INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA FLUMINENSE

PÓS- GRADUAÇÃO EM ANÁLISE, GERÊNCIA E PROJETO DE SISTEMAS DE INFORMAÇÃO

KAROLYNE ALMEIDA SIQUEIRA
MICHAEL CALDAS DA SILVA

HARDWARE LIVRE CONTROLADO POR SOFTWARE LIVRE: UMA ABORDAGEM ACESSÍVEL E DIDÁTICA

Campos dos Goytacazes/RJ
2012

KAROLYNE ALMEIDA SIQUEIRA

MICHAEL CALDAS DA SILVA

HARDWARE LIVRE CONTROLADO POR SOFTWARE LIVRE: UMA ABORDAGEM ACESSÍVEL E DIDÁTICA

Monografia apresentada ao Instituto Federal de Educação, Ciência e Tecnologia Fluminense Campus Campos-Centro como requisito parcial para a conclusão da Pós- graduação em Análise, Projeto e Gerência em Sistemas de Informação.

Orientador: Ms. Fábio de Sousa Duncan

Campos dos Goytacazes/RJ
2012

KAROLYNE ALMEIDA SIQUEIRA

MICHAEL CALDAS DA SILVA

HARDWARE LIVRE CONTROLADO POR SOFTWARE LIVRE: UMA ABORDAGEM ACESSÍVEL E DIDÁTICA

Monografia apresentada ao Instituto Federal de Educação, Ciência e Tecnologia Fluminense Campus Campos-Centro como requisito parcial para a conclusão do Curso Superior de Tecnologia em Desenvolvimento de Software.

Aprovada em

Banca Avaliadora:

_____

Prof Fábio de Sousa Duncan
Mestre em Pesquisa Operacional e Inteligência Artificial – UCAM\Campos
Instituto Federal de Educação, Ciência e Tecnologia Fluminense Campus Campos-Centro

_____

Prof David Vasconcelos Corrêa da Silva
Mestre em Pesquisa Operacional e Inteligência Artificial – UCAM\Campos
Instituto Federal de Educação, Ciência e Tecnologia Fluminense Campus Campos-Centro

_____

Prof Rogério Atem de Carvalho
Doutor em Ciências de Engenharia - UENF
Instituto Federal de Educação Ciência, e Tecnologia Fluminense Campus Campos-Centro

AGRADECIMENTOS

Primeiramente agradecemos a Deus, pelo dom das nossas vidas e por nos conceder sabedoria nessa e em outras etapas. Agradecemos as nossas famílias pelo apoio, compreensão, ajuda e carinho em todos os momentos. Agradecemos também ao professor e orientador Fábio Duncan, por todo o conhecimento passado, pela dedicação e orientação. E ao professor David Vasconcelos por sua expressiva contribuição. Aos nossos amigos, pela amizade e companheirismo.

RESUMO

O desenvolvimento de *software* livre tem sido fonte de inspiração para outras áreas da tecnologia. Dentre essas áreas, temos a de produção de *hardware*. O *hardware* livre é projetado e disponibilizado almejando os preceitos do *software* livre, que objetiva o desenvolvimento colaborativo, o compartilhamento do conhecimento e a disponibilização do material produzido. Estas características possibilitam aos usuários o acesso facilitado ao produto, à colaboração no que tange às idéias, implementação de novas funcionalidades e a produção de documentação, flexibilizando e agregando valor ao *software* em questão. O presente trabalho consiste da apresentação dos conceitos de *software* e *hardware* livre, assim como dos principais projetos e licenças norteados por estas áreas. O objetivo do estudo de caso é a publicação de um passo a passo descrevendo todo o processo de criação de um protótipo de robô, baseado em *hardware* e *software* livre, que receba comandos oriundos de um sistema computacional externo e os transforme em ações para o ambiente físico.

Palavras chave: *Hardware* Livre, *Software* Livre e Arduino.

ABSTRACT

The free software development has been a source of inspiration for other technology areas. Among these areas, we have the hardware production. The free hardware is designed and made available aiming the free software precepts, which aims the collaborative development, the knowledge sharing and the material produced availability. These features enable users the easily access the product, the collaboration with regard to ideas, new features implementation and documentation production, more flexible and adding value to the software in question. The present work aims presentation of the concepts of free software and hardware, as well as major projects and licenses guided by these areas. The objective of this case study is the publishing a step by step instructions describing the entire process of creating a prototype robot, based on free hardware and software that receives commands from an external computer system and turn them into actions to the physical environment.

Key words: Free Hardware, *Free Software and Arduino*.

LISTA DE FIGURAS

LISTA DE SIGLAS

AC/DC - *Alternating Current/Direct Current*

ADK - *Android Development Kit*

ASCII - *American Standard Code for Information Interchange*

ASF - *Apache Software Foundation*

BSD - *Berkeley Software Distribution*

CC - *Creative Commons*

CERN - *European Organization for Nuclear Research*

FSF - *Free Software Foundation*

FTDI - *Future Technology Devices International*

GUI - *Graphical User Interface*

GFDL - *GNU Free Documentation License*

GPL - *General Public License*

HDMI – *High Definition Multimedia Interface*

IBM - *International Business Machines*

ICSP - *In-Circuit Serial Programming*

IDE - *Integrated Development Environment*

LGPL - *Gnu Library or Lesser General Public License*

MIT - *Massachusetts Institute of Technology*

OHL - *Open Hardware License*

OSI - *Open Source Initiative*

PC – *Personal Computer*

PCB - *Printed circuit board*

PHP - *Personal Home Page*

PSF - *Python Software Foundation*

PWM - *Pulse-Width Modulation*

RISC - *Reduced Instruction Set Computer*

SWT - *Standard Widget Toolkit*

TTL - *Transistor-Transistor Logic*

UART - *Transmissor Universal Asynchronous Receiver*

USB - *Universal Serial Bus*

SUMÁRIO

INTRODUÇÃO

O *software* proprietário possui uma licença restrita, impedindo aos seus usuários de modificá-lo. Conforme Silveira (2004), a indústria de *software* proprietário, embora seja formada por informações agrupadas e de se basear em conhecimentos acumulados pela humanidade, tentou direcionar seus esforços com o intuito de evitar a semelhança com o desenvolvimento do conhecimento científico. A ciência cresce a partir do princípio de compartilhamento, e não a partir da idéia de propriedade.

O *software* livre, ao contrário do *software* proprietário, tem como essência a liberdade de usar, copiar, estudar e redistribuir sem restrições seu código fonte, possibilitando a colaboração de indivíduos e empresas. Stallman (2009) cita o *software* livre como uma questão de liberdade, não de preço.

Segundo Osier-Mixon (2010), o *software* livre tem sido um motivador bem-sucedido da inovação de *software*. Com a evolução da tecnologia, o sucesso do *software* livre tem inspirado um novo movimento: o *hardware* livre. Desde o final da década de 90, os engenheiros vem buscando formas de aplicar conceitos de *software* livre ao *hardware* eletrônico e de computador.

A área de desenvolvimento de tecnologias com licenças livres, além de ser composta por componentes eletrônicos, também evoluiu para outros segmentos. Um exemplo é o *Rally Fighter*, um carro totalmente livre, criado pela empresa *Local Motors*, de *Wareham*, *Massachusetts*/EUA. O *Rally Fighter* é licenciado sob a *Creative Commons* e projetado através das sugestões de vários colaboradores da *internet*. Os usuários escolhem as peças para compor cada exemplar, utilizando em sua maioria, peças encontradas no mercado de autopeças. A empresa *Local Motors* disponibiliza gratuitamente na sua página *web*, os desenhos do *chassi*, do corpo e o esquema da suspensão (Rally Fighter, 2012). Outro exemplo, é a *RepRap*, uma impressora 3D que pode ser copiada, adaptada e revendida. Esta impressora tem baixo custo e permite aos usuários criar peças ou até mesmo outras impressoras em poucas horas. Segundo Bowyer (2012), criador da impressora, as impressoras 3D foram criadas há mais de 25 anos, mas só recentemente se tornaram acessíveis, devida a expiração das patentes, onde os seus proprietários detinham o monopólio da tecnologia, não investindo no desenvolvimento, pois preferiam vender poucas máquinas a preços altíssimos.

Fatores que motivam o desenvolvimento e o uso de *hardware* livre são: a disponibilização dos esquemas e projetos dos dispositivos, permitindo aos usuários fazer modificações e evoluir os projetos conforme as suas necessidades, além da isenção de *royalty* para produção e comercialização dos produtos.

As pesquisas realizadas para a elaboração deste projeto apontaram uma escassez de literatura em língua portuguesa sobre o assunto *hardware* livre. Este fator motivou a produção do texto desenvolvido neste presente trabalho, com o objetivo de apresentar o referido assunto, suas origens, conceitos, projetos e aplicações, contribuindo assim como fonte de pesquisa para trabalhos futuros. Além disso, o trabalho visa a disponibilização de um conteúdo prático que apresenta detalhadamente o uso e a aplicação de *software* e *hardware* livres em um estudo de caso.

O conceito de *hardware* livre, também conhecido por *hardware* aberto (*Open Source Hardware*), está embutido em uma grande variedade de projetos de *hardware* (Lista de Projetos, 2012). Dentre estes se destaca o Arduino, por possibilitar a criação de projetos acessíveis, flexíveis e de fácil utilização. A plataforma do Arduino é formada por uma placa com circuitos de entrada/saída para um microcontrolador AVR, um gerenciador de inicialização (*bootloader*) e um ambiente de desenvolvimento.

O estudo de caso proposto nesse trabalho tem como objetivo apresentar um protótipo de robô que é composto pela plataforma Arduino, montada em um pequeno veículo de três rodas, dotado de dois motores, que proporcionarão a movimentação do robô para os lados, para frente e para trás. Embarcado no *hardware* livre existirá uma aplicação, implementada para receber instruções de um *hardware* externo, que comandará a movimentação do robô.

O presente trabalho está dividido em mais 5 capítulos, além desta Introdução. Os Capítulos 2 e 3 descrevem todo o referencial teórico pesquisado, envolvendo os conceitos de S*oftware* Livre, *Hardware* Livre e Arduino. Já o Capítulo 4 apresenta as tecnologias e arquiteturas para o desenvolvimento do estudo de caso. No Capitulo 5 é descrito de forma detalhada o estudo de caso, os componentes e o emprego das tecnologias. O Capítulo 6 apresenta as conclusões e resultados obtidos com a realização desse trabalho, e ainda trabalhos futuros que poderão ser desenvolvidos.

2 REFERENCIAL TEÓRICO

Essa sessão apresentará os fundamentos teóricos para o desenvolvimento dessa monografia. Os conceitos que serão abordados são: *software* livre e *hardware* livre.

2.1 *SOFTWARE* LIVRE

A *Free Software Foundation* (FSF) define que um *software* pode ser considerado "livre" quando oferece quatro liberdades fundamentais:

- Liberdade de executar o programa, para qualquer propósito.
- Liberdade de estudar como o programa funciona, e adaptá-lo para as suas necessidades (acesso ao código-fonte é um pré-requisito).
- Liberdade de redistribuir cópias do programa.
- Liberdade de aperfeiçoar o programa, e liberar os seus aperfeiçoamentos, de modo que toda a comunidade se beneficie.

Então, um *software* de computador é definido como "livre" se os usuários dispõem de todas essas liberdades. Essa é a essência do *software* livre. Sua origem tem motivações ideológicas e sua proposta altera substantivamente as condições nas quais um *software* de computador pode ser desenvolvido e, mais que isso, utilizado (FSF, 2012).

Portanto se um usuário adquirir qualquer *software* livre, o mesmo poderá alterá-lo desde que respeite os princípios básicos da liberdade definidos pela FSF, ou seja, sua alteração deverá ser disponibilizada da mesma forma para outros usuários, que por sua vez também poderão modificar, adaptar às suas necessidades e redistribuir suas alterações.

Assim o conceito de *software* livre é diferente quando se trata de *software* proprietário, que não atende a essas quatro liberdades fazendo com que seus usuários fiquem presos as suas restrições.

2.1.1 Histórico do *software* livre

O surgimento do *software* livre foi em 1983, quando Richard Stallman, um pesquisador do Laboratório de Inteligência Artificial do MIT (*Massachusets Institute of Technology)*, tomou uma decisão pessoal que iria marcar profundamente a história da tecnologia da informação. Stallman deu início ao projeto GNU. Este nome incomum é de um conhecido animal africano e também o acrônimo recursivo de GNU *is not* UNIX, ou seja, o projeto GNU teria como objetivo produzir um sistema operacional livre que pudesse fazer o mesmo que o sistema *Unix*. Sua proposta era construir um sistema capaz de rodar programas e aplicativos do *Unix*, mas que fosse livre e independente de licenças proprietárias de uso. A definição desse objeto foi publicada no Jornal Doutor *Dobb* de ferramentas de *software* por Stallman através do GNU Manifesto (Silveira, 2004).

O sistema operacional *Unix* é robusto, este foi projetado para ser utilizado em servidores e *mainframes*. O *Unix* foi criado em um projeto de pesquisa que envolveu a Bell *Laboratories*, a *General Electric* e o MIT. Segundo Guesser (2005) a primeira versão surgiu em 1971 e foi criado para a operação de microcomputadores. Em 1973, a versão três do *Unix* foi escrita em linguagem de programação C, uma linguagem de alto nível. Ao ser escrito na linguagem C, o *Unix* abriu o caminho de sua popularização, uma vez que mais projetistas de sistemas podiam trabalhar com ele.

Nas palavras escrita por Stallman, disponível no site da FSF, em 1983, ele explicava que “GNU será capaz de rodar programas do *Unix*, mas não será idêntico ao *Unix*. Nós faremos todos os aperfeiçoamentos que forem convenientes, baseados em nossa experiência com outros sistemas operacionais". Sua idéia de criar um sistema operacional livre foi ganhando fama e se fortalecendo na FSF no ano de 1984 (FSF, 2012).

Com o crescimento da comunicação em rede, várias pessoas trocavam mensagens contendo pedaços dos programas e linhas de códigos. Com isso, vários componentes do sistema operacional foram desenvolvidos com as propostas do compartilhamento.

Em agosto de 1991, Linus Torvalds, um jovem matemático finlandês, anunciou em um grupo da *internet* que havia desenvolvido o *kernel* para um sistema operacional do tipo *Unix*. O *kernel* se tratava de um *software* livre, porém não seria grande nem profissional como o GNU. O *software* se chamaria *Linux*, a junção do nome do autor, Linus, com o sistema operacional *Unix*. Baseado em um poderoso sistema operacional multiplataforma, agregando

esforços da comunidade de desenvolvedores em torno da FSF, as primeiras versões do *Linux* já se mostravam mais flexíveis e robustas que o MS-DOS e o *Windows* (Silveira, 2004; Guesser, 2005).

O *Linux* era semelhante ao *Unix*, seu código-fonte era aberto e costumava envolver em seu desenvolvimento as pessoas que se interessavam em aprimorá-lo. Com isso, uma grande comunidade de colaboradores se formou para aperfeiçoar as novas versões do *software*. Segundo Silveira (2004), não é exagerado dizer que sem a *internet* e a comunicação mediada por computador dificilmente teríamos o ambiente necessário ao desenvolvimento colaborativo, a alma do *software* livre.

Assim surgiu o sistema operacional GNU/*Linux*, dando inicio a era do *software* livre uma grande alternativa ao *software* proprietário. Depois do GNU/*Linux* vários *softwares* livres foram criados, tais como o *Apache*, servidor *web* livre, o *Gimp*, para tratar desenhos e imagens e o *OpenOffice.org*, que reúne editor de texto, planilha, editor de apresentações e outros. Atualmente, existem vários *softwares* livres que rodam sobre a licença GPL (Licença Pública Geral), criada pela FSF e várias outras licenças que serão tratadas na seção 2.1.5.

2.1.2 *Software* Livre X *Software* Proprietário

As características que definem o *software* livre são: uso, cópia, modificação e redistribuição. Richard Stallman costuma comparar o *software* livre a uma receita de bolo. Tanto o *software* livre quanto a receita de bolo são um conjunto de instruções. Por exemplo: o *software* livre diz ao computador o que ele deve fazer e a receita diz ao cozinheiro os ingredientes, as medidas e a ordem que os mesmos devem ser misturados, além de mostrar como deve ser feito. Segundo Silveira (2004), como seria se o cozinheiro fosse impedido de trocar suas receitas? Ou se fosse proibido de melhorar a receita que conseguiu de sua mãe ou de seu vizinho?

Para Silveira (2004) uma receita é um conjunto de idéias ou informações, assim como um *software*. Porém, o *software* proprietário é um modelo de desenvolvimento e distribuição baseado em licenças restritivas de uso, ou seja, o mesmo tem a autoria e propriedade do *software*. Com o *software* proprietário a receita não seria entregue junto com o bolo, impedindo a pessoa de modificar, recriar e redistribuir essa receita. O modelo do *software* proprietário esconde o código que o compõem.

A indústria de *software* proprietário se encaminhou para tentar bloquear e evitar que o seu desenvolvimento fosse semelhante ao desenvolvimento do conhecimento científico, apesar de ser composto por informações agrupadas e de se basear em conhecimentos acumulados pela humanidade. A ciência cresce a partir do princípio de compartilhamento, e não a partir da idéia de propriedade. Por ser essencialmente social, não se aplica ao conhecimento a idéia de apropriação privada (Silveira, 2004).

Quando o usuário compra um *software* proprietário, o mesmo não está comprando o *software* e sim a sua licença, pois o *software* continua com a empresa que o desenvolveu. Silveira (2004) comprara à compra do software proprietário a compra de um imóvel, por exemplo, quando uma pessoa compra uma casa, ela tem o direito de reformar, ampliar e demolir, além de vender para uma outra pessoa. Porém um *software* proprietário não dá ao seu usuário nenhuma destas opções. Ele continua a ser propriedade da empresa que o vendeu. As pessoas que usam *software* proprietário na verdade são como locatárias de um imóvel que nunca será seu.

Quanto ao *software* livre, ele é completamente diferente do *software* proprietário, pois a sua essência é a liberdade de usar e desenvolver o programa. Segundo Hexsel (2002), o modo de produção de *software* tem resultado em produtos de excelente qualidade e grande penetração em certos nichos do mercado mundial de *software*. A característica mais importante do *software* livre é a liberdade de uso, cópia, modificações e redistribuição. Esta liberdade é conferida pelos autores do programa e é efetivada através da distribuição do código-fonte dos programas, o que os transforma em bens públicos, disponíveis para utilização por toda a comunidade e da maneira que seja mais conveniente a cada indivíduo.

A liberdade para usar, copiar, modificar e redistribuir *software* livre lhe confere uma série enorme de vantagens sobre o *software* proprietário. A mais importante delas é a disponibilidade do código-fonte, porque isto evita que os usuários se tornem reféns de tecnologias proprietárias. Além desta, as vantagens técnicas são também consideráveis. A comunidade de desenvolvimento de *software* livre está espalhada pelo mundo todo e seus participantes cooperam nos projetos através da *internet*.

Com isso, o *software* livre possui uma licença não-proprietária de uso, pois ele tem um ou vários autores, mas não proprietários. Dessa forma, quem o adquire pode usá-lo para qualquer finalidade. A única privação que os usuários de *software* livre têm é a de não torná-

lo um *software* proprietário. Então, após realizar qualquer alteração nele, o mesmo não pode ser vendido, seu código-fonte deverá permanecer livre.

Em relação aos *softwares* distribuídos gratuitamente, o autor Silveira (2004) afirma que o *software* livre é *Open Source*. *Open Source* é um *software* que possui o código-fonte aberto. Entretanto é possível que um *software de código-fonte aberto* não assegure as quatro liberdades que definem o *software* livre. Por isso é importante distinguir as categorias: *software* aberto, *software* gratuito e *software* livre. Existem vários *softwares* gratuitos que são proprietários. O fato de ser um *software* distribuído gratuitamente não significa que ele seja livre.

O *software* gratuito (*freeware*) é um programa que pode ser utilizado sem custo pelo usuário. Com isso, um *software* pode ser gratuito e livre, como também pode ser gratuito e fechado. Segundo Alecrim (2011) quando se trata de um *software* gratuito e fechado, o mesmo possui restrição, ou seja, somente o autor ou a entidade que o desenvolve terá acesso ao seu código-fonte, não permitindo aos usuários estudá-lo, alterá-lo e redistribuí-lo, somente permitirá usá-lo da forma como foi disponibilizado, havendo várias limitações quanto à distribuição.

O *software* de código aberto (*Open Source*) teve inicio no ano de 1998, quando um movimento formado por um grupo de pessoas que não estavam de acordo com os ideais filosóficos ou com os aspectos do *software* livre criou a OSI (*Open Source Initiative*). A OSI não descarta as quatros liberdades definidas pela FSF, porém para um *software* ser considerado *Open* S*ource* o mesmo deve atender os dez requisitos abaixo:

1. Distribuição livre.
2. Acesso ao código-fonte.
3. Permissão para criação de trabalhos derivados.
4. Integridade do autor do código-fonte.
5. Não discriminação contra pessoas ou grupos.
6. Não discriminação contra áreas de atuação.
7. Distribuição da licença.
8. Licença não específica a um produto.

9. Licença não restritiva a outros programas;
10. Licença neutra em relação à tecnologia.

A diferença entre o *software* livre e o *Open Source* está no fato da OSI ter receptividade maior em relação às iniciativas de *software* do mercado. Dessa forma, empresas como *Microsoft* e *Oracle* que desenvolvem *software* proprietário, podem desenvolver soluções de código aberto utilizando suas próprias licenças, desde que estas respeitem os critérios da OSI. No *software* livre, empresas como estas provavelmente enfrentariam algum tipo de resistência, uma vez que suas atividades principais ou mesmo os programas oferecidos podem entrar em conflito com os ideais morais da FSF (Alecrim, 2011; Feller, 2005).

O modelo de desenvolvimento do *software* livre conquistou vários seguidores devido à disponibilização do seu código-fonte. Através da *internet* várias pessoas colaboram com a melhoria desse desenvolvimento. Um exemplo pode ser percebido no desenvolvimento do *software Blender* (Blender, 2012). Com a disponibilização do seu código-fonte, um grande número de pessoas passou a desenvolvê-lo em rede. Com isso, o *Blender* foi melhorado a cada nova versão e o valor econômico do trabalho de seus colaboradores já ultrapassa o valor pago à empresa que o criou.

Segundo Raymond (1998) esse tipo de processo de desenvolvimento colaborativo proposto pelo *software* livre é denominado “bazar” de construção de *software*. Já o processo de desenvolvimento de *software* proprietário que utiliza um modelo fechado e hierárquico é denominado “catedral”.

Na próxima seção, serão abordados esses dois modelos de desenvolvimento de *software* proposto pelo autor Raymond (1998).

2.1.3 Catedral e Bazar

Raymond (1998) define o modelo catedral como um processo de desenvolvimento fechado, sendo composto por um grupo específico de pessoas trabalhando de forma isolada, não disponibilizando versões betas para testes antes do tempo previsto para a sua entrega.

Para Raymond, com a disponibilização do código-fonte para testes, os erros são descobertos rapidamente. Quando se trata do modelo catedral uma quantidade de tempo e energia irregular deve ser gasta procurando por erros, pois as diversas versões de código são avaliadas por um número limitado de desenvolvedores.

Ao contrário do modelo catedral, Raymond define o modelo bazar como uma grande comunidade, formada por várias pessoas colaborando com o desenvolvimento de *software*, disponibilizando o código- fonte e versões betas para testes.

O desenvolvimento de *software* considerado bazar surgiu devido ao processo de desenvolvimento proposto por Linus Torvalds, no projeto GNU/*Linux*, onde todos que tivessem interesse e competência participavam do projeto que gerou um processo extremamente rico e veloz de melhoria do *software* (Raymond, 1998).

A verticalização e a burocratização do modelo proprietário não conseguem fazer frente ao modelo centrado na colaboração e na interação de milhares de pessoas, tais como em uma feira, em um bazar (Silveira, 2004).

Esse modelo de desenvolvimento de *software* livre permite que os usuários se tornem desenvolvedores. Esses usuários enviam mensagens aos coordenadores de projetos mostrando falhas e *bugs*, além de propor novas funcionalidades ou redefinir as existentes. Com isso, a nova versão de um *software* livre tende a ter as suas falhas rapidamente constatadas e corrigidas.

Hexsel (2002) analisa o modelo Catedral como: "O modelo tradicionalmente empregado na indústria é similar ao projeto de uma catedral medieval, no qual um restrito grupo de projetistas exerce controle férreo sobre o trabalho de um pequeno exército de operários. O modelo da catedral empregado na maioria dos projetos de desenvolvimento de *software* proprietário, onde o modelo descreve o relacionamento entre a gerência de projeto (e o departamento de *marketing*) estabelecem metodologias, tarefas e prazos, que devem ser cumpridos pelos programadores engajados no projeto".

O modelo Bazar, Hexsel (2002) caracteriza como: "O outro modo de organização, frequentemente empregado pela comunidade de *software* livre que se assemelha a um anárquico bazar, onde não há hierarquia entre os participantes e todos cooperam para que o bazar seja atrativo aos compradores, ao mesmo tempo em que competem pela atenção destes mesmos compradores. Na produção de *software* no bazar os projetos são informalmente organizados ao redor da proposta de desenvolvimento de algum aplicativo 'interessante', do

qual os interessados participam voluntariamente, e o/a líder do projeto emerge por seus méritos como programador/a ou projetista".

2.1.4 Vantagens e Desvantagens do *software* livre

O uso do *software* livre tem vantagens visto que sua essência é a liberdade, onde o programa pode ser copiado, modificado e redistribuído. Isso lhe confere vantagens sobre o *software* proprietário.

De acordo com Freitas e Teles (2002), é possível identificar as seguintes vantagens na utilização do *software* livre:

- a utilização de código aberto facilita a correção de *bugs*, pois viabiliza que usuários e desenvolvedores ajudem a resolver;
- permite descobrir mais facilmente as capacidades dos programas e alterá-las de acordo com as necessidades do usuário;
- algumas vezes o usuário é um co-desenvolvedor;
- o desenvolvimento e os testes extensivos realizados e compartilhados pelos usuários, garantem a confiabilidade e a atualização do *software*;
- o suporte pode ser realizado por companhias especializadas ou por usuário/desenvolvedor via plataformas na *web* e grupos virtuais;
- a disseminação e a ampliação da participação de usuários/desenvolvedores possibilita a rápida atualização, verificações e correção de problemas;

Quanto as supostas desvantagens, do ponto de vista de uma empresa, o maior impasse ao utilizar um projeto de *software* livre é a inexistência de uma entidade com identidade jurídica claramente definida e que seja legalmente responsável pelo software, pois nos casos de prejuízos decorrentes de erros deste software, não há nenhuma entidade que possa ser responsabilizada civil ou criminalmente por eventuais perdas e/ou danos (Hexsel, 2002). Ao mesmo tempo, é comum que os softwares proprietários apresentem nas suas licenças a informação de que o desenvolvedor se exime de quaisquer responsabilidades quanto aos prejuízos que podem ser causados pela utilização do produto. De qualquer forma a cláusula é

discutível e o foco de quem responsabilizar existe, gerando uma sensação de segurança para empresas que por vezes pode ser irreal.

Como não há um responsável pelos projetos, uma possível desvantagem é ter dificuldade para obtenção de suporte na utilização de *software* livre, pois os usuários não tem garantia de manutenibilidade.

Alguns projetos de *software* livre tem a interface pouco intuitiva, com instalações complicadas e o padrão diferente dos demais *softwares* utilizados mundialmente. Segundo Silveira (2004), as desvantagens do início de 1990 vão sendo colaborativamente superadas. Atualmente, as experiências demonstram que a instalação e as interfaces dos *softwares* livres tem se tornado mais amigáveis.

2.1.5 Licenças de *software* livre

A fim de preservar o conceito de *software* livre durante o desenvolvimento de novos trabalhos colaborativos, modificações e redistribuições, existem licenças nas quais são definidas as normas de utilização do *software*. Segundo Barbosa (2007), durante todo o caminho percorrido pelo movimento *software* livre, surgiram diversas licenças, portando políticas diferentes entre si, utilizando diferentes diretrizes defendidas em cada uma, embora possuam em comum as quatro liberdades anteriormente descritas que definem o *software* como livre. Em geral, as licenças se preocupam principalmente em estabelecer regras em domínios tais quais:

- Proteção da autoria do *software* original;
- Maneiras de redistribuição do *software*;
- Formas de distribuição de modificações e seu código-fonte;
- Não restrição de venda do *software*;
- Instalação em número irrestrito de computadores.

Nas próximas sessões, as principais licenças de *software* livre serão detalhadas.

2.1.5.1 Licença GNU GPL (GNU *General Public License*)

A licença GNU GPL teve início em janeiro de 1989. Atualmente é uma das licenças mais utilizadas por parte de projetos de *software* livre. Ela se baseia em quatro fundamentos:

1 - A liberdade de poder executar o *software* para qualquer propósito.

2 - A liberdade de estudar o funcionamento do *software* e adaptá-lo conforme suas necessidades, pelo fato do acesso ao código-fonte ser um pré-requisito para esta liberdade.

3 - A liberdade de poder redistribuir cópias podendo compartilhar com mais pessoas.

4 - A liberdade de aprimorar o *software* e liberar os seus aprimoramentos, fazendo com que outras pessoas se beneficiem dele.

Tendo em vista esses fundamentos, a licença GPL permite que os *softwares* sejam distribuídos e reaproveitados, desde que mantenha os direitos do autor, fazendo com que não permitam que o *software* modificado seja usado de maneira diferente das liberdades originais. Por exemplo, a licença GPL não permite que o código seja apossado pela pessoa que o modificou ou que seja colocado sobre esse *software* restrições que abstenha sua distribuição da mesma maneira que foi adquirido.

2.1.5.2 Licença BSD (*Berkeley Software Distribution*)

A licença BSD foi criada pelos "Regentes da Universidade da Califórnia" na Universidade de *Berkeley*. Essa licença tem o seu código aberto e é considerada como de domínio público, podendo ser modificada sem nenhuma restrição. Inicialmente era utilizada nos sistemas operacionais do tipo *Berkeley Software Distribution* que era derivado do *Unix*. Atualmente, outros sistemas que não pertencem ao grupo BSD são distribuídos sob esta licença (Wikipédia, 2012).

A licença BSD impõe poucas restrições quando comparada a outras, como a GNU *General Public License* ou mesmo as restrições padrão determinadas pelo *copyright*, colocando-a relativamente próxima do domínio público. A BSD foi caracterizada de *copycenter*, ou seja, o usuário dessa licença é levado ao centro de cópias e realiza quantas copias quiser (Wikipédia, 2012).

Por possibilitar que produtos sob a licença sejam incorporados em *software*s proprietários, empresas como Microsoft e a Apple fizeram seu uso para fins lucrativos. Como exemplo, a Apple utilizou como base uma versão do sistema operacional BSD para criar seu próprio sistema operacional, o Mac OS.

2.1.5.3 GNU LGPL *(Library or Lesser General Public License)*

A LGPL foi criada por Richard Stallman e Eben Moglen em 1991. Em 1999, sofreu uma atualização. A licença LGPL é semelhante a GPL, porém a LGPL permite também que programas que não estejam sob as licenças GPL e LGPL se associe a ela, incluindo o *software* proprietário.

Outra característica importante da licença é que os trabalhos decorrentes sob a LGPL devem estar disponíveis em bibliotecas.

A LGPL acrescenta restrições ao código-fonte desenvolvidos, mas não exige que seja aplicada a outros *softwares* que empreguem seu código, desde que este esteja disponível na forma de uma biblioteca. Logo, inclusão do código desenvolvido sob a LGPL como parte integrante de um *software* só é permitida se o código fonte for liberado (Wikipédia, 2012).

2.1.5.4 Licença MIT

A licença MIT conhecida também como licença X ou licença X11, foi criada pelo *Massachusetts Institute of Technology*. Sua característica não é de uma licença *copyleft*, ou seja, a mesma permite a reutilização de um *software* licenciado em *softwares* livres e proprietários.

Devido a MIT ser uma licença permissiva, não existe restrição ao seu uso, desde que o usuário do *software* e da licença siga o seu único termo: "O aviso de *copyright* acima e esta permissão deverá ser incluído em todas as cópias ou partes substanciais do *software*."

No termo da licença, apresenta as condições do *copyright* e também a ausência de garantias e responsabilidades semelhantes à licença BSD, no qual protege os detentores do direito autoral de qualquer processo judicial relacionado ao *software*.

2.1.5.5 Licença Apache

A licença *Apache* foi criada pelo ASF entre 1995 e 2000, tendo como suas primeiras versões a 1.0 e 1.1. Em 2004, a ASF aprovou a atualização da licença para a versão 2.0. Com essa atualização, os termos da licença que eram utilizados apenas nos projetos da ASF, passaram a ser adotados por outros projetos que não fazem parte dessa organização, por ser de fácil aplicação, sem necessidade de modificar o texto.

Segundo Campos (2010), essa facilidade de aplicação é uma das várias razões que explicam a popularidade das licenças *Apache*. Em 2009, mais de 5000 projetos externos à Fundação *Apache* hospedados no repositório *SourceForge* a adotavam, e em 2008 o *Google* informou que 25% dos 100.000 projetos então hospedados no seu *Google Code* a adotavam.

A licença Apache possui as condições gerais semelhantes à licença BSD, permitindo o livre uso, redistribuição e alteração, além do seu código poder ser reaproveitado em projetos proprietários.

Campos (2010) cita algumas características que a distinguem da simplicidade espartana das BSDs, a começar pela questão das patentes de *software*, que fica explícita na sua terceira cláusula, esclarecendo (de forma bastante extensa e específica) que todo contribuidor de código para o *software* em questão concede também uma licença mundial e perpétua para uso de suas patentes que sejam necessárias para uso ou distribuição do código contribuído por ele em combinação com o *software* em questão.

A próxima seção irá abordar os principais projetos de *software* livre que adotaram essas licenças disponibilizando os seus códigos-fonte conforme as restrições de cada uma delas.

2.1.6 Projetos de *Software* Livre

Vários projetos adotaram as licenças de *software* livre no seu desenvolvimento. Esses projetos disponibilizaram o código-fonte e ganharam importância na comunidade de desenvolvedores e usuários finais. Dentre esses projetos estão: *Mozilla Firefox, Android, OpenOffice, BrOffice, LibreOffice*, GNU/*Linux*, *GNOME*, Servidor *Apache* e *Eclipse*. Alguns desses projetos serão abordados a seguir.

- **GNU/*LINUX***

O GNU/*Linux* surgiu quando o *kernel* do *Linux*, desenvolvido por Linus Torvalds, se integrou com o projeto GNU, liderado por Richard Stallman. O núcleo do *Linux* é distribuído sob a licença *GNU General Public License*, onde os colaboradores podem contribuir com o seu desenvolvimento. O projeto GNU deu inicio ao desenvolvimento de *software* livre. Seu objetivo era criar um sistema operacional totalmente livre, onde qualquer pessoa teria direito de usar, estudar, modificar e redistribuir o seu código fonte garantindo os mesmos direitos para outros usuários (GNU/Linux, 2012).

Com a junção do GNU e *Linux* surgiu o sistema operacional *Linux*, com características de um bom desempenho e confiabilidade. Esse sistema operacional é bastante utilizado em servidores devido a essas características, além de ser utilizado em dispositivos com computação embarcada.

Figura 1 - Sistema Operacional Ubuntu (versão 12) baseado em Linux.

- **GNOME**

O GNOME é um projeto de *software* livre e parte do Projeto GNU. O projeto provê basicamente duas soluções. A primeira delas é o ambiente *desktop* GNOME (Figura 2), um ambiente intuitivo e atraente para usuários de sistemas operacionais, como as distribuições *Linux Ubuntu*, *Mint*, *Debian*, *Red Hat Enterprise* e *Fedora,* além de estar disponibilizado para outros sistemas operacionais como o *OpenSolaris*. A segunda delas é a plataforma de desenvolvimento GNOME, um *framework* robusto para o desenvolvimento de aplicações gráficas para o ambiente *desktop* (GNOME, 2012).

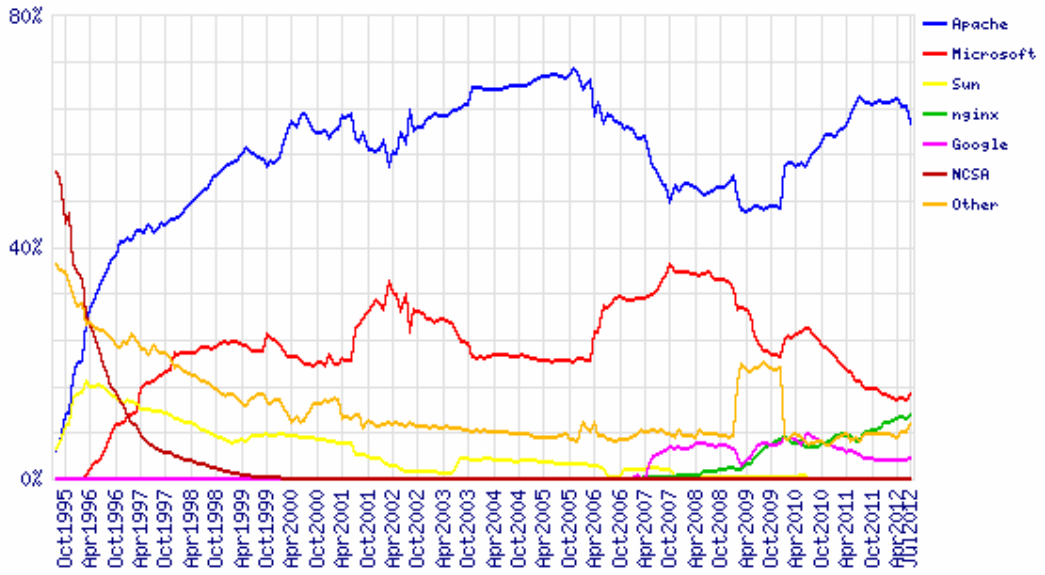
Figura 2 - Ambiente desktop GNOME.

- **Servidor *Apache***

O servidor *Apache* foi criado em 1995 por Rob McCool e tem como característica ser um servidor *Web* livre (HTTP). Atualmente, o servidor *Apache* é responsável pela maior parte

de processamento de páginas da *Internet*. Uma vantagem do servidor *Apache* em relação a alguns servidores *Web* proprietários é a característica de ser multiplataforma, sendo utilizado em sistemas *Unix*, *GNU/Linux*, *Windows* e *Mac OS* (Servidor Apache, 2012).

Baseado nos resultados de pesquisas da empresa Netcraft que analisou o uso dos diversos servidores *Web* referente ao mês de Outubro de 1995 até o presente mês (julho de 2012), percebe-se que o servidor *Apache* é o mais utilizado no mundo conforme a Figura 3 (Netcraft, 2012), sendo um exemplo bem sucedido na área de desenvolvimento de *software* livre.



Developer	June 2012	Percent	July 2012	Percent	Change
Apache	448,452,703	64.33%	409,185,675	61.45%	-2.89
Microsoft	95,891,537	13.76%	97,385,377	14.62%	0.87
nginx	72,881,755	10.46%	73,833,173	11.09%	0.63
Google	22,464,345	3.22%	22,931,169	3.44%	0.22

Figura 3 - Estática do uso de servidores Web.

- **Eclipse**

O projeto Eclipse foi desenvolvido pela IBM que disponibilizou sua primeira versão como um *software* livre. O Eclipse é um IDE (Figura 4) desenvolvido na linguagem de

programação Java, utilizando o modelo *open source* de desenvolvimento de *software*.

Atualmente, o Eclipse é o ambiente de desenvolvimento mais utilizado no mundo (Eclipse, 2012), pois dispõe de características como o uso do SWT (*Standard Widget Toolkit*), uma camada de componentes baseada nos componentes padrões do sistema operacional como botões, janelas, etc, possibilitando o desenvolvimento de *layouts* de interface gráfica sem a utilização do *Swing*, que apresenta um padrão específico no *layout* dos componentes e o desenvolvimento baseado em *plugins* que atendem as necessidades de diversos programadores. Com o uso desses *plugins*, é possível desenvolver nesse ambiente em outros linguagens de programação como C/C++, PHP, *ColdFusion* e *Python* entre outras.

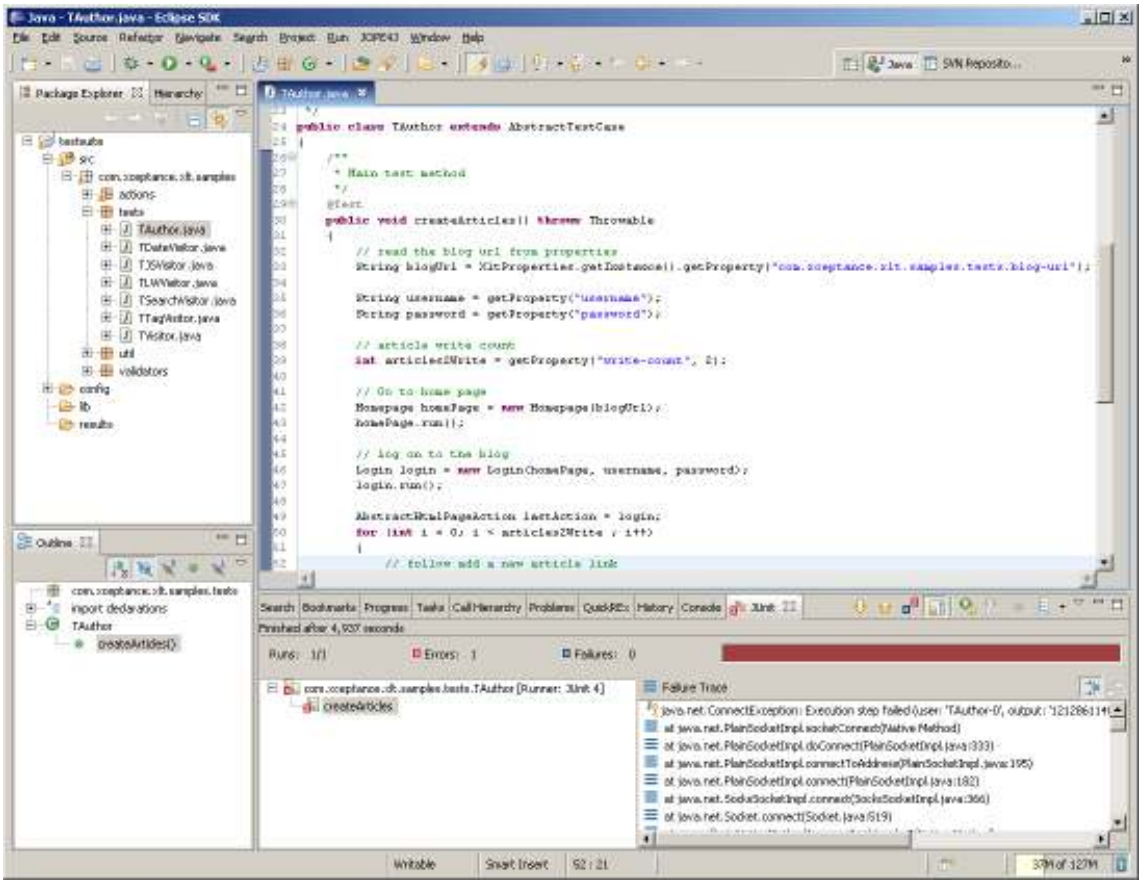


Figura 4 - Ambiente IDE do Eclipse.

O *software* livre é embasado em uma série de conceitos que tem sido fonte de inspiração para outras áreas da tecnologia. Entre essas áreas, temos a de produção de *hardware*, que é foco do trabalho e será apresentado na próxima seção.

2.2 *HARDWARE* LIVRE

A idéia de *hardware* livre é semelhante à de *software* livre. O objetivo é fazer pelos equipamentos o mesmo que as tecnologias livres fazem com os programas de computador, ou seja, disseminar o conhecimento do processo de desenvolvimento (Santos, 2011).

O principal obstáculo do *hardware* livre em relação ao *software* livre é que o *software* é fácil de duplicar e pode ser copiado gratuitamente, ao passo que o *hardware* é constituído por matéria, ou seja, átomos em vez de bits. Além disso, geralmente o *hardware* é patenteado, em vez de ter *copyright*, e o processo de obter e defender patentes gera um custo elevado (Ackermann, 2009).

O *hardware* livre é aberto no mesmo sentido do *software* baseado em padrão aberto - o conceito de livre, como no caso da liberdade de expressão que Stallman abordou no GNU Manifesto. Segundo Osier-Mixon (2010), o *hardware* nunca pode ser "grátis como a cerveja" porque a duplicação sempre custa alguma coisa e porque até mesmo os seus defensores mais bem-intencionados não podem se dar ao luxo de oferecer produtos físicos grátis indefinidamente.

Osier-Mixon (2010) cita que um produto físico é simplesmente a implementação de um *design*, e os *designs* de *hardware* podem ser oferecidos gratuitamente com uma licença aberta, com *copyright* ou patenteada. O licenciamento fica por conta do proprietário.

2.2.1 Histórico do *hardware* livre

Em 1970, um pequeno grupo de engenheiros da *Homebrew Computing Club* compartilhou projetos, que depois se tornaria o primeiro computador pessoal. O objetivo desse grupo era tornar os computadores mais acessíveis a todos. Segundo Wozniak (1984), os esquemas e desenhos de produtos da *Apple* I foram disponibilizados livremente, a fim de ajudar outros membros a construir seus próprios sistemas.

Ainda no ano de 1970, Carve Meade e Lynn Conway desenvolveram um método pelo qual os usuários podiam começar a projetar em larga escala os circuitos integrados. Diversos projetos foram agrupados em um único ciclo de produção a fim de reduzir os custos de

desenvolvimento. Esse método permitiu que as universidades tivessem acesso a produção de circuitos integrados de alta complexidade com um baixo custo (Acosta, 2009).

Atualmente podem ser obtidos preços acessíveis na produção de protótipos para qualquer tipo de projeto. Um exemplo disso é a possibilidade de envio do *design* de um *hardware* livre a uma fábrica na China, que produza a baixo custo, viabilizando o surgimento de inovações. Esse tipo de desenvolvimento permite que os usuários criem e produzam seus próprios bens, tornando uma liberdade impensável para o tipo de modelo econômico atualmente capitalista, onde grandes corporações se fundem na tentativa de monopolizar os mercados (Lima, 2010).

2.2.2 *Hardware* livre x *Hardware* Proprietário

O *hardware* livre tem como característica a disponibilização dos esquemas e arquivos de projetos, permitindo que os usuários adicionem, atualizem e troquem componentes dos dispositivos. Ao contrário do *hardware* livre, o fabricante do *hardware* proprietário mantém o *hardware* fechado e limita o conjunto de componentes, fazendo com que o mesmo não seja expansível pelo usuário final (McNamara, 2007). Outra característica do *hardware* livre é o baixo custo em relação aos *hardwares* proprietários.

A disponibilização da documentação nos projetos de *hardware* livre é um exemplo do que ocorre em face do *hardware* ser projetado e fornecido utilizando licenças livres que determinam e caracterizam o seu uso. Essas licenças serão abordadas na próxima seção.

2.2.3 Licença de *hardware* livre

As licenças utilizadas por alguns projetos de *hardware* livre são as mesmas utilizadas nos projetos de *software* livre. Dentre essas licenças estão a: GPL, LGPL e a BSD. Em 2011, a CERN (Organização Européia para Pesquisa Nuclear) criou uma nova licença de *hardware* livre, a OHL (*Open Hardware License*), que está disponível na sua página *web* (OHL, 2012). A licença é um repositório de desenvolvimento de equipamentos livres, criada para gerenciar o uso, a cópia, a modificação e a distribuição da documentação de projeto de *hardware*, e a fabricação e distribuição de produtos. A documentação de projeto de *hardware* livre inclui

diagramas esquemáticos, desenhos, esquemas de circuitos ou placas de circuito impresso, desenhos mecânicos, fluxogramas e textos descritivos, bem como outros materiais explicativos (*Open Hardware License*, 2012).

Segundo Javier Serrano (2011), engenheiro do CERN e fundador da organização que administra a licença de *hardware* livre, o impulso para o *hardware* livre foi amplamente motivado pela inveja bem-intencionada pelos seus colegas que desenvolvem os *drivers* para *Linux*. "Eles são parte de uma comunidade muito grande de designers que compartilham seus conhecimentos e seu tempo para deixar o sistema operacional o melhor possível. Sentimos que não havia nenhuma razão intrínseca para que o desenvolvimento de *hardware* devesse ser diferente".

Além disso, as modificações que são feitas e distribuídas, devem estar sob as condições da licença, o que garante que toda a comunidade vai continuar a se beneficiar de melhorias, no sentido de que todos, por sua vez, sejam capazes de fazer modificações para essas melhorias (Thompson, 2008).

2.2.4 Projetos de *hardware* livre

Assim como a área de desenvolvimento de *software* livre, o *hardware* livre é composto por diversos projetos que obtiveram êxito. Os microcontroladores comumente são partes integrantes destes projetos.

O microcontrolador é um computador em um *chip*, que contém processador, memória e periféricos de entrada/saída. Para Guimarães (2010) um microprocessador pode ser programado para funções específicas, em contraste com outros microprocessadores de propósito geral, como os utilizados nos PCs. Estes são embarcados no interior de algum outro dispositivo, no caso o Arduino, para que possam controlar suas funções ou ações.

O diferencial do projeto Arduino em relação às outras plataformas de desenvolvimento de microcotroladores é a sua facilidade de uso. Os usuários conseguem aprender o básico e criar os seus próprios projetos em um curto período de tempo. Além de ter uma comunidade formada por usuários de Arduinos, que compartilham os seus códigos e diagramas de circuitos para que outros usuários possam copiar e modificar. Essa comunidade também oferece fóruns, onde as dúvidas são sanadas e compartilhadas (McRoberts, 2010).

Essa seção irá descrever alguns dos principais projetos desenvolvidos com *hardware* livre.

- ***OpenSPARC***

O *OpenSparc* é um *hardware* de código aberto criado pela *Sun Microsystems.* Seu código-fonte está sob licença pública geral GNU. Em 2006 a *Sun* lançou o *OpenSPARC T1* (Figura 5), a primeira versão do projeto *OpenSparc*.

Figura 5 - Placa OpenSparc T1.

Segundo a *OpenSparc* (2012), esse modelo foi desenvolvido com oito núcleos, cada um capaz de executar quatro *threads* concorrentemente, para um total de trinta e duas *threads*. Cada núcleo executa instruções em ordem e sua lógica é dividida entre seis estágios de *pipeline*.

A técnica *pipeline* permite que o processador realize a busca de uma ou mais instruções além da próxima a ser executada. Estas instruções são colocadas em uma fila de memória dentro do processador onde aguardam o momento de serem executadas. Em 2007, foi lançado o OpenSPARC T2 (Figura 6) com oito núcleos, dezesseis *pipelines* e sessenta e quatro *threads*.

Figura 6 - Placa OpenSparc T2.

- ***BeagleBoard***

O *BeagleBoard* é um *hardware* de código aberto com recursos de *software* livre produzido pela *Texas Instruments* juntamente com *Digi-Key*. Seu microprocessador é baseado em ARM.

Figura 7 - Placa BeagleBoard.

Segundo o *BeagleBoard* (2012)*, a placa* usa o mesmo mecanismo de processamento de vários *smartphones* e *netbooks*, que o deixa suficientemente potente para executar uma distribuição integral de *Linux* e fornecer vídeo de alta definição. A saída do vídeo pode ser fornecida através de S-Vídeo e HDMI.

- ***OpenCores***

Fundada em 1999, a *OpenCores* é uma das maiores comunidades de *hardware open source* por meio de automação de *design* eletrônico*,* fornecendo designs de vários núcleos sob licenças abertas. Segundo o *OpenCores* (2012)*,* o objetivo do projeto é solucionar problemas de fabricação dos novos *chips* utilizando núcleos, onde com a ajuda de vários, cada núcleo terá uma melhor documentação e mais dados para futuras implementações. Como o código-fonte é disponibilizado, qualquer desenvolvedor pode descobrir o que precisa sobre o núcleo, sem custo para sua utilização e com suporte de diversos outros usuários. Com isso, os desenvolvedores obtêm conhecimento e produzem novas aplicações.

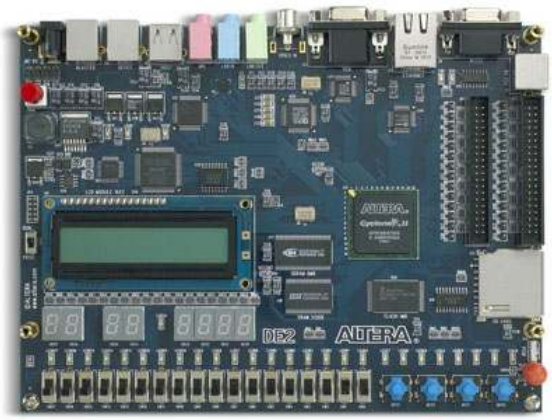


Figura 8 - Modelo de placa OpenCores.

- **Arduino**

O Arduino é uma plataforma *open-source* de computação física baseado em uma placa de microcontrolador simples, e um ambiente de desenvolvimento para escrever *software* para a placa (Arduino, 2010).

O Arduino pode ser usado para desenvolver objetos interativos, tendo como entradas uma variedade de interruptores ou de sensores, controlando uma variedade de luzes, motores, e outros resultados físicos. Os projetos do Arduino podem ser *stand-alone*, ou seja, os projetos são auto-suficientes, pois o seu funcionamento não necessita de um *software* auxiliar, como um interpretador, sob o qual terão de ser executados. Além disso, os projetos podem ser comunicar com *software* rodando em computadores. As placas podem ser montadas ou compradas pré-montadas e o IDE *open-source* pode ser baixado gratuitamente (Jamieson, 2010).

O Arduino faz parte do foco desse trabalho e será abordado no próximo capítulo.

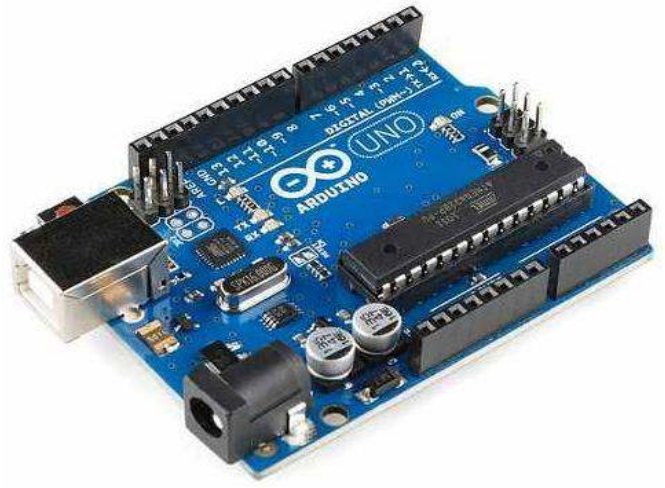


Figura 9 – Modelo UNO da placa Arduino.

3 ARDUINO

Arduino é uma plataforma de computação física que consiste em sistemas digitais ligados a sensores e atuadores, que permitem construir sistemas que percebam a realidade e respondam com ações físicas. Esta plataforma é baseada em uma placa de entrada/saída microcontrolada e desenvolvida sobre uma biblioteca que simplifica a escrita da programação em C/C++ (Arduino, 2012).

O Arduino faz parte do conceito de *hardware* e *software* livre e está aberto para uso e contribuição de toda sociedade. O conceito Arduino surgiu na Itália em 2005. A palavra Arduino em italiano significa "amigo forte". O Arduino é como um sistema de microcontrolador de código aberto, projetado para facilitar a aprendizagem, usabilidade e flexibilidade para o desenvolvimento de *hardware* livre (Beppu e Fonseca, 2010; Premeaux, 2011).

O objetivo do projeto Arduino é criar um dispositivo para controlar protótipos construídos de forma menos dispendiosa do que outros sistemas disponíveis no mercado. O Arduino consegue facilitar o desenvolvimento, pois fornece um *hardware* simples e uma plataforma de *software* que roda em qualquer computador *desktop* ou *laptop*.

O Arduino funciona como uma ponte entre os seres humanos e os sensores, devido à facilidade no aprendizado de desenvolvimento com microcontroladores, além da facilidade de compreender a tecnologia de sensor e a escrita do código (Evans, 2011).

A facilidade no aprendizado é relacionada ao ambiente Arduino que foi projetado para a usabilidade de qualquer pessoa com ou sem conhecimento de *software* ou experiência eletrônica. Com o Arduino, é possível construir objetos que podem responder e/ou controlar luz, som, toque e movimento. O ambiente Arduino tem sido utilizado para criar uma grande variedade de artefatos, incluindo instrumentos musicais, robôs de luz, jogos interativos e móveis (Buechley, 2010; Margolis, 2011).

3.1 Arquitetura e Característica

Para o desenvolvimento do Arduino é utilizado um IDE (*Integrated Development Environment*) conforme a Figura 10, um ambiente que possui um conjunto de bibliotecas que

apoiam o desenvolvimento do *software* com objetivo de agilizar este processo. O IDE permite gravar e editar código e converter este código em instruções que o *hardware* Arduino compreenda. Além de transferir as instruções para a placa Arduino através de um processo chamado de *upload*.

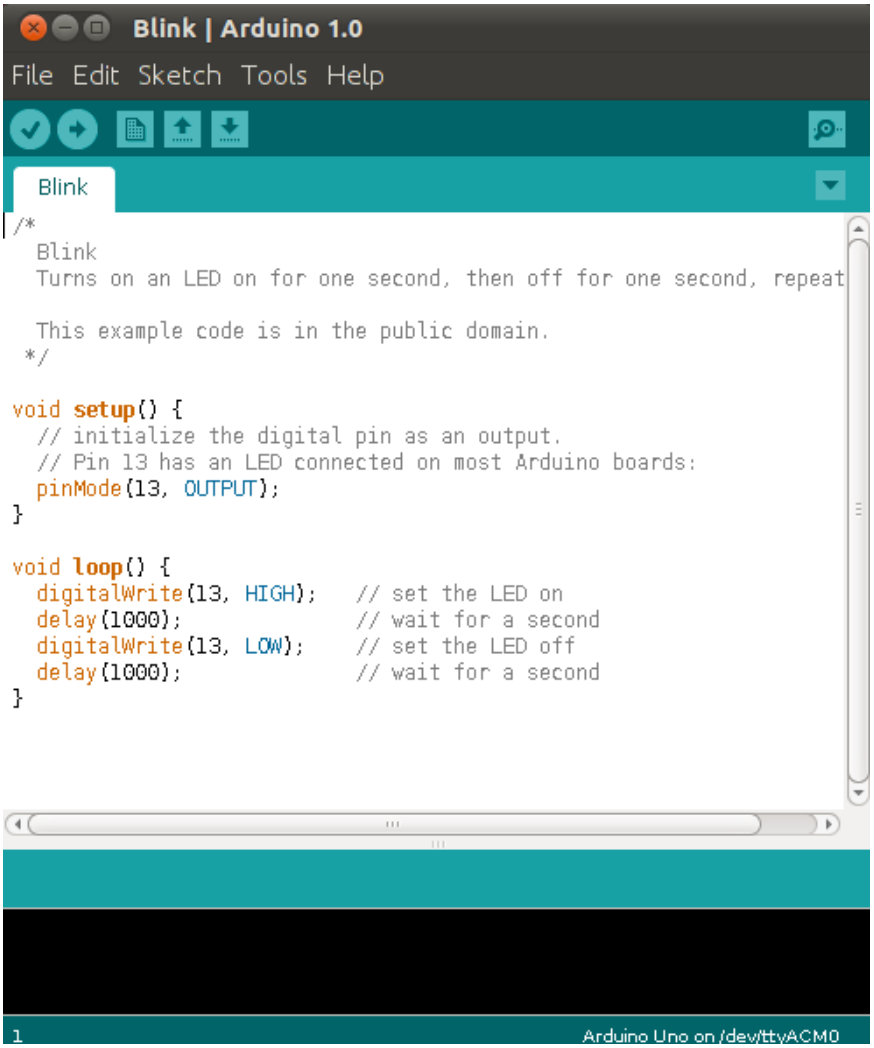


Figura 10 - Ambiente de desenvolvimento IDE

O funcionamento dos programas no ambiente de desenvolvimento do Arduino é definido por duas funções: *setup* e *loop*. A função *setup* permite configurar os dados de entrada do Arduino, como os pinos de entrada e saída. E a função *loop* possibilita configurar o código que ficará em execução por um tempo indefinido, fazendo com que o programador crie regras para a interrupção de seu programa.

As linguagens de programação utilizam por padrão o *Hello World* como o primeiro programa para ilustrar aos iniciantes as suas sintaxes básicas. No ambiente Arduino IDE, o primeiro programa que é executado tem essa função de piscar um LED conforme o código descrito na Figura 11:

```
1 # define LED_PIN 13
2
3 void setup () {
4     pinMode (LED_PIN, OUTPUT); // habilita o pino 13 para saída (OUTPUT).
5 }
6
7 void loop () {
8     digitalWrite (LED_PIN, HIGH);  // liga o LED.
9     delay (1000);                  // espera 1 segundo (milissegundos).
10    digitalWrite (LED_PIN, LOW);   // desliga o LED.
11    delay (1000);                  // espera 1 segundo.
12 }
```

Figura 11 - Código do Arduino para piscar um LED.

Já o *hardware* Arduino é a base onde o código escrito é executado. O Arduino possui um *kit* de desenvolvimento que interpretam variáveis no ambiente e as transformam em sinal elétrico correspondente. Para realizar essa função, o Arduino utiliza sensores ligados aos seus terminais de entrada, e atua no controle ou no acionamento de algum outro elemento eletro-eletrônico conectado ao terminal de saída. O Arduino é utilizado como uma ferramenta de controle de entrada e saída de dados, que pode ser acionada por um sensor e após passar por uma etapa de processamento através de um microcontrolador, poderá acionar um atuador (Beppu e Fonseca, 2010; Margolis, 2011).

Alguns exemplos de sensores incluem interruptores, acelerômetros e medidores de distância por ultra-som. Já os atuadores são luzes e LEDs, alto-falantes, motores e *displays*. Os autores Beppu e Fonseca (2010) definem o Arduino como um computador, que têm como sensores de entrada, por exemplo, o mouse e o teclado, e de saída, impressoras e caixas de som, e são estes que fazem interface com circuitos elétricos, podendo receber ou enviar informações/tensões neles.

Conforme a Figura 12 é possível identificar os elementos principais do circuito através de um diagrama.

Figura 12 - Elementos do circuito Arduino.

Alguns modelos de Arduino utilizam memória *flash* com o intuito de armazenar a programação, o que faz com que ele seja logicamente programável. A linguagem nativa de desenvolvimento para a plataforma Arduino é baseada em C/C++.

A plataforma Arduino também pode se comunicar através de um canal serial com outras linguagens de programação que possuam essa característica tais como: *Android*, *Java, PHP, Perl, Batch, Shell, Ruby, JavaScript, TCL, Python.* Essas linguagens possuem bibliotecas que podem se comunicar com a porta serial do Arduino. Algumas linguagens, como por exemplo, o *flash* não tem a capacidade nativa de série. Então, a comunicação com o Arduino é feita através de um intermediário que funciona como um tradutor, permitindo a comunicação entre estes.

No mercado há uma variedade de placas oficiais que podem de usadas com o *software* do Arduino e uma ampla gama de Arduinos compatíveis com placas produzidas por membros da comunidade Arduino. As placas mais populares contêm um conector USB que é usado para fornecer energia e conectividade para *upload* do seu *software*. Na próxima seção, serão abordados os principais modelos de placas Arduino.

3.2 Componentes das placas Arduino

O projeto Arduino apresenta uma diversidade de design de placas para o desenvolvimento de *hardware* livre. Uma placa Arduino padrão é composta por um microcontrolador AVR Atmel, sendo pré-programado, com um *bootloader* que gerencia a inicialização do *chip*. Dessa forma, a transferência de dados é facilitada e realizada através do processo de *upload* dos programas do computador para o *chip* de memória *flash* da placa Arduino.

Além do microcontolador, as placas Arduino são compostas por outros componentes que auxiliam na programação e incorporação da placa em outros circuitos. A seguir será apresentada a definição desses componentes.

- Microcontrolador AVR Atmel: são microcontroladores de 8 bits, compostos pela tecnologia *Reduced Instruction Set Computer* (RISC, 2012) e arquitetura *Havard* (HAVARD, 2012) que consegue separar a memória de dados da memória de programas. Com isso, o microcontrolador AVR tem um barramento para dados e outro para programas que permite uma maior velocidade no tratamento dos mesmos.
- PWM (*Pulse-Width Modulation*): os modelos de placas Arduino não possuem saídas analógicas, porém estes oferecem saídas PWM que simulam saídas analógicas com tensão de 0V a 2,5V. A saída padrão do PWM oferece sinal digital que assume os níveis lógicos alto (um) e baixo (zero), com isso, é possível transformar a saída com sinal digital em saída com sinal analógico.
- FTDI (*Future Technology Devices International*): converte os níveis de tensão para realizar a comunicação serial através da porta USB com um computador. Normalmente os pinos de comunicação serial dos microcontroladores utilizam níveis de tensão denominados *Transistor-Transistor Logic* (TTL, 2012). Então, para realizar a comunicação serial através da porta USB com um computador é necessário converter esses níveis de tensão utilizando o *chip* FTDI.
- Cabeçalho ICSP (*In-Circuit Serial Programming*): é um método de gravação de dispositivos eficiente que permite gravar os códigos no circuito do dispositivo, ou seja, o usuário monta o seu circuito na placa e depois programa o dispositivo através de uma interface serial (ICSP, 2003).
- Cristal oscilador de 16MHz: permite criar um sinal elétrico com uma frequência que pode ser usada para medir o tempo conhecido como relógio de *quartzo* que garante o funcionamento ordenado e sincronizado nos circuitos dos equipamentos eletrônicos (Braga, 2012).
- Adaptador AC/DC (*Alternating Current/Direct Current*): a alimentação do Arduino pode ser feita através de uma conexão USB ou através de uma alimentação externa que pode ser o adaptador AC/DC.

3.3 Modelos de placas Arduino

Esta seção apresentará os principais modelos das placas Arduino, suas características e demonstrará as principais diferenças entre elas. O modelo utilizado para o desenvolvimento do estudo de caso desse projeto é o Arduino Uno.

- **Arduino *Uno***

O modelo *Uno* é uma placa Arduino baseada no microcontrolador ATmega328. Esse modelo tem 14 pinos digitais de entrada/saída, onde 6 pinos podem ser usados como saídas PWM e 6 como entradas analógicas. O modelo também possui um cristal oscilador de 16 MHz, uma conexão USB, um conector de energia, um cabeçalho ICSP e um botão de *reset* (Modelos de Arduino, 2012).

O Arduino *Uno* difere de outras placas na medida em que não utilizam o *chip* condutor FTDI USB para série. Em vez disso, ele apresenta o Atmega16U2 que pode ser programado como um conversor USB para serial. Com isso, sua alimentação pode ser feita através de uma conexão USB ou através de uma fonte de alimentação externa que pode ser um adaptador AC/DC ou ainda uma bateria. A fonte de energia é selecionada automaticamente.

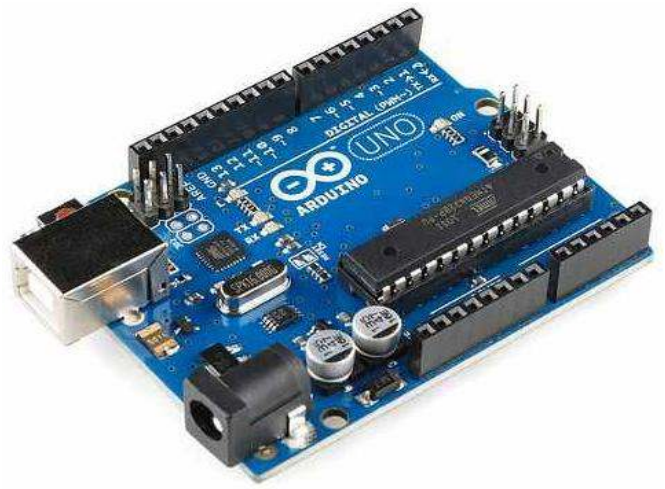


Figura 13 - Placa Arduino Uno

- **Arduino *Mega* 2560**

O Arduino *Mega* 2560 é uma placa baseada no microcontrolador ATmega2560. Esse modelo tem 54 pinos digitais de entrada/saída, dos quais 14 podem ser usados como saídas PWM, 16 como entradas analógicas, 4 UARTs que representam as portas seriais de *hardware*, um cristal oscilador de 16 MHz, uma conexão USB, uma conector de energia, um cabeçalho ICSP e um botão de *reset*. O *Mega* é compatível com a maioria das placas projetadas para o *Duemilanove* Arduino ou *Diecimila* (Arduino, 2012).

O Arduino *Mega* 2560 é uma atualização do Arduino *Mega*. A principal diferença entre o *Mega* e o *Mega* 2560 é o processador utilizado. O modelo *Mega* utiliza o ATmega1280 com 128KB de memória. Já o modelo *Mega* 2560 utiliza o ATmega2560, com 256KB de memória. As características remanescentes dos dois modelos são basicamente idênticas.

Em relação ao modelo *Uno*, a principal diferença também é o processador. O modelo ATmega2560 tem mais memória e mais periféricos do que o ATmega328 do *Uno*. A placa PCB *(Printed circuit board*) onde são soldados os demais componentes é maior no modelo *Mega* 2560, mas mantém as especificações da placa compatíveis com o Arduino padrão, adicionando três conectores de expansão adicionais ao longo do lado direito e prorroga o PCB por cerca de uma polegada de comprimento. O restante do circuito é idêntico ao Arduino *Uno* (Wheat, 2011).

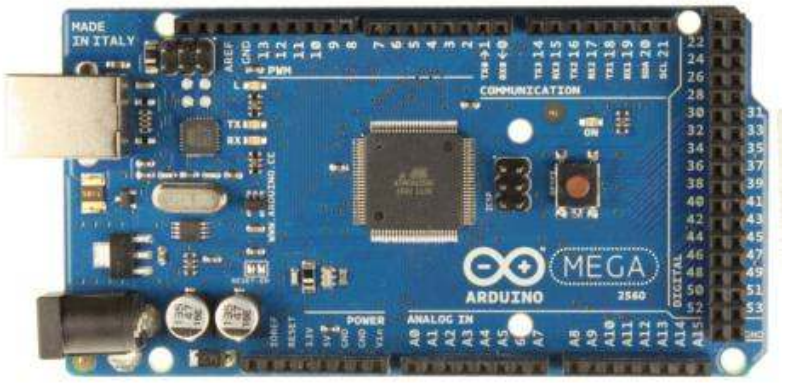


Figura 14 - Placa Arduino Mega 2560

- **Arduino ADK**

O Arduino *Android Development Kit* (ADK) é uma placa baseada no microcontrolador ATmega2560. Esse modelo tem uma interface USB para se conectar à celulares com *Android*, com base no IC MAX3421E que permite ao Arduino ADK conectar-se e interagir-se com qualquer tipo de dispositivo que possua uma porta USB. Além disso, esse modelo possue 54 pinos digitais de entrada/saída dos quais 14 podem ser usados como saídas PWM, 16 entradas analógicas, 4 portas seriais UARTs de *hardware* utilizadas para a comunicação serial de um ou mais computadores ou dispositivos periféricos. Além de um cristal oscilador de 16 MHz, uma conexão USB, um conector de energia, um cabeçalho ICSP e um botão de *reset* (Modelos de Arduino, 2012).

O moledo ADK é similar ao *Mega* 2560 e ao *Uno*. O mesmo apresenta um ATmega8U2 programado como um conversor USB para serial.

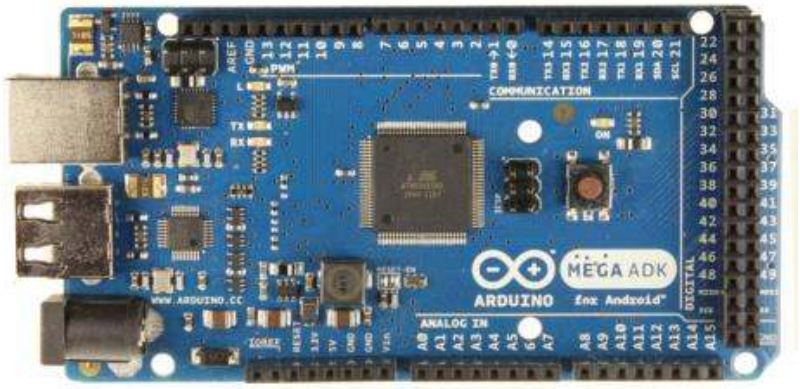


Figura 15 - Placa Arduino ADK

- **Arduino BT *(Bluetooth)***

O Arduino BT é uma placa baseada no microcontrolador ATmega168, mas atualmente é fornecido com o ATmega328. Esse modelo suporta a comunicação serial sem fio através do *bluetooth,* porém não é compatível com fones de ouvido *bluetooth* ou outros dispositivos de áudio.

O Arduino BT tem 14 pinos digitais de entrada/saída, dos quais 6 podem ser usados como saídas PWM e podem ser usados para reajustar o módulo WT11 que é baseado na versão 2.0 do *bluetooth*, sendo 3 vezes superior ao *bluetooth* 1.2 em relação a velocidade, além de consumir menos energia. Além de conter 6 entradas analógicas, um cristal oscilador de 16 MHz, um cabeçalho ICSP e um botão de *reset* (Modelos de Arduino, 2012).

A placa BT contém um convector de DC-DC que permite que ele seja alimentado com no mínimo 1,2V e um máximo de 5.5V. As tensões mais elevadas ou polaridade invertida na alimentação podem danificar ou destruir a placa.

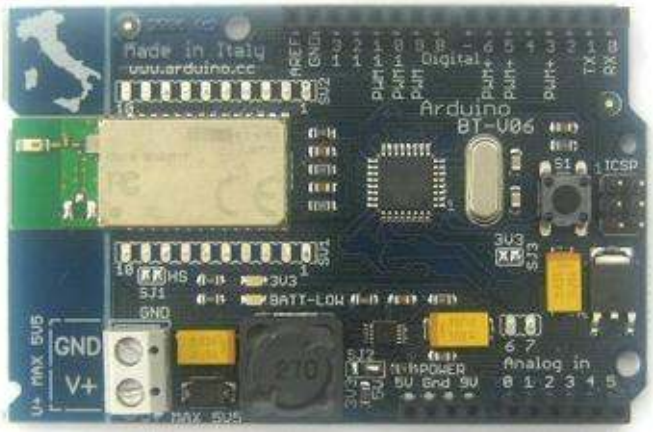


Figura 16 - Placa Arduino BT

- **Arduino *Duemilanove***

O Arduino *Duemilanove* é uma placa baseada no microcontrolador ATmega168 ou ATmega328. O modelo *Duemilanove* tem 14 pinos digitais de entrada/saída dos quais 6 podem ser usados como saídas PWM, 6 entradas analógicas, um cristal oscilador de 16 MHz, uma conexão USB, um conector de energia, um cabeçalho ICSP, e um botão de *reset* (Arduino, 2012).

O *Duemilanove* pode ser alimentado através da conexão USB ou com uma fonte de alimentação externa. A fonte de energia é selecionada automaticamente. Quando a conexão é externa, ou seja, não-USB, a energia pode ser fornecida através de um adaptador AC para DC ou uma bateria.

A placa pode operar com o fornecimento de energia de 7 a 12 volts. Se for alimentada com menos de 7V, a placa pode ficar instável e se for alimentada com mais de 12V, o regulador de voltagem pode super aquecer e danificar a placa.

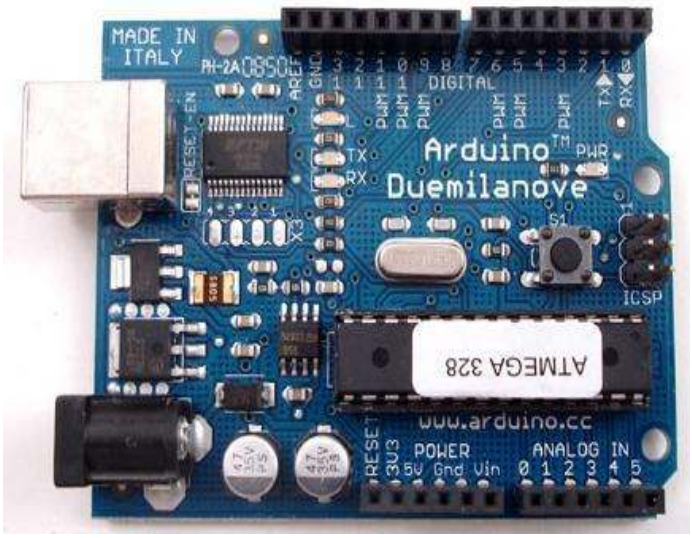


Figura 17 - Placa Arduino Duemilanove

- **Arduino *Pro***

O Arduino *Pro* é uma placa baseada no microcontrolador ATmega168 ou ATmega328. O modelo *Pro* tem duas versões, a de 3.3V/8 MHz e a de 5V/16 MHz. Esse modelo possui 14 pinos digitais de entrada/saída, dos quais 6 podem ser usados nas saídas PWM, 6 entradas analógicas, um interruptor, um botão de *reset*, furos para montagem de um conector de energia, um cabeçalho ICSP e um cabeçalho de alfinete. O cabeçalho de alfinete pode ser conectado a um cabo FTDI ou uma placa USB SparkFun para fornecer energia e comunicação ao *hardware* (Modelos de Arduino, 2012).

O Arduino *Pro* vem sem cabeçalhos pré-montados, permitindo o uso de vários tipos de conectores ou soldagem de fios diretos. A pinagem é compatível com placas de expansão para Arduino. As versões de 3.3V do *Pro* podem ser alimentadas com uma bateria.

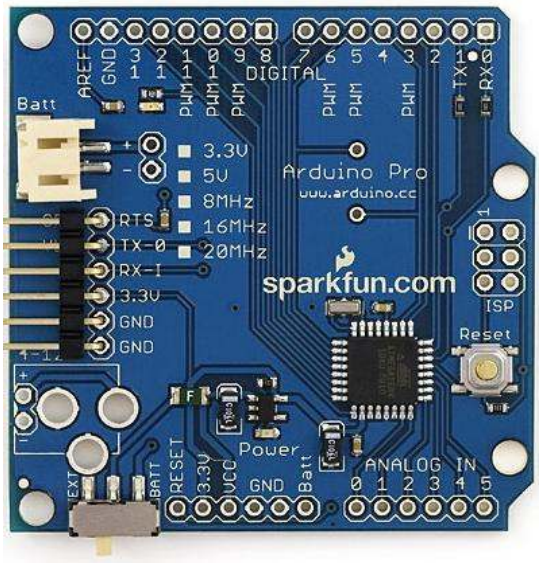


Figura 18 - Placa Arduino Pro

3.4 Vantagens e desvantagens do Arduino

Segundo Arduino (2012), as vantagens de desenvolver com Arduino são:

- baixo custo – as placas Arduino são relativamente baratas em comparação com outras plataformas de microcontroladores. A versão mais barata do módulo Arduino pode ser montada manualmente, e até mesmo os módulos pré-montados do Arduino atualmente chegam a custar menos de 50 dólares.
- ambiente multiplataforma - o *software* Arduino é executado em *Windows, Macintosh OSX*, e sistemas operacionais *Linux*. A maioria dos sistemas de microcontroladores são limitados para o sistema operacional *Windows*.
- ambiente de programação simples – os usuários têm facilidade de usar o ambiente de programação Arduino e criar os seus próprios projetos em um curto espaço de tempo, devido a sua flexibilidade.
- *open source* - o desenvolvimento do Arduino é aperfeiçoado por uma comunidade que divulga os seus projetos e seus códigos de aplicação, pois a concepção dela é *open-source*, ou seja, qualquer pessoa com conhecimento de programação pode

modificá-lo e ampliá-lo de acordo com a necessidade, visando sempre a melhoria dos produtos que possam ser criados aplicando o Arduino.

Em relação as desvantagens no desenvolvimento com Arduino percebe-se que a plataforma ainda não é utilizada em escala industrial, sendo o mesmo normalmente utilizado para pequenos projetos, criação de protótipos em nível acadêmico e projetos pessoais.

Após a abordagem dos conceitos de *software* livre, *hardware* livre e Arduino, os mesmo serão aplicados no estudo de caso descrito no próximo capítulo.

4 TECNOLOGIAS E ARQUITETURAS

O estudo de caso que será proposto no capítulo 5 desse trabalho foi desenvolvido utilizando a linguagem de programação *Python.* Através da biblioteca *pyserial* foi realizada a comunicação com a linguagem de programação C, nativa do Arduino. As próximas seções irão abordar os conceitos dessas tecnologias e arquiteturas.

4.1 Linguagem C

A linguagem C foi criada em 1972 por Dennis Ritchie no centro de Pesquisas da *Bell Laboratories*. Esta linguagem foi utilizada no projeto *Unix*, que até então era escrita em *assembly* (Unix, 2012). A linguagem C é uma linguagem compilada e possui características tais como: modularidade, portabilidade, estruturada, imperativa e procedural.

A linguagem de programação C é muito utilizada em desenvolvimento de microcontroladores. Os microcontroladores começaram a ser desenvolvidos usando a linguagem de programação *Assembly*. Embora essa linguagem tenha como característica o seu processamento rápido, a mesma apresenta algumas desvantagens como a montagem de um programa que consiste em mnemônicos, pois o aprendizado e a manutenção desse programa torna-se díficil e custoso. Além da linguagem C, os microcontroladores também podem ser programados usando outras linguagens de alto nível, como *Basic* e *Pascal* (Ibrahim, 2008).

As linguagens de alto nível são fáceis de aprender, pois sua estrutura faz com que essas se aproximem da linguagem humana, tornando a manutenção simples e facilitando o desenvolvimento de programas extensos e complexos. Os programas escritos nessas linguagens são convertidos para a linguagem *Assembly* através de um compilador (Nascimento, 2009).

Os compiladores de linguagem C, por exemplo, fazem a compilação dos programas em duas etapas, sendo essas transparentes ao programador. A primeira etapa da compilação, o código fonte escrito em C é transformado em código *Assembly*. Já na segunda etapa, é realizada a montagem, ou seja, é gerado o código binário com a ajuda de um *Assembler*, conhecido como programa montador que realiza a conversão dos mnemônicos em código de máquina para que o mesmo seja executado pelo microcontrolador (Lenz, 2012).

4.2 A Linguagem C e o ambiente Arduino

A linguagem nativa do ambiente de desenvolvimento do Arduino é a linguagem de programação C, que incorpora várias funções complexas de programação em comandos simples que torna o programa fácil de compreender. O Arduino possui um Ambiente de Desenvolvimento Integrado (IDE) que contém os projetos *Processing* e *Wiring* derivados da linguagem de programação Java no qual o IDE é escrito (Adams, 2011). A biblioteca *Wiring* tem como característica a capacidade de programar em C/C++ que permite que as operações de entrada e saída sejam criadas com facilidade. No ambiente Arduino IDE é definida duas funções para escrever um programa funcional. O método *setup*, utilizado para inicializar as configurações e o método loop que repete um bloco de comandos ou espera para que este seja desligado.

O *hardware* Arduino é baseado em um microcontrolador modelo Atmega, sendo este logicamente programável. Através da linguagem de programação C/C++ é possível a criação de programas, que, quando implementados fazem com que o *hardware* execute certas ações (Beppu e Fonseca, 2010).

4.3 Linguagem *Python*

A linguagem de programação *Python* foi lançada em 1991 por Guido van Rossum. A mesma possui uma estrutura de dados de alto nível, interpretada e imperativa. Além de uma abordagem simples e efetiva para a programação orientada a objetos. (Python, 2012)

O desenvolvimento de *software* livre utilizando linguagens dinâmicas comumente é realizado com processos de desenvolvimento ágeis. Entre essas linguagens, o *Python* se destaca como uma das mais populares e poderosas (Borges, 2010). O *Python* é considerado um *software* livre e o seu código-fonte está disponível sobre a licença GPL. Essa linguagem também pode ser incorporada em produtos proprietários, pois possui menos restrição.

Python é uma linguagem multi-plataforma, ou seja, o seu programa pode ser executado no sistema operacional *Windows* e sistemas *Unix-like*, como *Linux*, BSD e *Mac* OS X. O desenvolvimento da linguagem é gerenciado pela *Python Software Foundation* (PSF).

A sintaxe da linguagem *Python* é clara e concisa, favorecendo a legibilidade do código-fonte, tornando-a mais produtiva. A linguagem inclui diversas estruturas de alto nível, como por exemplo, listas, dicionários, data/hora e outras. Além de coleção de módulos prontos para uso e *frameworks* de terceiros que podem ser adicionados (Borges, 2010).

4.4 A linguagem *Python* e o ambiente Arduino

O desenvolvimento com a linguagem *Python* no ambiente Arduino é realizado através de uma biblioteca chamada *pyserial* que permite a comunicação com a porta serial do Arduino. Após a instalação da biblioteca *pyserial*, o código *Python* deve instanciar um objeto serial permitindo o envio de informações para executar as funções pré-definidas no Arduino conforme Figura 19.

```
1 import serial
2
3     porta = '/dev/ttyUSB0'
4     baud_rate = 9600
5     arduino = serial.Serial(porta, baud_rate)
6     arduino.write("H")
7     print arduino.read()
8     arduino.close()
```

Figura 19 - Código para enviar dados para a porta serial.

Na linha 1 é importada a biblioteca serial do *Python*. Já na linha 3 a porta para a comunicação serial configurada no Arduino é selecionada. A taxa de configuração de transferência dos dados é configurada na linha 4. Para realizar a comunicação serial com o Arduino é necessário instanciar um objeto serial como mostra a linha 5. Na linha 6 é enviado, como exemplo, o caractere “H” para a porta serial. A linha 7 exibe as informações recebidas na porta serial do Arduino e a linha 8 fecha a conexão serial.

Para que o Arduino possa executar suas funções através das informações fornecidas pelo código *Python*, há a necessidade da existência de um código embarcado para receber os dados provenientes do ambiente computacional externo conforme a Figura 20.

```
1  int ledPin = 13;
2  int incomingByte = 0;
3
4  void setup() {
5      pinMode(ledPin, OUTPUT);
6      Serial.begin(9600);
7  }
8
9  void loop() {
10     if (Serial.available() > 0) {
11         incomingByte = Serial.read();
12         if (incomingByte == 72){
13             digitalWrite(ledPin, HIGH);
14 }
15         else {
16             digitalWrite(ledPin, LOW);
17 }
18 }
```

Figura 20 - Código para receber dados da porta serial.

A linha 1 inicializa a variável ledPin com o valor “13” que corresponderá ao pino de saída digital 13 na placa Arduino. Já na linha 2 a variável *incomingByte* é inicializada com valor “0” (zero) e será responsável por armazenar a informação fornecida pelo Python enviada pela porta serial. A linha 4 define o método *setup*, que é responsável pela inicialização das variáveis. Na primeira linha desse método, a variável *ledPin* é passada como parâmetro para a função *pinMode* juntamente com o parâmetro “*OUTPUT*”, fazendo com que o pino 13 seja configurado como um pino de saída digital. Já na segunda linha é inicializada a comunicação serial, cujo o parâmetro “9600” corresponde a taxa de transferência dos dados que deve ser a mesma utilizada no código *Python*. Outro método utilizado no ambiente Arduino é o *loop* que ficará em execução indefinidamente conforme a linha 9 do código descrito na Figura 21. A primeira linha desse método é responsável por verificar se existe a comunicação serial para que o código possa verificar se existe informação na porta serial. Na segunda linha a variável *incomingByte* recebe os dados provenientes da porta serial que foram fornecidas através do sistema computacional. A terceira linha verifica se a variável *incomingByte* recebeu a letra

"H", que corresponde ao número 72 em código ASCII, fazendo com que o pino digital 13 acenda o *led* através do envio de sinal alto conforme a quarta linha do método *loop*. Se o caractere informado for diferente da letra "H", o pino digital 13 enviará um sinal baixo e apagará o *led* conforme a última linha do método.

5 ESTUDO DE CASO

O estudo de caso irá demonstrar o passo a passo do desenvolvimento de um protótipo de robô utilizando *software* e *hardware* livre. Através das tecnologias *Python* e Arduino pretende-se criar o protótipo de robô que receba comandos oriundos de um sistema computacional externo e os transforma em ações físicas.

O *hardware* livre Arduino pode ser operacionalizado de duas formas: uma autônoma e outra sincronizada com um dispositivo computacional. A autônoma é realizada embarcando um código na linguagem C, nativa do dispositivo, tornando-o independente. Porém desta forma o dispositivo não permite uma interação externa, ou seja, o Arduino não poderá receber dados externos que possam alterar o comportamento do código embarcado, sendo o Arduino o único responsável por todo o processamento. A segunda forma deverá ser realizada uma comunicação com um dispositivo externo, como por exemplo, um computador, *tablet*, celular, etc. Esta comunicação deverá ser feita utilizando uma linguagem que permita enviar e receber dados através da porta serial. Desta forma, os dados são enviados para o Arduino que conterá um código embarcado preparado para assumir comportamentos de acordo com esses dados. Essa segunda forma é vantajosa devido à comunicação ser realizada com um dispositivo externo, que poderá realizar o processamento mais custoso e informar ao Arduino os dados de que necessita para executar as funções que fora programado.

Nesse estudo de caso a comunicação será realizada de forma sincronizada com um dispositivo externo. O sistema computacional externo utilizado para os testes será um *notebook* com o sistema operacional *Linux*. Com o dispositivo computacional os dados são enviados para a porta serial do Arduino fazendo com que o protótipo de robô execute as funções definidas. É indispensável a implementação de um programa na linguagem C para embarcar no Arduino fazendo com que o mesmo receba a comunicação com o sistema computacional externo.

A linguagem de programação *Python* é utilizada com o objetivo de conter a maior parte de processamento necessário para guiar o protótipo de robô. Nesse contexto, qualquer demanda maior de processamento para guiar o protótipo de robô estaria localizado no sistema computacional externo, no caso o *notebook*. Porém, a aplicação é uma demonstração e o programa em *Python* é simples, que visa mostrar a funcionalidade de comunicação serial entre

a aplicação externa e o protótipo de robô. Para realizar a comunicação da linguagem *Python* com a porta serial do Arduino foi utilizada a biblioteca *pyserial*.

A parte mecânica do protótipo é composta de um chassi conforme a Figura 21, contendo dois motores de corrente contínua, duas rodas, uma esfera deslizante e uma bateria.

Figura 21 - Chassi do protótipo.

A bateria é utilizada para alimentar o controlador de motor e os motores, pois a alimentação fornecida pelo cabo USB não é suficiente, visto que a corrente dos motores somada a corrente do controlador de motor é superior a que é fornecida pelo cabo USB, ocasionando perda de performance dos dispositivos.

Já a parte eletrônica é composta por uma placa Arduino Uno, um controlador de motor Arduino *Motor Shield* (Figura 22), uma placa de expansão para o Arduino que contém uma ponte H, possibilitando controlar dois motores de corrente contínua ao mesmo tempo ou um motor de passo**,** além de tornar possível que o motor rode tanto para um sentido quanto o outro**.** Esta placa tem como característica duas saídas PWM para que a velocidade dos motores possa ser controlada.

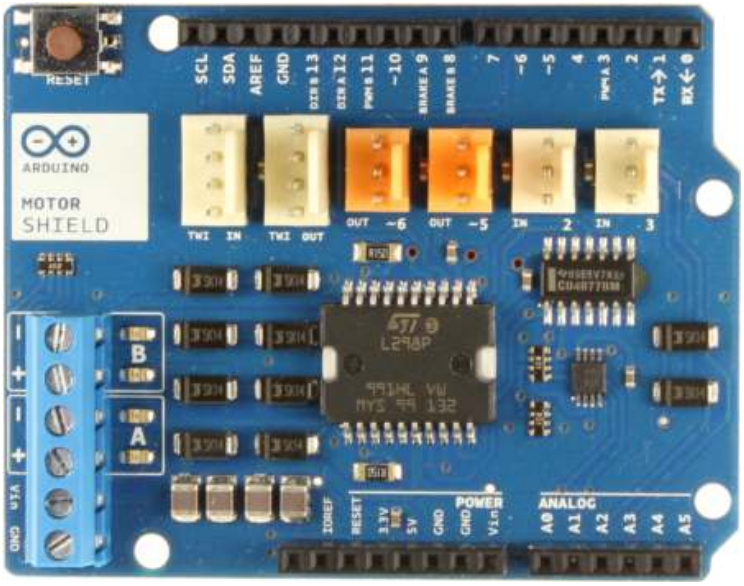


Figura 22 - Controlador de motor.

O código escrito na linguagem *Python* (Figura 23) fará com que o *hardware* externo faça o envio dos comandos, via porta serial, ao protótipo de robô que assumirá comportamentos, se movimentando para frente, para trás e para os lados. Os comandos são: f (*front*), b (*back*), l (*left*), r (*right*) e s (*stop*). O comando F aciona o protótipo para que se mova para frente, já o B fará o protótipo andar para trás. Para fazer o protótipo andar para a direita o comando R deve ser acionado e para a esquerda o comando L. O comando S fará com que o protótipo pare.

```
import serial
import time

class Arduino:
    def __init__(self):
        try:
            self.serial_connection = serial.Serial('/dev/ttyACM0', 9600)
        except:
            print "Fail to connect to the port /dev/ttyACM0!"

    def send_action_to_prototype(self, action):
        self.serial_connection.write(action)
        time.sleep(1)
```

```
def menu():
    print ":: Choose the direction to control the prototype ::"
    print "f - Move the prototype to front"
    print "b - Move the prototype to back"
    print "r - Move the prototype to right"
    print "l - Move the prototype to left"
    print "s - Stop the prototype"
    print "e - Exit the program"

arduino = Arduino()
menu()
action = raw_input('Choose the direction: ')
while action != 'e':
    arduino.send_action_to_prototype(action)
    menu()
    action = raw_input('Choose the direction: ')
```

Figura 23 - Código do Protótipo de robô na linguagem Python

O código embarcado no Arduino está escrito na linguagem de programação C (Figura 24) para receber os comandos enviados através da porta serial. As funções contidas nesse código estão descritas na seção Apêndice.

É comum que tecnologias livres sejam disponibilizadas em repositórios públicos, onde os interessados tenham acesso aos detalhes do projeto. Nesse sentido, os códigos desenvolvidos nesse trabalho, juntamente com a monografia na sua íntegra, estarão disponibilizados no *github* (Github, 2012), que é um repositório utilizado pela comunidade de software livre.

CONCLUSÕES

Foram apresentados os conceitos que envolvem *software* e *hardware* livre, abordando as licenças e os projetos que validam esses conceitos. Com o *hardware* livre é possível criar protótipos que são acessíveis, com baixo custo, flexíveis e de fácil utilização. Desta forma, pode-se perceber uma nova tendência no desenvolvimento de *hardware*.

O estudo de caso apresentado nesse trabalho abordou o desenvolvimento de um protótipo de robô utilizando como tecnologias a linguagem de programação *Python* e plataforma Arduino. Para o funcionamento do protótipo de robô, um *notebook* com a distribuição *Linux* foi utilizado como dispositivo externo para realizar a comunicação com o Arduino. Através da biblioteca *pyserial*, da linguagem *Python*, foi possível realizar a comunicação com a porta serial do Arduino. Os comandos definidos no programa *Python* permitiram o envio e o recebimento de dados para a porta serial do Arduino, onde o código em linguagem C foi embarcado no dispositivo fazendo com que o mesmo assumisse os comportamentos definidos no programa *Python*. Estes comportamentos possibilitaram ao protótipo de robô movimentar-se conforme os comandos recebidos.

O protótipo de robô desenvolvido atendeu as expectativas obtendo o resultado esperado e as ações do protótipo ocorreram de acordo com o que foi programado, validando o experimento.

Este trabalho contribui para a aquisição de conhecimento do leitor, fornecendo a base necessária, tanto em termos de conceitos teóricos sobre a plataforma Arduino, quanto aos conceitos práticos para iniciar seus primeiros projetos utilizando o *hardware* livre. Então, o presente trabalho servirá de referência para futuros estudos, pois aborda conceitos dessas tecnologias, detalhando os seus componentes e seus funcionamentos, além das definições de suas licenças, demonstrando como aplicá-las.

Com a disponibilização do código-fonte desse projeto, outras pessoas terão o acesso ao mesmo para estudá-lo, copiá-lo, modificá-lo, criar novas funcionalidades e distribuí-lo novamente.

O presente trabalho faz parte de um projeto maior que tem como objetivo criar um protótipo que seja autônomo com base no conceito de visão computacional. Esse protótipo será guiado por um sistema computacional móvel que processará as informações do ambiente e as enviará de forma serial para o Arduino fazendo com que o mesmo execute as funções

definidas. As possibilidades de desenvolvimento de novos projetos são grandes, utilizando como referência esse trabalho.

REFERÊNCIAS

ACKERMANN, John R. *Toward Open Source Hardware*. 2009.

ACOSTA, Roberto. *Open Source Hardware*. 2009.

ALECRIM. Emerson. *Software livre, código aberto e software gratuito:* as diferenças. 2011. Disponível em: http://www.infowester.com/freexopen.php. Acesso em: 05 de maio de 2012.

ARDUINO. Disponível em: http://arduino.cc/. Acesso em: 11 de janeiro de 2012.

BARBOSA, Luiz Felipe. *Proposta de utilização de software livre.* Rio de Janeiro. 2007. Disponível em: http://pt.scribd.com/doc/6688758/4/%E2%80%93-Licencas-Livres. Acesso em: 10 fevereiro de 2012.

BEAGLEBOARD. Disponível em: http://beagleboard.org/HARDWARE. Acesso em: 02 de maio de 2012.

BEPPU, Mathyan Motta; FONSECA, Erika G. Pereira. *Apostila Arduino*. Niterói, RJ. 2010.

BLENDER. Disponível em: http://www.blender.org/. Acesso em: 03 de maio de 2012.

BORGES, Luiz Eduardo. *Python para desenvolvedores*. 2. Ed. Rio de Janeiro: CIETEC, 2010.

BOWYER, Adrian. *RepRap*. Disponível em: http://www.kith-kin.co.uk/shop/reprap/. Acesso em: 24 de julho de 2012.

BRAGA, C. Newton. *Como funciona o cristal na eletrônica*. Disponível em: http://www.newtoncbraga.com.br/index.php/como-funciona/3081-art423.html. Acesso em: 28 de maio de 2012.

BUECHLEY, Leah. *LilyPad Arduino:* How an Open Source Hardware Kit is Sparking new Engineering and Design Communities. 2010.

CAMPOS, Augusto. *Conhecendo a licença Apache*. 2010. Disponível em: http://br-linux.org/2010/conhecendo-a-licenca-apache/. Acesso em: 07 de abril de 2012.

CERN. *Licença de Hardware Livre*. Disponível em: http://imasters.com.br/noticia/21394/software-livre/cern-lanca-iniciativa-de-hardware-livre. Acessado em: 08 de março de 2012.

DIPOLD, Rafael Draghetti. *Potencialidade econômica do software livre*. 2005.

ECLIPSE. Disponível em: http://www.eclipse.org. Acesso em: 30 de junho de 2012.

FALUDI, Robert. *Building Wireless Sensor Networks:* with ZigBee, XBee, Arduino, and Processing. 1. Ed. 2010.

FELLER, Joseph *et al. Perspectives on Free and Open Source Software*. 2005.

FREITAS, Solange Leandro de. TELES, Carlos Alberto M. de S. *Minimizando a Exclusão Digital - Utilização de Software Livre em Processos Educacionais*. In XII Seminário Nacional de Parques Tecnológicos e Incubadoras de Empresas. São Paulo: CIETEC, 2002.

FSF. *Free Software Foundation*. Disponível em: www.fsf.org. Acesso em: 14 de abril de 2012.

GITHUB. *DCMotorsArduinoPython*. Disponível em: https://github.com/michaelcaldas/dcmotorsarduinopython. Postado em: julho de 2012.

GNOME. Disponível em: http://www.gnome.org. Acesso em: 30 de junho de 2012.

GNU/Linux. Disponível em: http://www.gnu.org. Acesso em: 30 de junho de 2012.

GUESSER, Adalto Herculano. *Software livre e controvérsias tecnocientíficas*: uma análise sociotécnica no Brasil e em Portugal. Florianópolis, SC - Brasil. 2005.

GUIMARÃES Carlos, MAURER Henrique. *Iniciativa de uma plataforma robótica aberta de baixo custo utilizando metareciclagem*. Disponível em: http://www.santoangelo.uri.br/stin/Stin/trabalhos/12.pdf. Acesso em: 7 de abril de 2012.

HAVARD. *Arquitetura Havard*. Disponível em: http://pt.wikipedia.org/wiki/Arquitetura_Harvard. Acesso em: 08 de junho de 2012.

IBRAHIM, Dogan. *Advanced PIC microcontroller projects in C*: From USB to RTOS with the PIC18F Series. 2008.

ICSP. *In-Circuit Serial Programming (ICSP) Guide.* 2003.

JAMIESON, Peter. *Arduino for Teaching Embedded Systems. Are Computer Scientists and Engineering Educators Missing the Boat?* Miami, EUA. 2010.

LENZ, André Luis. *Linguagem para programar microcontroladores: Assembly, C ou Basic?* Disponível em: http://www.ebah.com.br/content/ABAAAAdWoAC/linguagem-programar-microcontroladores-assembly-c-basic. Acesso em: 22 de junho de 2012.

LICENCA BSD. Disponível em: http://pt.wikipedia.org/wiki/Licen%C3%A7a_BSD. Acesso em: 31 de março de 2012.

LICENCA DE *HARDWARE* LIVRE. Disponível em: http://imasters.com.br/noticia/21394/software-livre/cern-lanca-iniciativa-de-hardware-livre. Acesso em: 07 de abril de 2012.

LICENCA GPL. Disponível em: http://pt.wikipedia.org/wiki/Licen%C3%A7a_GPL. Acesso em: 11 de março de 2012.

LICENCA LGPL. Diponível em: http://pt.wikipedia.org/wiki/LGPL. Acessado em: 06 de abril de 2012.

LICENCA MIT. Disponível em: http://www.dirceupauka.com/entendendo-a-licenca-mit. Acesso em: 07 de abril de 2012.

LIMA, Paulo de Souza. *Que tal comprar um carro Open Source?* Disponível em: http://almalivre.wordpress.com/tag/hardware-livre/. Acesso em: 18 de julho de 2012.

LINGUAGENS DE PROGRAMAÇÕES COMPATÍVEIS COM O AMBIENTE ARDUINO. Disponível em: http://arduino.cc/playground/Main/InterfacingWithSoftware. Acesso em: 29 de maio de 2012.

LISTA DE PROJETOS. *List of open source hardware projects*. Disponível em: http://en.wikipedia.org/wiki/List_of_open_source_hardware_projects. Acesso em: 11 de julho de 2012.

MARGOLIS, Michael. *Arduino Cookbook*. 2. Ed. 2011.

MCNAMARA, Patrick. *Open hardware*. 2007. Disponível em: http://timreview.ca/article/76. Acesso em: 01 de julho de 2012.

MCROBERTS, Michael. *Beginning Arduino*. 1. Ed. 2010.

MODELOS DE ARDUINO. Disponível em: http://arduino.cc/en/Main/Hardware. Acesso em: 08 de maio de 2012.

NASCIMENTO, Erick Barros. *Aplicação da Programação de Microcontroladores 8051 utilizando Linguagem C*. 2009.

NETCRAFT. *Estáticas do uso de servidores Web*. Disponível em: http://news.netcraft.com/archives/category/web-server-survey/. Acesso em: 08 de julho de 2012.

OHL. *Open Hardware License*. Disponível em: http://www.ohwr.org/cernohl. Acesso em: 23 de maio de 2012.

OPENCORES. Disponível em: http://opencores.org/. Acesso em: 04 de maio de 2012.

OPEN HARDWARE LICENSE. *The TAPR Open Hardware License*. Disponível em: http://www.tapr.org/OHL. Acesso em: 08 de junho de 2012.

OPEN SOURCE. Disponível em: http://opensource.org/. Acesso em: 05 de maio de 2012.

OPENSPARC. Disponível em: http://opensparc.net. Acesso em: 03 de maio de 2012.

OSIER- MIXON, Jeffrey. *Hardware aberto:* como e quando funciona. Disponível em: http://imasters.com.br/artigo/19169/livre/hardware_aberto_como_e_quando_funciona/. Acesso em: 09 de janeiro de 2012.

PREMEAUX, Emery. EVANS, Brian. *Arduino Projects to Save the World*. 1. Ed. 2011.

PROJETO GNU. *Anúncio inicial do projeto GNU.* Disponível no site www.fsf.org. Acessado em: 30 de março de 2012.

PYTHON. Disponível em: www.python.org. Acesso em: 10 de abril de 2012.

PYTHON NO AMBIENTE ARDUINO. Disponível em: http://arduino.cc/playground/Main/InterfacingWithSoftware. Acesso em: 29 de maio de 2012.

RALLY FIGHTER. Disponível em: http://rallyfighter.com/. Acesso em: 18 de julho de 2012.

RAYMOND, Eric S. *A catedral e o bazar*. 1998.

RISC. *Reduced Instruction Set Computer*. Disponível em: http://pt.wikipedia.org/wiki/RISC. Acesso em: 08 de junho de 2012.

SANTOS, Ana Paula. *Saiba o que é hardware livre*. Disponível em: http://tiqx.blogspot.com/2011/09/saiba-o-que-e-hardware-livre.html. Acessado em: 23 de fevereiro de 2012.

SERRANO, Javier. *Licença de Hardware Livre*. Disponível em: http://www.ohwr.org/projects/cernohl/wiki. Acesso em: 23 de abril de 2012.

SERVIDOR APACHE. Disponível em: http://www.apache.org. Acesso em: 30 de junho de 2012.

SILVEIRA, Sérgio Amadeu da. *Software livre*: A luta pela liberdade do conhecimento. 1. Ed. São Paulo, SP – Brasil. 2004.

STALLMAN M. Richard. *Free Software, free society.* Boston, EUA. Create Space. 2009.

THOMPSON, Clive. *Build It. Share It. Profit. Can Open Source Hardware Work*? 2008.

TTL. *Transistor-Transistor Logic*. Disponível em: http://pt.wikipedia.org/wiki/Transistor-Transistor_Logic. Acesso em: 08 de junho de 2012.

UNIX. *História do Unix*. Disponível em: http://pt.wikipedia.org/wiki/UNIX. Acesso em: 17 de junho de 2012.

WHEAT, Dale. *Arduino Internal*. 1. Ed. 2011.

WOZNIAK, Stephen. *Homebrew and How the Apple Came to Be*. 1984.

APÊNDICE A – CÓDIGO EMBARCADO NO ARDUINO

```
/*

This code permit to control two DC motors interfacing with pyserial.

Below is shown the function of each pins to control the motors.

Function          Channel A     Channel B
Direction         Digital 12    Digital 13
Speed (PWM)       Digital 3     Digital 11
Brake              Digital 9    Digital 8
Current Sensing    Analog 0     Analog 1

Below is shown each attributes to control the motors.

The pwmA and pwmB set the speed of the motors
The dirA and dirB set the motors
The breakA and breakB set the breaks of motors

*/

const int pwmA = 3;
const int pwmB = 11;
const int breakA = 9;
const int breakB = 8;
const int dirA = 12;
const int dirB = 13;
char action = '  ';

void setup(){
 //Configuring the Channel A.
 pinMode(dirA, OUTPUT);
 pinMode(breakA, OUTPUT);

 //Configuring the Channel B.
 pinMode(dirB, OUTPUT);
 pinMode(breakB, OUTPUT);

 Serial.begin(9600);

}
void loop() {

 while (Serial.available()>0){
  action = Serial.read();
 }

 if(action == 'f'){
  move_to_front();
 }
 else if(action == 'b'){
  move_to_back();
 }
 else if(action == 's'){
  stop_motors();
 }
  else if(action == 'r'){
  move_to_right();
 }
  else if(action == 'l'){
```

```
  move_to_left();
 }

}
 /* This method allows to set the direction to the front.*/
 void move_to_front(){
  //Configuring the motor A to operate with partial speed.
  digitalWrite(dirB, HIGH);
  digitalWrite(breakB, LOW);
  analogWrite(pwmB, 96);

 //Configuring the motor B to operate with partial speed.
  digitalWrite(dirA, HIGH);
  digitalWrite(breakA, LOW);
  analogWrite(pwmA, 96);

  delay(1000);
 }

 /*This method allows to set the direction to the back.*/
 void move_to_back(){
  //Configuring the motor A to operate with partial speed.
  digitalWrite(dirA, LOW);
  digitalWrite(breakA, LOW);
  analogWrite(pwmA, 96);

 //Configuring the motor B to operate with partial speed.
  digitalWrite(dirB, LOW);
  digitalWrite(breakB, LOW);
  analogWrite(pwmB, 96);

  delay(1000);
 }

 /*This method allows to set the direction to the right.*/
 void move_to_right(){
  //Configuring the motor A to operate with partial speed.
  digitalWrite(dirB, HIGH);
  digitalWrite(breakB, LOW);
  analogWrite(pwmB, 96);

 //Stop the motor A.
  digitalWrite(breakA, LOW);
  analogWrite(pwmA, 0);

  delay(1000);
 }

 /*This method allows to set the direction to the left.*/
 void move_to_left(){
  //Configuring the motor A to operate with partial speed.
  digitalWrite(dirA, HIGH);
  digitalWrite(breakA, LOW);
  analogWrite(pwmA, 96);

 //Stop the motor B.
  digitalWrite(breakB, HIGH);
  analogWrite(pwmB, 0);

  delay(1000);
 }
```

```
/*This method allows to stop the motors.*/
void stop_motors(){
 digitalWrite(breakA, HIGH);
 digitalWrite(breakB, HIGH);
}
```