Capítulo 1

Desenvolvimento de Aplicações Ubíquas com Arduino e Raspbery Pi

Manoel Carvalho Marques Neto

Abstract

This course has a mission to explore the use of two hardware platforms that are very popular in the development of ubiquitous applications for building interactive multimedia environments: Arduino and Raspberry Pi. Arduino is an open-source prototyping platform with flexible software and hardware with low cost, created with the purpose of enabling the development and control of interactive systems. Raspberry Pi is a credit card size computer with all hardware integrated on a single board. The goal of creating the Raspberry Pi was to stimulate the teaching of basic computer science and electronic in schools. In both platforms we can plug a set of sensors and peripherals that allow to create an interactive multimedia environment by capturing context information, by using a communication network and by controling other electronic devices.

Resumo

Este curso tem a missão de explorar o uso de duas plataformas de hardware que são muito populares no desenvolvimento de aplicações ubíquas para a construção de ambientes multimídia interativos: Arduino e Raspberry Pi. O Arduino é uma plataforma de prototipagem open-source com software e hardware flexível, de baixo custo, criado com o objetivo de permitir o desenvolvimento e o controle de sistemas interativos. O Raspberry Pi é um computador do tamanho de um cartão de crédito com todo o hardware integrado em uma única placa. O objetivo da criação do Raspberry Pi foi estimular o ensino de ciência da computação básica e eletrônica em escolas. Em ambas as plataformas é possível acoplar um conjunto de sensores e periféricos que permitem montar um ambiente multimídia interativo através da captura de informação de contexto, da comunicação em rede, do controle de outros dispositivos eletroeletrônicos, etc.

1.1. Introdução

Ambientes multimídia interativos são aqueles nos quais a computação é usada para melhorar de forma imperceptível as atividades comuns do dia a dia. Uma das forças motrizes do interesse emergente nestes ambientes é tornar os computadores não apenas verdadeiramente amigáveis ao usuário, mas também essencialmente invisíveis para ele. Atualmente, o computador pode ser considerado um eletrodoméstico que se integra cada vez mais à vida das pessoas como um acessório diário para atividades profissionais e sociais. A tendência é que essa integração aumente mais ainda, graças ao desenvolvimento de interfaces e hardwares que "entenderão" os usuários e não serão apenas "tradutores", de seus comandos. Estas não são ideias novas. Na verdade elas surgiram no final do anos 80 e são a base da "Computação Ubíqua" (UbiComp) [de Araujo 2003, Krumm 2009, Kaufmann and Buechley 2010].

A UbiComp, em seus vários desdobramentos e aplicações, é considerada por muitos como o novo paradigma da Computação para o século XXI. Ela é a área da computação que estuda o acoplamento do mundo físico ao mundo da informação e fornece uma abundância de serviços e aplicações, permitindo que usuários, máquinas, dados e objetos do espaço físico interajam uns com os outros de forma transparente. O tema é considerado um dos grandes desafios da pesquisa em Computação pela *National Science Foundation* (NSF) [Armstrong et al. 2001] e está também presente no relatório Grandes Desafios da Pesquisa em Computação no Brasil 2006-2016 [Carvalho et al. 2006], publicado pela Sociedade Brasileira de Computação (SBC).

Pesquisas em computação ubíqua estão sendo realizadas em tópicos como: acesso básico a qualquer tipo de dispositivo sem fio, suporte à mobilidade na rede de forma transparente, segurança, tratamento de contexto, uso eficiente de energia, apresentação de conteúdo multimídia, etc. Este trabalho tem o seu foco na construção de ambientes interativos inteligentes. Nestes ambientes, a ideia fundamental é criar formas de evitar que o usuário necessite ir até ao computador/dispositivo, fazendo com que diversos deles funcionem à distância. O uso de plataformas que permitam integrar os dispositivos que compõem estes ambientes é um dos pontos fundamentais para sua criação. Atualmente existem algumas opções para preencher esta lacuna. Este texto dá enfase a duas delas, os projetos Arduino [Mellis and Buechley 2012, Vaughn 2012] e Raspberry Pi [Trapp 2013, Wirth and McCuaig 2014].

O Arduino e o Raspberry Pi são plataformas amplamente utilizadas por profissionais que têm interesse no campo das aplicações ubíquas. Elas possuem interfaces básicas para criação de pequenos projetos ou para aqueles que devem ser alimentados com bateria. Ambas usam a linguagens de programação de alto nível que são bastantes difundidas (C/C++, Phyton e Java). O Arduino, por ser uma plataforma mais antiga, tem uma grande rede de colaboradores ativos na web. O Raspberry é uma plataforma mais recente (lançada em 2012), tem um processador mais rápido que o Arduino, já vem pronto para acessar a web e possui mais interfaces que o Arduino.

Tanto Arduino quanto o Raspberry permitem o desenvolvimento de diversos tipos de projetos. Por exemplo: automação residencial (Ligar e desligar dispositivos elétricos, controle remoto para TV, aparelhos de Ar-condicionado, etc.), tocador de *audio books*, leitor de livros eletrônicos entre outros. Alguns exemplos de projetos podem ser encon-

trados no Youtube e em outros *sites* na Web [1] [2] [3] [4].

Este artigo apresenta as primeiras noções de como desenvolver aplicações ubíquas. O texto traz exemplos práticos com o uso de sensores que permitem controlar dispositivos comumente encontrados em um ambiente multimídia interativo. Esses conceitos introdutórios são um importante alicerce para uma gama de aplicações ubíquas em áreas distintas. O texto está organizado da seguinte forma: a seção 1.2 apresenta a computação ubíqua mostrando suas principais definições e princípios. A seção 1.3 apresenta o conceito de Ambientes Interativos Inteligentes e lista alguns dos principais projetos encontrados na literatura. A seção 1.4 apresenta as duas principais plataformas de desenvolvimento para aplicações ubíquas. Por fim a ultima seção apresenta as conclusões deste texto.

1.2. Computação Ubíqua: Definição, Princípios e Tecnologias

O termo Computação Ubíqua foi definido pela primeira vez por Mark Weiser [Weiser 1991] no final dos anos 80. Nesta época, Weiser previa um aumento das funcionalidades e da disponibilidade de serviços de computação para os usuários finais e, por outro lado, ele previa uma diminuição da visibilidade destes serviços. Para Weiser, a computação não seria exclusividade de um computador. Ele acreditava que no futuro existiriam diversos dispositivos diferentes conectados entre si. Em uma época que os usuários usavam PCs (*Desktops*) e para isso precisavam de conhecimento para operar um computador, Weiser apostou em um futuro onde foco dos usuários seria a tarefa em si, e não a ferramenta utilizada. Desta forma, eles usariam a computação sem perceber ou necessitar de conhecimentos técnicos específicos [Weiser 1994].

O passar do tempo demonstrou que a aposta de Weiser foi acertada. Para Weiser, [Weiser and Brown 1997] evolução da computação passou por duas eras até chegar a computação ubíqua. A primeira delas foi denominada era do mainframe onde muitas pessoas compartilhavam um mesmo computador. A segunda era foi a do PC onde cada computador era usado por uma pessoa. Atualmente, a evolução dos sistemas de informação distribuídos, a ampliação das opções de tipos conexões a rede, a computação móvel e os diversos tipos de aplicações sobre dispositivos computacionais não convencionais, são apenas alguns dos exemplos que permitem afirmar: a Computação Ubíqua (a terceira era) já é uma realidade.

1.2.1. Conceitos

Termos como computação ubíqua, computação pervasiva, computação nomádica, computação invisível, computação móvel e outros tantos, têm sido usados muitas vezes como sinônimos, embora sejam diferentes conceitualmente e empreguem diferentes ideias de organização e gerenciamento dos serviços computacionais. Na medida em que cada área evolui, esses conceitos vão sendo melhor compreendidos e suas definições se tornam

[1] https://www.youtube.com/watch?v=drTwrd6f3QU

[2] https://www.youtube.com/watch?v=FH_wI43xBw4

[3] https://www.youtube.com/watch?v=rqvo8RSH-BI&list=UUE0y2VBtNlh8fSvzWhyhHiQ

[4] http://www.itpro.co.uk/mobile/21862/raspberry-pi-top-10-projects

mais claras. Esta seção apresenta os principais conceitos necessários para compreensão da UbiComp além de apresentar alguns exemplos de projetos encontrados na literatura.

1.2.1.1. Computação Móvel

A computação móvel baseia-se na capacidade de um usuário carregar/mover (fisicamente) serviços computacionais para onde quer que ele se mova. Neste contexto, o computador torna-se um dispositivo sempre presente que expande a capacidade de um usuário utilizar os serviços que ele oferece, independentemente de sua localização. Combinada com a capacidade de acesso a rede, a computação móvel tem transformado a computação numa atividade que pode ser levada para praticamente qualquer local.

Uma importante limitação conceitual da computação móvel é que o modelo computacional utilizado na maioria das aplicações não muda enquanto os usuários se movem. Isto significa que um dispositivo não é capaz de obter informação sobre o contexto físico no qual a computação ocorre, e, como consequência, também não consegue se adequar ao novo contexto corretamente. Uma solução para acomodar a mudança de contexto seria passar aos usuários a responsabilidade de controlar e configurar manualmente uma aplicação/dispositivo na medida em que ele se movem. Contudo, esta solução não é bem aceita pela maioria dos usuários [de Araujo 2003, Krumm 2009]. Esta limitação foi uma das inspirações para a Computação Pervasiva.

1.2.1.2. Computação Pervasiva

O conceito de computação pervasiva implica que o computador está embarcado no ambiente de forma invisível para o usuário [Krumm 2009]. Nesta concepção, o computador tem a capacidade de: i) obter informação do ambiente no qual ele está embarcado e ii) utilizá-la para construir, dinamicamente, modelos computacionais que permitem controlar, configurar e ajustar a aplicação para melhor atender as necessidades de um dispositivo ou usuário. Para que isso seja possível, o ponto fundamental é a capacidade dos computadores poderem agir de forma "inteligente" no ambiente no qual os usuários se movem. Este ambiente é normalmente povoado por sensores e serviços computacionais.

1.2.1.3. Computação Ubíqua

Como pode ser visto na Figura 1.1, a UbiComp pode ser definida como uma área da computação posicionada entre a Computação Móvel e a Computação Pervasiva [de Araujo 2003, Krumm 2009]. A palavra Ubíquo é um adjetivo de origem no Latim (ubiquu) que significa "aquilo está ao mesmo tempo em toda a parte". A computação ubíqua beneficia-se dos avanços da computação móvel e da computação pervasiva e surge da necessidade de se integrar mobilidade com a funcionalidade da computação pervasiva. O termo computação ubíqua será usado aqui como uma junção da computação pervasiva e da computação móvel. A justificativa de se realizar uma diferenciação desses termos é que um dispositivo que está embutido em um ambiente, não necessariamente é móvel.

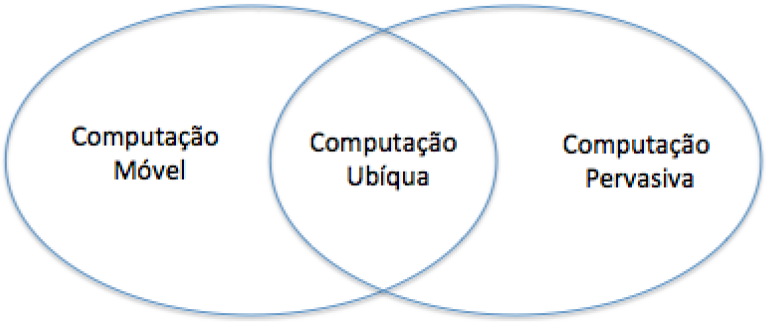

Figura 1.1. Relação entre as computações Móvel, Pervasiva e Ubíqua

1.3. Ambientes Interativos

Um dos requisitos necessários à UbiComp é tornar o processo de comunicação entre humanos e computadores o mais natural possível. Para isso, o desenvolvimento de Interfaces Naturais é ponto fundamental [Alisi et al. 2005]. Alguns trabalhos já realizados apostam no desenvolvimento de interfaces baseadas no reconhecimento da voz e da escrita [Hunt and Schalk 1996, Plamondon and Srihari 2000]. A Computação Ubíqua vai além e inspira o desenvolvimento de aplicações que estimulam a interação física entre humanos e computadores através de interfaces diferentes do mouse e do teclado. Para isso, o desafio dos programadores é criar interfaces capazes de reconhecerem não apenas a fala e a escrita, mas também os gestos, expressões e principalmente aliar todos estes dados ao contexto das funcionalidades disponíveis nos diversos componentes de um ambiente.

A partir de interfaces mais transparentes, o usuário pode se concentrar apenas nos processos de comunicação e no compartilhamento de informações. Estas interfaces são a base da construção de ambientes que procuram criar formas de evitar que o usuário necessite ir até o computador/dispositivo, fazendo com que diversos dispositivos funcionem à distância. Estes ambientes são chamados de "Ambientes Interativos" [Dey et al. 2000]. A criação dos ambientes interativos é feita a partir da combinação de diversos elementos diferentes. Entre eles pode-se destacar:

- **Interfaces Hands-Free**: Tecnologias de Reconhecimento de Voz, *Liveboards*, *Pads*, *Tabs*, entre outras que, quando utilizadas em conjunto, permitem a uma pessoa se mover pelo ambiente enquanto permanece em contínua interação com o computador;
- **Consciência de Contexto**: Sensores distribuídos em um ambiente que podem detectar o que acontece ao seu redor. Por exemplo, o que as pessoas em uma sala de reunião estão fazendo ou dizendo ? Qual é o índice de luminosidade ? Qual e temperatura da sala?
- **Ambientes Inteligentes**: É o conjunto de tecnologias que permitem o entendimento automático do contexto (**Consciência de Contexto**), ativando instruções ou respondendo a comandos pré-programados, mesmo sem intervenção explícita do

usuário. Por exemplo, computadores podem identificar a presença humana, desligando luzes e dispositivos na ausência dos mesmos, ou ativando o aquecedor ou ar-condicionado na temperatura ideal para cada usuário.

1.3.1. Exemplos de Projetos

O avanço da tecnologia no desenvolvimento de dispositivos, nos protocolos de comunicação, e a na miniaturização dos componentes já permitem a criação de aplicações práticas para as ideias de Weiser. Alguns projetos hoje já são realidade enquanto outros são promessas ou ainda estão em fase de protótipo. Esta seção apresenta alguns exemplos de projetos encontrados na literatura.

Desde 1988 o *Xerox Palo Alto Research Center* (PARC) vem pesquisando e desenvolvendo soluções de UbiComp. A partir de 1990 alguns protótipos foram desenvolvidos [Krumm 2009] e publicados. Entre eles pode-se destacar:

- **Liveboard** - A ideia desta solução era ser um quadro-negro eletrônico, sensível ao toque e que armazenasse informações inseridas através de uma caneta. Hoje esse a ideia deste produto já é realidade em escolas de ponta;
- **Pad** - Este protótipo, já ultrapassado tecnologicamente, é um notebook com microfone e caneta eletrônica acoplada, com comunicação por rádio a 240kbps (um avanço para a época). É um dispositivo fixo e sem mobilidade;
- **Tab** - Pequeno dispositivo portátil com tela sensível ao toque para a entrada de informações. Era ligado automaticamente quando o usuário interagia. Através de conexões infra-vermelho se comunicava com outros dispositivos. Pode-se dizer que este protótipo foi o avô da agenda eletrônica.

Casas onde a fechadura da porta principal é acionada pela íris do dono, as cortinas e as luzes são acionadas pela voz e o aquecedor ou ar-condicionado é ativado pelo celular não são mais ficção-científica. A tecnologia que possibilita a construção deste tipo de ambiente já existe e é denominada de *Smart House* [Chan et al. 1995]. Os primeiros esforços para conceber uma *Smart House* foram feitos pela Microsoft em um projeto chamado de *Microsoft Easyliving* [Shafer et al. 1998]. Ele foi criado pelo grupo de pesquisa em computação ubíqua da empresa e visava desenvolver/aplicar tecnologias para a construção de ambientes inteligentes.

O *Easyliving* usava câmeras ligadas a um PC que calculava a imagem em profundidade e detectava a presença de um determinado usuário. Este usuário era monitorado no ambiente, sem o auxilio de sensores adicionais. Se um ambiente estivesse vazio as luzes, aquecimento e outros equipamentos poderiam ser desligados automaticamente. Além disso, no *Easyliving*, cada pessoa que entrava no laboratório recebia uma identidade provisória e podia acessar um computador ou dispositivo com uma senha ou leitor biométrico. Se esta pessoa mudasse de sala e acessasse outro dispositivo, sua tela era automaticamente transferida do dispositivo anterior para o tela do computador em uso.

O *Georgia Institute of Technology* (Georgia Tech) também desenvolveu alguns projetos através do grupo de pesquisa de novos ambientes computacionais (denominado

Future Computing Enviroments). Entre eles, merecem destaque o *The Aware Home Research Initiative* e o *Eclass* [Essa 2000]. O primeiro é muito semelhante ao *Easyliving*, e visava construir um modelo computacional para uma casa interativa e consciente de seus moradores, da sua localização e das atividades dos seus usuários. Por exemplo: identificar quando uma pessoa estiver lendo jornal, vendo TV, etc. O *Eclass* pesquisava o impacto da Computação Ubíqua na educação, desenvolvendo técnicas e softwares para o uso de *Liveboards* nas salas de aula.

Outra classe de exemplos de aplicações ubíquas são os guias turísticos eletrônicos. Eles trabalham normalmente com informações de localização e utilização de dispositivos portáteis para conferir maior mobilidade aos usuários. Os principais objetivos dessa classe de aplicações são registrar os locais visitados e identificar a posição atual do usuário dentro de um espaço de interação como campus universitário, museu, etc. Por exemplo, o *CampusAware* [Burrell et al. 2002] é um sistema de turismo que utiliza informações de localização para fornecer serviços aos usuários. O sistema permite que o usuário faça anotações em seus dispositivos sobre os locais que estão visitando, fornecendo também informações de como encontrar determinados locais de interesse e a posição atual do usuário. O *CampusAware* possui um repositório central com três bancos de dados para informações contextuais e sociais, como as anotações dos usuários e locais visitados. Ele utiliza o GPS para obtenção de informações de localização em ambientes abertos.

O sistema *Rememberer* [Fleck 2002] é parte do projeto *Cooltown* [Barton et al. 2001], dirigido pelo *Hewlett-Packard Laboratories*. Seu objetivo é capturar experiências pessoais e estimular discussões ou outras formas de interação pessoal através de dispositivos portáteis sem fio. Para capturar informação de localização em recintos fechados, esse sistema utiliza RFID (*Radio Frequency Identification*) [Ni et al. 2004]. Um exemplo de aplicação desse sistema é a visita a museus, onde câmeras fotográficas registram interações dos usuários, e disponibiliza essas informações na ordem em que os locais foram visitados, permitindo ainda que o usuário faça anotações sobre os pontos visitados.

1.4. Plataformas de Desenvolvimento de Sistemas Ubíquos

O desenvolvimento de aplicações ubíquas faz parte do processo de criação de ambientes interativos. Para isso, um dos requisitos fundamentais é uso de sensores que permitem, entre outras possibilidades, transformar utilização desses ambientes a partir de interfaces mais transparentes. Atualmente, existem algumas plataformas que permitem inserir e controlar diversos tipos de sensores. Esta seção apresentada e detalhada duas delas, as plataformas Arduino e a Raspberry Pi.

1.4.1. Arduino

O Arduino [Banzi 2008, Brunvand 2013] é uma plataforma que populariza o conceito de hardware livre. Assim como o software, o hardware *open source* é desenvolvido de forma aberta, sem patentes, e seu projeto pode ser recriado de diferentes formas. Criado em 2005, o Arduino surgiu como um projeto para estudantes. Aprender eletrônica era caro: um microcontrolador custava 100 euros. Por isso estudantes do Instituto de Design de Interação de Ivrea, na Itália, decidiram fazer sua própria placa. Buscaram colaboradores e, assim, criaram uma tecnologia eficiente e acessível, compatível com Windows, Mac e

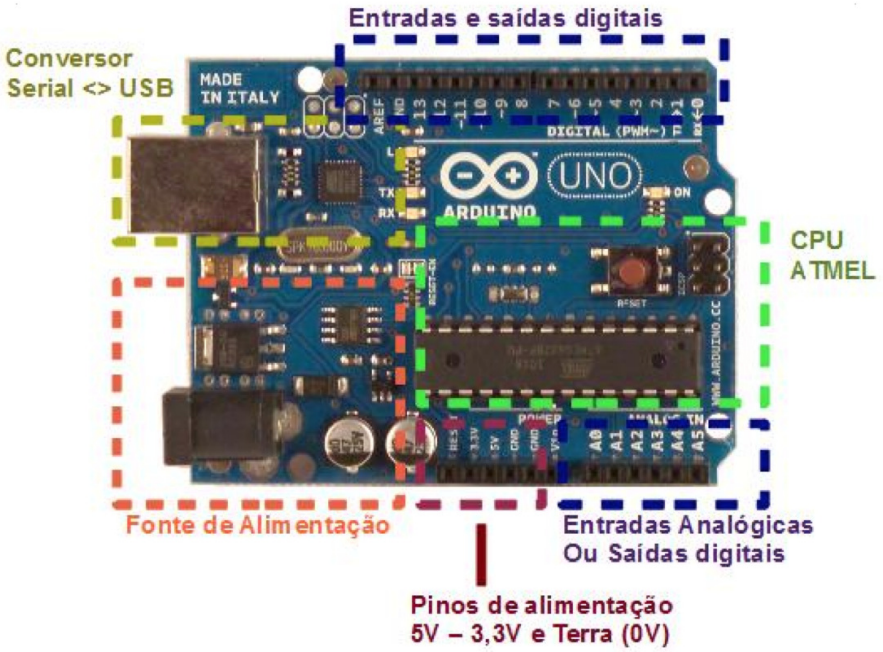

Figura 1.2. Arquitetura de hardware do Arduino

Linux.

A revolução do Arduino vai além do hardware. Seu software transforma uma linguagem relativamente fácil para humanos (C) em outra língua, mais difícil para homens, porém mais fácil para o hardware entender. Isso simplifica a criação de comandos para a placa, que vai conversar com o ambiente via sensores e pode ser conectada a outros objetos que vão desde lâmpadas, aparelhos de ar-condicionado até a um celular Android/IOS.

É importante ressaltar que os exemplos apresentados neste texto são apenas uma pequena amostra das possibilidades de uso do Arduino. A adoção do conceito de hardware aberto e *open source* motiva aqueles que criam projetos a contribuir com funções e bibliotecas para o Arduino. É conhecimento sobre conhecimento, o mesmo princípio do software livre.

1.4.1.1. Hardware

Existem dezenas de tipos de Arduinos (placas) com tamanhos, quantidade de memória e interfaces diferentes[5]. Para este artigo será utilizado o modelo "Arduino UNO". O hardware do Arduino é composto dos blocos (ver Figura 1.2) a seguir. É importante ressaltar que não é necessário conhecimento avançado em eletrônica para usar a plataforma. Porém, para aqueles que têm interesse em aprofundar os seus conhecimentos nesta área exitem diversos materiais disponíveis que podem ajudar, como por exemplo em [Henrique B. Filho 2012].

- **Fonte de Alimentação**: Responsável por receber a energia de alimentação externa,

[5]`http://arduino.cc/en/Main/Products`

que pode ter uma tensão de no mínimo 7 Volts e no máximo 35 Volts e uma corrente mínima de 300mA. A fonte filtra e depois regula a tensão de entrada para duas saídas: 5 Volts e 3,3 Volts. O requisito deste bloco é entregar as tensões de 5 e 3,3 Volts para que a CPU e os demais circuitos funcionem.

- **Núcleo CPU**: O núcleo de processamento de uma placa Arduino é um chip que possui todo hardware para obter dados, processá-los e devolvê-los para o mundo externo. Isso inclui uma CPU, memória (RAM e ROM), uma unidade de processamento de aritmética e os dispositivos de entrada e saída. Os desenvolvedores do Arduino optaram em usar a linha de microcontroladores da empresa ATMEL. A linha utilizada é a ATMega. Existem placas Arduino oficiais com diversos modelos desta linha, mas os mais comuns são as placas com os chips ATMega8, ATMega162 e ATMega328p. Esses modelos diferem na quantidade de memória de programa (ROM) e na configuração dos módulos de entrada e saída disponíveis.

- **Entradas e Saídas**: O número e as opções de entrada e saída de um Arduino dependem do microcontrolador (chip) usado. Por exemplo, o ATMega8 possui 28 pinos de conexões elétricas. É através desses pinos que é possível acessar as funções do microcontrolador, enviar dados para sua memória e acionar dispositivos externos. Os pinos são subdivididos em:

  - 14 pinos digitais de entrada ou saída (programáveis);
  - 6 pinos de entrada analógica ou entrada/saída digital (programáveis);
  - 5 pinos de alimentação (gnd, 5V, 3,3V);
  - 1 pino de reset;
  - 2 pinos para conectar o cristal oscilador;

  Os dois primeiros itens da lista são os pinos úteis, disponíveis para um programador utilizar. É através destes pinos que o Arduino é acoplado aos sensores que permitem capturar informações do ambiente. Dentre os 14 pinos de entrada/saída digitais existem 2 pinos que correspondem ao módulo de comunicação serial USART (*Universal Synchronous Asynchronous Receiver Transmitter*) [Crowe et al. 1990]. Esse módulo permite comunicação entre um computador (por exemplo) e o Arduino (ver seção 1.4.1.2). Todos os pinos (digitais ou analógicos) possuem mais de uma função. Os pinos podem ser de entrada ou de saída, alguns podem servir para leituras analógicas e também para entrada digital. Os papéis de cada um deles são definidas no código dos possíveis programas que podem ser executados no Arduino.

- **Entradas Digitais**: Existem 20 pinos que podem ser utilizados como entradas digitais. São 14 pinos digitais mais 6 pinos analógicos, podem ser programados para serem entradas digitais. Quando um pino é programado para funcionar como entrada digital, usa-se um comando que, ao ser executado, efetua a "leitura" da tensão aplicada a ele. Então, após a execução deste comando, é possível saber se o pino encontra-se em um estado "alto" ou "baixo" (ligado ou desligado). Do ponto de vista elétrico, o programa pode saber se um pino está alimentado com 0 (zero) ou 5

Volts. Essa função é utilizada geralmente para identificar se um botão está pressionado, ou se um sensor está capturando alguma informação do ambiente. Note que a função de entrada digital apenas entrega os valores 0 ou 1 (sem tensão ou com tensão). Não é possível saber quanto de tensão está sendo aplicada no pino. Para isso é preciso usar as entradas analógicas.

- **Entradas Analógicas**: Existem 6 entradas analógicas. Ao contrário de uma entrada digital, que informa apenas se existe ou não uma tensão aplicada em seu pino, a entrada analógica é capaz de medir a tensão aplicada. Através da entrada analógica, é possível utilizar sensores que convertem alguma grandeza física em um valor de tensão que depois é lido pela entrada analógica.

- **Saídas Digitais**: Com uma saída digital é possível fazer com que um pino apresente apenas dois tipos de valores (0 ou 5 volts, 1 ou 0, etc.). A partir de um pino programado como saída digital, pode-se por exemplo: acender um led, ligar um relé, acionar um motor, etc. Pode-se programar o Arduino (UNO) para no máximo 20 saídas digitais. Porém é possível utilizar um ou mais pinos para controlar um bloco de outros pinos e isso permite aumentar este limite de uso de pinos digitais.

Além dessa estrutura, o Arduino oferece o conjunto de placas de expansão de hardware que encaixam na estrutura da placa principal denominada de *Shields*. Os circuitos contidos nos diversos *Shields* contém uma eletrônica que adiciona funções que a placa principal não possui. Existem diversos *Shields*, com diversas funções[6]. Eles podem servir como entrada/saída e permitem, por exemplo, fazer com que o Arduino se comunique numa rede *Ethernet* ou via USB com um celular Android. Alguns *Shields* possuem circuitos integrados prontos para controle de outros dispositivos sem que seja necessário lidar com os detalhes de eletrônica envolvidos.

1.4.1.2. Pinos Especiais

Alguns pinos do Arduino possuem características especiais que podem ser usadas a partir das funções de um software codificado por um programador. São eles:

- **PWM - *Pulse Width Modulation*** [Barr 2001]: Tratado como saída analógica, na verdade é uma saída digital que gera um sinal alternado (entre 0 e 1) onde o tempo que o pino fica em nível 1 (ligado) é controlado. É usado para, por exemplo, controlar velocidade de motores, ou gerar tensões com valores controlados pelo programa. Para a comunicação PWM pode-se usar os pinos 3, 5, 6, 9, 10 e 11. A função PWM será detalhada na seção 1.4.1.7;

- **Porta Serial USART**: Pode-se usar um pino para transmitir e um pino para receber dados no formato serial assíncrono (USART). Por exemplo, é possível conectar um módulo de transmissão de dados via bluetooth e permitir a comunicação com o Arduino remotamente. São reservados para USART os pinos 0 (rx recebe dados) e 1 (tx envia dados);

[6] `http://shieldlist.org`

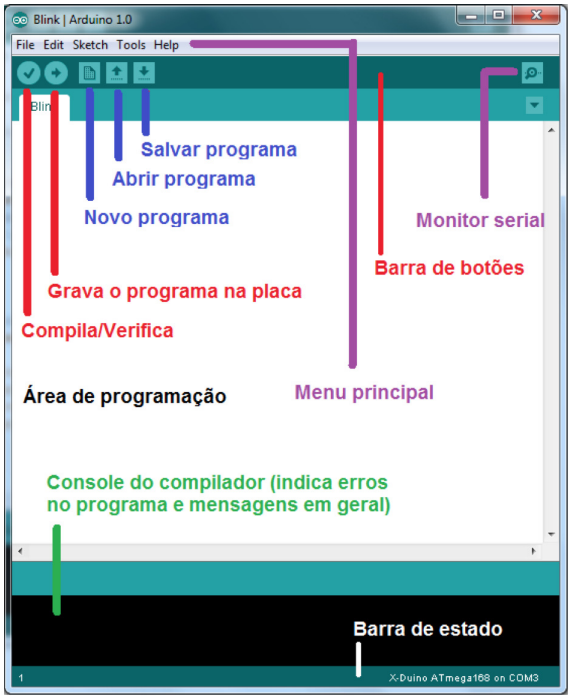

Figura 1.3. IDE de desenvolvimento de programas para o Arduino

- **Comparador Analógico**: Estes pinos podem ser usados para comparar duas tensões externas, sem precisar fazer um programa que leia essas tensões e as compare. Essa é uma forma muito rápida de comparar tensões e é feita pelo hardware sem envolver programação. São usados os pinos 6 e 7;

- **Interrupção Externa**: Os pinos 2 e 3 podem ser programados para avisar ao software em execução no Arduino sobre alguma mudança em seu estado. Por exemplo, pode-se ligar um botão a esses pinos e cada vez que alguém pressiona o botão o programa é desviado para um bloco específico que definido pelo programador.

- **Porta SPI - *Serial Peripheral Interface*** [Rooke 2001]: São pinos que permitem comunicação serial síncrona mais rápida que USART. Eles permitem por exemplo conectar cartões de memória (SD) e muitas outras coisas. São usados para este fim os pinos 10 (SS), 11 (MOSI), 12 (MISO) e 13 (SCK).

1.4.1.3. Ambiente de Desenvolvimento

O ambiente de desenvolvimento do Arduino é um compilador gcc (C e C++) que usa uma interface gráfica (Figura 1.3) construída em Java. Ele é um IDE que permite a um desenvolvedor: i) escrever o código de um programa, ii) usar bibliotecas adicionais, iii) compilar o código, iv) enviar o código compilado para o Arduino e v) monitorar (se necessário) a execução do programa a partir de uma interface serial. Tanto o IDE como um grande número de bibliotecas adicionais podem ser encontrados a página oficial do

Arduino[7]. O IDE está disponível para as plataformas Linux, Mac e Windows e é gratuito. Como é feito em Java não é necessário instalar ou configurar, basta apenas que o ambiente Java (JRE) esteja previamente instalado na plataforma.

O primeiro passo para utilizar o IDE é escolher a versão do Arduino usada para desenvolvimento na barra de menu principal em "*Tools->Board*". O ambiente oferece também uma grande quantidade de exemplos que podem ser acessados através do menu "*File->Examples*". Existem ainda a barra de botões que oferece 6 opções para: compilar/verificar um programa, gravar o programa na placa, criar um novo código fonte (novo programa), abrir um arquivo fonte, salvar as alterações realizadas e abrir a interface serial de execução de um programa. Além disso, o IDE oferece uma barra de estado, que permite verificar se um programa foi carregado corretamente no Arduino, o console de mensagens do compilador e a área reservada para programação. Todos os elementos do IDE podem ser vistos na Figura 1.3.

As funções são referências essenciais para o desenvolvimento de um projeto usando o Arduino. Essas funções já implementadas e disponíveis em bibliotecas direcionam e exemplificam as funcionalidades básicas do microcontrolador. Entre as funções e bibliotecas disponíveis, merecem destaque:

- I/O Digital: *pinMode(), digitalWrite()* e *digitalRead()*;
- I/O Analógico: *analogReference(), analogRead()* e *analogWrite()* - PWM;
- I/O Avançado: *tone(), noTone(), shiftOut()* e *pulseIn()*;
- Tempo: *millis(), micros(), delay()* e *delayMicroseconds()*;
- Matemática: min(), max(), abs(), constrain(), map() e pow();
- *Ethernet* - para se conectar a uma rede Ethernet usando o Arduino Ethernet Shield;
- *LiquidCrystal* - para controlar telas de cristal líquido (LCDs);
- Servo - para controlar servo motores;
- Constantes: HIGH, LOW, INPUT, OUTPUT entre outras.

Alguns dos itens listados acima serão usados e explicados nos exemplos deste artigo. Toda a documentação das funções e bibliotecas padrões do Arduino pode ser encontrada no site oficial[8].

1.4.1.4. Estrutura Básica de um Programa

Para explicar esta estrutura básica de um programa no Arduino, este artigo optou por usar um dos exemplos disponíveis no IDE que permite acender/apagar uma lâmpada de

[7] `http://www.arduino.cc/`

[8] `http://playground.arduino.cc/Portugues/Learning`

LED. Um programa escrito para o Arduino é tem como requisito duas funções principais denominadas *Setup* e *Loop*. Estas funções serão detalhadas a seguir.

A função *Setup* serve para inicialização da placa e do programa. Ela é executada penas uma vez quando a placa é ligada ou reiniciada. No exemplo, a função *Setup* usa o comando ***pinMode*** para informar ao Arduino que o pino 13 (onde está conectado um LED) será uma saída digital. A função *Loop* funciona como o bloco principal (função *main*) de um programa escrito em C. O código escrito nesta função é executado repetidamente até que a placa seja desligada ou reiniciada.

Uma vez que o programa esteja pronto, um desenvolvedor deve inicialmente acionar o botão compilar para verificar se não existe nenhum erro. Depois disso, caso o programa não contenha erro, deve-se enviá-lo para placa através do botão de gravar. Após o envio, os Leds RX e TX presentes no Arduino deverão piscar, informando que o código está sendo carregado. Logo após, ele começará a executar o programa que lhe foi enviado.

No código do programa visto a seguir, o comando ***digitalWrite*** escreve na saída do pino 13 o nível de tensão HIGH (5v), acendendo o LED. O comando ***delay*** serve apenas para fazer o programa pausar a sua execução por 1000 milésimos de segundo. Em seguida, o nível de tensão é alterado para LOW (0v) e o Led apaga. Esta dinâmica de acender e apagar o LED é repetida indefinidamente.

```
int led = 13; // Definição do pino usado.
void setup() {
    Serial.begin(9600);    // Inicia o monitor serial
    pinMode(led, OUTPUT); // Configura o pino como Saída
    Serial.println("Programa␣Iniciado"); // Escreve no MS
}
void loop() {
    digitalWrite(led, HIGH); // Liga o LED
    delay(1000);             // Pausa 1 segundo
    digitalWrite(led, LOW);  // Desliga o LED
    delay(1000);             // Pausa 1 segundo
}
```

O IDE oferece ainda uma interface denominada Monitor Serial que permite criar um canal de comunicação com o Arduino e também realizar a depuração de um programa em execução. Para isso, basta conectar a placa no computador e clicar no botão "Serial Monitor". Por exemplo, no mesmo código visto acima, o comando ***Serial.begin(9600)*** inicializa a comunicação com uma taxa de 9600 baunds (taxa de bits). O comando ***Serial.println ("Programa Iniciado")*** envia a mensagem para o monitor serial. A mensagem aparecerá apenas um vez no monitor pois foi enviada dentro da função *Setup*.

1.4.1.5. Sintaxe e Principais Funções

Conforme visto anteriormente, as linguagens base para a programar um Arduino são C/C++. Logo, as estruturas de controle, seus elementos de sintaxe, operadores aritméticos, operadores de comparação, entre outras podem ser utilizados no IDE. Além dos

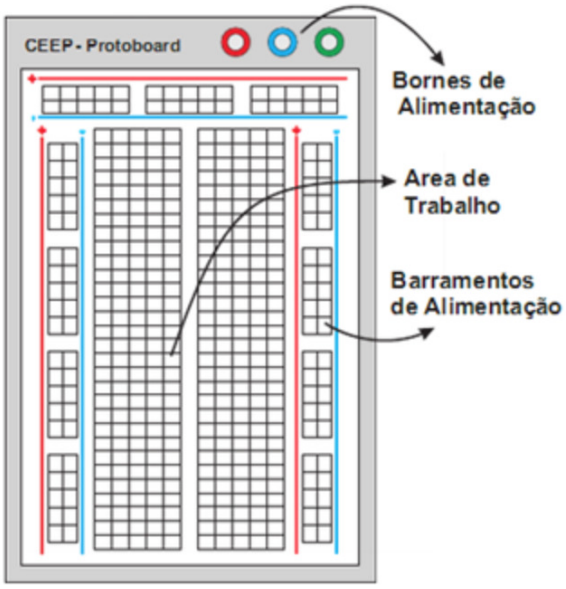

Figura 1.4. Exemplo de *Protoboard*

recursos das linguagens C/C++, o Arduino oferece uma vasta biblioteca[9] com funções que facilitam o desenvolvimento de aplicações. Dentre estas funções, pode-se destacar:

- ***pinMode (pin, mode)***: configura o pino especificado para que se comporte como entrada ou saída, sendo **pin** = número do pino representado por uma variável do tipo inteiro e **mode** = INPUT ou OUTPUT constantes inteiras.

- ***digitalWrite (pin,value)***: escreve um valor HIGH ou LOW (constantes inteiras) em um pino digital. Se o pino foi configurado como saída sua voltagem correspondente será: 5V para HIGH e 0V para LOW. Se o pino estiver configurado como entrada escrever um HIGH levantará o resistor interno de 20kΩ. Escrever um LOW rebaixará o resistor.

- **int *digitalRead (pin)***: lê e retorna o valor de um pino digital especificado, HIGH ou LOW.

- **int *analogRead (pin)***: lê o valor de um pino analógico especificado. Pode mapear voltagens que variam entre 0 a 5v.

- ***analogWrite (pin, value)***: escreve um valor analógico (onda PWM, ver seção 1.4.1.7). Pode ser utilizada para acender um LED variando o brilho ou girar um motor a velocidade variável. Após realizar essa função o pino vai gerar uma onda quadrada estável com ciclo de rendimento especificado até que o próximo analogWrite() seja realizado (ou que seja realizado um ***digitalRead()*** ou ***digitalWrite()*** no mesmo pino).

[9] `http://arduino.cc/en/Reference/HomePage`

1.4.1.6. Uso do *Protoboard*

A *protoboard* é um dos equipamentos mais úteis no uso do Arduino para o desenvolvimento de aplicações ubíquas. Ela é uma placa composta de uma matriz de contatos que permite a construção de circuitos experimentais sem a necessidade de efetuar a soldagem dos componentes. Isso permite que seja efetuada uma série de experimentos com os mesmos componentes inserindo-os ou removendo-os com rapidez e segurança. Esta seção detalha as possibilidades que a *protoboard* oferece.

A Figura 1.4 exibe uma *protoboard*. Neste exemplo, os bornes[10] de alimentação devem ser conectados a fonte que será utilizada no experimento. Para energizar os barramentos de alimentação é necessário utilizar fios e fazer a conexão elétrica entre os bornes e a fonte. Os barramentos de alimentação podem ser duas linhas ou colunas indicadas pelas cores azul (-) e vermelha (+) de pontos de inserção, cada uma dessas linhas tem seus pontos conectados eletricamente através de uma sustentação metálica condutora que fica abaixo da capa plástica da *protoboard*. Esses barramentos irão fornecer o acesso a energia necessária ao um experimento que será montado na área de trabalho. A área de trabalho consiste em dois blocos distintos (colunas) cada um com diversas linhas de pontos de inserção. Os pontos de uma mesma linha estão conectados eletricamente um ao outro, porém as linhas estão isoladas umas das outras.

A Figura 1.5 apresenta o esquema do exemplo usado para ligar/desligar um LED. No exemplo, é necessário conectar ao Arduino, um LED e um resistor (para evitar que a lâmpada queime). Cada um dos polos do LED é conectado a uma linha da *protoboard*. A linha referente ao polo positivo é conectada através de um fio ao pino 13 do Arduino. Da mesma forma, o polo negativo do LED é conectado ao pino GND (aterramento) do Arduino. Entre o polo negativo e o GND é posicionado o resistor. Depois disso, basta usar o IDE para carregar o código compilado do programa.

1.4.1.7. *Pulse Width Modulation*

A modulação por largura de pulso de um sinal ou em fontes de alimentação (mais conhecida pela sigla em inglês PWM (*Pulse-Width Modulation*)) envolve a modulação de sua razão cíclica (*duty cycle*) para transportar qualquer informação sobre um canal de comunicação ou controlar o valor da alimentação entregue à carga [Barr 2001]. Esta técnica permite obter resultados analógicos a partir de pinos digitais.

A técnica consiste em manipular o *duty cycle* afim de transportar informação ou controlar a potência de algum outro circuito. Com isso, pode-se fazer com que um sinal digital oscile entre 0v e 5v em uma determinada frequência. O *duty cycle* é a razão do tempo em que o sinal permanece em 5v sobre o tempo total de oscilação, como está ilustrado na Figura 1.6. O *duty cycle* permite controlar o percentual de tempo em que o sinal fica em 5v. Dessa forma, pode-se utilizar essa técnica para limitar a potência de algum circuito como, por exemplo, aumentar ou diminuir a intensidade de um LED, controlar um servo motor, etc.

[10] São terminais de conexão que unem fios ou cabos por meio de parafusos

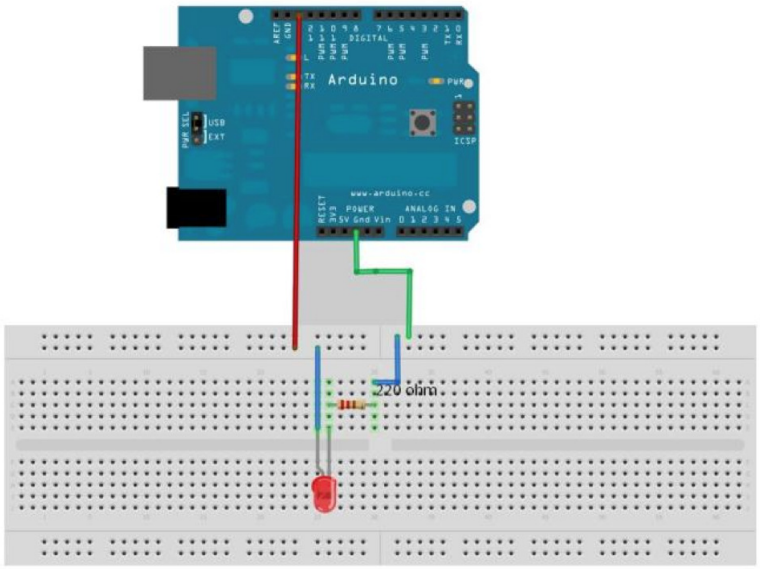

Figura 1.5. Esquema de ligações de uma *Protoboard*:exemplo liga LED

Para facilitar o entendimento da técnica, o código a seguir usa PWM para controlar a intensidade de uma lâmpada de LED. Suponha que as conexões elétricas configuradas são semelhantes aquelas exibidas na Figura 1.5 e ainda que, conforme dito na seção 1.4.1.2, os pinos para PWM (Arduino UNO) são 2, 5, 6, 9, 10 e 11.

```
int led = 11;
void setup()  {
  pinMode(led, OUTPUT); //Configura pino 11 como saída
}

void loop()  {
 int i;
 for(i=0;i<255;i++){
  analogWrite(led, i); //escreve o valor de i no pino 11
  delay(30); // pausa o programa por 30 milésimos
 }
}
```

A função ***analogWrite()***, apesar de usada sobre uma entrada digital (pino 11), é a responsável pelo PWM. Ela recebe como parâmetro o pino e um valor entre 0 e 255, onde 0 corresponde a 0% e 255 corresponde a 100% do *duty cycle*. Ao executar este exemplo, pode-se perceber que o LED inicia desligado e acende de maneira suave até chegar ao máximo de luminosidade. Cada etapa de luminosidade diferente corresponde a uma iteração do "for". Um vídeo com este exemplo em execução pode ser visto no YouTube[11].

[11] https://www.youtube.com/watch?v=8ocVoEDD5mw

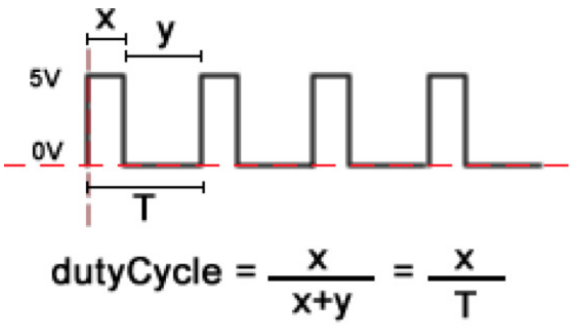

Figura 1.6. *Duty Cycle* do sinal PWM

1.4.1.8. Uso de Sensores e Atuadores

Sensores são dispositivos que permitem capturar informações como temperatura, pressão, presença, umidade, intensidade luminosa, entre outras do ambiente no qual eles estão inseridos. Em geral os sensores atuam transformando partes de uma grandeza física em um sinal elétrico, que por sua vez pode ser interpretado por equipamentos eletrônicos [Borges and de Carvalho Dores 2010]. Em outras palavras, sensores são componentes que permitem que um equipamento eletrônico possa interagir com o mundo real. Segundo [Borges and de Carvalho Dores 2010], os sensores quando operam de forma direta, transformando uma forma de energia em outra são chamados de transdutores. Os sensores onde as operações ocorrem de forma indireta alteram suas propriedades, como a resistência, capacitância ou indutância, sob a ação da grandeza de forma que essa alteração ocorre mais ou menos proporcional. Por exemplo, os sensores de luz LDR (*Light-dependent resistors*) variam sua resistência inversamente a quantidade de luz que incide sobre o mesmo. Dessa forma, quando há uma grade quantidade de luz incidindo sobre o sensor, eles têm uma resistência muito baixa e isso permite que o fluxo da corrente elétrica aumente, enquanto que quando há pouca luz, eles apresentam uma resistência elevada e evitam a passagem corrente.

Um atuador, assim como um sensor, é um transdutor que transforma uma forma de energia em outra, e ainda pode fazer o caminho inverso [Borges and de Carvalho Dores 2010]. Em outras palavras, ao invés de apenas transformar partes de uma grandeza física em um sinal elétrico, ele pode transformar um sinal elétrico em uma grandeza física como: movimento, magnetismo, calor entre outras. Por exemplo, os relés são dispositivos eletromecânicos que funcionam com pequenas correntes, mas são capazes de controlar circuitos externos que envolvem correntes elevadas, e são formados basicamente por uma bobina e um conjunto de contatos. Quando uma corrente circula pela bobina cria um campo magnético que atrai, fechando os contatos, permanecendo assim enquanto houver alimentação de energia na bobina e, como consequência, permitindo a passagem de energia através do relé.

Diversos sensores podem ser acoplados e usados no Arduido[12]. O uso de sensores no Arduino será detalahdo a partir de um exemplo de uso. Para o experimento escolheu-se

[12] `http://store.arduino.cc/category/19`

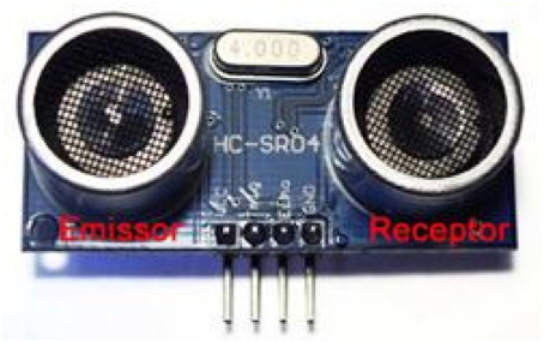

Figura 1.7. Exemplo de sensor de Ultrassom

o sensor de presença chamado de Ultrassom. Ele emite um sinal na faixa de frequência do ultrassom (por volta de 30kHz) que se propaga pelo ar até encontrar um obstáculo. Ao colidir com o obstáculo uma parte do sinal é refletida e captada pelo sensor. Um único sensor de ultrassom, possui um módulo receptor e outro emissor conforme pode ser visto na Figura 1.7. Dessa forma, são necessários dois pinos para conectar cada um dos módulos: um como saída (que emite o sinal) e outro como entrada (que recebe o sinal). O pino que envia o pulso é chamado de *trigger* e o que recebe *echo*. O código a seguir representa um programa que permite capturar e imprimir no monitor serial a distância entre um obstáculo e o sensor.

```
#define echoPin 13
#define trigPin 12

void setup(){
   Serial.begin(9600);
   pinMode(echoPin, INPUT); // echoPin como entrada
   pinMode(trigPin, OUTPUT); //trigPin como saída
}

void loop(){
  //seta o trigPin com um pulso baixo LOW
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
  //seta o trigPin com pulso alto HIGH
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
  //seta o trigPin com pulso baixo novamente
    digitalWrite(trigPin, LOW);
    int long duracao=pulseIn(echoPin,HIGH);
    //usa o echoPin para calcular duração do pulso.
    int distancia = duracao/29/2;
    //v=s/t onde s=distância, v= velocidade do som,
    //t=tempo dobrado(ida e volta do sinal)
    Serial.print("Distancia em CM: ");
    Serial.println(distancia);
    delay(1000);
```

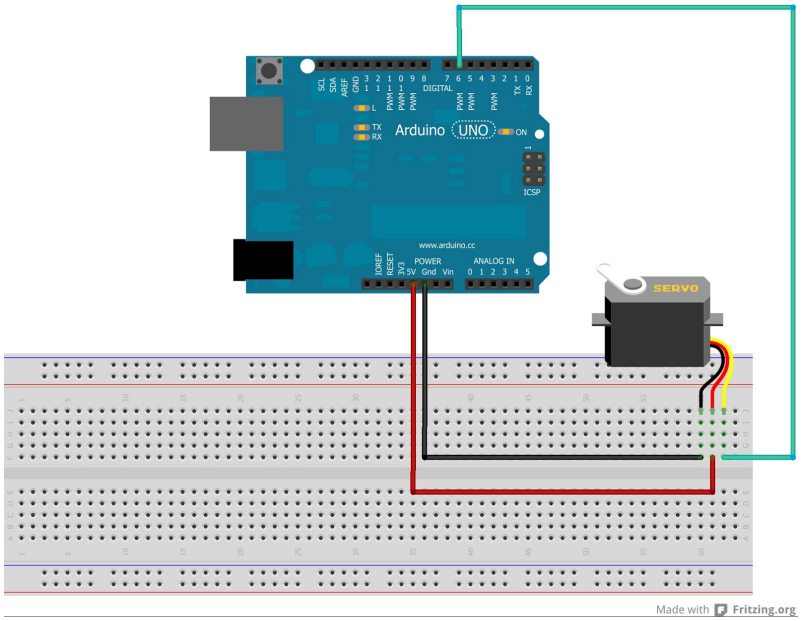

Figura 1.8. Controle de Servo Motor

}

No código anterior, pode-se perceber que o módulo *trigger* começa desligado, e logo após 2 microssegundos ele é ligado e emite um sinal. Após 10 microssegundos o *trigger* volta a ser desligado. Na variável *duration* fica armazenando o tempo entre a emissão do sinal até o módulo *echo* ser ligado. Assim, a distância é calculada utilizando a formula de velocidade V = S/t. Como o tempo é de ida e volta, deve-se dividi-lo por 2. A velocidade do som no ar é de aproximadamente 340 m/s. Para que o resultado saia em centímetros, é necessário ainda dividir o tempo (*duration*) por 29. Por fim o resultado é impresso no serial monitor.

1.4.1.9. Controle de Servo Motor

O servo motor é um dispositivo eletromecânico que, a partir de um sinal elétrico em sua entrada, pode ter seu eixo posicionado em uma determinada posição angular. Servo motores não foram feitos para girar livremente, e sim para ir para uma posição escolhida de acordo com um limite, que é geralmente é de 180 graus. Por serem pequenos, compactos, além de permitir um posicionamento preciso de seu eixo, eles são comumente utilizados em robótica e modelismo. O IDE do Arduino já possui uma biblioteca que permite controlar um servo motor denominada é a “Servo.h”.

Como pode-se observar na Figura 1.8, o servo motor possui 3 fios: 2 para alimentação (vcc e gnd) e um para controlar sua posição. Este controle da posição é feito através do PWM. A conexão do servo no Arduino é simples. O fio escuro (geralmente preto ou marrom) é o gnd e deve ser ligado no pino gnd. O vermelho é o vcc e deve ser ligado no pino de 5v. E por fim o fio amarelo ou laranja é o fio de controle e deve ser ligado em algum pino que possua PWM.

Uma vez que as ligações tenham sido devidamente realizadas pode-se controlar um servo motor a partir da biblioteca Servo.h. O código a seguir apresenta um exemplo de uso de servo motor. O programa faz um servo girar de 0 a 180 graus e depois fazer o procedimento inverso.

```
#include <Servo.h>

Servo motor;
int pos=0;

void setup(){
 motor.attach(8);
}

void loop() {
 for (pos=0;pos<180;pos++){
  motor.write(pos);
  delay(15);
 }

for (pos=180;pos>=1;pos--){
  motor.write(pos);
  delay(15);
 }
 delay(1000);
}
```

No programa descrito acima, a variável "motor" do tipo *Servo* encapsula toda a complexidade do controle de um servo motor. Ela oferece como interface diversas funções de controle. Entre elas pode-se destacar: i) *void attach(int)* que permite associar um motor a um pino, ii) *void write(int)* que permite escrever o ângulo para o qual o servo deverá girar (entre 0 e 180), e iii) *int read()* que permite saber o ângulo que o servo se encontra.

1.4.2. Raspberry Pi

O Raspberry Pi é um computador do tamanho de um cartão de crédito com todo o hardware integrado em uma única placa que foi desenvolvido pela Fundação Raspberry Pi. O objetivo principal do Raspberry é estimular o ensino de ciência da computação básica em escolas. Existem diversas possibilidades de uso deste computador como, por exemplo: acoplar sensores para ajudar os proprietários a monitorarem o seu consumo de energia, criar sistemas para pessoas com deficiência que permitem que eles possam desbloquear remotamente sua porta de entrada para permitir que os visitantes entrem, acoplar uma TV e adicionar as funcionalidades de um mídia *center*, e entre outras.

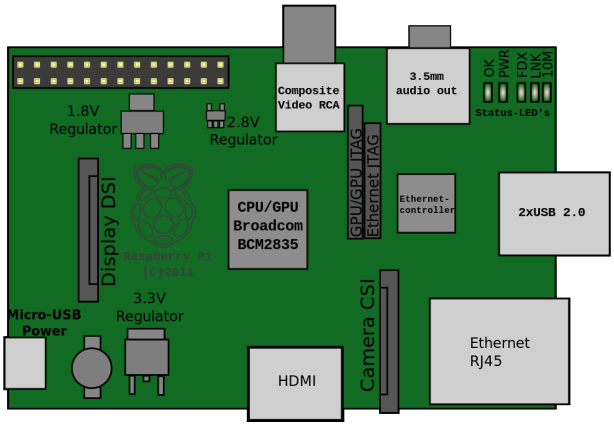

Figura 1.9. Arquitetura de Hardware do RapsBerry Pi

1.4.2.1. Hardware

Dentre modelos de Raspberry Pi disponíveis atualmente[13], os Modelos A e B são as principais referências. A grande diferença entre os dois modelos é que o Modelo B possui um controlador *Ethernet* e duas portas USB, enquanto que o Modelo A possui apenas uma porta USB e nenhuma porta de *Ethernet*. O computador é baseado em um SoC (*system on a chip*) *Broadcom BCM2835*, que inclui um processador *ARM1176JZF-S* de 700 MHz, GPU *VideoCore IV*, e 512 MB de memória RAM em sua última revisão. O projeto não inclui uma memória não-volátil (como um disco rígido) mas possui uma entrada de cartão SD para armazenamento de dados. A Figura 1.9 exibe a arquitetura do Modelo B do Raspberry Pi.

Apesar de não possuir a porta *Ethernet*, o Modelo A pode ser conectado a internet através de um adaptador USB de *Ethernet* ou Wi-Fi. O Raspberry Pi não possui um relógio de tempo real (RTC), criando a necessidade do sistema operacional usar um *Network Time Protocol* (NTP) ou do usuário fornecer a hora ao sistema.

O Raspberry Pi é compatível com sistemas operacionais baseados em Linux. O *Raspbian* é a distribuição Linux oficial do Raspberry Pi. As distribuições *Arch Linux* e *Debian* são também oficialmente suportadas e disponíveis para download. O sistema operacional é normalmente armazenado no cartão SD. Além disso, qualquer linguagem que possa ser compilada na arquitetura ARMv6 pode ser usada para o desenvolvimento de software no Raspberry Pi.

[13] `http://www.raspberrypi-spy.co.uk/2012/09/raspberry-pi-board-revisions/`

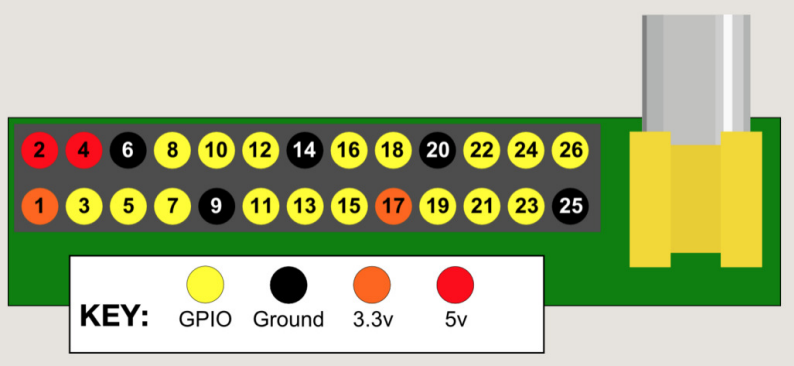

Figura 1.10. Interface de *General Purpose Input/Output* (GPIO)

1.4.2.2. *General Purpose Input/Output*

Outro importante aspecto de hardware do Raspberry é o conjunto de pinos de GPIO (*General Purpose Input/Output*). Estes pinos são portas programáveis de entrada e saída de dados usadas para prover uma interface entre a placa e periféricos, microcontroladores/microprocessadores, sensores, atuadores, etc. A interface de GPIO é um ponto fundamental para construção de ambientes interativos inteligentes. Ela é a interface entre o Raspberry e o mundo real. De forma simplificada, é possível considerar os pinos de GPIO como interruptores que podem ser ligados/desligados.

Da mesma forma que o Arduino, além do GPIO o Raspberry também suporta PWM, UART e SPI. Dos 26 pinos disponíveis, 17 são reservados para GPIO e 8 são usados para as alimentação e aterramento. A Figura 1.10 exibe a interface de GPIO. Os pinos podem ser utilizados via o código escrito em uma linguagem de programação compatível como Python, Scratch, Java, entre outras [Brock et al. 2013]. Este texto foca no uso da linguagem Java e do arcabouço Pi4J[14].

1.4.2.3. Pi4J

O projeto Pi4J tem como objetivo fornecer uma API orientada a objetos escrita em Java para manipulação da interface de GPIO da plataforma de Raspberry Pi. Ele abstrai a integração nativa de baixo nível e o monitoramento de interrupções para permitir que programadores Java mantenham o foco em implementar a lógica de negócios de uma aplicação.

A API suporta todos os modelos de hardware disponíveis do Raspberry Pi além de oferecer funções que permitem: i) configurar a direção de uso de um pino (leitura/escrita), ii) controlar o estado de um pino, iii) ler o estado de um pino, iv) acessar informações de sistema e de rede oriundos do Raspberry Pi, entre outras.

O Pi4J tem apenas dois pré-requisitos que são o *Java Runtime Environment* e a biblioteca nativa WiringPi[15]. Eles já estão presentes no *Raspian* mas também podem ser

[14] `http://pi4j.com/index.html`
[15] `http://wiringpi.com/`

instalados em qualquer um dos sistemas operacionais disponíveis. A instalação do Pi4J pode ser feita a partir de linhas de comando como, por exemplo, "curl -s get.pi4j.com | sudo bash". Este método vai realizar o *download* e a execução do *scripts* de instalação que:

- Adiciona o repositório APT (*Advanced Packaging Tool*) na lista de repositórios APT local;
- Baixa e instala a chave pública Pi4J GPG para validação de assinatura;
- Atualiza o repositório APT do Pi4J;
- Baixa e instala o a API Pi4J.

Existem outras formas de instalação e configuração do arcabouço Pi4J. Estas e outras informações podem ser encontradas no site da API[16].

1.4.2.4. Principais Objetos e Métodos do Pi4J

A implementação de um programa que use elementos do PiJ4 não é diferente do uso de outras APIs da linguagem Java. Para isso, o primeiro passo é incluir os pacotes necessários a um programa. Todos os pacotes do arcabouço estão agrupados em um mesmo diretório e podem ser acessados em *com.pi4j.io.gpio*.

O principal elemento do *framework* é o *GpioController*. Ele representa uma interface que descreve todas as operações disponíveis para os pinos de GPIO. Em um projeto, apenas uma instância de *GpioController* precisa ser criada. A criação de uma instância de um *GpioController* é feita a partir de uma fábrica de controladores denominada *GpioFactory*. Este elemento permite a criação de instancias de controladores GPIO a partir do método *createInstance*.

Cada modelo de Raspberry Pi segue um padrão de numeração de pinos. Para evitar que possíveis mudanças de numeração em novas versões dos computadores possam provocar problemas no arcabouço, o Pi4J usa um esquema de numeração que ajuda a isolar o software do hardware. Nos modelos A e B o Pi4J implementa o mesmo esquema de numeração usado no projeto *Wiring Pi* e oferece uma enumeração Java denominada *RaspiPin* para gerenciar os pinos de GPIO acessíveis. Para o modelo "Compute" o Pi4J usa uma outra enumeração denominada *RCMPin* que é baseada no esquema de numeração de pinos GPIO *Broadcom*.

Para usar um pino GPIO é necessário que ele seja provisionado. Isto significa que ele deve ser configurado de acordo com a necessidade de uso (I/O, estado inicial, nome amigável, etc.). O trecho de código a seguir apresenta um exemplo que provisona o pino 4 do Rapspberry em modo de saída (output) e com estado inicial desligado.

```
GpioPinDigitalOutput myLed;
myLed= gpio.provisionDigitalOutputPin(RaspiPin.GPIO_04,
```

[16] http://pi4j.com/install.html

```
                                        "LedNomeAmigavel",
                                        PinState.LOW);
```

Outras operações importantes na manipulação dos pinos de GPIO são aquelas relacionadas ao controle do estado destes pinos. O Pi4J oferece uma série de métodos que permitem obter o estado atual de um pino como, por exemplo, *myLed.getState()* ou ainda verificar se ele está ligado como em *myLed.isHigh()*. Há ainda uma série de métodos que permitem mudar o estado de um pino, como pode ser visto no código a seguir:

```
//configura explicitamente o estado de um pino provisionado
myLed.setState(PinState.HIGH);
//inverte o estado de um pino
myLed.toggle();
/*mantém o estado de um pino ligado por um período definido
(neste caso 1000 milissegundos)*/
myLed.pulse(1000);
```

O Pi4J permite também o monitoramento de interrupções ocorridas em pinos de GPIO. Esta funcionalidade permite realizar o monitoramento ativo de interrupções que é uma opção mais eficiente do que percorrer a lista de pinos e interrogar o estado de cada um deles. Para isso, é necessário criar uma classe que implemente uma das interfaces *GpioListeners*. Estas interfaces contêm apenas um método chamado de *pinStateChanged(GpioPinStateChangeEvent event)*. Este método funciona como um observador que informa quando uma interrupção ocorre. Para usar um observador de eventos, é preciso registra-lo no pino instanciado. Isso deve ser feito a partir do método *myLed.addListener(new ExampleListener())*. O trecho de código a seguir demonstra um exemplo simples da implementação desta interface:

```
class ExampleListener implements GpioPinListenerDigital {

 void handleGpioPinDigitalStateChangeEvent
              (GpioPinDigitalStateChangeEvent event) {
 // Exibe o estado  de um pino no console
 System.out.println("Mudou␣Estado:␣"
                  + event.getPin()
                  + "␣=␣"
                  + event.getState());
    }
}
```

O arcabouço oferece ainda a possibilidade de associar gatilhos a alguns pinos. Estes gatilhos podem ser acionados a partir de uma mudança de estado em outro pino. Esta funcionalidade é implementada a partir da série de interfaces *GpioTriggers*. Para usá-la, basta adicionar uma instâncioa de um gatilho no pino a ser monitorado. Por exemplo *myLedOutro.addTrigger(new GpioSyncStateTrigger(myLed))*. Neste caso, quando o pino *myLed* mudar o seu estado, o pino *myLedOutro* será notificado.

Outra funcionalidade muito útil do Pi4J é a que permite configura um pino para ser

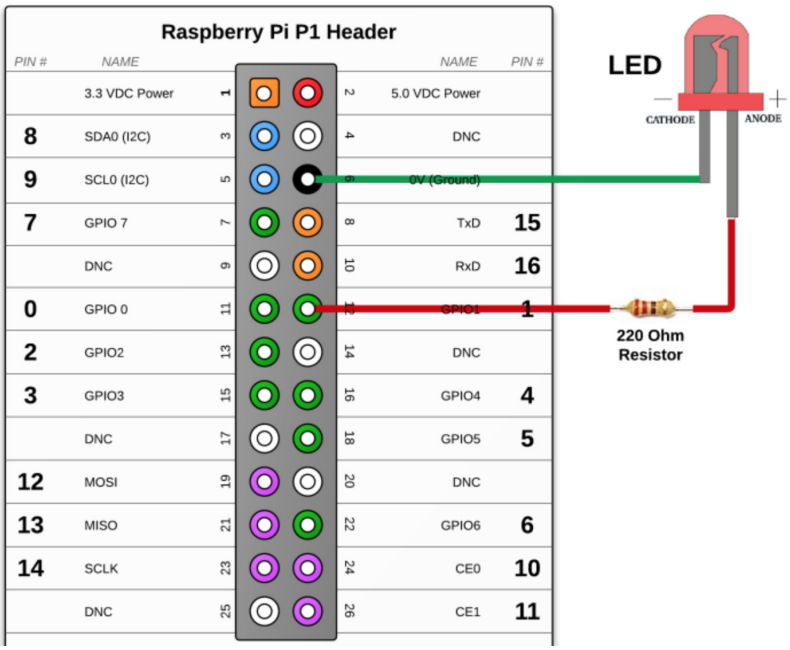

Figura 1.11. Esquema de conexões do exemplo Blink LED no Pi4J

desligado automaticamente quando um programa é encerrado. Isto garante que os pinos não ficarão eletricamente ativos (e possivelmente executando alguma operação) depois que um programa for encerrado. O método *setShutdownOptions* permite configurar o comportamento adequado do desligamento de uma instância de um pino.

Além dos métodos e objetos apresentados nesta seção, existem diversos outros exemplos de uso da API Pi4J. Estes exemplos podem ser encontrados no site do projeto[17]. O site também contém a documentação da API e outros materiais complementares.

1.4.2.5. Exemplos de Uso do Pi4J

O primeiro exemplo de uso do Pi4J exibido neste texto consiste em reescrever a aplicação apresentada na seção 1.4.1.4 que permite acender e apagar um LED. Este exemplo tem como objetivo demonstrar a simplicidade do controle dos pinos de GPIO no Raspberry Pi a partir do Pi4J. O esquema de conexão entre o LED, o resistor e os pinos de GPIO pode ser visto na Figura 1.11. O código desse exemplo de uso é exibido a seguir:

```
import com.pi4j.io.gpio.GpioController;
import com.pi4j.io.gpio.GpioFactory;
import com.pi4j.io.gpio.GpioPinDigitalOutput;
import com.pi4j.io.gpio.PinState;
import com.pi4j.io.gpio.RaspiPin;

public class ControlGpioExample {
  public static void main(String[] args)
                      throws InterruptedException {
```

[17] http://pi4j.com/usage.html

```
        // cria o gpio controller
        final GpioController gpio = GpioFactory.getInstance();
        // provisiona pino #01 como output
        final GpioPinDigitalOutput pin;
        pin = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_01,
                                             "MyLED",
                                             PinState.HIGH);
        Thread.sleep(5000);
        // desliga o pino
        pin.low();
        Thread.sleep(5000);
        //Inverte o estado do pino
        pin.toggle();
        Thread.sleep(5000);
        //Inverte o estado do pino
        pin.toggle();
        Thread.sleep(5000);
        // Mantém o pino ligado 1 segundo
        pin.pulse(1000);
        // Desliga todos os pinos
        gpio.shutdown();
    }
}
```

Para compilar e executar o exemplo pode-se usar as linhas de comando: "**javac -classpath .:classes:/opt/pi4j/lib/'*' -d . ControlGpioExample.java**" e "**sudo java -classpath .:classes:/opt/pi4j/lib/'*' ControlGpioExample**" respectivamente. O programa em execução vai fazer o LED acender e apagar (com um intervalo de 5 segundos) duas vezes. Depois disso o LED pisca (um segundo) e o programa termina.

O próximo exemplo de uso apresentado aqui tem como objetivo demonstrar o uso de sensores que podem ser acoplados ao Raspberry Pi. Para este experimento foi escolhido o mesmo sensor de distância utilizado na seção 1.4.1.8. Este sensor permite descobrir a distância entre ele um objeto do mundo real. Os pinos *trigger* e *echo* do sensor foram conectados respectivamente aos pino 0 e 3 do Raspberrt Pi. A lógica do cálculo é quase a mesma usada anteriormente. A única diferença é o resultado final que aqui é dado em milímetros. O código deste exemplo de uso é exibido a seguir. Para compilar e executar o exemplo pode-se usar as linhas de comando: "**javac -classpath .:classes:/opt/pi4j/lib/'*' -d . RangeFinder.java**" e "**sudo java -classpath .:classes:/opt/pi4j/lib/'*' RangeFinder**" respectivamente.

```
import com.pi4j.io.gpio.GpioPinDigitalInput;
import com.pi4j.io.gpio.GpioPinDigitalOutput;
import com.pi4j.io.gpio.PinState;
import com.pi4j.io.gpio.event.GpioPinDigitalStateChangeEvent;
import com.pi4j.io.gpio.event.GpioPinListenerDigital;

public class RangeFinder {
 public static void main(String args[]){
```

```
  double result = 0;
  GpioPinDigitalOutput firepulse;
  GpioPinDigitalInput result_pin;
   try {
    firepulse= gpio.provisionDigitalOutputPin(RaspiPin.GPIO_00,
                          "Range Finder Trigger",PinState.LOW);

    result_pin = gpio.provisionDigitalInputPin(RaspiPin.GPIO_03,
             "Range Pulse Result", PinPullResistance.PULL_DOWN);
    // Dispara o sinal
    firepulse.high();
    Thread.sleep(20);
   } catch (InterruptedException e) {
      e.printStackTrace();
  }
  firepulse.low();
  // Aguarda o retorno do sinal
  double startTime = System.currentTimeMillis();
  double stopTime = 0;
  do {
       stopTime = System.currentTimeMillis();
      if ((System.currentTimeMillis() - startTime) >= 40) {
      break;//time out
     }
  } while (result_pin.getState() != PinState.HIGH);

  // calcula a distância .

  if ((stopTime - startTime) <= 38) {
   result = (stopTime - startTime) * 165.7;
  } else {
   System.out.println("Timed out");
   result = -1;
  }
  System.out.println(result + " mm é a distância");
 }
}
```

1.5. Conclusão

Este artigo apresentou os conceitos básicos necessários para aqueles que buscam desenvolver aplicações ubíquas. O texto apresentado é especialmente útil para aqueles têm interesse em construir ambientes inteligentes composto por um conjunto de objetos interativos. O artigo lista as principais tecnologias da área (tanto de hardware quanto de software) e apresenta exemplos práticos e didáticos que facilitam a sua compreensão.

Os sistemas ubíquos modificam a forma de como os usuários interagem com os computadores, pois seus serviços devem estar em toda parte, auxiliando os usuários em diversas atividades do dia-a-dia. Esse auxílio deve acontecer de forma transparente, no

qual nenhuma ou pouca atenção e entrada de dados do usuário são necessárias.

A criação de sistemas ubíquos envolve uma combinação de disciplinas, elementos e pessoas diferentes. O uso de plataformas como Arduino e Raspberry Pi é ponto fundamental para ajudar a diminuir o esforço demandado na construção destes sistemas. Eles permitem que os requisitos essenciais para computação ubíqua como, captura de informação do ambiente, comunicação sem fio, interfaces não convencionais, entre outros possam ser cumpridos com um nível de complexidade semelhante ao de construir um software.

Apesar de serem considerados as plataformas mais populares e potentes para muitos projetos interessantes, tanto o Arduino quanto Raspberry não são as únicas iniciativas nesta área. Existem outros projetos que se apresentam como concorrentes diretos do Raspberry e também do Arduino. Entre eles é possível destacar *BeagleBoard* e o Intel Galileo.

O *BeagleBoard*[18] é um computador de placa única de baixo consumo elétrico, classificada como hardware livre. Sua primeira versão foi lançada em 28 de julho de 2008 pela parceria entre a *Texas Instruments* e a *Digi-Key*. Seu processador é o Cortex-A8 da Arquitetura ARM. Esse computador se destaca pela portabilidade e por consumir apenas 2 watts.

A Intel Galileo[19] representa a primeira de uma família de placas de desenvolvimento certificadas para Arduino baseadas na arquitetura Intel e projetadas especialmente para fabricantes, estudantes educadores e entusiastas da eletrônica. A placa de desenvolvimento Intel Galileo é projetada para que seu hardware, software e pinos sejam compatíveis com uma grande variedade de blindagens Arduino, além de permitir que os usuários incorporem *firmware* do Linux nas suas programações de projetos Arduino. O hardware é baseado no Intel Quark SoC X1000 - um sistema da classe de processadores Intel Pentium de 32 bits em um chip (SoC) - o autêntico processador Intel. Os recursos nativos de E/S da placa Intel Galileo oferecem um amplo suporte a periféricos de hardware e software para aqueles que buscam um único controlador de placa de baixo custo.

Referências

Alisi, T. M., Del Bimbo, A., and Valli, A. (2005). Natural interfaces to enhance visitors' experiences. *MultiMedia, IEEE*, 12(3):80–85.

Armstrong, J. A., Ferguson, P. A., Gaillard, M. K., Greenwood, M., Jaskolski, S. V., Jones, A. K., Langford, G. M., Lubchenco, J., Menger, E. L., Miller Jr, J. A., et al. (2001). National science foundation.

Banzi, M. (2008). *Getting Started with Arduino*. Make Books - Imprint of: O'Reilly Media, Sebastopol, CA, ill edition.

Barr, M. (2001). Pulse width modulation. *Embedded Systems Programming*, 14(10):103–104.

Barton, J., Barton, J. J., and Kindberg, T. (2001). The cooltown user experience.

[18] `http://beagleboard.org`

[19] `http://arduino.cc/en/ArduinoCertified/IntelGalileo`

Borges, L. P. and de Carvalho Dores, R. (2010). Automação predial sem fio utilizando bacnet/zigbee com foco em economia de energia. *Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT. TG*, (06).

Brock, J. D., Bruce, R. F., and Cameron, M. E. (2013). Changing the world with a raspberry pi. *J. Comput. Sci. Coll.*, 29(2):151–153.

Brunvand, E. (2013). Lights! speed! action!: Fundamentals of physical computing for programmers. In *ACM SIGGRAPH 2013 Courses*, SIGGRAPH '13, pages 13:1–13:108, New York, NY, USA. ACM.

Burrell, J., Gay, G. K., Kubo, K., and Farina, N. (2002). Context-aware computing: A test case. In *UbiComp 2002: Ubiquitous Computing*, pages 1–15. Springer.

Carvalho, A. d. L., Ponce de Leon, F. d., et al. (2006). Grandes desafios da pesquisa em computação no brasil–2006–2016. *São Paulo: Sociedade Brasileira de Computação*.

Chan, M., Hariton, C., Ringeard, P., and Campo, E. (1995). Smart house automation system for the elderly and the disabled. In *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*, volume 2, pages 1586–1589. IEEE.

Crowe, C., Gulick, D. E., and Lawell, T. G. (1990). Enhanced universal asynchronous receiver-transmitter. US Patent 4,949,333.

de Araujo, R. B. (2003). Computação ubíqua: Princípios, tecnologias e desafios. In *XXI Simpósio Brasileiro de Redes de Computadores*, volume 8, pages 11–13.

Dey, A. K., Abowd, G. D., and Salber, D. (2000). A context-based infrastructure for smart environments. In *Managing Interactions in Smart Environments*, pages 114–128. Springer.

Essa, I. A. (2000). Ubiquitous sensing for smart and aware environments. *Personal Communications, IEEE*, 7(5):47–49.

Fleck, M. e. (2002). Rememberer: A tool for capturing museum visits. In *UbiComp 2002: Ubiquitous Computing*, pages 48–55. Springer.

Henrique B. Filho, O. (2012). Componentes eletronicos e unidades de medida, conceitos basicos.

Hunt, A. K. and Schalk, T. B. (1996). Simultaneous voice recognition and verification to allow access to telephone network services. US Patent 5,499,288.

Kaufmann, B. and Buechley, L. (2010). Amarino: A toolkit for the rapid prototyping of mobile ubiquitous computing. In *Proceedings of the 12th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '10, pages 291–298, New York, NY, USA. ACM.

Krumm, J. (2009). *Ubiquitous Computing Fundamentals*. Chapman & Hall/CRC, 1st edition.

Mellis, D. and Buechley, L. (2012). Collaboration in open-source hardware: Third-party variations on the arduino duemilanove. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, CSCW '12, pages 1175–1178, New York, NY, USA. ACM.

Ni, L. M., Liu, Y., Lau, Y. C., and Patil, A. P. (2004). Landmarc: indoor location sensing using active rfid. *Wireless networks*, 10(6):701–710.

Plamondon, R. and Srihari, S. N. (2000). Online and off-line handwriting recognition: a comprehensive survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(1):63–84.

Rooke, A. M. (2001). System for serial peripheral interface with embedded addressing circuit for providing portion of an address for peripheral devices. US Patent 6,304,921.

Shafer, S., Krumm, J., Brumitt, B., Meyers, B., Czerwinski, M., and Robbins, D. (1998). The new easyliving project at microsoft research. In *Proceedings of the 1998 DARPA/-NIST Smart Spaces Workshop*, pages 127–130.

Trapp, B. (2013). Raspberry pi: The perfect home server. *Linux J.*, 2013(229).

Vaughn, J. (2012). Hands-on computing with arduino. *J. Comput. Sci. Coll.*, 27(6):105–106.

Weiser, M. (1991). The computer for the 21st century. *Scientific american*, 265(3):94–104.

Weiser, M. (1994). The world is not a desktop. *interactions*, 1(1):7–8.

Weiser, M. and Brown, J. S. (1997). The coming age of calm technology. In *Beyond calculation*, pages 75–85. Springer.

Wirth, M. and McCuaig, J. (2014). Making programs with the raspberry pi. In *Proceedings of the Western Canadian Conference on Computing Education*, WCCCE '14, pages 17:1–17:5, New York, NY, USA. ACM.