórios por meio da xAPI [1] (*Experience* API), que pode ser facilmente implantada em experimentos que utilizam o modelo de dispositivos inteligentes.

Para trabalhos futuros, algumas investigações são propostas, uma delas é a adoção de ferramenta autenticação e agendamento padronizadas, isto tende a tornar mais fácil a integração de laboratórios que necessitam de autenticação/autorização para acesso. Outra diz respeito a inclusão de novos metadados referente aos experimentos, para que por meio deles, seja possível a criação de interfaces gráficas de usuário por meio de ferramentas automáticas, como abordado no trabalho "Enabling the Automatic Generation of User Interfaces for Remote Laboratorie" de (HALIMI et al., 2017).

Para fortalecer o incentivo de adoção da especificação da arquitetura de dispositivos inteligentes, vários parceiros do projeto Go-Lab estão envolvidos no Grupo de Trabalho IEEE P1876, que trabalha na padronização deste modelo de comunicação como norma do IEEE. Padrões estabelecem especificações e procedimentos, por meio deles que os requisitos de interconectividade e interoperabilidade podem ser assegurados.

[1] A Experience API (ou xAPI) é uma nova especificação para tecnologia de aprendizado que possibilita a coleta de dados sobre a ampla variedade de experiências que uma pessoa tem (online e off-line)diponível em:<https://xapi.com/overview>.

REFERÊNCIAS

ALMEIDA, M. E. B. de. Tecnologia na educação: dos caminhos trilhados aos atuais desafios. *Bolema*, n. 29, p. 99–129, 2008.

BARRETO, R. G. Tecnologia e educação: trabalho e formação docente. *Educação & Sociedade*, scielo, v. 25, p. 1181 – 1201, 12 2004. ISSN 0101-7330. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0101-73302004000400006&nrm=iso>.

BASSETT, L. Book. *Introduction to JavaScript Object Notation*. 1. ed. Sebastpol CA: O'Reilly Media Inc., 2015. ISBN 978-1-491-92948-3.

BECKER, S. A. et al. *NMC/CoSN Horizon Report: 2016 K–12 Edition*. Austin, Texas, 2016.

BRAY, T. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC Editor, mar. 2014. RFC 7159. (Request for Comments, 7159). Disponível em: <https://rfc-editor.org/rfc/rfc7159.txt>.

BRAY, T. *The JavaScript Object Notation (JSON) Data Interchange Format*. [S.l.], December 2017.

CAMPOS, J. B. da C.; CRUZ, G. B. da. *Laboratórios*. 4. ed. Cuiabá: Universidade Federal de Mato Grosso, 2013. 107 p. ISBN 978-85-230-0977-9.

CHANIOTIS, I. K.; KYRIAKOU, K.-I. D.; TSELIKAS, N. D. Is node.js a viable option for building modern web applications? a performance evaluation study. *Computing*, v. 97, n. 10, p. 1023–1044, Oct 2015. ISSN 1436-5057. Disponível em: <https://doi.org/10.1007/s00607-014-0394-9>.

CHITUNGO, H. H. C. *O uso de laboratórios remotos no ensino de física na educação básica: Estudo de caso em escola da rede pública*. 110 p. Dissertação (Mestrado em Tecnologias da Informação e Comunicação) — Universidade Federal de Santa Catarina, Araranguá, 02 2018.

CUPANI, A. La peculiaridad del conocimiento tecnológico. *ScientiaeStudia*, v. 4, n. 3, p. 353–371, 2006.

DEAKY, B. A. Contribution to online laboratory implementation and standardization. In: *2013 IEEE Global Engineering Education Conference (EDUCON)*. [S.l.: s.n.], 2013. p. 1342–1346. ISSN 2165-9559.

DORIGONI, G. M. L.; SILVA, J. C. da. *Mídia e educação: o uso das novas tecnologias no espaço escolar.* 2008. Disponível em: <http://www.diaadiaeducacao.pr.gov.br/portals/pde/arquivos/1170-2.pdf>. Acesso em: 09 mai. 2018.

FETTE, I.; MELNIKOV, A. *The WebSocket Protocol*. [S.l.], December 2011. (Request for Comments, 6455). Disponível em: <https://rfc-editor.org/rfc/rfc6455.txt>.

FILHO, M. C. F.; FILHO, E. J. M. A. *Planejamento da pesquisa científica.* 1. ed. São Paulo: Atlas, 2013.

FOUNDATION, W. W. W. *History of the Web*. 2018. Disponível em: <https://webfoundation.org/about/vision/history-of-the-web/>. Acesso em: 8 abr. 2018.

FREEMAN, S. et al. Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences*, National Academy of Sciences, v. 111, n. 23, p. 8410–8415, 2014. ISSN 0027-8424. Disponível em: <http://www.pnas.org/content/111/23/8410>.

FREIRE, P. de S. *Aumente a Qualidade e Quantidade de Suas Publicações Científicas: Manual para elaboração de projetos e artigos científicos*. Curitiba: Crv, 2013. 90 p.

FURGERI, S. *Java 7: ensino didático*. 2. ed. São Paulo: Érica, 2012.

GARCíA, M. L. et al. Rethinking remote laboratories: Widgets and smart devices. In: *2013 IEEE Frontiers in Education Conference (FIE)*. [S.l.: s.n.], 2013. p. 782–788. ISSN 0190-5848.

GERHARDT, T. E. e. a. *Métodos de pesquisa*. Editora da ufrgs. Porto Alegre: Universidade Federal do Rio Grande do Sul, 2009.

GIL, A. C. *Como elaborar projetos de pesquisa.* 4. ed. São Paulo: Atlas, 2008.

GOMES, L.; BOGOSYAN, S. Current trends in remote laboratories. *IEEE Transactions on Industrial Electronics*, v. 56, n. 12, p. 4744–4756, Dec 2009. ISSN 0278-0046.

GOVAERTS, S.; SALZMANN, C. *Specifications of the Smart Device for Remote labs*. [S.l.], 2014. 99 p.

HALIMI, W.; SALZMANN, C.; GILLET, D. The smart wind turbine lab. In: *2015 3rd Experiment International Conference (exp.at'15)*. [S.l.: s.n.], 2015. p. 118–119.

HALIMI, W.; SALZMANN, C.; GILLET, D. The mach-zehnder interferometer - a smart remote experiment based on a software template. In: *2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV)*. [S.l.: s.n.], 2016. p. 287–292.

HALIMI, W. et al. Enabling the automatic generation of user interfaces for remote laboratories. *Proceedings of the 14th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, Springer, 2017.

HECK, C. *Integração de tecnologia no ensino de física na Educação Básica: um estudo de caso utilizando a experimentação remota móvel*. 133 p. Dissertação (Mestrado em Tecnologias da Informação e Comunicação) — Universidade Federal de Santa Catarina, Araranguá, 02 2017.

JUNIOR, V. F. et al. Design science research methodology enquanto estratégia metodológica para a pesquisa tecnológica. *Revista Espacios*, v. 38, n. 6, p. 25–35, 2017. Disponível em: <http://www.revistaespacios.com/a17v38n06/17380625.html>. Acesso em: 20 ago. 2018.

LOWE, D. et al. Evolving remote laboratory architectures to leverage emerging internet technologies. *IEEE Transactions on Learning Technologies*, v. 2, n. 4, p. 289–294, 2009. ISSN 1939-1382.

MULDER, P.; BRESEMAN, K. *Node.js for Embedded Systems: Using Web Technologies to Build Connected Devices*. 1. ed. O'Reilly Media, 2016. ISBN 1491928999,9781491928998. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=27c019d38d2c69eb95467d15f831afb4>.

NASCIMENTO, J. K. F. do. *Informática aplicada à educação.* [S.l.]: Universidade de Brasília, 2009.

NEDIC, Z.; MACHOTKA, J.; NAFALSKI, A. Remote laboratories versus virtual and real laboratories. In: *Frontiers in Education, FIE 2003 33rd Annual*. [S.l.: s.n.], 2003. p. T3E–1 – T3E–6.

NETCRAFT. *April 2018 web server survey*. 2018. Disponível em: <https://news.netcraft.com/archives/2018/04/26/april-2018-web-server-survey.html>.

NICOLETE, P. C. *Integração de tecnologia na Educação: Grupo de Trabalho em Experimentação Remota Móvel (GT-MRE) um estudo de caso*. 219 p. Dissertação (Mestrado em Tecnologias da Informação e Comunicação) — Universidade Federal de Santa Catarina, Araranguá, 2016.

O'GRADY, S. *The RedMonk Programming Language Rankings: June 2018*. 2018. Disponível em: <https://redmonk.com/sogrady/2018/08/10/language-rankings-6-18/>. Acesso em: 21 ago. 2018.

ORDUñA, P. et al. An extensible architecture for the integration of remote and virtual laboratories in public learning tools. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, v. 10, n. 4, p. 223–233, Nov 2015. ISSN 1932-8540.

PHET. *PhET Interactive Simulations*. 2018. Disponível em: <https://phet.colorado.edu>. Acesso em: 24 mai. 2018.

POSLAD, S. Book. *Ubiquitous Computing: Smart Devices, Environments and Interactions*. 1. ed. Queen Mary, University of London, UK: John Wiley e Sons, Ltd, 2009. ISBN 978-0-470-03560-3.

REKIK, W.; MHIRI, M.; KHEMAKHEM, M. A smart cloud repository for online instrument. In: *International Conference on Education and e-Learning Innovations*. [S.l.: s.n.], 2012. p. 1–4.

ROCHA, C. A. *Mediações tecnológicas na educação superior*. [S.l.]: Ibpex, 2009.

RODRIGUEZ-GIL, L. et al. Towards new multiplatform hybrid online laboratory models. *IEEE Transactions on Learning Technologies*, v. 10, n. 3, p. 318–330, July 2017. ISSN 1939-1382.

SALZMANN, C. et al. The smart device specification for remote labs. In: *Proceedings of 2015 12th International Conference on Remote Engineering and Virtual Instrumentation (REV)*. [S.l.: s.n.], 2015. p. 199–208.

SEVERANCE, C. Discovering javascript object notation. *Computer*, v. 45, n. 4, p. 6–8, April 2012. ISSN 0018-9162.

SILVA, J. B. da. *A utilização da experimentação remota como suporte para ambientes colaborativos de aprendizagem*. Tese (Doutorado em Engenharia e Gestão do Conhecimento) — Universidade Federal de Santa Catarina, 02 2007.

SILVA, J. B. da et al. *Manual Técnico do Experimento Remoto Plano Inclinado*: Experimentação remota móvel para a educação básica e superior. [S.l.], 2016. Disponível em: <http://relle.ufsc.br/docs/58012229dd76d.pdf>. Acesso em: 25 abr. 2018.

SIMãO, J. P. S. et al. Utilização de experimentação remota móvel no ensino médio. *RENOTE - Revista Novas Tecnologias na Educação*, v. 11, n. 1, p. 1–11, 2013. ISSN 1679-1916. Disponível em: <http://seer.ufrgs.br/index.php/renote/article/view/41701>.

TEIXEIRA, P. *Professional Node.js: Building Javascript Based Scalable Software*. 1. ed. Wrox, 2012. ISBN 1118185463,9781118185469. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=0444caaf277112fc579607d0d036adbd>.

THOMPSON, C. W. Smart devices and soft controllers. *IEEE Internet Computing*, v. 9, n. 1, p. 82–85, Jan 2005. ISSN 1089-7801.

TSOURLIDAKI, E.; ZERVAS, P. *The Go-Lab Inventory and Integration of Online Labs – Labs Offered by Universities*. [S.l.], 2014. 247 p.

W3SCHOOLS. *W3Schools Online Web Tutorials*. 2018. Disponível em: <https://www.w3schools.com>. Acesso em: 8 abr. 2018.

WANG, V.; SALIM, F.; MOSKOVITS, P. *The Definitive Guide to HTML5 WebSocket*. [S.l.]: Apress, 2013. ISBN 978-1-4302-4741-8.

WEBSOCKET.ORG. *About HTML5 WebSocket*. 2018. Disponível em: <http://websocket.org/aboutwebsocket.html>. Acesso em: 8 abr. 2018.

ZUTIN, D. G. et al. Lab2go - a repository to locate educational online laboratories. In: *IEEE EDUCON 2010 Conference*. [S.l.: s.n.], 2010. p. 1741–1746. ISSN 2165-9559.

APÊNDICE A – Metadados JSON

A.1 METADADOS DO LABORATÓRIO

```
{
  "apiVersion":"1.0.0",
  "basePath": "http://relle.rexlab.ufsc.br/inclinedplane/",
  "info": {
    "title":"Inclined Plane",
    "description":"Study of Newton's second law of motion
        and decomposition of forces in vectors",
    "termsOfServiceUrl":"",
    "contact":"henriquegolinelli@hotmail.com",
    "license":"Apache 2.0",
    "licenseUrl":"http://www.apache.org/licenses/LICENSE
        -2.0.html"
  },
  "authorizations":{},
  "concurrency": {
    "interactionMode": "synchronous",
    "concurrencyScheme": "roles",
    "roleSelectionMechanism": ["race"],
    "roles": [
    {
      "role": "controller",
      "selectionMechanism": ["race"]
    }
    ]
  },
  "apis": [
  {
    "path": "/client",
    "protocol": "WebSocket",
    "produces": [
    "application/json"
    ],
    "operations": [
    {
      "method": "Send",
      "nickname": "getClients",
      "summary": "Return a list of all available clients",
      "notes": "Returns a JSON array with all the available
          clients",
      "type": "ClientResponse",
      "parameters": [
      {
        "name": "message",
        "description": "the payload for the getClients
            service.",
        "required": true,
        "paramType": "message",
        "type": "SimpleRequest",
```

```
45          "allowMultiple": false
46        }
47        ],
48        "authorizations": {},
49        "responseMessages": [
50        {
51          "code": 402,
52          "message": "Too many users"
53        },
54        {
55          "code": 404,
56          "message": "Clients not found"
57        },
58        {
59          "code": 405,
60          "message": "Method not implemented. The requested
                method is not allowed by this server."
61        },
62        {
63          "code": 412,
64          "message": "JSON string is invalid."
65        },
66        {
67          "code": 413,
68          "message": "method key missing."
69        },
70        {
71          "code": 422,
72          "message": "The request body is unprocessable"
73        }
74        ]
75      }
76      ]
77    },
78    {
79      "path": "/sensor/",
80      "protocol": "WebSocket",
81      "produces": [
82      "application/json"
83      ],
84      "operations": [
85      {
86        "method": "Send",
87        "nickname": "getSensorMetadata",
88        "summary": "List all sensors and their metadata",
89        "type": "SensorMetadataResponse",
90        "parameters": [
91        {
92          "name": "message",
93          "description": "the payload for the
                getSensorMetadata service",
94          "required": true,
```

```
95          "paramType": "message",
96          "type": "SimpleRequest",
97          "allowMultiple": false
98        }
99        ],
100       "responseMessages": [
101       {
102         "code": 401,
103         "message": "Unauthorised access. The authentication
                token is missing"
104       },
105       {
106         "code": 402,
107         "message": "Too many users"
108       },
109       {
110         "code": 403,
111         "message": "Unauthorised access. The access role is
                not valid"
112       },
113       {
114         "code": 405,
115         "message": "Method not allowed. The requested
                method is not allowed by this server."
116       },
117       {
118         "code": 412,
119         "message": "JSON string is invalid."
120       },
121       {
122         "code": 413,
123         "message": "method key missing."
124       },
125       {
126         "code": 414,
127         "message": "accesRole key missing."
128       },
129       {
130         "code": 422,
131         "message": "The request body is unprocessable"
132       }
133       ],
134       "authorisations":{}
135     },
136     {
137       "method": "Send",
138       "nickname": "getSensorData",
139       "summary": "Get data from the sensor with the given
            sensor identifier",
140       "type": "SensorDataResponse",
141       "parameters": [
142       {
```

```
143           "name": "message",
144           "description": "The payload for the getSensorData
                  service",
145           "required": true,
146           "type": "SensorDataRequest",
147           "paramType": "message",
148           "allowMultiple": false
149         }
150         ],
151         "responseMessages": [
152         {
153           "code": 401,
154           "message": "Unauthorised access. The authentication
                  token is missing"
155         },
156         {
157           "code": 402,
158           "message": "Too many users"
159         },
160         {
161           "code": 403,
162           "message": "Unauthorised access. The access role is
                  not valid"
163         },
164         {
165           "code": 404,
166           "message": "No sensor found"
167         },
168         {
169           "code": 405,
170           "message": "Method not allowed. The requested
                  method is not allowed by this server."
171         },
172         {
173           "code": 406,
174           "message": "Value coerced for updatefrequency."
175         },
176         {
177           "code": 408,
178           "message": "Value invalid for this sensor"
179         },
180         {
181           "code": 412,
182           "message": "JSON string is invalid."
183         },
184         {
185           "code": 413,
186           "message": "method key missing."
187         },
188         {
189           "code": 414,
190           "message": "accesRole key missing."
```

```
191         },
192         {
193           "code": 422,
194           "message": "The request body is unprocessable"
195         }
196         ]
197       }
198       ]
199     },
200
201
202     {
203       "path": "/actuator/",
204       "protocol": "WebSocket",
205       "produces": [
206       "application/json"
207       ],
208       "operations": [
209       {
210         "method": "Send",
211         "nickname": "getActuatorMetadata",
212         "summary": "List all actuators and their metadata",
213         "type": "ActuatorMetadataResponse",
214         "parameters": [
215         {
216           "name": "message",
217           "description": "the payload for the
                getActuatorMetadata service",
218           "required": true,
219           "paramType": "message",
220           "type": "SimpleRequest",
221           "allowMultiple": false
222         }
223         ],
224         "responseMessages": [
225         {
226           "code": 401,
227           "message": "Unauthorised access. The authentication
                token is missing"
228         },
229         {
230           "code": 402,
231           "message": "Too many users"
232         },
233         {
234           "code": 403,
235           "message": "Unauthorised access. The access role is
                not valid"
236         },
237         {
238           "code": 405,
```

```
239         "message": "Method not allowed. The requested
                method is not allowed by this server."
240       },
241       {
242         "code": 412,
243         "message": "JSON string is invalid."
244       },
245       {
246         "code": 413,
247         "message": "method key missing."
248       },
249       {
250         "code": 414,
251         "message": "accesRole key missing."
252       },
253       {
254         "code": 422,
255         "message": "The request body is unprocessable"
256       }
257       ],
258       "authorizations": {}
259     },
260     {
261       "method": "Send",
262       "nickname": "sendActuatorData",
263       "summary": "Send new data to the actuator with the
              given actuator identifier",
264       "type": "ActuatorDataResponse",
265       "notes": "The parameters go into a JSON object send
              over the WebSocket",
266       "parameters": [
267       {
268         "name": "message",
269         "description": "The payload for the
                sendActuatorData service",
270         "required": true,
271         "type": "ActuatorDataRequest",
272         "paramType": "message",
273         "allowMultiple": false
274       }
275       ],
276       "responseMessages": [
277       {
278         "code": 401,
279         "message": "Unauthorised access. The authentication
                token is missing"
280       },
281       {
282         "code": 402,
283         "message": "Too many users"
284       },
285       {
```

```
286         "code": 403,
287         "message": "Unauthorised access. The access role is
                not valid"
288       },
289       {
290         "code": 404,
291         "message": "Actuator not found"
292       },
293       {
294         "code": 405,
295         "message": "Method not allowed. The requested
                method is not allowed by this server."
296       },
297       {
298         "code": 406,
299         "message": "Value coerced for actuator data."
300       },
301       {
302         "code": 408,
303         "message": "Value invalid for actuator data"
304       },
305       {
306         "code": 409,
307         "message": "AccesRole controller already assigned"
308       },
309       {
310         "code": 411,
311         "message": "Payload reply lost"
312       },
313       {
314         "code": 412,
315         "message": "JSON string is invalid."
316       },
317       {
318         "code": 413,
319         "message": "method key missing."
320       },
321       {
322         "code": 414,
323         "message": "accesRole key missing."
324       },
325       {
326         "code": 422,
327         "message": "The request body is unprocessable"
328       }
329       ]
330     }
331     ]
332   },
333
334   {
335     "path": "/logging",
```

```
336       "description": "A general endpoint that allows to
              access any operation of any service"
337     },
338     {
339       "path": "/",
340       "description": "A general endpoint that allows to
              access any operation of any service"
341     }
342     ]
343   }
```

A.2 RESPOSTA DA REQUISIÇÃO GETCLIENTS

```
1   {
2     "method": "getClients",
3     "clients":
4     {
5       "type": "Web page",
6       "url": "http://relle.ufsc.br/labs/7/moodle"
7     }
8   }
```

A.3 RESPOSTA DA REQUISIÇÃO GETSENSORMETADATA

```
1   {
2     "method": "getSensorMetadata",
3     "sensors": [
4     {
5       "sensorId":"time",
6       "fullName":"Ball time in the distance 0, 10, 20, 30,
            40, 50, e 60 cm",
7       "description":"Time in which the ball passes the
            distance sensors 0, 10, 20, 30, 40, 50 and 60cm,
            data are given in array [d0, d1, d2, d3, d4, d5, d6
            ]. The value can be obtained by the pull method,
            however, after the sphere falls, the experiment
            sends the resulting values by the push method.",
8       "webSocketType":"text",
9       "singleWebSocketRecommended":false,
10      "produces":"application/json",
11      "values":[
12      {
13
14        "name":"d0",
15        "unit":"ms",
```

```
16        "type":"int",
17        "lastMeasured":"YYYY-MM-DDThh:mm:ssZ"
18      },
19      {
20        "name":"d1",
21        "unit":"ms",
22        "type":"int",
23        "lastMeasured":"YYYY-MM-DDThh:mm:ssZ"
24      },
25      {
26        "name":"d2",
27        "unit":"ms",
28        "type":"int",
29        "lastMeasured": "YYYY-MM-DDThh:mm:ssZ"
30      },
31      {
32        "name":"d3",
33        "unit":"ms",
34        "type":"int",
35        "lastMeasured": "YYYY-MM-DDThh:mm:ssZ"
36      },
37      {
38        "name":"d4",
39        "unit":"ms",
40        "type":"int",
41        "lastMeasured":"YYYY-MM-DDThh:mm:ssZ"
42      },
43      {
44        "name":"d5",
45        "unit":"ms",
46        "type":"int",
47        "lastMeasured":"YYYY-MM-DDThh:mm:ssZ"
48      },
49      {
50        "name":"d6",
51        "unit":"ms",
52        "type":"int",
53        "lastMeasured":"YYYY-MM-DDThh:mm:ssZ"
54      }
55      ],
56      "accessMode":{
57        "type":"pull",
58        "type":"push"
59      }
60    },
61    {
62      "sensorId":"angle",
63      "fullName":"Inclination angle of the plane",
64      "description":"Returns the current tilt angle value of
            the seesaw.",
65      "webSocketType":"text",
66      "singleWebSocketRecommended":false,
```

```
67      "produces":"application/json",
68      "values":{
69        "name":"A",
70        "unit":"degree",
71        "type":"int",
72        "rangeStep":1,
73        "rangeMinimum":-15,
74        "rangeMaximum":90,
75        "lastMeasured":"YYYY-MM-DDThh:mm:ssZ"
76      },
77      "accessMode":{
78        "type":"pull"
79      }
80    },
81    {
82      "sensorId":"video",
83      "fullName":"Video camera in front of experiment.",
84      "description":"Video camera in front of experiment,
            video stream is sent frame by frame encoded in
            base64",
85      "webSocketType":"text",
86      "singleWebSocketRecommended":true,
87      "produces":"image/png",
88      "configuration": [
89      {
90        "parameter": "width",
91        "value": 352
92      },
93      {
94        "parameter": "height",
95        "value": 288
96      }
97      ],
98      "accessMode": {
99        "type": "push",
100       "nominalUpdateInterval": 70,
101       "userModifiableFrequency": false
102     }
103   }
104   ]
105 }
```

A.4 EXEMPLOS DE RESPOSTAS DA REQUISIÇÃO GETSENSORDATA

```
1  {
2    "method": "getSensorData",
3    "sensorId": "angle",
4    "accessRole": "controller",
5    "responseData": {
```

```
    "valueNames": ["angle"],
    "data": [90],
    "lastMeasured": ["2018-05-31T18:28:43.511Z"]
  }
},
{
  "method": "getSensorData",
  "sensorId": "time",
  "accessRole": "controller",
  "responseData": {
    "valueNames": ["d0", "d1", "d2", "d3", "d4", "d5", "d6"
        ],
    "data": [0, 187, 267, 329, 381, 427, 468],
    "lastMeasured": [
      "2018-05-15T18:28:43.511Z",
      "2018-05-15T18:28:43.511Z",
      "2018-05-15T18:28:43.511Z",
      "2018-05-15T18:28:43.511Z",
      "2018-05-15T18:28:43.511Z",
      "2018-05-15T18:28:43.511Z",
      "2018-05-15T18:28:43.511Z",
    ],
  },
}
```

A.5 RESPOSTA DA REQUISIÇÃO GETACTUATORMETADATA

```
{
  "method": "getActuatorMetadata",
  "actuators": [
  {
    "actuatorId":"angle",
    "fullName":"Angle of Inclination of the Plane",
    "description":"Servo motor that tilts the plane at the
        requested angle",
    "webSocketType": "text",
    "produces":"application/json",
    "consumes":"application/json",
    "values":{
      "name": "A",
      "type":"int",
      "unit": "degree",
      "rangeMinimum": -15,
      "rangeMaximum": 90,
      "rangeStep": 1,
      "lastMeasured": "2018-05-15T18:25:43.511Z"
    },
    "accessMode":{
      "type":"pull"
```

```
22      }
23    },
24    {
25      "actuatorId":"drop",
26      "fullName":"Servo motor locking and loosening the ball"
            ,
27      "description":"Servo motor used to hold and release the
             ball, to release, send value 1 = drop.",
28      "webSocketType": "text",
29      "produces":"application/json",
30      "consumes":"application/json",
31      "values":{
32        "name": "drop",
33        "type":"boolean",
34        "unit": "binary boolean",
35        "lastMeasured": "2018-05-15T18:25:43.511Z"
36      },
37      "accessMode":{
38        "type":"pull"
39      }
40    },
41    {
42      "actuatorId":"setup",
43      "fullName":"Function tilting plane to -15 degrees and
            hold ball",
44      "description":"Function to tilt the seesaw with the
            latch open to -15 degrees and hold the ball, to
            perform send 1 = true",
45      "webSocketType": "text",
46      "produces":"application/json",
47      "consumes":"application/json",
48      "values":{
49        "name": "setup",
50        "type":"boolean",
51        "unit": "binary boolean",
52        "lastMeasured": "2018-05-15T18:25:43.511Z"
53      },
54      "accessMode":{
55        "type":"pull"
56      }
57    }
58    ]
59  }
```

A.6 EXEMPLOS DE REQUISIÇÃO SENDACTUATORDATA

```
1  {
2    "authToken": "dskds909ds8a76as675sa54",
3    "method": "sendActuatorData",
```

```
  "accessRole": "controller",
  "actuatorId": "setup",
  "valueNames": ["setup"],
  "data": [1]
},
{
  "authToken": "dskds909ds8a76as675sa54",
  "method": "sendActuatorData",
  "accessRole": "controller",
  "actuatorId": "angle",
  "valueNames": ["A"],
  "data": [45]
},
{
  "authToken": "dskds909ds8a76as675sa54",
  "method": "sendActuatorData",
  "accessRole": "controller",
  "actuatorId": "drop",
  "valueNames": ["drop"],
  "data": [1]
}
```

APÊNDICE B – Servidor Node.js

Este servidor foi implementado com Node.js e possui as seguintes dependências de pacotes: cors versão 2.8.4, express 4.16.3 e websocket 1.0.26

```
1  //Servidor Basico
2  var app = require('express')();
3  var server = require('http').Server(app);
4  var WebSocketServer = require('websocket').server;
5  var cors = require('cors')(); //permitir requisicao de
       outras origens
6  var fs = require('fs');
7  port=2018;
8
9  app.use(function(req, res, next) {
10   res.header("Access-Control-Allow-Origin", "*");
11   res.header("Access-Control-Allow-Headers", "Origin, X-
         Requested-With, Content-Type, Accept");
12   next();
13 });
14
15 app.get('/', function(req, res) {
16   res.sendFile(__dirname + '/indexfinal.html');
17 });
18
19 app.get('/metadata', function(req, res) {
20   res.sendFile(__dirname + '/metadata.json');
21 });
22
23 function agora(){ // funcao para gerar lasMeasured
24   var z = new Date();
25   var a = z.toJSON();
26   return a;
27 };
28
29 // objeto com valores de resultados dos sensores a cada
       angulo de descida
30 var data_set = {
31   "0":{"d0":0,"d1":1,"d2":2,"d3":3,"d4":4,"d5":5,"d6":6},
32   "1":{"d0":0,"d1":1,"d2":2,"d3":3,"d4":4,"d5":5,"d6":6},
33   /// continua ate 90
34   "90":{"d0":0,"d1":143,"d2":206,"d3":255,"d4":294,"d5"
         :329,"d6":360}
35 };
36
37 var d = new Date();
38 var n = d.toJSON();
39
40 var ESTADO = -3;
41 var data_tempo = { "d0":0, "d1":0, "d2":0, "d3":0, "d4":0,
       "d5":0, "d6":0, "lastMeasured":""};
42 var data_angle = { "angle":ESTADO, "lastMeasured":n};
43
44 io = new WebSocketServer({
```

```
45    httpServer: server
46  });
47
48  io.on('request', function (request) {
49
50    var connection = request.accept(null, request.origin);
51    console.log('cliente conectado');
52
53    connection.on('message', function(message){
54    if (message.type === 'utf8'){
55      try {
56      var obj = JSON.parse(message.utf8Data);
57      console.log(message);
58
59        switch(obj["method"]) { // inicio switch method
60        case "getSensorMetadata":
61          var data = fs.readFileSync('getSensorMetadata.json'
              , 'utf8');
62          connection.send(data);
63        break;
64        case "getActuatorMetadata":
65          var data = fs.readFileSync('getActuatorMetadata.
              json', 'utf8');
66          connection.send(data);
67        break;
68        case "getClients":
69          var data = fs.readFileSync('getClients.json', 'utf8
              ');
70          connection.send(data);
71        break;
72        case "getSensorData":
73          switch(obj["sensorId"]) { // inicio switch sensorId
74          case "time":
75            time = {
76            "method": "getSensorData",
77            "sensorId":"time",
78            "lastMeasured": data_tempo.lastMeasured,
79            "accessRole": "controller",
80            "responseData": {
81              "valueNames": ["d0", "d1", "d2", "d3", "d4", "
                    d5", "d6"],
82              "data": [data_tempo.d0, data_tempo.d1,
                    data_tempo.d2, data_tempo.d3, data_tempo.d4
                    , data_tempo.d5, data_tempo.d6],
83              "lastMeasured": [data_tempo.lastMeasured,
                    data_tempo.lastMeasured, data_tempo.
                    lastMeasured, data_tempo.lastMeasured,
                    data_tempo.lastMeasured, data_tempo.
                    lastMeasured, data_tempo.lastMeasured]
84           }
85          };
86          connection.send(JSON.stringify(time));
```

```
87          break;
88          case "angle":
89            angulo = {
90            "method": "getSensorData",
91            "sensorId": "angle",
92            "lastMeasured": data_angle.lastMeasured,
93            "accessRole": "controller",
94            "responseData": {
95              "valueNames": [obj.sensorId],
96              "data": [data_angle.angle],
97              "lastMeasured": [data_angle.lastMeasured]
98             }
99            };
100           connection.send(JSON.stringify(angulo));
101         break;
102         case "video":
103           vid = {
104           "method": "getSensorData",
105           "sensorId": "video",
106           "lastMeasured": "",
107           "accessRole": "controller",
108           "responseData": {
109             "valueNames": ['video'],
110             "data": [''], //imagem
                    codificada base64
111            }
112            };
113           connection.send(JSON.stringify(vid));
114         break;
115         default:
116           connection.send('{"code": 404 }');
117         }
118
119       break;
120       case "sendActuatorData":
121         var qtd = 1;
122         if ((obj.actuatorId == 'setup') && (ESTADO == -3)){
                // setup
123             var myVar1 = setInterval(myTimer1, 70, 'SETUP',
                  'A', 150);//enviando imagens
124
125           ESTADO = -15; //atualizando sensor angulo
126           data_angle['angle']= -15 ;
127           var abc = agora();
128           data_angle['lastMeasured']= abc;
129
130         } else if ((obj.actuatorId == 'angle') && (ESTADO
              == -15) && (obj.data[0] < 91) && (obj.data[0] >
               0 )){ //angle
131
132           var myVar1 = setInterval(myTimer1, 70, obj.data
                [0], 'A', 200); //enviando imagens
```

```
133             ESTADO = obj.data[0]; //atualizando sensor angle
134             data_angle['angle']= obj.data[0];
135             var abc = agora();
136             data_angle['lastMeasured']= abc;
137
138           } else if  ((obj.actuatorId == 'drop') && (obj.data
                  [0] == 1) && (ESTADO > 0)){ // DROP
139
140             var myVar1 = setInterval(myTimer1, 70, ESTADO, 'D
                    ', 250); //enviando imagens
141             //atualizando sensor time
142               data_tempo.d1 = data_set[ESTADO].d1;
143             data_tempo.d2 = data_set[ESTADO].d2;
144             data_tempo.d3 = data_set[ESTADO].d3;
145             data_tempo.d4 = data_set[ESTADO].d4;
146             data_tempo.d5 = data_set[ESTADO].d5;
147             data_tempo.d6 = data_set[ESTADO].d6;
148
149             ESTADO = -15; //atualizando sensor angle
150             data_angle['angle']= -15;
151             var c = agora();
152             data_angle['lastMeasured']= c;
153             data_tempo.lastMeasured = c;
154
155             tempo = {
156             "method": "getSensorData",
157             "sensorId": "time",
158             "lastMeasured": data_tempo.lastMeasured,
159             "accessRole": "controller",
160             "responseData": {
161               "valueNames": ["d0", "d1", "d2", "d3", "d4", "
                      d5", "d6"],
162               "data": [data_tempo.d0, data_tempo.d1,
                      data_tempo.d2, data_tempo.d3, data_tempo.d4
                      , data_tempo.d5, data_tempo.d6],
163               "lastMeasured": [data_tempo.lastMeasured,
                      data_tempo.lastMeasured, data_tempo.
                      lastMeasured, data_tempo.lastMeasured,
                      data_tempo.lastMeasured, data_tempo.
                      lastMeasured, data_tempo.lastMeasured]
164              }
165             };
166
167           setTimeout(function(){  // send SensorId:time
                  method push
168             connection.send(JSON.stringify(tempo));
169           }, 10000);
170
171         }else{ //tratar
172         if ( ESTADO == -15){
173           qtd = 140;
174           var myVar1 = setInterval(myTimer1, 70, 'SETUP', 'A'
```

```
            , 150);
175       }else{
176       var myVar1 = setInterval(myTimer1, 70, 'SETUP', 'A',
              5);
177       ESTADO = -3;
178       data_angle['angle']= -3;
179     }
180   }
181
182   function myTimer1(grau, t, max2){ //Adaptacao para
          fornecer pseudovideo no prototipo
183     var doc = 'fake/'+grau+'/'+t+'/image'+qtd+'.png'; //200
184     var se1 = fs.readFileSync(doc,'binary');
185     var base64Image = new Buffer(se1, 'binary').toString('
          base64');
186     connection.send('{"method":"getSensorData", "sensorId
          ":"video", "responseData": {"valueNames": ["video
          "], "data":["'+base64Image+'
          "]}}');
187     qtd++;
188     if ( qtd == max2 ){
189       clearInterval(myVar1); // para de enviar quadros
190     }
191   };  // fim adaptacao para fornecer pseudovideo no
          prototipo
192
193   res = {
194     "method": "sendActuatorData",
195     "lastMeasured": abc,
196     "accessRole": "controller",
197     "actuatorId": obj.actuatorId,
198     "payload": {
199       /* pode ser util para retornar um resultado, mas o
            payload e opcional */
200       "actuatorId": obj.actuatorId,
201       "valueNames": obj.valueNames,
202       "data": [obj.data]
203     }
204   };
205   connection.send(JSON.stringify(res));
206
207   break;
208
209   default:
210   connection.send('{"code": 405,"message": "Method not
          allowed. The requested method is not allowed by this
          server."}');
211
212    }// fim break
213
214     }catch (e){
215      console.log("ERRO");
```

```
216       console.log(e.message);
217     }
218    }
219   });
220
221   connection.on('close', function(e){
222     var ESTADO = -3;
223     data_angle['angle']= -3;
224     console.log('onclose');
225     });
226     connection.on('disconnect', function(e){
227     var ESTADO = -3;
228     data_angle['angle']= -3;
229     console.log('ondisconect');
230     });
231   });
232   server.listen(port, function () {
233   console.log('Server listening at port %d', port);
234 });
```

APÊNDICE C – Cliente WEB

```
1  !DOCTYPE html>
2  <html>
3  <head>
4  <title>RExLab - Inclined Plane - Prototipo SMART DEVICE</
       title>
5  <link rel="shortcut icon" type="image/x-icon" href="http://
       relle.ufsc.br/favicon.png"/>
6  <meta charset="UTF-8">
7  </head>
8  <body>
9  <link rel="stylesheet" href="https://cdn.jsdelivr.net/
       jquery.roundslider/1.3/roundslider.min.css">
10 <link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com
       /bootstrap/3.3.6/css/bootstrap.min.css" integrity="
       sha384-1q8mTJOASx8j1Au+
       a5WDVnPi2lkFfwwEAa8hDDdjZlpLegxhjVME1fgjWPGmkzs7"
       crossorigin="anonymous">
11 <link rel="stylesheet" href="http://relle.ufsc.br/flat/dist
       /css/flat-ui.css">
12 <link rel="stylesheet" href="http://relle.ufsc.br/css/app.
       css" >
13 <link rel="stylesheet" href="http://relle.ufsc.br/css/
       shepherd-theme-arrows.css">
14 <style>
15 h1{
16   text-align: center;
17   padding-bottom: 20px;
18 }
19 #slider{
20   padding-top: 20px;
21 }
22 .rs-control .rs-range-color {
23   background-color: #00aa8a;
24 }
25 .rs-control .rs-path-color {
26   background-color: #e2e2e2;
27   border:none;
28 }
29 .rs-control .rs-handle {
30   background-color: #838383;
31 }
32 .rs-control .rs-bg-color {
33   background-color: white;
34 }
35 .rs-tooltip-text{
36   font-size: 15pt;
37 }
38 #forca {
39   cursor: default;
40 }
41 .rslider{
```

```
42    margin-bottom: 30px;
43  }
44  .introjs-helperNumberLayer{
45    width:25px !important;
46    height:25px !important;
47  }
48  </style>
49  <script src="http://cdnjs.cloudflare.com/ajax/libs/jquery
        /2.1.3/jquery.min.js"></script>
50  <script src="https://cdn.jsdelivr.net/jquery.roundslider
        /1.3/roundslider.min.js"></script>
51  <div id="content">
52   <div class="container">
53    <div class="row">
54     <div id="pre_experiment">
55      <div class="row">
56       <div id="exp">
57        <div id="DivExp" class="container">
58         <div class="col-lg-12">
59          <center><h3>Plano Inclinado</h3></center>
60         </div>
61         <div class="row">
62          <div class="col-lg-6" style="margin:0px 0 15px 0;">
63           <img id="play" width="100%">
64          </div>
65          <div class="col-lg-6">
66           <div class="row controllers" style="">
67           <center>
68            <div id="slider" class="rslider rs-control rs-
                  animation" style="height: 170px; width: 170px
                  ;">
69            </div>
70           <center><p> Angulo</p></center>
71       <div class="col-lg-12 col-xs-12" style="margin: 0px 0
             5px 0;">
72       <div class="row">
73          <div class="col-lg-4 col-xs-12">
74           <input type="button" class="btn btn-fresh" id="
                enviar" value="Enviar" />
75        </div>
76        <div class="col-lg-4 col-xs-12">
77         <input type="button" class="btn btn-fresh" id="
              inicial" value="Setup" onclick="setup();"/>
78        </div>
79        <div class="col-lg-4 col-xs-12">
80         <input type="button" class="btn btn-fresh" id="drop"
               value="Soltar" onclick="drop();"/>
81        </div>
82          </div>
83         </div>
84       <div class="col-lg-12  col-xs-12" style="margin:5px 0
             5px 0;">
```

```
85          <div class="row">
86            <div class="col-lg-6  col-sm-6 col-xs-12 forcay">
87          <label for="forcay"><b><i>mg </i>cos(<span class="
                angle"></span>) [N] :</b></label>
88          <input class="input-sm" style="max-width: 70px;border
                :1px solid #cccccc; box-shadow:0 1px 1px rgba(0,
                0, 0, 0.075) inset;margin:5px;" id="forcay"
                disabled="">
89           </div>
90           <div class="col-lg-6 col-sm-6 col-xs-12 forcax">
91          <label for="forcax"><b><i>mg </i>sin(<span class="
                angle"></span>) [N] :</b></label>
92          <input class="input-sm" style="max-width: 70px;border
                :1px solid #cccccc; box-shadow:0 1px 1px rgba(0,
                0, 0, 0.075) inset;margin:5px;" id="forcax"
                disabled="">
93           </div>
94          </div>
95           </div>
96         <div class="col-lg-12 col-xs-12 tabela" style="margin
               :5px 0 5px 0;">
97        <table class="table table-condensed">
98        <thead> <tr align="center">
99         <td>d(cm)</td>
100        <td>10</td>
101        <td>20</td>
102        <td>30</td>
103        <td>40</td>
104        <td>50</td>
105        <td>60</td>
106       </tr></thead>
107       <tbody>
108       <tr align="center">
109        <td>t(ms)</td>
110        <td><input type="text" id="d1" size="1"></td>
111        <td><input type="text" id="d2" size="1"></td>
112        <td><input type="text" id="d3" size="1"></td>
113        <td><input type="text" id="d4" size="1"></td>
114        <td><input type="text" id="d5" size="1"></td>
115        <td><input type="text" id="d6" size="1"></td>
116       </tr>
117       </tbody>
118       </table>
119        </div>
120      </div> <!-- fim div row controller -->
121       </div> <!-- fim div col-lg6 controller -->
122      </div> <!-- fim div row img + controller -->
123    </div> <!-- fim div id="DivExp" class="container -->
124    <div class="col-lg-6 col-xs-6">
125     <div>
126      <h3>Teste obtendo dados via WebSockets</h3>
127      <select id="message"/>
```

```
128   <option value='{"method":"getSensorMetadata"}'>
          getSensorMetadata</option>
129   <option value='{"method":"getSensorData"}'>getSensorData
          </option>
130   <option value='{"method":"getActuatorMetadata"}'>
          getActuatorMetadata</option>
131   <option value='{"method":"sendActuatorData"}'>
          sendActuatorData</option>
132   <option value='{"method":"getSensorData", "sensorId":"
          angle"}'>getSensorData angle</option>
133   <option value='{"method":"getSensorData", "sensorId":"
          time"}'>getSensorData tempo</option>
134   <option value='{"method":"getSensorData", "sensorId":"
          video"}'>getSensorData video</option>
135   </select>
136     <input type="button" id="env_msg" value="enviar"
            onclick="sendMessage(message.value);" />
137    </div>
138    <textarea id="logs"rows="12" cols="80" ></textarea>
139   </div>
140
141 <script> //SLider
142 $(function () {
143   $("#slider").roundSlider({
144     startAngle: -15,
145     endAngle: 90,
146     value: 0,
147     min: -15,
148     max: 90,
149     sliderType: "min-range",
150     handleShape: "dot",
151     mouseScrollAction: true,
152     create: function () {
153       $('.rs-seperator').hide();
154       $('.rs-tooltip-text').append('o');
155     },
156     change: function () {
157       $('.rs-tooltip-text').append('o');
158     }
159   });
160   if (window.DeviceOrientationEvent) {
161     window.addEventListener("deviceorientation", function (
          event) {
162       if (event.gamma > rotation + 20) {
163         rotation = rotation + 20;
164         $("#slider").roundSlider({value: rotation + 10})
165       } else if (event.gamma < rotation - 20) {
166       rotation = rotation - 20;
167       $("#slider").roundSlider({value: rotation - 10})
168     }
169   });
170 }
```

```
171   $('#enviar').click(function () {
172   var angle = $("#slider").data("roundSlider").option("
          value");
173   enviar(angle);
174   });
175 });
176 </script>
177
178 <script>
179 var rpi_server = "ws://127.0.0.1:2018";
180 var lab_socket = null;
181 var token = "kGTKLsdpg6XkYxWiYFv5tIArsuUUOdJ2SaX4UVCQ";
182 var rotation = 0;
183
184 lab_socket = new WebSocket(rpi_server);
185 console.log("connect");
186
187 lab_socket.onopen = function(evt)    {   onOpen(evt) };
188 lab_socket.onclose = function(evt)   {   onClose(evt) };
189 lab_socket.onmessage = function(data) { onMessage(data) };
190 lab_socket.onerror = function(evt)   {   onError(evt) };
191
192 function onOpen(evt){
193   console.log("onOpen");
194 };
195
196 function onClose(evt){
197   console.log("onClose");
198 };
199
200 function onMessage(data){
201  try { // tentar parse JSON
202   var str = JSON.parse(data.data);
203    if ( str["method"] == "getSensorData" ){
204   switch(str["sensorId"]) {
205    case "angle":
206      document.getElementById("slider").value = str.
           responseData.data[0];
207      logs('SmartDevice => cliente', str);
208    break;
209    case "time":
210      document.getElementById("d1").value = str.responseData
           .data[1];
211      document.getElementById("d2").value = str.responseData
           .data[2];
212      document.getElementById("d3").value = str.responseData
           .data[3];
213      document.getElementById("d4").value = str.responseData
           .data[4];
214      document.getElementById("d5").value = str.responseData
           .data[5];
```

```
215       document.getElementById("d6").value = str.responseData
              .data[6];
216       logs('SmartDevice => cliente', str);
217     break;
218     case "video":
219      var img = document.getElementById("play");
220      img.src = str.responseData.data[0];
221     break;
222     default:
223      console.log('nenhum sensorId');
224     }
225   }; // fim getSensorData
226
227   if ( str["method"] == "getSensorMetadata" ){
228     logs('SmartDevice => cliente', str);
229   }; // fim getSensorMetadata
230   if ( str["method"] == "getActuatorMetadata" ){
231     logs('SmartDevice => cliente', str);
232   }; // fim getSensorMetadata
233
234   } catch (e){ // erro parse JSON
235     console.log('erro parse JSON');
236     console.log(e);
237   };
238 };
239
240 function onError(evt){
241   console.log("onError");
242 };
243
244 function logs(direcao, logs){
245   var j = JSON.stringify(logs);
246   document.getElementById("logs").value += '\n';
247   document.getElementById("logs").value += direcao;
248   document.getElementById("logs").value += ': \n';
249   document.getElementById("logs").value += j;
250 };
251
252 function drop(){
253  console.log("drop()");
254   data={
255    "authToken": token,
256    "method": "sendActuatorData",
257    "accessRole": "controller",
258    "actuatorId": "drop",
259    "valueNames": ["drop"],
260    "data": [1]
261   };
262  sendMessage(JSON.stringify(data));
263  logs('cliente => SmartDevice', data);
264 };
265
```

```
function setup(){
 console.log("setup()");
 data={
  "authToken": token,
  "method": "sendActuatorData",
  "accessRole": "controller",
  "actuatorId": "setup",
  "valueNames": ["setup"],
  "data": [1]
  };
 sendMessage(JSON.stringify(data));
 logs('cliente => SmartDevice', data);
};

function enviar(angle){
 console.log("enviar(angle)" + angle);
 data={
  "authToken": token,
  "method": "sendActuatorData",
  "accessRole": "controller",
  "actuatorId": "angle",
  "valueNames": ["A"],
  "data": [angle]
  };
 sendMessage(JSON.stringify(data));
 logs('cliente => SmartDevice', data);
};

function sendMessage(data){ // funcao sendMessage
 lab_socket.send(data);
 console.log('sendMessage(data)' + data);
};
</script>
</body>
</html>
```

APÊNDICE D – Cliente C#

O código abaixo apresenta apenas as linhas relevantes relacionadas ao funcionamento do software cliente. Os códigos criados automaticamente pela IDE referentes à interface gráfica foram suprimido.

```
1  using System;
2  using Gtk;
3  using WebSocketSharp;
4  using Newtonsoft.Json; // Trabalhar com JSON
5  using System.IO; //  Ler String
6  using System.Collections.Generic; //list
7
8  public partial class MainWindow : Gtk.Window
9  {
10   Gdk.Pixbuf castle; //buffer para imagem
11
12   public class ResponseData //classe para converter JSON em
         Objeto
13   {
14     public List<string> valueNames { get; set; }
15     public List<string> data { get; set; }
16     public List<string> lastMeasured { get; set; }
17   }
18
19   public class III //classe para converter JSON em Objeto
20   {
21     public string method { get; set; }
22     public string sensorId { get; set; }
23     public string accessRole { get; set; }
24     public string lastMeasured { get; set; }
25     public ResponseData responseData { get; set; }
26   }
27
28   public class HHH  // Classe criacao, conexao e onMessage
29   {
30     public WebSocket ws2 = new WebSocket("ws
           ://127.0.0.1:2018");
31
32     public void connect(){
33       Console.WriteLine("Conectando");
34       ws2.Connect();
35     }
36
37     public void send(string e){
38       ws2.Send(e);
39     }
40   }
41
42   HHH smart = new HHH(); //declarando o objeto smart da
       classe publica HHH
43
44   public MainWindow() : base(Gtk.WindowType.Toplevel)
45   {
```

```
46      Build();
47
48      smart.connect(); // Conectando o WebSocket
49
50      smart.ws2.OnMessage += (sender, e) =>
51      {
52        //Console.WriteLine(e.Data);
53        III jstxt = JsonConvert.DeserializeObject<III>(e.Data
              );
54        //getSensorData //getActuatorData //sendActuatorData
55
56        if (jstxt.sensorId == "time") // atualizando dados
              sensor time
57        {
58          entry1.Text = jstxt.responseData.data[1];
59          entry2.Text = jstxt.responseData.data[2];
60          entry3.Text = jstxt.responseData.data[3];
61          entry4.Text = jstxt.responseData.data[4];
62          entry5.Text = jstxt.responseData.data[5];
63          entry6.Text = jstxt.responseData.data[6];
64
65          Gtk.TextBuffer buffer; // buffer para novos textos
66          buffer = textlog.Buffer; // pegando o que ja tem no
                buffer
67          buffer.Text += "\n"; // append textlog new line
68          var json2 = JsonConvert.SerializeObject(e.Data);
69          buffer.Text += json2; // append text json envio
70        }
71
72        if (jstxt.sensorId == "video") // atualizar video
73        {
74          string base64 = jstxt.responseData.data[0].
                Substring(jstxt.responseData.data[0].IndexOf(',
                ') + 1);
75          byte[] imgBytes = Convert.FromBase64String(base64);
76          Image img2 = new Image(castle);
77          MemoryStream ms = new MemoryStream(imgBytes);
78          castle = new Gdk.Pixbuf(ms);
79          image2.Pixbuf = img2.Pixbuf;
80        }
81      };
82
83    }
84
85    protected void OnDeleteEvent(object sender,
          DeleteEventArgs a)
86    {
87      Application.Quit();
88      a.RetVal = true;
89    }
90
91    protected void cliqueSetup(object sender, EventArgs e)
```

```
92  {
93    var j = "{\"authToken\":\"
          kGTKLsdpg6XkYxWiYFv5tIArsuUUOdJ2SaX4UVCQ\"," +
94      "\"method\": \"sendActuatorData\",\"accessRole\":\"
            controller\"," +
95      "\"actuatorId\": \"setup\",\"valueNames\": [\"setup
            \"]," +
96      "\"data\": [1]}";
97    Console.WriteLine(j);
98    Gtk.TextBuffer buffer; // buffer para novos textos
99    buffer = textlog.Buffer; // pegando o que ja tem no
          buffer
100   buffer.Text += "\n"; // append textlog new line
101   buffer.Text += j; // append text json envio
102
103   try{
104     smart.send(j);
105   }catch{
106     MessageDialog md = new MessageDialog(this,
107     DialogFlags.DestroyWithParent, MessageType.Error,
108     ButtonsType.Close, "Erro ao enviar Setup (WS)");
109     md.Run();
110     md.Destroy();
111   }
112 }
113
114 protected void cliqueEnviar(object sender, EventArgs e)
115 {
116   var a = angle.Value;
117   var j = "{\"authToken\":\"
          kGTKLsdpg6XkYxWiYFv5tIArsuUUOdJ2SaX4UVCQ\"," +
118   "\"method\": \"sendActuatorData\",\"accessRole\":\"
          controller\"," +
119   "\"actuatorId\":\"angle\",\"valueNames\": [\"A\"],\"
          data\":[" + a + "]}";
120   Console.WriteLine(j);
121   Gtk.TextBuffer buffer; // buffer para novos textos
122   buffer = textlog.Buffer; // pegando o que ja tem no
          buffer
123   buffer.Text += "\n"; // append textlog new line
124   buffer.Text += j; // append text json envio
125   try{
126     smart.send(j);
127   }catch{
128     MessageDialog md = new MessageDialog(this,
129     DialogFlags.DestroyWithParent, MessageType.Error,
130     ButtonsType.Close, "Erro ao enviar comando Angle (WS)
            ");
131     md.Run();
132     md.Destroy();
133   }
134 }
```

```
135
136   protected void cliqueDrop(object sender, EventArgs e)
137   {
138     var j = "{\"authToken\":\"
          kGTKLsdpg6XkYxWiYFv5tIArsuUUOdJ2SaX4UVCQ\"," +
139     "\"method\": \"sendActuatorData\",\"accessRole\": \"
          controller\"," +
140     "\"actuatorId\": \"drop\",\"valueNames\": [\"drop\"],\"
          data\": [1]}";
141     Console.WriteLine(j);
142     Gtk.TextBuffer buffer; // buffer para novos textos
143     buffer = textlog.Buffer; // pegando o que ja tem no
          buffer
144     buffer.Text += "\n"; // append textlog new line
145     buffer.Text += j; // append text json envio
146     try {
147       smart.send(j);
148     } catch {
149       MessageDialog md = new MessageDialog(this,
150       DialogFlags.DestroyWithParent, MessageType.Error,
151       ButtonsType.Close, "Erro ao enviar comando DROP (WS)"
            );
152       md.Run();
153       md.Destroy();
154     }
155   }
156 }
```

APÊNDICE E – Cliente Java

O código abaixo apresenta apenas as linhas relevantes relacionadas ao funcionamento do software cliente. Os códigos criados automaticamente pela IDE referentes à interface gráfica foram suprimido.

```
1  import java.awt.Image;
2  import java.net.URI;
3  import java.net.URISyntaxException;
4  import java.util.Map;
5  import java.util.logging.Level;
6  import java.util.logging.Logger;
7  import javax.swing.JOptionPane;
8  import org.java_websocket.client.WebSocketClient;
9  import org.java_websocket.drafts.Draft;
10 import org.java_websocket.handshake.ServerHandshake;
11 import org.json.*;
12 import java.awt.image.BufferedImage; //image
13 import java.io.ByteArrayInputStream; // image
14 import javax.imageio.ImageIO;
15 import javax.swing.ImageIcon;
16 import sun.misc.BASE64Decoder; // image
17
18 /**
19 * @author Henrique
20 */
21
22 public class NewJFrame extends javax.swing.JFrame {
23   ExampleClient ws = null; /* Criando objeto para ws*/
24
25    /* Gravar Logs no TextArea */
26    public void GravaLog(String Msg){
27    jTextLogs.append(Msg);
28    }
29
30    /**
31    * Creates new form NewJFrame
32    */
33    public NewJFrame() {
34     initComponents();
35    }
36
37   /**
38   * This method is called from within the constructor to
         initialize the form.
39   * WARNING: Do NOT modify this code. The content of this
         method is always
40   * regenerated by the Form Editor.
41   *
42   * Codigos ocultos referente a interface grafica
43   */
44   private void cliqueConectar(java.awt.event.ActionEvent
         evt) {
45     try {
```

```
46      ws = new ExampleClient(new URI("ws://127.0.0.1:2018"));
47      } catch (URISyntaxException ex) {
48      Logger.getLogger(NewJFrame.class.getName()).log(Level.
            SEVERE, null, ex);
49      }
50      ws.connect();
51    }
52
53      private void cliqueEnviar(java.awt.event.ActionEvent
            evt) {
54      if (ws == null) {
55      JOptionPane.showMessageDialog(null, "conexao nao
            encontrada");
56      } else {
57        int a = jSlider1.getValue();
58        String j = "{\"authToken\":\"
              kGTKLsdpg6XkYxWiYFv5tIArsuUUOdJ2SaX4UVCQ\","+
59      "\"method\": \"sendActuatorData\", \"accessRole\":\"
            controller\"," +
60      "\"actuatorId\": \"angle\",\"valueNames\": [\"A\"],\"
            data\":["+a+"]}";
61        ws.send(j);
62        GravaLog("\n Cliente -> Servidor: ");
63        GravaLog(j);
64      }
65      }
66
67    private void cliqueSetup(java.awt.event.ActionEvent evt)
          {
68      if (ws == null) {
69      JOptionPane.showMessageDialog(null, "conexao nao
            encontrada");
70      } else {
71        String j = "{\"authToken\":\"
              kGTKLsdpg6XkYxWiYFv5tIArsuUUOdJ2SaX4UVCQ\"," +
72      "\"method\": \"sendActuatorData\",\"accessRole\": \"
            controller\"," +
73      "\"actuatorId\": \"setup\",\"valueNames\": [\"setup
            \"],\"data\": [1]}";
74        ws.send(j);
75        GravaLog("\n Cliente -> Servidor: ");
76        GravaLog(j);
77        }
78    }
79
80    private void cliqueDrop(java.awt.event.ActionEvent evt) {
          //GEN-FIRST:event_cliqueDrop
81      if (ws == null) {
82      JOptionPane.showMessageDialog(null, "conexao nao
            encontrada");
83      } else {
84        String j = "{\"authToken\":\"
```

```
            kGTKLsdpg6XkYxWiYFv5tIArsuUUOdJ2SaX4UVCQ\"," +
85      "\"method\": \"sendActuatorData\",\"accessRole\": \"
            controller\"," +
86      "\"actuatorId\": \"drop\",\"valueNames\": [\"drop\"],\"
            data\": [1]}";
87      ws.send(j);
88      GravaLog("\n Cliente -> Servidor: ");
89      GravaLog(j);
90      }
91    }
92
93  /**
94  * @param args the command line arguments
95  */
96  public static void main(String args[]) {
97    /* Set the Nimbus look and feel */
98    //<editor-fold defaultstate="collapsed" desc=" Look and
          feel setting code (optional) ">
99    /* If Nimbus (introduced in Java SE 6) is not available,
          stay with the default look and feel.
100   * For details see http://download.oracle.com/javase/
          tutorial/uiswing/lookandfeel/plaf.html
101   */
102   try {
103     for (javax.swing.UIManager.LookAndFeelInfo info : javax
            .swing.UIManager.getInstalledLookAndFeels()) {
104       if ("Nimbus".equals(info.getName())) {
105         javax.swing.UIManager.setLookAndFeel(info.
              getClassName());
106         break;
107       }
108     }
109   } catch (ClassNotFoundException ex) {
110   java.util.logging.Logger.getLogger(NewJFrame.class.
          getName()).log(java.util.logging.Level.SEVERE, null,
          ex);
111 } catch (InstantiationException ex) {
112 java.util.logging.Logger.getLogger(NewJFrame.class.getName
      ()).log(java.util.logging.Level.SEVERE, null, ex);
113 } catch (IllegalAccessException ex) {
114 java.util.logging.Logger.getLogger(NewJFrame.class.getName
      ()).log(java.util.logging.Level.SEVERE, null, ex);
115 } catch (javax.swing.UnsupportedLookAndFeelException ex) {
116 java.util.logging.Logger.getLogger(NewJFrame.class.getName
      ()).log(java.util.logging.Level.SEVERE, null, ex);
117 }
118 //</editor-fold>
119
120 /* Create and display the form */
121 java.awt.EventQueue.invokeLater(new Runnable() {
122   public void run() {
123     new NewJFrame().setVisible(true);
```

```
124     }
125 });
126 }
127
128 // Variables declaration -
129 /*
130 * LINHAS REFERENTES A DECLARACOES DE VARIAVEIS OCULTA
131 */
132 // End of variables declaration//GEN-END:variables
133
134 public class ExampleClient extends WebSocketClient {
135
136   public ExampleClient(URI serverUri, Draft draft) {
137     super(serverUri, draft);
138   }
139
140   public ExampleClient(URI serverURI) {
141     super(serverURI);
142   }
143
144   public ExampleClient(URI serverUri, Map<String, String>
          httpHeaders) {
145     super(serverUri, httpHeaders);
146   }
147
148   @Override
149   public void onOpen(ServerHandshake handshakedata) {
150     System.out.println("opened connection");
151   }
152
153   @Override
154   public void onMessage(String message) {
155     JSONObject my_obj = new JSONObject(message);
156     String method = my_obj.getString("method");
157
158     if ( method.equals("getSensorData")) {
159       String sensorId = my_obj.getString("sensorId");
160     if ( sensorId.equals("time")) {
161       JSONObject response = my_obj.getJSONObject("
              responseData");
162       JSONArray data = response.getJSONArray("data");
163
164       Number d1 = data.getNumber(1);
165       Number d2 = data.getNumber(2);
166       Number d3 = data.getNumber(3);
167       Number d4 = data.getNumber(4);
168       Number d5 = data.getNumber(5);
169       Number d6 = data.getNumber(6);
170       txt_d1.setText(String.valueOf(d1));
171       txt_d2.setText(String.valueOf(d2));
172       txt_d3.setText(String.valueOf(d3));
173       txt_d4.setText(String.valueOf(d4));
```

```
174         txt_d5.setText(String.valueOf(d5));
175         txt_d6.setText(String.valueOf(d6));
176 
177         GravaLog("\n Servidor -> Cliente: ");
178         GravaLog(message);
179       }
180 
181       if ( sensorId.equals("video")) {
182         JSONObject response = my_obj.getJSONObject("
              responseData");
183         JSONArray data = response.getJSONArray("data");
184         String sourceData = String.valueOf(data.get(0));
185         String base64Image = sourceData.split(",")[1];
186         byte[] imageByte = javax.xml.bind.DatatypeConverter.
              parseBase64Binary(base64Image);
187         ByteArrayInputStream inputStream = new
              ByteArrayInputStream(imageByte);
188         try {
189           BufferedImage bufImage = ImageIO.read(inputStream);
190           inputStream.close();
191           ImageIcon icon = new ImageIcon(bufImage);
192           jLabel1.setIcon(icon);
193         } catch (Exception e) {
194           e.printStackTrace();
195         }
196       }
197       }
198   }
199 
200   @Override
201     public void onClose(int code, String reason, boolean
          remote) {
202     System.out.println("Connection closed by " + (remote ?
          "remote peer" : "us") + " Code: " + code + " Reason
          : " + reason);
203   }
204 
205   @Override
206     public void onError(Exception ex) {
207     ex.printStackTrace();
208   }
209   }
210 }
```