UNIFACS UNIVERSIDADE SALVADOR
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO

LILIAN PITANGA DE OLIVEIRA

AVALIAÇÃO DE ESTRATÉGIAS DE UTILIZAÇÃO DE ARCABOUÇOS ABERTOS PARA A INTEGRAÇÃO DE SERVIÇOS EM REDES RESIDENCIAIS

Salvador
2015

LILIAN PITANGA DE OLIVEIRA

AVALIAÇÃO DE ESTRATÉGIAS DE UTILIZAÇÃO DE ARCABOUÇOS ABERTOS PARA A INTEGRAÇÃO DE SERVIÇOS EM REDES RESIDENCIAIS

Dissertação apresentada ao Programa de Pós-Graduação em Sistemas e Computação da UNIFACS Universidade Salvador, Laureate International Universities, como requisito parcial para obtenção do título de Mestre.

Orientador: Prof. Dr. Joberto S. B. Martins.

Salvador
2015

FICHA CATALOGRÁFICA
(Elaborada pelo Sistema de Bibliotecas da UNIFACS Universidade Salvador, Laureate International Universities)

Oliveira, Lilian Pitanga de

Avaliação de estratégias de utilização de arcabouços abertos para a integração de serviços em redes residenciais./ Lilian Pitanga de Oliveira.- Salvador, 2015.

148 f.: il.

Dissertação apresentada ao Programa de Pós-Graduação em Sistemas e Computação da UNIFACS Universidade Salvador, Laureate International Universities como requisito parcial à obtenção do título de Mestre.

Orientador: Prof. Dr. Joberto S. B. Martins.

1. Redes de computadores. 2. Redes residenciais. 3. Gateway. I. Martins, Joberto S. B., orient. II. Universidade Salvador – UNIFACS. III. Título

CDD: 004.6

LILIAN PITANGA DE OLIVEIRA

AVALIAÇÃO DE ESTRATÉGIAS DE UTILIZAÇÃO DE ARCABOUÇOS ABERTOS PARA A INTEGRAÇÃO DE SERVIÇOS EM REDES RESIDENCIAIS

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre em Sistemas e Computação, Universidade Salvador – UNIFACS, Laureate International Universities, pela seguinte banca examinadora:

Joberto S. B. Martins – Orientador _____
Doutor em Computação (PhD) pela Université Pierre et Marie Curie, França
UNIFACS Universidade Salvador, Laureate International Universities

Jorge Alberto Prado de Campos _____
Doutor em Spatial Information Science and Engineering, University of Maine at Orono
UNIFACS Universidade Salvador, Laureate International Universities

Romildo Martins da Silva Bezerra _____
Doutor em Ciência da Computação pela Universidade Federal da Bahia - UFBA
UNIFACS Universidade Salvador, Laureate International Universities

Salvador, 17 de novembro de 2015.

AGRADECIMENTOS

Gostaria de agradecer à minha família por todo apoio e compreensão ao longo desta jornada.

Agradeço também aos meus professores do mestrado, por deixarem suas contribuições de conhecimento. Em especial, agradeço ao Prof. José Augusto Suruagy pela paciência e pelo suporte nos relatórios finais das disciplinas de POM e ADS.

Aos meus colegas do mestrado, o meu muito obrigada! Um agradecimento especial ao meu colega e já mestre Fernando Borges, por ter contribuído com alguns conceitos utilizados neste trabalho.

Meus agradecimentos a Josiane Melo da Secretaria do Mestrado pelo apoio e a Alexsandra Aragão por me ajudar com os materiais de pesquisa na biblioteca.

Agradeço ao Maurício Mattos que me apresentou um conceito simples, mas fundamental nas minhas pesquisas para este trabalho.

Agradeço ao Eliseu Torres pelas boas contribuições em vários conceitos da minha dissertação.

Um agradecimento ao Prof. Romildo Martins e ao Prof. Jorge Campos por terem aceitado participar da minha banca.

Agradeço ao meu orientador Prof. Joberto Martins pela paciência, incentivo e motivação durante todo o processo desta dissertação.

RESUMO

A evolução das tecnologias de comunicação tem possibilitado a expansão na utilização das redes de comunicação e estas, por suas vezes, têm permitido diversas alternativas e formas de troca de informação, que ocorre entre usuários e equipamentos inteligentes. Uma forma de comunicação tem se evidenciado em diversas áreas e uma destas, que é o foco deste trabalho de dissertação, diz respeito à comunicação nas redes residenciais. Os equipamentos domésticos são capazes de estabelecer comunicação em rede, as tecnologias de comunicação estão evoluindo e os custos, reduzindo, disponibilizando aos usuários diversos serviços de redes residenciais. Esta dissertação propõe o uso de arcabouços de código aberto para desenvolver aplicações, promovendo uma avaliação comparativa entre estes, e o desenvolvimento de um protótipo de um *gateway* residencial, como estudo de caso, que concentrará os serviços de redes residenciais, permitindo a sua integração. Os arcabouços serão avaliados de acordo com os requisitos de uma aplicação para redes residenciais, bem como a escolha da linguagem de programação. O *Arduino* será utilizado para compor o cenário do *gateway* residencial e, por se tratar de uma plataforma de *hardware* aberta, tem mostrado resultados satisfatórios.

Palavras-chave: Arcabouços. Redes residenciais. *Gateway*. Serviços. OSGi.

ABSTRACT

The evolution of communication technologies has allowed the expansion in the use of communication networks and these have allowed alternatives and ways of information exchange that occurs between users and smart devices. A form of communication has been evident in several areas and of these, which is the focus of this thesis concerns the communication in home networks. The appliances are able to establish network communication, communication technologies are evolving and costs are reducing, enabling users to use home networking services. This thesis proposes the use of open source frameworks for developing applications, promoting benchmarking between these and the development of a prototype of a residential gateway, as a case study, which will focus home networking services, allowing their integration. The frameworks will be evaluated according to the application requirements of the case study, and the choice of programming platform. The Arduino is used to set the scene of the residential gateway and, because it is an open hardware platform, has shown satisfactory results.

Keywords: Skeleton. Residential Networks. Gateway. Services. OSGi.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CDI	*Context Dependency Injection*
EIA	*European Industries Association*
HAN	*Home Area Network*
HG	*HomeGateway*
HGI	*Home Gateway Initiative*
HTTP	*HyperText Transfer Protocol*
ID	*IDentification*
IDE	*Integrated Development Environment*
IEEE	*Institute for Electrical and Eletronic Engineers*
IoT	*Internet of Things*
J2EE	*Java 2 Enterprise Edition*
J2SE	*Java 2 Standard Edition*
JAR	*Java ARchive*
JCP	*Java Community Process*
JDK	*Java Development Kit*
JEP	*Java Enhancement Proposals*
JSF	*JavaServer Faces*
JSR	*Java Specification Request*
JVM	*Java Virtual Machine*
KF	*Knopflerfish*
LAN	*Local Area Network*
LED	*Light Emitting Diode*
LTE	*Long Term Evolution*

MVC	*Model- View-Controller*
OSGI	*Open Service Gateway Initiative*
PC	*Personal Computer*
PDA	*Personal Digital Assistant*
RG	*Residential Gateway*
RS	*Recommended Stardard*
SOA	*Service Oriented Architecture*
UPNP	*Universal Plug aNd Play*
USB	*Universal Serial Bus*
WAN	*World Area Network*
WIFI	*Wireless Fidelity*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta os principais conceitos que motivaram esta dissertação, as justificativas e os objetivos gerais e específicos deste trabalho e como a escrita da dissertação está organizada.

1.1 MOTIVAÇÃO

Atualmente, com os avanços da tecnologia de redes de acesso e a disponibilização de uma série de serviços através do uso de dispositivos embarcados, redes de sensores e computadores portáteis, e com a crescente capacidade computacional de equipamentos domésticos inteligentes, existe a possibilidade de que os usuários finais interajam com estes aparelhos de forma transparente. Esta particularidade nos remete às características de um paradigma computacional comumente conhecido como Computação Ubíqua (CHUNG, 2013; FRIEDEWALD, RAABE, 2011; WANG, QU, 2014; WEISER, 1991).

A Computação Ubíqua, que também possui características de pervasividade (Computação Pervasiva), é um modelo computacional introduzido há mais de duas décadas através de um artigo escrito por Weiser (1991, tradução nossa) nos seguintes termos: “o método de melhorar o uso do computador fazendo com que muitos computadores fiquem disponíveis por todo o ambiente físico, mas tornando-os invisíveis para o usuário”. Esta definição é fundamentada na seguinte visão: “As mais profundas tecnologias são aquelas que desaparecem. Elas se entrelaçam no cotidiano até que não estejam visíveis”. (WEISER, 1991, tradução nossa). Segundo Obaidat e outros (2011) a essência desta previsão era a aspiração de ter um ambiente que fosse integrado às necessidades dos usuários, onde este possuísse tecnologias da rede tradicional complementando às novas capacidades da computação avançada e de comunicação sem fio.

Os inúmeros dispositivos dotados de poder computacional devem estar interconectados a qualquer hora, invisíveis e disponíveis em qualquer lugar, constituindo esta nova infraestrutura de redes de computação. De acordo com Weiser (1994), isto caracteriza uma boa ferramenta, que é aquela que é transparente, onde o usuário deve focar na tarefa e não na ferramenta e nas

tecnologias que a suportam. Esta afirmação corrobora o objetivo da Computação Ubíqua que vem no intuito de melhorar a experiência humana e a qualidade de vida, sem que o usuário tenha necessidade de conhecer explicitamente as tecnologias de computação e comunicação utilizadas.

Este modelo exige mudanças fundamentais em aspectos diferentes da ciência da computação e engenharia, no que tange normas e tecnologias de comunicação que estão atualmente disponíveis. Estas mudanças envolvem características como mobilidade, ciência de contexto e heterogeneidade dos padrões de comunicação (OBAIDAT *et al*., 2011), que não serão detalhadas neste trabalho.

Face ao cenário de evolução exposto, a proposta desta dissertação é avaliar a utilização de alguns arcabouços abertos, com a finalidade de implantá-los em um protótipo de integração de serviço básico de redes residenciais, com vistas nos objetivos da Computação Ubíqua e Pervasiva. Para compor a estrutura física do protótipo foi escolhido o *Arduino,* visando facilitar o acionamento dos serviços básicos complementando, assim, a estrutura do *gateway* doméstico.

Na seção a seguir são mostradas as justificativas, os objetivos gerais e específicos.

1.2 JUSTIFICATIVA E OBJETIVOS

O êxito do crescimento das redes de comunicação ocorre devido à evolução das suas tecnologias atualmente disponíveis e que permitem aos usuários a troca de informações. Estas tecnologias também permitem uma comunicação heterogênea que está cada vez mais em evidência, onde usuários podem utilizar uma diversidade de aparelhos e dispositivos para efetuar a troca de dados.

Na automação industrial, os operadores podem ler informações de sensores de pressão e de temperatura, por exemplo, utilizando-se de tecnologias de rede em aparelhos como PDAs[1] e *tablets*, além de aparelhos específicos na área da indústria. Na área de saúde, as tecnologias de rede têm evoluído para permitir, por exemplo, conhecer o *status* dos pacientes, no que diz respeito a valores de pressão

[1] *Personal Digital Assistant* (Assistente Digital Pessoal) são computadores portáteis (TANENBAUM, 2003), também conhecidos como *palmtops* e *handhelds*.

arterial, frequência cardíaca e temperatura corporal. Fornecedores, de maneira geral, apresentam algumas alternativas de soluções tanto no campo da indústria, como no campo da saúde.

Nas redes domésticas, o cenário é essencialmente o mesmo. Se considerarmos o conceito da Internet das Coisas (IoT - *Internet of Things*) (ASHTON, 2009), onde "coisas" podem se comunicar, existem aparelhos como *smartphones*, TVs, refrigeradores, câmeras e diversos outros, permitindo a interação entre estes aparelhos e usuários.

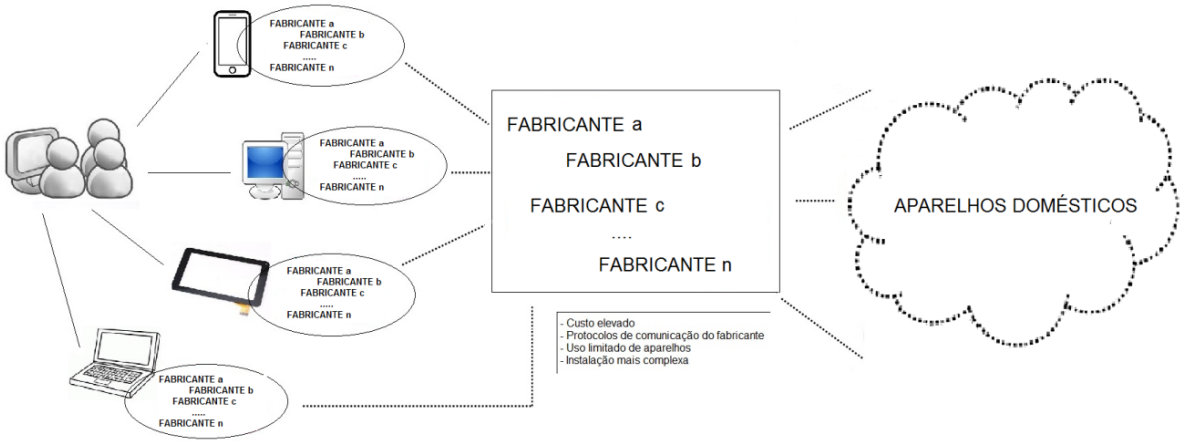
É perceptível a crescente capacidade computacional instalada nestes equipamentos, assim como existe um suporte de redes de comunicação que permitem que estes aparelhos troquem informações com o meio externo, onde é possível citar, como exemplo de tecnologias, o WiFi[2] (IEEE, 2011), o *Bluetooth* (IEEE, 2004), o *ZigBee* (IEEE, 2003; KAY, 2006; LI et al., 2014; MEDAGLIANI; MARTALÒ; FERRARI, 2011), o LTE/4G (3GPP, 2008) (em breve, o 5G (3GPP, 2015; LYNGGAARD, SKOUBY, 2015)), o PLC[3] (MUDRIIEVSKYI, 2014), o 6LowPAN (KUSHALNAGAR; MONTENEGRO; SCHUMACHER, 2007) e outras.

No âmbito das redes residenciais, empresas como Samsung (SAMSUNG, 2015), Philips (KONINKLIJKE PHILIPS, 2015) e LG (LG, 2015), também comercializam alternativas de soluções, que possibilitam a comunicação com aparelhos domésticos. A Figura 1 representa este processo de comunicação muito comum nos dias atuais.

[2] WiFi “é um termo de marketing inventado por uma aliança de um grupo de vendedores que vendiam tecnologia de rede sem fio 802.11” (COMER, 2007, p. 608).

[3] PLC é o acrônimo de *Powerline Communications* (Comunicação através da Rede Elétrica).

Figura 1 – Aparelhos domésticos com protocolos de comunicação proprietários

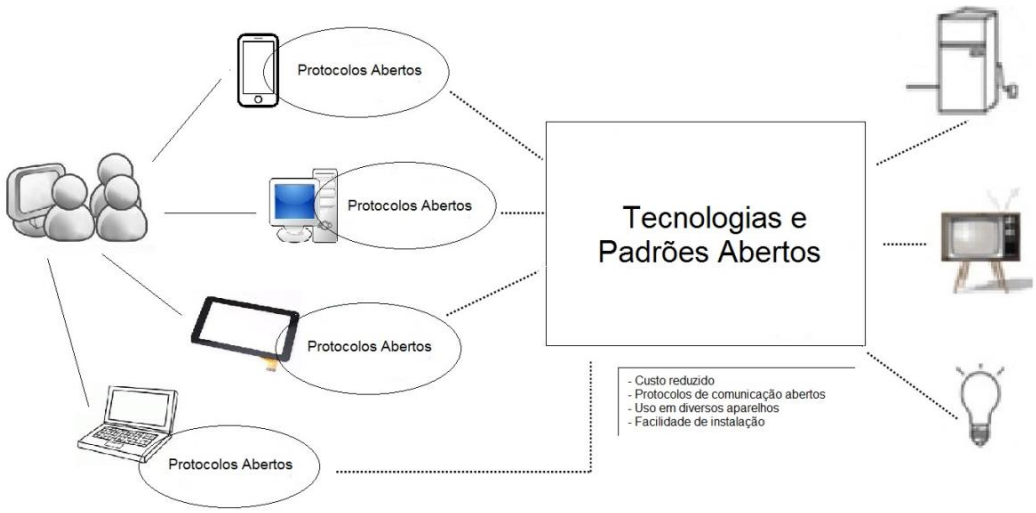


Fonte: Elaborada pela autora.

Nesta figura, é possível identificar a possibilidade e existência desta troca de informações entre usuários e aparelhos domésticos, entretanto as soluções que encontramos atualmente são proprietárias e, normalmente, restritas a determinados fabricantes. O uso destas tecnologias pode ser restrito a um número reduzido de aparelhos e a instalação dos serviços pode caracterizar um procedimento mais complexo.

A ideia a ser implementada nesta dissertação, pode ser resumida conforme a Figura 2.

Figura 2 – Proposta de solução para redes domésticas utilizando padrões abertos



Fonte: Elaborada pela autora.

Nesta figura é possível observar a possibilidade de utilizar padrões e tecnologias abertos, para promover a comunicação com os aparelhos domésticos, utilizando os protocolos de comunicação já existentes.

Como é possível identificar, os custos tendem a ser reduzidos, considerando a crescente evolução das tecnologias de redes de acesso, e a complexidade na instalação para uso destas também é igualmente reduzida, o que fomenta o uso de serviços de redes por diversos usuários numa infinidade de áreas, além das supracitadas. Além disto, concomitantemente, investir nesta alternativa de construção de aplicações utilizando tecnologias e padrões abertos existem uma diversidade de opções para os desenvolvedores de aplicações, que podem promover melhorias nos sistemas e aplicações de redes residenciais.

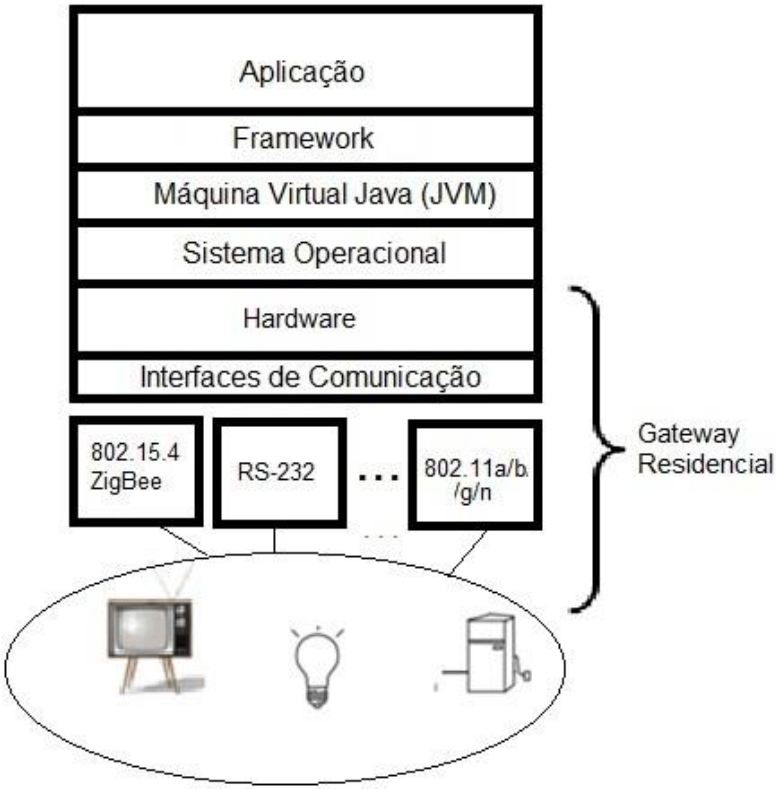
Dado este conjunto de fatores e cenário, este trabalho apresenta como justificativa promover a eficiência e a qualidade nos serviços de rede oferecidos através do uso de arquiteturas e padrões abertos de *hardware* e *software.* Estes padrões são utilizados em dispositivos, tais como *smartphones*, *tablets* e computadores, para controlar os aparelhos domésticos, que estão no ambiente das redes residenciais. Estes aparelhos devem ser computacionalmente inteligentes[4], onde lhes permita trocar informações com demais equipamentos de rede. Neste contexto são propostos arcabouços de desenvolvimento de aplicações abertos, fazendo o contraponto em relação às soluções de fornecedores, que, tipicamente, são proprietárias e fechadas.

Neste cenário, este trabalho também se justifica em desenvolver um protótipo contendo um conjunto de soluções em serviços de redes residenciais, investigando aspectos tecnológicos envolvidos com suas arquiteturas e soluções globais por melhores serviços e integração de aplicações. Além disto, é possível promover uma avaliação comparativa entre alguns arcabouços abertos, visto que já existem algumas soluções para desenvolver aplicações de redes residenciais.

A Figura 3 que segue representa a arquitetura que valida a prototipação construída nesta dissertação.

[4] Este conceito está relacionado a capacidade computacional dos equipamentos domésticos, diferente do conceito de Inteligência Computacional adotado por James Bezdek.

Figura 3 – Arquitetura de validação da prototipação



Fonte: Elaborada pela autora.

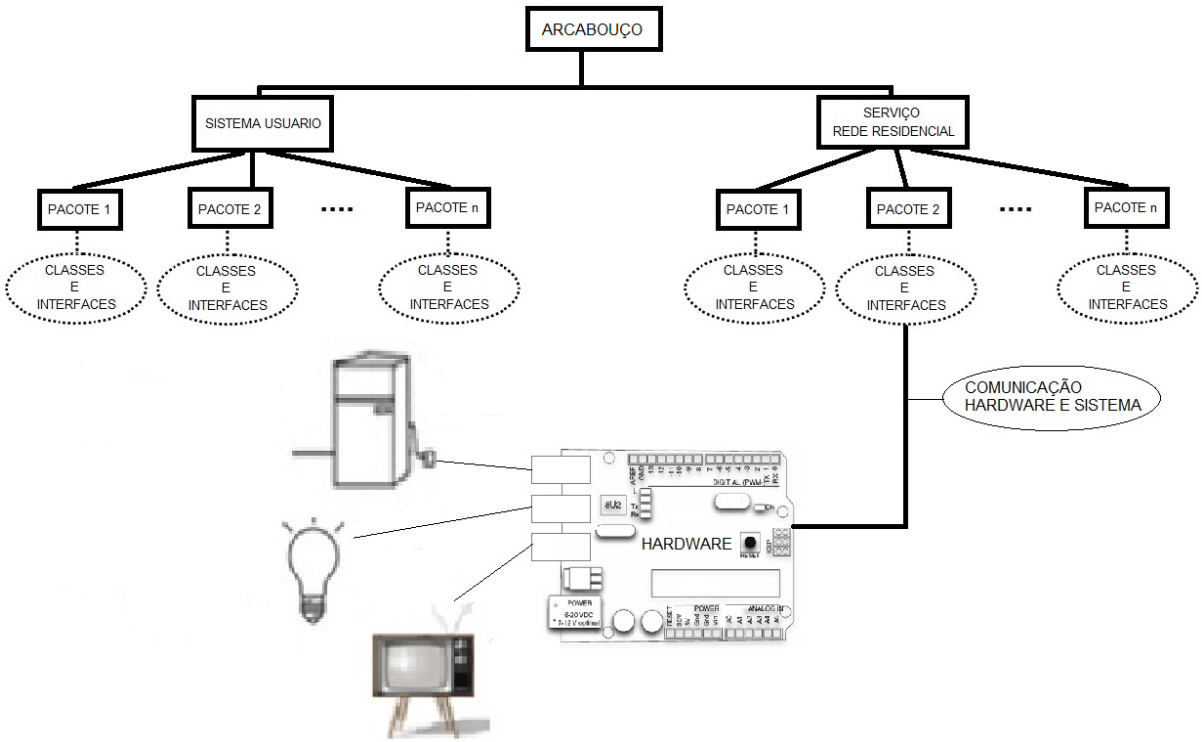
Nesta estrutura de camadas desta figura, é possível identificar as seguintes informações:

a) a camada de Aplicação que representa a aplicação a ser utilizada pelo usuário, que estará acessível em dispositivos móveis;
b) a camada do *Framework* ou arcabouço, onde os serviços de redes serão implantados;
c) a camada que representa a máquina virtual *Java*, para interpretar as informações da linguagem *Java* no sistema operacional;
d) a camada de Sistema Operacional, onde o serviço de redes serão construídos;
e) a camada de Hardware que representa o componente que fará a comunicação com os aparelhos domésticos, através das Interfaces de Comunicação, utilizando os diversos padrões de comunicação.

As camadas de Hardware, de Interface de Comunicação e dos protocolos de comunicação representam o gateway residencial, que é o controlador que fará o gerenciamento dos comandos para os aparelhos domésticos.

A Figura 4 representa uma proposta de solução a ser implementada nesta dissertação.

Figura 4 – Proposta de solução para o protótipo



Fonte: Elaborada pela autora.

Nesta figura é sugerida a utilização de um arcabouço aberto que atenda aos requisitos da aplicação de redes residenciais, atendendo também às características de orientação a serviços. Neste arcabouço é possível construir serviços básicos de redes residenciais. Estes serviços básicos são as principais tarefas que são providas pelos aparelhos domésticos e que podem ser convertidas em serviços, possibilitando o controle pelo usuário. Como exemplo, é possível citar a possibilidade de acesso do usuário aos itens que constam no refrigerador através de um sistema. Este sistema, que está representado na figura como Sistema Usuário, também será construído neste arcabouço.

Na figura é possível identificar que existirá um componente de *hardware* que fará a comunicação com os aparelhos residenciais. Este *hardware* servirá para acionar os comandos nos aparelhos. Estes comandos serão enviados para o *hardware* através dos serviços de redes residenciais.

Diante das justificativas citadas e do cenário apresentado, os objetivos desta dissertação são elencados a seguir:

a) avaliar a utilização de arcabouços abertos para o desenvolvimento de aplicações e serviços em redes residenciais, onde estes arcabouços serão baseados em *frameworks* e linguagens conhecidos;
b) desenvolver um protótipo de *gateway* residencial como estudo de caso da avaliação de arcabouços desenvolvida, utilizando o *Arduino* como plataforma de *hardware* aberta.

Estes objetivos visam a proposição e a criação de uma estrutura que permitirá ao usuário final o controle de seus aparelhos domésticos, utilizando qualquer dispositivo que esteja no ambiente residencial.

Sendo assim, é possível citar como objetivos específicos:

a) estabelecer critérios e requisitos da aplicação para facilitar a escolha do arcabouço a ser utilizado;
b) avaliar as alternativas de arcabouços disponíveis na literatura para desenvolver as aplicações;
c) comparar os arcabouços escolhidos, utilizando como parâmetros as características da aplicação desenvolvida e os recursos dos arcabouços;
d) validar a alternativa de arcabouço mais adequada, considerando os requisitos da aplicação;
e) escolher um serviço de redes domésticas para utilizar no protótipo;
f) escolher um protocolo de comunicação e adaptar ao serviço de redes já definido.

1.3 ORGANIZAÇÃO DA DISSERTAÇÃO

A escrita desta dissertação, além do que já foi apresentado neste capítulo, está secionada como segue abaixo:

a) no capítulo 2 estão os conceitos de redes residenciais, as principais aplicações, alguns protocolos conhecidos, sendo que alguns destes foram utilizados no projeto;
b) no capítulo 3 são apresentadas alternativas de *middlewares* que podem ser utilizados nas aplicações de redes residenciais, em consonância com os seus requisitos principais. Neste capítulo são apresentados alguns aspectos da arquitetura orientada a serviços e também são discutidas as características relevantes para definição do *middleware* mais indicado. É feita uma avaliação qualitativa entre os *middlewares* ressaltando a modularidade como ponto-chave;
c) no capítulo 4 são apresentadas as alternativas de uso dos arcabouços abertos, parametrizadas características de uma aplicação para redes residenciais no intuito de definir a estrutura mais adequada, além da avaliação comparativa das características relevantes entre os principais arcabouços. Estes arcabouços são baseados no *framework* escolhido no capítulo 3;
d) no capítulo 5 estão os conceitos do *gateway* residencial e as principais características da plataforma do *Arduino*;
e) o capítulo 6 apresenta a prototipação do estudo de caso, com informações sobre a codificação utilizada na construção do serviço básico de rede residencial;
f) no capítulo 7 estão as considerações finais e sugestões para outros trabalhos.

2 REDES RESIDENCIAIS

Este capítulo segue com conceitos sobre as redes de comunicação, citando as redes residenciais, e exemplifica alguns protocolos comuns utilizados na comunicação destas redes e que também podem estar disponíveis para comunicação com equipamentos domésticos.

As redes de computadores permitem a realização de atividades através de computadores separados, mas interconectados. Estas redes proporcionam a utilização de diversas aplicações, tais como (TANENBAUM, 2003):

a) as aplicações comerciais, onde é possível compartilhar recursos e utilizar o correio eletrônico para tornar a comunicação eficiente;
b) as aplicações de usuários móveis, que compreendem a comunicação sem fio entre usuários que necessitam trocar informações e necessitam de mobilidade;
c) as aplicações domésticas, que compreendem a comunicação dos usuários utilizando equipamentos em suas residências, trocando informações através da *Web* e utilizando mensagens instantâneas, além de outros recursos.

Para formar as redes de computadores e, consequentemente, utilizar as aplicações supracitadas, é necessário entender “os componentes de *hardware* e de *software* básicos” e a “infraestrutura de redes que fornece serviços para as aplicações.” (KUROSE; ROSS, 2010, p. 2).

No que diz respeito a componentes de *hardware*, para Comer (2007) o termo computador (*host* ou sistema final) pode se referir a qualquer sistema de computador que esteja conectado a uma inter-rede e execute aplicativos.

Kurose e Ross (2010) destacam que:

> Os componentes de *hardware* eram apenas PCs e servidores de aplicações. No entanto, cada vez mais sistemas finais modernos da *Internet*, como TVs, *laptops*, consoles para jogos, telefones celulares, *webcams*, automóveis, dispositivos de sensoriamento ambiental, quadros de imagens e sistemas internos elétricos de segurança estão sendo conectados à rede.

As aplicações de serviço que serão executadas pelos usuários nos sistemas finais incluem correio eletrônico, navegação na *Web*, mensagem instantânea, Voz sobre IP (VoIP), *Internet* via rádio, vídeo em tempo real, jogos distribuídos, compartilhamento de arquivos *peer-to-peer* (P2P)[5], televisão pela *Internet*, *login* remoto, dentre outras. Estas aplicações são conhecidas como aplicações distribuídas, uma vez que envolvem diversos sistemas finais que trocam informações mutuamente (KUROSE; ROSS, 2010).

Para executar estas aplicações, criam-se os componentes de *software*. Estes componentes podem ser desenvolvidos em muitas linguagens, como *Java* (ORACLE, 2015a), *C* (RITCHIE, 1993) e *Python* (PYTHON SOFTWARE FOUNDATION, 2015). O funcionamento destas aplicações depende de como ocorre o processo de comunicação e do que é necessário para que este seja estabelecido, que é função dos protocolos de comunicação (KUROSE; ROSS, 2010). Esta forma de comunicação é padronizada através do modelo de referência OSI[6] (*Open Systems Interconnection* – Interconexão de Sistemas Abertos) (BARRETT; KING, 2010, p.35; ISO/IEC, 1994), que é um modelo de referência de sete camadas [7]que descreve uma maneira de dividir o processo de comunicação em subpartes (COMER, 2007, p.244) e determina como o *hardware*, *software*, topologias e protocolos se comportam na troca de informações em uma rede (BARRETT; KING, 2010, p. 35). Os detalhes sobre como ocorre o processo de comunicação não serão abordados nesta dissertação.

Os protocolos especificam o formato das mensagens e ações apropriadas exigidas para cada mensagem (COMER, 2007) entre duas ou mais entidades comunicantes, bem como as ações realizadas na transmissão ou no recebimento de

---

[5] *Peer-to-Peer* é um tipo de comunicação entre pessoas, conhecida por permitir o compartilhamento de arquivos, onde não existe a intervenção de servidores dedicados. Exemplos de aplicações baseadas neste modelo são o *BitTorrent*, *eMule*, *Skype*, *PPLive* e outros (KUROSE, ROSS, 2010; TANENBAUM, 2003).

[6] A Organização Internacional para Padronização (*International Organization for Standardization* - ISO) (ISO/IEC, 1994) desenvolveu uma arquitetura (OSI) que permitisse que dispositivos de diferentes fabricantes trabalhassem juntos para se comunicarem com sistemas operacionais diferentes, devido à necessidade pelo uso das redes quando se tornou norma para empresas (BARRETT; KING, 2010, p. 35).

[7] As sete camadas do Modelo OSI são as camadas de Aplicação, Apresentação, Sessão, Transporte, Rede, Enlace de Dados e Física (ISO/IEC, 1994; KUROSE, ROSS, 2010; TANENBAUM, 2003). Os detalhes sobre cada uma destas camadas são serão abordados neste trabalho.

uma mensagem ou outro evento (KUROSE; ROSS, 2010). Diferentes tipos de protocolos são usados para realizar diferentes tarefas de comunicação.

Quanto à infraestrutura de redes é possível citar a *Internet*, que é uma infraestrutura de fornecimento de serviços a aplicações distribuídas. Tendo em vista os avanços na tecnologia os componentes da *Internet* estão sendo guiados pelas necessidades de novas aplicações. Isto posto é possível dizer que a *Internet* é uma infraestrutura, na qual novas aplicações estão constantemente sendo inventadas e disponibilizadas.

Para facilitar o entendimento dos conceitos das redes e por razões técnicas de projetos de redes, alguns critérios são levados em consideração. A classificação das redes através do escopo geográfico compõe um destes critérios. Nesta classificação, é possível elencar as redes de longa distância, conhecidas como WANs (*Wide Area Networks*), as redes metropolitanas, as chamadas MANs (*Metropolitan Area Networks*), as redes locais conhecidas como LANs (*Local Area Networks*), as redes domésticas (HANs – *Home Area Networks*) e redes pessoais (PANs – *Personal Area Networks)* (TANENBAUM, 2003).

Estes dois últimos conceitos de redes são utilizados nesta dissertação, pois nestas redes são instalados e utilizados os serviços de redes disponibilizados pelos aparelhos domésticos dotados de capacidade computacional. Nestes ambientes as linguagens de programação, os arcabouços abertos e os serviços serão implantados e executados em um protótipo para uma aplicação de redes residenciais desenvolvido no capítulo 6, para que seja possível efetuar as avaliações comparativas. Portanto, as outras classificações de redes não serão abordadas neste trabalho.

As redes domésticas ou redes residenciais consistem em dispositivos domésticos capazes de estabelecer comunicação “com cada um dos outros dispositivos, e todos eles estarão acessíveis pela *Internet*” (TANENBAUM, 2003). Muitos dispositivos possuem capacidade de se conectar em rede. Alguns são como seguem:

a) computadores (PC de mesa, *notebook*, PDA, periféricos compartilhados);
b) entretenimento (TV, DVD, câmera de vídeo, câmera fotográfica, equipamento estéreo);

c) telecomunicações (telefone, celular, intercomunicador, fax);
d) eletrodomésticos (microondas, refrigerador, relógio, forno, condicionador de ar, lâmpadas);
e) telemetria (medidor de consumo de serviços de utilidade pública, alarme de fumaça/arrombamento, termostato, câmaras [sic] para monitorar bebês) (TANENBAUM, 2003).

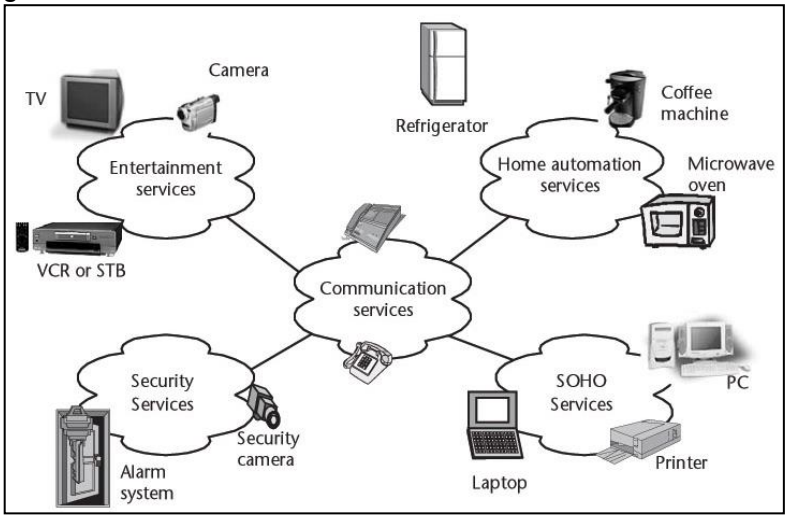
Muitos autores referenciam a rede doméstica como casa inteligente (*smart home*) como em Yamazaki (2007). Neste projeto, é possível utilizar qualquer um destes termos.

O cenário de uma rede residencial é composto por elementos, tais como:

a) rede interna, composta pelos meios físicos de conexão (cabeados ou não);
b) um controlador inteligente, que fará interconexão dos elementos da rede;
c) aparelhos e dispositivos, bem como os serviços que serão consumidos, podendo estar dentro ou fora do ambiente residencial.

A Figura 5 demonstra um típico cenário de uma rede doméstica. Esta mostra aparelhos eletrodomésticos e dispositivos que possuem capacidade computacional, podendo prover serviços para diversas finalidades, tais como entretenimento, automação residencial, *home office* e outros. Para aproveitar esta capacidade, se faz necessário ter um elemento central, que interligará os aparelhos e dispositivos, além de interpretar os comandos acionados pelo usuário.

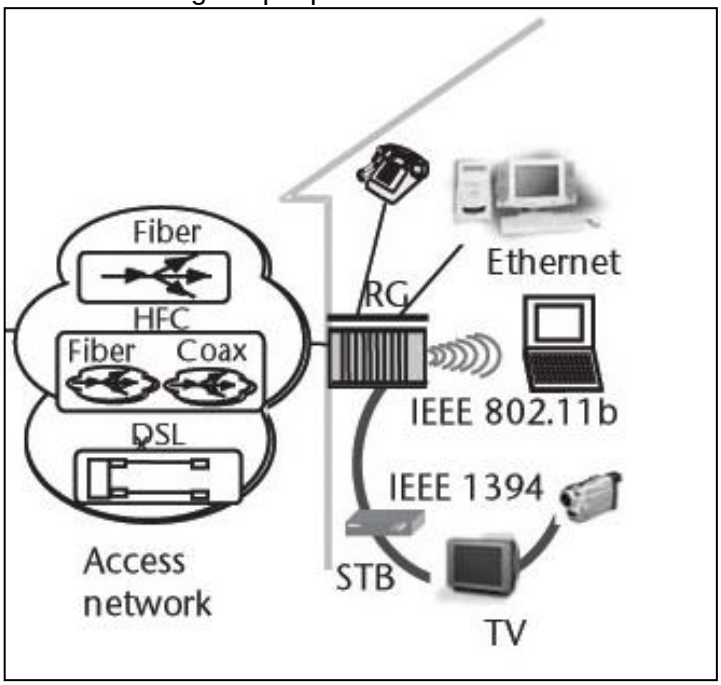
Figura 5 – Cenário de uma Rede Doméstica



Fonte: Zahariadis (2003).

Para que os componentes supracitados componham uma casa inteligente, a sua arquitetura pode ser vista como na Figura 6. Nesta, observa-se que os aparelhos domésticos devem possuir protocolos e estruturas compatíveis com a do elemento central, representado pelo RG (*Residential Gateway* – *Gateway* Residencial), para que seja possível estabelecer a comunicação.

Figura 6 – Tecnologias que podem ser Instaladas nas Redes Residenciais



Fonte: Adaptada de Zahariadis (2003).

Muitas tecnologias podem ser utilizadas no contexto das redes residenciais. Algumas destas tecnologias são (MONTPETIT, 2007; ZAHARIADIS, 2003):

a) o padrão *Wireless* (sem fio) LAN ou padrão IEEE[8] 802.11;
b) o protocolo *Bluetooth* ou padrão IEEE 802.15.1;
c) a tecnologia *ZigBee* (padrão IEEE 802.15.4);
d) o 6LowPAN;
e) o padrão USB (USB, 2000).

[8] IEEE, o acrônimo de *Institute for Electrical and Electronic Enginneers* – Instituto de Engenheiros Eletricistas e Eletrônicos, é uma organização que publica especificações para equipamentos de comunicação conhecidos como padrões (COMER, 2007).

Estes protocolos supracitados serão detalhados nas seções a seguir. Existem muitos outros padrões que podem ser instalados numa rede doméstica, mas não serão abordados nesta dissertação, para que sejam destacados os protocolos que foram utilizados na construção do serviço básico nas aplicações de redes residenciais.

2.1 PADRÃO DE REDES SEM FIO

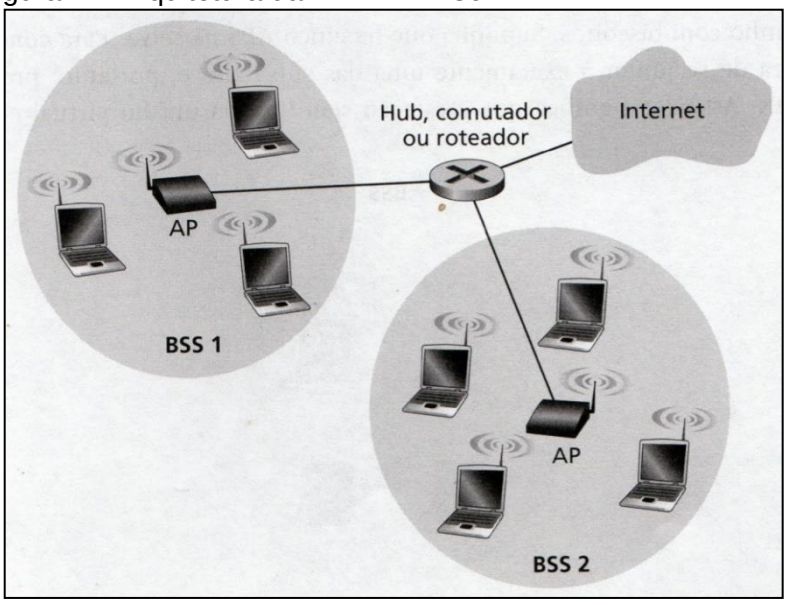
O padrão *Wireless* revolucionou as redes residenciais. Ele fornece comunicação *Ethernet* sem fio, permitindo a troca de informações entre equipamentos compatíveis com a tecnologia, através dos pontos de acesso (APs – *Access Points*) ou estação-base central. O padrão transmite informações a uma taxa de dados que varia de 11 *Megabits* por segundo (padrão 802.11b) (COMER, 2007) a quase 300 *Megabits* por segundo (padrões IEEE 802.11b, a, g e n). A arquitetura de uma rede sem fio é composta pelas estações sem fio e APs, caracterizando um conjunto básico de serviço (BSS – *Basic Service Set*) (KUROSE; ROSS, 2010).

A transmissão de sinais de uma rede sem fio ocorre através de antenas que enviam sinais de RF (radiofreqüência) através do ar (COMER, 2007; MORAES, 2010). O *hardware* LAN envia o sinal e os outros computadores recebem, caso não haja obstrução entre estes dispositivos. As LANs sem fio são configuradas para usar uma mesma frequência de rádio. Deste modo, os dispositivos alternam ao enviar os pacotes (COMER, 2007).

A

Figura 7 apresenta o exemplo de um cenário com APs em suas respectivas BSSs, conectados a um equipamento central, representado pelo *hub*, comutador ou roteador, para, a partir deste elemento central, estabelecer a comunicação com a *Internet*.

Figura 7 – Arquitetura da LAN IEEE 802.11



Fonte: Kurose e Ross (2010).

Numa rede doméstica típica esta quantidade de BSSs não é necessária, apenas um é suficiente para permitir a comunicação com o meio externo.

2.2 O PROTOCOLO *BLUETOOTH*

O *Bluetooth* ou IEEE 802.15.1 é um padrão aberto que possibilita a troca de informações entre dispositivos diversos (BARRETT; KING, 2010). Ele é caracterizado por ser um protocolo de taxas baixas, faixas curtas e baixas potências. Este padrão pode ser útil para interconectar *laptops*, aparelhos periféricos, telefones

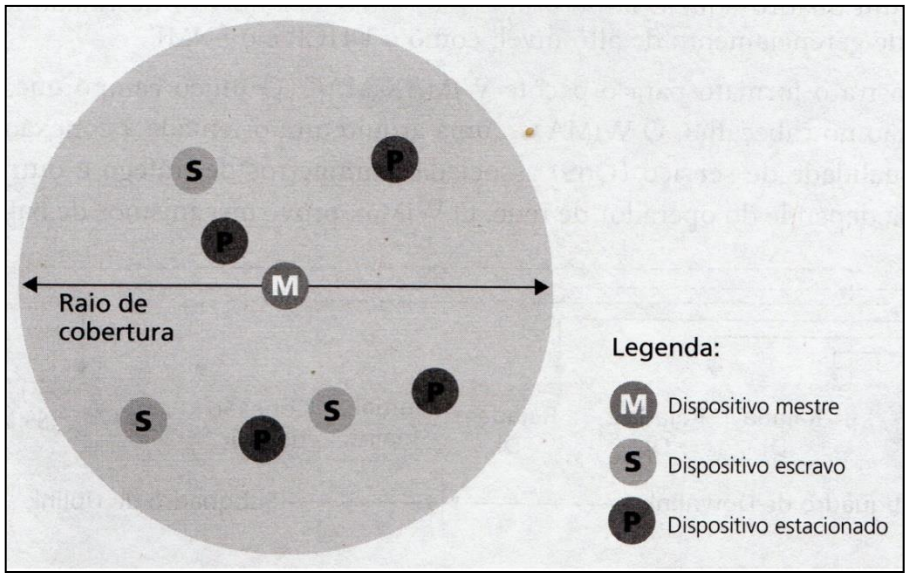
celulares e PDAs, enquanto o padrão 802.11 é uma tecnologia de acesso de taxas altas, faixas médias e altas potências (KUROSE; ROSS, 2010).

O IEEE 802.15.1 é um protocolo que foi planejado para interconectar dispositivos em ambientes com muitos ruídos, devido a sua resistência a interferências (BARRETT; KING, 2010), o que se tornou uma vantagem sobre a tecnologia de redes sem fio. O *Bluetooth* permite comunicação entre dispositivos distantes a menos de 05 metros, de acordo com Montpetit (2007). Zahariadis (2003) indica que o alcance do sistema *Bluetooth* fica entre 10cm e 10m. Por ter uma taxa de transmissão muito baixa e um alcance curto (COMER, 2007, p. 124), o *Bluetooth* é muito útil nas redes pessoais e redes domésticas. Estas redes operam em faixas de rádio não licenciadas de 2,4 *GigaHertz* e pode prover velocidades de dados de até 4 *Megabits* por segundo.

O modo de conectividade no sistema *Bluetooth* é *ad hoc* (BARRETT; KING, 2010), ou seja, não é preciso qualquer infraestrutura para interconectar os dispositivos que utilizem este protocolo. Estes dispositivos se organizam em uma picorrede (*piconet* - pequena rede) de até oito dispositivos ativos.

Um exemplo de uma picorrede pode ser visto na Figura 8. Nesta figura é possível observar que um dos dispositivos é designado como o dispositivo mestre e os outros agem como dispositivos escravos. Esta picorrede pode conter até 255 dispositivos estacionados, e estes só poderão se comunicar após o dispositivo mestre mudar os seus estados para ativos (KUROSE; ROSS, 2010).

Figura 8 – Uma picorrede Bluetooth



Fonte: Kurose e Ross (2010).

2.3 O PROTOCOLO *ZIGBEE*

O *ZigBee* é um padrão baseado na especificação IEEE 802.15.4 de camada física e subcamada MAC (DOU et al., 2009; GOMEZ; PARADELLS, 2010; IEEE, 2003; RAMYA; SHANMUGARAJ; PRABAKARAN, 2011). O padrão especifica a faixa de frequência de operação, que deve ser conforme segue no Quadro 1, onde pode variar de 868 *MegaHertz* a 2.483 *MegaHertz* e, a depender da frequência, existir parâmetros de espalhamento e parâmetros de dados diferentes. Um dado relevante é com relação às taxas de transmissão que são muito baixas (ordem de *kilobits* por segundo) (DOU et al., 2009; MONTPETIT, 2007).

Quadro 1 – Banda de Frequência e Taxa de Dados do *ZigBee*

PHY (MHz)	Frequency band (MHz)	Spreading parameters		Data parameters		
		Chip rate (kchip/s)	Modulation	Bit rate (kb/s)	Symbol rate (ksymbol/s)	Symbols
868/915	868–868.6	300	BPSK	20	20	Binary
	902–928	600	BPSK	40	40	Binary
2450	2400–2483.5	2000	O-QPSK	250	62.5	16-ary Orthogonal

Fonte: IEEE (2003).

Este protocolo compõe um conjunto de camadas construídas no topo do padrão 802.15.4. Estas camadas adicionam três pontos importantes (FALUDI, 2011):

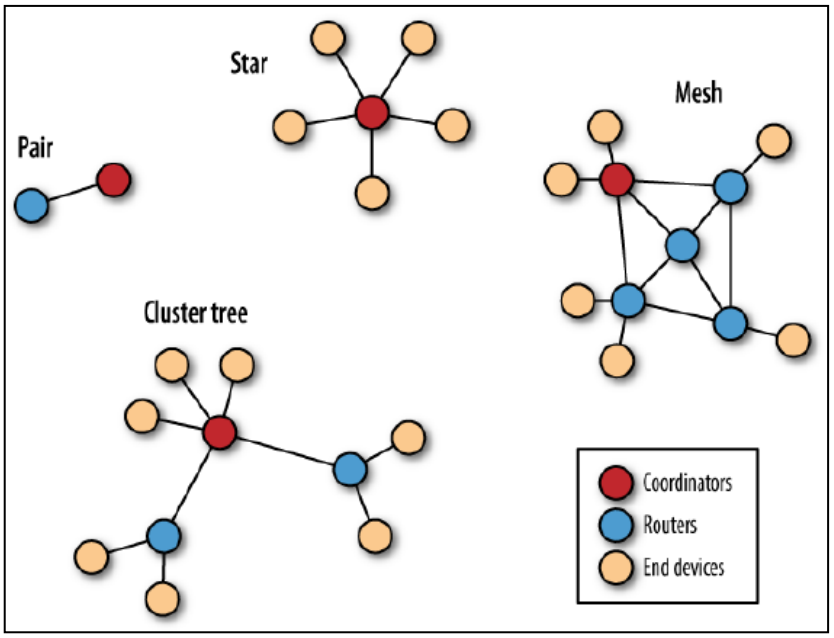
a) o roteamento, onde as suas tabelas definem como um rádio (elemento ou componente que possui a tecnologia *ZigBee*) pode passar mensagens através de uma série de outros rádios ao longo do caminho para o seu destino final;
b) a criação de redes *ad hoc*, que é um processo automatizado em que se cria uma rede de rádios dinamicamente, sem qualquer intervenção humana;
c) a malha de autorrecuperação, que é um processo onde é possível descobrir automaticamente se um ou mais rádios está em falta na rede e reconfigurar a rede para reparar qualquer falha nas rotas.

Na rede *ZigBee* existem três tipos de dispositivos (FALUDI, 2011):

a) coordenador (*coordinator*), onde existe apenas um por rede. Este rádio é responsável por formar a rede, distribuir endereços e gerenciar outras funções que definem a rede, questões de segurança e manutenção do funcionamento;
b) roteador (*router*), que é o nó destaque da rede *ZigBee*. Ele pode ligar redes existentes, enviar, receber e rotear informações;
c) dispositivo final (*end device*), que é uma versão mais simples do roteador. Como ele não age como mensageiro entre dispositivos, o seu *hardware* é mais simples para que seja possível poupar energia. Os dispositivos finais sempre precisam de um coordenador e um roteador para ser o dispositivo mestre.

A Figura 9 exemplifica os tipos de topologias de redes para a tecnologia *ZigBee*.

Figura 9 – Topologias de Redes *ZigBee*



Fonte: Faludi (2011).

O protocolo *ZigBee* possui quatro tipos, a saber (FALUDI, 2011; RAMYA; SHANMUGARAJ; PRABAKARAN, 2011):

a) estrela (*star*), que é um dos tipos de rede mais simples. O nó coordenador fica no centro da topologia em estrela e se conecta a um círculo de dispositivos finais. Todas as mensagens no sistema devem passar pelo coordenador. Os dispositivos finais não se comunicam uns com os outros diretamente;
b) malha (*mesh*), onde sua configuração emprega roteadores além do nó coordenador. Estes nós podem passar mensagens ao longo de outros roteadores e dispositivos finais, conforme necessário;
c) par (*pair*), que é a rede mais simples, com apenas dois nós. Um nó deve ser um coordenador para que a rede seja formada. O outro pode ser configurado como um roteador ou um dispositivo final;
d) árvore de cluster (*cluster tree*), que é um modelo de rede onde roteadores formam um *backbone*[9], com dispositivos finais agrupados em torno de cada roteador. Não é muito diferente de uma configuração de malha.

[9] *Backbone* é uma estrutura central que interliga os elementos na rede *ZigBee*. Na *Internet*, o *backbone* é uma estrutura formada por provedores de serviço de *Internet* capazes de transmitir informações a velocidades muito altas (provedores de nível 1) e que conecta outras redes (KUROSE; ROSS, 2010).

O *ZigBee* provê comunicação confiável e pode ser utilizado no âmbito da automação residencial em dispositivos de controle e sensores. Além disto, pode ser utilizado na automação industrial, na automação comercial, na comunicação de periféricos com o computador pessoal, dentre outras aplicações (MONTPETIT, 2007).

2.4 6LOWPAN

O padrão LoWPAN (*Low-power Wireless Personal Area Networks* - Redes de Áreas Pessoais Sem Fio de Baixa Potência) é uma rede de comunicação de baixo custo que permite conectividade sem fio em aplicações com energia limitada (KUSHALNAGAR; MONTENEGRO; SCHUMACHER, 2007). O 6LoWPAN utiliza mecanismos de transmissão de pacotes utilizando o protocolo de rede IPv6[10] (*Internet Protocol Version* 6 – Protocolo de Internet Versão 6). O *6LoWPAN*, assim como o *ZigBee*, tem como base as especificações do padrão IEEE 802.15.4 (GOMEZ; PARADELLS, 2010).

O grupo IETF (*Internet Engineering Tasf Force*) padroniza mecanismos para expandir as redes de sensores e atuadores na Internet (GOMEZ; PARADELLS, 2010). Os sensores e atuadores compõem um conjunto de dispositivos indispensáveis na automação residencial e o uso do 6LoWPAN representa uma tendência promissora para soluções baseadas em IP nas redes residenciais.

Algumas características na utilização do padrão 6LoWPAN são como seguem (KUSHALNAGAR; MONTENEGRO; SCHUMACHER, 2007):

a) uso do padrão em dispositivos com baterias de longa duração;
b) pouca memória e baixa capacidade de processamento;
c) baixa largura de banda. As faixas de frequência são as mesmas do padrão *ZigBee* (868 *MegaHertz*, 915 *MegaHertz* e 2.400 *MegaHertz*) e operam a taxas de dados de 250, 40 e 20 *kilobits* por segundo, respectivamente;
d) baixo custo;

[10] IP é um protocolo da camada rede do modelo OSI, utilizado para atribuir endereços a dispositivos que acessam a rede. A versão 4 do protocolo IP (IPv4) possui tamanho de 32 bits, endereçando um número limitado de equipamentos na rede. Por conta do aumento exponencial do número de dispositivos, a versão 6 passou a ser adotada. Esta versão possui cabeçalho de 128 bits.

e) por ser um padrão baseado em IP, a infraestrutura que já existe pode ser aproveitada;
f) ferramentas de gerenciamento e diagnósticos da rede já existem;
g) os dispositivos que são baseados no protocolo IP podem ser conectados a outras redes IP sem a necessidade de entidades intermediárias.

2.5 O PADRÃO DE COMUNICAÇÃO USB

De acordo com Pham, Syed e Halgamuge (2011), a tecnologia USB (*Universal Serial Bus*) é um padrão de comunicação que tem sido muito adotado na indústria da computação. Consideram-se três pontos que serviram de motivação para a criação desta tecnologia (USB, 2000):

a) conexão do PC ao telefone;
b) facilidade de uso;
c) expansão de portas.

O objetivo do padrão USB é permitir que os dispositivos de diferentes fabricantes tenham possibilidades de interoperar numa arquitetura aberta. Uma das características relevantes está relacionada à taxa de transferência de dados, que é de 480 *Megabits* por segundo. O sistema USB pode ser descrito em três áreas (USB, 2000):

a) interconexão USB;
b) dispositivos USB;
c) *host* (hospedeiro) USB.

A interconexão USB é a forma com que os dispositivos estão conectados e como vão se comunicar com os *hosts*. Isto inclui (USB, 2000):

a) topologia barramento, que é o modelo de conexão entre dispositivos USB e o *host*;
b) relacionamentos entre camadas onde, em termos de capacidade, as tarefas USB são realizadas em cada camada no sistema;

c) modelo de fluxo de dados, que é a maneira com que os dados se movem no sistema USB;
d) agenda USB, que proporciona suporte na transferência de dados, devido a uma interconexão compartilhada do USB.

A comunicação USB substitui, atualmente, a comunicação serial conhecida como RS-232 (PHAM; SYED; HALGAMUGE, 2011). A recomendação RS-232 foi aprovada pela EIA[11] (*Electronic Industries Associates* – Associação das Indústrias Eletrônicas) como um padrão de comunicação serial. Em 1987 foi lançada uma extensão do padrão chamado EIA-232D, com a junção da EIA com a TIA (*Telecommunications Industry Association* – Associação da Indústria de Telecomunicações). O padrão recebeu o nome de EIA/TIA RS-232 (MORAES,2010).

Este padrão de comunicação serial ponto-a-ponto permite a troca de dados entre um DTE[12] (*Data Terminal Equipment* – Equipamento Terminal de Dados) e um DCE[13] (*Data Communication Equipment* – Equipamento de Comunicação de Dados) (COMER, 2007; MORAES,2010). As interfaces DTE e DCE são dois dispositivos de comunicação como modems. Eles definem os níveis de sinais elétricos das interfaces, os tipos de conectores a serem utilizados e a sua pinagem (MORAES,2010).

As informações são transmitidas numa sequência binária, a uma velocidade baixa, na ordem de *kilobits* por segundo (COMER, 2007; MORAES, 2010), com alcance máximo de três metros. Quanto aos conectores, o padrão especifica o DB-9 (nove pinos) e o DB-25 (vinte e cinco pinos) (MORAES, 2010). Este protocolo ainda é utilizado em áreas muito específicas, como para estabelecer comunicação entre equipamentos mais antigos na indústria e também na área de TI.

O RS-232 especifica a transmissão dos caracteres. Algumas informações como *bit* de começo, *bits* de dados, *bit* de paridade e *bits* de parada são configuradas no emissor e no receptor para efetivar a transferência dos dados. O

[11] EIA é uma organização que publica especificações para equipamentos de comunicação conhecidos como padrões (COMER, 2007).

[12] DTE é relativo ao tipo de interface no equipamento terminal, por exemplo, um micro conectado a um modem é um equipamento terminal (MORAES, 2010).

[13] DCE é relativo ao tipo de interface do equipamento de comunicação (MORAES, 2010).

RS-232 é configurado de modo que cada caractere consista em sete *bits* de dados (COMER, 2007). Antes da emissão dos *bits* de dados, o *bit* de começo, que é um *bit* extra de valor 0, é transmitido pelo emissor para indicar que uma sequência binária será enviada. O *bit* de paridade é opcional e tem a função de detectar erros na transmissão (TANENBAUM, 2003). O *bit* de parada indica o fim da transmissão dos dados (COMER, 2007).

Uma característica que vale ressaltar, diz respeito à configuração das interfaces seriais no sistema operacional, onde recebem uma nomenclatura para identificação. No sistema operacional *Windows* (MICROSOFT, 2015), a nomenclatura atribuída a estas portas seriais são COM e LPT seguida de um número, de acordo com a quantidade de portas. No sistema operacional *Linux* (LINUX FOUNDATION, 2015), a descrição é seguida por TTY (*Teletype Writer*[14]). No padrão de comunicação USB esta nomenclatura se mantém, quando alguns tipos de *hardware* são instalados.

Na seção 5.1 sobre a construção do protótipo de um *gateway* residencial é necessário configurar o *Arduino* no sistema operacional. Neste processo de configuração, foi atribuída uma identificação de porta serial ao *hardware* que está relacionada ao padrão RS-232.

A seguir, são apresentadas algumas alternativas de soluções ou *middlewares* para utilização no desenvolvimento das aplicações de redes residenciais.

[14] Máquina de escrever útil para transmitir dados.

3 *MIDDLEWARES* PARA APLICAÇÕES EM REDES RESIDENCIAIS

Neste capítulo são apresentadas as alternativas de *middlewares* que possibilitam o desenvolvimento de aplicações para redes residenciais. Este capítulo também apresenta características de uma arquitetura orientada a serviços, algumas soluções de *middlewares* que podem ser utilizadas nas aplicações de redes residenciais e apresenta, ao final do capítulo, qual a solução mais viável para desenvolver um serviço para estas aplicações de redes residenciais.

As tecnologias de redes com fio e sem fio provêm acesso a redes de dados e voz e permitem conectividade de banda larga. Como as tecnologias de acesso evoluíram dentro do contexto das redes residenciais, atualmente os aparelhos domésticos podem fornecer serviços que podem ser controlados através do uso de dispositivos, incluindo *smartphones*, PDAs e computadores pessoais.

Analisando o modelo OSI, estas tecnologias de redes utilizadas para permitir esta comunicação com os aparelhos domésticos encontram-se nas camadas inferiores[15]. Estas camadas fazem a interligação com diferentes tipos de meios físicos de rede. Para que exista a interoperabilidade entre estes diversos meios físicos e a interconexão de dispositivos de fabricantes e padrões de redes diferentes é necessário desenvolver *softwares*, arquiteturas e sistemas que são conhecidos como *middlewares* (ZAHARIADIS, 2003).

Middlewares são *softwares* para o desenvolvimento, implantação, execução e interação de aplicações. Estas camadas de *software* têm a função de interconectar dois sistemas (COMER, 2007, p. 537). Segundo Ibrahim (2009), o *middleware* evoluiu a partir de um princípio básico (ocultando detalhes da rede, a partir de aplicações em sistemas sofisticados que lidam com muitas funcionalidades importantes para aplicações distribuídas) fornecendo suporte para a distribuição, a heterogeneidade e mobilidade. Estes *softwares* proporcionam um conjunto comum de interfaces e funções para ocultar a heterogeneidade de dispositivos ou recursos e permitir que diferentes tipos de recursos possam se comunicar de forma transparente.

[15] Nesta dissertação, as camadas inferiores referem-se às camadas Física, de Enlace de Dados e de Rede.

Ao analisar a possibilidade de troca de informação dos aparelhos domésticos em rede, uma lâmpada, pode, por exemplo, fornecer serviços como: ligar, desligar, aumentar e diminuir a intensidade do seu brilho; a TV pode fornecer os serviços de, além de ligar e desligar, trocar de canal, aumentar, diminuir ou interromper a saída de áudio e outras. Neste projeto, no contexto do desenvolvimento de uma aplicação para controlar os aparelhos domésticos, é necessário converter estas tarefas em serviços de rede e estes poderão ser manipulados pelo usuário.

O usuário poderá ter acesso a um sistema, que objetiva ser de fácil uso, onde estas tarefas (ou serviços) estarão disponíveis para serem controladas. Os comandos acionados pelo usuário, para que determinado equipamento execute determinada tarefa, serão interpretados por um sistema central (que concentrará todos os serviços de todos os equipamentos). Após a interpretação dos comandos, um sinal será enviado para o equipamento, que realizará a tarefa escolhida pelo usuário.

Recentemente, há uma tendência no desenvolvimento de aplicações móveis baseadas em arquitetura orientada a serviços (SOA – *Services Oriented Architecture*) em várias áreas e dentre estas áreas é possível citar as redes residenciais e a Internet das Coisas (LEE; LEE; WANG, 2015). A orientação a serviços é um paradigma de projeto de *software* independente da tecnologia, baseada em um conjunto de princípios que visam produzir *software* reutilizável, flexível, interoperável e modular. Os princípios que caracterizam o projeto SOA buscam restringir o acoplamento do componente à interface ou nível do contrato e formato de mensagem (CRUZ; BOUTALEB; TIANFIELD, 2013).

Para construir aplicações baseadas em SOA, o conjunto de princípios que devem ser adotados é composto por (VASILIEV, 2007):

a) o acoplamento fraco, que representa uma relação que permite que a lógica subjacente de um serviço não impacte em outros serviços utilizados dentro da mesma aplicação. Acoplamento fraco é o princípio fundamental de orientação para o serviço. Construir serviços com partes de *software* fracamente acopladas permitem que os outros princípios fundamentais da orientação a serviços sejam mantidos, tais como a reutilização de serviços, a autonomia de serviços e serviços sem estado;

b) o contrato de serviços, que representa descrições de serviço e outros documentos informando como um serviço será acessado programaticamente;
c) a abstração da lógica subjacente, onde um serviço expõe publicamente apenas lógica descrita no seu contrato, ocultando os detalhes da implementação dos consumidores de serviços. Isto significa que os serviços interagem uns com os outros apenas através das suas interfaces públicas;
d) o conceito de composição, que representa a capacidade dos serviços de serem agrupados em serviços compostos;
e) a autonomia, onde os serviços controlam apenas a lógica que eles encapsulam. Dividir a lógica da aplicação dentro de um conjunto de serviços autônomos permite construir aplicações flexíveis, alcançando o baixo acoplamento, reutilização e aspectos de composição;
f) a reusabilidade, que é obtida mediante a distribuição da lógica da aplicação entre os serviços, para que cada um deles possa ser potencialmente utilizado por mais de um cliente de serviço. Construção de serviços reutilizáveis dá suporte ao princípio de conceito de composição;
g) *statelessness,* onde os serviços não mantêm seus estados específicos para uma atividade. Construir serviços sem estado incentiva o baixo acoplamento, reutilização e o conceito de composição;
h) a interoperabilidade, que é alcançada desde que os serviços interajam uns com os outros através de interfaces que sejam independentes da plataforma e da implementação;
i) a descoberta, que se refere a mecanismos padrão que tornam possível a descrição de serviços que serão descobertos pelos seus clientes. A especificação UDDI (*Universal Description, Discovery, and Integration* – Descrição, Descoberta e Integração Universal) fornece tal mecanismo, que permite a publicação de documentos de descrição de serviço em um registro baseado em XML (*eXtensible Markup Language* – Linguagem de Marcação Extensível), tornando-os disponíveis para uso público.

Algumas características que também podem ser consideradas em aplicações orientadas a serviço são a granularidade e a coesão. Em alguns aspectos granularidade é a consequência geral de acoplamento e coesão, ou seja, estas características estão diretamente relacionadas entre si (MCAFFER; VANDERLEI; ARCHER, 2010).

Para Mcaffer, Vanderlei e Archer (2010), a granularidade é a medida da quantidade de código que está em um pacote. Pacotes de granularidade grossa são fáceis de gerenciar, mas são inflexíveis e aumentam o volume do sistema. Pacotes de granularidade fina controlam melhor o sistema, mas exigem mais atenção.

Já a coesão é uma visão interna da relevância dos elementos de um pacote para outro. Em um pacote altamente coeso, todas as suas partes estão diretamente relacionadas e focadas em um tópico mais restrito. Pacotes de baixa coesão são mal definidos. Pacotes altamente coesos são mais fáceis de testar e reutilizar, e eles permitem a entrega apenas da função necessária.

No desenvolvimento das aplicações residenciais, foram considerados importantes alguns requisitos para o seu funcionamento, baseando-se nas características de aplicações fundamentadas nos conceitos de SOA vistas acima, tais como:

a) modularidade, para que exista a possibilidade de, por exemplo, iniciar um serviço, interromper outro serviço e instalar um novo, dentro da mesma aplicação, percebendo que este conjunto de serviços poderá ser acionado pelo usuário. Além disto, o sistema modular evita a carga de bibliotecas duplicadas para funcionamento destes serviços;
b) reusabilidade, devido a existência de muitos serviços comuns, como, por exemplo, “ligar” e “desligar”, evitando que o mesmo código seja repetido para inúmeros serviços e, com isto, o aproveitamento de código se torna relevante;
c) integração de tecnologias, para que uma diversidade de equipamentos domésticos que possuem uma diversidade de tecnologias e protocolos instalados, e, consequentemente, proveem serviços básicos de redes, possam estar integrados, tendo o suporte da aplicação para permitir tal integração;

d) adaptabilidade, que permitirá à aplicação aglutinar novos serviços para disponibilizar ao usuário, à medida que novos aparelhos surgem na rede residencial e uma vez que eles possuam capacidade computacional para trocar informações.

Dados os requisitos para o desenvolvimento da aplicação em redes residenciais com diferentes suportes de linguagens de programação, existem algumas destas que podem atender estes requisitos, dentre as quais é possível salientar *Python*, *Ruby* (RUBY, 2015), *C++* e *Java*. Dentre estas citadas, a linguagem *Java*, além de ser conhecida e utilizada para uma variedade de aplicações, tem se mostrado útil na construção de aplicações para a *Internet* das Coisas, por também possuir uma variedade de bibliotecas e *frameworks*. Esta linguagem atende os requisitos da aplicação do projeto - que pode ser considerada como uma aplicação *Web* - além de permitir o uso em sistemas embarcados.

A linguagem *Java*, por ser rica em *frameworks*, disponibiliza muitas soluções para atender aos requisitos elencados para a programação da aplicação. Dentre as soluções *Java* possíveis, tem-se algumas alternativas, como o *JavaServer Faces* (CHÁVEZ, 2005; JCP, 2010; ORACLE, 2010) e *Contexts and Dependency Injection* (JCP, 2014b; RED HAT, 2015), o *JigSaw* (ORACLE, 2015b) e o *Open Service Gateway initiative* (OSGI, 2013; VILAS et al., 2010).

A seguir estão descritas as características de cada uma das soluções que foram escolhidas como alternativa para a aplicação.

3.1 *JAVASERVER FACES* E *CONTEXTS AND DEPENDENCY INJECTION*

JavaServer Faces (JSF) é um *framework* que incorpora características do modelo MVC (*Model-View-Controller* – Modelo-Visão-Controlador) para *Web* e de um modelo de interfaces gráficas baseado em eventos. MVC foi um modelo prevalente em J2EE (*Java 2 Enterprise Edition*). A razão para utilizar o MVC é a possibilidade de separar a lógica dentro de três unidades distintas: o Modelo, a Visão e o Controlador (YU; YOU; TSAI, 2010), para que seja permitido oferecer uma separação limpa entre a apresentação e a lógica (JUNWU; JUNLING, 2010).

O JSF também possibilita a criação de aplicações *Web*, através de uma arquitetura de componentes, um conjunto padrão de *widgets* (botões, recursos de

hyperlinks, caixas de checagem, caixas de textos e outros) de interface de usuário e uma infraestrutura de aplicação. Ele pode manter automaticamente os componentes de interface do usuário sincronizados com objetos *Java* que coletam informações de entrada do usuário e responder aos eventos (clicar em um botão, alterar informações de caixas de textos etc.), que são chamados *beans* de suporte. E ainda possui um sistema de navegação e suporte para outras linguagens (MANN, 2005; ORACLE, 2013).

Além destas características, o JSF utiliza recursos da *Applications Programming Interfaces*[16] (APIs – Interfaces de Programação de Aplicações) de *Servlets*. Um *servlet* funciona como um servidor de aplicações, atendendo a requisições do usuário. Eles ajudam a fornecer acesso seguro em aplicações *Web*, interação com banco de dados, geração dinâmica de documentos (DEITEL, 2005), segurança, *logging*, eventos de ciclo de vida, filtros, *packaging* (empacotamento) e implementação (MANN, 2005), dentre outros. Estes recursos formam a base do JSF e está disponível usando APIs *Java* padrão e arquivos de configuração baseados em XML (ORACLE, 2010).

Para utilizar o JSF, é importante adicionar alguns recursos de gerenciamento de dependências. Um dos recursos adequados para funcionar em conjunto com esta solução é o *Contexts and Dependecy Injection* (CDI – Injeção de Dependências e Contextos).

O CDI define um conjunto de serviços para ambientes da linguagem *Java* que facilitam a construção de aplicações, influenciando uma integração simples entre as camadas *Web* e lógica de negócios, resultando num modelo de programação simplificada para aplicações *Web*. Também define um conjunto padrão de serviços de gerenciamento de componentes da *Java*.

A solução CDI dispõe de uma camada de ciclo de vida e um modelo de interação sobre os tipos de componentes da *Java* existentes, incluindo os *beans* gerenciados e *Enterprise Java Beans* (EJBs)[17] (JCP, 2007a). Os serviços que o CDI fornece são como seguem (KING et al., 2015):

---

[16] Interfaces de Programação de Aplicações ou bibliotecas de classes *Java* são pacotes coletivos predefinidos para serem reutilizados em outras aplicações (DEITEL, 2005, p.33).

[17] EJB é um *framework* da plataforma JEE que roda em um *container* de um servidor de aplicação (JCP, 2007a).

a) um ciclo de vida melhorado para objetos *stateful* (que mantém o seu estado), vinculados a contextos bem definidos;
b) uma abordagem *typesafe* (certificado de que o programa esteja bem definido) para injeção de dependência;
c) interação com objetos através de um mecanismo de notificação de eventos;
d) uma melhor abordagem para associar interceptadores a objetos, juntamente com um novo tipo de interceptador, chamado de decorador, que é mais adequado para uso na resolução de problemas de negócio;
e) uma interface de Provedor de Serviços para desenvolvimento de extensões portáveis para o *container*.

O CDI provê uma proposta geral de esquema de injeção de dependência que pode ser utilizado por qualquer componente. A solução apresenta um conjunto pré-definido de escopo e anotações e cada CDI tem um ciclo de vida distinto determinado pelo escopo a que pertence (JCP, 2014b; RED HAT, 2015).

O Quadro 2 abaixo relaciona o escopo do CDI, suas anotações e respectivas descrições:

Quadro 2 – Escopo do CDI

Escopo	Descrição
@RequestScoped	Esta anotação é compartilhada através do uso do tamanho de uma requisição simples.
@ConversationScoped	Esta anotação é compartilhada através de múltiplas requisições na mesma sessão do Protocolo de Transferência de Hipertexto[18] (HTTP - *HyperText Transfer Protocol*), mas apenas se existe uma comunicação ativa mantida.
@SessionScoped	Esta anotação é compartilhada entre todas as requisições que ocorrem na mesma sessão HTTP e é encerrada quando a sessão é encerrada.
@ApplicationScoped	Esta anotação ficará ativa durante o tempo em que a aplicação estiver executando e será encerrada quando a aplicação for encerrada.
@Dependent	Esta anotação nunca é compartilhada entre pontos de injeção. Qualquer injeção de um *bean* dependente é uma nova instância cujo ciclo de vida está vinculado ao ciclo de vida do objeto em que está sendo injetado.

Fonte: Adaptado de Red Hat (2015).

[18] Protocolo da camada de aplicação da *Web*, que define como clientes *Web* requisitam páginas *Web* aos servidores e como eles as transferem aos clientes (IETF, 1999).

Para aplicações que exigem as características de ciclo de vida de pacotes, gerenciamento de dependências de pacotes e adaptabilidade, esta solução pode ser suficiente.

3.2 *JIGSAW*

É um projeto que implanta um sistema modular para *Java*, com a proposta de tornar a plataforma *Java* SE (*Java* 2 *Standard Edition*) e o JDK (*Java Development Kit* – *Kit* de Desenvolvimento *Java*) escalonáveis, melhorar a segurança e a manutenção de aplicações na plataforma *Java* SE, garantir melhor desempenho das aplicações e facilitar a construção e manutenção das bibliotecas e grandes aplicações para as plataformas *Java* SE e EE (ORACLE, 2015b).

O projeto ainda propõe instalar um sistema padrão de módulos para a plataforma *Java* SE e aplicar este sistema. O sistema de módulos deve ser capaz de modularizar o JDK e outras bases de código, para possibilitar o acesso a todos os desenvolvedores (ORACLE, 2015b).

Documentos, conhecidos como JEPs (*Java Enhancement Proposals* – Propostas de Melhorias *Java*) trazem informações e códigos relacionados ao projeto. Existem muitas propostas e, dentre estas, estão o JEP 200, o JEP 201 e o JEP 220 (ORACLE, 2015b), que serão citados a seguir.

O JEP 200 - *The Modular* JDK (JDK Modular) define uma estrutura baseada nos aspectos de modularidade para o JDK, dividindo-o em um conjunto de módulos que podem ser combinados durante a compilação, a construção, a instalação ou a execução. Atualmente o ambiente de programação *Java* não possui recursos de modularidade, o que torna necessário o uso de ferramentas adicionais para programar aplicações modulares (ORACLE, 2015c).

O JEP 201 – *Modular Source Code* (Código-fonte Modular) propõe a reorganização do código-fonte do JDK em módulos, a fim de melhorar o sistema de compilação de módulos. O código-fonte do JDK atual data da década de 90 e sua estrutura de armazenamento de arquivos dificulta a organização e manutenção. O

novo esquema visa reestruturar todo o código para facilitar a manutenção (ORACLE, 2015d).

O JEP 220 – *Modular Runtime Images* (Imagens de Execução Modulares) trata da reestruturação do JDK e das imagens modulares dinamicamente para acomodar módulos e melhorar o desempenho, segurança e manutenibilidade, definindo um esquema URI (*Universal Resource Identifier* – Identificador de Recurso Universal) para nomear módulos, classes e recursos armazenados em uma imagem em tempo de execução sem revelar a estrutura interna ou o formato da imagem. A estrutura de imagens atual é dividida entre imagens JRE (*Java* SE *Runtime Environment* – Ambiente de Execução *Java* SE) e JDK e ambos contêm os diretórios *lib* e *bin*, dentre outras características que diferem a estrutura de imagens do JRE e do JDK. A proposta atual visa eliminar estas diferenças (ORACLE, 2015e).

Para aplicações que tenham como requisito a modularidade, adaptabilidade e integração de tecnologias, utilizar esta solução pode ser suficiente e ainda prover mais benefícios, como facilidade de manutenção da aplicação.

3.3 OPEN SERVICES GATEWAY INITIATIVE

O *framework Open Service Gateway initiative* (OSGi) é um conjunto de especificações que define um sistema de componente dinâmico para *Java*. Estas especificações permitem um modelo de desenvolvimento em que as aplicações são (dinamicamente) compostas por muitos componentes diferentes (reutilizáveis). As especificações do OSGi permitem que componentes ocultem as suas implementações de outros componentes durante a comunicação através de serviços, que são objetos especificamente compartilhados entre componentes (OSGI, 2015a).

A Aliança OSGi formou-se em Março de 1999 e atualmente possui seis especificações, a saber (OSGI, 2015a):

a) a versão *Release* 1 (R1) que foi liberada em Maio de 2000 e especifica, além de outros tópicos, as classes e pacotes do *framework* e algumas APIs de serviços que o OSGi suporta (OSGI, 2000);

b) a versão *Release* 2 (R2) que foi liberada em Outubro de 2001 e melhora e estende as APIs de serviços de R1 e incrementa mecanismos de gerenciamento (OSGI, 2001);
c) a versão *Release* 3 (R3), liberada em Março de 2003 e conta com especificações de serviços para o protocolo UPnP e implantação de uso do XML (OSGI, 2003);
d) a versão Release 4 (R4) contém um conjunto de especificações e, dentre estas, uma que foi produzida pelo grupo de especialistas em OSGi Residencial. O foco estava no mercado crescente das redes domésticas. A união de entidades como *Broadband Forum*, UPnP *Forum* e HGI tornou possível a criação de um modelo de dispositivos de *gateway*, além de protocolos para gerenciamento em redes residenciais. A primeira versão (4.0) foi liberada em Outubro de 2005. A versão mais recente (4.3) foi liberada em Maio de 2012 (OSGI, 2012a);
e) a versão *Release* 5 (R5), liberada em Junho de 2012, possui um conjunto de especificações e, dentre estas, definições técnicas para cenários empresariais. A especificação do OSGi *Compendium* foi liberada em Maio de 2013 com definições técnicas acerca de serviços OSGi (OSGI, 2012b);
f) a versão *Release* 6 (R6) para especificação do núcleo OSGi foi liberada em Junho de 2014, mas as atualizações para definições técnicas para as redes residenciais e ambientes corporativos foram liberadas em Julho de 2015 (OSGI, 2014).

Ao longo do período de formação da Aliança OSGi, foram criados os *containers*[19] de *bundles* (um conjunto de componentes de execução em um formato de arquivo JAR, que entrega um serviço. Os *bundles* serão detalhados na seção 3.3.1) de serviços, que utilizam as especificações supracitadas. Os *containers* atuais utilizam as definições a partir de R4.

Ainda no que se refere a especificações, o OSGi tem a aprovação do JCP[20] (*Java Community Process*) sob as especificações JSR[21] 291 - *Dynamic Component*

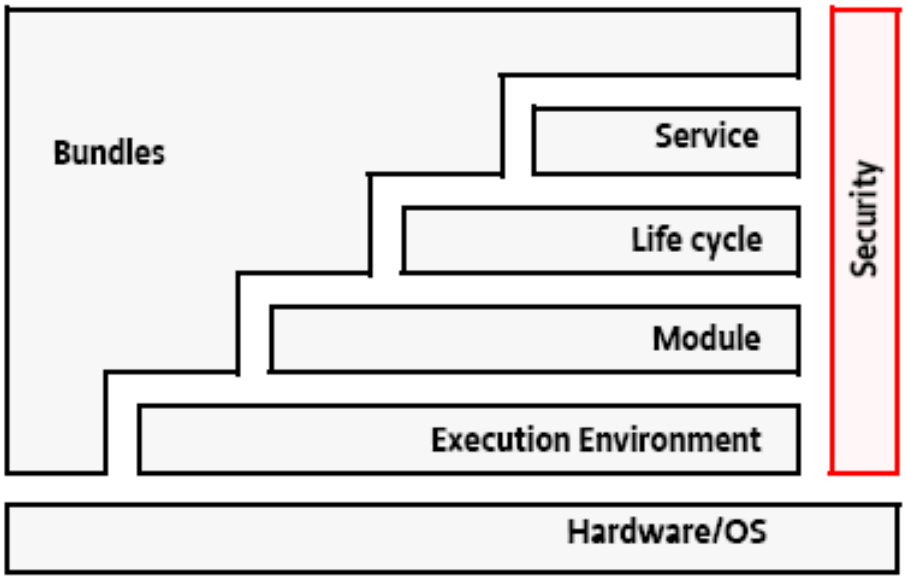
[19] De acordo com Cogoluègnes, Templier e Piper (2011), o *container* (contêiner) é uma implementação da especificação OSGi que será útil para gerenciar seus componentes e serviços. Muitos *containers* abertos estão disponíveis. Nesta dissertação são utilizados alguns.

[20] *Java Community Process* (JCP) é uma comunidade de desenvolvimento de especificações da linguagem *Java* (JCP, 2014a; MANN, 2005).

Support for Java (Suporte de Componente Dinâmico para *Java*) e JSR 232 - *Mobile Operational Management* (Gerenciamento Operacional Móvel). O JSR 291 define os componentes dinâmicos incluindo seus ciclos de vida para plataformas existentes da linguagem *Java* e permite que estes componentes sejam declarados através de arquivos metadados (dados sobre dados) e manipulados de forma dinâmica (JCP, 2006b; OSGI, 2015c). O JSR 232 define um *framework* de gerenciamento de componente para dispositivos móveis (JCP, 2012; OSGI, 2015b).

O *framework* OSGi possui uma arquitetura que se baseia no modelo de camadas composto por componentes com responsabilidades bem determinadas. Segundo OSGI(2015a), o OSGi traz muitas vantagens com este modelo, tais como a redução da complexidade no desenvolvimento de aplicações, a facilidade na codificação, a possibilidade maior de reuso de código e a implantação gerenciável. A Figura 10 apresenta o modelo de camadas que representa a arquitetura do OSGi.

Figura 10 – Modelo de Camadas da Arquitetura do OSGi



Fonte: OSGi (2014).

O componente *Hardware*/OS, que está relacionado à plataforma de *hardware* e o sistema operacional que o OSGi será instalado, é externo e somente reflete a necessidade de utilizar uma estrutura física capaz de executar os serviços do *framework*.

[21] *Java Specification Request* (JSR) é um documento que padroniza as especificações técnicas da linguagem *Java* (JCP, 2014a; MANN, 2005).

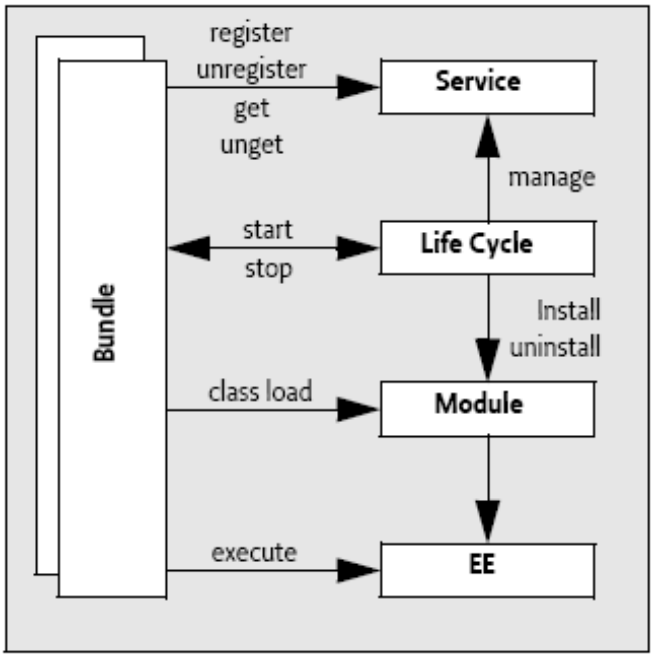
A seguir são detalhadas as camadas do ambiente OSGi:

a) a camada de Ambiente de Execução ou *Execution Environment* define quais métodos e classes estão disponíveis em determinada plataforma. Esta estrutura deve executar sobre uma *Java* VM e sobre um Sistema Operacional;
b) a camada de Módulos ou *Modules* define o conceito de *bundle*, que é um arquivo JAR com metadados. No *bundle* é possível declarar explicitamente que os pacotes contidos nele serão visíveis externamente (exportação de pacotes) e de qual pacote externo o *bundle* é dependente (importação de pacotes). Estes mecanismos são tratados também na camada de módulos (HALL et al., 2011);
c) a camada de Ciclo de Vida ou *Life Cycle* fornece uma API para controlar a segurança e as operações do ciclo de vida dos *bundles* (instalar, atualizar, iniciar, parar e desinstalar). Esta camada toma como base as camadas de módulo e segurança (HALL et al., 2011; OSGi, 2014, p.93);
d) a camada de Serviços ou *Services* define um modelo de colaboração dinâmica que é integrada com a camada de Ciclo de Vida. O modelo de serviço é um modelo *publish-find-bind* (publicar-localizar-conectar). Um serviço é um objeto normal *Java* que está registrado sob uma ou mais interfaces *Java* com o registro de serviço. *Bundles* podem registrar serviços, procurar por eles ou receber notificações quando seus estados de registro mudam (OSGi, 2014, p.127);
e) a camada de Segurança ou *Security* é opcional e está na base da estrutura OSGi. A camada é baseada na arquitetura de segurança da plataforma *Java* e fornece a infraestrutura para implantar e gerenciar aplicações que devem ser executadas em ambientes controlados de granularidade fina. Esta camada possibilita o controle de autenticidade de arquivos JARs através da verificação de uma assinatura digital e a autenticação do código (OSGI, 2014).

A partir das características de cada camada da arquitetura do OSGi, observa-se a existência de correlação entre suas funções independentes.

Figura 11 representa a interação entre estas camadas.

Figura 11 – Interação entre as camadas da arquitetura do OSGi



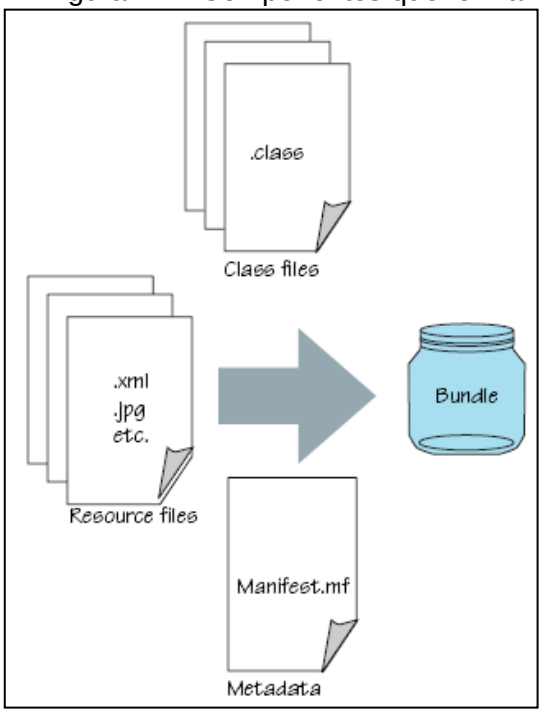
Fonte: OSGi (2014).

Alguns detalhes desta figura foram vistos na descrição das camadas. O *bundle*, por ser o componente dinâmico, realiza as ações diretas com as principais camadas da arquitetura OSGi. A camada de serviços provê a estrutura para a inserção ou remoção do registro de serviços disponíveis no *bundle*. Uma vez disponibilizado o serviço, o *bundle* pode iniciar ou finalizar este serviço, que são as operações controladas pela camada de ciclo de vida. A camada de módulos realiza o processo de importação e exportação de pacotes. E a camada de ambiente de execução provê recursos para a execução do *bundle*, no que diz respeito a plataforma disponível.

3.3.1 Os *Bundles*

Os *Bundles* são os componentes dinâmicos desenvolvidos para gerenciar os serviços e são definidos na camada de Módulo, como foi visto na seção anterior. Um *bundle* contém os arquivos de classe e seus recursos relacionados, além de um manifesto, contendo metadados, dentro de um arquivo JAR. A Figura 12 apresenta os componentes que formam um *bundle*.

Figura 12 – Componentes que formam um *bundle*



Fonte: Hall (2011).

O arquivo *MANIFEST.MF* atua como um descritor de implantação de *bundles*. O formato deste arquivo é o mesmo de um arquivo JAR normal, de modo que é constituído por um conjunto de cabeçalhos específicos com informações relevantes para a descrição do *bundle* sob diversos aspectos. A especificação OSGi define este conjunto para uso no seu *container*. A Figura 13 ilustra o referido arquivo com o exemplo de um *plug-in* [22] desenvolvido em um *container* OSGi conhecido como *Equinox*, do *Eclipse Foundation* (ECLIPSE, 2015a) – que será visto na seção 4.2 – que dá suporte à construção de arquivos do tipo XSD (*XML Schema Definition Language* – Linguagem de Definição de Esquema XML) (W3C WORKING GROUPS, 2012).

[22] *Plug-in* é uma aplicação ou ferramenta que se acopla a uma aplicação ou programa principal com a finalidade de adicionar recursos a ele e melhorar a sua funcionalidade.

Figura 13 – Típico arquivo *MANIFEST.MF* do OSGi

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Toast Emergency
Bundle-SymbolicName: org.equinoxosgi.toast.client.emergency
Bundle-Version: 1.0.0.qualifier
Bundle-RequiredExecutionEnvironment: J2SE-1.4
Bundle-Activator: org.equinoxosgi.toast.client.emergency.Activator
Import-Package: org.equinoxosgi.toast.dev.airbag, org.equinoxosgi.toast.dev.gps,
org.osgi.framework;version=”1.5.0”
```

Fonte: Adaptado de Mcaffer, Vanderlei e Archer (2010).

Atualmente o *MANIFEST.MF* possui 28 cabeçalhos (OSGI, 2014), que estão disponíveis para consulta na especificação do OSGi. Neste trabalho de dissertação são apresentados os mais relevantes. A seguir estão alguns dos cabeçalhos disponíveis para serem utilizados nos *bundles* e sua semântica:

a) o cabeçalho *Manifest-Version*, que informa a versão do arquivo *MANIFEST.MF*;
b) o cabeçalho *Bundle-ManifestVersion* que define que o *bundle* segue as regras de determinada especificação. Se no cabeçalho contiver o valor 2 indica o uso do OSGi R4 ou superior;
c) o cabeçalho *Bundle-Name*, que define um nome legível para o *bundle*;
d) o cabeçalho *Bundle-SymbolicName*, que é o identificador único do *bundle*. O nome simbólico do *bundle* deve ser baseado na convenção de nome de domínio de forma reversa;
e) o cabeçalho *Bundle-Version* que indica a versão do *bundle*;
f) o cabeçalho *Bundle-RequiredExecutionEnvironment* que contém uma lista de ambientes de execução (separada por vírgulas) que devem estar presentes no *framework* OSGi. Embora apareça no exemplo da Figura 13, este cabeçalho está obsoleto;
g) o cabeçalho *Bundle-Activator*, que indica o nome da classe utilizada para iniciar e parar o *bundle*. As classes especificadas nesta propriedade devem instalar ou importar a interface *org.osgi.framework.BundleActivator*;
h) o cabeçalho *Import-Package*, que declara os pacotes importados pelo *bundle*.

A seguir, é apresentada uma lista com outros cabeçalhos que também podem ser encontrados no arquivo *MANIFEST.MF*:

a) o cabeçalho *Bundle-ClassPath* que define uma lista (separada por vírgulas) de nomes de caminho de arquivos JAR ou diretórios (dentro do *bundle*) que contenha classes e recursos. O caractere ponto "." especifica o diretório raiz do JAR do *bundle*;
b) o cabeçalho *DynamicImport-Package*, que contém uma lista de nomes de pacotes, separada por vírgulas, que devem ser dinamicamente importados quando necessários;
c) o cabeçalho *Export-Package*, que contém uma declaração de pacotes exportados;
d) o cabeçalho *Bundle-Vendor*, que contém uma descrição legível do fornecedor do pacote;
e) o cabeçalho *Bundle-Localization*, que contém a localização do pacote e onde os seus arquivos podem ser encontrados.

Uma característica que vale ressaltar diz respeito à interface *org.osgi.framework.BundleActivator*. Esta interface é responsável por iniciar e interromper a ação do *bundle* e esta informação deve constar no cabeçalho do arquivo *MANIFEST.MF*, como visto nos detalhes supracitados. O código fonte da classe *Activator.java*, que implementa a interface *BundleActivator* está como segue na Figura 14.

Figura 14 – Código fonte típico da classe *Activator.java*

```
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class Activator implements BundleActivator {

	public void start(BundleContext context) throws Exception {

	}

	public void stop(BundleContext context) throws Exception {

	}

}
```

Fonte: Elaborada pela autora.

A interface *BundleActivator* tem um construtor público sem parâmetros. O *framework* OSGi pode criar um *BundleActivator* chamando o método *Class.newInstance()*. A interface *BundleActivator* possui dois métodos para inicializar e interromper um *bundle*:

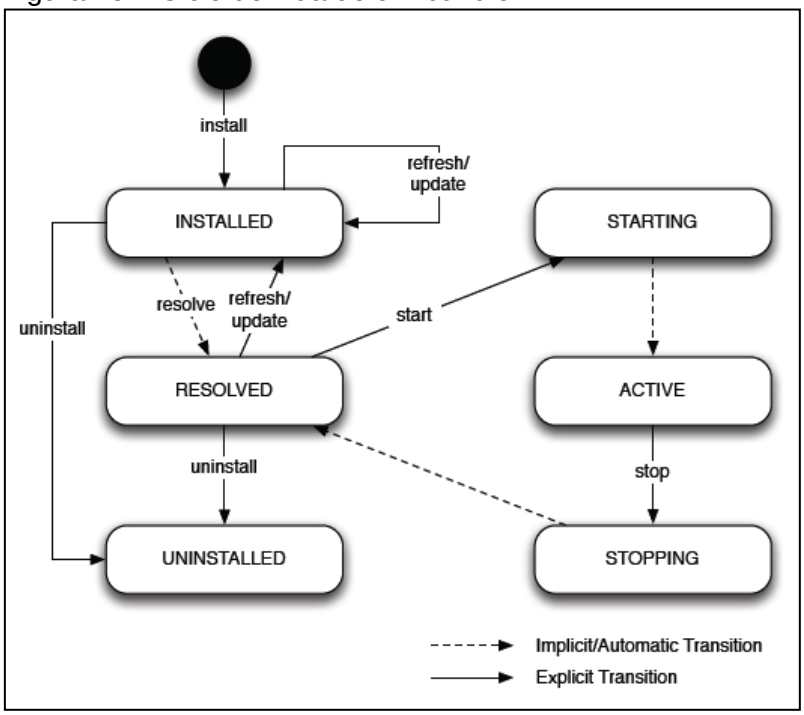
a) o método *start(BundleContext)*, que aloca recursos que um *bundle* necessita, inicia *threads*[23], registra serviços, dentre outras funções. O método *start()* tem um argumento, o objeto *BundleContext.* Este permite que os *bundles* interajam com o *framework* providenciando o acesso a informação do *container* OSGi relacionado. O estado do *bundle* passa para *ACTIVE* (este estado será visto na Seção 3.3.1.1 sobre o ciclo de vida dos *bundles*);
b) o método *stop(BundleContext)*, que encerra o *bundle* em questão, desfazendo a funcionalidade do método *start()*. Este método é chamado apenas quando o *bundle* está no estado *ACTIVE*.

3.3.1.1 Ciclo de Vida dos Bundles

O ciclo de vida de um *bundle* passa por vários estados, desde quando é feita a sua instalação no *framework* até quando ele finaliza a sua atividade ou função. Estes estados são controlados através de mecanismos utilizados pela camada de ciclo de vida da estrutura OSGi. A Figura 15 apresenta os estágios que os *bundles* podem passar.

[23] *Threads* são tarefas de um processo de uma aplicação ou sistema, que ocorrem concorrentemente (DEITEL, 2005).

Figura 15 – Ciclo de vida de um *bundle*



Fonte: Bartlett (2009).

De acordo com a Figura 15, é possível identificar quais são os estados do ciclo de vida, que são como seguem abaixo com suas respectivas ações (ALVES, 2012; BARTLETT, 2009; CUMMINS; WARD, 2013; GÉDÉON, 2010; HALL et al., 2011; MCAFFER; LEMIEUX; ANISZCZYK, 2010):

a) *installed* é o estado inicial que é atribuído ao *bundle* quando ele é instalado através do comando *install*, representado na figura pelo círculo escuro, que é o ponto de entrada do diagrama;
b) *resolved* é o estado que indica que o *bundle* está pronto para ser ativado ou pronto para ser desinstalado;
c) *active* é o estado que indica que o *bundle* está ativo. Se o comando *start* for executado para o *bundle* em questão, ele passa a assumir o estado de *ACTIVE*, embora o diagrama não mostre o laço de direção entre estes

dois estados. No estado *ACTIVE*, o *bundle* não está necessariamente ativo, executando qualquer código. A funcionalidade do *bundle*, que acontece durante o estado *ACTIVE* de um *bundle*, depende dos recursos liberados pelo método *start()* da interface *BundleActivator*;

d) *starting* é o estado que indica um processo de ativação. Os *bundles* só podem ir para este estado quando estão no estado de *RESOLVED*. Assim, para o *bundle* passar para o estado de *ACTIVE*, o *framework* procede com o estágio *RESOLVED* primeiro;
e) *stopping* é o estado que encerra as ações do *bundle*. Quando o comando *stop* é executado, o *bundle* passa pelo estado *STOPPING* enquanto o OSGi chama o método *stop()* da interface *BundleActivator*. Neste momento, o *bundle* encerra os recursos acionados na fase *STARTING*. O estado *STOPPING* também é transitório para que o *bundle* retorne ao estado *RESOLVED*.
f) *uninstalled* é o estado final do *bundle* e ocorre quando a desinstalação do *bundle* é feita. Embora este estado esteja na figura, ele não é visto na prática, porque não é possível transitar deste estado para qualquer outro, mesmo que o *bundle* seja reinstalado, usando o mesmo arquivo JAR. O *framework* o considerará um *bundle* diferente e o atribuirá a outro ID (identificação).

3.3.2 Arquitetura Orientada a Serviços e o *framework* OSGi

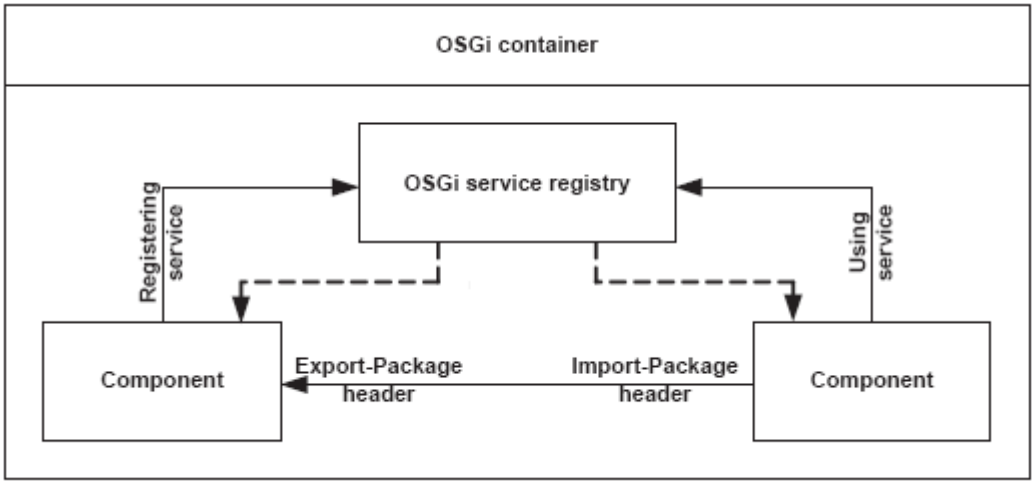
SOA é uma arquitetura que faz a integração de várias aplicações distribuídas para acessar os serviços. Cada aplicação possui suas próprias exportações como uma unidade de serviço. A lógica interna da constituição do serviço é encapsulada na aplicação. Esta expõe apenas interfaces de serviços sob a forma de métodos exportados (BREIER; PEREIRA; 2011). Quando o pacote é exportado por algum componente, ele se torna visível para quem vai consumir o pacote (COGOLUÈGNES; TEMPLIER; PIPER, 2011).

Para utilizar os pacotes exportados, a forma mais adequada é especificá-los no cabeçalho *Import-Package* no arquivo *MANIFEST.MF*, facilitando o uso das classes destes determinados pacotes. O componente que provê um serviço também

deve certificar-se de que os pacotes necessários pelo serviço estão disponíveis (COGOLUÈGNES; TEMPLIER; PIPER, 2011).

A Figura 16 demonstra o processo de exportação e importação de pacotes e a relação com o registro de serviços do OSGi, que é mencionado a seguir.

Figura 16 - Exportação e Importação de Pacotes

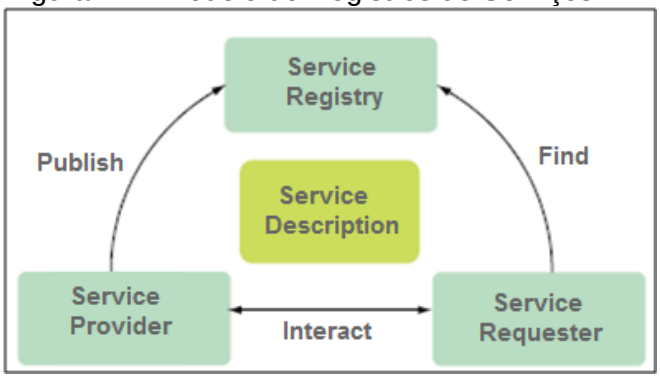


Fonte: Adaptado de Cogoluègnes, Templier e Piper (2011).

Como foi visto no início da seção 3.3, a especificação do núcleo do OSGi define seu *framework*, abordando aspectos de modularidade, ciclo de vida, serviços e segurança. Juntos, estes permitem que os *bundles* sejam reutilizáveis, encapsulados e fracamente acoplados (ALVES, 2012).

O *framework* OSGi incorpora os conceitos de SOA, observando as definições da camada de serviços, que utiliza abordagens de interação no padrão da computação orientada a serviços, através do modelo *publish-find-bind*. Os provedores de serviços (*Service Providers*) publicam seus serviços em um registro de serviços (*Service Registry*); os clientes de serviços (*Service Requesters*) buscam o registro para localizar os serviços disponíveis para uso. Esta relação pode ser vista na Figura 17 (HALL et al, 2011).

Figura 17 – Modelo de Registros de Serviços



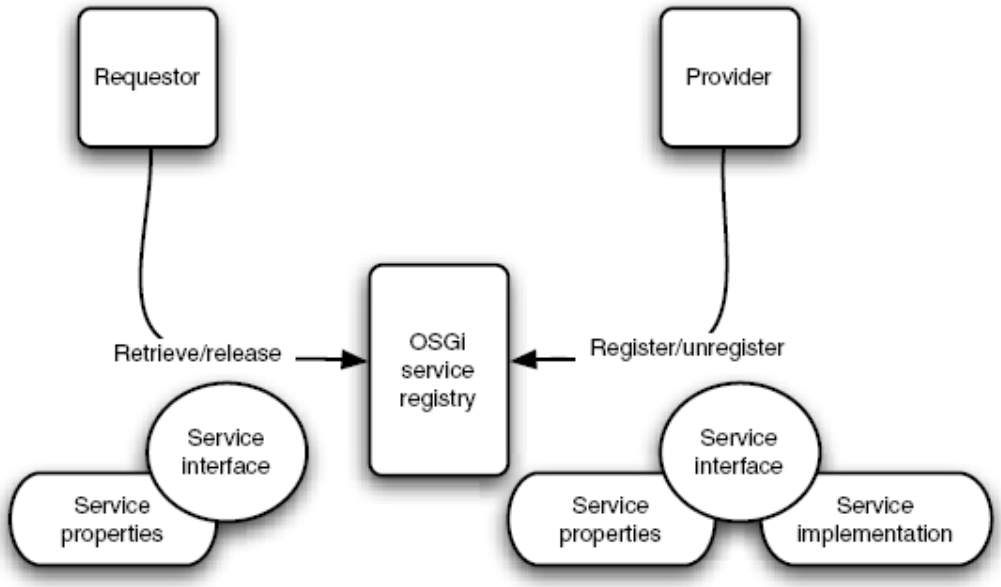
Fonte: Adaptado de Hall e outros (2011).

De acordo com Alves (2012) o modelo de serviço OSGi permite que os serviços sejam alterados dinamicamente sem a necessidade de interromper o *container*. Os serviços são gerenciados pelo registro de serviço OSGi, que possui as seguintes funções (ALVES, 2012):

a) permitir que os serviços efetuem ou cancelem os registros pelo *bundle* provedor de serviços;
b) permitir que os serviços sejam recuperados e liberados pelo *bundle* solicitante do serviço.

Nestas funções, o registro de serviço OSGi atua como mediador, isolando o provedor do solicitante. O *bundle* fornecedor de serviços detém a interface do serviço, as suas propriedades e implementação. O pacote solicitante do serviço utiliza a interface de serviço e suas propriedades, mas não podem acessar a sua implementação. A Figura 18 demonstra estas funções (ALVES, 2012).

Figura 18 - Funções do Registro de Serviços

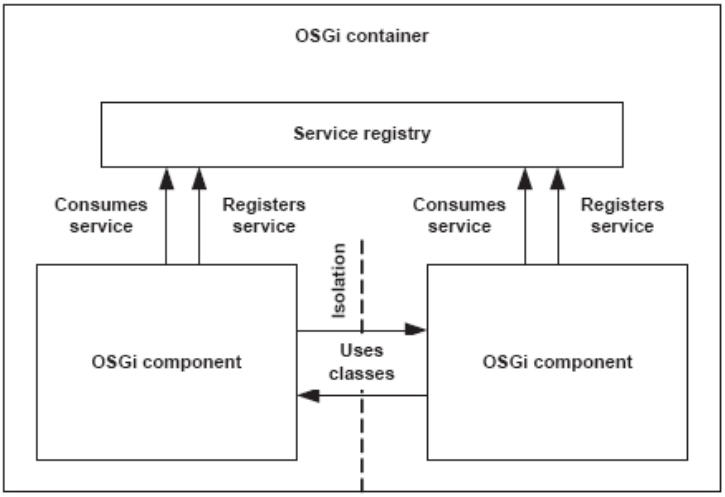


Fonte: Alves (2012).

Hall e outros (2011) também afirmam que a camada de serviços promove a separação entre a interface e a implementação. Os serviços OSGi são interfaces em *Java* que representam um contrato conceitual entre provedores de serviços e clientes. Os provedores são objetos em *Java* acessados através de invocação de métodos diretos.

A Figura 19 apresenta as atribuições dos componentes, que podem tanto consumir serviços como registrar serviços no *Service Registry* dentro do *container* OSGi. O uso de classes entre os componentes acontece mediante a exportação e importação de pacotes, que são declarados no *MANIFEST.MF*.

Figura 19 – Algumas funções dos componentes OSGi



Fonte: Cogoluègnes, Templier e Piper (2011).

Para registrar um serviço no registro de serviços do OSGi, a classe *BundleContext* fornecida pela interface *BundleActivator* é utilizada (ALVES, 2012). Esta classe é responsável pela implantação e gerenciamento do ciclo de vida dos *bundles*, devido à presença de métodos que escutam os seus estados, e pela interação dos *bundles* através dos serviços (HALL et. al, 2011).

3.3.3 Modularidade

A modularidade abrange muitos aspectos da programação. Ela é definida pela junção de pequenas partes (módulos) de um sistema, que são logicamente independentes. Um módulo define um limite lógico executável. O código ora é parte do módulo, ora não é. Os detalhes internos de um módulo são visíveis apenas para o código que faz parte deste módulo. Para todos os outros códigos, os detalhes visíveis de um módulo são aqueles que estão expostos explicitamente (uma API pública). Este aspecto de módulos torna-os parte integrante da concepção da estrutura lógica de uma aplicação (HALL et al., 2011).

Os arquivos JAR são uma forma de obter a modularidade. Entretanto, mesmo já tendo sua existência desde a década de 90, estes arquivos são difíceis de versionar (devido à necessidade de implantar versões e usar recursos de dependências entre arquivos JAR de forma confiável), distribuir (devido à necessidade de utilizar mecanismos para empacotar muitos arquivos JAR) e referenciar (JCP, 2006a).

Os requisitos JSR 294 *Improved Modularity Support in the Java Programming Language* (Suporte a Modularidade Melhorada na Linguagem de Programação *Java*) e JSR 277 *Java Module System* (Sistema Modular *Java*) especificam a modularidade em *Java*. A especificação JSR 294 tem o objetivo de estender recursos na linguagem *Java* de forma a permitir uma organização hierárquica dos módulos através do conceito de superpacote, para evitar que desenvolvedores declarem elementos que são necessários a algumas partes do programa como públicos, tornando-os acessíveis globalmente (JCP, 2007a).

A especificação JSR 277 tem como objetivo resolver as questões relacionadas à (JCP, 2006a):

a) distribuição, em que seu formato e o formato de metadados são como unidades de entrega, onde os metadados possuirão informações, tais como dependências entre módulos;
b) um esquema de controle de versão para os módulos e dependências;
c) repositórios para armazenagem e recuperação de módulos.

Segundo Bartlett (2009), a especificação JSR 277 se manteve a mesma, ou seja, possui nada implementado. Esta especificação não dá suporte a dependências de módulos e não é dinâmico, pois depende da máquina virtual *Java* para instalar, atualizar e desinstalar os módulos. Em comparação ao OSGi, este possui as ferramentas desejáveis para efetuar a modularidade nas aplicações. Para aplicações que tenham a modularidade, como requisito, e também a reusabilidade, adaptabilidade e integração de tecnologias, utilizar esta solução pode ser adequada.

Após a discussão sobre as soluções para a aplicação, segue uma breve comparação entre estas, levando-se em consideração as características da aplicação de redes residenciais.

3.4 AVALIAÇÃO DAS CARACTERÍSTICAS DOS *MIDDLEWARES* PARA O DESENVOLVIMENTO DE APLICAÇÕES EM REDES RESIDENCIAIS

A avaliação dos *middlewares* foi feita tendo como base os requisitos para a aplicação desenvolvida no capítulo 6 para sistemas de redes residenciais. Características como modularidade, reusabilidade, integração de tecnologias e adaptabilidade foram levadas em consideração.

O Quadro 3, como segue adiante, apresenta as soluções analisadas e os requisitos das aplicações de redes residenciais, onde é possível identificar qual das soluções provê mais recursos.

Quadro 3 – Comparação entre as alternativas de *middlewares*

	Modularidade	Adaptabilidade	Reusabilidade	Integração de Tecnologias
JSF e CDI	Sim	Sim	Sim	Não
Jigsaw	Sim	Não	Não	Não
OSGi	Sim	Sim	Sim	Sim

Fonte: Elaborada pela autora.

A primeira alternativa de solução a ser avaliada é a utilização do JSF e do CDI. A utilização de ambas as soluções pode ser uma alternativa viável, por possuírem recursos para criação de aplicações modulares. O JSF apresenta as ferramentas para implementar aplicações *Web*, o que é um ponto a considerar para uso nas aplicações de redes residenciais. Nestas, os módulos são dependentes, apesar de estarem isolados. Assim, é necessário instalar recursos de gerenciamento de dependências e controlar o ciclo de vida dos módulos. Estes recursos estarão disponíveis com a utilização do CDI associado ao *framework* JSF. Considerando as aplicações de redes residenciais, a junção das duas soluções de *Java* pode atender alguns dos seus requisitos. Com a utilização do JSF e do CDI não é possível identificar características que permitam a integração de tecnologias, que são úteis nas aplicações de redes residenciais. Assim, esta alternativa não será utilizada.

A solução *Jigsaw* foi escolhida como alternativa pela proposta de prover modularidade nas aplicações baseadas no *Java*. É uma solução mais antiga que o OSGi e pode oferecer recursos para aplicações em redes residenciais. É possível que o *Jigsaw* seja incorporado à linguagem *Java* nas próximas versões, com recursos que promovam a interoperação com o OSGi. O *Jigsaw* ainda não oferece recursos para atender aos requisitos de adaptabilidade e integração de tecnologias.

Por fim, avalia-se o OSGi como alternativa de *middleware* para a aplicação de redes residenciais, sendo que o *framework* possui um aspecto proeminente que está relacionado à modularidade e ao não compartilhamento de arquivos JAR clássicos. Na linguagem *Java* os pacotes, por padrão, são visíveis em toda a aplicação, e isto não acontece com os pacotes do OSGi. Para que estes se tornem visíveis, é necessário utilizar o recurso de exportação de pacotes.

No âmbito das redes residenciais, na ideia de integrar os serviços que são fornecidos pelos aparelhos desta rede, é necessário adotarmos um sistema que permita tal integração. Os aparelhos domésticos são na prática de fabricantes

diferentes e a proposta é aplicar um sistema que utilize padrões abertos e que tenha como uma das características a modularidade no desenvolvimento da aplicação. A alternativa ideal de solução é utilizar esta abordagem OSGi.

A utilização desta solução levou-se a três alternativas de arcabouços para construção dos serviços e aplicações para redes residenciais. Isto posto, o próximo capítulo segue com uma discussão sobre os arcabouços e suas características.

4 ARCABOUÇOS DE DESENVOLVIMENTO DE SERVIÇOS E APLICAÇÕES EM REDES RESIDENCIAIS

Neste capítulo serão apresentados os principais arcabouços que possibilitam o desenvolvimento de serviços e aplicações em redes residenciais. Estes arcabouços são estruturas que facilitam a criação de pacotes de serviços. Alguns critérios são apresentados como forma de determinar a estrutura mais adequada para a aplicação de redes residenciais. Após a estrutura escolhida, esta será utilizada na construção do serviço no protótipo previsto neste projeto de dissertação.

Na abordagem *Java* OSGi foram desenvolvidas algumas alternativas de arcabouços, que são entidades de código aberto e utilizam as especificações do *framework*. Estas entidades possibilitam a criação de *bundles* de serviços de redes residenciais. Dentre as alternativas de arcabouços abertos, é possível citar:

a) o *Eclipse Equinox* (ECLIPSE, 2015a);
b) o *Apache Felix* (APACHE, 2015c);
c) o *Makewave Knopflerfish* (KNOPFLERFISH, 2015a).

Existem alguns outros arcabouços abertos, como o *Concierge* (ECLIPSE, 2015c), porém foram selecionados os três mais referenciados na literatura para aplicarmos no escopo da proposta da dissertação, qual seja, redes residenciais.

As três estruturas possuem características peculiares que podem determinar o método mais adequado para desenvolver a aplicação numa rede residencial. Para a aplicação do protótipo desenvolvida no capítulo 6, alguns requisitos serão levados em consideração para efeitos de comparação entre os arcabouços abertos disponíveis, dentre os quais é possível citar:

a) o conjunto de serviços disponíveis, bem como a disponibilidade de APIs e suas características;
b) a estrutura interna do arcabouço que possibilita o gerenciamento dos pacotes de serviços;
c) a curva de aprendizagem e recursos disponíveis para o uso do arcabouço, onde será utilizado um projeto de teste para avaliação.

As seções a seguir apresentam uma breve discussão sobre o projeto de teste que será implementado e os principais arcabouços da abordagem OSGi, com suas características e algumas considerações importantes na definição do arcabouço adequado para a criação da aplicação de serviços de redes residenciais.

4.1 *BUNDLE* DE TESTE

Esta seção apresenta o projeto de teste que será implementado nos arcabouços, na avaliação do critério da curva de aprendizagem. O projeto de teste é um projeto comum a todos os arcabouços, contendo os principais arquivos de um *bundle*, que são o *MANIFEST.MF* e o *Activator.java*.

O exemplo de teste utilizará a principal interface da API do OSGi (*BundleActivator*), onde deverá aparecer no console do *container* os textos “Iniciar *Bundle*” (através do método *start()*) e “Finalizar *Bundle*” (através do método *stop()*). De maneira geral, para construir o pacote de teste com estas informações, a classe *Activator.java* em todos os arcabouços fica como segue na Figura 20.

Figura 20 – Classe *Activator.java* utilizada nos arcabouços

```
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class Activator implements BundleActivator {

	public void start(BundleContext context) throws Exception {
		System.out.println("Iniciar Bundle");

	}

	public void stop(BundleContext context) throws Exception {
		System.out.println("Finalizar Bundle");

	}
}
```

Fonte: Elaborada pela autora.

Além da classe *Activator.java*, o arquivo que consta os cabeçalhos que descrevem o *bundle* de serviço (na maioria dos casos, chamado de *MANIFEST.MF*) também deve ser criado, é comum em todos os arcabouços e fica de acordo com a Figura 21.

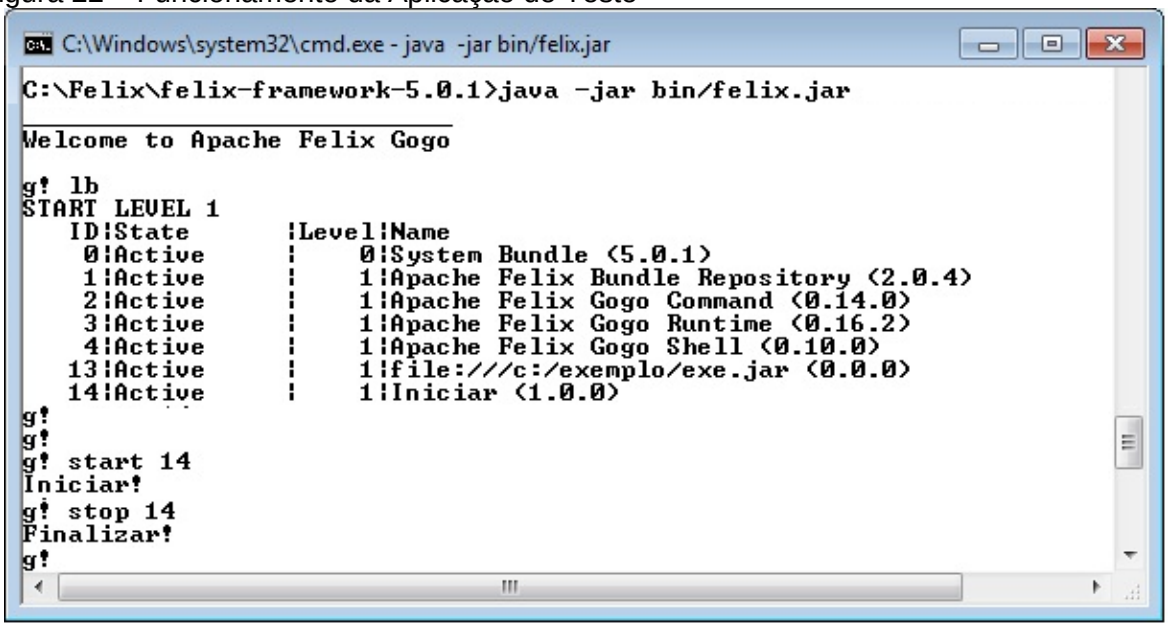
Figura 21 – Arquivo *MANIFEST.MF* utilizado nos arcabouços

```
Manifest-Version: 1.0
Bundle-SymbolicName: exemplo.projeto.osgi.bundle
Bundle-Name: Iniciar
Bundle-Version: 1.0.0
Bundle-ManifestVersion: 2
Bundle-Activator: exemplo.projeto.osgi.Activator
Bundle-Vendor: Projeto
Import-Package: org.osgi.framework;version=”1.3.0”
```

Fonte: Elaborada pela autora.

O *Bundle* de Teste tem como resultado a visualização da informação no console do *framework*. A classe *Activator* possui as informações que serão vistas no *console*. Se o comando start for acionado, o *bundle* executa o método *start()* do *Activator*. Se o comando *stop* for acionado, o *bundle* executa o método *stop()* do *Activator*. A Figura 22 representa o funcionamento da aplicação de teste.

Figura 22 – Funcionamento da Aplicação de Teste



Fonte: Elaborada pela autora.

De acordo com a figura, considera-se que a identificação do bundle d teste é igual a 14.

4.2 IMPLEMENTAÇÃO OSGI – ARCABOUÇO *ECLIPSE EQUINOX*

O *Equinox* é a base de execução de ferramentas do *Eclipse*, aplicações cliente, servidor e projetos embarcados. Ele utiliza como referência as especificações de *Core* do *framework*, as especificações de serviço e a especificação JSR 291 (JCP, 2006b). O arcabouço *Eclipse Equinox* utiliza a versão R4 das especificações do OSGi, onde instala um conjunto de *bundles* com vários serviços adicionais e com infraestrutura para executar sistemas baseados na plataforma OSGi.

O *Equinox* já está integrado ao *Integrated Development Environment*[24] (IDE – Ambiente de Desenvolvimento Integrado) *Eclipse*, o que facilita o desenvolvimento de *bundles* (OSGI, 2015a).

As seções 4.2.1, 4.2.2 e 4.2.3 a seguir apresentam, respectivamente, características como o conjunto de serviços disponíveis, a estrutura interna do arcabouço e a curva de aprendizagem dos seus serviços.

4.2.1 Conjunto de Serviços Disponíveis

Nesta seção são elencados os serviços que o *Eclipse Equinox* disponibiliza ao utilizarmos sua estrutura.

A equipe de desenvolvedores que mantém o *Equinox* é dividida em vários times, que são grupos que atuam em serviços distintos providos pelo *Equinox* para desenvolver as soluções e cada time, por sua vez, lida com vários projetos. Estas áreas são secionadas como seguem no Quadro 4 (ECLIPSE, 2015b):

[24] O Ambiente de Desenvolvimento Integrado é um programa que contém ferramentas e recursos que auxiliam os desenvolvedores na implementação e desenvolvimento de *software* e aplicações. Neste ambiente é possível criar códigos, editá-los, compilá-los, depurá-los, dentre outras funções.

Quadro 4 – Grupos de Trabalho do *Eclipse Equinox*

Grupo	Descrição
Framework	Este grupo lida com a execução da especificação R4 do OSGi e produz mecanismos de execução do *Eclipse Equinox*, além de infraestrutura de inicialização e modelos de aplicação para facilitar o uso do *Equinox*.
Bundles	Este grupo pratica serviços *add-on* nas especificações OSGi e define pacotes e serviços para utilidades nos sistemas OSGi.
P2	Este grupo lida com recursos que permitem aos desenvolvedores a definição, estrutura e implantação de sistemas de forma mais flexível.
Incubator	Este grupo utiliza da prática de técnicas de configurações da plataforma de execução do *Eclipse*.
Security	Este grupo lida com a garantia de segurança da execução do *Equinox* e na implantação de aplicações desenvolvidas neste ambiente.
Server	Este grupo efetiva a integração e o uso do *Equinox* em servidores.
Weaving	Este grupo busca relacionar o uso do *Equinox* com o *AspectJ* (linguagem de programação orientada a aspectos).

Fonte: Elaborada pela autora.

O grupo *Bundles* disponibiliza as informações acerca dos serviços e APIs que o *Equinox* pode prover para auxiliar na criação de projetos OSGi. Dentre os serviços disponíveis, seguem os mais comuns, conforme o Quadro 5 (ECLIPSE, 2015g):

Quadro 5 – Serviços disponíveis pelo arcabouço *Equinox*

Serviços	Descrição
Log	O serviço de *Log* é um registro de mensagens provido pelo OSGi;
Serviços de HTTP	Os serviços de HTTP fornecem suporte a páginas HTML (*HyperText Markup Language* – Linguagem de Marcação de Hipertexto) e *servlets*.
Device Access	O serviço de *Device Access* especifica a permissão de acesso a dispositivos.
Serviços de *Configuration Admin*	Os serviços de *Config Admin* especificam a configuração de informações de *bundles*.
Meta Type	O serviço de *Meta Type* especifica a permissão do uso de metadados.
Serviços de *Preferences*	Serviços de *Preferences* armazenam informações referentes às preferências de usuários.
Serviço *User Admin*	O serviço *User Admin* é útil na autenticação e na autorização do usuário ao utilizar algum serviço.
Serviços declarativos	Serviços declarativos simplificam tarefas de autoria de serviços OSGi, registrando serviços e manipulando suas dependências.
Serviços de administração de eventos	Os serviços de administração de eventos proveem comunicação entre *bundles* através de eventos.

Fonte: Elaborada pela autora.

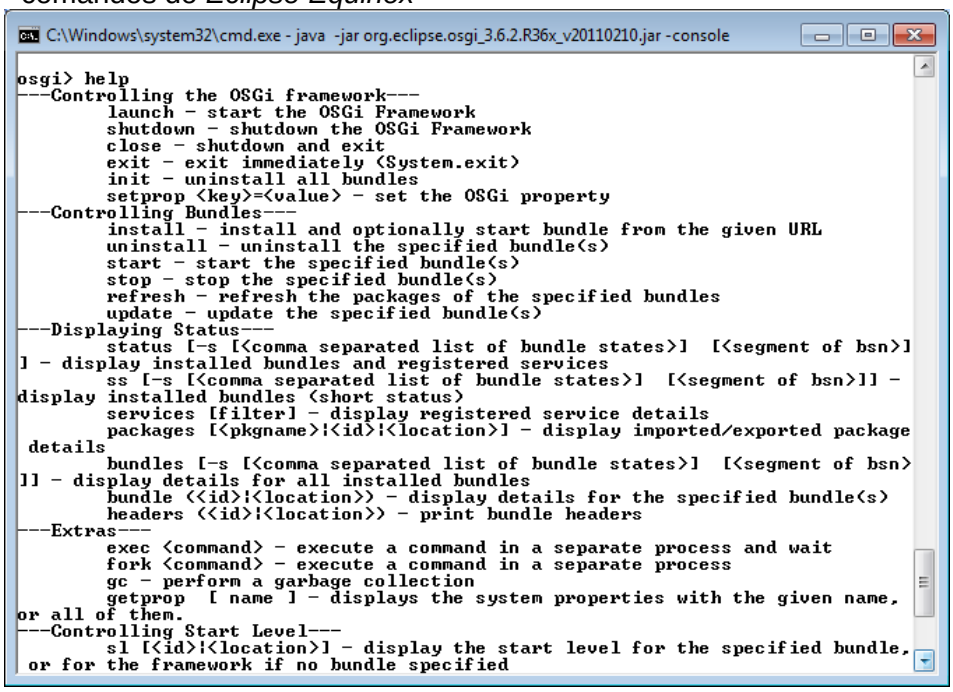
Mesmo com a especificação mais antiga entre os arcabouços, observa-se que o *Equinox* pode prover muitos recursos que possibilitam a construção de serviços para aplicações de redes residenciais.

4.2.2 Estrutura Interna do Arcabouço

Nesta seção será apresentada a estrutura interna do *Eclipse Equinox*, que é representada pelo interpretador de comandos. É possível observar os comandos disponíveis para gerenciar os *bundles* que são desenvolvidos no seu ambiente de programação.

O interpretador de comandos do *Equinox* é acessível após o seu processo de instalação. Este processo de instalação pode ser visto no Apêndice A. Com o processo de instalação concluído, os comandos disponíveis para acesso são como seguem na Figura 23 – Lista de Comandos do *Framework* OSGi no interpretador de comandos do *Eclipse Equinox*Figura 23.

Figura 23 – Lista de Comandos do *Framework* OSGi no interpretador de comandos do *Eclipse Equinox*


C:\Windows\system32\cmd.exe - java -jar org.eclipse.osgi_3.6.2.R36x_v20110210.jar -console
osgi> help
---Controlling the OSGi framework---
launch - start the OSGi Framework
shutdown - shutdown the OSGi Framework
close - shutdown and exit
exit - exit immediately (System.exit)
init - uninstall all bundles
setprop <key>=<value> - set the OSGi property
---Controlling Bundles---
install - install and optionally start bundle from the given URL
uninstall - uninstall the specified bundle(s)
start - start the specified bundle(s)
stop - stop the specified bundle(s)
refresh - refresh the packages of the specified bundles
update - update the specified bundle(s)
---Displaying Status---
status [-s [<comma separated list of bundle states>] [<segment of bsn>]
] - display installed bundles and registered services
ss [-s [<comma separated list of bundle states>] [<segment of bsn>]] -
display installed bundles (short status)
services [filter] - display registered service details
packages [<pkgname>|<id>|<location>] - display imported/exported package
details
bundles [-s [<comma separated list of bundle states>] [<segment of bsn>
]] - display details for all installed bundles
bundle (<id>|<location>) - display details for the specified bundle(s)
headers (<id>|<location>) - print bundle headers
---Extras---
exec <command> - execute a command in a separate process and wait
fork <command> - execute a command in a separate process
gc - perform a garbage collection
getprop [name] - displays the system properties with the given name,
or all of them.
---Controlling Start Level---
sl [<id>|<location>] - display the start level for the specified bundle,
or for the framework if no bundle specified

Fonte: Elaborada pela autora.

Nesta figura, para visualizar os comandos disponíveis entra-se com o comando *help* ou *?* ou ainda qualquer informação que seja diferente dos comandos internos do console do arcabouço. Este comando tem o propósito equivalente ao comando *help* do *prompt* de comandos do *Windows*, para auxiliar na sintaxe e

função dos comandos. Os comandos são organizados em categorias, como Controlar o *Framework*, Controlar os *Bundles*, Exibir *Status*, dentre outros.

Os comandos da categoria de Gerenciamento do *Framework* OSGi, alguns comandos de Exibição de *Status* e os comandos Extras são específicos do *Equinox* e podem ser acionados em qualquer ponto do interpretador de comandos.

4.2.3 Curva de Aprendizagem

A avaliação da curva de aprendizagem do *Eclipse Equinox* é considerada em seguida, onde será utilizado o exemplo de teste, após a instalação do arcabouço.

Para criar o *bundle* de Teste foram efetivadas algumas etapas, que podem ser vistas no Apêndice A.

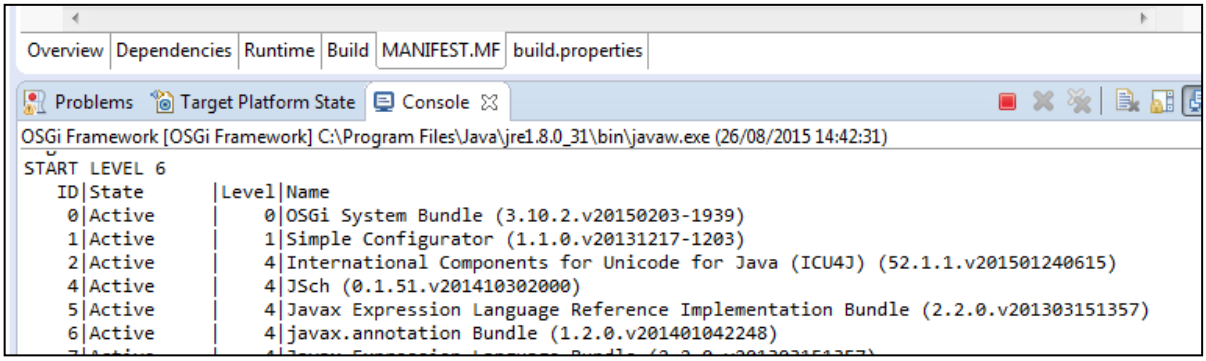
Existe a opção de criar *bundles* de serviços utilizando os *templates* (arquivos prontos) disponíveis no *Eclipse* PDE. Um *template* que está disponível e é equivalente ao *bundle* de teste é o *Hello World*, mas será feita a criação do *bundle* de teste sem o uso destes arquivos prontos. O *Eclipse* gera os arquivos *Activator.java* e o *MANIFEST.MF*, com os códigos já vistos na seção 4.1.

No *Equinox*, o arquivo *MANIFEST.MF*, que está localizado na pasta *META-INF*, possui, praticamente, os mesmos cabeçalhos. Uma diferença é que ele acrescenta ao arquivo o cabeçalho *Bundle-Localization* indicando que o *bundle* criado é considerado um *plugin*, para adicionar em aplicações OSGi.

Definidos os arquivos *Activator.java* e *MANIFEST.MF*, o *bundle* pode ser executado no *container* (maiores detalhes podem ser vistos no Apêndice A). O IDE exibe um *console* OSGi, de acordo com a

Figura 24. Neste console é possível manipular os *bundles*, como instalar, remover, iniciar, parar ou atualizar, por exemplo.

Figura 24 – Console do Ambiente de Desenvolvimento *Eclipse*



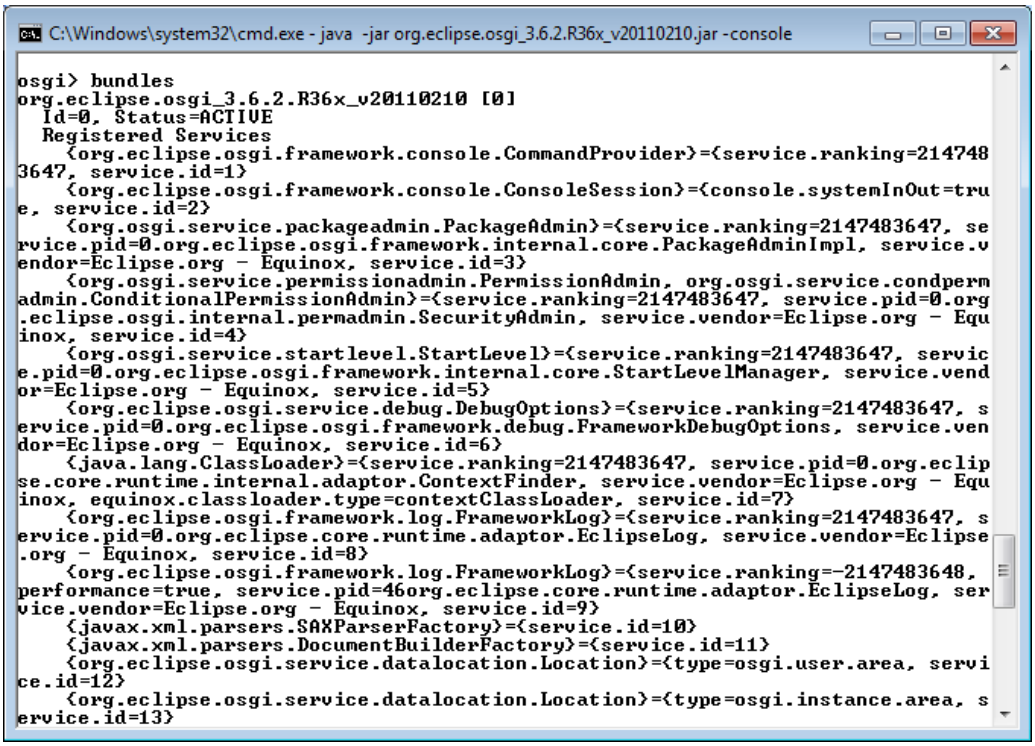
Fonte: Elaborado pela própria autora.

Também é possível manipular os *bundles* que foram programados utilizando o *console* do *framework* que foi descompactado no processo de instalação. Ele é o mesmo que está incluso no *container* e os mesmos comandos podem ser realizados. A diferença de utilizar o *console* que está fora do ambiente de programação, é que após a criação do *bundle*, o projeto deve ser compilado. Com o arquivo no formato JAR, ele será instalado no *framework* (maiores detalhes podem ser vistos no Apêndice A).

O estado do *bundle* depois que ele é instalado é *INSTALLED*, como foi visto na seção 3.3.1.1. Neste estado ele não disponibiliza qualquer serviço. Para iniciar o *bundle*, o comando *start* é utilizado (deve ser informado o ID do *bundle* a ser iniciado), seu estado é alterado para *ACTIVE* e neste momento ele passa a disponibilizar os serviços, caso ele possua. Para finalizar o *bundle*, entra-se com o comando *stop* acrescido do seu ID. O estado do *bundle* altera para *RESOLVED*, podendo retornar para *ACTIVE*, caso o comando *start* seja novamente digitado.

Para verificar o estado, a localização e os IDs dos *bundles*, o comando utilizado é *bundles*, e a saída fica de acordo com a Figura 25.

Figura 25 – Saída do comando *bundles* no arcabouço *Equinox*



Fonte: Elaborada pela autora.

No que diz respeito a esta característica, o *Equinox* disponibiliza um ambiente de criação de pacotes com recursos para aplicações em redes residenciais. Como o *bundle* de Teste tem apenas uma classe, a instalação bem como a inicialização do pacote, utilizando o console, atendeu os requisitos.

4.3 IMPLEMENTAÇÃO OSGI – ARCABOUÇO *APACHE FELIX*

O *Apache Felix* é um *container* de código aberto baseado no projeto *Oscar* de *ObjectWeb* (OW2 CONSORTIUM, 2005). Os desenvolvedores fizeram uma série de melhorias e, em 2007, o *Felix* saiu do nível de incubação e se tornou um dos mais importantes projetos da *Apache Software Foundation*. Atualmente executa as especificações de Serviços OSGi R6 sob licença da *Apache* (APACHE, 2015c).

As seções 4.3.1, 4.3.2 e 4.3.3 a seguir apresentam, respectivamente, características como o conjunto de serviços disponíveis, a estrutura interna do arcabouço e a curva de aprendizagem dos serviços.

4.3.1 Conjunto de Serviços Disponíveis

Nesta seção, os serviços que o arcabouço *Apache Felix* disponibiliza ao utilizarmos sua estrutura para a criação de serviços são elencados.

Além da execução do *framework*, o *Felix* provê serviços que constam nesta especificação, tais como seguem no Quadro 6 (GÉDÉON, 2010).

Quadro 6 – Serviços disponíveis pelo arcabouço *Apache Felix*

Serviço	**Descrição**
Mensagens de *Log*	As mensagens de *Log* são utilizadas pelos *bundles* no contexto do *framework*.
Serviços de HTTP	Os serviços de HTTP proveem interface *Web* aos *bundles* e permite a interação com usuários da rede.
Serviços de Configuração *Admin*	Os serviços de Configuração *Admin* são usados para gerenciar a configuração dos dados do *bundle*.
Serviços de Metatipo	Os serviços de Metatipo permitem descrever tipos de atributos que os *bundles* podem usar como argumentos.
Configuração de Preferências	As configurações de Preferências são usadas para armazenar configurações e preferências dos *bundles*, como perfis de usuário.
Execução de Componente	A execução de componente provê um modelo de componente orientado a serviço para simplificar o desenvolvimento de aplicações baseadas em OSGi.
O serviço de *Event Admin*	O *Event Admin* facilita a troca de eventos como meio de comunicação entre *bundles* usando o modelo *publish-subscribe*.
Dispositivo UPnP	Os dispositivos UPnP ajudam na integração destes dispositivos em uma rede ponto a ponto usando XML sobre HTTP.

Fonte: Elaborada pela autora.

Além destes serviços, o *Felix* provê outros serviços com o objetivo de facilitar o desenvolvimento de aplicações e tarefas de administração do *framework*, tais como seguem no Quadro 7.

Quadro 7 – Outros serviços do arcabouço *Felix*

Serviço	Descrição
Gerenciamento de Dependência	O Gerenciamento de Dependências usa uma abordagem declarativa para facilitar este gerenciamento.
Instalação de Arquivos	A Instalação de Arquivos visa facilitar a implantação do *bundle*.
Gogo	O serviço *Gogo* é um método utilizado para que haja interação entre o sistema operacional e os *frameworks* OSGi.
iPOJO	O iPOJO provê componentes orientados a serviço.
Maven Bundle Plugin (MBP)	O MBP provê a automação no processo de criação do *bundle*, com o propósito de reduzir intervenções manuais.
Maven SCR Plugin (MSP)	O MSP permite o uso de serviços declarativos para automatizar a criação de descritores de metatipos.
Serviço de Repositório OSGi	Este serviço de repositório permite habilitar a conexão remota de repositório de *bundles*, listar os *bundles* instalados no *framework* e manipular as implantações de dependências dos *bundles*.
Shell Service, *Remote Shell Service* e *Shell* TUI	Estes serviços proveem meios de interação entre *bundles* no *framework*, utilizando linhas de comando do console.
Serviço de Console *Web*	Este serviço provê um console gráfico de administração, através de um navegador *Web*

Fonte: Elaborada pela autora.

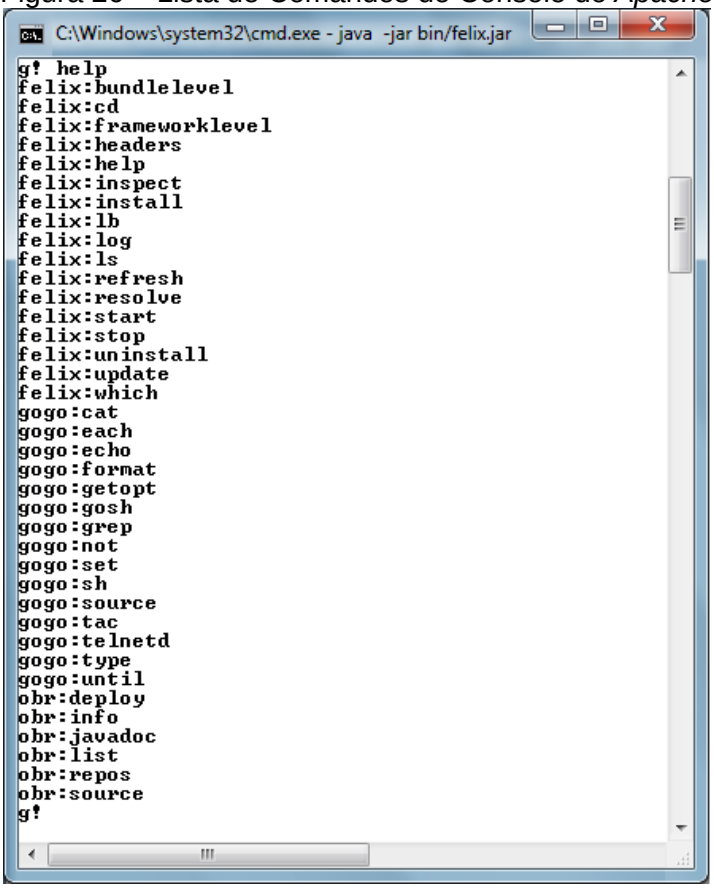
Observa-se, nestes serviços, a presença do *Maven Bundle Plugin* e do *Maven* SCR *Plugin*, sugerindo a possibilidade de uso do *Maven* (APACHE, 2015a) como recurso para auxiliar no gerenciamento de dependências e ciclo de vida na criação de *bundles* de serviços. Isto torna o *Apache Felix* um arcabouço que possui serviços que constituem uma seleção para construção de diversas aplicações.

4.3.2 Estrutura Interna do Arcabouço

Esta seção apresenta a estrutura interna do *Apache Felix*, sendo representada pelo interpretador de comandos. Esta estrutura estará disponível apenas após o processo de instalação do *Apache Felix*, que pode ser visto no Apêndice B.

Com o processo de instalação concluído, alguns comandos disponíveis no arcabouço do *Felix*, utilizados para interagir no seu ambiente, são como seguem na Figura 26.

Figura 26 – Lista de Comandos do Console do *Apache Felix*



Fonte: Elaborada pela autora.

Conforme a figura, para ter acesso aos comandos disponíveis para interação com o console, entra-se com o comando *help*. Observa-se que os comandos são categorizados de acordo com a versão do pacote instalado no *framework*.

Para a criação do *bundle*, as etapas deste processo podem ser vistas no Apêndice B. Após estes procedimentos, o IDE pode para criar um pacote de serviços OSGi, utilizando o arcabouço *Apache Felix*, que, neste projeto, está usando a especificação R6 do OSGi.

A versão do pacote que foi instalado não possui alguns comandos para gerenciar os serviços dos *bundles*. Para isto, é necessário instalar pacotes adicionais. Os comandos para gerenciar o ciclo de vida dos *bundles* no *Felix* são os

mesmos que gerenciam o ciclo de vida dos *bundles* no *Eclipse Equinox*. Mas é possível observar algumas diferenças entre os interpretadores de comandos dos arcabouços *Felix* e *Equinox*.

4.3.3 Curva de Aprendizagem

A seguir estão os detalhes da curva de aprendizagem do arcabouço *Apache Felix*, utilizando o projeto de teste já citado anteriormente.

Concluídas a instalação e a configuração, cria-se um novo projeto *Java* para construir o pacote de serviços OSGi. No processo de criação do projeto, após informar o seu nome, o *Eclipse* solicita que seja informada qual biblioteca será utilizada. Informa-se o uso do *User Library*, selecionando, assim, a biblioteca com as APIs do OSGi utilizando o *Felix*, configurada na seção anterior.

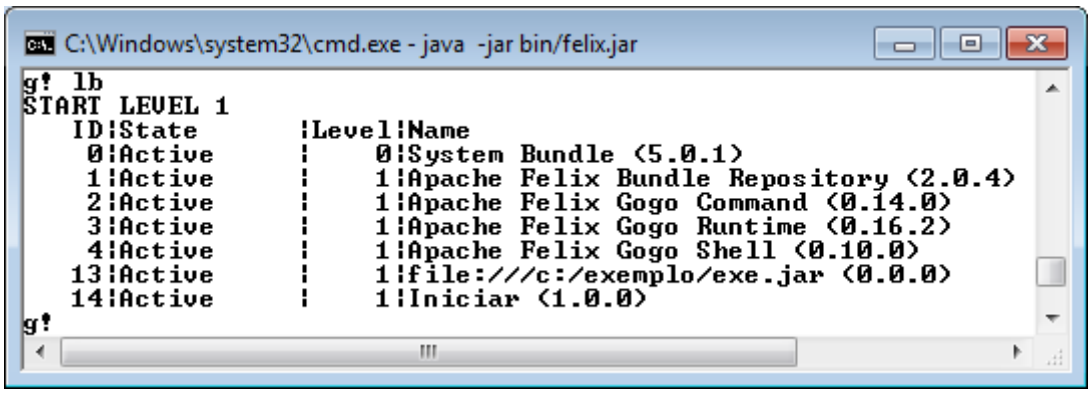
A partir deste ponto, o arquivo *Activator.java* pode ser programado, com os códigos já vistos anteriormente. Notou-se uma particularidade ao criar o manifesto (arquivo *MANIFEST.MF*), que foi efetivada apenas após a compilação do projeto. Nos procedimentos de compilação, é solicitada a criação do *MANIFEST.MF*. Finalizada a compilação, o arquivo é criado apenas com o cabeçalho *Manifest-Version*, sendo que os outros cabeçalhos, que são importantes na execução do *bundle*, devem ser colocados manualmente. O código deste arquivo foi visto na seção 4.1.

Observou-se, ainda, que o arquivo poderia ter sido criado antes da compilação, solicitando a criação de uma nova pasta (*META-INF*), dentro do projeto, e a criação de um novo arquivo dentro desta pasta (*MANIFEST.MF*). Mas, ainda assim, os cabeçalhos são inseridos pelo desenvolvedor.

Os *bundles* podem ser instalados no interpretador de comandos. As etapas deste processo de instalação podem ser vistas no Apêndice B. Após a instalação, o *framework* informa o número do ID do *bundle*, que será útil para controlar os seus estados. Os comandos *start* e *stop* podem ser digitados no console, para, respectivamente, iniciar e finalizar as atividades do *bundle* de teste.

Para verificar o estado, o nome e os IDs dos *bundles*, o comando utilizado é *lb*, e a saída fica de acordo com a Figura 27.

Figura 27 – Saída do comando lb do arcabouço Apache Felix



Fonte: Elaborada pela autora.

Como foi visto na seção 4.3.1, o *Apache Felix* possui serviços adicionais que possibilitam o uso do *Maven*, um recurso útil no gerenciamento de dependências. Isto pode indicar que o *Felix* é um *framework* compatível com o *Maven*, podendo o desenvolvedor utilizá-lo para controle de dependências. Observou-se que o IDE *Eclipse* possui meios para configurar o uso do *Maven* integrado ao seu ambiente, sugerindo que qualquer *framework* pode efetivar o uso do recurso. Entretanto, isto não indica que para a construção de *bundles*, a instalação e o uso do *Maven* sejam necessários.

Para concluir, nesta etapa de programação dos *bundles*, no que diz respeito à criação do arquivo manifesto, pode ser uma característica determinante na seleção do arcabouço. Neste exemplo de teste foi criado apenas um *bundle*, onde este possuía apenas uma classe que implementa uma interface e um arquivo manifesto. Considerando uma situação real, onde mais projetos, pacotes e classes estejam envolvidos, criar o manifesto manualmente pode ser uma solução pouco eficaz, devido a possibilidade de erros e dificuldades na localização destes erros, o que parece tornar o processo pouco eficiente.

4.4 IMPLEMENTAÇÃO OSGI – ARCABOUÇO *MAKEWAVE KNOPFLERFISH*

O projeto *Knopflerfish* (KF) é baseado no *middleware* OSGi, conhecido como Plataforma de Serviços Distribuída da *Gatespace* (*Gatespace's Distributed Service Platform* - GDSP) e está em desenvolvimento desde 1999. A organização *Gatespace* foi um dos membros que deram origem a Aliança OSGi, que desenvolvia também aplicações voltadas para a área de telecomunicações (MAKEWAVE AB, 2015).

Em 2003, no processo de reestruturação da *Gatespace*, que foi separada em duas áreas de negócios, uma delas conhecida como *Gatespace Telematics AB*, foi criada uma organização conhecida como *Knopflerfish*, que recebeu toda a estrutura do GDSP e passou a distribuir publicamente o código-fonte. Em seguida, a *Gatespace Telematics AB* mudou o nome para *Makewave* e continua mantendo soluções para as diversas áreas, tais como automotiva, empresarial, industrial e residencial, utilizando o *framework* OSGi.

O KF é uma plataforma de serviços compatível com as especificações OSGI R4 e R5, mantida e liderada pela *Makewave* e é um sistema de *container* para projetos *Java* (KNOPFLERFISH, 2015a). Atualmente a *Makewave* disponibiliza também o *Knopflerfish Pro*, que possui alguns recursos adicionais. Mas neste projeto será utilizada a versão mais simples.

As seções 4.4.1, 4.4.2 e 4.4.3 a seguir apresentam, respectivamente, características como o conjunto de serviços disponíveis, a estrutura interna do arcabouço e a curva de aprendizagem dos serviços.

4.4.1 Conjunto de Serviços Disponíveis

Os serviços disponibilizados pelo KF são discutidos a seguir, considerando a versão do arcabouço KF que utiliza a especificação R5 do OSGi.

Estes serviços são elencados como seguem (KNOPFLERFISH, 2015b):

a) o *Framework* OSGi *Knopflerfish*, que é prática da Especificação do Núcleo (*Core*);

b) os *bundles* OSGi *Knopflerfish*, que fazem parte da execução do Resumo de Serviços (*Service Compendium*);
c) os componentes e extras *Knopflerfish*, que são os *bundles* e utilitários *Knopflerfish*.

Os quadros que seguem detalham cada um dos serviços e seus componentes suportados pelo arcabouço, informando a descrição do serviço ou componente e o seu estado atual considerando o nível de implantação no arcabouço. O Quadro 8 apresenta os componentes do KF, que estão relacionados com o modelo de camadas da arquitetura do OSGi, que já foram vistos na seção 3.3.

Quadro 8 – Especificação OSGi R5 – *Framework* OSGi

Seção	***Status***	**Descrição**
Camada de Segurança (*Security Layer*)	Completo	A camada de Segurança do OSGi é baseada no mecanismo de segurança do *Java*2.
Camada de Módulo (*Module Layer*)	Completo	A solução do OSGi para modularização *Java*, conhecido como *bundles*
Camada Ciclo de Vida (*Life Cycle Layer*)	Completo	Função para controlar operações de segurança e ciclo de vida dos *bundles*.
Camada de Serviço (Service Layer)	Completo	Provê o modelo Publicar – Localizar – Conectar (*publish,find and bind model*) através do registro de serviço.

Fonte: Adaptado de Knopflerfish (2015).

O Quadro 9 relaciona as APIs suportadas pelo KF, relacionando-as com as especificações do OSGi.

Quadro 9 – Especificação de APIs OSGi R5

Seção	***Status***	**Descrição**
API Recurso (*Resource* API)	Completo	Esta API não é usada diretamente pelo arcabouço, mas permite que seja usada na construção de blocos para outras especificações.
API Ligação *Bundle* (*Bundle Wiring* API)	Completo	API que provê acesso ao mecanismo interno de compartilhamento de pacotes do *framework*.

Namespaces do *Framework*	Completo	Provê modularidade no cabeçalho do manifesto na forma genérica.
API Nível de Início (*Start Level* API)	Completo	API para controlar a ordem relativa de início e interrupção dos *bundles* na plataforma OSGi.
API *Framework*	Completo	API para interação entre *bundles* e o *framework* OSGi.

Fonte: Adaptado de Knopflerfish (2015).

A relação de outros serviços providos pelo KF é apresentada no Quadro 10.

Quadro 10 – Especificação de Serviços OSGi R5

Seção	***Status***	**Descrição**
Conditional Permission Admin	Completo	Política de segurança e sistema de permissão para *bundles*.
Permission Admin	Completo	Sistema de permissão para *bundles*.
URL *Handler*	Completo	Mecanismo para estender a execução do *Java* com novos esquemas URL e manipulador de conteúdo através de *bundles*.
Resolver Hooks	Completo	A camada de módulo é responsável por resolver operações que ligam requisitos (Import-Package,Require-Bundle etc.) aos recursos (*Export-Package*, *Bundle-SymbolicName*/*Bundle-Version* etc.).
Bundle Hooks	Completo	Conjunto de mecanismos para controlar a visibilidade de *bundles* por outros *bundles*.
Service Hooks	Completo	Conjunto de mecanismos para interagir com o Registro de Serviços.
Weaving Hooks	Completo	Provê meios para um *bundle* manipulador interceptar qualquer chamada de classes no *framework* para transformar *bytecodes* em classes para prover funcionalidades adicionais.
Tracker Specification	Completo	Utilitário para rastreio de *bundles* e serviços.

Fonte: Adaptado de Knopflerfish (2015).

As especificações e resumo de serviços do KF são como seguem no Quadro 11.

Quadro 11 – Especificação e Resumo de Serviços - *Bundles*

Especificação	***Status***	**Descrição**
Log	Estável	Registrador de mensagem da Plataforma de Serviços OSGi.
HTTP	Estável	Servidor HTTP para Plataforma de Serviços OSGi. Suporta páginas HTML estáticas e *servlets* em conteúdo dinâmico.
Device Access	Estável	Especificação de Acesso a Dispositivo suporta a coordenação de detecção automática e interligação de dispositivos existentes na Plataforma de Serviços OSGi.
Configuration Admin	Completo	Serviço para configuração de informações de *bundles*.
Meta Type	Completo	Serviço que permite aos desenvolvedores descrevem tipos de atributos de *bundle* utilizando metadados.
Preferences	Estável	Serviço que provê uso de armazenamento persistente de dados de *bundles*, preferências de usuários.
User Admin	Estável	Serviço utilizado pelo *bundle* para Autenticação e Autorização.
IO *Connector*	Estável	Serviço conector que adota o *Java* 2 ME *javax.microedition.io* para OSGi.
Declarative Services	Completo	Serviço que define um modelo declarativo para publicar, localizar e conectar aos serviços OSGi.
Event Admin	Completo	Provê mecanismo de comunicação entre *bundles* através do envio de eventos.
Repository	Em andamento	Serviço para busca e recuperação de recursos gerais (*bundles*).
XML *Parser Service*	Estável	Utilitário de XML *parser* padronizado.
Position	Estável	Utilitário para manipular posições geográficas.

Measurement and State	Estável	Utilitário que provê uma forma consistente de manipular um conjunto de medidas para desenvolvedores de *bundles*.

Fonte: Adaptado de Knopflerfish (2015).

O arcabouço também inclui um conjunto de *bundles* e utilitários específicos. Alguns dos componentes incluem, ou são baseados em, outros projetos ou componentes de código aberto (como acontece com o *Common Logging* – Apache; Portas Seriais – *Sun* e RXTX; e KF *Test* – *JUnit*). Nestes casos, o *Knopflerfish* provê uma classe que envolve estes componentes. No Quadro 12 os componentes específicos do KF estão relacionados.

Quadro 12 – Componentes *Knopflerfish*

Componentes *Knopflerfish*		
Componente	***Status***	**Descrição**
Desktop	Estável	Ferramenta gráfica para gerenciar a execução do *framework* e pode ser estendido usando *plug-ins*.
Utilitários de Log	Estável	Classes de utilitários para uso em registro de informações.
Utilitários Misc	Estável	Várias classes de utilitários para textos, tarefas e manipulação de conexões I/O.
Console	Estável	Provê meios para ferramentas baseadas em comandos para publicá-los. Qualquer *bundle* pode prover comandos.
Console TTY	Estável	Console utilizando conexões I/O de padrões de entrada (*stdin*) e saída (*stdout*)
Console Telnet	Estável	Console usando protocolo *Telnet*
Console – Comandos do *Framework*	Estável	Comandos para manipular o núcleo do *Knopflerfish*.
Console – Comandos de *Log*	Estável	Comandos para manipular o *log* do OSGi.
Console – Comandos CM	Estável	Comandos para manipular os dados de gerenciamento de Configuração.

Fonte: Adaptado de Knopflerfish (2015).

O Quadro 13 apresenta os componentes extras, que utilizam código de outros projetos.

Quadro 13 – Componentes Adicionais do *Knopflerfish*

Componente	***Status***	**Comentário**
Commons Logging	Estável	O *bundle Commons logging* envolve a API Apache *Commons Logging* sobre o *log* OSGi
Porta serial	Estável	Suportada através do modelo de dispositivo OSGi e API *javax.comm*. As implementações da *Sun* (*Win32*) e RXTX (*Linux*) são usadas nos *bundles*.
KF *Test Suite*	Estável	Conjunto de testes baseado em JUnit para testar *bundles* e arcabouço *Knopflerfish*.

Fonte: Adaptado de Knopflerfish (2015).

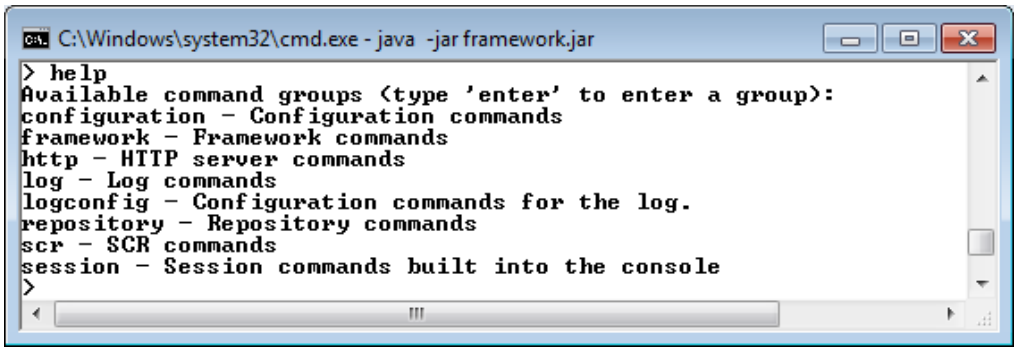
A disponibilização das informações acerca dos serviços e APIs do OSGi disponíveis no arcabouço KF tornou mais fácil a identificação do que ele pode prover de recursos que auxiliem na construção de um projeto OSGi.

4.4.2 Estrutura Interna do Arcabouço

Esta seção apresenta a estrutura interna do KF, representada pelo interpretador de comandos, onde é possível manipular os *bundles* que foram construídos no ambiente de programação. Para ter acesso ao interpretador de comandos do KF, é necessário efetuar a sua instalação. As etapas deste processo de instalação podem ser vistas no Apêndice C.

Os comandos internos utilizados no console do KF podem ser acessados utilizando o comando *help*. A saída do comando é como segue na Figura 28.

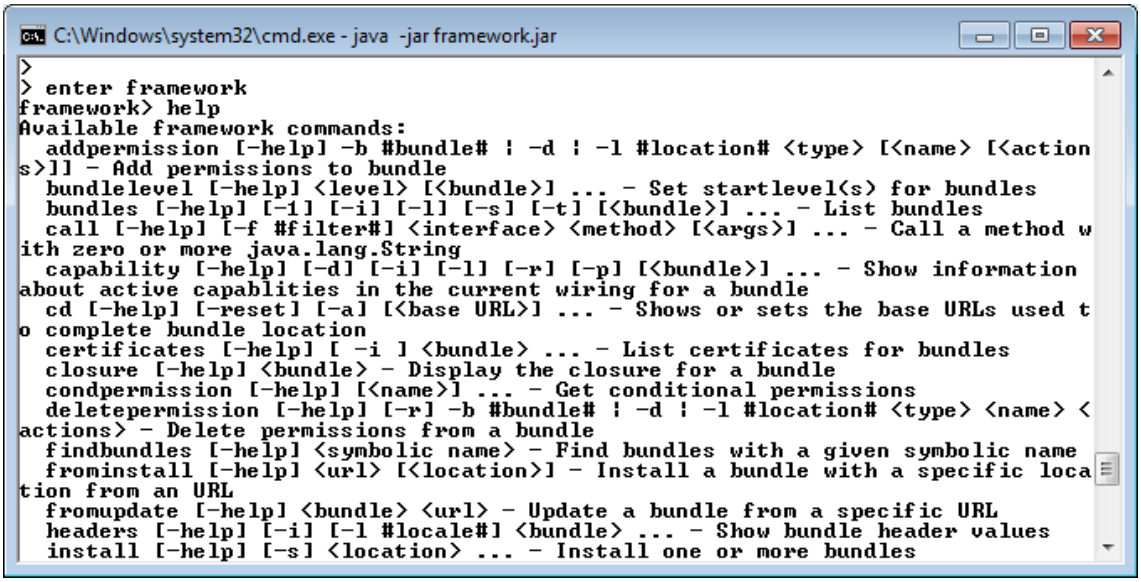
Figura 28 – Saída do comando help no arcabouço KF



Fonte: Elaborada pela autora.

Os comandos internos são separados em categorias. Para acessá-los, a categoria do comando deve ser informada, digitando o comando *enter* e o nome da categoria. Segue a demonstração de um exemplo na Figura 29.

Figura 29 – Saída do comando enter para acessar um grupo de comandos no KF



Fonte: Elaborada pela autora.

Para construir o *bundle*, deve-se configurar a IDE para dar suporte às APIs que são utilizadas pelo arcabouço KF (maiores detalhes constam no Apêndice C).

É possível observar que o interpretador de comandos do KF é parecido com o *Equinox*, pois ambos separam os comandos em categorias. Portanto, para acesso aos comandos do KF é necessário acessar as categorias, enquanto que no *Equinox*, esta exigência não existe.

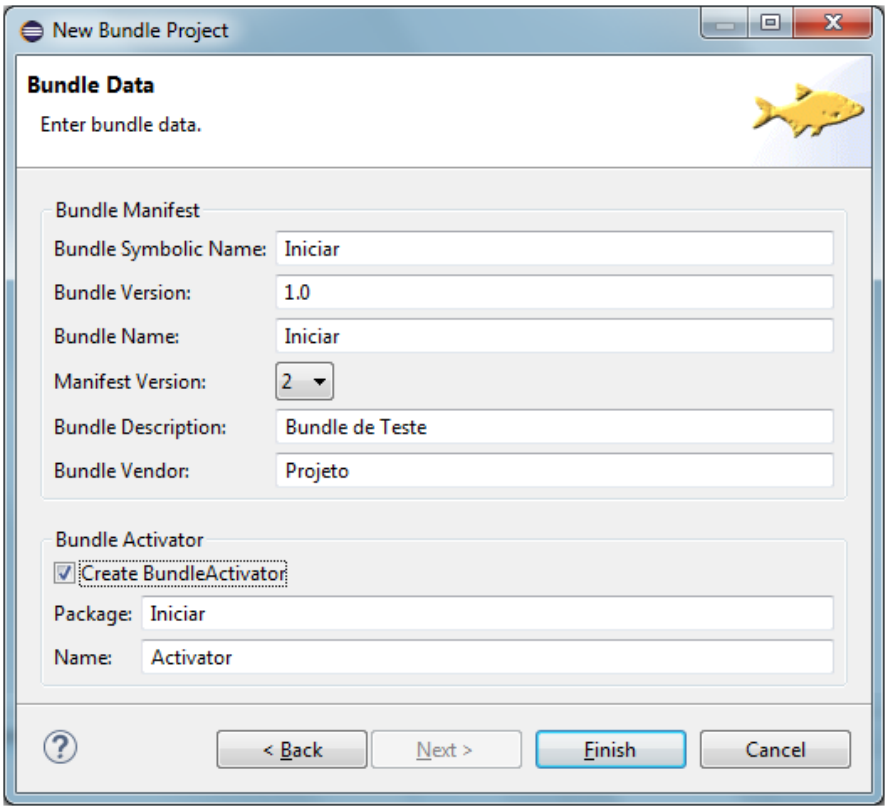
4.4.3 Curva de Aprendizagem

A curva de aprendizagem do arcabouço será discutida adiante, utilizando o projeto de teste já apresentado na seção 4.1.

Para construir o projeto de teste, é possível seguir as etapas que constam no Apêndice C.

Conforme exibe a Figura 30 é possível criar o arquivo manifesto informando os dados relacionados ao projeto que está sendo construído. Diferentemente dos outros arcabouços, o nome do arquivo manifesto no arcabouço KF é *bundle.manifest*. A função do arquivo é a mesma, conforme foi visto na seção 3.3.1.

Figura 30 – Criando e configurando o arquivo *bundle.manifest*

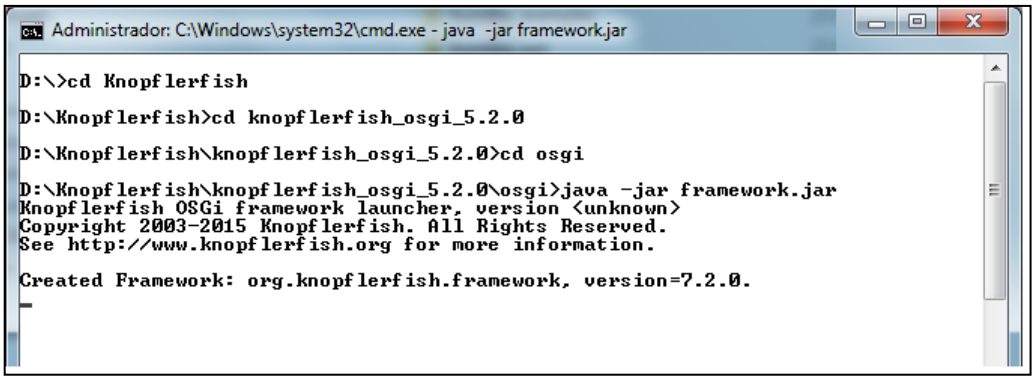


Fonte: Elaborada pela própria autora.

Depois de finalizado, o arquivo *bundle.manifest* fica de acordo o que foi visto no início deste capítulo. Apenas uma diferença com relação ao arquivo manifesto já visto, é que o KF insere o cabeçalho *Bundle-Description*, com a descrição do pacote de serviços OSGi. Relacionando com o arcabouço *Equinox*, o arquivo manifesto do KF não possui a versão dos pacotes importados, o que consta no referido cabeçalho do arquivo manifesto do *Equinox*. O arquivo *Activator.java*, já criado junto com o projeto, foi atualizado de acordo com a funcionalidade do *bundle*. O seu código também foi visto anteriormente.

O *Knopflerfish Desktop* é utilizado para manipular os *bundles* que foram criados. A inicialização do *Knopflerfish Desktop* está como na Figura 31.

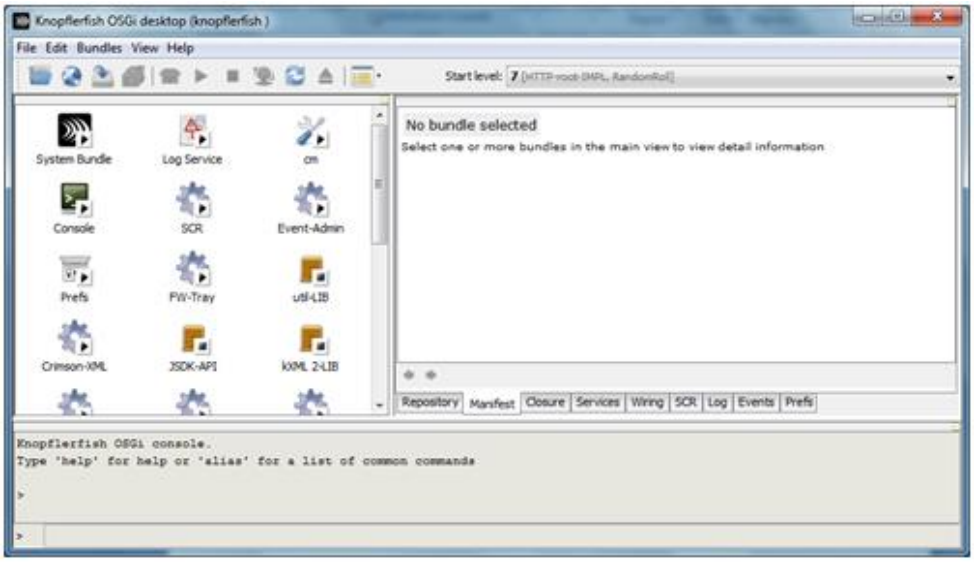
Figura 31 – Inicializando o *Knopflerfish Desktop*



Fonte: Elaborada pela própria autora.

Depois de executar o comando para acionar o *Desktop* do KF, a aplicação se torna disponível para que seja possível abrir o *bundle* que foi criado, além de possibilitar a verificação de outros detalhes, como a sua identificação. A Figura 32 apresenta o *Knopflerfish Desktop*.

Figura 32 – Aplicação *Knopflerfish Desktop* para testar o projeto

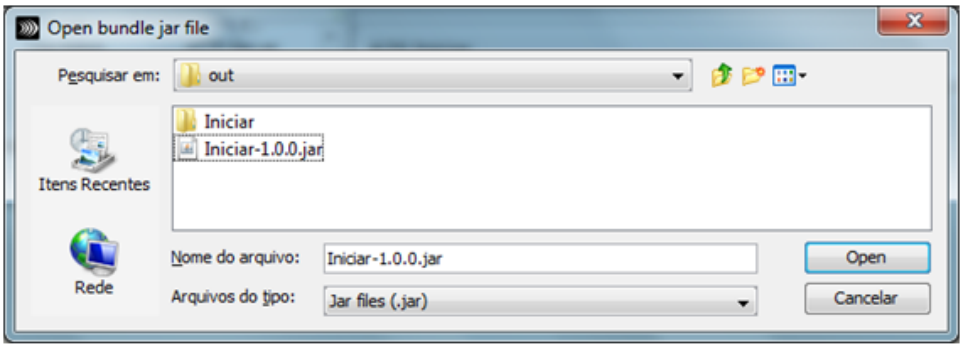


Fonte: Elaborada pela própria autora.

Uma diferença entre o KF e os outros arcabouços é que, ao executar o comando para acessar o console do *framework*, todo controle pode ser feito de forma gráfica, através da aplicação *desktop* e do uso de botões de comando. Existe também a opção de utilizar o interpretador de comandos no *framework* KF. Nos arcabouços *Equinox* e *Felix* toda a manipulação com os *bundles* é feita, exclusivamente, através de linhas de comando no interpretador de comandos.

Para abrir o *bundle* que foi criado, seleciona-se a pasta do *workspace* onde está localizado o projeto, conforme a Figura 33. O *bundle* estará na pasta *Out*, na mesma localização do projeto. Isto é um padrão do KF.

Figura 33 – Abrindo o *Bundle* Inciar



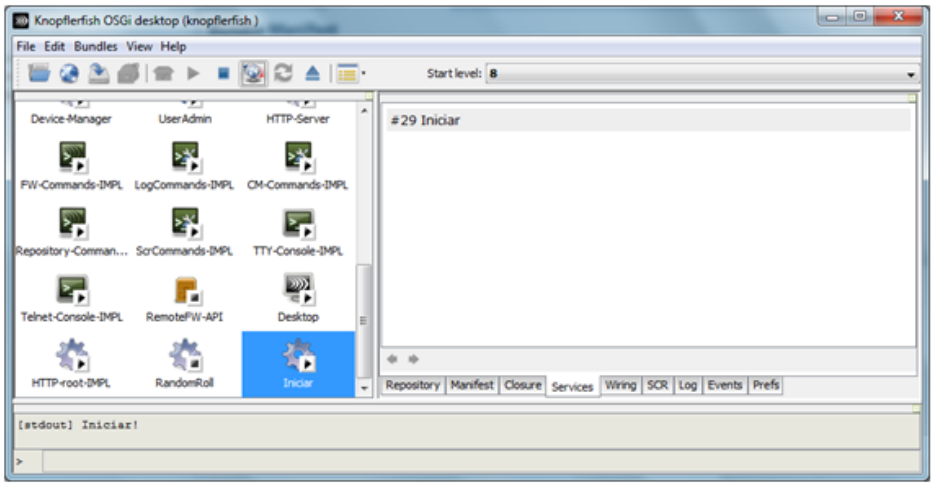
Fonte: Elaborada pela própria autora.

O procedimento da figura acima é equivalente a realizar a compilação do projeto, criar o arquivo manifesto durante a compilação e instalar o *bundle* utilizando o comando *install* no interpretador de comandos. Caso seja feito o uso da alternativa de instalar o *bundle* através de linhas de comando, utiliza-se a sintaxe conforme segue:

install file:///<caminho/do/arquivo>/<arquivo>.jar

Na aplicação *Desktop*, depois que o *bundle* é aberto, ele pode ser iniciado e finalizado, apenas para visualizarmos o teste, e também pode permitir outras funcionalidades, como instalação e desinstalação, de acordo com as características de todo *bundle* OSGi. Na parte inferior da tela aparece o console com as informações de saída do *bundle,* como pode ser visto na Figura 34.

Figura 34 – Executando o *Bundle* Iniciar

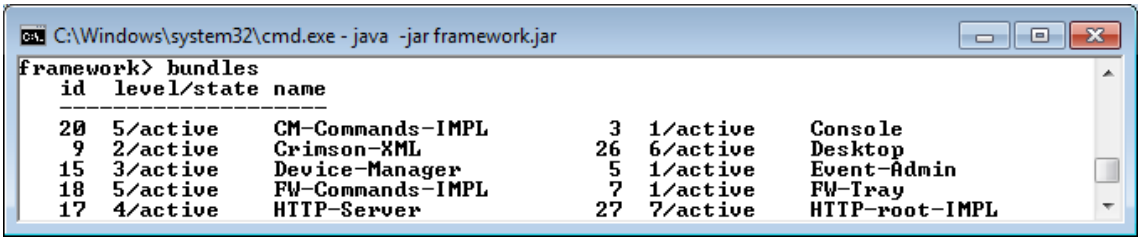


Fonte: Elaborada pela própria autora.

Para iniciar e finalizar o *bundle*, entra-se com os comandos *start* e *stop*, respectivamente, na linha de comandos (na parte inferior) do *Desktop* ou entra-se com estes comandos no interpretador de comandos. Uma alternativa é clicar com o botão direito do *mouse* sobre o *bundle* e selecionar os comandos *start* ou *stop*.

Para ter acesso aos estados e aos IDs dos *bundles*, se a opção for utilizar as linhas de comando, o comando *bundles* pode ser digitado, conforme a Figura 35.

Figura 35 – Saída do comando *bundles* no *shell* do arcabouço KF



Fonte: Elaborada pela autora.

Para concluir, a respeito desta característica de programação de serviços OSGi, utilizar o KF tornou o processo mais rápido que os demais. A presença da aplicação *Desktop* apenas confirma a agilidade na criação dos *bundles* de serviço bem como testes e verificação de outros detalhes destes *bundles*, úteis no projeto de modo geral.

4.5 AVALIAÇÃO DE CARACTERÍSTICAS DE ARCABOUÇOS PARA O SUPORTE AO DESENVOLVIMENTO DE *GATEWAY* RESIDENCIAL

Os arcabouços foram avaliados considerando algumas características que irão balizar a escolha para uso na prototipação do projeto, considerando mais uma avaliação qualitativa, no que tange o desenvolvimento dos *bundles* de serviços. Não faz parte do objetivo deste projeto identificar qual o melhor, mas sim o que se adapta no desenvolvimento da aplicação proposta neste trabalho.

O Quadro 14 busca relacionar os arcabouços e as características escolhidas para determinarem qual deles o protótipo desenvolvido no Capítulo 6 será implementado.

Quadro 14 – Comparação entre os arcabouços e as características selecionadas

Arcabouço	Conjunto de Serviços, APIs e bibliotecas Disponíveis (I)	Estrutura Interna (II)	Curva de Aprendizagem (III)
Eclipse Equinox	Versão R4	Comandos organizados em categorias	Construção dos arquivos *Activator* e manifesto de forma manual
Apache Felix	Versão R6; *plugin* de integração com o *Maven*	Ausência de comandos para gerenciar os serviços dos *bundles*	Construção dos arquivos *Activator* e manifesto de forma manual
Makewave KF	Versão R5	Comandos organizados em categorias, aplicativo Desktop para utilizar os comandos internos	Construção dos arquivos *Activator* e manifesto automática, durante a criação do projeto

Fonte: Elaborada pela autora.

É possível observar na característica (I), que o *Eclipse Equinox* possui um conjunto considerável de pacotes de serviços, visto que utiliza uma versão já consolidada do *framework* OSGi (Versão R4) e inclusive possui no seu ambiente de desenvolvimento recursos que facilitam a inserção destes pacotes para desenvolver aplicações OSGi, caso seja utilizado o *Eclipse* PDE. As APIs disponíveis no *Equinox* atende a uma variedade de aplicações OSGi pela diversidade de recursos, mas é o arcabouço que utiliza a versão mais antiga das especificações OSGi.

Na característica (II), observa-se que o *Eclipse Equinox* disponibiliza comandos por categorias, facilitando a localização e função dos comandos.

Qualquer palavra digitada que não seja um comando interno no Equinox, ele interpreta como sendo um comando de ajuda do interpretador de comandos.

Na característica (III), observa-se que o *Eclipse Equinox*, por já possuir os *templates*, pode tornar o processo simples e a possibilidade de usar o console no próprio ambiente de desenvolvimento objetiva na facilidade ao testar a aplicação. Se o projeto for construído sem base nos *templates*, a criação do manifesto é feita de forma manual. Isto deve ser levado em consideração se constam, no projeto, muitas classes e muitos manifestos. E nele também é possível utilizar um console externo ao ambiente de execução, caso seja necessário testar a aplicação em outro equipamento.

Quanto o arcabouço *Apache Felix*, é possível identificar na característica (I) que ele também pode fazer uso do *Eclipse* como IDE de desenvolvimento de aplicações OSGi e é o mais atual (versão R6) em termos de serviços e bibliotecas disponíveis. Sem dúvida, o número de recursos disponíveis permite que o desenvolvedor tenha uma diversidade de opções ao programar os *bundles*, podendo aprimorar e acrescentar mais serviços que o OSGi disponibiliza. Um ponto a ressaltar é a presença de *plugins* para o uso de recursos do *Maven*, o que não é recurso dos outros arcabouços, muito embora o IDE *Eclipse Mars*, que foi utilizado na programação dos *bundles* no *Felix* e no KF, esteja integrado ao *Maven*, sendo possível configurá-lo no IDE.

Na característica (II) referente a estrutura interna com o uso de interpretador de comandos, é possível identificar que alguns comandos para gerenciar serviços não estão presentes no arcabouço *Felix*. Para que seja possível utilizar estes comandos, é necessário instalar pacotes adicionais no *framework*.

Na característica (III), é possível verificar que a construção do *bundle* de serviços no *Felix* foi muito parecida com o *Equinox*, com a exceção de que não tinha o recurso dos *templates*. Então a criação do *MANIFEST.MF* foi feita manualmente. A mesma análise feita no *Equinox*, também se aplica ao *Felix*, considerando um projeto mais extenso, onde tenham muitas classes e arquivos manifestos a serem configurados. Foi feita a tentativa de utilização do *plugin* que integra ao *Maven*, entretanto o recurso não funcionou conforme esperado nesta aplicação de teste.

Quanto o arcabouço *Knopflerfish*, é possível observar na característica (I) que, mesmo tendo sido um dos primeiros a adotar o *framework* OSGi, a última atualização de conjunto de serviços e bibliotecas é a R5. Neste arcabouço ficou mais simples identificar exatamente quais os serviços ele pode utilizar, referenciando as seções da especificação OSGi nas suas próprias especificações. Isto também facilita ao determinar por qual arcabouço optar ao programar aplicações.

Na característica (II), é possível verificar que a estrutura interna do arcabouço KF pode ser utilizada através do *Desktop*, que é um ambiente gráfico que permite o gerenciamento dos *bundles* e consequentemente a utilização de todos os comandos internos. Este recurso facilita os testes com os *bundles* e demais serviços.

Na característica (III) é possível identificar que o KF utiliza o ambiente familiar do *Eclipse*. Os *bundles* são criados utilizando recursos que o próprio arcabouço disponibiliza no processo de instalação e a criação do arquivo manifesto é feita de forma bastante simplificada. Na execução do *bundle* é necessário acessar o *Knopflerfish Desktop*, outro ambiente totalmente integrado ao *Eclipse*, e ao abrir o *bundle* para testar a execução, o processo ocorre, praticamente, de forma automática, com riqueza de detalhes, que, para que o mesmo aconteça em outros arcabouços, é necessário entrar com algumas linhas de comandos.

Diante do exposto acima, pode-se concluir que um framework que pode atender a implementação de aplicação de redes residenciais é o *Makewave Knopflerfish*. Nele podem ser desenvolvidos os pacotes necessários para funcionamento do serviço básico de redes residenciais do protótipo construído no Capítulo 6.

A seguir são discutidas as características de um *gateway* residencial e o uso do *Arduino* como *hardware* escolhido para compor a estrutura deste *gateway* e os recursos oferecidos nesta plataforma de código aberto.

5 *GATEWAY* RESIDENCIAL – DEFINIÇÃO E PROTOTIPAÇÃO

Segundo Wei e outros (2010), existem três coisas que uma residência precisa para se tornar inteligente;

a) uma é a rede interna, que é utilizada para ligar os aparelhos;
b) a outra é de automação residencial, aparelhos dentro das casas e enlaces para serviços e sistemas fora de casa;
c) a última é o controle inteligente, um *gateway* para gerir os sistemas.

De maneira geral, costuma-se ler os termos de *gateway* residencial, *home gateway*, *gateway* doméstico ou *residential gateway*, porém não são estritamente definidos e é amplamente utilizado para muitos dispositivos diferentes (WEI et al., 2010). Nesta dissertação, o *gateway* residencial é definido baseado nos conceitos de plataformas de serviço e no *framework* OSGi, no que tange serviços para casas inteligentes.

Home Gateway (HG) é definido como um sistema que suporta diversas tecnologias de rede, localizado nas instalações dos consumidores. O *Gateway* fornece recursos ao usuário residencial para acessar os serviços da *Internet* entregues à casa e também para acessar os diferentes serviços oferecidos pelos vários *appliances* (aparelhos ou equipamentos domésticos) localizados dentro dela (HGI, 2008).

Tecnicamente, o *gateway* residencial possui protocolos e tecnologias de rede de banda larga que permitem a comunicação da rede externa com a rede residencial (CRUZ et al., 2013), compartilha acesso à *Internet*, impressão, monitoramento remoto, conectividade de *appliances* utilizando tecnologias sem fio e comunicação de voz e vídeo sobre IP (HGI, 2008).

O funcionamento do HG é padronizado por uma entidade conhecida como HGI (*Home Gateway Initiative*) (HGI, 2014a) e estabelece algumas diretivas que compõem as características físicas do dispositivo. O HGI é uma organização sem fins lucrativos que publica instruções (protocolos, guias), documentos de requisitos e testa planos para equipamentos e serviços de banda larga que são implantados na "*connected home*" – rede residencial.

O escopo do trabalho desta organização engloba:

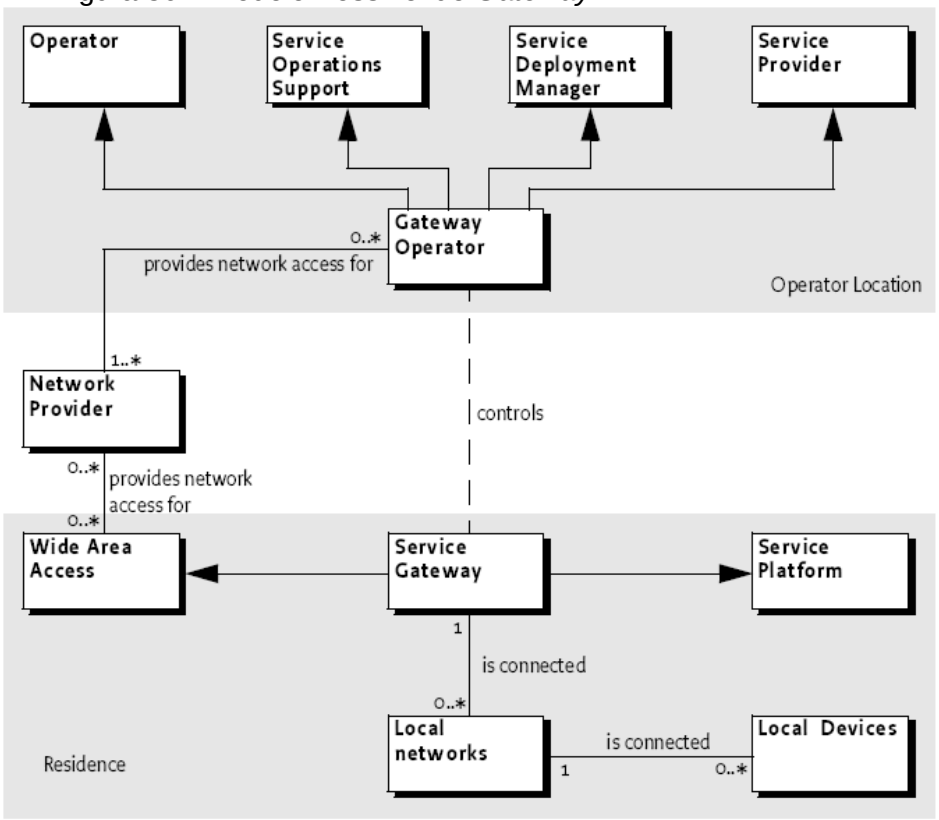
a) os dispositivos domésticos de interconexão;
b) a conexão com a rede de acesso e de banda larga;
c) os requisitos de redes residenciais;
d) os requisitos de *software;*
e) outras tecnologias necessárias para a entrega de serviços de banda larga e entrega de serviços para clientes residenciais, além das demandas de serviços *triple play* (voz, dados e multimídia) para a integração de dispositivos residenciais e aparelhos dentro de infraestruturas de serviços da WAN.

Esta integração requer uma base de *software* na rede residencial e a variedade de tecnologias disponíveis torna muito difícil o confronto com todos que utilizam funções padronizadas, especialmente para dispositivos distribuídos. Então, para obter o próximo nível de integração de serviço, existe uma necessidade por *softwares* modulares no HG.

Em se tratando de *softwares* modulares, a especificação OSGi propôs um modelo básico de *Gateway* de Serviços, dada a importância da área de aplicação, que lida com sistemas de grande escala com HGs (OSGI, 2003). A

Figura 36 apresenta o modelo proposto pela aliança OSGi e, na sequência, as características dos principais elementos envolvidos nesta arquitetura.

Figura 36 – Modelo Possível de *Gateway*



Fonte: OSGI (2003).

Em linhas gerais, é possível identificar duas estruturas onde os componentes estão disponibilizados:

a) a primeira estrutura é a Residência (*Residence*) composta pelos seguintes elementos:

   - o *Gateway* de Serviços (*Service Gateway*), que comumente está conectado à WAN. O *Gateway* de Serviços possui uma divisão lógica e pode representar uma Plataforma de Serviços (*Service Platform*) e prover Acesso a WAN (na

   Figura 36, o Acesso a WAN está representado como *Wide Area Access*), utilizando o mesmo recurso de *hardware*;

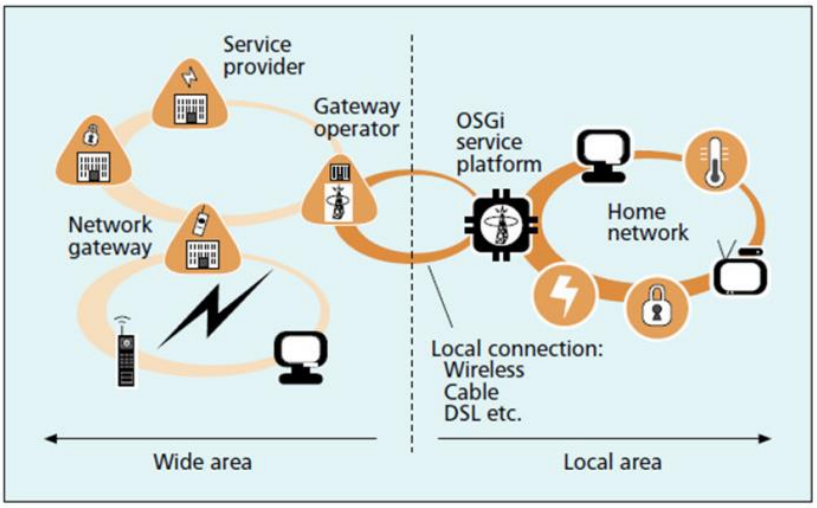
   - as Redes Locais (*Local Networks*), que também pode se referenciar a uma Rede Residencial;

   - os Dispositivos Locais (*Local Devices*), que são computadores, impressoras e qualquer equipamento que esteja apto a acessar a rede.

b) a segunda estrutura é a Localização do Operador (*Operator Location*), que é composta pelo Operador de *Gateway* e pode assumir as funções de Operador (*Operator*), de Suporte ao Serviço de Operações (*Service Operations Support*), de Gerenciador de Implantação de Serviço (*Service Deployment Manager*) e de Provedor de Serviços (*Service Provider*).

A Figura 37 ilustra um exemplo de um cenário de uma rede residencial, com os elementos supracitados.

Figura 37 – Exemplo de um cenário residencial com os elementos do modelo do *Gateway* Residencial sugerido pelo OSGi



Fonte: Breier e Pereira (2011).

Basicamente, estas funções são assumidas pelo Operador de *Gateway*, quando o *Gateway* ou Servidor da Plataforma de Serviços (SPS - *Service Plataform Server*) representar um *gateway* de comunicação. Neste caso, o Operador de *Gateway* responde pelas seguintes funções (OSGI, 2003):

a) de gerenciar o *gateway* de conexão WAN e as redes locais (função de *Service Provider*);
b) de Operador, para as Plataformas de Serviços ou *Gateway* de Serviços;
c) de gerenciador de implantação de serviços, para os serviços em execução no *Gateway* de Serviços;
d) de Operações de Serviços de Suporte, para os serviços em execução no *Gateway* de Serviços.

O elemento Provedor de Rede (*Network Provider*) é comum entre as duas estruturas. A sua relação com a Localização do Operador indica que pelo menos um Provedor de Rede disponibiliza uma ou mais tecnologias de acesso a WAN para Operadores de *Gateway*. Já a relação do Provedor de Redes com a Residência indica que pode ter ou não tecnologias de acesso a WAN disponíveis para as redes locais.

Como foi visto acima, o *gateway* de serviços possui uma divisão lógica, podendo executar as funções de uma plataforma de serviços e de um *gateway* para acesso à rede de banda larga. Nesta dissertação, é utilizada a função de plataforma de serviços do *gateway*, consoante informação no início deste capítulo. Não serão abordadas as funções de acesso à banda larga. Este último tópico poderá ser abordado em trabalhos futuros, onde seja possível, através dos recursos providos pelo *gateway* e uma estrutura adequada, controlar os equipamentos utilizando a *Internet*.

Nesta dissertação, a plataforma de serviços do *gateway* será útil no que diz respeito ao fornecimento de serviços básicos de rede doméstica, onde foi utilizado um arcabouço baseado no *framework* OSGi para o desenvolvimento do serviço e posterior acesso deste pelo usuário. O serviço será registrado no Registro de Serviços e tudo isto será gerenciado pelo arcabouço. O *gateway* possibilitará, enquanto plataforma de serviços, a utilização de protocolos de comunicação para acionar o equipamento que será controlado na rede residencial. Ele interpretará os comandos acionados pelo usuário no nível mais baixo, onde são manipulados os sinais binários ou níveis de tensão, de acordo com a tarefa que for designada para o controle do equipamento.

Por questões de modularidade e de programabilidade na escrita das classes em *Java*, a possibilidade de integração de muitas tecnologias de rede e adaptabilidade, o *framework* OSGi é o mais indicado. Consequentemente, o *hardware* que adotará o *framework* deverá ser compatível e coerente com este *framework*, por isto a escolha por um sistema de *gateway* especificado pelo OSGi.

O *hardware* escolhido para a construção do protótipo da aplicação de redes residenciais foi o *Arduino*, por ser uma plataforma de código aberto e que atende a uma série de requisitos que já foram elencadas anteriormente. A seguir, é apresentado o *Arduino* como esta alternativa e como foi desenvolvida a aplicação para funcionamento no protótipo.

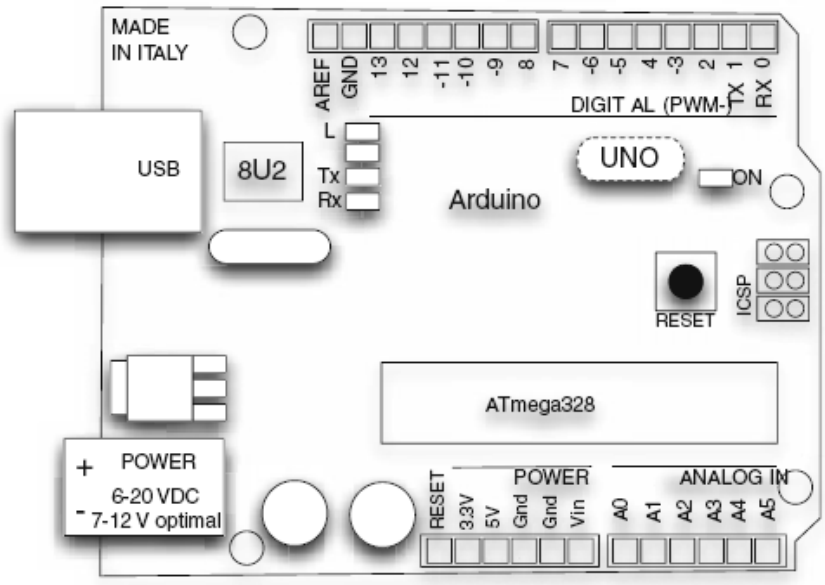
5.1 ARDUINO

Como elemento de validação do estudo do arcabouço para o desenvolvimento de aplicações residenciais, objeto desta dissertação, foi desenvolvido um protótipo de um *gateway*, utilizando padrões abertos de *hardware* e de *software*, para acionamento de serviços básicos de redes domésticas. Para isto, um dos passos foi determinar qual *hardware* seria utilizado para estas finalidades.

Por se tratar de uma plataforma que possui *hardware* e *software* de código aberto, para desenvolver a aplicação para redes residenciais, a placa *Arduino Uno Rev*. 3 foi escolhida. O *Arduino Uno* é uma plataforma de computação física ou embarcada baseada no microcontrolador *ATMega328* da família AVR da *ATmel Corporation* (EVANS; NOBLIN; HOCHENBAUM, 2013; MCROBERTS, 2011).

A Figura 38 apresenta a arquitetura da placa Arduino Uno utilizada no protótipo de *gateway*.

Figura 38 – Arquitetura do *Arduino Uno* Rev. 3



Fonte: Mcroberts (2011).

O *Arduino* fornece o suporte necessário para o microcontrolador. Para iniciar a placa é necessário conectá-la ao computador com entrada USB, utilizando um cabo USB, ou ligá-la num adaptador de energia AC-DC ou bateria.

As placas antecessoras ao *Arduino Uno Rev* 3 utilizam o chip *Future Technology Devices International* (FTDI) (FTDI, 2015), responsável por desenvolver dispositivos e dar suporte aos seus programas e *drivers* na conversão de sinais RS-232 para sinais USB. Isto é feito para tornar possível a comunicação de *hardwares*

mais antigos (que utilizam conexão RS-232) com aparelhos mais novos que já utilizam o padrão USB.

As placas *Arduino* mais recentes não utilizam o chip FTDI, elas possuem um microcontrolador adicional, o *AtMega8U2*, que também é programado para converter a conexão USB para serial. Além do custo da placa ter reduzido, a presença do *chip* tornou mais fácil a atualização do *firmware* do USB, que permite que, ao conectar ao computador, o *Arduino* seja reconhecido como um periférico (como um *mouse*).

É possível desenvolver muitas aplicações interativas independentes utilizando a placa *Arduino*, ou ainda é possível conectá-lo a um computador, a uma rede, ou à *Internet*, para recuperar e enviar dados para controlá-los. É possível conectar ao *Arduino* Diodos Emissores de Luz (LEDs – *Light Emitting Diodes*), botões, interruptores, motores, sensores de temperatura, sensores de pressão, sensores de distância, receptores GPS (*Global Positioning System* – Sistema de Posicionamento Global), módulos *Ethernet* ou qualquer outro dispositivo que emita dados ou que possa ser controlado (MCROBERTS, 2011).

O *Arduino* pode ser estendido utilizando *Shields* (escudos), que são placas adicionais que contém outros dispositivos incluindo *displays* LCD, módulos *Ethernet* e módulos *Bluetooth*. Existe também a opção de utilizar outras placas de circuito impresso, tais como *protoboard*, tendo a mesma funcionalidade de um *shield* (EVANS; NOBLIN; HOCHENBAUM, 2013) e normalmente com o custo mais reduzido. No projeto, será utilizado o *protoboard* para efetuar as ligações com o *Arduino*. O circuito da arquitetura que será utilizado no protótipo será ilustrado no capítulo 6. Alguns detalhes adicionais sobre o *Arduino* podem ser vistos no Apêndice D.

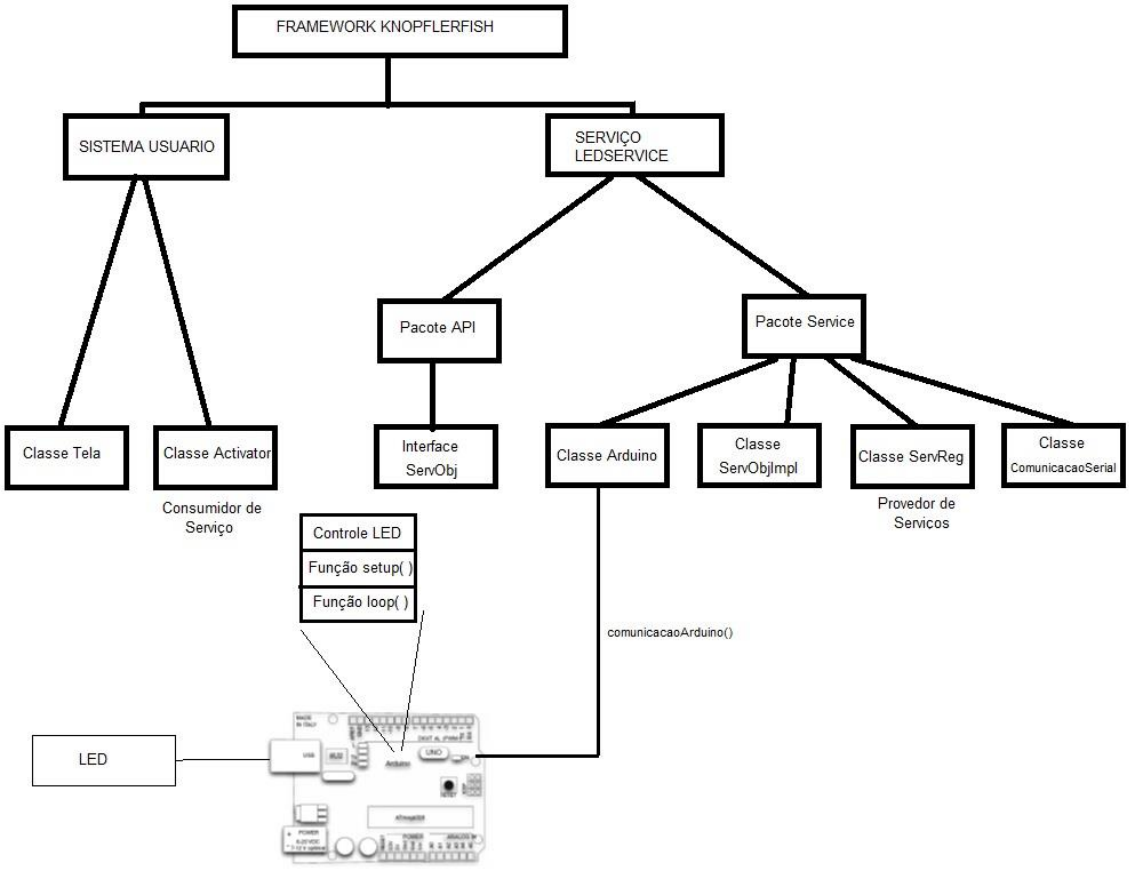
O capítulo a seguir discute como foi feita a criação do serviço básico, que será executado na aplicação de redes residenciais e o uso do arcabouço *Knopflerfish* como estrutura adequada para o desenvolvimento desta aplicação.

6 ESTUDO DE CASO - PROTOTIPAÇÃO

Os protocolos e padrões bem como as arquiteturas de *hardware* e os *softwares* possibilitam a implantação do protótipo de um *gateway* residencial para acionar um serviço básico de rede residencial. Estes elementos foram vistos ao longo desta dissertação para validar o protótipo desenvolvido neste capítulo.

A Figura 39 apresenta o cenário da prototipação, especificando os serviços implementados.

Figura 39 – Prototipação utilizando o arcabouço KF



Fonte: Elaborada pela autora.

Nesta figura, é possível identificar o seguinte:

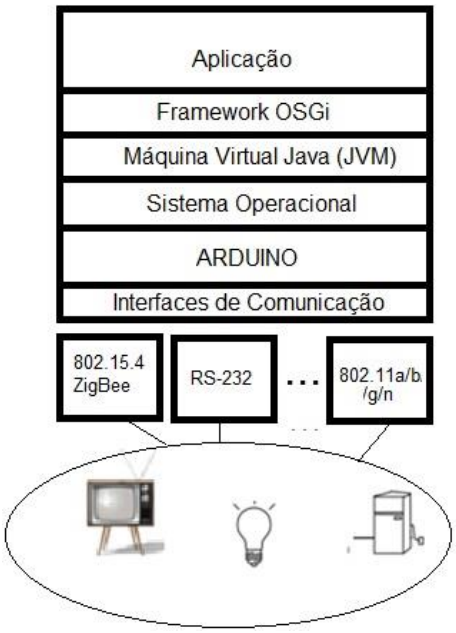
a) a escolha do serviço básico de redes, que é um serviço simples para acionamento de um LED (ligar e desligar);

b) a utilização do arcabouço KF para a criação do sistema do usuário e do serviço básico de redes;
c) a utilização de uma plataforma de hardware aberta (Arduino) para compor o cenário do gateway residencial;
d) a construção do código que faz comunicação entre o serviço de redes e o hardware.

A estratégia da criação do Sistema Usuário é para estabelecer comunicação com o aparelho doméstico a ser controlado, através do *gateway*. O Sistema Usuário é um sistema amigável, onde o usuário poderá acionar os comandos de controle dos aparelhos. A ideia é que a aplicação possa estar disponível em diversos dispositivos móveis, facilitando o seu manuseio. Na prototipação, a aplicação foi feita para ser utilizada em um computador, mas pode ser adaptada para outros dispositivos.

O *gateway* residencial fará a ligação com os equipamentos a serem controlados, interpretando as informações enviadas pelo Sistema Usuário e emitindo os sinais de controle para a aparelhagem. Ele terá o suporte para interfaces (que conectarão os aparelhos através do uso de diversos padrões de rede), um sistema operacional, máquina virtual *Java* e o suporte para comunicação com os arcabouços do *framework* OSGi. A Figura 40 representa a arquitetura para o protótipo.

Figura 40 – Arquitetura do Protótipo

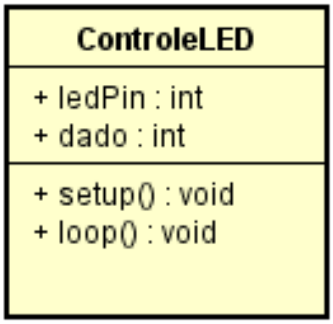


Fonte: Elaborada pela autora.

Foi elaborada uma estrutura para acionar um LED, utilizando uma placa de circuito impresso (um *protoboard*) com um resistor. Esta estrutura pode ser vista no Apêndice E.

As principais funções para acionar o LED que está conectado ao Arduino estão como segue o diagrama de classes na Figura 41.

Figura 41 – Diagrama de Classes com as principais funções de acionamento do LED no *Arduino*



Fonte: Elaborada pela autora.

De acordo com a figura, as variáveis *ledPin* e *dado* foram criadas para armazenar informações do estado do LED. A variável *ledPin* é referente à porta digital (5) do *Arduino* onde a placa de circuito impresso faz a ligação do LED. A variável *dado* envia à esta porta digital um sinal para ligar (energização), se o valor enviado para ela for 1 ou um sinal para desligar (desenergização), se o valor enviado para ela for 2.

As funções que efetivam o acionamento do LED são *setup()* e *loop()*, e ambas não retornam qualquer valor. Na primeira função constam as configurações da porta de comunicação, tais como a taxa de transferência que tem o valor de 9600 bits por segundo (configuração padrão de uma porta serial) e a definição do pino de saída (porta digital). Esta informação é importante, pois este pino irá receber o sinal (1 ou 2) para controlar o LED.

A função *loop()* contém informações sobre o envio dos sinais. Esta função fica em repetição até que o *Arduino* seja desligado ou que um novo programa seja carregado para sua memória. Mais detalhes sobre as linhas de código podem ser vistas no Apêndice F.

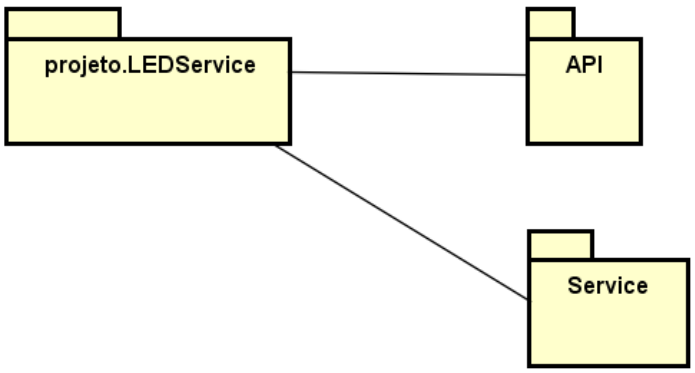
O código escrito é compilado e carregado no ambiente de programação do *Arduino*. Esta é a base de comunicação do *software* com o *Arduino*, que fará acesso direto ao LED. Este código será acessado por uma aplicação feita no arcabouço do *framework* OSGi.

Como foi mostrado no capítulo 4, os principais arcabouços que utilizam as APIs do OSGi são o *Apache Felix*, o *Makewave Knopflerfish* e o *Eclipse Equinox*. Após a análise das suas características, ficou definido que o serviço básico será implantado e executado no arcabouço *Knopflerfish*, pois o objetivo final no que diz respeito à utilização dos arcabouços é avaliar características que determinem o ambiente adequado à criação dos pacotes de comunicação para executar em aplicações de redes residenciais.

6.1 PROTOTIPAÇÃO NO ARCABOUÇO *MAKEWAVE KNOPFLERFISH*

O KF possui uma estrutura de desenvolvimento bem definida, facilitando a utilização das bibliotecas do OSGi. Para desenvolver a aplicação que fará comunicação com o *Arduino*, foi criado o *bundle LEDService*. O *LEDService* possui dois pacotes. Um pacote permitirá comunicação com a aplicação a ser utilizada pelo usuário, no caso desta dissertação, o Sistema Usuário. O outro pacote contém uma interface que será útil no consumo do serviço. A divisão dos pacotes do referido *bundle* é como segue na Figura 42.

Figura 42 - Representação da divisão dos pacotes que formam o *bundle LEDService*



Fonte: Elaborada pela autora.

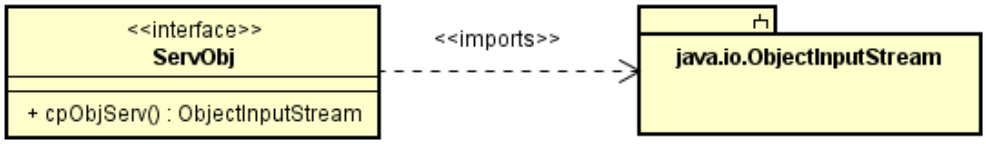
O primeiro pacote é o *projeto.LEDService.API* é composto por uma classe que cria uma interface de acesso que consumirá o serviço provido pelo *bundle*. A representação do diagrama pode ser visto na Figura 43.

Figura 43 – Representação da composição de classes do pacote *projeto.LEDService.API*

Fonte: Elaborada pela autora.

Esta interface foi criada para fazer ligação com a aplicação do usuário. O detalhe da implementação desta interface pode ser visto na Figura 44.

Figura 44 - Representação da Interface *ServObj*

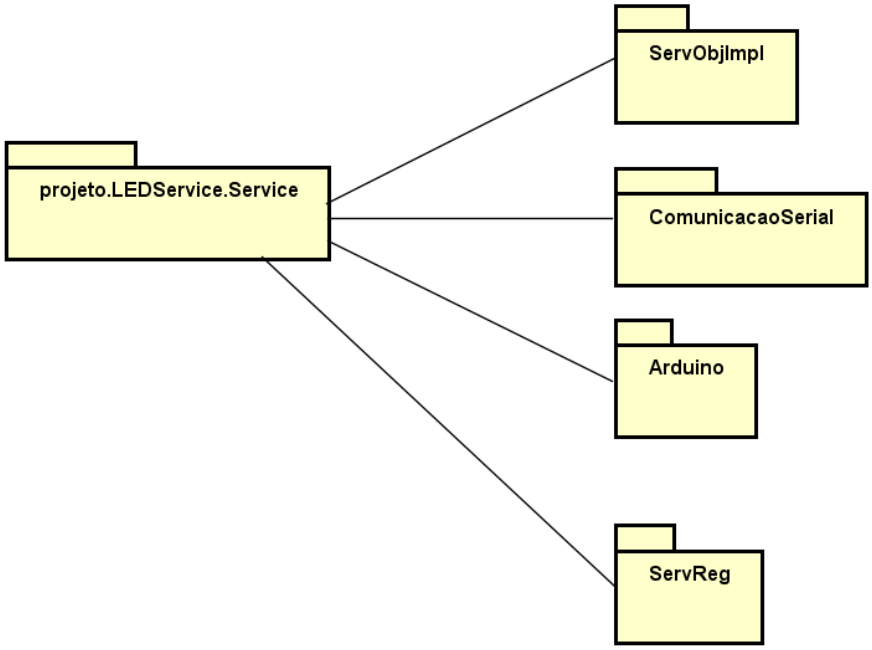


Fonte: Elaborada pela autora.

A função do método *cpObjServ()* é criar uma cópia de qualquer objeto que seja passado como parâmetro. Esta cópia será retornada para o cliente que requisitou o objeto. No caso deste projeto, o objeto será chamado na aplicação do usuário, quando for solicitado o início e término do serviço do *bundle*.

O segundo pacote do *bundle LEDService* é o pacote de serviços que foi chamado de *projeto.LEDService.Service*. Ele é composto por classes que irão prover a comunicação com a interface serial que está conectada ao *Arduino*, como foi visto na seção 5.1. O diagrama da Figura 45 representa as classes que foram criadas no pacote *projeto.LEDService.Service*.

Figura 45 - Diagrama com as classes criadas no pacote *projeto.LEDService.Service*



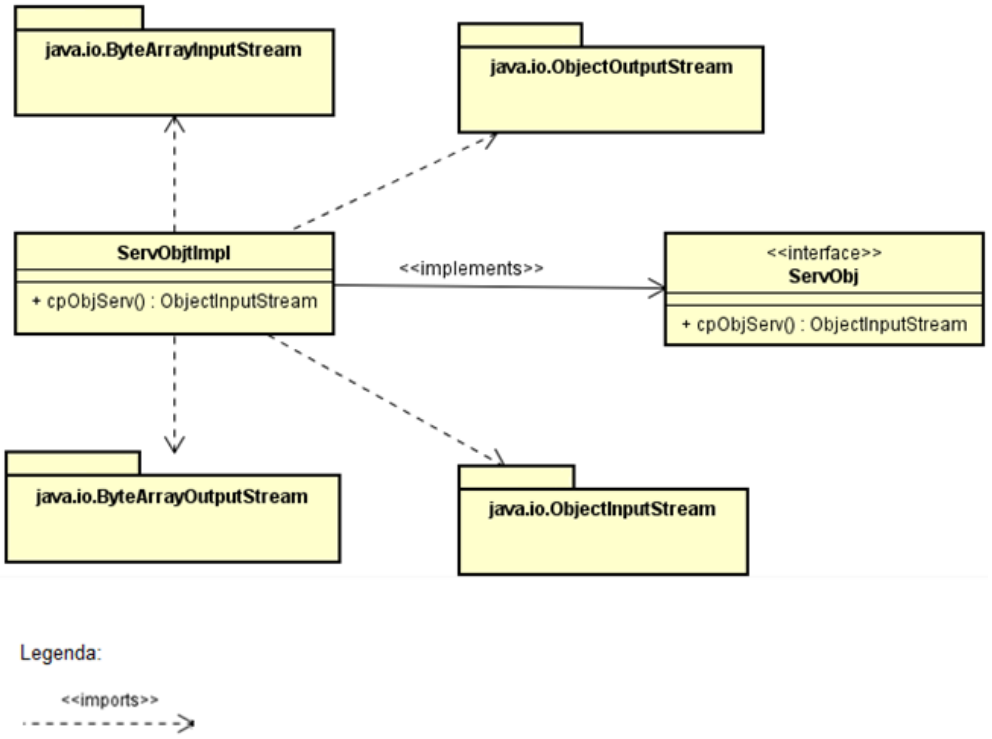
Fonte: Elaborada pela autora.

Conforme o diagrama da Figura 45, o pacote *Service* é composto por:

a) uma classe que implanta a interface *ServObj*, que foi chamada de *ServObjImpl*;
b) uma classe que parametriza a comunicação serial, que está sendo chamada de *ComunicacaoSerial*;
c) uma classe para realizar a comunicação com o *Arduino*, que é a classe *Arduino*;
d) uma classe que importa as bibliotecas do *framework* OSGi e registra os serviços disponíveis no *container*, que é *ServReg*.

A classe *ServObjImpl.java* implementa a interface *ServObj* e absorve a função de fazer uma cópia de qualquer objeto que seja passado como parâmetro. Considerando a boa prática de desenvolvimento de aplicações, a interface *ServObj* e a classe *ServObjImpl.java* estão em pacotes distintos, evitando, assim, que os detalhes da implementação fiquem expostos. A Figura 46 representa a classe *ServObjImpl.java*.

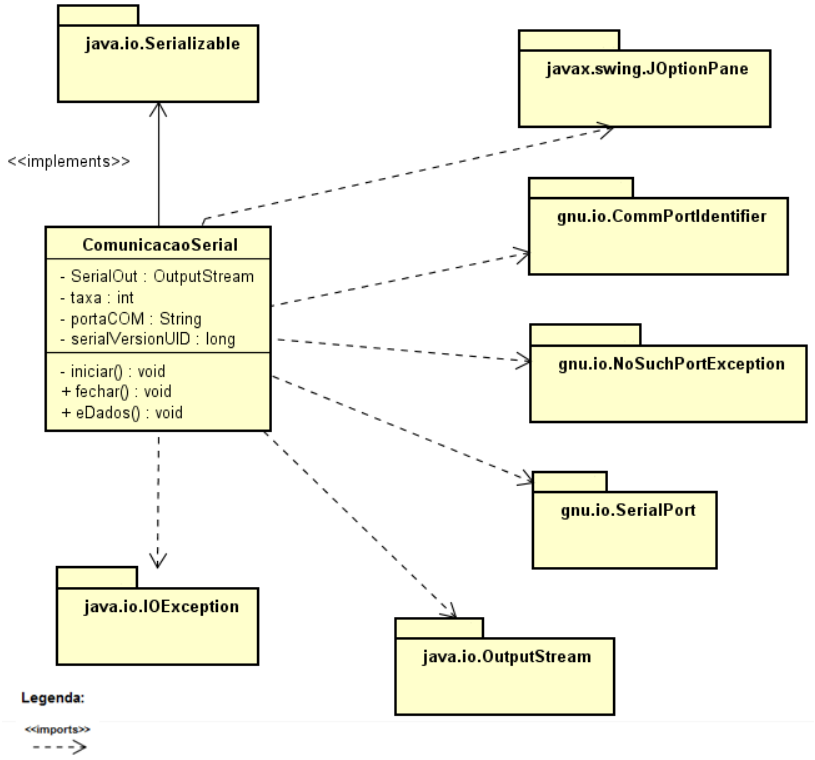
Figura 46 - Representação da Classe *ServObjImpl*



Fonte: Elaborada pela autora.

A classe que implanta a comunicação serial importa pacotes do padrão RS-232, que também pode ser utilizada pelo padrão USB. O pacote é o *RXTX* e possui um conjunto de classes que habilita a comunicação das portas COM. A classe *ComunicacaoSerial.java* fornece este suporte para a comunicação serial, informando o tipo de porta, parametrizando a taxa de transferência, os *bits* de paridade, *bit* de parada e *bit* de dados, que são informações importantes para configurar a comunicação serial, como foi visto na seção 2.5. A representação desta classe pode ser vista na Figura 47.

Figura 47 - Representação da Classe *ComunicacaoSerial.java*

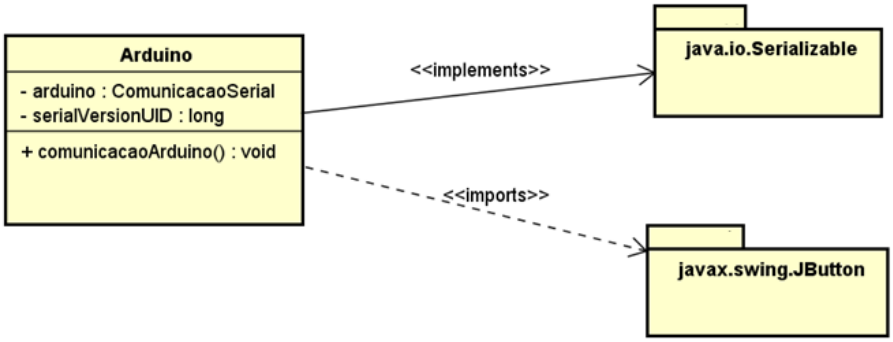


Fonte: Elaborada pela autora.

O método *iniciar()* tem a função de permitir conexão com a porta serial COM possibilitando a troca de dados com quem estiver conectado a ela, neste caso, o *hardware Arduino*. Caso a porta não seja localizada, não sendo possível, assim, acessá-la para encaminhar os dados, uma mensagem é exibida. O método *fechar()* é responsável por finalizar a conexão com a porta COM. E o método *eDados()* foi criado para enviar os sinais para acionar o LED, caso a comunicação com a porta serial já tenha sido estabelecida. Os três métodos citados não retornam informações ao finalizarem suas tarefas.

A classe *Arduino.java* efetiva a comunicação com o *Arduino*, fazendo também uma ligação com a classe *Comunicacaoserial.java* e relacionando-se com o código que foi escrito no ambiente de programação do *Arduino*. Esta classe possui o método *comunicacaoArduino()* (que não retorna qualquer dado) que chama as funções criadas no *Arduino*, para ligar e desligar o LED. A ligação com o *Arduino* foi realizada na porta COM5, logo esta classe necessita deste parâmetro para estabelecer o contato com o *hardware*. A sua representação é como segue na Figura 48.

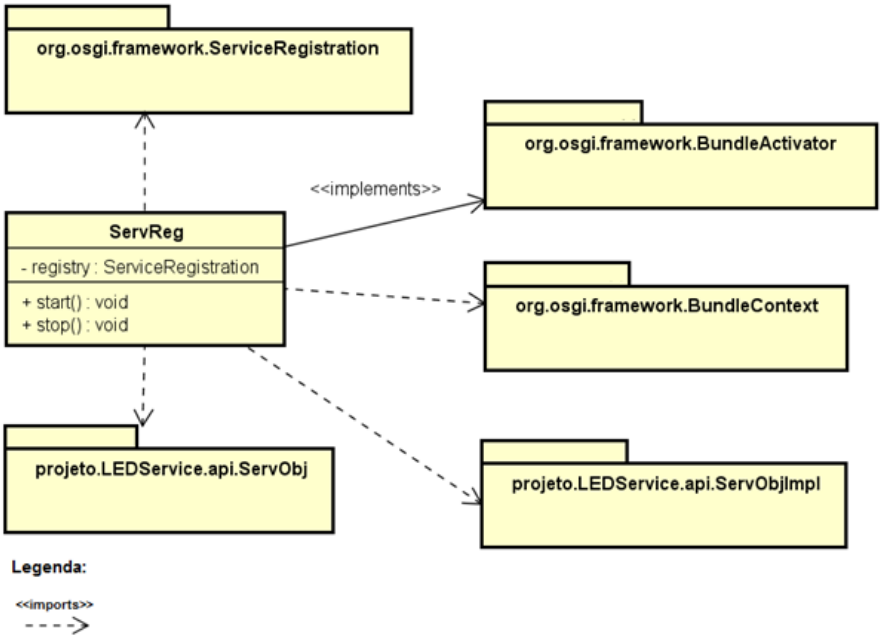
Figura 48 - Representação da Classe *Arduino.java*



Fonte: Elaborada pela autora.

O registro de serviços publica os serviços que ficam no *container* e são reconhecidos através de suas interfaces, como foi visto na seção 3.3.2. A construção da classe *ServReg.java* tem a finalidade de efetuar o registro dos serviços, onde nela são implantadas bibliotecas do *framework* OSGi e é utilizada a interface *BundleActivator*. O método *start()* (que implementa *BundleActivator*) passa como parâmetro uma variável (*context*) do tipo *BundleContext*. A variável *registry* do tipo *ServiceRegistration* armazenará uma informação, através do atributo *registerService* de *context*, efetivando o registro do serviço *LEDService*. No método *stop()* o registro do serviço é retirado. O diagrama na Figura 49 exibe a classe *ServReg.java*.

Figura 49 - Representação da Classe ServReg.java



Fonte: Elaborada pela autora.

Finalizando a construção do *bundle LEDService*, o arquivo *bundle.manifest* é parametrizado para permitir o funcionamento dos pacotes criados. Nele são informados os pacotes que serão exportados, no caso a o pacote *Service*, que será útil ao sistema do usuário e é considerado externo ao pacote *Service*, e o cabeçalho *bundle-Activator*, que é configurado para a classe que registra os serviços, no caso a classe *ServReg.java*. A Figura 50 mostra as informações do arquivo.

Figura 50 - Arquivo *bundle.manifest* do pacote *projeto.LEDService.Service*

```
Manifest-Version: 1.0
Bundle-Description: Projeto LEDService
Bundle-SymbolicName: projeto.LEDService
Bundle-Name: projeto.LEDService
Bundle-Version: 1.0.0
Bundle-ManifestVersion: 2
Bundle-Activator: projeto.LEDService.service.ServReg
Bundle-Vendor: Projeto
Import-Package: org.osgi.framework, gnu.io, javax.swing
Export-Package: projeto.LEDService.Service
```

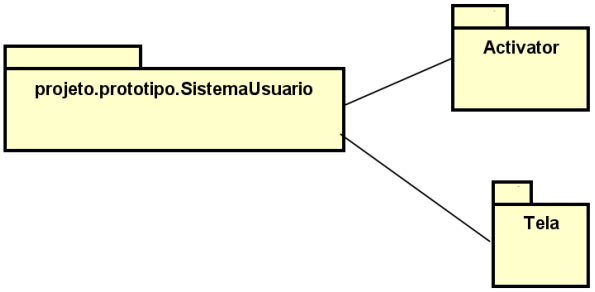
Fonte: Elaborada pela autora.

Em seguida, o serviço pode ser iniciado. O arquivo JAR criado com os *bundles* desenvolvidos está disponível para ser instalado e utilizado por alguma aplicação cliente. Lembrando que para instalar o *bundle*, pode ser inserido o comando *install* no console do *framework* e, em seguida, o comando *start* (que executará o que contiver dentro do método *start()* da classe de registro de serviços *ServReg*). Com a classe iniciada, é possível acessar a lista de serviços disponíveis, com o comando *services*.

A aplicação cliente que irá consumir este serviço é o sistema do usuário. Este sistema também utiliza bibliotecas do *framework* OSGi para acessar o *bundle LEDService*, mas isto não quer dizer que apenas aplicações baseadas no OSGi consumam estes serviços. Para que aplicações não OSGi acessem *bundles* OSGi, é necessário que o número de classes criadas na camada de serviços seja o mínimo possível, para diminuir o acoplamento com o *framework*.

Neste projeto foi construído o pacote *projeto.prototipo.SistemaUsuario* e este possui duas classes. A estrutura do referido pacote é dividida conforme diagrama da Figura 51.

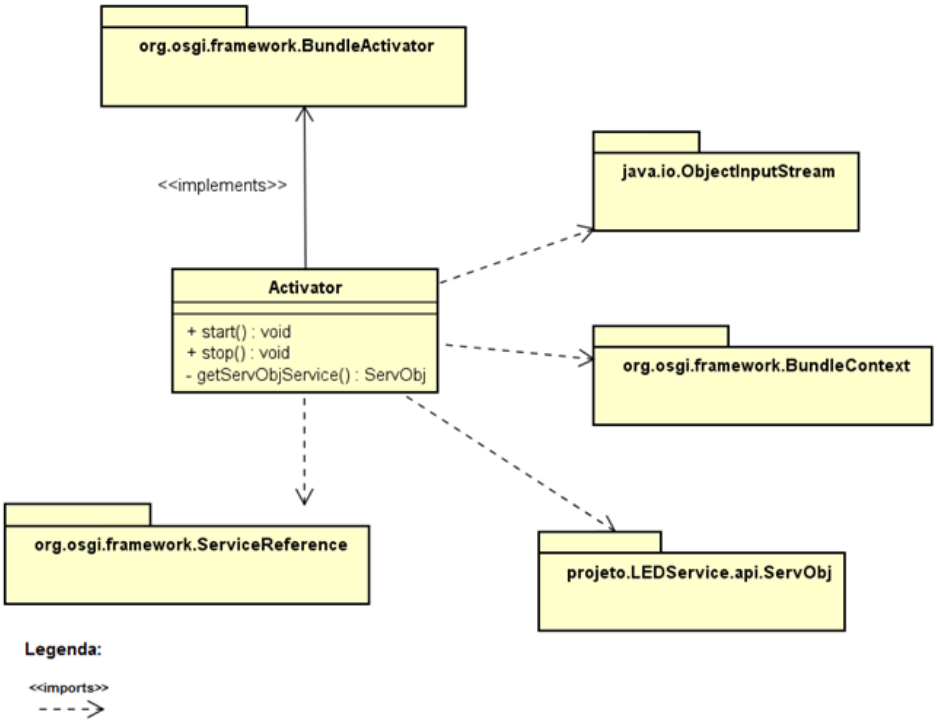
Figura 51 - Representação das classes que formam o Sistema Usuário



Fonte: Elaborada pela autora.

A classe *Activator.java* tem a função de localizar o serviço no registro de serviços, através da interface *ServiceReference*, e de se conectar ao sistema do usuário para que este possa realizar os comandos de acionamento do LED. A Figura 52 demonstra o diagrama de classes da classe *Activator.java*.

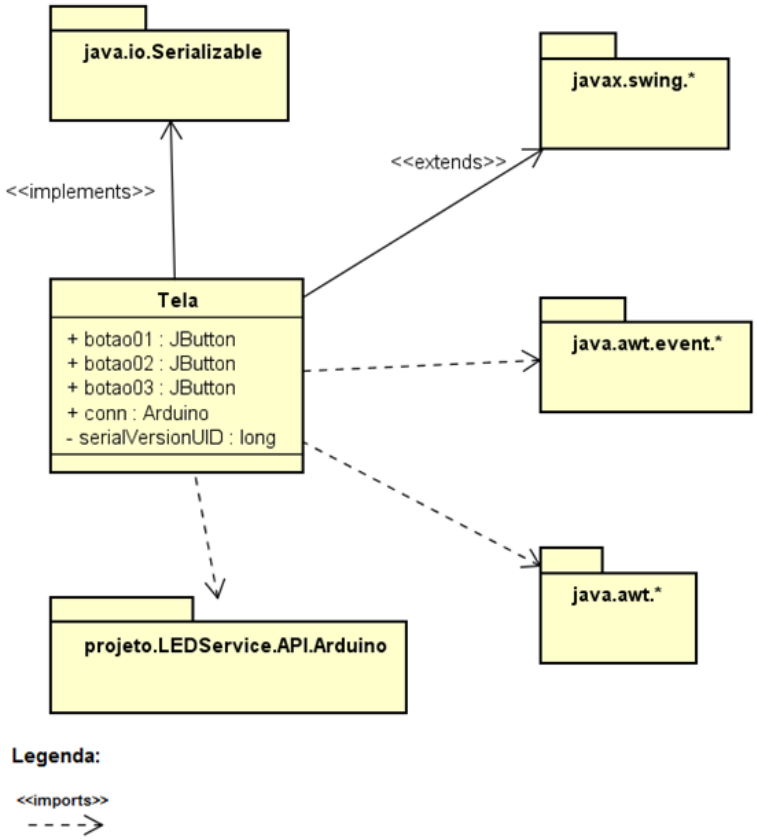
Figura 52 - Representação da classe *Activator.java* do pacote *Sistema Usuário*



Fonte: Elaborada pela autora.

A classe *Tela.java*, que representa a aplicação que faz interface com o usuário, foi criada para permitir que o usuário interaja com as operações com o LED. É um sistema amigável, com botões de acionamento dos comandos. Esta classe relaciona-se com a classe *Arduino.java* do pacote *projeto.LEDService.API*, já que é nesta que constam os métodos que manipulam os comandos do LED. O diagrama da Figura 53 demonstra a classe *Tela.java*.

Figura 53 - Representação da classe *Tela.java*



Fonte: Elaborada pela autora.

Quando a aplicação é executada e o usuário seleciona o botão *Ligar* (*botao01*), internamente, o método *AddActionListener()* é acionado. Ele lê o texto do botão (“Ligar”) e aciona outro método na classe *Arduino.java*, que se comunica com a porta serial já parametrizada, e que, por sua vez, chama a aplicação que foi desenvolvida no ambiente do *Arduino*. O resultado desta sequência de ações e métodos é o LED aceso. Uma sequência similar acontece quando o botão de *Desligar* (*botao02*) é acionado pelo usuário.

Por fim o arquivo *bundle.manifest* que parametriza o pacote criado, informando os pacotes importados no campo *Import-Package* e o *bundle* que habilitará o pacote, no campo *Bundle-Activator*. A Figura 54 exibe os parâmetros para este arquivo.

Figura 54 - Arquivo *bundle.manifest* do pacote *projeto.prototipo.SistemaUsuario*

```
Manifest-Version: 1.0
Bundle-Description: Sistema Usuario
Bundle-SymbolicName: projeto.prototipo.SistemaUsuario
Bundle-Name: projeto.prototipo.SistemaUsuario
Bundle-Version: 1.0.0
Bundle-ManifestVersion: 2
Bundle-Activator: projeto.prototipo.SistemaUsuario.Activator
Bundle-Vendor: Projeto
Import-Package: org.osgi.framework, javax.swing, projeto.LEDService.Service
```

Fonte: Elaborada pela autora.

Os códigos das classes com alguns métodos utilizados para a construção do *bundle LEDService* e do *Sistema Usuário* podem ser vistos nos Apêndices G, H e I.

O que se pode concluir ao utilizar este arcabouço, diz respeito ao suporte à criação de *bundles* e os recursos utilizados para testar o que foi criado. Por ser uma estrutura que implanta completamente as especificações do OSGi, ela caracteriza um ambiente adequado para aplicações simples, como a deste projeto, por ter uma grande lista de suporte a APIs do *framework*, facilidade na instalação e no uso da sua estrutura.

Adiante, seguem as considerações finais desta dissertação e sugestões para trabalhos futuros, que podem agregar a este trabalho.

7 CONSIDERAÇÕES E SUGESTÕES PARA TRABALHOS FUTUROS

Esta dissertação se propõe a avaliar o uso de arcabouços abertos no desenvolvimento de serviços básicos de redes residenciais. Este uso faz um contraponto com o cenário que temos atualmente, onde as soluções existentes são proprietárias e fechadas, o que impossibilita, por parte dos desenvolvedores, a construção de serviços básicos para uso em qualquer aparelho doméstico.

Além da proposta citada, esta dissertação buscou desenvolver um pequeno protótipo para possibilitar a avaliação dos arcabouços. De acordo com as características da aplicação para redes residenciais e proposta para uma solução aberta, optou-se por utilizar o *Arduino*, como plataforma de *hardware* de código aberto para análise e criação dos serviços básico para casas inteligentes. Este protótipo possibilitou uma análise comparativa entre os arcabouços utilizados na construção da aplicação, no intuito de validar um *framework* adequado para o desenvolvimento de serviços.

As principais contribuições desta dissertação são como seguem:

a) a avaliação de *middlewares* para criação de serviços de redes, de forma a identificar o mais adequado, de acordo com os requisitos de aplicações de redes residenciais;
b) a construção de um protótipo para validar a utilização de arcabouços abertos na implementação de um serviço básico de rede residencial.

Na avaliação dos *middlewares* foram considerados os requisitos de modularidade, reusabilidade, integração de tecnologias e adaptabilidade. De acordo com o quadro comparativo da seção 3.4, o *framework* OSGi proporcionou melhores resultados, principalmente no que diz respeito à modularidade, além de ser uma solução muito praticada.

Quanto aos arcabouços abertos, observam-se três alternativas de ambientes de programação, baseados no OSGi, que serviram para a criação e teste do *bundles* de serviço de rede residencial, e, nestes, é possível concluir o seguinte:

a) o *framework Eclipse Equinox* foi avaliado considerando sua estrutura interna e curva de aprendizagem, que tiveram um ponto positivo porque a

IDE *Eclipse* já é baseada em OSGi. Assim, a construção de *bundles* de serviços tornou o processo simples. Para testar a aplicação, o *bundle* deve ser instalado e executado utilizando o console do *Eclipse*. Alguns erros apareceram, mas a aplicação funcionou;

b) o *framework Apache Felix* no quesito curva de aprendizagem teve um resultado similar ao *Equinox*, e por conta do processo de construção do arquivo *MANIFEST.MF* foi o menos eficiente. Houve a tentativa de utilizar o recurso *Maven*, que é o diferencial em relação aos outros arcabouços, por razão do *Felix* já possuí-lo na sua especificação, mas o êxito não foi obtido. O *Maven* não funcionou adequadamente, portanto a aplicação de teste não funcionou como esperado. O *bundle* foi instalado com sucesso, mas a aplicação não executou completamente. Este recurso poderia ser testado na aplicação de redes residenciais desta dissertação, para verificar o funcionamento, mas isto não foi efetivado e, portanto, não é possível determinar se a sua funcionalidade seria vantajosa;

c) o *framework Makewave Knopflerfish* apresentou melhores resultados, no que diz respeito a sua estrutura interna, serviços e APIs do OSGi disponíveis e curva de aprendizagem. Até mesmo o desenvolvedor iniciante conseguirá construir um serviço baseado no OSGi pela facilidade e agilidade no processo de criação. O ambiente *Desktop Knopflerfish* permite testar a aplicação mais rapidamente, uma vez que ele já disponibiliza o *bundle* para instalação em uma pasta sugestiva, necessitando apenas acessar a pasta e abrir o pacote que será testado. Assim, este *framework* parece ser o mais recomendado para esta aplicação, considerando os seus recursos disponíveis;

d) a escolha pelo uso do *Arduino* como alternativa de hardware se deu por ele possuir uma plataforma de código aberto, o que permite que qualquer outra placa seja desenvolvida utilizando como base os seus circuitos, possibilitando diversas funcionalidades. O *gateway* de serviços, como foi visto no capítulo 5, a nível lógico, pode ter a função de um *gateway* de acesso a rede de banda larga e de uma plataforma de serviços. Este último conceito foi utilizado neste trabalho e o *Arduino* respondeu como esperado. Como *gateway* de acesso a WAN, ele não atenderia, uma vez

que o *hardware* não está preparado para tal funcionalidade, a menos que se instale módulos adicionais à sua estrutura.

O estudo de caso desenvolvido teve o objetivo de mostrar a possibilidade de integrar serviços básicos de redes residenciais, utilizando arcabouços e *frameworks* abertos e tornando viável o uso por fabricantes de diversos aparelhos domésticos e dispositivos. A linguagem *Java* hoje é bastante rica em recursos, APIs, *frameworks* e em praticamente todos os sistemas computacionais temos aplicações que foram escritas nesta linguagem.

Com esta não tão nova abordagem, o arcabouço OSGi mostrou ser um *framework* estável, que ainda pode evoluir para aprimorar cada vez mais seus *players* para facilitar a construção de aplicações e, consequentemente, facilitar o manuseio destas pelos usuários finais, tornando estas mais inteligíveis e transparente ao usuário.

Como sugestão para trabalhos futuros, é possível seguir com as alternativas abaixo:

a) criar outros serviços utilizando outros protocolos de comunicação e integrar à solução já criada. Evoluir o serviço criado, que foi baseado no protocolo RS-232, utilizando alguns dos protocolos citados nesta dissertação como protocolos comuns na rede residencial, tais como *ZigBee*, *Bluetooth*, *Wireless*. Se o *Arduino* for utilizado, é possível acrescentar *shields* à sua estrutura e adaptar aos protocolos citados. E, neste caso específico, ele poderá complementar o cenário no sistema de redes residenciais, como o *gateway* de acesso a WAN, que é uma funcionalidade prevista no modelo do *Gateway* Residencial proposta pelo OSGi;
b) criar uma interface que permita o controle dos equipamentos utilizando a *Web*, através de provedores de serviço, implementando boas estratégias de segurança da informação, como forma de garantir integridade dos dados. É possível utilizar *WebServices*, transformando os pacotes criados no *framework* OSGi em serviços *Web*, e estes serviços podem ser acessados através da *Internet*;

c) usar simuladores de redes e de outros modelos de plataforma de *hardware*, baseando-se no código aberto, de forma a criar ambientes de simulação e testar os protocolos de comunicação dos diversos aparelhos domésticos. É possível estabelecer comparações entre ambientes de simulação e avaliar o que traz melhor desempenho;
d) construir uma aplicação mais robusta integrando todos os serviços e implementado-os em um servidor de aplicações. A aplicação que foi criada como protótipo do estudo de caso desta dissertação pode ser melhorada, criando menos classes e diminuindo o acoplamento com o *framework* OSGi;
e) utilizar o *Python* para criação de *bundles* OSGi, através do processo de integração da linguagem citada com o Java, conhecido como *Jython*, e testar se a solução que esta integração propõe é satisfatória para aplicações de redes residenciais.

REFERÊNCIAS

3GPP. **LTE.** 2008. Disponível em: <http://www.3gpp.org/technologies/keywords-acronyms/98-lte>. Acesso em: 19 out. 2015.

3GPP. **3GPP system standards heading into the 5G era.** 2015. Disponível em: <http://www.3gpp.org/news-events/3gpp-news/1614-sa_5g>. Acesso em: 19 out. 2015.

ALVES, Alexandre de Castro. **OSGi in Depth.** New York: Manning Publications, 2012. 394 p.

ARDUINO (Itália). **Arduino - Arduino BoardUno**. 2015. Disponível em: <https://www.arduino.cc/en/Main/ArduinoBoardUno>. Acesso em: 21 jul. 2015.

ARDUINO E CIA. **Arduino e Cia**. 2015. Disponível em: <http://www.arduinoecia.com.br/>. Acesso em: 26 jul. 2015.

ASHTON, Kevin. **That 'Internet of Things' Thing - RFID Journal.** 2009. Disponível em: <http://www.rfidjournal.com/articles/view?4986>. Acesso em: 30 out. 2015.

BARRETT, Diane; KING, Todd. **Redes de Computadores.** Rio de Janeiro: Ltc, 2010. 478 p. Tradução Daniel Vieira.

BARTLETT, Neil. **OSGi In Practice.** [S.l]: [s.n.], 2009. Livro disponibilizado pelo autor no seu site. Disponível em: <http://njbartlett.name/files/osgibook_preview_20091217.pdf>. Acesso em: 25 ago. 2015.

BREIER, Guilherme Petry; PEREIRA, Carlos Eduardo. FEMTO-OSGi: Architecture for reconfigurable embedded systems to service management by OSGi.In: IEEE INTERNATIONAL CONFERENCE ON INDUSTRIAL TECHNOLOGY, 2011. **Procd….** 2011. DOI: 10.1109/icit.2011.5754353.

CHÁVEZ, Héctor Zenil. JavaServer Faces: An Excellent Learning Tool. **Ieee Distributed Systems Online,** v. 2, n. 6, p.1-2, fev. 2005.

CHUNG, Kyung-yong. Recent trends on convergence and ubiquitous computing. **Pers Ubiquit Comput,** v. 18, n. 6, p.1291-1293, 2 nov. 2013. DOI: 10.1007/s00779-013-0743-2.

COGOLUÈGNES, Arnaud; TEMPLIER, Thierry; PIPER, Andy. **Spring Dynamic Modules in Action.** Tradução de Álvaro Strube de Lima. Stamford: Manning Publications, 2011. 543 p.

COMER, Douglas E.. **Redes de computadores e internet.** 4. ed. Porto Alegre: Bookman, 2007. 640 p..

CRUZ, Andry B.; BOUTALEB, Tuleen; TIANFIELD, Huaglory. Simulation of service-oriented systems for Mobile Ad hoc Networks. **Simulation Modelling Practice And Theory,** v. 32, p.42-63, mar. 2013. Elsevier BV. DOI: 10.1016/j.simpat.2012.11.010.

Disponível em: <http://api.elsevier.com/content/article/PII:S1569190X12001657?httpAccept=text/x>. Acesso em: 29 ago. 2015.

CRUZ, Tiago et al. An Architecture for Virtualized Home Gateways. In: IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT (IM2013):MINI-CONFERENCE**,** Ghent, 2013. **Procd….** 2013. p.520-526.

CUMMINS, Holly; WARD, Timothy. **Enterprise OSGi in Action.** Shelter Island: Manning Publications, 2013. 400 p.

DEITEL, Harvey M.; DEITEL, Paul J.. **Java:** como programar. Tradução de Edson Furmankiewicz, revisão técnica de Fábio Lucchini. 6. ed. São Paulo: Pearson Prentice Hall, 2005. 1110 p.

DOU, Niu et al. The Networking Technology within Smart Home System - ZigBee Technology. **2009 International Forum On Computer Science-technology And Applications,** p.29-33, 2009. DOI: 10.1109/ifcsta.2009.129.

EVANS, Martin; NOBLIN, Joshua; HOCHENBAUM, Jordan. **Arduino in Action.** New York: Manning Publications, 2013.

FALUDI, Robert. **Building Wireless Sensor Networks.** Sebastopol: O'reilly, 2011.

FRIEDEWALD, Michael; RAABE, Oliver. Ubiquitous computing: An overview of technology impacts. **Telematics And Informatics,** v. 28, n. 2, p.55-65, maio 2011. Elsevier BV. DOI: 10.1016/j.tele.2010.09.001. Disponível em: <http://api.elsevier.com/content/article/PII:S0736585310000547?httpAccept=text/xml>. Acesso em: 15 out. 2015.

FUTURE TECHNOLOGY DEVICES INTERNATIONAL LTD. **FTDI Chip Home Page.** 2015. Disponível em: <http://www.ftdichip.com/>. Acesso em: 30 out. 2015.

GÉDÉON, Walid. **OSGi and Apache Felix 3.0:** Build your very own OSGi applications using the flexible and powerful Felix Framework. Birmingham: Packt Publishing, 2010.

GOMEZ, Carles; PARADELLS, Josep. Wireless home automation networks: A survey of architectures and technologies. **Ieee Commun. Mag.,** v. 48, n. 6, p.92-101, jun. 2010. DOI: 10.1109/mcom.2010.5473869.

HALL, Richard S. et al. **OSGi in Action:** Creating Modular Applications in Java. Stamford: Manning Publications, 2011. 576 p.

HOME GATEWAY INITIATIVE. **Www.homegatewayinitiative.org.** 2014a. Disponível em: <http://www.homegatewayinitiative.org/>. Acesso em: 20 ago. 2015.

HOME GATEWAY INITIATIVE. **HG REQUIREMENTS FOR HGI OPEN PLATFORM 2.0.** 2014b. Disponível em: <http://www.hoMegatewayinitiative.org/publis/HGI-RD048-HG_Requirements_for_HGI_Open_Platform_2_0_published_text.pdf>. Acesso em: 22 ago. 2015.

HOME GATEWAY INITIATIVE. **Home Gateway Technical Requirements: Residential Profile.** 2008. Disponível em: <http://www.hoMegatewayinitiative.org/publis/RD-001-R2-01_HG-Tech-Req-Residential_V1-01.pdf>. Acesso em: 22 ago. 2015.

IBRAHIM, Noha. Orthogonal Classification of Middleware Technologies. INTERNATIONAL CONFERENCE ON MOBILE UBIQUITOUS COMPUTING, SYSTEMS, SERVICES AND TECHNOLOGIES**,** 3., 2009. **Proced….** 2009. p.46-51. DOI: 10.1109/ubicomm.2009.24

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE STANDARDS 802.15.4**: IEEE Standard for Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements. Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs). Nova Iorque: Institute Of Electrical And Electronics Engineers, 2003.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE Standard for Information technology-- Local and metropolitan area networks-- Specific requirements-- Part 11**: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 8. S.l.**,** [s.l.], v. , n. , p.1-433, 2011. Institute of Electrical & Electronics Engineers (IEEE). DOI: 10.1109/ieeestd.2011.5716530.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE 802.15.1:** IEEE 802.15 WPAN Task Group 1 (TG1). 2004. Disponível em: <http://www.ieee802.org/15/pub/TG1.html>. Acesso em: 19 out. 2015.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION AND INTERNATIONAL ELECTROTECHNICAL COMMISSION. **7498-1**: Information Technology - Open Systems Interconnection - Basic Reference Model: The Basic Model. Genève: Iso/iec, 1994. 68 p.

INTERNET ENGINNERING TASK FORCE. **RFC 2616**: Hypertext Transfer Protocol -- HTTP/1.1. S.l.: S.n., 1999. 152 p.

JAVA COMMUNITY PROCESS. **The Java Community Process (SM) Program - JSRs:** Java Specification Requests - detail JSR#232. 2012. Oracle Corporation. Disponível em: <https://jcp.org/en/jsr/detail?id=232>. Acesso em: 25 ago. 2015.

JAVA COMMUNITY PROCESS. **The Java Community Process (SM) Program - JSRs:** Java Specification Requests - detail JSR#277. 2006a. Oracle Corporation. Disponível em: <https://jcp.org/en/jsr/detail?id=277>. Acesso em: 25 ago. 2015.

JAVA COMMUNITY PROCESS. **The Java Community Process (SM) Program - JSRs:** Java Specification Requests - detail JSR#291. 2006b. Oracle Corporation. Disponível em: <https://jcp.org/en/jsr/detail?id=291>. Acesso em: 25 ago. 2015.

JAVA COMMUNITY PROCESS. **The Java Community Process (SM) Program - JSRs:** Java Specification Requests - detail JSR#294. 2007a. Oracle Corporation. Disponível em: <https://jcp.org/en/jsr/detail?id=294>. Acesso em: 25 ago. 2015.

JAVA COMMUNITY PROCESS. **The Java Community Process (SM) Program - JSRs:** Java Specification Requests - detail JSR#220**.** 2007b. Oracle Corporation. Disponível em: <https://jcp.org/en/jsr/detail?id=220>. Acesso em: 30 out. 2015.

JAVA COMMUNITY PROCESS. **The Java Community Process(SM) Program.** 2014a. Oracle Corporation. Disponível em: <https://www.jcp.org/en/home/index>. Acesso em: 25 ago. 2015.

JAVA COMMUNITY PROCESS. **The Java Community Process(SM) Program - JSRs**: Java Specification Requests - detail JSR#346 2014b. Oracle Corporation. Disponível em: <https://jcp.org/en/jsr/detail?id=346>. Acesso em: 30 out. 2015.

JAVA COMMUNITY PROCESS. **The Java Community Process (SM) Program - JSRs:** Java Specification Requests - detail JSR#314. 2010. Oracle Corporation. Disponível em: <https://jcp.org/en/jsr/detail?id=314>. Acesso em: 28 ago. 2015.

JUNWU, Xu; JUNLING, Liang. Developing CRM System of Web Application Based on JavaServer Faces. In: INTERNATIONAL WORKSHOP ON EDUCATION TECHNOLOGY AND COMPUTER SCIENCE, 2., 2010. **Proced. …** 2010. DOI: 10.1109/etcs.2010.83.

KAY, Russel. ZigBee. **Computer World: Knowledge Center Mobile and Wireless,** S.l., p.46-46, maio 2006.

KING, Gavin et al. **JSR-299:** o novo padrão Java para injeção de dependência e gerenciamento de estado contextual.Tradução para o Italiano: Nicola Benaglia, Francesco Milesi; Tradução para o Espanhol: Gladys Guerrero; Tradução para o Coreano: Eun-Ju Ki; Tradução para o Chinês Tradicional: Terry Chuang. Disponível em: <https://docs.jboss.org/weld/reference/1.1.5.Final/pt-BR/pdf/weld-reference.pdf>. Acesso em: 27 ago. 2015.

KONINKLIJKE PHILIPS. **Philips Brasil.** 2015. Disponível em: <http://www.philips.com.br/>. Acesso em: 10 out. 2015.

KUROSE, James F.; ROSS, Keith W.. **Redes de Computadores e a Internet:** uma abordagem top-down. 5. ed. São Paulo: Addison Wesley, 2010. Tradução de Opportunity Translations do original Computer Networking 5th ed a top-down approach featuring the Internet.

KUSHALNAGAR, N.; MONTENEGRO, G.; SCHUMACHER, C.. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. **RFC 4919**, v. 1, n. 1, p.1-12, ago. 2007. RFC Editor. DOI: 10.17487/rfc4919.

LEE, Jonathan; LEE, Shin-jie; WANG, Ping-feng. A Framework for Composing SOAP, Non-SOAP and Non-Web Services. **Ieee Transactions On Services Computing,** [s.l.], v. 8, n. 2, p.240-250, mar. 2015. Institute of Electrical & Electronics Engineers (IEEE). DOI: 10.1109/tsc.2014.2310213.

LG. **The Official Site of LG Group.** 2015. Disponível em: <http://www.lgcorp.com/main/main.dev>. Acesso em: 10 out. 2015.

LI, Qian Clara et al. 5G Network Capacity**:** Key Elements and Technologies. **Ieee Veh. Technol. Mag**. v. 9, n. 1, p.71-78, mar. 2014. Institute of Electrical & Electronics Engineers (IEEE). DOI: 10.1109/mvt.2013.2295070.

LINUX FOUNDATION. **Linux.com | The source for Linux Information.** 2015. Disponível em: <https://www.linux.com/>. Acesso em: 30 out. 2015.

LYNGGAARD, Per; SKOUBY, Knud Erik. Deploying 5G-Technologies in Smart City and Smart Home Wireless Sensor Networks with Interferences. **Wireless Pers Commun,** v. 81, n. 4, p.1399-1413, 15 mar. 2015. Springer Science + Business Media. DOI: 10.1007/s11277-015-2480-5.

MAKEWAVE AB. **Makewave.** 2015. Disponível em: <http://www.makewave.com/site.en/news/docs/old/gstm.pressrelease-ubiserv_en-md-004-B.shtml>. Acesso em: 15 ago. 2015.

MANN, Kito D.. **JavaServer Faces in Action.** Greenwich: Manning, 2005. 1073 p.

MCAFFER, Jeff; LEMIEUX, Jean-michel; ANISZCZYK, Chris. **Eclipse Rich Client Platform.** 2. ed. Boston: Pearson Education, 2010. 553 p.

MCAFFER, Jeff; VANDERLEI, Paul; ARCHER, Simon. **OSGi and Equinox:** Creating Highly Modular Java Systems. Crawfordsville: Pearson Education, 2010.

MCROBERTS, Michael. **Arduino básico.** São Paulo: Novatec, 2011. Rafael Zanolli.

MEDAGLIANI, P.; MARTALÒ, M.; FERRARI, G.. Clustered Zigbee networks with data fusion: Characterization and performance analysis. **Ad Hoc Networks,** v. 9, n. 7, p.1083-1103, set. 2011. Elsevier BV. DOI: 10.1016/j.adhoc.2010.10.009. Disponível em: <http://api.elsevier.com/content/article/PII:S1570870510001721?httpAccept=text/xml>. Acesso em: 14 out. 2015.

MICROSOFT. **Windows e Windows 10 - Microsoft.** 2015. Disponível em: <https://www.microsoft.com/pt-br/windows/>. Acesso em: 30 out. 2015.

MONTPETIT, Marie-jose. **Home Networking Standards.** Las Vegas: Sonoro-vídeo, 2007. Color. Consumer Communications and Networking Conference, Sponsored by: IEEE Communications Society.

MORAES, Alexandre Fernandes de. **Redes de computadores:** fundamentos. 7. ed. São Paulo: Érica, 2010. 256 p.

MUDRIIEVSKYI, Stanislav. Power Line Communications: State of the art in research, development and application. **Aeu - International Journal Of Electronics And Communications,** [v. 68, n. 7, p.575-577, jul. 2014. DOI: 10.1016/j.aeue.2014.04.003.

OBAIDAT, Mohammad S. et al (Ed.). **Pervasive Computing and Networking.** Nova Delhi: Wiley, 2011

OPENHAB UG. **Home . openhab/openhab Wiki . GitHub.** 2015. Disponível em: <https://github.com/openhab/openhab/wiki>. Acesso em: 12 set. 2015.

OPENREMOTE. **OpenRemote | Open Source for Internet of Things**. 2015a. Disponível em: <http://www.openremote.com/>. Acesso em: 16 out. 2015.

OPENREMOTE. **OpenRemote | Home of the Digital Home.** 2015b. Disponível em: <http://www.openremote.org/display/HOME/OpenRemote>. Acesso em: 16 out. 2015.

ORACLE CORPORATION. **Java EE 6 APIs - The Java EE 6 Tutorial.** 2013. Disponível em: <http://docs.oracle.com/javaee/6/tutorial/doc/bnacj.html#bnacp>. Acesso em: 27 ago. 2015.

ORACLE CORPORATION. **JavaServer Faces Technology Documentation.** 2010. Disponível em: < http://www.oracle.com/technetwork/java/javaee/documentation/index-137726.html >. Acesso em: 28 ago. 2015.

ORACLE CORPORATION. **Java Software | Oracle**. 2015a. Disponível em: <https://www.oracle.com/java/index.html>. Acesso em: 01 out. 2015.

ORACLE CORPORATION. **OpenJDK:** Project Jigsaw**.** 2015b. Disponível em: <http://openjdk.java.net/projects/jigsaw/>. Acesso em: 25 out. 2015.

ORACLE CORPORATION. **JEP 200:** The Modular JDK. 2015c. Disponível em: <http://openjdk.java.net/jeps/200>. Acesso em: 25 out. 2015.

ORACLE CORPORATION. **JEP 201:** Modular Source Code. 2015d. Disponível em: <http://openjdk.java.net/jeps/201>. Acesso em: 25 out. 2015.

ORACLE CORPORATION. **JEP 220:** Modular Run-time Images. 2015e. Disponível em: <http://openjdk.java.net/jeps/220>. Acesso em: 25 out. 2015.

OSGI. **OSGi Alliance | Main / OSGi Alliance.** 2015a. Disponível em: <http://www.osgi.org>. Acesso em: 04 jul. 2015.

OSGI. **OSGi Alliance | JSR232 / Mobile Operational Management.** 2015b**.** Disponível em: <http://www.osgi.org/JSR232/HomePage>. Acesso em: 25 ago. 2015.

OSGI. **OSGi Alliance | JSR291 / Dynamic Component Support for Java SE.** 2015c. Disponível em: <http://www.osgi.org/JSR291/HomePage>. Acesso em: 25 ago. 2015.

OSGI. **OSGi Service Gateway Specification.** 2000. Disponível em: <https://osgi.org/download/r1/r1.osgi-spec.pdf>. Acesso em: 24 ago. 2015.

OSGI. **OSGi Service Platform Release 2.** 2001. Disponível em: <https://osgi.org/download/r2/sp-r2.book.pdf>. Acesso em: 24 ago. 2015.

OSGI. **OSGi Service Platform Release 3.** 2003. Disponível em: <https://osgi.org/download/r3/r3.book.pdf>. Acesso em: 24 ago. 2015.

OSGI. **OSGi Service Platform Core Specification Release 4.** 2012a. Versão 4.3. Disponível em: <https://osgi.org/download/r4v43/osgi.core-4.3.0.pdf>. Acesso em: 24 ago. 2015.

OSGI. **OSGi Core Release 5.** 2012b. Disponível em: <https://osgi.org/download/r5/osgi.core-5.0.0.pdf>. Acesso em: 24 ago. 2012.

OSGI. **OSGi Core Release 6.** 2014. Disponível em: <https://osgi.org/download/r6/osgi.core-6.0.0.pdf>. Acesso em: 24 ago. 2015.

OW2 CONSORTIUM. **OW2 Forge:** Project Info- Oscar - OSGi Framework. 2005. Disponível em: <http://forge.ow2.org/projects/oscar/>. Acesso em: 28 out. 2015.

PHAM, Dung Vu; SYED, Ali; HALGAMUGE, Malka N.. Universal serial bus based software attacks and protection solutions. **Digital Investigation**, v. 7, n. 3-4, p.172-184, abr. 2011. Elsevier BV. DOI: 10.1016/j.diin.2011.02.001. Disponível em: <http://api.elsevier.com/content/article/PII:S1742287611000041?httpAccept=text/xml>. Acesso em: 19 set. 2015.

PYTHON SOFTWARE FOUNDATION. **Welcome to Python.org**. 2015. Disponível em: <https://www.python.org/>. Acesso em: 1 out. 2015.

RAMYA, C. Muthu; SHANMUGARAJ, M; PRABAKARAN, R. Study on ZigBee technology. In: INTERNATIONAL CONFERENCE ON ELECTRONICS COMPUTER TECHNOLOGY, 3., 2011. **Procd.…** 2011. p.297-301. DOI: 10.1109/icectech.2011.5942102.

RED HAT INC. **Chapter 5. Scopes and Contexts.** 2015. Disponível em: <https://docs.jboss.org/weld/reference/latest/en-US/html/scopescontexts.html>. Acesso em: 27 ago. 2015.

RITCHIE, Dennis M.. The development of the C language. **Acm Sigplan Notices,** v. 28, n. 3, p.201-208, 1 mar. 1993. DOI: 10.1145/155360.155580.

RUBY. **Ruby Programming Language.** 2015. Disponível em: <https://www.ruby-lang.org/en/>. Acesso em: 30 out. 2015.

SAMSUNG. **Samsung | Celulares | Smartphones | Televisores | Eletrônicos | Informática | Notebooks | Impressoras | Áudio | Video | Fotografia | Eletrodomésticos | Acessórios.** 2015. Disponível em: <http://www.samsung.com/br/home/>. Acesso em: 10 out. 2015.

TANENBAUM, Andrew S.. **Redes de computadores.** 4. ed. Rio de Janeiro: Campus, 2003. Tradução de Vandenberg D. de Souza do livro Computer Networks, 4th ed.

THE APACHE SOFTWARE FOUNDATION. **Maven - Welcome to Apache Maven.** 2015a. Disponível em: <http://maven.apache.org/>. Acesso em: 27 ago. 2015.

THE APACHE SOFTWARE FOUNDATION. **Apache Felix - Apache Felix UPnP.** 2015b. Disponível em: <http://felix.apache.org/documentation/subprojects/apache-felix-upnp.html>. Acesso em: 16 out. 2015.

THE APACHE SOFTWARE FOUNDATION. **Apache Felix - Welcome to Apache Felix.** 2015c. Disponível em: <http://felix.apache.org/>. Acesso em: 26 out. 2015.

THE APACHE SOFTWARE FOUNDATION. **Apache Felix - Apache Felix Framework Usage Documentation.** 2013. Disponível em: <http://felix.apache.org/documentation/subprojects/apache-felix-framework/apache-felix-framework-usage-documentation.html>. Acesso em: 30 out. 2015.

THE ECLIPSE FOUNDATION. **Equinox**. 2015a. Disponível em: <http://www.eclipse.org/equinox/>. Acesso em: 10 jul. 2015.

THE ECLIPSE FOUNDATION. **Equinox Framework.** 2015b. Disponível em: <http://www.eclipse.org/equinox/framework/>. Acesso em: 17 ago. 2015.

THE ECLIPSE FOUNDATION. **Concierge**. 2015c. Disponível em: <http://www.eclipse.org/proposals/rt.concierge/>. Acesso em: 09 out. 2015.

THE ECLIPSE FOUNDATION. **Mars Eclipse.** 2015d. Disponível em: <http://www.eclipse.org>. Acesso em: 27 out. 2015.

THE ECLIPSE FOUNDATION. **Equinox Framework QuickStart Guide.** 2015e. Disponível em: <http://www.eclipse.org/equinox/documents/quickstart-framework.php>. Acesso em: 30 out. 2015.

THE ECLIPSE FOUNDATION. **PDE.** 2015f. Disponível em: <http://www.eclipse.org/pde/>. Acesso em: 30 out. 2015.

THE ECLIPSE FOUNDATION. **Equinox Bundles.** 2015g. Disponível em: <http://www.eclipse.org/equinox/bundles/>. Acesso em: 30 out. 2015.

THE KNOPFLERFISH PROJECT. **Knopflerfish OSGi - open source OSGi service platform. OSGi R5, R4 and R3.** 2015a. Disponível em: <http://www.knopflerfish.org/>. Acesso em: 30 jul. 2015.

THE KNOPFLERFISH PROJECT. **Knopflerfish OSGi, version 5.2.0 - What is included in the Knopflerfish distribution.** 2015b. Disponível em: <http://www.knopflerfish.org/releases/current/docs/components.html>. Acesso em: 18 ago. 2015.

THE KNOPFLERFISH PROJECT. **Knopflerfish - OSGi R5.** 2015c. Disponível em: <http://www.knopflerfish.org/kf5_osgi_r5.html>. Acesso em: 18 ago. 2015.

THE KNOPFLERFISH PROJECT. **Knopflerfish OSGi, version 5.2.0 - Running Knopflerfish.** 2015d. Disponível em:

<http://www.knopflerfish.org/releases/current/docs/running.html>. Acesso em: 30 out. 2015.

UNIVERSAL SERIAL BUS. **Universal serial bus specification revision 2.0**: Universal Serial Bus Specification. [s.l.]: [s.n], 2000

VASILIEV, Yuli. **SOA and WS-BPEL:** composing service-oriented solutions with PHP and ActiveBPEL. Birmingham: Packt Publishing, 2007. 313 p.

VILAS, Ana Fernández et al. Context-aware personalization services for a residential gateway based on the OSGi platform. **Expert Systems With Applications,** v. 37, n. 9, p.6538-6546, set. 2010. Elsevier BV. DOI: 10.1016/j.eswa.2010.02.132. Disponível em: <http://api.elsevier.com/content/article/PII:S0957417410001715?httpAccept=text/xml>. Acesso em: 29 ago. 2015.

W3C WORKING GROUPS. **W3C XML Schema Definition Language (XSD) 1.1 Part1:** Structures. 2012. Disponível em: <http://www.w3.org/TR/xmlschema11-1/>. Acesso em: 17 out. 2015.

WANG, Lan; QU, Wenyu. Advances in ubiquitous computing and communications. **Future Generation Computer Systems,** v. 38, p.11-12, set. 2014. Elsevier BV. DOI: 10.1016/j.future.2014.04.005. Disponível em: <http://api.elsevier.com/content/article/PII:S0167739X14000806?httpAccept=text/xml>. Acesso em: 15 out. 2015.

WEI, Zhiqiang et al. A residential gateway architecture based on Cloud computing. In: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND SERVICE SCIENCES, 2010. **Procd.…** 2010. p.245-248. DOI: 10.1109/icsess.2010.5552422.

WEISER, M.. The computer for the 21st Century. **Scientific American,** v. 1, n. 1, p.94-10, set. 1991.Disponível em: <http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html>. Acesso em: 29 jun. 2015.

WEISER, M.. The world is not a desktop. **Interactions,** v. 1, n. 1, p.7-8, 2 jan. 1994. Association for Computing Machinery (ACM).

WISTRAND, Erik. **Knopflerfish OSGi Tutorial Version 2:** A Step by Step Introduction to OSGi Programming Based on the open source Knopflerfish OSGi Framework. 2009. Texto original de Sven Haiges (2004). Disponível em: <http://www.knopflerfish.org/releases/current/docs/tutorials/kf_osgi_tutorial/kf_osgi_tutorial.pdf>. Acesso em: 30 jul. 2015.

YAMAZAKI, Tatsuya. The Ubiquitous Home. **International Journal Of Smart Home,** Daejeon, v. 1, n. 1, p.17-22, jan. 2007.

YU, Yuan-chih; YOU, Shing-chern D.; TSAI, Dwen-ren. An intelligent Aspect-Oriented Framework for Web Application. In: INTERNATIONAL CONFERENCE ON NETWORKED COMPUTING (INC), 6., 2010. **Procd.…** 2010. p.1-5, maio 2010. IEEE.

ZAHARIADIS, Theodore B.. **Home Networking Technologies and Standards.** Londres: Artech, 2003.

APÊNDICE A – ARCABOUÇO *ECLIPSE EQUINOX*: ASPECTOS DE INSTALAÇÃO, ETAPAS DE CRIAÇÃO E EXECUÇÃO DO *BUNDLE* DE TESTE E INSTALAÇÃO DO *BUNDLE* NO INTERPRETADOR DE COMANDOS

Para utilizar as APIs da especificação R4 do OSGi com o *Eclipse Equinox* deve ser feita a descompactação de arquivos necessários para seu funcionamento. Os arquivos descompactados podem ser salvos em uma pasta sugestiva, que, para facilitar a localização e organização, pode ser chamada de *Equinox*.

Desta forma, a estrutura do comando para acessar o console ou interpretador de comandos (*shell*) do arcabouço *Equinox* é como segue (ECLIPSE, 2015e):

Unidade:\equinox > java -jar (framework) -console

Considerando a unidade principal **C:** e o *framework* o arquivo *org.eclipse.osgi_3.2.0.jar*, utilizando a instrução acima, o comando para executar o console do *framework* do *Eclipse Equinox*, para que seja possível carregar as APIs do OSGi é como segue:

C:\equinox > java -jar org.eclipse.osgi_3.2.0.jar -console

Para programar os *bundles*, realiza-se a instalação de uma IDE, neste trabalho foi escolhido o *Eclipse*, por também ter como base o *framework* OSGi. Para a criação de *bundles*, o *Eclipse PDE* (*Plugin Development Environment* – Ambiente de Desenvolvimento de *Plugins*) (ECLIPSE, 2015f) é o mais adequado, já que neste é possível carregar as APIs e bibliotecas do OSGi mais facilmente. É possível também criar os *bundles* em outras versões do *Eclipse*, como o *Mars*, o *Luna* ou qualquer outro.

É possível efetuar o *download* do *Eclipse* PDE na página oficial do *Eclipse* (ECLIPSE, 2015d) e seguir com os passos de instalação que são sugeridos ao executar seu arquivo de instalação.

Outros arcabouços podem utilizar o IDE *Eclipse* para programar os *bundles* de serviço. Por esta razão, é necessário configurar o ambiente para que as APIs e bibliotecas do OSGi estejam disponíveis.

No *Eclipse* PDE, este procedimento pode ser feito da seguinte forma:

a) selecionar o menu *Preferences* em *Window*;
b) selecionar a opção *Plug-in Development*;
c) selecionar a opção OSGi *Frameworks* e escolher o *Equinox* como *framework* padrão.

Após este processo, o *Eclipse Equinox* pode ser utilizado para construção de *bundles* de serviço. Este processo não precisou de muitos detalhes para ser efetivado devido a IDE já estar preparada para criação de pacotes OSGi.

Para criar o *bundle* Teste foram efetivadas as seguintes etapas no *Eclipse* PDE:

a) seleciona-se um Novo Projeto (***New Project***);
b) na caixa de Diálogo de Projeto, seleciona-se o tipo ***Plug-in Project*** e na caixa seguinte são informados o nome do projeto e a plataforma. A plataforma escolhida é ***OSGi Framework Standard***, para permitir que o *bundle* utilize todas as APIs possíveis do *framework* OSGi;
c) os valores padrão são mantidos até a conclusão da configuração.

Para executar o *bundle* de teste, seleciona-se a opção ***Run*** e a aplicação carregará a interface e executará os métodos de acordo com o que foi programado. Se a aplicação for executada pela primeira vez, o *Eclipse* solicita a seleção de opção de execução da aplicação. Entre as alternativas estão o ***Eclipse Application***, ***Java Applet***, ***Java Application*** e ***OSGi Framework***, devendo ser selecionada esta última.

Para efetuar a instalação do *bundle* no interpretador de comandos do *framework*, entra-se com o comando interno ***install***. A sintaxe do comando é como segue abaixo:

Install file:///<caminho/do/arquivo>/<arquivo>.jar

Após a instalação, o *framework* informa o número do ID do *bundle*, que será útil para controlar os seus estados.

APÊNDICE B – ARCABOUÇO APACHE FELIX: ASPECTOS DE INSTALAÇÃO, ETAPAS DE CRIAÇÃO E EXECUÇÃO DO *BUNDLE* DE TESTE E INSTALAÇÃO DO *BUNDLE* NO INTERPRETADOR DE COMANDOS

Para instalar o *framework*, efetua-se o *download* do arquivo que contém os *bundles* e serviços compactados, descompacta-os em alguma pasta sugestiva e executa-se o comando para iniciar o *framework* (considerando que os arquivos descompactados estão na pasta *Felix*), conforme segue (APACHE, 2013):

Unidade:\Felix> java –jar bin/felix.jar

O executável do *framework* fica localizado no diretório *Bin*, após o processo de descompactação do arquivo.

Para construir o *bundle*, uma IDE deve ser utilizada e, neste trabalho, o uso do *Eclipse* foi mantido. Para a criação de *bundles* utilizando a especificação R6 do OSGi com o *Apache Felix*, as suas bibliotecas e APIs devem ser incorporadas ao IDE. Para que isto seja efetivado, os seguintes passos são seguidos:

a) selecionar o menu *Preferences* em *Window*;
b) selecionar as opções *Java*, em seguida *Build Path* e, logo após, *User Library*;
c) adicionar uma nova biblioteca. O nome a ser utilizado deve ser sugestivo (*Felix*, por exemplo) e, ao informar o caminho do *framework*, o diretório onde ele foi descompactado deve ser colocado, buscando o arquivo com a extensão JAR;

Após a construção do pacote e sua compilação, o console do *framework* foi acessado para possibilitar a instalação do *bundle* e a sua inicialização. O comando de instalação é como segue:

Install file:///<caminho/do/arquivo>/<arquivo>.jar

APÊNDICE C – ARCABOUÇO MAKEWAVE KNOPFLERFISH: ASPECTOS DE INSTALAÇÃO, ETAPAS DE CRIAÇÃO E EXECUÇÃO DO *BUNDLE* DE TESTE E INSTALAÇÃO DO *BUNDLE* NO INTERPRETADOR DE COMANDOS

Os procedimentos para instalação do arcabouço KF são apresentados a seguir, assim como o uso do seu console.

Para instalar o KF deve-se efetuar o *download* do *framework* (que estará compactado em um arquivo com extensão JAR), colocá-lo numa pasta sugestiva (podendo ser chamada de KF) e executar o comando a seguir, no *prompt* de comandos:

C:\KF> java -jar knopflerfish_osgi_sdk_<version>.jar

No comando acima, a palavra-chave <version> indica a versão do arquivo JAR do *Knopflerfish* que foi descarregado do seu site oficial.

Executando este comando, será aberto um auxiliar de instalação, que questionará em qual pasta e quais os componentes do *framework* (ambiente de execução, código-fonte e documentação) serão instalados. Depois de selecionar a pasta e instalar o KF, ele pode ser iniciado através do comando (KNOPFLERFISH, 2015d):

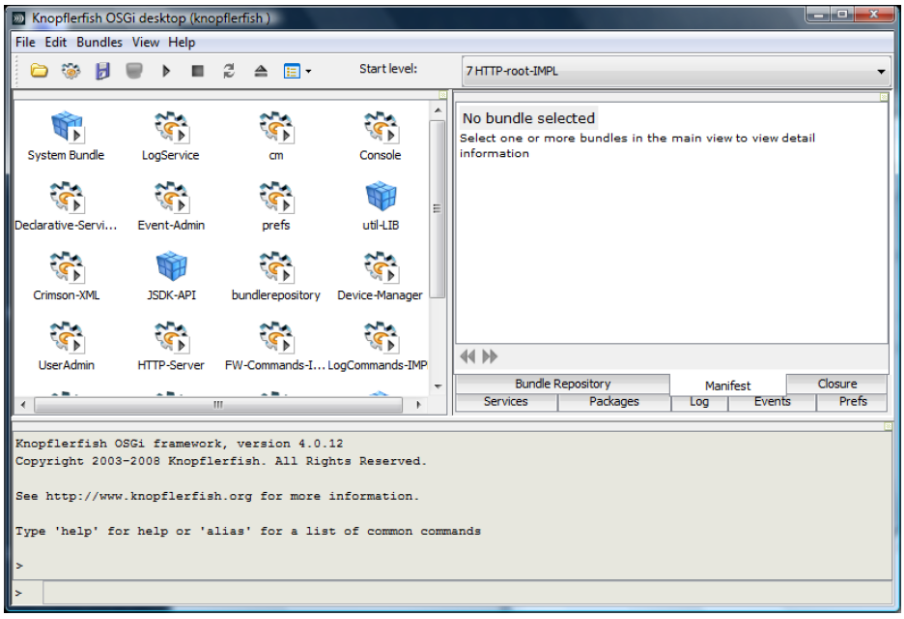
C:\Knopflerfish\osgi> java -jar framework.jar

O arquivo framework.jar está localizado na pasta OSGI, que fica no diretório principal do KF. Para efetuar a instalação do KF, uma das alternativas é fazê-la utilizando o próprio IDE, no *menu* ***Help*** e selecionando ***Install New Software***. Ao clicar no botão ***Adicionar***, pode-se colocar o endereço do site de atualização do KF:

http://www.knopflerfish.org/eclipseupdate/

Uma tela será aberta indicando que o KF está pronto para ser utilizado, de acordo com a figura abaixo.

Aplicação *Knopflerfish Desktop*



Fonte: Wistrand (2009).

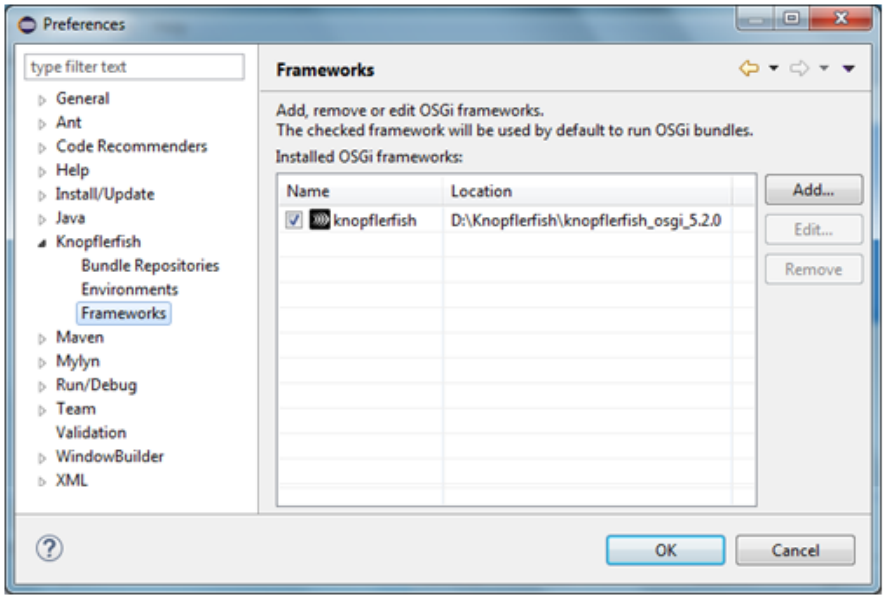
Este ambiente permite gerenciar o *framework Knopflerfish*, possibilitando a instalação, desinstalação, inicialização, interrupção e atualização de *bundles*, dentre outras ações.

A configuração das APIs dos OSGi pode ser feita selecionando as configurações de Preferências do projeto, no IDE, da seguinte forma:

a) selecionar a opção *Preferences* em *Window*;
b) selecionar a opção *Knopflerfish* e, em seguida, *Framework;*
c) adicionar o Knopflerfish, inserindo a pasta onde ele foi descompactado.

A demonstração desta configuração está como segue na figura abaixo

Configurando o *KF no IDE Eclipse*

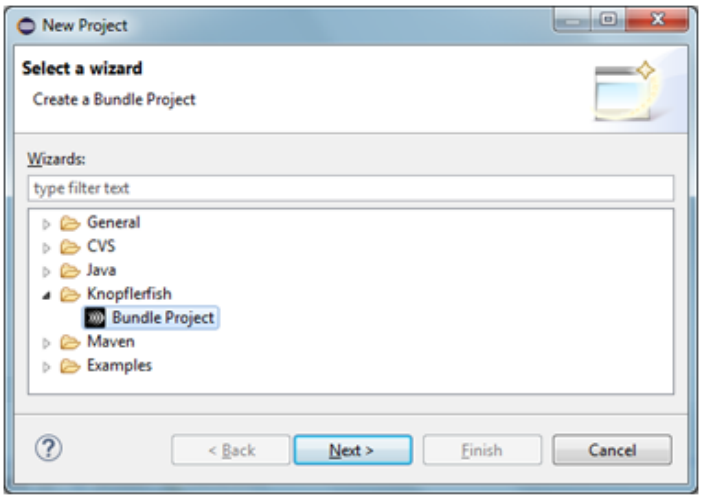


Fonte: Elaborada pela própria autora.

O processo de instalação e configuração do KF foi efetivado de maneira simples, apesar de possuir mais passos que os arcabouços anteriores, devido a presença do *Desktop*. Esta aplicação auxiliar parece facilitar a construção e testes dos pacotes de serviços OSGi, um ponto que pode agregar na criação de projetos maiores, em termos de facilidade e eficiência.

Para criar o exemplo de teste, cria-se um novo Projeto e seleciona-se a opção ***Bundle Project*** em *Knopflerfish*, de acordo com a figura abaixo.

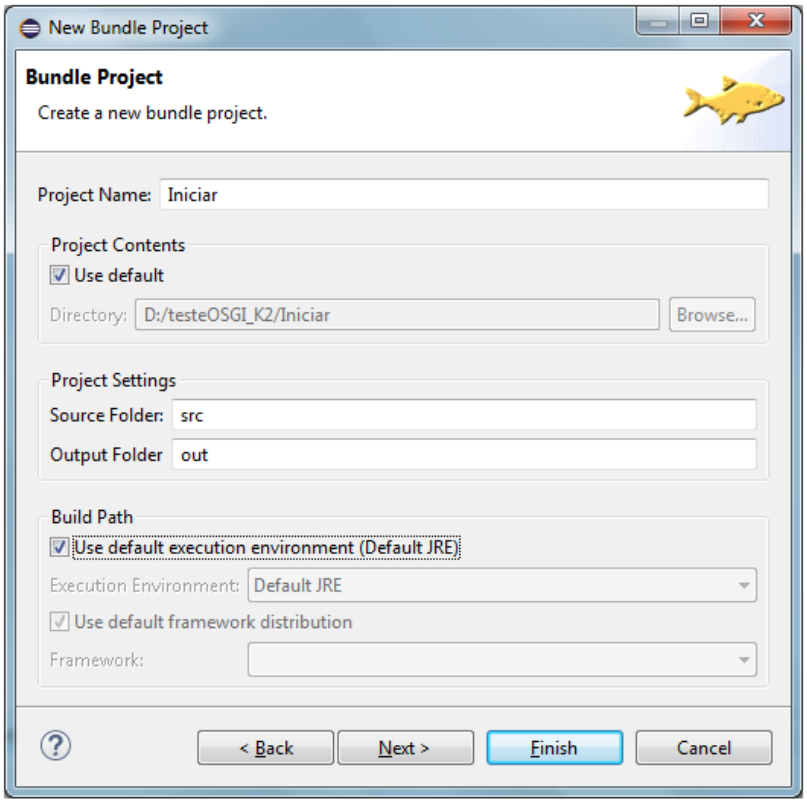
Criando um projeto *Bundle* usando KF



Fonte: Elaborada pela própria autora.

Em seguida, as informações acerca do projeto são preenchidas, mantendo as opções padrão já selecionadas. A figura abaixo mostra como ficará o exemplo para o projeto de teste.

Configurando o novo projeto

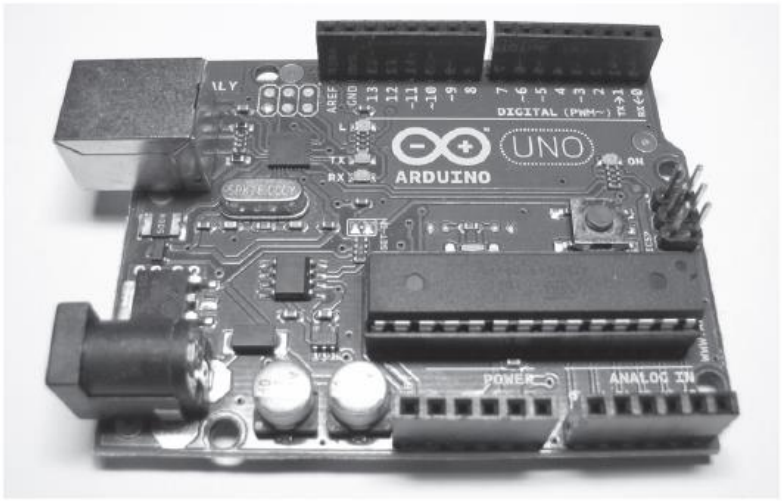


Fonte: Elaborada pela própria autora.

APÊNDICE D – ARDUINO: DETALHES DO *HARDWARE* E AMBIENTE DE PROGRAMAÇÃO

O *Arduino Uno* é uma plataforma de computação física ou embarcada baseada no microcontrolador *ATMega328* da família AVR da *ATmel Corporation*. A figura abaixo apresenta o modelo de seu *hardware* .

Arduino Uno Rev. 3



Fonte: Mcroberts (2011).

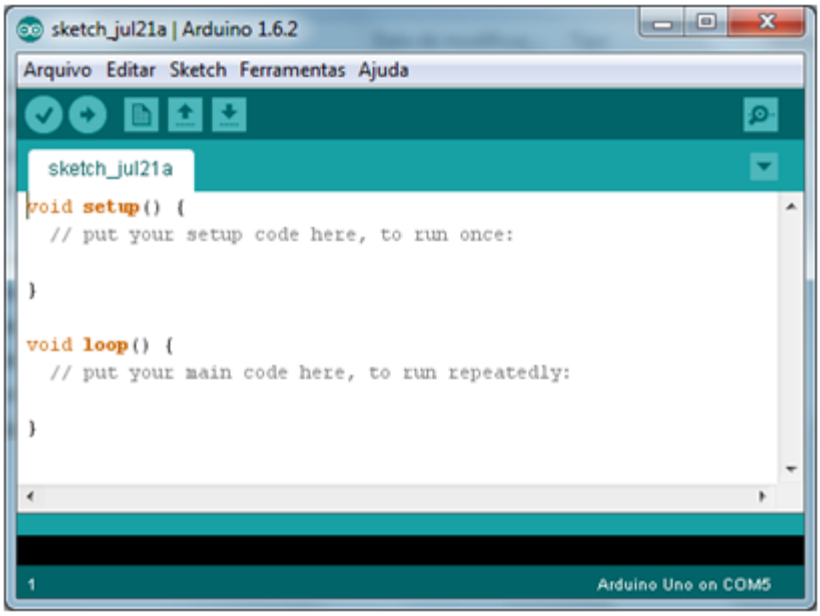
Esta plataforma é composta por (ARDUINO, 2015):

a) 14 pinos de entrada e saída digitais (dentre as quais seis podem ser usadas como saídas PWM - *Pulse-Width Modulation* – Modulação por Largura de Pulso), seis entradas analógicas, permitindo conectar a outros circuitos ou sensores;
b) uma conexão USB, que permite carregar códigos ou recuperar dados;
c) entrada de energia.

A capacidade de memória do *Arduino* é reduzida, mas suficiente para as operações que a placa pode realizar. Possui uma memória *Flash* de 32 *kilobytes*, dentre os quais 0,5 *kilobyte* é utilizado para *bootloader* (carregador de inicialização). Possui memórias SRAM de 2 *kilobytes* e EEPROM de 1 *kilobyte* de capacidade (ARDUINO, 2015).

Na programação das ações do *Arduino*, é necessário utilizar o seu próprio IDE para escrever os códigos (também conhecidos como *sketches* – rascunhos), que são baseados na linguagem C (MCROBERTS, 2011). Após a escrita do conjunto de instruções, estas são carregadas para o *Arduino*, para que ele possa interagir com o que estiver conectado a ele. A figura abaixo apresenta o ambiente de escrita das instruções para o *Arduino*.

Ambiente de Desenvolvimento de Códigos do *Arduino*



Fonte: Elaborada pela autora.

Ele inicia com as duas funções principais:

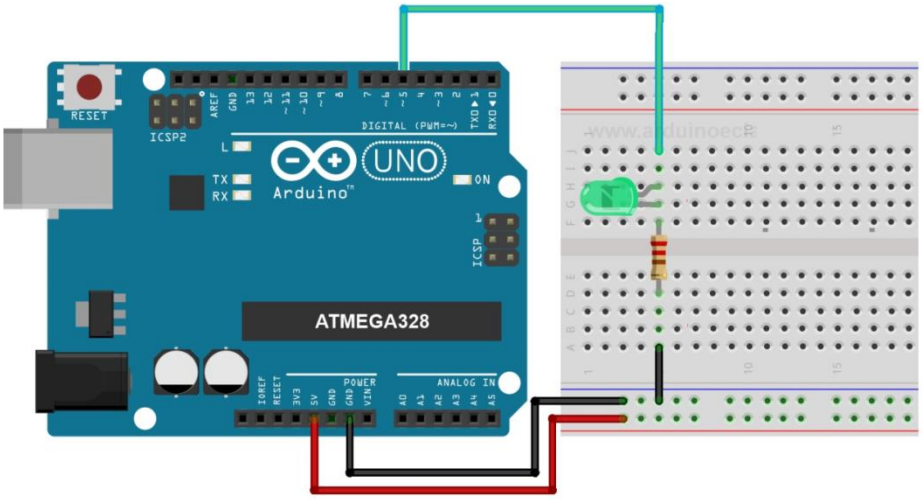
a) a função **void setup()**, que é executada apenas no início do programa e onde consta as instruções gerais de preparação do programa. Esta função não retorna nenhum dado;
b) a função **void loop()**, que executa todas as declarações continuamente, enquanto o *Arduino* estiver ligado.

Após a inserção do código, este pode ser analisado no próprio ambiente de desenvolvimento, para verificar a sintaxe e a existência de erros, acionando o botão de Verificar. Em seguida, ele já poderá ser carregado – acionando o botão ***Carregar*** – para a placa. O programa fica armazenado na memória do *Arduino* até que seja substituído (carregado) por outro conjunto de instruções.

APÊNDICE E – LIGAÇÃO DOS COMPONENTES NO PROTOBOARD PARA O PROTÓTIPO

A ligação dos componentes no *protoboard* está conforme a figura abaixo.

Ligação dos componentes na placa de circuito impresso



Fonte: Adaptado de Arduino e Cia (2015).

As especificações dos componentes utilizados no projeto são como seguem:

a) um LED que permita tensão máxima de 2,5 Volts;
b) um resistor de 1000 Ohms, utilizado para limitar a passagem de corrente pelo LED;
c) fios jumper, para efetuar a ligação do circuito na placa de circuito impresso e no Arduino.

APÊNDICE F – CÓDIGO FONTE DAS FUNÇÕES DO ARDUINO

Listagem 1 – Funções do Arduino – IDE Arduino

```
int ledPin =  5; //atribui o pino 5 a variável ledPin
int dado; //variável que receberá os dados da porta serial

void setup(){
	Serial.begin(9600);//frequência da porta serial
	pinMode(ledPin,OUTPUT); //define o pino o ledPin como saída
}

void loop(){
	if(Serial.available() > 0) { //verifica se existe comunicação com a porta serial
		dado = Serial.read();//lê os dados da porta serial
		switch(dado){
			case 1:
			digitalWrite(ledPin,HIGH); //liga o pino ledPin
			break;

		case 2:
			digitalWrite(ledPin,LOW); //desliga o pino ledPin
			break;
		}
	}
}
```

APÊNDICE G – CÓDIGO FONTE DAS CLASSES DO PACOTE PROJETO.LEDSERVICE.SERVICE

Listagem 2 – Trecho do Código da classe Projeto.LEDService.Service.Arduino.java

```
public class Arduino implements Serializable{
  //...

  public void comunicacaoArduino(JButton button) {
    if("Ligar".equals(button.getActionCommand())){
      arduino.eDados(1);
      System.out.println(button.getText());
    }
    else if("Desligar".equals(button.getActionCommand())){
      arduino.eDados(2);
      System.out.println(button.getText());
    }
    else{
      arduino.fechar();
      System.out.println(button.getText());
    }
  }
}
```

Listagem 3 - Trecho do código da Classe
Projeto.LEDService.Service.ComunicacaoSerial.java

```
public class ComunicacaoSerial implements Serializable {
//...

  private void iniciar() {
    try {

      CommPortIdentifier portId = null;
      try {

        portId = CommPortIdentifier.getPortIdentifier(this.portaCOM);
      }catch (NoSuchPortException npe) {

        JOptionPane.showMessageDialog(null, "Porta serial COM não encontrada.",
                  "Porta COM", JOptionPane.PLAIN_MESSAGE);
      }

/...

  public void fechar() {
    try {
        serialOut.close();
    }
// ...

  public void eDados(int opcao){
    try {
      serialOut.write(opcao);
    }
//...
```

Listagem 4 – Trecho do Código da Classe Projeto.LEDService.Service.ServObjImpl.java

```
public class ServObjImpl implements ServObj{

	// ....
		try{
			ByteArrayOutputStream byteOutput = new ByteArrayOutputStream();
			output = new ObjectOutputStream(byteOutput);
			output.writeObject(obj);
			//....
		}

		catch (Exception e){
			e.printStackTrace();
			return null;
		}

		finally{
			try{
				output.close();
				input.close();
			}
			//....
		}

	}
}
```

Listagem 5 – Trecho do Código da Classe Projeto.LEDService.Service.ServReg.java

```
public class ServReg implements BundleActivator {

	// ...

	public void start(BundleContext bundleContext) throws Exception {

		System.out.println("Iniciar");
		this.registry =
bundleContext.registerService(ServObj.class.getName(), new ServObjImpl(), null);
	}

	public void stop(BundleContext bundleContext) throws Exception {

	// ...
	}

}
```

APÊNDICE H – CÓDIGO FONTE DAS CLASSES DO PACOTE PROJETO.LEDSERVICE.API

Listagem 6 - Projeto.LEDService.API.ServObj.java

```
public interface ServObj {
	public ObjectInputStream cpObjServ(Object object);

}
```

APÊNDICE I – CÓDIGO FONTE DAS CLASSES DO PACOTE PROJETO.PROTOTIPO.SISTEMAUSUARIO

Listagem 7 – Trecho do Código da Classe Projeto.prototipo.SistemaUsuario.Activator.java

```
public class Activator implements BundleActivator {

  public void start(BundleContext context) throws Exception {

        ServObj srv = getServObjService(context);
      //...

  }

  public void stop(BundleContext context) throws Exception {

  }

  private ServObj getServObjService(BundleContext context){
        ServiceReference ref =
context.getServiceReference(ServObj.class.getName());
        //...

  }
```

Listagem 8 – Trecho do Codigo da Classe Projeto.prototipo.SistemaUsuario.Tela.java

```
public class Tela  extends JFrame implements Serializable{

      //...

      public Tela(){
            super("Comunicação Serial");
            Container tela = getContentPane();
            setLayout(null);

            botao01 = new JButton ("Ligar");
```

```
            // ...


            botao02 = new JButton ("Desligar");
            botao02.setBounds(200,30,100,30);
            // ...

            botao02.addActionListener(
                        new ActionListener(){
                  // ...       }
                        }
            );

            botao03 = new JButton ("Sair");
            botao03.setBounds(120,80,100,30);
            //...
      }
}
```