Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Curso de Graduação em Engenharia Elétrica

ANDRÉ ROLIM ALMEIDA GUIMARÃES

PROPOSTA DE UM SISTEMA DE CONTROLE DE ACESSO UTILIZANDO TECNOLOGIA RFID

Campina Grande, Paraíba
Setembro de 2013

ANDRÉ ROLIM ALMEIDA GUIMARÃES

PROPOSTA DE UM SISTEMA DE CONTROLE DE ACESSO UTILIZANDO TECNOLOGIA RFID

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Sistemas de Automação

Orientador:

Professor George Acioli Júnior, M. Sc.

Campina Grande, Paraíba
Setembro de 2013

ANDRÉ ROLIM ALMEIDA GUIMARÃES

PROPOSTA DE UM SISTEMA DE ACESSO CONTROLADO POR TAGS RFID

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Sistemas de Automação

Aprovado em / /

Professor Avaliador
Universidade Federal de Campina Grande
Avaliador

Professor George Acioli Júnior, M. Sc.
Universidade Federal de Campina Grande
Orientador, UFCG

Dedico este trabalho a minha família, que sempre me apoiou nas decisões mais importantes de minha vida.

Agradecimentos

Agradeço a minha família pelo apoio incondicional em toda minha vida. Principalmente em minha vida acadêmica, que mesmo optando por caminhos diferentes dos óbvios, estiveram sempre lá ao meu socorro.

Agradeço principalmente aos meus colegas de curso, no qual realmente se aprende o valor do companheirismo e do apoio nas horas difíceis, aprendido somente vencendo as batalhas lado a lado nestes longos anos de graduação.

Enfim agradeço a todos que de alguma forma contribuíram para a minha formação profissional e pessoal, professores, colegas e funcionários de coordenação.

"Eu posso não ter ido aonde
eu pretendia ir, mas eu acho que
acabei terminando onde
eu pretendia estar."

Douglas N. Adams

Resumo

Sistemas de automação pervasivos são cada vez mais presentes hoje em dia. Com base neste fato, este trabalho de conclusão de curso tem como objetivo descrever um sistema de acesso controlado por cartões RFID. Esta proposta também almeja um sistema de controle de pessoal, com um banco de dados central hospedado em um servidor físico e acessados remotamente por uma interface web, possibilitando também o cadastro e associações de tags RFID a usuários utilizando um microcontrolador arduino. Para projetar um sistema com esta finalidade, o engenheiro de automação necessita ter uma grande base de conhecimento. Neste contexto, este trabalho apresenta um estudo sobre cada tecnologia utilizada na especificação do projeto, que vão desde aspectos de hardware, como microcontroladores, leitores e cartões RFID, até aspectos de engenharia de software como UML e projeto de banco de dados. Por fim a especificação do projeto é apresentada com todos os diagramas e descrições.

Palavras-chave: RFID, Arduino, UML, Sistemas de Automação, Controle de acesso.

Abstract

Pervasive automation systems are becoming more present each year, based on that; the main objective of this work is to describe an access system controlled by RFID tags. Since this project stores user data, the software described in this paper also aims to describe a basic personnel control system, with a central database hosted in a local machine remotely accessed via an web interface. This web application also aims to register and associate RFID tags to users using an Arduino microcontroller .To model and execute such a project, the automation engineer must have a wide knowledge base. In this context, this paper presents a study describing the main points of each technology used in the specification of this project, that range from hardware specific, like RFID tags and Arduino microcontrollers, to software engineering like UML and data base modeling. Finally, all the description and diagrams that enable this project to be reproduced are presented.

Key-words: RFID, Arduino, UML, Automation Systems, Access Control

Lista de Figuras

Figura 1. Tag RFID em formato quadrado 3
Figura 2. Tag RFID dentro de um chaveiro 3
Figura 3. Leitor RFID de mão 6
Figura 4. Leitor RFID acoplado a um PDA 7
Figura 5. Nokia 6131, primeiro celular a possuir tecnologia NFC. 8
Figura 6. Mecanismo de comunicação Near-Field. 10
Figura 7. Mecanismo de funcionamento Far-Field. 12
Figura 8. Arduino MEGA 2560 13
Figura 9. Ambiente de desenvolvimento do Arduino 15
Figura 10. Diagrama de conexões 19
Figura 11. Shield RFID RC522 21
Figura 12. Diagrama simples de caso de uso. 32
Figura 13. Diagrama de caso uso com vários atores 33
Figura 14. Diagrama de classe. 34
Figura 15. Relação de herança 34
Figura 16. Diagrama de classe com associações 35
Figura 17. Diagrama de classes com agregação 35
Figura 18. Diagrama de classes com composição 36
Figura 19. Diagrama de sequência de uma geração de relatórios 37
Figura 20. Diagrama de componentes de projeto 38
Figura 21. Sistema de banco de dados DATE, 2004, P.6 (Adaptado) 39
Figura 22. Cadastro de usuários e tags. 47
Figura 23. Acesso ao ambiente com tags RFID. 47
Figura 24. Formato das informações armazenadas na tag RFID. 48
Figura 25. Arduino Mega 2560. 51
Figura 26. Shield Arduino RC522 52
Figura 27. Tags RFID de 13,56 MHz em dois encapsulamentos diferentes. 53
Figura 28. Conexões entre arduino e shield 54
Figura 29. Caso de uso de cadastro 55
Figura 30. Caso de uso de associações 55
Figura 31. Casos de uso de pesquisas no banco de dados 56
Figura 32. Diagrama de classe para usuários 57
Figura 33. Diagrama de classe DAO 58
Figura 34. Diagrama de Sequência: Criação de usuário 58
Figura 35. Diagrama de Sequência para o cadastro de usuários 59
Figura 36. Diagrama de sequência: Operações RFID 60

Figura 37. Diagrama de componentes do sistema 61
Figura 38. Diagrama do banco de dados 62

Lista de Tabelas

Tabela 1. Modelo de tabela 41
Tabela 2. Exemplo da tabela proposta com registros. 41
Tabela 3. Tabela remodelada com chave primária. 41
Tabela 4. Definição da tabela tipo de usuário.....42
Tabela 5. Tabela de usuário com chave secundaria. 42
Tabela 6. Operadores da cláusula WHERE. 44
Tabela 7. Especificações técnicas de um Arduino Mega 2560 52

Sumário

1 Introdução 1
1.1 Automação no mundo atual 1
1.2 Tecnologia RFID 2
1.3 Soluções Web 4
1.4 Objetivo 4
2 Tecnologia RFID 5
2.1 Introdução 5
2.1.1 Tags RFID 5
2.1.2 Leitores 6
2.1.3 Hosts 6
2.2 Histórico 7
2.3 Funcionamento 8
2.3.1 Near-field RFID 9
2.3.2 Modulação de carga 10
2.3.3 Far-Field RFID 11
3 Plataforma Arduino 12
3.1 Introdução 12
3.2 Histórico 14
3.3 Linguagem de programação 14
3.3.1 A função `setup()` 15
3.3.2 A função `loop()` 16
3.3.3 Tipos de dados suportados 17
3.3.4 Semelhanças com a linguagem C 18
3.3.5 Exemplo prático: Comunicação com a porta serial 18

3.4 Shields 20
4 Programação orientada a objetos 21
4.1 Introdução 21
4.2 Conceitos básicos 22
4.2.1 Classes de objetos 22
4.2.2 Membros de classes 24
4.2.3 Métodos 25
4.3 Encapsulamento 25
4.4 Herança 26
4.4.1 Definição 26
4.4.2 Exemplo (C#) 26
4.5 Polimorfismo 28
4.5.1 Definição 28
4.5.2 Exemplo 28
5 UML 30
5.1 Introdução 30
5.2 Diagrama de caso de uso 32
5.3 Diagrama de classes 33
5.3.1 Herança 34
5.3.2 Associação 35
5.3.3 Agregação 35
5.3.4 Composição 36
5.4 Diagrama de sequência 36
5.5 Diagrama de componentes 37
6 Banco de dados 38
6.1 Introdução 38
6.2 Banco de dados relacionais 39
6.2.1 SGBD 40
6.2.2 Aspecto estrutural 40
6.2.3 Aspecto de integridade 41
6.3 Linguagem SQL 42

6.3.1 Instrução SELECT 43
6.3.2 Instrução INSERT 44
6.3.3 Instrução DELETE 45
6.3.4 Instrução UPDATE 45
7 Sistema de controle de acesso 45
7.1 Especificações de projeto 46
7.1.1 Sistema de grupos 46
7.1.2 Daemon arduino 48
7.1.3 Funcionalidades previstas 49
7.2 Atores 49
7.2.1 Administrador 49
7.2.2 Secretário 49
7.2.3 Professor 50
7.2.4 Aluno 50
7.3 Arduino 51
7.4 Componentes RFID 52
7.4.1 Shield RFID RF522 52
7.4.2 Tags RFID 53
7.5 Diagrama elétrico 54
7.6 Diagramas UML 54
7.6.1 Diagramas de casos de uso 54
7.6.2 Diagramas de classe 56
7.6.3 Diagramas de sequência 58
7.6.4 Diagramas de componentes 60
7.7 Banco de dados 61
7.7.1 Tuplas 61
7.7.2 Diagrama do banco de dados 62
8 Conclusão 63
Bibliografia 65

1 Introdução

1.1 AUTOMAÇÃO NO MUNDO ATUAL

Sistemas de automação são uma parte fundamental na engenharia moderna e possuem aplicações nas mais diversas áreas, desde sistemas complexos como plantas industriais a sistemas de automação residencial. Estes sistemas, sejam eles complexos ou simples, possuem como objetivo principal facilitar de alguma forma o processo em que eles serão inseridos.

A vontade do homem de reduzir o trabalho que seria realizado de forma manual levou a criação do conceito de automação, com o objetivo de eliminar a participação direta do ser humano em sistemas de produção e atividades do dia a dia. Esta vontade esteve presente desde a época de 3500 e 3200 a.C. com a invenção da roda, facilitando assim a locomoção em tempos antigos.

Outro objetivo bastante lógico que se pode inferir na utilização de sistemas automatizados é a maior segurança em processos que antes era necessário a intervenção do homem, sendo estes as vezes completamente substituídos por maquinas e aparelhos que monitoram e controlam sistemas completos, que poderiam apresentar riscos a segurança do operador.

Contudo, o maior impacto que sistemas automatizados exercem sob processos é a do aumento significativo no desempenho, já que como citado anteriormente, com a eliminação parcial do componente humano, sistemas podem atuar de forma muito mais rápida e com uma eficiência que não seria possível se fossem realizados de forma artesanal nas mãos do ser humano.

No mundo moderno com o advento do computador e do avanço na tecnologia, é muito comum se encontrar estes componentes no nosso dia a dia de forma pervasiva, ou seja, de forma que é um fato tão corriqueiro que o homem não nota mais a sua presença. Podemos citar o autoatendimento em caixas eletrônicos de bancos, em caixas de supermercado, controle de acesso entre outros. Todos estes sistemas possuem as características acima citadas, que são de

melhorar desempenho, aumentar a segurança, tentar eliminar o esforço e a presença humana a fim de evitar erros.

Nos dias atuais é bastante comum o controle de acesso por meio de sistemas automatizados, onde o individuo que tenta acessar alguma localidade se identifica ao sistema de diversas maneiras, como por exemplo, senhas, leitores de impressão digital, identificação utilizando cartões RFID e até mesmo scanners de retina.

O objetivo deste projeto de conclusão de curso é a de implementação de um sistema automatizado de controle de acesso para o laboratório de instrumentação eletrônica e controle (LIEC), que se utiliza da tecnologia RFID (*Radio-Frequency Identification*), de forma a automatizar o acesso, já que não mais seria necessária a presença de um porteiro, melhorar a segurança, devido ao fato de que será possível um registro de todas as pessoas que utilizaram o cartão de acesso, como também na rapidez de acesso aos ambientes.

1.2 TECNOLOGIA RFID

Radio-Frequency Identification identificado pela sigla RFID, é uma tecnologia que permite a transferência de informações sem fio através de campos eletromagnéticos, com um propósito de identificar ou rastrear objetos que possuam algum dispositivo RFID presente. Estes dispositivos possuem dimensões pequenas que podem muitas vezes chegar ao tamanho de um grão de arroz.

Estes dispositivos que possuem este mecanismo de identificação são denominados de tags RFID. Estas tags possuem informações armazenadas de forma eletrônica. Tags RFID podem ser lidas desde distâncias muito pequenas, como alguns centímetros, até distâncias maiores como alguns metros.

Tags RFID podem ser energizadas através de campos magnéticos, outras se utilizam de uma bateria, e alguns modelos não possuem qualquer tipo de bateria e utilizam apenas a energia transmitida via ondas magnéticas a partir dos leitores de cartões, e desta forma atuam como componentes passivos que emitem ondas UHF para transmitir os dados.

Tags RFID podem apresentar diversas formas, as figuras 1 e 2 exibem algumas delas.

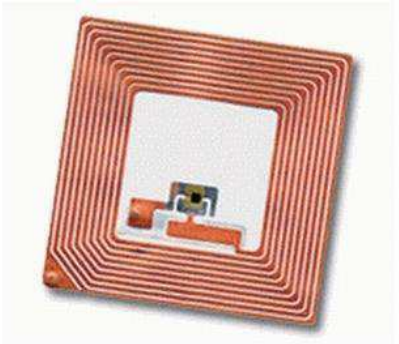

Figura 1. Tag RFID em formato quadrado

Tags podem ser produzidas em formatos ainda mais comuns do nosso dia a dia, abaixo está uma imagem de tags que foram inseridas dentro de chaveiros.

Figura 2. Tag RFID dentro de um chaveiro

Soluções que utilizam a tecnologia RFID possuem várias vantagens sobre outros tipos de identificação, como o código de barras. A primeira delas sendo a vantagem do tamanho e dos formatos variados que tags podem possuir, outra é a condição de que pode-se ler informações destas tags de distancias significativas, pode-se enumerar também com vantagem o fato de que a tag não precisa estar no campo de visão de um leitor para se extrair informações e por fim que uma tag RFID pode conter muito mais informações que não seria possível codificar em um código de barras convencional.

Tags RFID possuem varias aplicações, como por exemplo podemos citar,

i. Rastreamento de um automóvel em uma linha de montagem
ii. Identificação de animais

iii. Em plataformas petrolíferas, tags são utilizadas para identificar trabalhadores.
iv. Identificação em passaportes
v. Controle de acesso

1.3 SOLUÇÕES WEB

O advento da internet é outro fato importante para sistema de automação. Devido ao fato de que o mundo hoje em dia está conectado, sistemas que se utilizam de plataformas que possuem um servidor que se comunica com o processo em que está inserido é de grande vantagem. Pode-se então com a comunicação digital à distância, realizar a total configuração e monitoração de sistemas de automação.

A interface de controle de sistemas por meio de um navegador também é uma vantagem, pois tudo o que se necessita é de uma máquina com acesso a internet e um navegador que dê suporte a qualquer tipo de tecnologia que o site necessite. Portanto evita-se o transtorno de ter que instalar softwares que muitas vezes não são suportados por alguns sistemas operacionais, ou que até mesmo seria necessário modificar o código a fim de se realizar uma portabilidade deste software para outro sistema operacional. Desta forma, o uso de uma interface web reduz muito o custo de operação de sistemas que necessitem de algum tipo de configuração ou monitoração.

Com base nestas vantagens o sistema descrito neste texto implementa uma interface web, que pode ser instalado em uma intranet em algum servidor, que pode ser acessado através de um browser, a fim de facilitar o acesso e permitir a configuração e monitoração por parte de usuários em vários pontos do prédio, podendo ser ou não de forma simultânea, reduzindo também custos na instalação de softwares dependentes de sistemas operacionais e no treinamento de pessoal, visto que um dos objetivos é manter uma interface limpa e de fácil manuseio.

1.4 OBJETIVO

A proposta do projeto que este texto descreve é a de utilizar esta tecnologia de grande importância em um sistema de controle de acesso para o LIEC. As tags seriam então no formato

de um crachá de acesso, em que o usuário se identificaria em leitores montados nas fechaduras das portas.

2 Tecnologia RFID

2.1 INTRODUÇÃO

O acrônimo RFID vem do inglês *Radio-Frequency Identification*, ou identificação por rádio frequência. RFID em um termo geral é usado para descrever sistemas que transmitem identificação (podendo ser na forma de um numero serial único) de um objeto de forma sem fio através de ondas de radio. Estes sistemas são compostos basicamente por três componentes. São eles:

i. Tags
ii. Leitores
iii. Hosts

2.1.1 TAGS RFID

Tag RFID é basicamente um pequeno dispositivo de transmissão de ondas de radio que também pode ser referido como transponder, smart tag, entre outros. Tags são basicamente constituídas de um chip de silício, muitas vezes menor que um milímetro, que por sua vez são conectados a uma antena geralmente em formato de bobina, e por fim, todo o circuito é envolto em algum tipo de encapsulamento.

2.1.2 LEITORES

Leitores RFID, muitas vezes chamados de scanners, são dispositivos capazes de enviar e receber sinais codificados em radio frequência que são emitidas pelas tags. Um leitor pode chegar a ter múltiplas antenas acopladas que são responsáveis por emitir estas ondas de rádio.

A figura 3 abaixo exibe um tipo de leitor de tags RFID.

Figura 3. Leitor RFID de mão

2.1.3 HOSTS

Hosts são os sistemas que irão controlar e processar os sinais enviados pelos leitores. Estes hosts podem ser um computador, smartphone, tablet ou qualquer outro dispositivo capaz de processamento. Cabe então ao engenheiro de controle programar o host para realizar as leituras, processar os sinais e tomar uma ação com base nos dados coletados.

A figura 4 exibe um leitor RFID acoplado a um host do tipo PDA.

Figura 4. Leitor RFID acoplado a um PDA

2.2 HISTÓRICO

Como muitas tecnologias presentes na atualidade, os primeiros indícios de identificação via ondas de rádios se deram na segunda guerra mundial. Durante a guerra, os alemães, japoneses e americanos estavam usando radares, que tinha sido descoberto em 1935 pelo físico escocês Sir Robert Alexander Watson, para detectar a presença de aviões a quilômetros de distância. O problema era que não havia como identificar os aviões de cada exército. Os alemães então descobriram que se os aviões fizessem algum tipo de manobra, como virar a aeronave, o sinal refletido pelo radar seria modificado, sendo assim, este método primitivo foi o primeiro relato de identificação por radio frequência.

Durante a guerra os britânicos desenvolveram então o primeiro sistema de identificação de inimigos intitulado de *Identify Friend or Foe* (IFF) *system.* O sistema consistia basicamente de um transmissor que era acoplado a cada aeronave, que no caso de receber um sinal, este dispositivo transmitia um sinal de volta, que identificava a aeronave como sendo um aliado.

Avanços continuaram sendo feitos durante as décadas de 50 e 60 nos Estados Unidos e no Japão, mas foi somente nos anos 70 que a primeira patente de RFID foi registrada. Em 1973, Mario W. Cardullo diz ter recebido a primeira patente sobre a tecnologia RFID emitida pelo governo americano. Neste mesmo ano na Califórnia, Charles Walton recebe uma patente para

um sistema que usava um transponder passivo usado para destravar uma porta de maneira sem fio.

Anos depois, companhias de engenharia desenvolveram sistemas que operavam em baixa frequência (125 kHz), que possuíam transponders cada vez menores. Estes transponders eram encapsulados em vidro e por sua vez eram introduzidos embaixo do couro de animais, tipicamente gado, e por sua vez permitiam a identificação e o rastreamento desses animais. Por sua vez ao longo dos anos, a tecnologia RFID foi evoluindo, e sistemas então eram capazes de operar em altas frequências (13,56 MHz). Estes sistemas em frequências mais altas possibilitavam um maior alcance e uma maior taxa de transferência de dados nas leituras de tags. Estes sistemas eram muito utilizados para a identificação de containers reutilizáveis em transportes de grandes cargas. Em 1990, engenheiros da IBM desenvolveram sistemas que operavam em frequências mais altas chamadas de UHF (*Ultra High Frequency*), e que permitiam leituras a distâncias ainda maiores (até 6 metros em boas condições) e taxas de transferências ainda mais altas.

Hoje se pode notar a presença de sistemas que utilizam tags e leitores RFID por vários lugares, os mais modernos porem são os smartphones que possuem tecnologias NFC (*Near Field Communication*), que são um conjunto de especificações que usam a tecnologia RFID para a transmissões de informações sem fio a curtíssimas distâncias. Estes dispositivos operam em uma frequência de 13,56 MHz.

Figura 5. Nokia 6131, primeiro celular a possuir tecnologia NFC.

2.3 FUNCIONAMENTO

Basicamente existem dois tipos de tags RFID, tags passivas ou ativas. Tags ativas dependem de uma fonte de energia externa, o que eleva seu custo e também diminui o tempo de vida da tag, pois este tempo está diretamente relacionado ao tempo útil de vida da bateria. Este tipo de tag pode ser encontrado em transponders que são instalados em aviões. Tags do tipo passiva por sua vez não necessitam de uma fonte de alimentação externa, este tipo de tag retira sua energia necessária para funcionamento a partir das ondas eletromagnéticas emitidas pelo leitor. Este tipo de tag então é muito mais interessante, pois o custo de fabricação é bastante reduzido e a tag possui um numero ilimitado de leituras.

Tags passivas são constituídas basicamente de três componentes, uma antena, um chip semicondutor responsável pelo processamento dos sinais recebidos e emitidos, e um tipo de encapsulamento que protege a antena e o chip contra reagentes externos. Estas tags podem se utilizar de duas propriedades eletromagnéticas para se energizarem, a indução eletromagnética ou a captura de ondas eletromagnéticas. Estas tags se utilizam destes princípios, para gerarem potências na faixa de 1μW a 10mW, capazes de energizar o circuito para o recebimento e envio de sinais.

2.3.1 NEAR-FIELD RFID

Near-Field RFID, do inglês RFID de campos próximos, tem o seu funcionamento baseados na lei de *Faraday*. O principio da lei de Faraday de indução magnética diz que a corrente elétrica induzida em um circuito fechado por um campo magnético é proporcional ao numero de linhas do fluxo que atravessa a área envolvida no circulo, por unidade de tempo. Em termos práticos, quando um leitor RFID aplica uma alta corrente alternada por um circuito fechado definido por uma bobina de leitura, um campo magnético alternado será gerado nas proximidades do leitor. Quando se coloca uma tag próxima ao leitor, a antena da tag, que é uma bobina, será envolta pelo campo induzido pelo leitor e uma tensão será criada na tag. Retificando esta tensão e acoplando um capacitor, um acúmulo de cargas será armazenada, que em seguida será utilizada para energizar o circuito e o chip da tag.

O limite de operação deste tipo de tag é definido pelo fator de acoplamento entre a antena do leitor e a antena da tag. Se o tamanho da antena da tag for muito pequeno, a distância de operação vai caindo, até que não será mais possível a comunicação com o leitor. Esta distância

também é proporcional a frequência de operação, ou em termos de frequência angular definida pela equação,

$$\frac{c}{\omega} = \frac{c}{2\pi f}$$

Quanto mais se aumenta a frequência de operação, menor será à distância de operação de sistemas que usam este tipo transmissão. Outra limitação deste tipo de aplicação é que a quantidade de energia disponível em função da distância da bobina do leitor decai com uma taxa de $1/r^3$, em que r é a separação entre o leitor e a tag. O esquema abaixo descreve o funcionamento do acoplamento magnético.

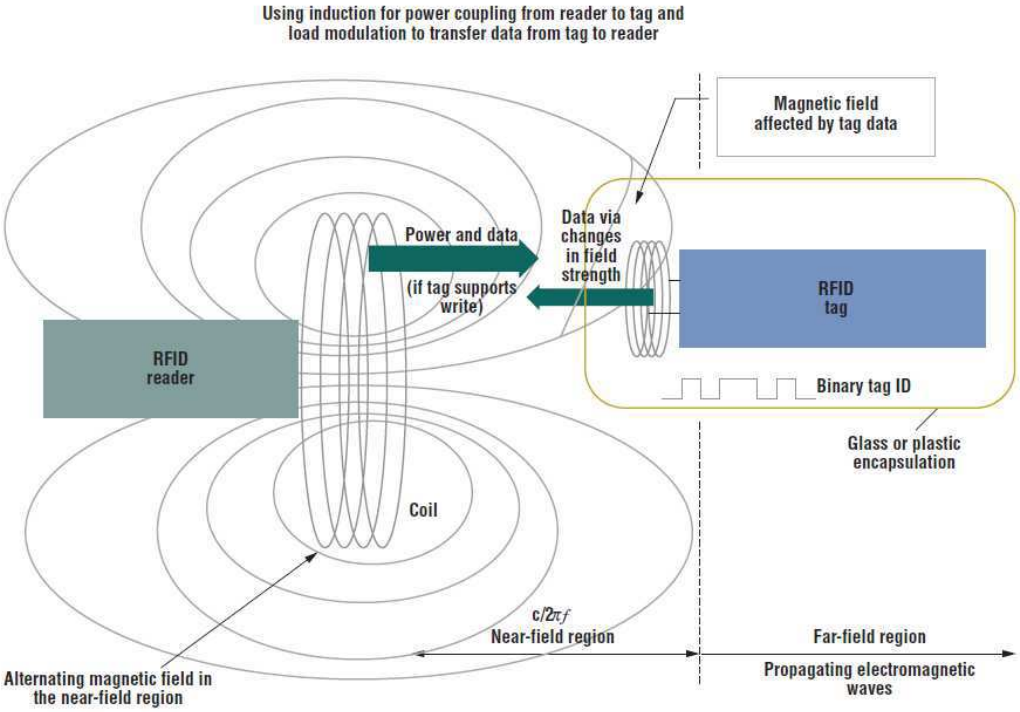

Figura 6. Mecanismo de comunicação Near-Field.

2.3.2 MODULAÇÃO DE CARGA

Mensagens que são transmitidas entre o leitor e a tag, que usam o acoplamento magnético definido na lei de Faraday, são moduladas usando a técnica de modulação de carga. Estes dispositivos operam geralmente na frequência de 13,56 MHz e estes padrões são definidos

principalmente nos padrões ISO/IEC 14443, 15693, 18000-3 e 18092, porém a maioria das aplicações é definida no padrão 14443.

A modulação de carga é baseada no fato em que qualquer corrente que seja absorvida da bobina da tag, acarretará na criação do seu próprio campo magnético, que por sua vez ira se opor ao campo magnético aplicado pelo leitor. O leitor então pode detectar este efeito como um pequeno aumento na corrente que flui por ele. O chip da tag, pode então codificar a informação a ser transmitida, variando no tempo a carga aplicada a sua antena.

Para se realizar a modulação de carga, uma resistência é conectada em paralelo com a antena da tag, esta que será ligada e desligada seguindo a taxa do clock da mensagem a ser enviada. O padrão ISO/IEC 14443 especifica que o resistor de modulação é chaveado a uma frequência de subportadora de 848 kHz. A subportadora utiliza modulação ASK e é codificada utilizando o código Manchester.

2.3.3 FAR-FIELD RFID

Tags RFID do tipo Far-Field operam capturando ondas eletromagnéticas, propagadas por uma antena dipolo acoplada ao leitor. Uma antena dipolo menor acoplada a tag recebe esta energia como uma diferença de potencial sob os polos da antena. Um simples diodo pode retificar este sinal, que em seguida é conectado a um capacitor. Este capacitor será carregado e então poderá suprir energia e ativar o circuito.

Porém, o esquema de transmissão de informação não pode ser baseado na modulação de carga, pois a tag estará muito longe do campo magnético gerado pelo leitor. A solução para este problema foi usar a técnica de *back-scattering*. Para isto, projetistas devem criar uma antena com dimensões precisas para que sejam “sintonizadas” em uma frequência especifica e que absorvam a maior quantidade de energia de sinais transmitidos nesta frequência. Portanto, se ocorrer uma diferença de impedância nesta frequência, a antena ira refletir de volta uma parte desta energia em direção ao leitor, que por usa vez ira detectar estas ondas usando um receptor de radio muito sensível. A tag então varia a sua impedância no tempo, variando então a quantidade de energia que é refletida para o leitor. Para se transmitir informações de volta, a tag varia sua impedância num padrão que codifique a mensagem.

Tags que utilizam este tipo de transmissão de energia e codificação operam em frequências maiores que 100 MHz, tipicamente nas bandas UHF (*Ultra High-Frequency*), como por exemplo, em 2,4 GHz. O esquema da figura 7 ilustra o funcionamento de tags Far-Field.

Obviamente as limitações deste tipo de sistema se dão pela quantidade de energia que é capturada pela tag, e pela quantidade de energia que é refletida pela tag para o leitor, como também pelo o quão sensível a antena receptora será para as ondas refletidas pela tag. Outra limitação que pode ser mencionada é o nível de atenuação que se obtém, pois o sinal que vai da tag para o receptor é o resultado de duas atenuações, uma quando as ondas são propagadas do leitor para a tag, e outra quando as ondas são refletidas para o leitor. Esta energia então ira decair com uma taxa de $1/r^4$, onde r é a distância entre as duas antenas.

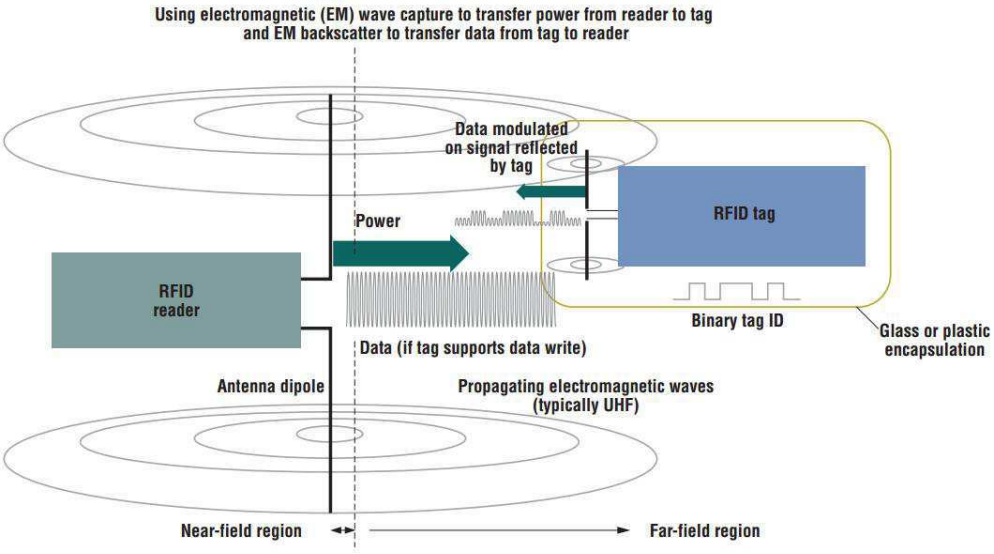

Figura 7. Mecanismo de funcionamento Far-Field.

3 Plataforma Arduino

3.1 INTRODUÇÃO

Arduino é uma plataforma eletrônica open-source, feita para a realização de protótipos de projetos que necessitem de um microcontrolador. Esta plataforma de desenvolvimento possui uma grande vantagem que é a facilidade de uso e aprendizagem, por isto é um dos

microcontroladores mais utilizados por designers e engenheiros que necessitem de uma solução pratica e rápida para realizar o controle de dispositivos eletrônicos.

Existem vários tipos de arduinos, com o ponto em comum que todos possuem alguma variação do microprocessador da série ATmega da empresa Atmel Corporation, uma fabricante de dispositivos semicondutores fundada em 1984. Placas arduino possuem vários pinos de conexões que facilita bastante o trabalho de prototipagem de produtos, como por exemplo, portas seriais, interrupções externas, PWM, SPI, LEDs, entradas e saídas analógicas e digitais entre outros. Tipicamente a conexão entre o arduino e o computador em que se deseja programa-lo é feita através da porta serial do arduino que é ligada na porta USB do computador.

A imagem abaixo ilustra o Arduino Mega 2560, modelo escolhido para a realização do projeto descrito neste trabalho de conclusão de curso.

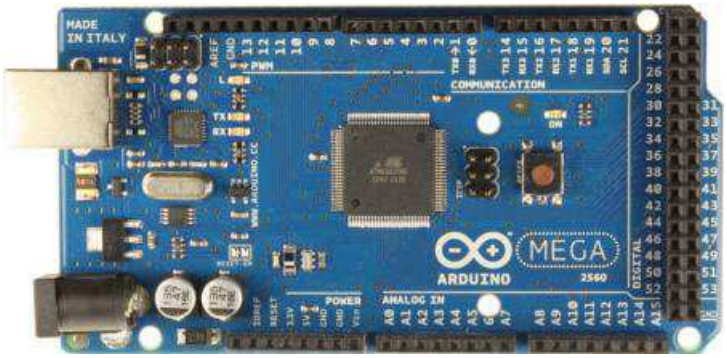

Figura 8. Arduino MEGA 2560

Protótipos feitos com arduinos podem ser *stand-alone* ou conectados via porta serial a um computador para realizar o controle. Como a plataforma arduino possui varias entradas digitais e analógicas, é possível conectar vários sensores capazes de coletar informações, e em seguida processar estas informações seguindo uma estratégia de controle programada no próprio arduino. O microcontrolador na placa é programado em uma linguagem própria, baseada na linguagem *Wiring*¸ muito semelhante à linguagem C.

3.2 HISTÓRICO

A plataforma de desenvolvimento arduino é bem recente, começou no ano de 2005 como um projeto para estudantes do instituto de design Ivrea, na Itália. O projeto começou seguindo a tendência DIY, que significa "*Do It Yourself"*, ou "Faça você mesmo", que vinha crescendo muito devido aos microcontroladores PIC e Wiring. Surgiu também como uma resposta ao desejo de artistas e designers de quererem fazer obras interativas, o que levou a tese de mestrado do artista e designer Hernando Barragan em 2004, que tinha a intenção de levar este tipo de controle para uma audiência não técnica de artistas, designers e arquitetos. Em 2005 o time do arduino foi formado por Barragan, Massimo Banzi, David Cuartielles, Dave Mellis, Gianluca Marino e Nicholas Zambetti.

O objetivo dos criadores do arduino era de criar uma plataforma de protótipos para projetos de controle que simplificasse ainda mais a plataforma *Wiring*, e que fosse o mais acessível possível para usuários não técnicos, facilitando o desenvolvimento de projetos e criações de artes interativas.

A plataforma arduino atingiu um sucesso muito rápido nos dois primeiros anos de existência, vendendo mais de 50.000 unidades. Hoje em dia existem vários tipos de arduinos, pois como a plataforma é open-source, os esquemas do hardware são abertos, e vários projetistas podem adaptar o circuito para suas necessidades. Podem-se citar vários exemplos como o arduino "Lilypad", feito especialmente para projetos que possam ser usados em roupas, ou o arduino BT, que já possui um dispositivo de conexão Bluetooth embarcado no circuito.

3.3 LINGUAGEM DE PROGRAMAÇÃO

A linguagem de programação utilizada para programar o microcontrolador embarcado no arduino é bastante parecida com C/C++, dando suporte até mesmo a criação de classes. Esta linguagem é baseada na linguagem *Wiring*, possuindo uma pequena curva de aprendizado e um tempo muito curto para o desenvolvimento de aplicações simples.

O ambiente de desenvolvimento (IDE) é uma aplicação feita na linguagem de programação Java, e pode ser obtida de forma gratuita no site oficial da plataforma arduino. O

ambiente de desenvolvimento possui varias funcionalidades, como um compilador que traduz código de alto nível para linguagem de maquina pronto para ser escrito no microcontrolador, possui também um terminal serial, capaz de se comunicar com o arduino enviando e recebendo mensagens, e por fim pode-se configurar de forma simples e rápida o arduino a partir do próprio ambiente de desenvolvimento, definindo portas seriais a serem utilizadas e o tipo de arduino a ser utilizado.

A imagem abaixo exibe a tela inicial do ambiente de desenvolvimento.

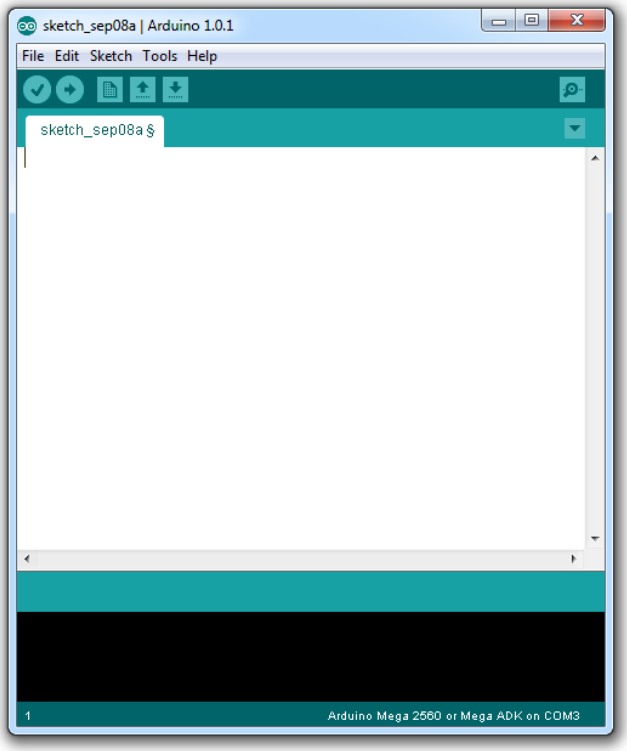

Figura 9. Ambiente de desenvolvimento do Arduino

Em seguida serão discutidas as principais características da linguagem arduino, como também exemplos práticos.

3.3.1 A FUNÇÃO SETUP( )

Existem alguns termos que a linguagem arduino adota para seus códigos. O primeiro deles é a terminologia *sketch*, que vem do inglês e significa “esboço”. Cada projeto realizado no arduino se chama um *sketch*.

A função `setup()` é chamada assim que um sketch é iniciado, e é nesta função que deve-se realizar a inicialização de variáveis, modos de pinos, carregar bibliotecas ou iniciar alguma configuração necessária para o projeto. Esta função será executada apenas uma vez. Esta execução será chamada quando o arduino for energizado ou se o arduino sofrer uma ação de reset.

O trecho de código abaixo foi retirado da documentação oficial do arduino, disponível no site oficial (arduino.cc).

```
1.     const int buttonPin = 3;
2.
3.     void setup()
4.     {
5.       Serial.begin(9600);
6.       pinMode(buttonPin, INPUT);
7.     }
```

A linha numero 1 define uma variável que no contexto do código indica o numero do pino que será utilizado. A linha 3 define a função `setup()` que será chamada assim que o arduino for energizado. A linha 5 define o *baudrate* que será utilizado na porta serial e inicia a comunicação. A linha 6 determina o modo do pino definido na linha 1, como sendo do tipo INPUT, ou seja, este pino vai servir como uma entrada de dados.

3.3.2 A FUNÇÃO `LOOP()`

O próximo passo após a função setup ser implementada, é implementar a função `loop()`. Esta função será executada logo após a função `setup()` e será executada em um loop infinito até que o arduino seja desligado, ou seja, todo o código que será executado estará nesta função. A maneira que esta função está implementada no código interno do arduino, é da forma que o ambiente do arduino define uma função `main()`, da mesma maneira que na linguagem de programação C, e em seguida chama a função loop indefinitivamente. Caso a função `loop()` retorne, a função `main()` também retornara e o programa será encerrado.

O trecho de código abaixo demonstra o uso da função `loop()`.

```
1.     const int buttonPin = 3;
2.
```

```
3.      void setup()
4.      {
5.       Serial.begin(9600);
6.          pinMode(buttonPin, INPUT);
7.      }
8.
9.      void loop()
10.     {
11.         if (digitalRead(buttonPin) == HIGH)
12.             Serial.write('H');
13.         else
14.             Serial.write('L');
15.
16.         delay(1000);
17.     }
```

As linhas de 1 a 7 são idênticas ao exemplo anterior. A linha 9 declara a função `loop()`, a linha 11 utiliza a condição "if" para testar o estado do pino. Se ele estiver em nível alto, o programa ira escrever na porta serial a letra "H", como descrito na linha 12, caso o pino esteja em nível lógico baixo, o programa vai escrever a letra "L" na porta serial. A linha 16 faz o programa parar de rodar por 1000 milissegundos, em seguida a função `loop()` será executada novamente.

3.3.3 TIPOS DE DADOS SUPORTADOS

A linguagem arduino da suporte a vários tipos de dados. Estes por sua vez também são suportados pela maioria das linguagens de programação, facilitando assim o aprendizado de engenheiros que estão tendo um primeiro contato com esta linguagem. Os tipos de dados suportados são:

i. void
ii. boolean
iii. char
iv. unsigned char
v. byte
vi. int
vii. unsigned int

viii. word
ix. long
x. unsigned long
xi. short
xii. float
xiii. double
xiv. string (vetor de chars)
xv. String (objeto do tipo string)
xvi. array

3.3.4 SEMELHANÇAS COM A LINGUAGEM C

Podemos notar a semelhança que a linguagem de programação arduino tem com varias linguagens de programação, principalmente com a linguagem C. Podem-se listar então estas semelhanças, a fim de contextualizar ainda mais esta linguagem e facilitar o entendimento de código para engenheiros com alguma experiência prévia com linguagens de programação. A linguagem arduino possui então as seguintes estruturas em comum com C:

i. Estruturas de controle (if, then, else, while, do while etc)
ii. Definição de variáveis através de #define
iii. Operações aritméticas
iv. Comparadores (== , != , <, >, <=, >=)
v. Operadores bit a bit (&, |, ^, ~)
vi. Funções trigonométricas (sin(), cós(), tan())

3.3.5 EXEMPLO PRÁTICO: COMUNICAÇÃO COM A PORTA SERIAL

O exemplo demonstrado a seguir define um programa que recebe uma sequência de números inteiros da porta serial, separados por vírgula, em seguida o exemplo define a cor de um LED RGB a partir dos valores lidos. Este programa demonstra o uso também da função `Serial.parseInt()`, que realiza a leitura de valores inteiros na porta serial. Este exemplo pode ser encontrado na documentação oficial disponibilizado no site oficial do arduino.

A imagem a seguir ilustra as conexões que devem ser feitas entre o arduino e um protoboard contendo 3 resistores conectados a cada conector do LED RGB, e que por sua vez estão conectados com o arduino.

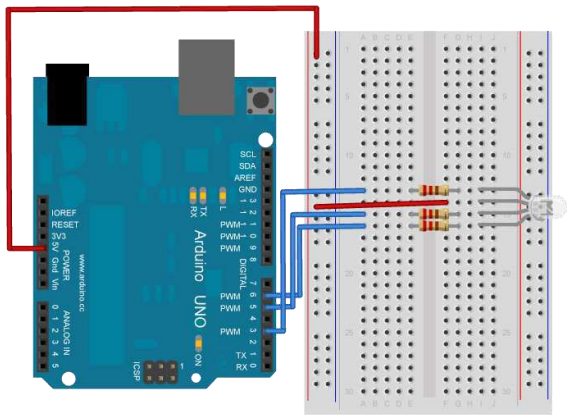

Figura 10. Diagrama de conexões

O código fonte para este exemplo esta abaixo.

```
1.    const int redPin = 3;
2.    const int greenPin = 5;
3.    const int bluePin = 6;
4.
5.    void setup() {
6.      Serial.begin(9600);
7.      pinMode(redPin, OUTPUT);
8.      pinMode(greenPin, OUTPUT);
9.      pinMode(bluePin, OUTPUT);
10.
11.   }
12.
13.   void loop() {
14.     while (Serial.available() > 0) {
15.       int red = Serial.parseInt();
16.       int green = Serial.parseInt();
17.       int blue = Serial.parseInt();
18.
19.       if (Serial.read() == '\n') {
20.         red = 255 - constrain(red, 0, 255);
21.         green = 255 - constrain(green, 0, 255);
22.         blue = 255 - constrain(blue, 0, 255);
23.
24.         analogWrite(redPin, red);
25.         analogWrite(greenPin, green);
26.         analogWrite(bluePin, blue);
27.
28.         Serial.print(red, HEX);
29.         Serial.print(green, HEX);
30.         Serial.println(blue, HEX);
```

```
31.         }
32.       }
33.     }
```

As linhas de 1 a 3 definem os pinos a serem utilizados, que correspondem aos pinos no diagrama de conexões. A função `setup()` é então declarada, e nela a porta serial é inicializada e os pinos são configurados como saídas. Na linha 13 a função `loop()` é declarada, e em seguida uma estrutura condicional do tipo `while()` é definida. A condição é de que enquanto existir dados na porta serial para serem lidos, o código contido na estrutura será executado. As linhas de 15 a 17 utilizam a função `Serial.paserInt()`, que procura por números inteiros no buffer da porta serial. A linha 19 procura pela condição de próxima linha, que indica o fim da cadeia de inteiros passados na porta serial. As linhas 24 a 26 colocam as saídas dos pinos que estão conectadas com o LED de acordo com os números recebidos na porta serial e as linhas 28 a 30 escrevem os números lidos de volta na porta serial para o programador conferir o funcionamento do programa.

3.4 SHIELDS

Como a plataforma arduino foi concebida de forma que suas conexões estão expostas em sua parte superior, vários fabricantes produzem placas que podem ser facilmente conectadas e desconectadas em cima do arduino. Estas placas são chamadas de *shields*, que vem do inglês e significa “escudos”. Alguns shields estão listados abaixo.

i. Touchscreen
ii. Display de cores
iii. Joystick
iv. Xbee
v. WiShield
vi. Ethernet

Este projeto procura então utilizar a plataforma arduino para realizar a leitura de tags RFID. Portanto, um shield de escrita e leitura RFID foi escolhido para isto. A imagem abaixo ilustra o shield escolhido.

Figura 11. Shield RFID RC522

4 Programação orientada a objetos

4.1 INTRODUÇÃO

Programação orientada a objetos é um paradigma de programação que representa vários conceitos que são aplicados na construção de softwares. Estes conceitos estão baseados no conceito de objetos que possuem métodos de ação, que por sua vez são definidos em estruturas chamadas classes. Estes objetos possuem também campos de dados, chamados de membros de classe. Esses objetos são então instanciados e utilizados em um programa.

Estes conceitos auxiliam o programador de forma que este pode abstrair todo o modelo do programa em objetos, deixando o objetivo e a maneira como foi escrito o código muito mais fácil de manter, além de conferir uma grande modularidade ao sistema.

A programação orientada a objetos é baseada então em três conceitos muito poderosos. O encapsulamento, a herança e o polimorfismo. Esses três conceitos então ajudam ainda mais o programador a manter um nível mais alto de abstração para o código.

Este capítulo tem o objetivo de descrever estes conceitos a fim de se estabelecer uma base teórica que foi amplamente utilizada no decorrer deste projeto. Como a linguagem de

programação escolhida para a implementação do software foi C#, todos os exemplos serão então realizados nesta linguagem.

4.2 CONCEITOS BÁSICOS

4.2.1 CLASSES DE OBJETOS

Classes são estruturas que podem definir objetos, bem como também a maneira que eles irão se comportar. Uma maneira clássica de se explicar este conceito, e muito difundida na literatura, é através de exemplos que estão inseridos no nosso dia a dia.

Supondo uma analogia entre a orientação a objetos e um carro, se uma pessoa deseja dirigir um carro, antes de tudo alguém precisa projetar o carro. Portanto um designer de carros iria criar um modelo, dando a ele atributos como cor, potencia do motor, tipo de direção e até mesmo o nome do carro. A próxima ação do projetista seria determinar as ações que o carro iria poder tomar. De forma simples pode-se dizer que um carro pode ir para frente, para trás, parar, limpar o para-brisa entre outras ações. Pode-se também definir o estado em que o carro se encontra, como por exemplo, parado, andando, cintos afivelados ou não. Todas estas informações então definem a classe de um carro. Se alguém deseja utilizar este carro, uma instância deste carro será criada na forma de um objeto e então esta pessoa poderá dirigi-lo.

O código abaixo define uma classe em C# chamada carro para exemplificar o conceito de classes.

```
1.      class Carro
2.         {
3.             public string nome { get; set; }
4.             public string modelo { get; set; }
5.             public string cor { get; set; }
6.             public Motor motor { get; set; }
7.
8.             public bool carroParado = false;
9.             public bool cintosAfivelados = false;
10.
11.            public void irParaFrente(int velocidade)
12.            {
13.            }
14.            public void irParaTras(int velocidade)
```

```
15.             {
16.             }
17.             public void parar()
18.             {
19.             }
20.             public void limparParaBrisa()
21.             {
22.             }
23.         }
```

Definimos então uma classe chamada `Carro` que possui alguns membros de classe e algumas ações definidas. Na linha 6, temos um tipo de objeto chamado `Motor` que também foi definido em uma classe.

Para utilizar o carro é preciso criar uma instância deste carro, definir seus membros de classes, que também podem ser chamados de atributos, e em seguida chamar alguma ação do carro. Em C# é possível realizar isto na função `Main()` que é chamada assim que o programa é iniciado.

```
1.      class Program
2.         {
3.             static void Main(string[] args)
4.             {
5.                 Carro meuCarro = new Carro();
6.                 meuCarro.cor = "Preto";
7.                 meuCarro.modelo = "Sandeiro";
8.                 meuCarro.nome = "Meu carro";
9.                 meuCarro.motor = new Motor();
10.
11.                meuCarro.irParaFrente(30);
12.                meuCarro.parar();
13.            }
14.        }
```

É possível então notar o grande poder que este conceito implica como também o nível de abstração que se pode usar na modelagem de softwares.

4.2.2 MEMBROS DE CLASSES

Como foi exemplificado anteriormente, membros de classe são campos definidos que dão características a classe que se esta modelando. Cada instância de classe possui os seus próprios membros de classe, que então irão diferenciar uma instância de uma classe de outra. Tomando como base o exemplo do carro, cada carro vai saber que tipo de motor ele possui, qual o seu modelo e o seu nome. Mas cada instância de carro não saberá as informações de outros carros.

Pode-se notar na classe `Carro` que todos os seus atributos foram definidos como públicos, ou seja, todo usuário do carro terá acesso às informações que foram marcadas como publicas. Porém existem certas características que não são de interesse do usuário, portanto devem permanecer escondidas. Em C# é possível realizar esta *encapsulação* através da palavra chave `private`.

O trecho de código abaixo modifica a classe Motor para exemplificar o uso da palavra chave `private`.

```
1.      class Motor
2.        {
3.            private string tipoDeVelas { get; set; }
4.            private string tipoDeCilindro { get; set; }
5.            private string tipoDeValvula { get; set; }
6.            private string tipoDePistão { get; set; }
7.
8.            public void ligarMotor()
9.            {
10.           }
11.           public void desligarMotor()
12.           {
13.           }
14.       }
```

Os membros de classe da classe `Motor` foram definidos como `private`, portanto eles não serão capazes de serem acessados diretamente em suas instâncias, como foi feito na classe `Carro`. Porém, os métodos estão definidos como `public`, e estes poderão ser chamados a partir do método `Main()` ou de qualquer outra classe que venha a ter um atributo do tipo `Motor`.

4.2.3 MÉTODOS

Métodos de classe são as ações que a classe pode realizar. Nos exemplos anteriores temos que a classe do tipo `Carro` possui alguns métodos definidos, estes que podem então ser chamados a partir de suas instâncias.

A mesma regra que se usou para encapsularmos membros que não se deseja expor a usuários de instancias pode também ser utilizado com métodos de classe, utilizando a palavra chave `private` antes das declarações de métodos. Desta forma, estas ações só serão possíveis de serem chamadas dentro das próprias classes.

```
1.          public void ligarCarro()
2.          {
3.              iniciarParteEletrica();
4.          }
5.
6.
7.          private void iniciarParteEletrica()
8.          {
9.          }
```

Adicionando estes dois métodos a classe `Carro`, nota-se que o método `ligarCarro()` é definido como `public`, e o método `iniciarParteEletrica()` é definido como `private` e só pode ser chamado em métodos da própria classe.

4.3 ENCAPSULAMENTO

Em orientação a objetos existe a noção de encapsulamento. Este termo é usado para definir o fato que pela maneira que este paradigma de programação define suas classes, as informações definidas podem ficar escondidas do usuário, ou seja, o acesso aos membros da classe fica restrito.

Em C# isto é realizado através da palavra chave `private` ou `protected`. Para algumas linguagens de programação como C++, somente pelo fato de membros de classe possuírem estas restrições de acesso, não impede o usuário de conhecer a estrutura interna da definição de uma classe, pois mesmo que não se tenha o código fonte de um programa ou uma biblioteca a ser usada, é necessário possuir os arquivos de cabeçalho, e são nesses arquivos em

que as classe são definidas. Em C# isto não é verdade, pois pode-se usar somente o arquivo DLL para se usar uma biblioteca, mantendo assim a estrutura interna de uma classe completamente escondida do usuário final.

Porém o principal motivo de se definir membros de classe como private é o de aumentar a robustez de um programa. Quando o engenheiro de software restringe o acesso a certos membros de classe, pode-se evitar que o programa entre em algum estado inválido ou inconsistente. Muitas vezes restringe-se o acesso direto a todos os membros de classe, sendo a única maneira de se modificar estes atributos é a utilização de métodos chamados de “getters” e “setters”. Estes métodos tem o papel somente de definir e retornar os valores dos membros de classe, dando então o controle total à classe de verificar se o valor a ser atribuído a algum membro de classe vai levar o programa a algum estado invalido.

4.4 HERANÇA

4.4.1 DEFINIÇÃO

Talvez uma das características mais poderosas da programação a orientada a objetos seja o mecanismo de herança. Isto ocorre quando uma nova classe é criada a partir de outra classe, em que esta nova classe irá “absorver” membros de classe e métodos da classe do qual ela foi criada. Tem-se então uma forma de reuso de código, que é um dos principais objetivos da programação moderna, pois se podem reutilizar códigos já definidos em um contexto de forma a especializar ainda mais a classe a ser herdada.

O conceito de herança também é importante para a modelagem de programas, pois se pode então dar um maios contexto a todo o código. Pode-se demonstrar isto a partir de um exemplo.

4.4.2 EXEMPLO (C#)

Supondo que um programador esteja implementando um programa que necessite desenhar formas geométricas na tela, o programador pode então definir uma classe genérica a qual outras classes irão herdar. O termo em C# e em outras linguagens para esta classe é a de

classe base. Voltando ao exemplo, um nome apropriado para esta classe base seria de `FormaGeometrica`. Modelando então uma forma geométrica, o engenheiro de software vai definir então membros de classes apropriados. Esta classe esta definida no trecho de código abaixo.

```
1.          class FormaGeometrica
2.          {
3.              public int X { get; set; }
4.              public int Y { get; set; }
5.              public int Altura { get; set; }
6.              public int Largura { get; set; }
7.
8.              public FormaGeometrica() { }
9.              ~FormaGeometrica() { }
10.         }
```

A classe `FormaGeometrica` possui então alguns membros de classe que definem seu comportamento, como por exemplo sua posição na tela e as suas dimensões. No âmbito de modelagem, nota-se que estas características são comuns a todas as formas geométricas, portanto estas características podem ser reutilizadas por modelos de objetos que antes de tudo são formas geométricas.

A partir da classe base, se pode então definir outras classes que podem herdar da classe `FormaGeometrica`. Exemplos de formas geométricas são triângulos, círculos, quadrados etc. O código abaixo demonstra então como é possível herdar da classe `FormaGeometrica`, e reaproveitar seu código.

```
1.          class Circulo : FormaGeometrica
2.          {
3.              public Circulo() { }
4.              ~Circulo() { }
5.
6.              private string cor;
7.          }
8.
9.          class Triangulo : FormaGeometrica
10.         {
11.             public Triangulo() { }
12.             ~Triangulo() { }
13.
14.             public void arrendondarPontas()
15.             {
16.                 //Código para arredondar pontas de um triângulo.
```

```
17.             }
18.         }
```

As classes que herdam da classe base absorvem todos os seus membros, exceto os construtores e o destrutor. Uma classe que herda de uma classe base possui a denominação de *classe derivada*. Uma classe derivada é então uma especialização da classe base, podendo assim definir novos membros de classe e métodos que se acomodem a modelagem do contexto em que o software está sendo construído. Portanto, os membros de classe `X`, `Y`, `Altura` e `Largura` também estão presentes nas classes derivadas, e podem ser utilizados normalmente no programa como se elas estivessem definidas na própria classe.

4.5 POLIMORFISMO

4.5.1 DEFINIÇÃO

O quarto conceito da programação orientada a objetos é o de polimorfismo. A palavra polimorfismo vem do grego e significa “varias formas”.

Na programação orientada a objetos este conceito está relacionado ao fato de que quando uma classe derivada herda de uma classe base, no âmbito do programa, esta instância de classe pode ser tratada como um objeto do tipo da classe base. Isto é muito poderoso, e confere muita flexibilidade ao programa, pois se pode conferir uma maior uniformidade ao código. Outro aspecto deste conceito é que classes base podem definir métodos virtuais, ou seja, que podem ou não possuírem nenhuma implementação, e em seguida classes derivadas podem sobrescrever estes métodos e definirem suas próprias implementações.

4.5.2 EXEMPLO

Voltando ao exemplo dado na seção anterior, supondo que o programador queira definir um método que desenhe uma forma geométrica na tela, porém ele não sabe qual o tipo de forma geométrica será desenhada por vez. Se pode resolver este problema usando o conceito de polimorfismo e definido um método virtual `Desenhar()`. Em seguida as classes derivadas

reimplementam este método se adequando para cada tipo de forma. O trecho de código abaixo demonstra isto.

```
1.          class FormaGeometrica
2.          {
3.              public int X { get; set; }
4.              public int Y { get; set; }
5.              public int Altura { get; set; }
6.              public int Largura { get; set; }
7.
8.              public FormaGeometrica() { }
9.              ~FormaGeometrica() { }
10.
11.             public virtual void Desenhar()
12.             {
13.             }
14.         }
```

Em seguida as classes derivadas vão reimplementar o método virtual `Desenhar()` utilizando a palavra chave `override`.

```
1.          class Circulo : FormaGeometrica
2.          {
3.              private string cor;
4.
5.              public Circulo() { }
6.              ~Circulo() { }
7.
8.              public override void Desenhar()
9.              {
10.                 base.Desenhar();
11.                 Console.WriteLine("Desenhar Circulo!");
12.             }
13.         }
14.
15.         class Triangulo : FormaGeometrica
16.         {
17.             public Triangulo() { }
18.             ~Triangulo() { }
19.
20.             public void arrendondarPontas()
21.             {
22.                 //Código para arredondar pontas de um triângulo.
23.             }
24.
```

```
25.             public override void Desenhar()
26.             {
27.                 base.Desenhar();
28.                 Console.WriteLine("Desenhar Triangulo!");
29.             }
30.         }
```

O trecho de código abaixo exemplifica então o fato de que se pode tratar objetos de classes derivadas como sendo de classes base. No exemplo abaixo é criada uma lista de objetos do tipo `FormaGeometrica` e em seguida é chamado o método `Desenhar()` de cada objeto na lista.

```
1.              static void Main(string[] args)
2.              {
3.                  List<FormaGeometrica> formas = new List<FormaGeometrica>();
4.                  formas.Add(new Triangulo());
5.                  formas.Add(new Circulo());
6.
7.                  foreach(FormaGeometrica forma in formas)
8.                  {
9.                      forma.Desenhar();
10.                 }
11.
12.                 Console.ReadKey();
13.             }
```

Se deve então enfatizar que na linha 9, o código não sabe qual o tipo de forma geométrica está chamando o método `Desenhar()`, tratando então todos objetos derivados como sendo do tipo `FormaGeometrica`.

5 UML

5.1 INTRODUÇÃO

UML vem do inglês e significa *Unified Modeling Language*, que em português significa linguagem de modelagem unificada. Esta linguagem de modelagem foi criada no começo dos anos 90 e foi idealizada por Grady Booch, Ivar Jacobson e James Rumbaugh. UML é então um

conjunto de técnicas de notação gráfica para criar modelos visuais de softwares projetados usando programação orientada a objetos.

O propósito principal desta linguagem de modelagem é o de definir para a comunidade de desenvolvedores um padrão para se descrever softwares. Como UML é um padrão, profissionais de TI podem finalmente ler e descrever estruturas e design de programas, pois quando uma notação universal é utilizada existe pouca margem para ambiguidades e interpretações errôneas.

UML é reconhecido pela organização internacional de padrões (ISO) e esta reconhecida pela ISO/IEC 19501:2005 e foi lançada oficialmente pela *Object Management Group* (OMG) em 1997.

Uma grande vantagem de se conhecer UML é percebida em projetos que envolvem muitas pessoas e que já estão em andamento. Quando um novo membro se junta a uma equipe com um projeto nesta situação, caso o projeto possua uma boa descrição em UML, este novo membro pode entender e se tornar mais produtivo em um tempo muito menor do que se ele fosse realizar uma inspeção de todo o código do projeto. Portanto, conhecer UML é de extrema importância para desenvolvedores e gerentes de projetos, pois deixa o processo de desenvolvimento muito mais ágil. Outra grande vantagem é que um projeto UML não possui nenhuma linha de código nem implementação. Do mesmo jeito que não se constrói um prédio sem uma planta baixa, é muito complicado se desenvolver um software de qualidade sem antes ocorrer um bom processo de planejamento.

UML é então composta por um conjunto de diagramas que podem descrever o comportamento estático ou dinâmico de um software. Diagramas que descrevem o comportamento estático de um programa são representados por estruturas que descrevem objetos, atributos, operações e relacionamentos. Diagramas dinâmicos por sua vez descrevem comportamentos de interações entre objetos, que podem então modificar o estado do programa.

UML foi utilizada nesse projeto de conclusão de curso para descrever o comportamento do software de controle que foi desenvolvido para atuar como um servidor, como também para o *daemon* que ira controlar o arduino e o shield RFID. O objetivo deste capítulo é então descrever as principais estruturas e diagramas desta linguagem de modelagem, assim como prover exemplos para um melhor entendimento.

5.2 DIAGRAMA DE CASO DE USO

Diagramas de caso de uso são utilizados para descrever uma funcionalidade do sistema. É lógica então a elaboração deste diagrama no momento da enumeração de requisitos e funcionalidades do sistema. Diagramas de caso de uso definem então uma entidade chamada de "ator", que podem representar seres humanos, maquinas ou outros elementos que interagem com o sistema. Esta interação tem por fim um resultado na forma de um valor visível e mensurável para um determinado ator. O ator não faz parte do sistema, pois ele apenas executa algum tipo de interação e recebe um resultado.

A outra entidade que este tipo de diagrama define é a do caso de uso. Casos de uso descrevem então uma funcionalidade que vai interagir com o ator. Todos os casos de uso podem ser descritos em um único diagrama ou podem ser quebrados em módulos. Para se representar um caso de uso, desenha-se uma elipse e se escreve o nome do caso de uso no meio ou em cima desta elipse. Atores são representados por bonecos que ficam do lado esquerdo ou direito do diagrama e as interações entre atores e casos de uso são definidas por linhas. A imagem abaixo exemplifica este tipo de diagrama.

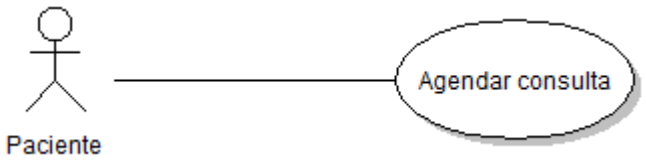

Figura 12. Diagrama simples de caso de uso.

Diagramas de caso de uso podem ter vários atores e casos ao mesmo tempo. A linha sólida no diagrama da figura 12 representa a ação de comunicação, que na notação UML é descrita como `<<communicates>>`. Existem outras relações, por exemplo, quando não se deseja copiar um relacionamento que já foi descrito em outro diagrama, utiliza-se a notação `<<include>>`, e uma linha pontilhada é utilizada. Quando existe uma variação no comportamento normal de uma interação se usa a notação `<<extend>>`, também representada por linhas pontilhadas. A figura 13 exibe outro exemplo de diagrama de caso de uso.

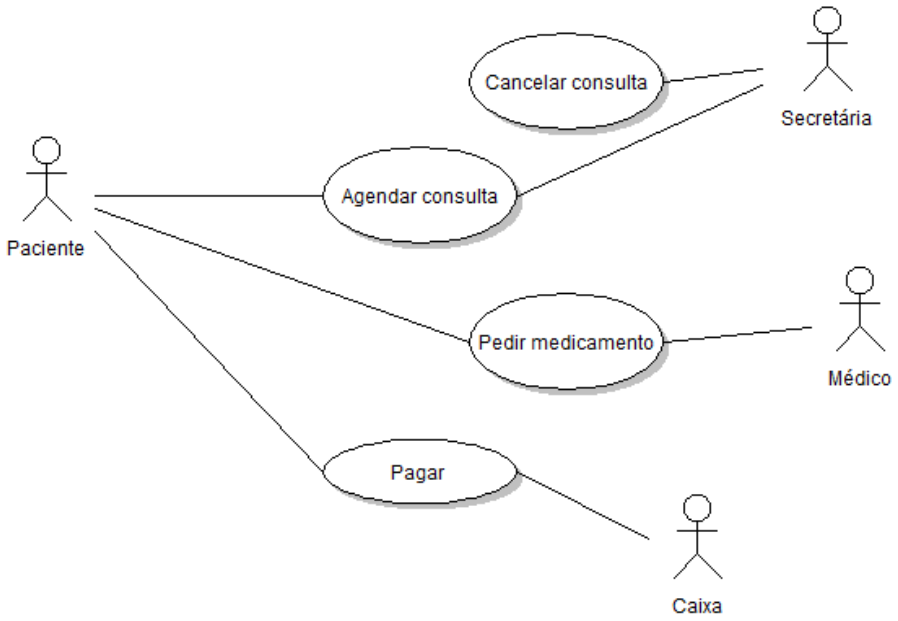

Figura 13. Diagrama de caso uso com vários atores

5.3 DIAGRAMA DE CLASSES

Diagrama de classes é o tipo mais conhecido da notação UML. Este é um tipo de diagrama estático, pois ele descreve classes de objetos e as relações que estas classes possuem entre si. Diagramas de classe demonstram então diferentes entidades que modelam o sistema (pessoas, coisas, dados etc.), ou seja, este tipo de diagrama define uma abstração da futura implementação do sistema.

A notação gráfica para uma classe é um quadrado que pode conter um ou mais linhas horizontais, que dividem o quadrado em compartimentos. O compartimento superior exibe o nome da classe, o segundo compartimento define os atributos da classe e o terceiro compartimento define a lista de métodos que esta classe poderá executar. A figura abaixo exemplifica uma estrutura de uma classe na notação UML.

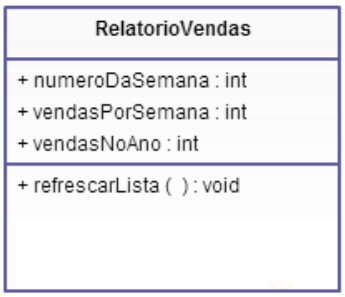

Figura 14. Diagrama de classe.

Diagramas UML de classes são capazes também de descrever as relações entre as classes. A seção seguinte descreve estes relacionamentos.

5.3.1 HERANÇA

Quando uma classe herda de uma classe base, este relacionamento é descrito com uma seta saindo da classe derivada em direção a classe base. A linha que une as classes é solida e a seta é uma seta fechada e sem preenchimento. A figura 15 exemplifica a relação de herança.

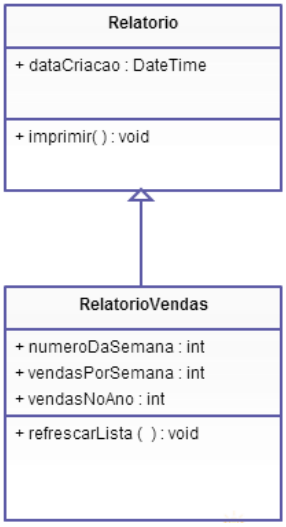

Figura 15. Relação de herança

5.3.2 ASSOCIAÇÃO

Uma relação de associação significa que uma classe está ciente da existência da outra. Esta relação é representada por uma linha sólida entre as classes. Se apenas uma classe sabe da existência da outra, esta condição é representada por uma seta aberta. A figura abaixo exemplifica uma relação de associação.

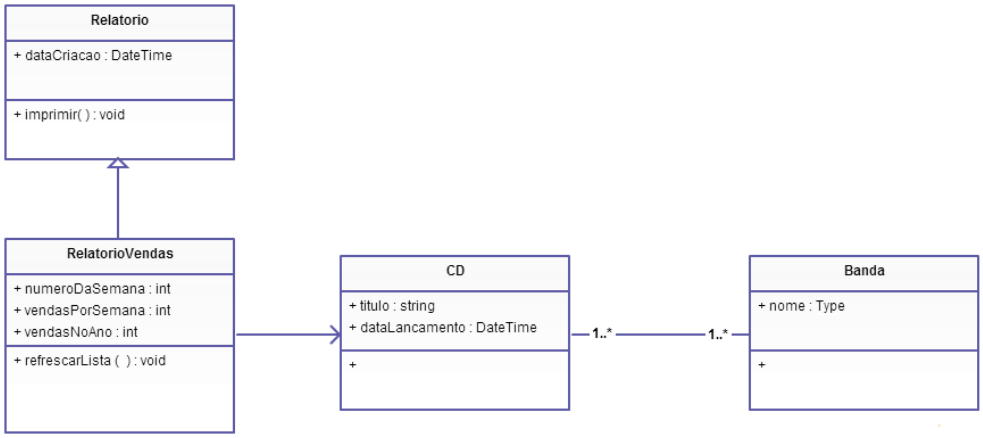

Figura 16. Diagrama de classe com associações

5.3.3 AGREGAÇÃO

Agregação é um tipo especial de associação. Agregação é um tipo mais forte de associação, pois este tipo de relação acontece quando uma classe possui algum tipo de container que armazena outro tipo de classe. É uma relação do tipo “tem um”. Uma classe “tem um” , ou mais instâncias de outro objetos de outra classe. Esta relação é representada na forma de um diamante em branco.

Figura 17. Diagrama de classes com agregação

5.3.4 COMPOSIÇÃO

Composição é um tipo de associação mais forte que agregação. Simplesmente pelo fato que um objeto está contido dentro de outro objeto, e este só pode existir enquanto o objeto que o contém existir. Um exemplo de composição pode ser o de uma figura que só pode estar contido dentro de um desenho. Esta notação é representada por um diamante preenchido.

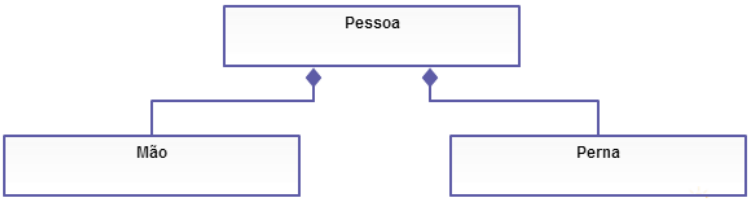

Figura 18. Diagrama de classes com composição

5.4 DIAGRAMA DE SEQUÊNCIA

Diagramas de sequencia descrevem o comportamento e a interação entre objetos. Obviamente este é um diagrama dinâmico. Diagramas de sequência são auto explanatórios e fáceis de ler. No topo do diagrama se desenha caixas com nomes de instâncias de objetos que vão descrever o comportamento de objetos utilizados em um determinado caso de uso. Em seguida desenham-se linhas verticais a partir de cada caixa, que representam o tempo de vida de cada objeto. Se um objeto executa uma chamada para outro objeto, uma seta horizontal ligando uma linha vertical a outra é desenhada com o nome da função que foi chamada. Caso o método do objeto chamado possua algum retorno, uma seta pontilhada voltando para o objeto que realizou a chamada é desenhada com o valor de retorno da função. A seta contendo o valor de retorno é opcional, mas ajuda no melhor entendimento do diagrama.

O diagrama na figura 19 descreve a geração de relatórios de um sistema de vendas de CDs.

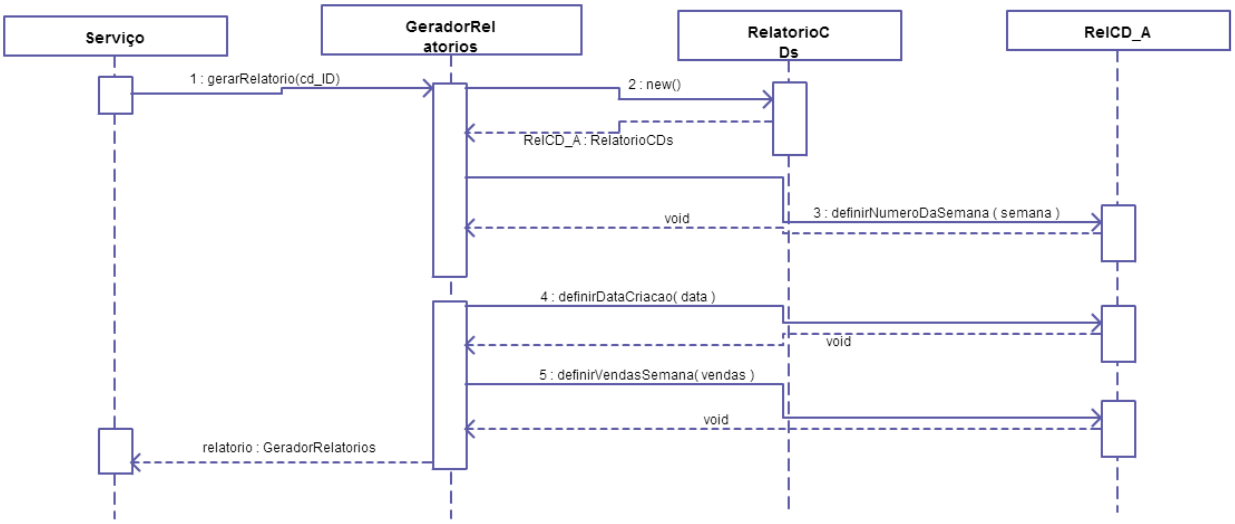

Figura 19. Diagrama de sequência de uma geração de relatórios

5.5 DIAGRAMA DE COMPONENTES

Diagramas de componentes descrevem a maneira física com que os componentes do sistema se relacionam. O objetivo é mostrar as dependências do sistema, e como cada componente vai se comunicar com o resto do sistema. Um componente pode ser definido de várias formas, a forma como é descrito vai então depender da linguagem de programação, como também do nível de detalhe que será descrito no componente. Em C#, pode-se definir cada componente utilizando *namespaces*. Namespaces são uma forma de definir classes, funções e variáveis que ficam restritos a um determinado namespace que possui um nome único, ajudando na organização e evitando erros de definições de classes com nomes repetidos.

Diagramas de componentes também podem conter a interação entre hardware e software. Isto é feito utilizando caixas 3D para designar hardware no diagrama. Componentes por sua vez são representados por caixas 2D. Setas pontilhadas representam interações entre componentes de software, e setas sólidas representam ligações de componentes de hardware.

O diagrama da figura 20 demonstra uma parte da definição de componentes do sistema proposto neste trabalho de conclusão de curso.

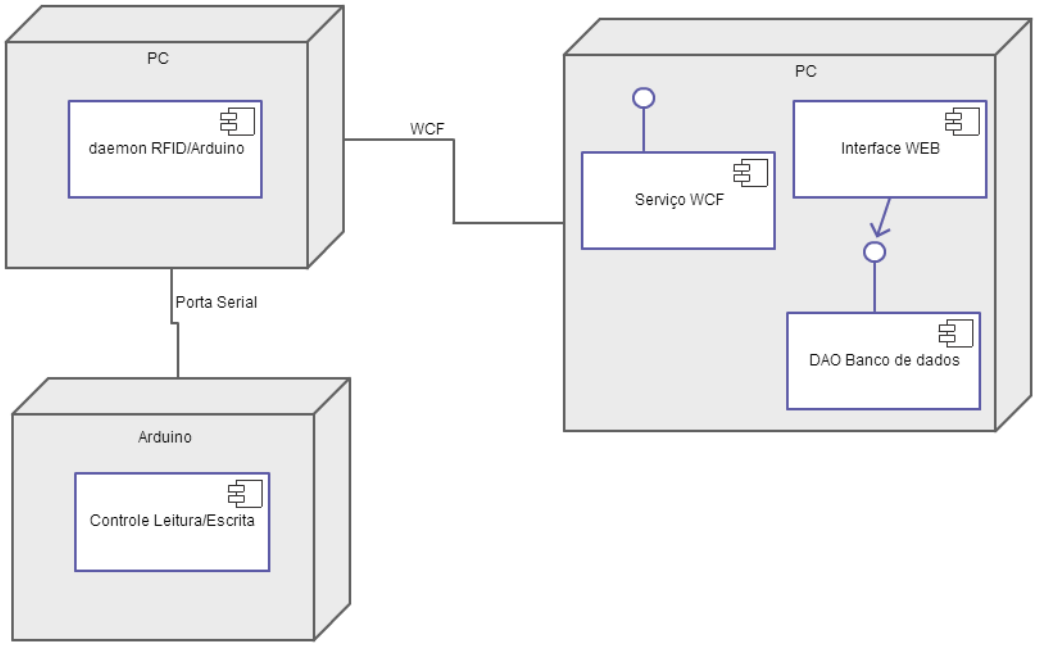

Figura 20. Diagrama de componentes de projeto

6 Banco de dados

6.1 INTRODUÇÃO

Programas de computadores realizam operações sob dados e produzem resultados. Estes dados são em sua grande maioria, necessários após as operações para posteriores análises, distribuição de informação entre pessoas e até mesmo outros sistemas computacionais. Bancos de dados resolvem este problema de maneira que eles armazenam informações coerentes na forma de dados estruturados.

A figura abaixo exemplifica a importância e o uso de bancos de dados relacionais por varias entidades.

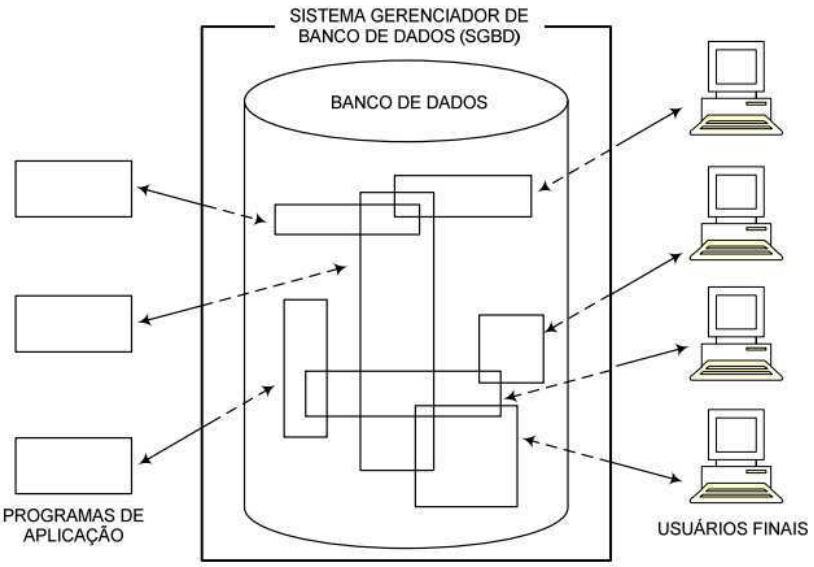

Figura 21. Sistema de banco de dados DATE, 2004, P.6 (Adaptado)

É de extrema importância que a maneira em que estas estruturas de dados sejam concebidas de uma forma organizada e padrão, para assim facilitar a posterior recuperação dos dados. Existem varias maneiras de se organizar dados em forma de bancos de dados, porém a forma mais usada em sistemas hoje em dia são os bancos de dados relacionais.

O objetivo deste capítulo é o de introduzir bancos de dados relacionais e seus principais conceitos, introduzindo também a linguagem *Structured Query Language*, ou SQL.

6.2 BANCO DE DADOS RELACIONAIS

Uma base de dados relacionais é um banco de dados que armazena suas informações na forma de tabelas, linhas, colunas, atributos e domínios. Estas tabelas por sua vez se relacionam de forma que uma pode vir a depender da outra. Esta seção tem objetivo então de descrever como os dados são armazenados, apresentando aspectos estruturais e de integridade, como também a maneira de como os dados armazenados são gerenciados.

Estas relações e funcionalidades de um banco de dados relacionais são gerenciadas pelo *Sistema de Gerenciamento de Banco de Dados* (SGBD) também denominado de *Sistema de Gerenciamento de Banco de Dados Relacionais* (SGBDR).

6.2.1 SGBD

Atualmente existem vários SGBDs disponíveis no mercado, desde SGBDs gratuitos e muitas vezes open source, como MariaDB, como também SGBDs que necessitam de licenças de uso como Oracle e SQLServer. Porém, todos estes possuem as mesmas responsabilidades em uma base de dados. Se pode então enumerar estas funcionalidades básicas.

i. Controlar o armazenamento
ii. Recuperação de dados
iii. Exclusão de dados
iv. Segurança dos dados
v. Garantir a integridade dos dados

Existe uma terminologia em inglês denominada de CRUD (*Create, Read, Update and Delete*) que em português significa criar, ler, atualizar e apagar. Esta terminologia resume bem as principais funções de um SGBD.

A maneira como as operações CRUD são realizadas em um SGBD é por meio da linguagem SQL. Cabe então ao SGBD receber comandos na linguagem SQL e interpreta-los corretamente para que as instruções sejam corretamente realizadas.

Como banco de dados relacionais são formas padronizadas de se armazenar dados, estas informações podem ser acessadas pela maioria das linguagens de programação, dando assim a flexibilidade de escolha de SGBD e de linguagem para o desenvolvedor.

6.2.2 ASPECTO ESTRUTURAL

No modelo relacional, o banco de dados é representado como um conjunto de relações. Considerando que uma relação é, de certo modo, similar a uma tabela de valores e aplicando a terminologia do MR diz-se que as linhas denominam-se tuplas; as colunas, atributos; e a tabela em si, relação (ELMASRI; NAVATHE, 2011, p. 39).

A definição das tuplas a partir do problema geral do sistema que se está implementando é o passo inicial ao se modelar uma base dados relacionais. Um exemplo para uma tupla de um banco de dados que armazena informações pessoais pode ser da forma:

(André Guimarães; andre; 12345; Rua Auta Leite.717,Bela Vista)

Onde a partir disto podemos modelar uma tabela que armazena estas informações que possui a estrutura de tabela proposta na Tabela 1.

Nome	Login	Senha	Endereço

Tabela 1. Modelo de tabela

O usuário do software iria então popular esta tabela utilizando a linguagem SQL, que o SGBD estaria então em cargo de interpretar e executar. A Tabela 2 exemplifica algumas entradas no banco.

Nome	Login	Senha	Endereço
André Guimarães	Andre	12345	Rua Auta Leite, 717
George Acioli	George	12345	Rua Aprigio Veloso

Tabela 2. Exemplo da tabela proposta com registros.

O desenvolvedor após a modelagem das tuplas e suas relações, pode então criar este modelo em seu SGBD de escolha através de alguma ferramenta gráfica, ou até mesmo utilizando a linguagem SQL.

6.2.3 ASPECTO DE INTEGRIDADE

"Uma superchave é um conjunto de um ou mais atributos que, tomados coletivamente, nos permitem identificar de maneira unívoca uma entidade em um conjunto de entidades" (SILBERSCHATZ; KORTH; SUDARSHAN, 1999, p. 32).

O aspecto de integridade esta então diretamente atrelado ao conceito de superchave ou chave primaria. De forma simples, uma chave primaria identifica um registro na tabela como sendo único. No exemplo da Tabela 1 podemos inserir uma chave primaria para indicar a unicidade da entrada no banco. Modificando o exemplo temos então a Tabela 3.

Id	Nome	Login	Senha	Endereço

Tabela 3. Tabela remodelada com chave primária.

Onde a coluna “Id”, é geralmente um numero inteiro que pode ser ou não incrementado de forma automática pelo SGBD a fim de garantir uma identificação única para cada usuário inserido na tabela.

Com base no conceito de chave primaria, é introduzido agora o conceito de chave secundaria. Em um banco de dados é necessário manter certo nível de organização que relacione de maneira clara as tabelas presentes. Como por exemplo, na modelagem de um sistema de acesso, pode existir uma grande variedade de tipos de usuários. Logicamente estes tipos devem ser armazenados em uma tabela separada da de usuários, garantindo assim a integridade e a unicidade destes tipos de usuário. A tabela de usuários por sua vez vai referenciar o tipo de usuário de um determinado registro através da chave primaria da tabela de tipos. Isto está exemplificado nas Tabelas 4 e 5.

Id_tipo	Tipo

Tabela 4. Definição da tabela tipo de usuário.

Id	Nome	Login	Senha	Endereço	Id_tipo

Tabela 5. Tabela de usuário com chave secundaria.

Portanto a coluna “Id_tipo” na tabela de usuários é denominada como chave estrangeira para a tabela de tipos de usuário. O desenvolvedor pode então configurar o SGBD de forma que ele garanta a existência da entrada correspondente à chave estrangeira antes de se inserir uma entrada na tabela de usuários. Garantindo assim a integridade das informações no banco.

6.3 LINGUAGEM SQL

A linguagem SQL define uma forma de se inserir instruções em um SGBD a fim de executar ações que modifiquem o banco de dados, seja adicionando, apagando, atualizando ou pesquisando registros nesta tabela. Esta seção irá expor de forma direta algumas das principais instruções SQL como também exemplificar estes comandos.

É importante deixar claro que as instruções descritas nesta seção são apenas uma parte das instruções disponíveis. Outras instruções como CREATE, DROP e ALTER não serão discutidas, podendo ser facilmente encontradas em livros referenciados como bibliográfica deste texto.

6.3.1 INSTRUÇÃO SELECT

A instrução SELECT tem o papel de selecionar registros na tabela de acordo com um conjunto de parâmetros passados ao SGBD. Esta é provavelmente a instrução SQL mais conhecida entre desenvolvedores que já tiveram algum contato com banco de dados relacionais.

Um comando SELECT pode ter até seis cláusulas e tem a seguinte estrutura:

```
SELECT < lista de atributos e funções >
FROM < lista de tabelas >
WHERE < condição >
GROUP BY < atributos de agrupamento >
HAVING < condição de grupo >
ORDER BY < lista de atributos >
```

A fim de facilitar o entendimento, a instrução SELECT abaixo pesquisa na tabela definida nas seções anteriores por um usuário que possui um determinado tipo de usuário, e ordena estes pela sua chave primária.

```
SELECT Id, Nome, Login, Senha, Endereço
FROM Usuarios
WHERE Id_Tipo=2
ORDER BY Id
```

O SGBD por sua vez interpreta esta instrução e realiza uma operação de pesquisa no banco de dados que satisfaça as condições impostas.

A clausula WHERE pode possuir outros operadores diferentes de igual (=), a Tabela 6 resume as operações disponíveis.

Operação	Descrição
=	Igual
<> ou !=	Diferente
>	Maior que
<	Menor que
>=	Maior ou igual que
<=	Menor ou igual que
BETWEEN	Define um intervalo
LIKE	Define um padrão
IN	Igual a múltiplos valores
IS ou IS NOT	Compara a NULL

Tabela 6. Operadores da cláusula WHERE.

6.3.2 INSTRUÇÃO INSERT

A instrução INSERT é usada para inserir um registro na tabela, caso este registro possua alguma chave estrangeira, é aconselhável que o SGBD seja configurado de forma a garantir a existência desta chave em outra tabela. O exemplo abaixo insere um registro na tabela de usuários definido na seção 6.2.3.

```
INSERT INTO Usuarios
VALUES (“André Guimarães”, “andre”, “12345”, “Rua Auta Leite”, 5)
```

É importante notar que na instrução acima não se passa o valor da coluna “Id”, pois este valor é automaticamente gerado pelo SGBD.

6.3.3 INSTRUÇÃO DELETE

A instrução DELETE tem o papel de apagar registros na tabela. Existem SGBDs que dão suporte a uma operação de remoção recursiva. Onde caso um registro seja apagado, todos os registro que referenciem a entrada apagada, também serão apagados. Garantindo assim a consistência dos dados.

O exemplo abaixo apaga uma entrada na tabela "Usuario" que possua a identificação passada.

```
DELETE FROM Usuarios
WHERE Id=2
```

6.3.4 INSTRUÇÃO UPDATE

Por fim a instrução UPDATE é responsável por indicar ao SGBD uma atualização em algum registro já existente no banco de dados. O exemplo abaixo altera o endereço do usuário que possui o valor 2 como chave primária. Ressaltando que o comando UPDATE pode ser utilizado para vários registros de uma vez, bastando apenas a modificação do filtro imposto pela cláusula WHERE.

```
UPDATE Usuarios
SET Endereço = "Rua Treze de Maio, 23, Centro"
WHERE Id=2
```

7 Sistema de controle de acesso

Este capítulo descreve as especificações como também a implementação realizada do projeto, colocando em prática os conceitos estudados e descritos nas seções anteriores.

7.1 ESPECIFICAÇÕES DE PROJETO

A proposta do projeto tem como objetivo a descrição de um sistema de acesso controlado por cartões RFID.

Este sistema vai contar com uma interface web, onde usuários poderão visualizar seus dados, como também poderão modifica-los caso possuam o nível de acesso requerido para tais modificações. Este sistema estará hospedado em um servidor local. Caso o usuário possua nível suficiente e o computador possua um módulo arduino conectado, este poderá realizar o cadastro de tags RFID, como também a associação de tags e usuários.

Este sistema deve contar com um servidor que possui um microcontrolador arduino com um shield leitor e escritor RFID, capaz de escrever em tags RFID dados que identifiquem o portador, como também os grupos a que ele pertence. O sistema de acesso terá o papel também de armazenar informações básicas de usuários como,

i. Nome completo de usuário
ii. Login
iii. Senha
iv. Matrícula na faculdade
v. Telefone
vi. Tipo de usuário

O sistema será também responsável por armazenar informações pertinentes ao ambiente em que está inserido. Para cada ambiente controlado pelo sistema, este ambiente deverá estar registrado no banco de dados. O banco de dados também será responsável por guardar as informações de todas as tags RFID designadas aos usuários.

7.1.1 SISTEMA DE GRUPOS

Para manter o controle de quais usuários terão acesso a quais ambientes, é proposto um sistema de grupos. Cada usuário cadastrado no sistema possuirá um ou mais grupos associados. Estas informações juntamente com o seu número único de identificação, serão escritos na tag

RFID embutida em seu crachá. Desta forma, cada modulo arduino, ou outro leitor RFID escolhido, deverão possuir armazenados em sua memória interna os grupos de usuários que possuem acesso ao ambiente em que este está controlando o acesso. Estes grupos serão identificados por um número inteiro sem sinal de 32 bits, podendo então existir um numero total de 2,147,483,647 grupos diferentes.

O diagrama abaixo descreve um caso de uso de cadastro e associação de usuário a um grupo, como também a associação de uma tag ao usuário.

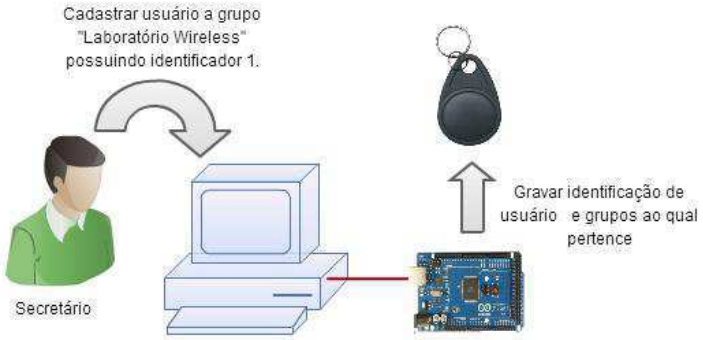

Figura 22. Cadastro de usuários e tags.

Após o cadastro do usuário e das tags, este já estará habilitado a acessar os ambientes associados a cada grupo. O diagrama a seguir descreve este funcionamento.

Figura 23. Acesso ao ambiente com tags RFID.

Cada tag RFID possuirá por sua vez o numero único de identificação do usuário, seguido pelos grupos que este usuário pertence. Cada tag RFID possui 1 KB de memória interna. Esta informação será guardada da maneira descrita a seguir.

Os primeiros 32 bits irão armazenar a identificação única do usuário. Em seguida um separador na forma de " ; " será utilizado para identificar o fim do número de identificação. Os próximos bytes irão identificar cada grupo, também separados por um caractere ponto e vírgula. Abaixo segue um exemplo de informação embutida em uma tag RFID.

203	;	10	;	32	;	14	;	2

Figura 24. Formato das informações armazenadas na tag RFID.

O leitor possuindo este protocolo embutido, realizará a leitura da informação na tag, dividindo cada 32 bits identificando os separadores.

7.1.2 DAEMON ARDUINO

Para manter a modularidade do sistema, dividindo funcionalidades de banco de dados com a escrita e leitura de tags, é proposto um daemon. Um daemon é um software que permanece em execução em plano de fundo em um sistema. Este daemon ficaria responsável pela comunicação com o arduino, realizando as operações de leitura e escrita nas tags. Para realizar a comunicação entre a interface web e o daemon, um serviço WCF deverá ser executado no daemon, bastando apenas a interface web realizar chamadas para este daemon. Os serviços WCF que devem ser implementados estão listados abaixo.

i. iniciarRFID( )
ii. desligarRFID( )
iii. escreverTag(string mensagem)
iv. lerTag( )

Desta forma, caso algum outro leitor RFID seja escolhido para ser utilizado, basta apenas reimplementar o serviço WCF fazendo chamadas que realizem as operações específicas do novo leitor.

7.1.3 FUNCIONALIDADES PREVISTAS

Inicialmente o sistema proposto não possuirá todas as funcionalidades, porém, as seguintes funcionalidades serão implementadas para a apresentação deste trabalho.

i. Cadastro de usuários no sistema
ii. Cadastro de tas RFID
iii. Associação de tags a usuários
iv. Cadastro de grupos
v. Cadastro de tipos de usuários
vi. Cadastro de ambientes
vii. Escrita de tags com um arduino controlado via porta serial
viii. Leitura da tag por parte de um leitor acoplado em um modulo arduino
ix. Acesso ao sistema via uma interface web

7.2 ATORES

Os usuários que irão interagir ao sistema são denominados de “atores”. Estes atores por sua vez possuem uma lista de ações permitidas. A seguir estão descritas estas ações que implicam também restrições dependendo do nível de acesso.

7.2.1 ADMINISTRADOR

O administrador possui total acesso ao sistema, podendo realizar toda e qualquer ação disponível, que vão desde funcionalidades no contexto do sistema, como também poderá ter acesso direto ao banco de dados e modifica-lo.

7.2.2 SECRETÁRIO

O secretário possui um nivel de acesso um pouco abaixo do administrador. O secretário será capaz então de realizar as seguintes ações:

i. Cadastrar/Remover/Modificar secretários
ii. Cadastrar/Remover/Modificar alunos
iii. Cadastrar/Remover/Modificar professores
iv. Realizar pesquisas em todo o banco de dados
v. Cadastrar/Remover/Modificar tags RFID
vi. Associar/Desassociar tags RFID a usuários
vii. Cadastrar/Remover/Modificar grupos de usuários
viii. Cadastrar/Remover/Modificar ambientes de acesso
ix. Adicionar/Remover usuários do tipo "Professor" e "Aluno" a grupos de usuários

7.2.3 PROFESSOR

O professor por sua vez possui um nível de acesso inferior ao do secretário, com a restrição que este não poderá realizar o cadastro de alunos, outros professores nem de secretários. As ações previstas para um usuário do tipo professor estão listadas abaixo.

i. Realizar pesquisas em todo o banco de dados
ii. Modificar cadastro de alunos
iii. Modificar cadastro do próprio usuário
iv. Cadastrar/Remover/Associar grupos de usuários
v. Cadastrar/Remover/Associar ambientes de acesso
vi. Adicionar/Remover usuários do tipo "Aluno" a grupos de usuários.

7.2.4 ALUNO

O ator do tipo "Aluno" possui o nível mais baixo de acesso, tendo este capacidade apenas de acessar informações que lhe façam alguma referência. As ações disponíveis para o aluno estão listadas abaixo.

i. Realizar pesquisas de informações pessoas do próprio usuário
ii. Modificar informações do próprio usuário
iii. Pesquisar grupos em que o usuário está associado

Com base nestas descrições, é possível então realizar o projeto do sistema, descrevendo estas ações e casos de uso por meio de diagramas UML, como também a maneira como estas informações serão armazenadas em um banco de dados.

7.3 ARDUINO

Para se realizar o controle de escrita de informações nas tags RFID, foi escolhido um microcontrolador arduino. O arduino escolhido foi o Arduino Mega 2560. A figura 25 exibe uma imagem deste modelo.

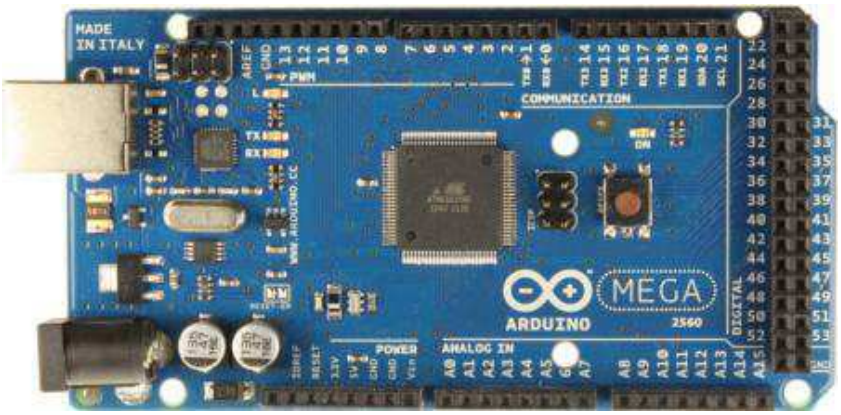

Figura 25. Arduino Mega 2560

Este é um dos modelos mais avançados de arduino, possuindo as especificações técnicas listadas abaixo.

Microcontrolador	ATmega2560
Tensão e operação	5V
Tensão de entrada (recomendada)	7-12V
Tensão de entrada (limites)	6-20V
Pinos digitais de entrada e saída	54 (dos quais 15 funcionam como saída PWM)
Pinos analógicos	16
Corrente DC por pino de entrada e saída	40 mA

Corrente DC para pino 3,3V	50 mA
Memória Flash	256 KB dos quais 8 KB são usados bootloader
SRAM	8 KB
EEPROM	4 KB
Frequência de clock	16 MHz

Tabela 7. Especificações técnicas de um Arduino Mega 2560

7.4 COMPONENTES RFID

Os componentes RFID a serem escolhidos são o modulo de leitura/escrita e as tags RFID. As seções em seguir listam as especificações técnicas de cada componente escolhido.

7.4.1 SHIELD RFID RF522

Como a tecnologia de controle de leitura das tags foi baseada na plataforma Arduino, um shield que suporta a leitura de tags que operam na frequência 13,56 MHz foi escolhido. Este shield está ilustrado na figura 26.

Figura 26. Shield Arduino RC522

As especificações técnicas deste shield encontram-se listadas abaixo.

i. Baseado no chip MFRC522

ii. Consumo em funcionamento: 13-26mA / CC 3.3V

iii. Consumo em inatividade: 10-13mA / CC 3.3V
iv. Consumo em modo sleep: < 80uA
v. Pico de corrente: < 30mA
vi. Frequência de funcionamento: 13,56MHz
vii. Faixa de leitura: 0 a 60 mm (Cartão Mifare1)
viii. Interface: SPI
ix. Taxa de transferência de dados: 10Mbit/s máximo
x. Cartões suportados: Mifare1 S50, S70 Mifare1 MIFARE Ultralight, Mifare Pro, MIFARE DESFire
xi. Dimensões: 40 milímetros × 60 milímetros
xii. Temperatura de operação: -20 - 80 graus Celsius
xiii. Temperatura de armazenamento: -40 - 85 graus Celsius
xiv. Umidade: 5% a 95% de umidade relativa

7.4.2 TAGS RFID

Para o projeto foram escolhidas tags que operam utilizando o sistema Near-Field, e que operam em uma frequência de 13,56 MHz, definidas no padrão ISO/IEC 14443 A/MIFARE. A figura 27 ilustra um tipo desta tag, dando ênfase às dimensões da tag.

Figura 27. Tags RFID de 13,56 MHz em dois encapsulamentos diferentes.

7.5 DIAGRAMA ELÉTRICO

O diagrama abaixo descreve as conexões que devem ser realizadas entre o Arduino Mega 2560 e o shield RC522.

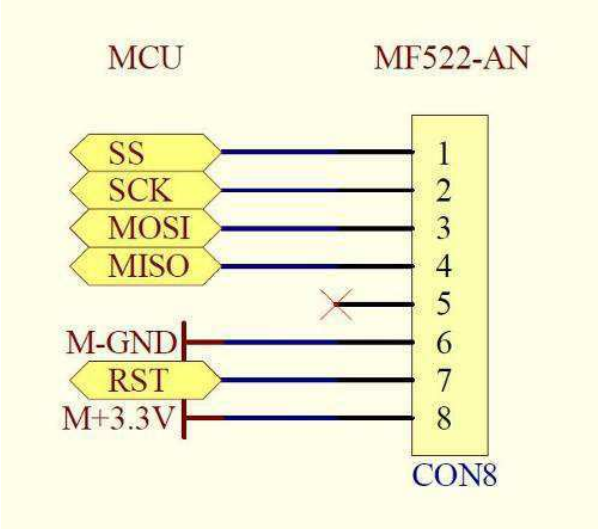

Figura 28. Conexões entre arduino e shield

O arduino por sua vez é conectado ao computador por um cabo USB, simulando uma porta serial.

7.6 DIAGRAMAS UML

Os diagramas UML a seguir descrevem o sistema que funcionará no servidor. Estes diagramas então podem ser usados para se realizar uma implementação do sistema na linguagem de escolha, que no caso deste trabalho de conclusão de curso foi a linguagem C# da Microsoft.

7.6.1 DIAGRAMAS DE CASOS DE USO

Os diagramas de caso de uso descrevem vários cenários que um ou mais ator pode atuar. Todos estes seguem as descrições da seção 5.2. Supõe-se que o ator “Administrador” está inserido em todos os casos, visto que este possui total acesso ao sistema.

O diagrama da figura 29 descreve as ações de cadastro no sistema.

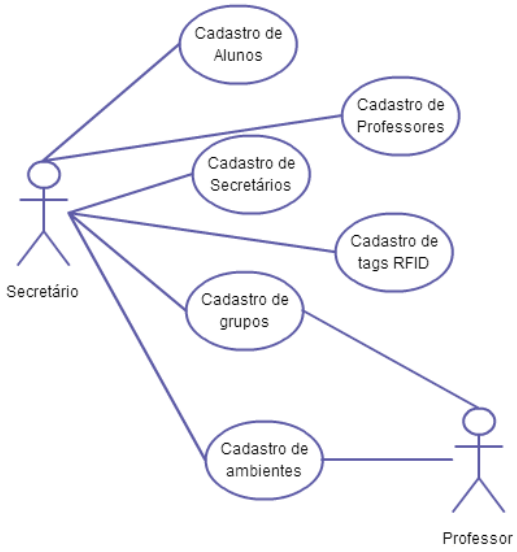

Figura 29. Caso de uso de cadastro

O diagrama da figura 30 descreve o caso de associações.

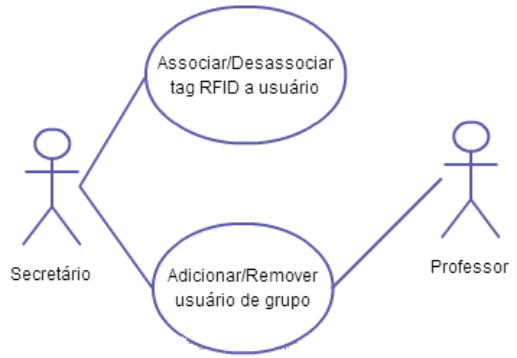

Figura 30. Caso de uso de associações

O diagrama da figura 31 descreve os casos de uso de pesquisas no banco de dados.

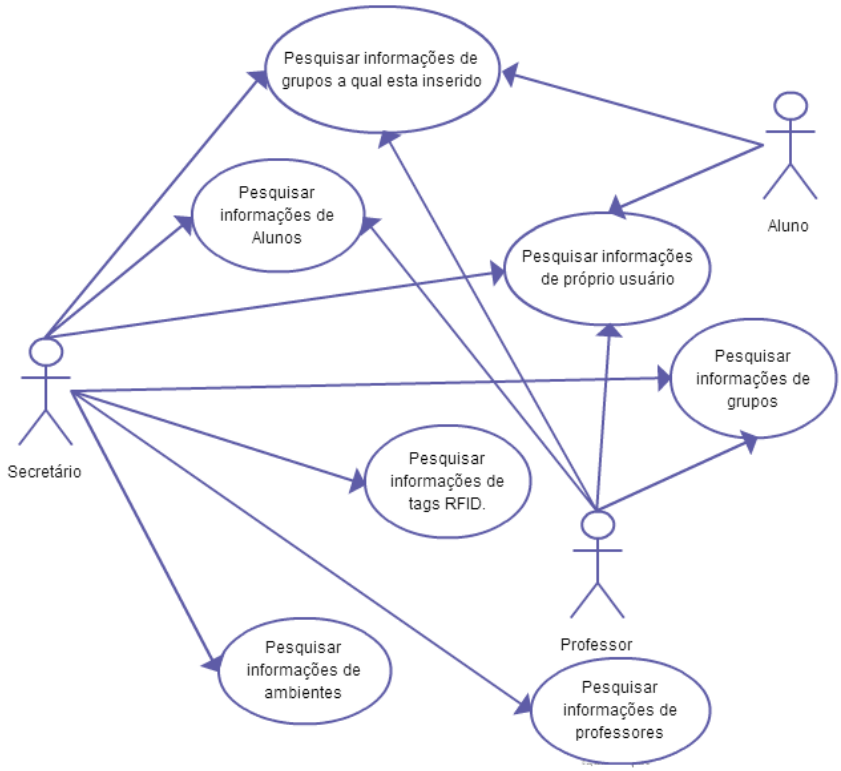

Figura 31. Casos de uso de pesquisas no banco de dados

7.6.2 DIAGRAMAS DE CLASSE

Os diagramas de classe descrevem os modelos de classes que foram implementados, descrevendo suas relações, ações e atributos.

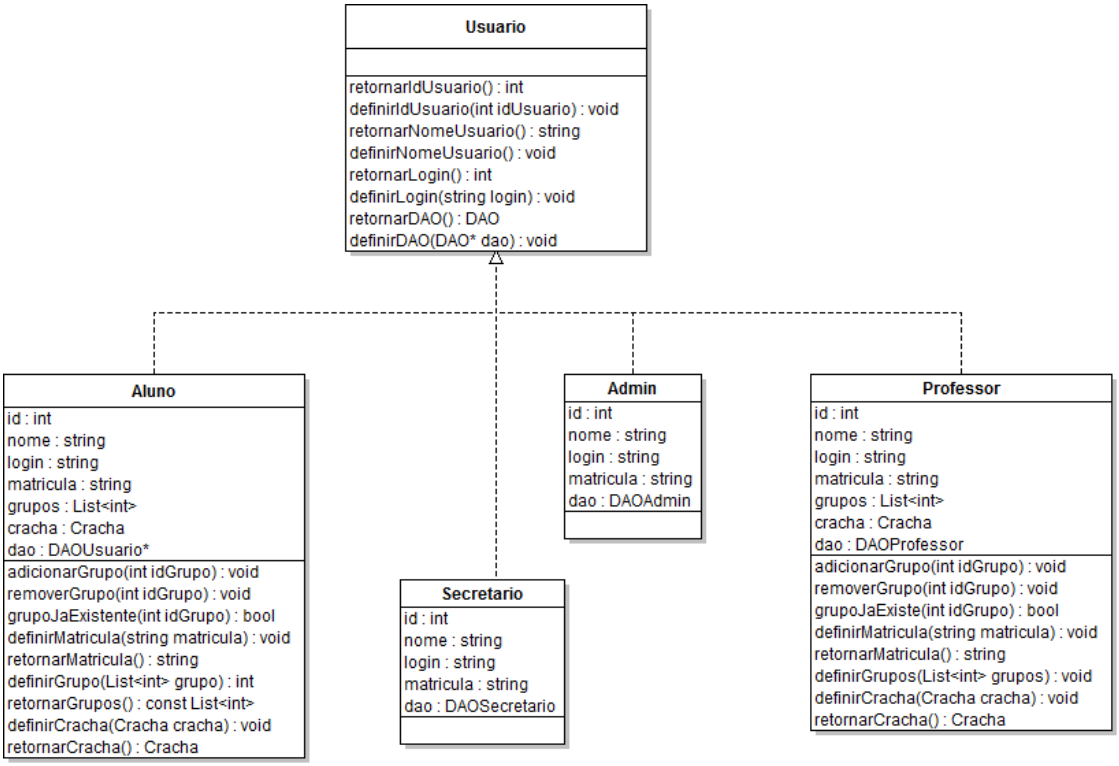

Figura 32. Diagrama de classe para usuários

Foi utilizado o conceito de DAO (*Data Access Objects*) onde se tem classes que servem de intermédio entre o sistema e o acesso ao banco de dados. Facilitando assim a modificação do banco de dados acarretando em um efeito mínimo na refatoração de código. Abaixo se encontra o diagrama UML destes objetos DAO.

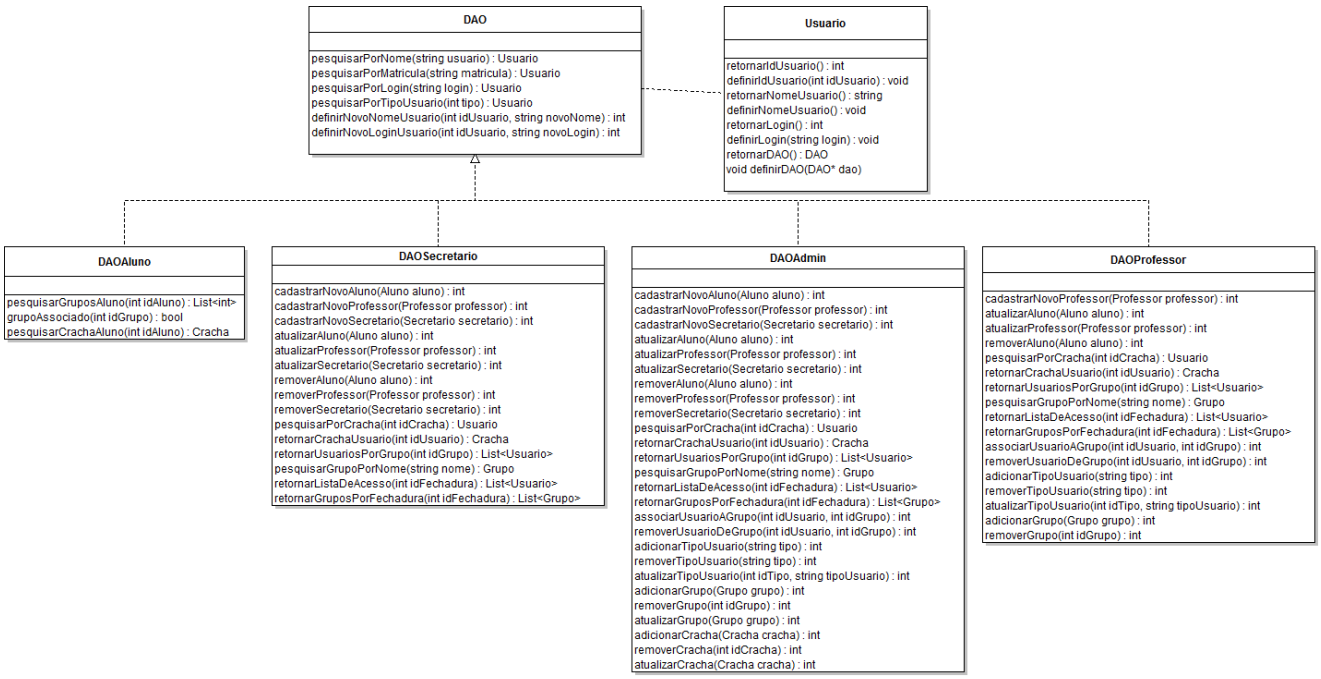

Figura 33. Diagrama de classe DAO

7.6.3 DIAGRAMAS DE SEQUÊNCIA

Os diagramas de sequência da figura 34 demonstram as interações entre objetos para se realizar ações no sistema.

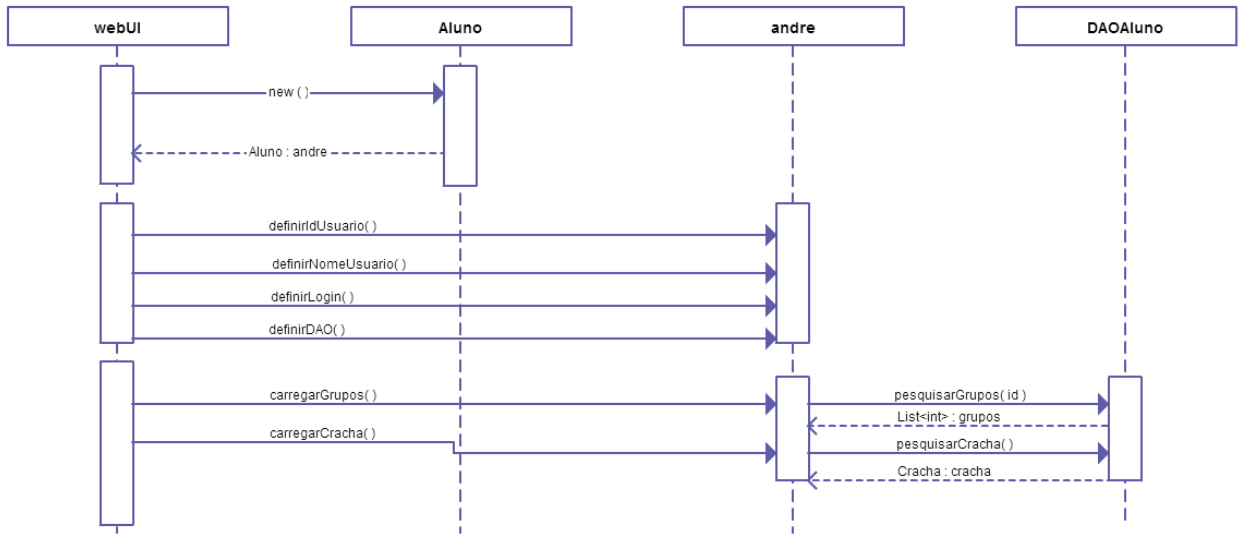

Figura 34. Diagrama de Sequência: Criação de usuário

O digrama da figura 34 descreve a criação de um usuário do tipo “Aluno”, esta mesma sequência de ações deve ser repetida para a criação de objetos do tipo “Professor” e “Secretário”.

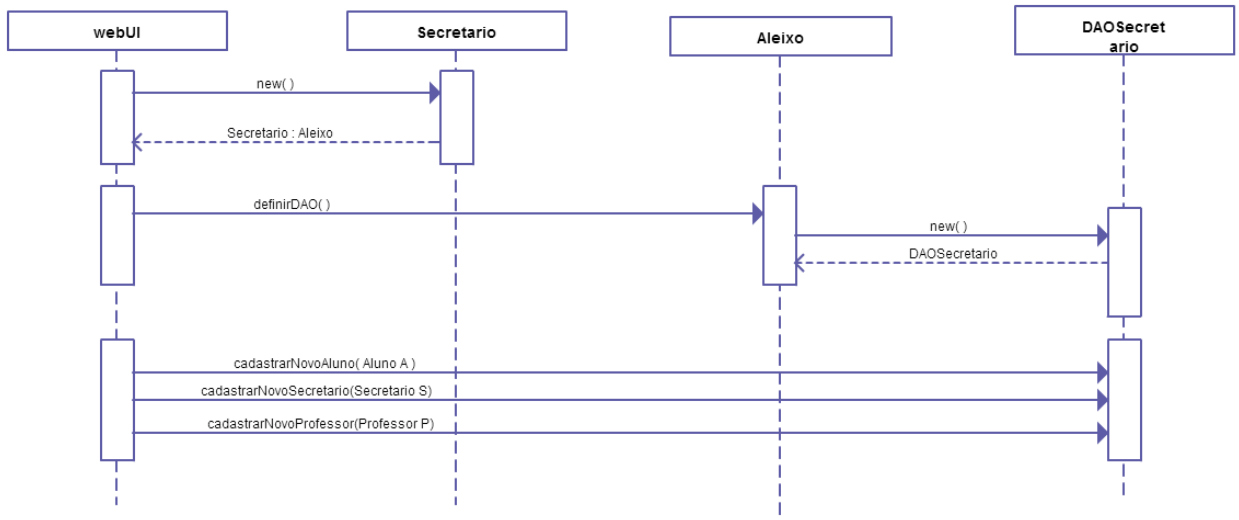

Figura 35. Diagrama de Sequência para o cadastro de usuários

O diagrama da figura 35 descreve o cadastro no banco de dados de usuários do tipo “Aluno”, “Professor” e “Secretário”. É importante destacar que toda a interação com o banco é feita através do DAO de cada usuário. No caso da figura 35, o usuário é um “Secretário”, e este possui um DAO com ações bem definidas, que por sua vez descrevem as restrições impostas na descrição do projeto. Esta mesma sequência de ações devem ser utilizadas para todas as interações com o banco de dados, como a pesquisa, remoção e atualização de informações.

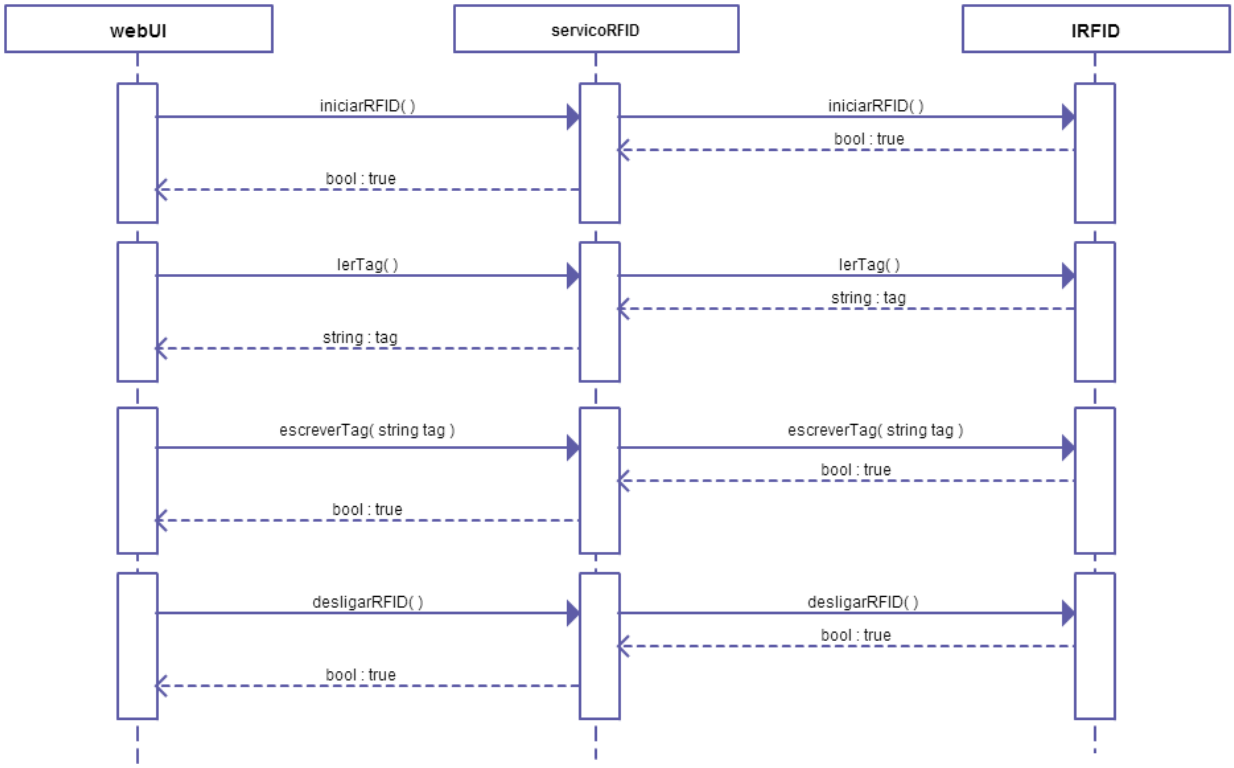

Figura 36. Diagrama de sequência: Operações RFID

O ultimo diagrama de sequência descreve as operações de escrita e leitura utilizando o serviço WCF rodando no daemon. Analisando a sequência de ações, é importante explicitar que o serviço WCF funciona apenas como uma interface comum entre o sistema e as funcionalidades relacionadas ao modulo RFID.

7.6.4 DIAGRAMAS DE COMPONENTES

O diagrama de componentes abaixo descreve a maneira com que os módulos do projeto, bem como os componentes de hardware e software se relacionam.

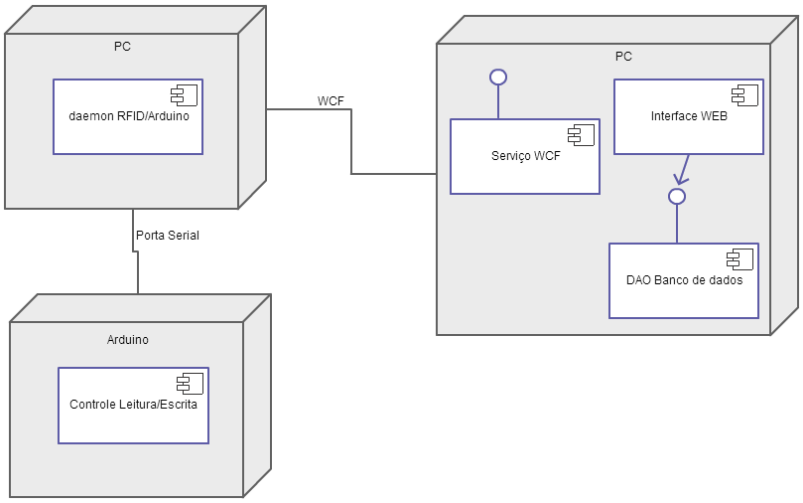

Figura 37. Diagrama de componentes do sistema

7.7 BANCO DE DADOS

O projeto do banco de dados foi realizado com o intuito de refletir as especificações, e de modelar da melhor maneira possível as informações descritas.

7.7.1 TUPLAS

As tuplas a seguir descrevem o modelo de cada tabela

Tabela grupos: (id_grupo, grupo)

Tabela ambientes: (id_ambiente, ambiente)

Tabela usuario_grupo (id_usuario_grupo, id_usuario, id_grupo)

Tabela acesso (id_acesso, id_ambiente, id_grupo)

Tabela usuario (id_usuario, nome, login, senha, matricula, telefone, id_tipo_usuario, id_cracha)

Tabela tipo_usuairo (id_tipo_usuario, tipo)

Tabela crachás (id_cracha, tag_rfid)

7.7.2 DIAGRAMA DO BANCO DE DADOS

O diagrama abaixo foi projetado tomando como base as tuplas da seção anterior, levando também em consideração as relações entre as tabelas e as relações de chave primária e chave estrangeira.

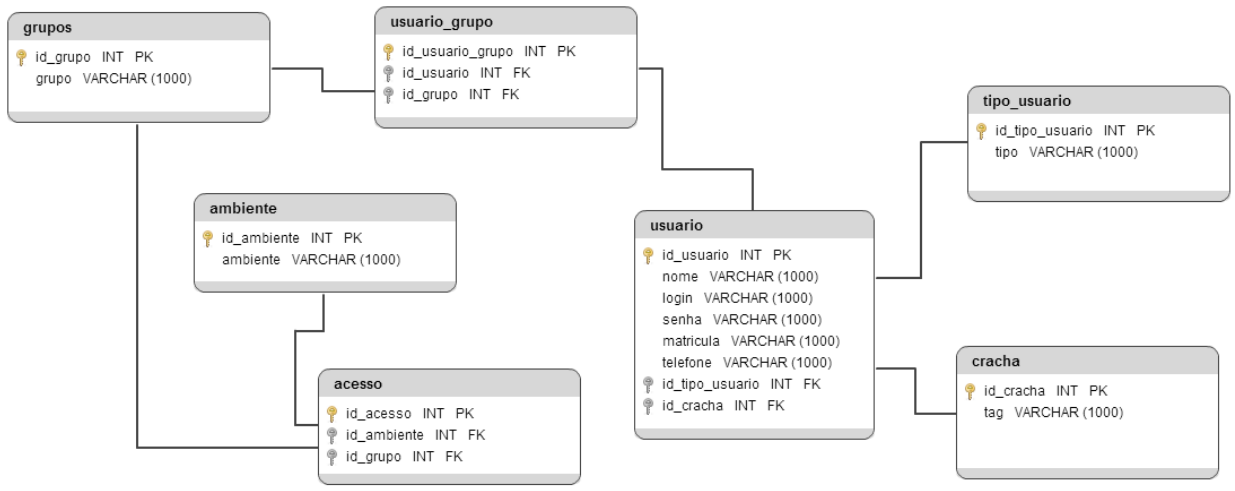

Figura 38. Diagrama do banco de dados

8 Conclusão

Sistemas de acesso controlados por cartões RFID são uma realidade hoje em dia. Porém, as soluções existentes no mercado muitas vezes possuem preços bastante elevados. Conclui-se então que é possível realizar a construção de tal sistema de forma simples e barata, podendo até mesmo expandir o funcionamento do sistema com mais funcionalidades como controle de pessoas que trabalham em uma empresa.

Após a conclusão do estudo sobre as tecnologias do projeto, conclui-se que o engenheiro com ênfase em automação necessita possuir uma grande base de conhecimento, indo muitas vezes longe de sua formação, chegando a conceitos de alto nivel de programação.

A plataforma arduino se mostrou uma opção interessante para a prototipagem de projetos, com muitos recursos esta tecnologia foi de fácil configuração e manuseio. E se comportou bem em testes com o shield RFID.

Sistemas que possuem como paradigma de programação a orientação a objetos são maioria hoje no mercado. O sistema projetado fez grande uso deste conceito, o que possibilitou uma grande flexibilidade no sistema, principalmente no que concerne a escolha de leitores e cartões RFID. Com o uso de herança e polimorfismo, caso algum terceiro deseje utilizar outro tipo de leitor, basta apenas reimplementar as interfaces que controlam o leitor, e o sistema ira funcionar da mesma maneira reutilizando desta forma grande parte do código.

Um bom projeto de banco de dados evita muitos problemas que venham a surgir em futuras mudanças em projetos de médio e de grande porte. Uma modificação em um banco de dados acarreta uma grande mudança no código, que devido ao efeito cascata, muitas vezes pode ser mais interessante reimplementar toda a pilha que controla o banco de dados. Portanto nota-se claramente a importância de se conhecer técnicas de projeto de banco de dados e principalmente de pesquisas SQL e suas varias formas de otimização.

Outro aspecto importante analisado durante a execução do projeto foi a importância da modelagem prévia do software através da linguagem unificada e modelagem UML. Este método se mostrou como uma ferramenta importantíssima, pois ela permite uma visão geral do projeto, como também uma visão mais detalhada de cada a ser realizada. Constatou-se também que uma

boa modelagem evita refatorações de códigos futuros, pois na fase de especificação de projeto através de UML, não é necessário mudar uma linha de código caso se queira realizar alguma modificação. Outro aspecto foi o do melhor entendimento para outros desenvolvedores de todo o código através destes diagramas, facilitando muito a integração de novos membros em projetos computacionais.

Este projeto foi realizado com o intuito de ser implantado no laboratório de instrumentação eletrônica e controle (LIEC) localizado na universidade federal de Campina Grande. Os próximos passos então são a finalização das funcionalidades, pois a proposta inicial foi de uma implementação parcial com funcionalidades básicas como prova de conceito, e em seguida a implantação deste sistema nas portarias e nas dependências do laboratório.

Se encontra planejado como trabalhos futuros a implementação total do sistema e uma possível instalação do mesmo nas dependências do laboratório de instrumentação eletrônica e controle.

BIBLIOGRAFIA

ARDUINO. Arduino, 2013. Disponivel em: <http://www.arduino.cc/>. Acesso em: 1 Setembro 2013.

BELL, D. IBM Corporation. UML basics: An introduction to the Unified Modeling Language, 2003. Disponível em: <http://www.ibm.com/developerworks/rational/library/769.html>. Acesso em: 05 Setembro 2013.

DARWEN, H. An Introduction to Relational Database Theory. 1ª Edição. ed. [S.l.]: Ventus Publishing ApS, 2010.

DEITEL, P.; DEITEL, H. Visual C# 2012 How to Program. 5ª Edição. ed. [S.l.]: Prentice Hall, 2013.

PRESSMAN, R. Software Engineering: A Practitioner's Approach. 7ª Edição. ed. [S.l.]: McGraw-Hill Science/Engineering/Math, 2009.

ROBERTI, M. RFID Journal. RFID Journal, 2005. Disponível em: <http://www.rfidjournal.com/articles/view?1338>. Acesso em: 28 Agosto 2013.

WANT, R. An Introduction to RFID Technology. IEEE CS, Janeiro 2006. 25-33.

NAVATHE, E. Sistemas de Banco de Dados. 6ª. ed. [S.l.]: [s.n.], 2011.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. Sistema de Banco de Dados. 1ª. ed. [S.l.]: ELSEVIER BRASIL, 2012.