ANDRÉ GUSTAVO RIGON E FERNANDO KISHIDA KOREEDA

ORIENTADOR: PROF. DR. JOSÉ KLÉBER DA CUNHA PINTO

USO DE BLUETOOTH PARA ACESSO A PRONTUÁRIOS EM HOSPITAIS

SIGLA: BLUE

Escola Politécnica da Universidade São Paulo

14 de Dezembro de 2009, São Paulo, SP, Brasil

ANDRÉ GUSTAVO RIGON E FERNANDO KISHIDA KOREEDA

USO DE BLUETOOTH PARA ACESSO A PRONTUÁRIOS EM HOSPITAIS

SIGLA: BLUE

Relatório Final
Projeto de Formatura II

Áreas de Concentração: Sistemas Eletrônicos, Programação e Comunicações

Professor Orientador:
Prof. Dr. José Kléber da Cunha Pinto

Departamento de Sistemas Eletrônicos
Escola Politécnica da Universidade São Paulo
14 de Dezembro de 2009, São Paulo, SP, Brasil

RESUMO

Em hospitais, as duas formas mais comuns de armazenamento de prontuários são o uso de papel e o uso de computadores compartilhados por todos os usuários. Esses dois casos causam uma grande perda de eficiência dos médicos e enfermeiros e, conseqüentemente, do estabelecimento.

O objetivo do projeto apresentado neste texto é a criação de um sistema de armazenamento de prontuários de pacientes em que cada usuário é capaz de acessar os dados através de seu computador pessoal, utilizando comunicação Bluetooth. Os prontuários são armazenados em uma placa Arduino e apresentados ao usuário através de uma interface gráfica. São também levadas em conta questões de segurança, como criptografia de dados e autenticação de usuários.

O projeto apresentado tem como principal produto uma grande quantidade de componentes de software, como a interface gráfica, a criptografia e compressão de dados e a comunicação por Bluetooth, desenvolvidos principalmente em C++ e utilizando algumas bibliotecas e toolkits open source. Além disso, são levados em conta problemas envolvendo protocolos de comunicação e características de hardware dos componentes utilizados.

Palavras-Chave: Prontuário Digital. Bluetooth. Arduino BT. Notebook. Laptop. Computador. C++. Qt.

OK

ABSTRACT

In hospitals, the two most common ways of storing medical records are the use of paper forms and the use of computers shared by many users. These options cause a great loss of efficiency of doctors and nurses and, consequently, of the establishment.

The objective of the project presented in this document is the creation of a medical record storage system in which every user is capable of accessing the data using his personal computer, through Bluetooth communication. The records are stored in an Arduino board and presented to the user through a graphical interface. Security concerns are considered, such as the use of data cryptography and user authentication.

The product generated by the project has great amount of software components, implementing the user interface, cryptography, data compression and Bluetooth communication, written primarily in C++ and using open source toolkits and libraries. Communication protocols and hardware specifications are also considered.

Keywords: Digital medical record. Bluetooth. Arduino BT. Notebook. Laptop. Computer. C++. Java. Qt.

Conteúdo

EQUIPE DE PROJETO E FORMAS DE CONTATO

André Gustavo Rigon

Aluno do quinto ano de Engenharia Elétrica com ênfase em Sistemas Eletrônicos na Escola Politécnica da Universidade de São Paulo.

Formatura: 2009

E-mail: andregrigon@gmail.com

Telefone: +55 (11) 9635-9570

Fernando Kishida Koreeda

Aluno do quinto ano de Engenharia Elétrica com ênfase em Sistemas Eletrônicos na Escola Politécnica da Universidade de São Paulo.

Formatura: 1º semestre de 2010

E-mail: fernando.koreeda@gmail.com

Telefone: +55 (11) 8915-3461

Informações sobre o projeto, os seus criados ou licenças de uso podem ser obtidas através dos contatos acima.

ORIENTADOR

Professor Doutor José Kléber da Cunha Pinto

Professor Associado no Departamento de Engenharia de Sistemas Eletrônicos. Engenheiro Eletricista.

E-mail: jkcunhap@lme.usp.br

Telefone: +55 (11) 3091-5255

Escritório: Escola Politécnica da USP, Prédio de Engenharia Elétrica, sala C2 - 69

1 O PROBLEMA

1.1 INTRODUÇÃO

O objetivo deste texto é apresentar o projeto realizado por André Gustavo Rigon e Fernando Kishida Koreeda, orientados pelo professor José Kléber da Cunha Pinto, para a disciplina de Projeto de Formatura do Departamento de Sistemas Eletrônicos da Escola Politécnica da Universidade de São Paulo.

Serão apresentados o problema a ser resolvido, as idéias para a solução, o sistema projetado, a sua implementação, os resultados obtidos e possíveis desenvolvimentos futuros. Além disso, será mostrada a infraestrutura utilizada para o projeto.

1.2 O PROBLEMA

Atualmente, hospitais utilizam dois métodos principais para o armazenamento de informações de pacientes. O primeiro é a utilização de prontuários de papel. Nesse tipo de sistema, cada página do prontuário é em geral representada por uma folha. Todos os médicos e possivelmente enfermeiros tem acesso ao prontuário.

O segundo tipo de sistema, introduzido com o avanço da tecnologia, é a instalação de computadores em diferentes locais do hospital, compartilhados por todos os médicos e enfermeiros. Nesse sistema cada usuário tem, em geral, uma conta que permite o seu acesso a dados de prontuários.

Conforme será explicado a seguir, esses dois sistemas muito difundidos apresentam diversos problemas.

1.2.1 Problemas do uso de papel

A primeira desvantagem aparente do uso de papel para prontuários é o desperdício de materiais. Esse é um problema que se torna cada vez mais importante com a crescente preocupação com o meio ambiente. Além disso, o armazenamento dos prontuários resultantes ocupa um espaço muito grande nos estabelecimentos dos hospitais.

Em segundo lugar, o uso de prontuários de papel reduz fortemente a eficiência dos estabelecimentos e dos seus trabalhadores. A localização de informações antigas é tornada muito lenta. Os trabalho para arquivar os prontuários toma muito tempo.

1.2.2 Computadores compartilhados

Com o avanço da tecnologia, a introdução de computadores resolveu de forma muito interessante os problemas encontrados com o uso de papel. Foram introduzidos, porém, novos problemas.

Como os computadores são, em geral, compartilhados entre diversos usuários, ocorrem filas para o seu uso. Essas filas diminuem consideravelmente a eficiência de médicos e enfermeiros, por desperdiçarem o seu tempo. Além disso, a espera necessária aumenta a chance de erros na hora da digitação das informações do prontuário.

Existe ainda o risco da ocorrência de problemas nos computadores disponíveis. Como eles são compartilhados por diversos funcionários, qualquer problema poderia ter grandes conseqüências.

2 A IDÉIA

2.1 OBJETIVO

O objetivo do projeto proposto é criar uma forma de prontuário digital que pode ser acessada diretamente do próprio quarto do paciente. Assim, médicos e enfermeiros podem ser capazes de modificá-la localmente, sem a necessidade de deslocamento somente para esse fim.

Além disso, o acesso deve ser o mais prático e barato possível e o sistema não deve ocupar um grande espaço no quarto do paciente. A instalação de um computador em cada quarto, por exemplo, apesar de permitir o acesso ao prontuário, ocuparia um espaço considerável, traria problemas de manutenção e limpeza (que é realizada a cada troca de paciente), além de aumentar consideravelmente os custos de manutenção dos equipamentos e de aquisição dos mesmos. A solução procurada deve evitar esses problemas.

Para tornar o acesso prático aos funcionários, a solução utilizada deve usar comunicação sem fios. Essa comunicação deve ser feita, porém, utilizando tecnologias já existentes e de uso comum, facilitando a utilização por usuários leigos com pouco conhecimento e até com aversão a novas tecnologias.

A consideração acima faz parte de um dos principais focos do projeto. De forma a realizar o menor número de modificações necessárias, deseja-se que os usuários do sistema sejam capazes de utilizar os seus dispositivos pessoais, como notebooks, para acessarem os prontuários. Assim, o aprendizado para utilização do novo sistema e a sua aceitação são facilitados consideravelmente.

Finalmente, o acesso ao prontuário deve ser seguro, de forma a manter sigilosas as informações dos pacientes. Somente funcionários autorizados devem ser autorizados a ler e modificar informações contidas nos prontuários.

2.2 A SOLUÇÃO

Uma solução para os problemas expostos abrangendo os objetivos propostos acima é a utilização de dois módulos. O primeiro, o módulo de usuário, é onde as informações do prontuário são mostradas aos funcionários e onde estes podem fazer todas as modificações necessárias. O segundo módulo, de dados, deve ser o responsável por

armazenar o prontuário no quarto do paciente e transmiti-lo ou recebê-lo quando necessário. Os dois módulos são responsáveis pela segurança dos dados.

Diversos dispositivos podem funcionar como módulo de usuários. Isso facilita muito a aceitação pelos usuários finais, pois não exige muita atenção em treinamento para sua utilização. Podem ser utilizados laptops, palmtops ou até celulares. Esses sistemas são portáteis, leves e realizam todas as funções necessárias.

O segundo módulo, o de dados, deve ser pequeno, de forma a ser praticamente invisível ao paciente e fácil de manter. Ele deve ser capaz de realizar processamento suficiente para enviar a receber os dados no formato correto, seguir o protocolo de comunicação e implementar os mecanismos de segurança.

2.3 RESUMO DA PROPOSTA APRESENTADA

A proposta aqui apresentada atende a todos os objetivos citados para a resolução dos problemas referidos.

Basicamente, a proposta é a utilização de um sistema, que permita ao profissional que deseje alterar o prontuário do paciente, facilidade para a realização de tal tarefa. Para isso, basta conectar um dispositivo habilitado a comunicar-se sem fios (módulo de usuário) ao dispositivo que armazena o arquivo com o prontuário eletrônico do paciente (módulo de dados).

Depois de realizada essa conexão, altera-se o prontuário existente ou cria-se um novo prontuário para, em seguida, armazená-lo novamente no módulo de dados.

As tecnologias e a metodologia utilizadas para tal fim serão discutidas mais adiante.

3 O PROJETO

3.1 O SISTEMA

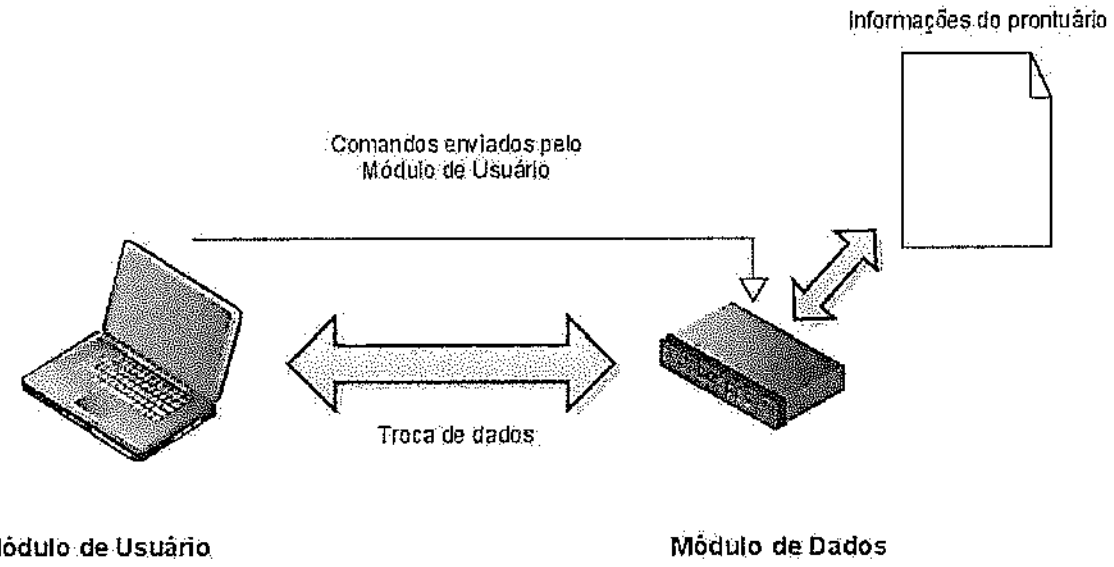


Figura 1 - Interação entre Módulo de Dados e de Usuário

O sistema projetado para a resolução dos problemas apresentados é composto por dois módulos: o Módulo de Usuário e o Módulo de Dados. O primeiro é formado pelo computador do usuário e uma série de componentes de software. O seu papel é apresentar uma interface visual que permita a visualização e manipulação do prontuário pelo usuário.

Ele é também capaz de enviar e receber as informações do prontuário através de comunicação wireless. Por isso, é também incluído no sistema o adaptador de comunicação wireless, presente no computador do usuário ou instalado separadamente.

Já o Módulo de Dados tem a função de armazenar os dados do prontuário. De acordo com comandos enviados pelo Módulo de Usuário, ele é ser capaz de enviar ou receber as informações.

Como a comunicação é realizada sem fios, o módulo de dados contém um transceptor. Em outras palavras, o Módulo de Dados atua como um servidor wireless do prontuário. Ele é pequeno, leve e consome pouca energia, de forma que pode ser colocado em um quarto de hospital.

3.2 BLUETOOTH

Para a implementação da comunicação wireless do sistema foi escolhida a tecnologia Bluetooth. A escolha levou em conta a presença da tecnologia no mercado, o alcance de comunicação, o preço dos dispositivos e a documentação existente atualmente, entre outros fatores.

Em comparação às outras tecnologias analisadas, como o Zigbee, o Bluetooth teve vantagem em todos esses fatores. Atualmente, a grande maioria dos computadores e dispositivos portáteis têm dispositivos Bluetooth. Além disso, existe uma disponibilidade considerável de transceptores dessa tecnologia.

O alcance do Bluetooth é o suficiente para o sistema mesmo em seu menor valor, de cerca de 6 metros. O seu preço, em comparação com outras tecnologias como Zigbee, foi também o menor (devido principalmente à sua maturidade no mercado).

Com a grande difusão da tecnologia, existe também uma grande quantidade de documentação sobre o seu uso, além de bibliotecas e programas. Esse é um requisito fundamental para facilitar o cumprimento do prazo do projeto.

3.3 MÓDULO DE USUÁRIO

3.3.1 Objetivo

Os principais objetivos do Módulo de Usuário são a apresentação de uma interface gráfica com o conteúdo do prontuário para o usuário e a recepção e envio dos dados correspondentes para o Módulo de Dados. Para que isso seja possível, esse Módulo é formado por diversos componentes de hardware e de software.

3.3.2 Hardware

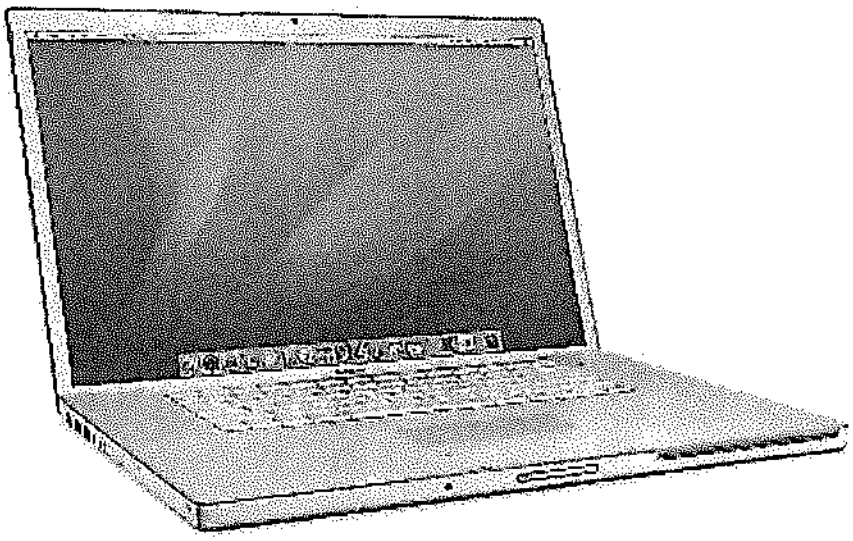


Figura 2 - Módulo de Usuário

O Módulo de Usuário é formado pelo computador do usuário e por um dispositivo Bluetooth, que pode ser integrado no computador ou apenas um adaptador externo. A vantagem da utilização do computador do usuário é a redução do tempo de aprendizado de uso do sistema e o aumento da familiaridade do usuário com o ambiente.

3.3.3 Software

O software do Módulo de Usuário é formado por 5 componentes: Autenticação, Interface, Criptografia, Compressão e Comunicação Serial.

Autenticação

O componente de Autenticação é responsável por garantir que apenas usuários cadastrados no sistema possam visualizar o prontuário do paciente. Para isso, são utilizados pares de nomes de usuário e senhas. Para manter a segurança do sistema, são usadas técnicas de criptografia na verificação dos usuários.

Interface

A Interface é o componente principal do Módulo de Usuário. Ela é responsável por gerar e mostrar a interface gráfica a ser vista e manipulada pelo usuário. Além disso, ela controla os componentes de Criptografia, Compressão e Comunicação Serial para realizar a manipulação, envio e recepção dos dados do prontuário. Conforme será mostrado adiante, esse componente inclui diversas funcionalidades para garantir a segurança e consistência dos dados do sistema.

Criptografia

O componente de Criptografia é um dos principais responsáveis por garantir a segurança dos dados do prontuário. Ele permite que os dados sejam criptografados antes do seu envio ao Módulo de Dados e também que sejam decifrados quando recebidos. Dessa forma, mesmo que os dados sejam interceptados durante a transmissão ou que o Módulo de Dados seja capturado, os dados do prontuário permanecerão em segurança.

Compressão

O componente de Compressão permite que os dados do prontuário sejam comprimidos para reduzir o uso de memória do sistema. No caso da existência de múltiplos arquivos, ele também tem a função de reuni-los.

Comunicação Serial

O componente de Comunicação Serial permite o envio e recepção dos dados, vindos do Módulo de Dados. Conforme será explicado adiante, ele implementa um protocolo definido no projeto e realiza as conversões de dados necessárias entre o armazenamento no sistema de arquivos do Módulo de Usuário e a comunicação com o Módulo de Dados.

3.4 MÓDULO DE DADOS

3.4.1 Objetivo

O Módulo de Dados tem a função de armazenar, receber e enviar o prontuário do paciente. Ele funciona como um servidor wireless simplificado.

3.4.2 Hardware

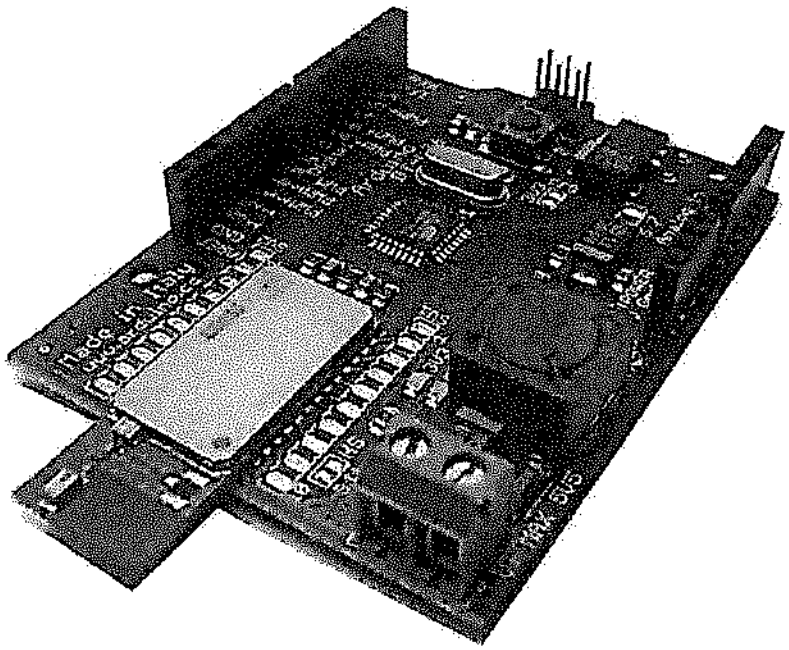


Figura 3 - Módulo de Dados (placa Arduino)

Esse módulo é constituído por uma placa Arduino. Essa placa, importada da Itália, contém um microcontrolador Atmega 168 e um transceptor Bluetooth Bluegiga WT11, além de outros componentes eletrônicos como resistores e capacitores.

A decisão pela compra do Arduino, ao invés da montagem manual de uma placa semelhante, foi devida a diversos motivos. Em primeiro lugar, a montagem manual, além de propícia a erros, gera resultados muito inferiores ao processo industrial utilizado na montagem do Arduino.

Além disso, a economia de tempo obtida evitando-se a montagem foi considerável. Por último, a existência de muita documentação sobre o uso do Arduino, além de muitos exemplos, tornou a placa muito interessante para o projeto.

3.4.3 Software

O software executado pelo Módulo de Dados é constituído por apenas um programa. Esse programa apresenta duas possíveis operações: envio e recepção de dados. Conforme será mostrado na seção de desenvolvimento, ele implementa um protocolo criado no projeto para a comunicação.

3.5 VIABILIDADE DO PROJETO

Após estudo detalhado que envolveu boa parte do primeiro semestre do ano de 2009, concluiu-se ser viável a realização do projeto proposto. Os diversos aspectos abordados pelo estudo estão relacionados a seguir.

3.5.1 Viabilidade para o Grupo

Para que o projeto seja considerado viável do ponto de vista do grupo, deve-se considerar possível a sua conclusão até o término do ano letivo de 2009.

Concluiu-se sua viabilidade levando em consideração o tempo de execução do projeto, a complexidade técnica da proposta e a sua coerência com as disciplinas oferecidas durante o curso de Engenharia Elétrica com ênfase em Sistemas Eletrônicos, os custos envolvidos e a demora na entrega dos dispositivos necessários para a confecção do protótipo.

Tempo de Execução e Complexidade Técnica do Projeto

Para que fosse possível concluir o protótipo final a tempo, foram estudadas todas as tecnologias possíveis de serem utilizadas para a obtenção dos objetivos finais. As opções feitas: tecnologia Bluetooth para comunicação, computador pessoal como módulo de usuário e placa Arduino BT como módulo de dados, foram estudadas de modo a diminuir o tempo total até a conclusão do projeto.

Assim, a escolha pelo Bluetooth foi feita por ser uma tecnologia largamente empregada em todo o mundo. Portanto, não foi difícil encontrar soluções para problemas relacionados a essa tecnologia e havia uma larga oferta de hardwares compatíveis a serem utilizados.

A escolha pelo computador pessoal como módulo de usuário também facilitou muito, pois já havia um em mãos para utilização.

Por fim, a escolha pela placa Arduino BT evitou um desperdício de tempo na confecção de uma placa que teria a mesma finalidade.

O projeto em si apresenta complexidades relacionadas a diversas disciplinas de engenharia elétrica vistas ao longo do curso. Portanto, não havia nada de extraordinariamente complexo que chegasse ao ponto de inviabilizar o estudo necessário para a sua conclusão.

Em relação à fase de desenvolvimento, o cronograma de realização das tarefas foi planejado de modo a alocar mais tempo às tarefas que apresentassem maior grau de complexidade como, por exemplo, a programação de toda a parte de comunicação entre os módulos e a efetuação de testes de desempenho.

Concluiu-se, portanto, que após toda essa análise o tempo disponível seria mais que suficiente para o término do protótipo.

Custo de Realização

É essencial para a realização de um projeto de formatura de um curso de Engenharia Elétrica, que os custos totais envolvidos sejam estudados de maneira cuidadosa. Como deve haver hardware envolvido no projeto e o custo de componentes específicos ainda seja muito alto no país, a falta de planejamento nos custos poderia levar facilmente à inviabilidade do projeto, pois os recursos públicos não foram corretamente distribuídos para esse fim. Portanto, todos os custos deste projeto deveriam ser arcados pelos próprios integrantes, que não têm grande disponibilidade.

A tabela abaixo mostra os principais custos envolvidos até a finalização do protótipo:

Tabela 1 - Custos envolvidos na Confecção do Protótipo

Componente	Preço Unitário	Unidades Necessárias	Preço Total
Equipamento Portátil (Módulo de Usuário)	-	1	-
Arduino Bluetooth (Módulo de Dados)	R$ 615,00	1	R$ 615,00
Pilhas AA	R$ 2,00	2	R$ 4,00
Adaptador Bluetooth USB	R$ 15,00	1	R$ 15,00
Suporte para Pilhas AA	R$ 2,00	1	R$ 2,00
Botão On/Off	R$ 5,50	1	R$ 5,50
		Total	R$ 640,50

Conclui-se, portanto, a partir do valor total necessário, que o projeto é viável.

Tempo de Entrega dos Componentes

Mesmo com recursos financeiros disponíveis e toda a parte de software concluída, seria impossível finalizar o projeto sem os componentes necessários. Portanto, o tempo de entrega deles é um fator essencial para a viabilidade do projeto.

O único componente que poderia apresentar alguma demora na entrega era a placa Arduino BT, que faria a função de módulo de dados. Este poderia atrasar em, no máximo um mês, de acordo com a empresa contatada para a compra. Portanto, esse aspecto não seria o suficiente para inviabilizar o projeto.

Conclusão sobre a Viabilidade para o Grupo

Analisando cada aspecto possível de inviabilizar o projeto após o término do mesmo, pode-se concluir:

O tempo disponível para a finalização do projeto foi suficiente. Como o previsto, muita informação sobre as tecnologias utilizadas puderam ser encontradas, o que auxiliou na conclusão de cada tarefa do cronograma. Este estava muito bem elaborado e foi seguido com certa coerência, mostrando a utilidade do estudo feito na sua elaboração para alocar o tempo necessário de modo eficiente.

O custo de realização foi o mesmo previsto, excluindo custos de transporte e outros custos variáveis que não foram contabilizados. Felizmente, não houve problemas graves com nenhum componente adquirido. Por outro lado, a esperança de ressarcimento pela Universidade de São Paulo não foi atendida, ficando todos os custos a cargo dos integrantes. Apesar de altos, os custos puderam ser bancados.

Finalmente, o tempo de entrega dos componentes não foi empecilho algum. A placa Arduino BT foi entregue em menos de uma semana após o contato com a empresa e os outros componentes foram adquiridos pessoalmente.

Portanto, conclui-se que, como o previsto, o projeto foi viável e concluído em tempo hábil para a disciplina.

3.5.2 Viabilidade para o Cliente

O foco principal de um projeto de engenharia é a sua utilização pelo usuário final. Portanto, o mais importante para tornar um projeto viável ou não são os motivos que levariam terceiros a utilizar a proposta.

Conclui-se a viabilidade para o cliente considerando a atratividade, os preços envolvidos de implementação, operação e manutenção e restrições técnicas do projeto.

Atratividade

Primeiro, deve-se definir quem seriam os possíveis clientes da solução proposta. Neste caso, seriam os hospitais.

Foi analisado se a proposta resolve os problemas encontrados pelos clientes no caso analisado, que seria com os prontuários médicos e todos os empecilhos envolvidos discutidos anteriormente. Constatou-se que médicos, enfermeiros e os demais profissionais atuantes nestes estabelecimentos economizariam tempo, diminuiriam a ocorrência de erros e tornariam os ambientes hospitalares mais eficientes e seguros com a implementação desta proposta.

Observando os diferenciais desta solução, podem ser citados diversos atrativos. Entre eles o tamanho reduzido do módulo de dados, tornando-o menos visível aos pacientes, a possibilidade do uso de diversos dispositivos como módulo de usuário para o acesso e a comunicação sem fio, que facilita a movimentação física dos equipamentos, a limpeza e a aparência de modernidade e alta tecnologia que proporciona.

Preço de Aquisição

Anteriormente, foi informado que o custo do protótipo ficou na faixa de R$ 640,00. Utilizando esse valor como base, constata-se que pelos benefícios que proporciona, não é inviável para um hospital efetuar esse gasto por leito.

Mesmo assim, o valor final para o cliente pode ser drasticamente reduzido. Para a confecção do protótipo, optou-se pela utilização da placa Arduino BT, que inclui todo o hardware necessário, inclusive toda a parte de gravação. Para o cliente, não é necessária a parte de gravação, visto que o microcontrolador da placa já deve ser entregue funcionando. Além disso, a confecção em larga escala de placas pode reduzir ainda mais o valor final.

Após consulta a um representante de uma empresa especializada em importação de módulos Bluetooth e microcontroladores, obteve-se um orçamento aproximado de R$200,00 para cada módulo de dados, reduzindo o valor final para um terço do valor do protótipo.

Quanto ao módulo de usuário, basta um dispositivo com comunicação Bluetooth para efetuar a tarefa. Além disso, não é necessário um módulo de usuário por leito.

Custo de Operação e Manutenção

Como mencionado anteriormente, muitos dispositivos comuns como laptops e palmtops podem ser usados como módulo de usuário. Portanto a manutenção pode ser feita por qualquer empresa do ramo.

Quanto ao módulo de dados, não apresenta custo de operação considerável. No máximo a energia gasta para mantê-lo funcionando. Quanto ao custo de manutenção, por ser um dispositivo de baixo valor, caso haja um problema, poderia ser inteiramente substituído pelo cliente, sem maiores problemas.

Restrições Técnicas da Solução Proposta

Analisou-se a existência de restrições para a utilização da proposta no ambiente hospitalar. O foco do projeto é a sua utilização em quartos de pacientes estáveis, fora das UTIs.

Podem ser feitas duas abordagens: uma a respeito dos potenciais efeitos prejudiciais à saúde causados pela exposição à rádio freqüência e outra a respeito da possível incompatibilidade eletromagnética do módulo de dados com os equipamentos médicos presentes em um quarto de hospital.

Quanto a possíveis efeitos prejudiciais que podem ser causados pelo Bluetooth à saúde dos pacientes, conclui-se que, embora não haja estudos específicos da organização mundial de saúde (WHO) a respeito do Bluetooth, existem muitos estudos da própria WHO sobre efeitos de campos eletromagnéticos até 300 GHz.

O Bluetooth opera em freqüências próximas a 2.4 GHz. Mesma faixa de freqüências dos fornos de microondas, por exemplo. Os efeitos destes são bem conhecidos pelo público em geral, que os utiliza para esquentar em minutos alimentos para o consumo. No entanto, a potência dos fornos é da ordem de 1 kW, enquanto a potência do dispositivo Bluetooth é entre 1mW e 100 mW. Portanto, a comparação com os fornos de microondas não é possível.

Segundo o anexo à resolução nº 303 de 2 de Julho de 2002 da ANATEL (Agência Nacional de telecomunicações), que regulamenta a limitação da exposição humana a campos elétricos, magnéticos e eletromagnéticos na faixa de radiofreqüências entre 9 kHz e 300 GHz, pode-se concluir que a utilização da proposta apresentada está dentro das normas da ANATEL e não apresenta riscos à saúde das pessoas envolvidas com sua utilização. As tabelas com os valores podem ser verificadas abaixo:

Tabela 2 - Limites de exposição da população em geral na faixa entre 9 kHz e 300 GHz. Densidade de potência de onda plana equivalente

Faixa de Radiofreqüências	Intensidade de Campo. E (V / m)	Intensidade de Campo. H (A / m)	Densidade de potência da onda plana equivalente, S_{eq} (W / m^2)
9 kHz a150 kHz	87	5	—
0,15 MHz a 1 MHz	87	$0,73/f$	—
1 MHz a 10 MHz	$87/f^{1/2}$	$0,73/f$	—
10 MHz a 400 MHz	28	0.073	2
400 MHz a 2000 MHz	$1.375f^{1/2}$	$0.0037f^{1/2}$	$f/200$
2 GHz a 300 GHz	61	0,16	10

Tabela 3 - Limites de exposição da população em geral na faixa entre 9 kHz e 300 GHz. SAR (Specific Absorption Rate)

Características de exposição	Faixa de Radiofreqüências	Densidade de corrente para cabeça e tronco (mA / m^2) (RMS)	SAR média do corpo inteiro (W / kg)	SAR localizada (cabeça e tronco) (W / kg)	SAR localizada (membros) (W / kg)
Exposição Ocupacional	9 kHz a 100 kHz	$f/100$	—	—	—
	100 kHz a 10 MHz	$f/100$	0.4	10	20
	10 MHz a 10 GHz	—	0.4	10	20
Exposição da população em geral	9 kHz a 100 kHz	$f/500$	—	—	—
	100 kHz a 10 MHz	$f/500$	0,08	2	4
	10 MHz a 10 GHz	—	0,08	2	4
f é o valor da freqüência, em Hz.					

Quanto à incompatibilidade eletromagnética, citando Mats e Samsom (2004), que publicaram um artigo sobre a substituição dos cabos por Bluetooth nos equipamentos hospitalares nas áreas de cirurgia e de terapia intensiva, que são áreas que apresentam maiores riscos e equipamentos mais sensíveis do que os encontrados nos quartos dos pacientes, que são o escopo do projeto. A conclusão de seu estudo foi que entre os 44 equipamentos comumente encontrados nestas áreas hospitalares mais críticas, nenhum afetou a comunicação sem fios. Ainda, o Bluetooth não causou

nenhuma interferência nem mudanças de operação dos equipamentos médicos testados.

Além disso, não há qualquer restrição nos hospitais a respeito do uso de celulares, que apresentam potência de sinal maior que o Bluetooth, nos quartos dos pacientes. Obviamente, existem setores onde são proibidos quaisquer tipos de metais ou dispositivos que possam causar alterações nos campos magnéticos, como é o caso dos setores de ressonância magnética.

Portanto, a partir do estudo apresentado, conclui-se que a utilização da proposta nos quartos dos pacientes não apresenta qualquer restrição.

Conclusão sobre a Viabilidade para o Cliente

Após o término do projeto, pode-se comprovar a viabilidade para o cliente como o estudo feito anteriormente.

A respeito da atratividade, os testes finais feitos comprovaram a facilidade de utilização e todos os benefícios de economia de tempo e de diminuição de erros previstos. Além disso, comprovou-se ser possível a utilização do sistema com qualquer dispositivo devidamente configurado e com acesso à comunicação Bluetooth.

Não foi possível avaliar o preço mínimo real de cada módulo de usuário, pois não foi feita a confecção em massa do dispositivo de baixo custo. Porém, crê-se que a estimativa feita se aproxima muito do valor, pois o orçamento foi feito por profissional especializado do ramo. Porém, constatou-se a viabilidade mesmo se o preço para o cliente fosse o mesmo do protótipo.

Finalmente, a respeito das restrições técnicas, os estudos apresentados são suficientes para demonstrar que o uso da tecnologia é inofensivo à saúde dos pacientes, pois respeita as normas da ANATEL e da OMS. Quanto à compatibilidade eletromagnética, o estudo citado mostra não haver problemas com os equipamentos mais sensíveis do ambiente hospitalar.

Portanto, reitera-se a viabilidade do projeto como um todo, respeitando todos os aspectos relevantes a essa conclusão.

3.6 COMPROMISSOS COM O CLIENTE

Ao final do primeiro semestre de 2009, alguns compromissos foram firmados e documentados como objetivos principais do desenvolvimento do projeto para o cliente.

Essas obrigações serão relembradas a fim de elucidar as tarefas a serem realizadas para a concretização da proposta.

Mais adiante neste mesmo documento, com o projeto já finalizado, será feita análise mais detalhada de verificação do cumprimento desses compromissos.

As obrigações firmadas com o cliente estão abaixo relacionadas.

3.6.1 Interface com o Usuário

Deve ser desenvolvida uma interface gráfica no módulo de usuário, possibilitando a visualização e a alteração do prontuário do paciente.

3.6.2 Autenticação

Um sistema de autenticação deve ser implementado possibilitando que somente profissionais autorizados possam ter acesso aos dados presentes nos módulos de dados.

3.6.3 Criptografia

Um algoritmo de criptografia deve ser desenvolvido para o módulo de usuário, possibilitando o sigilo das informações do arquivo com os dados do paciente. Um algoritmo de decifragem também deve ser implementado no mesmo módulo, possibilitando a leitura do arquivo criptografado.

3.6.4 Hardware

O protótipo final deve estar funcionando, com todas as funções descritas devidamente implementadas em hardware e com um sistema de alimentação independente.

3.6.5 Comunicação sem fios

O protótipo deve ser capaz de se comunicar com o módulo de usuário via Bluetooth com todas as configurações do microcontrolador prontas para este fim.

3.7 CRONOGRAMA

O Cronograma planejado antes do início da fase de desenvolvimento pode ser encontrado logo abaixo:

Tabela 4 - Cronograma Planejado

	Julho	Agosto	Setembro	Outubro	Novembro
Interface com o Usuário					
Compra dos Componentes					
Criptografia					
Autenticação					
Montagem do Hardware					
Teste de Comunicação Sem Fio					
Implementação do Algoritmo					
Main do Módulo de Dados					
Main do Módulo de Usuário					
Montagem e Teste do Protótipo					
Relatório Final, Apresentação e Pôster					

Este cronograma foi planejado com base nas datas finais de entrega do projeto, nos possíveis problemas de entrega de componentes e nas diferenças de complexidade entre etapas do desenvolvimento.

Em outra seção mais adiante será feita a comparação desse cronograma planejado com o realizado. Diversos fatores influenciaram na mudança de algumas datas e todos serão discutidos.

4 O DESENVOLVIMENTO

4.1 INTERFACE

Essa seção apresenta o principal componente do sistema desenvolvido, a Interface. Serão apresentados o seu objetivo, as suas funções, as ferramentas utilizadas na sua criação, o seu funcionamento e a sua estrutura.

4.1.1 Objetivo

O objetivo da Interface é apresentar uma representação gráfica de um prontuário na tela do computador do usuário e gerenciar os outros componentes do sistema para que os dados do prontuário sejam recebidos e enviados por Bluetooth com segurança.

4.1.2 Funções

A funcionalidade da Interface pode ser dividida em três partes: a apresentação de uma interface gráfica, o gerenciamento dos componentes do sistema e a realização de IO (input e output) dos dados necessários (como o conteúdo dos prontuários). Essas funções são apresentadas de forma simplificada a seguir. A sua implementação é explicada nos itens posteriores.

Interface gráfica

Toda interação entre o usuário e o sistema é feita através da interface gráfica criada pela Interface. Ela permite que o usuário observe e modifique dados dos prontuários com facilidade e também que este execute funções do sistema como envio e recepção dos dados por Bluetooth.

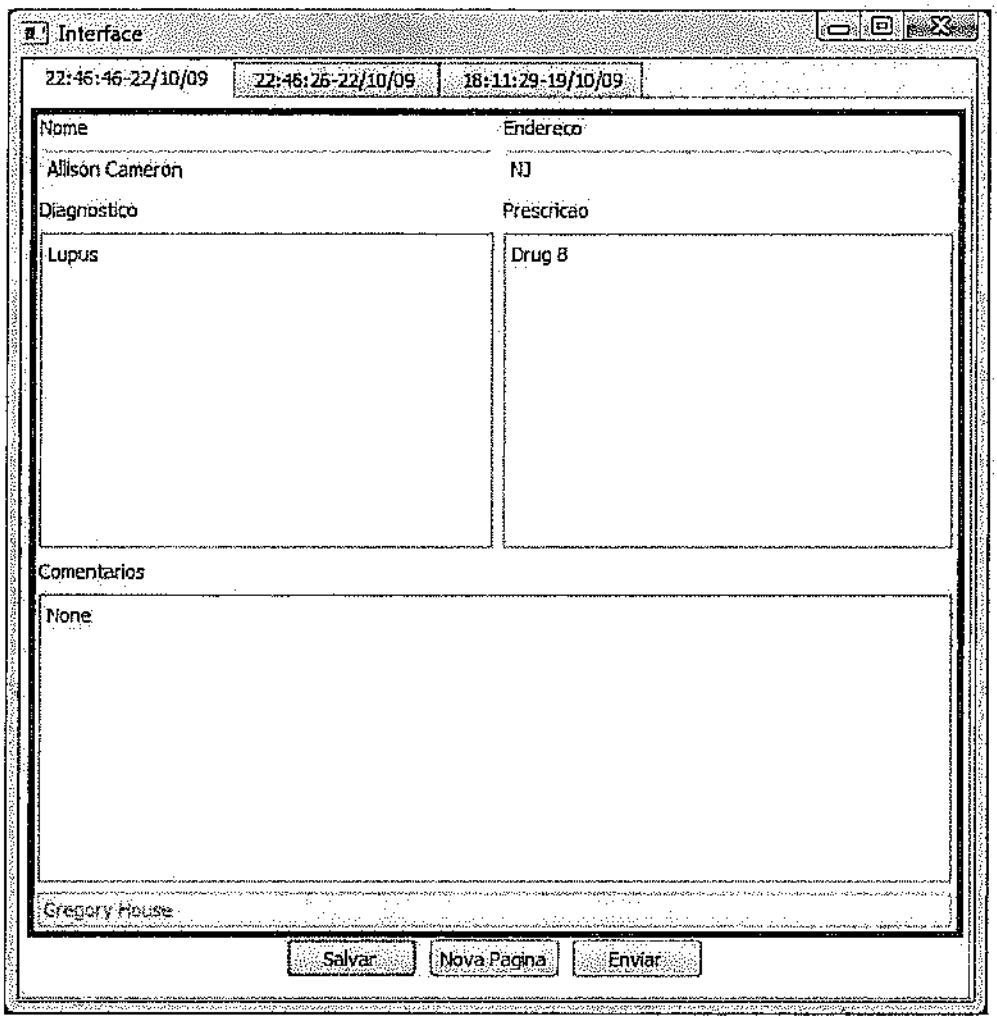


Figura 4 - Interface com os campos de texto destacados

A tela principal da interface apresenta uma série de campos de texto contendo as informações do paciente. Esses campos permitem que os usuários do sistema observem e modifiquem informações como o diagnóstico e as prescrições. Uma grande flexibilidade da interface, que será detalhada adiante, é que os campos apresentados na tela podem ser configurados pelo próprio usuário. Dessa forma, um hospital poderia, com muita facilidade, adicionar novas informações ao prontuário de forma organizada.

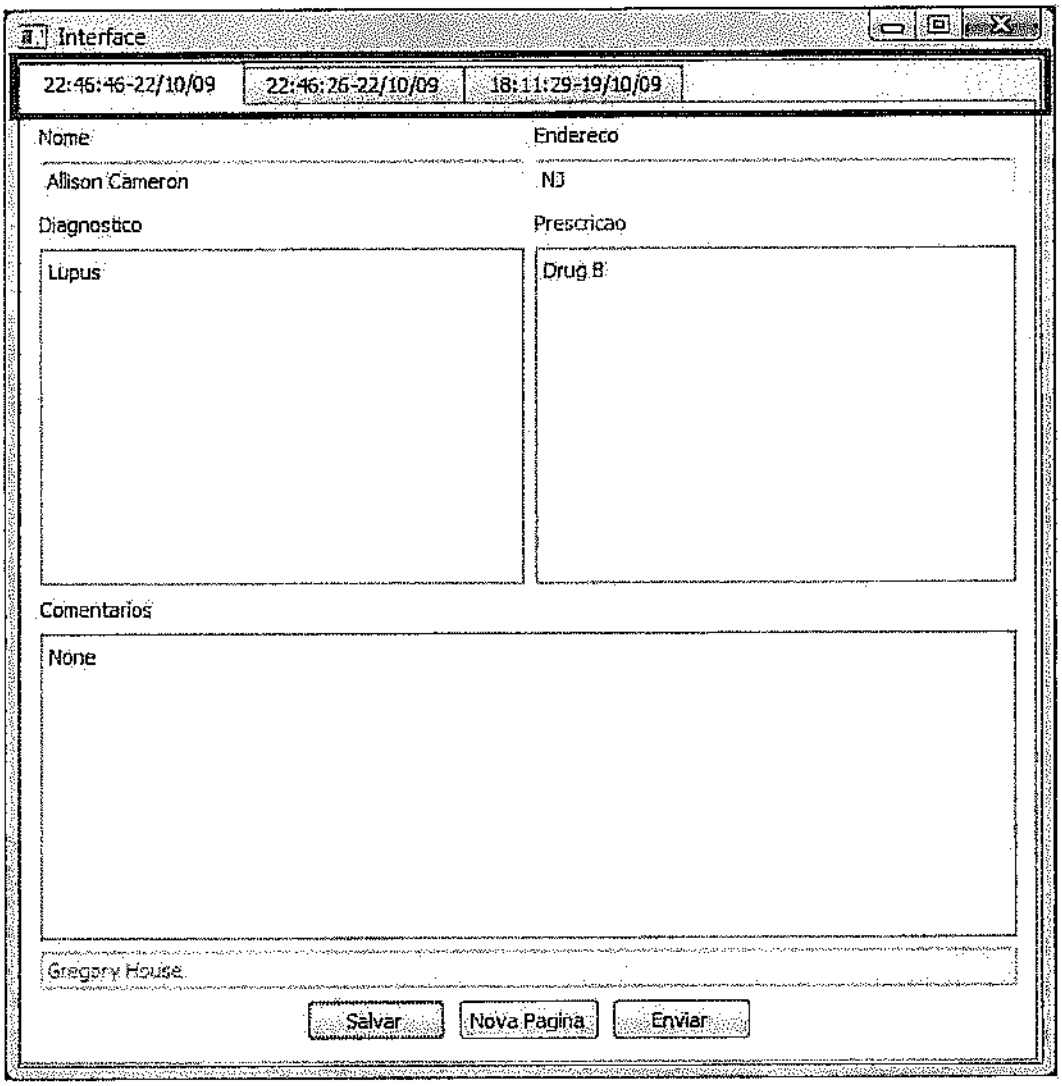


Figura 5 - Interface com as abas, representando instantes diferentes (páginas) do prontuário.

Pode ser observado também na tela a presença de diversas abas, cada uma com uma data. No sistema, cada aba representa um instante no prontuário. Essa decisão de design veio de três requisitos do projeto. Em primeiro lugar, foi preciso considerar que, ao longo do período de internação do paciente, diversas modificações serão feitas no prontuário, como novas prescrições e diagnósticos.

Em segundo lugar, não é permitido em hospitais que um médico ou enfermeiro altere ou adicione informações a uma folha de prontuário criada por um colega. Esse requisito é devido a questões de segurança; o seu objetivo é evitar que um profissional mal-intencionado modifique informações criadas por outro profissional.

O último requisito é que não deve ser permitido que um usuário modifique informações criadas anteriormente no prontuário. Esse requisito também é devido a considerações de segurança; ele tem como objetivo impedir que possíveis erros de usuários sejam omitidos através de modificações posteriores.

O design de abas representando folhas de prontuário atende aos três requisitos de projeto. O primeiro requisito (criação de diversas folhas de prontuário) é atendido através da criação de diversas abas, cada uma com uma data preenchida automaticamente representando o momento de criação. O segundo requisito (impedir a modificação de informações criadas por outros usuários) é atendido por um mecanismo que permite que apenas o criador de uma seja capaz de modificá-la. Todas as outras abas do prontuário são visíveis apenas para leitura (read-only). Já o último requisito (impedir a modificação de informações passadas) também é atendido através do mecanismo read-only; apenas a aba mais recente pode ser modificada e apenas no momento de sua criação. Uma vez que o prontuário for fechado, essa aba será considerada parte do passado, e a sua modificação será impedida. Para adicionar novas informações será preciso criar uma nova aba.

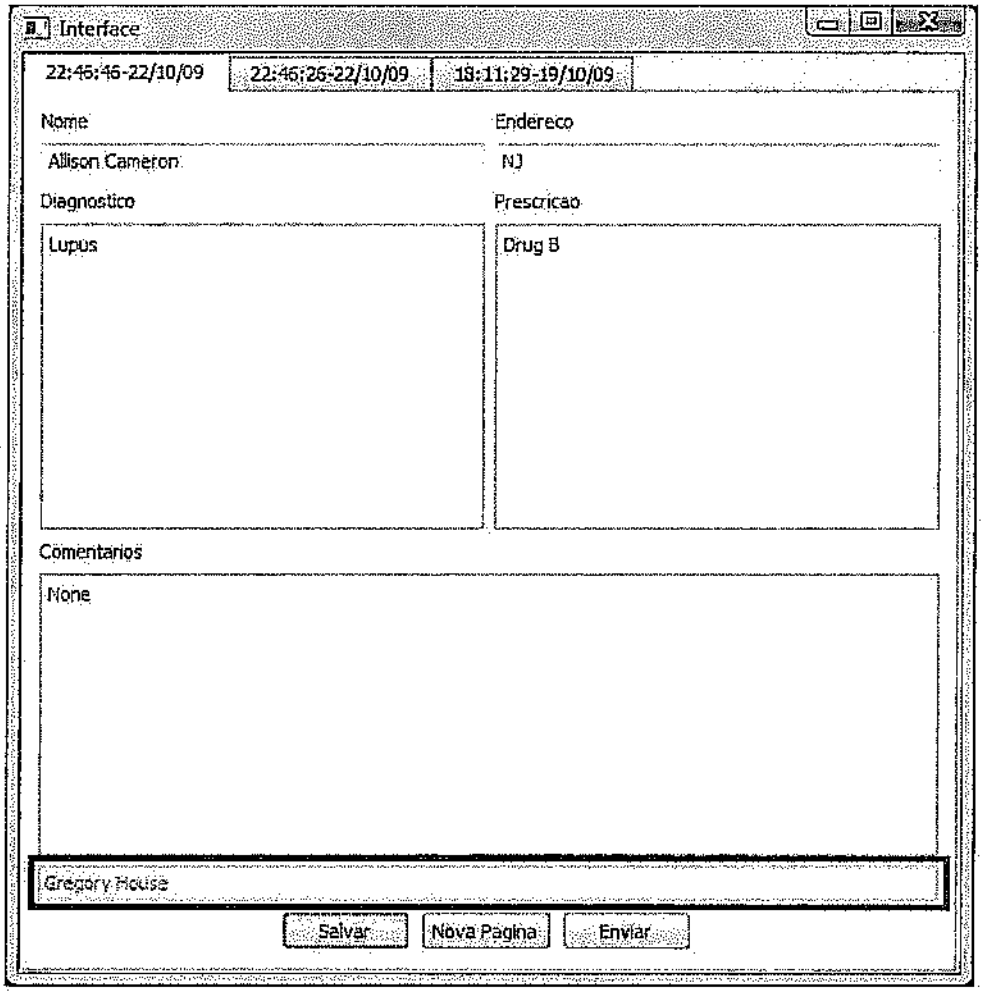


Figura 6 - Interface com o nome de usuário (read-only) destacado

É também possível observar na janela a existência de um campo contendo o nome do usuário. Esse campo, acessível apenas para leitura, é preenchido automaticamente pelo sistema após o login.

Ele foi criado para atender a mais um requisito de segurança: a associação de modificações nos prontuários com os usuários que as realizaram. Esse requisito é fundamental para permitir que os responsáveis por alterações no prontuário sejam identificados. O motivo desse campo ser read-only é impossibilitar que usuários mal-intencionados não se identifiquem ou atribuam mudanças no prontuário a outros usuários.

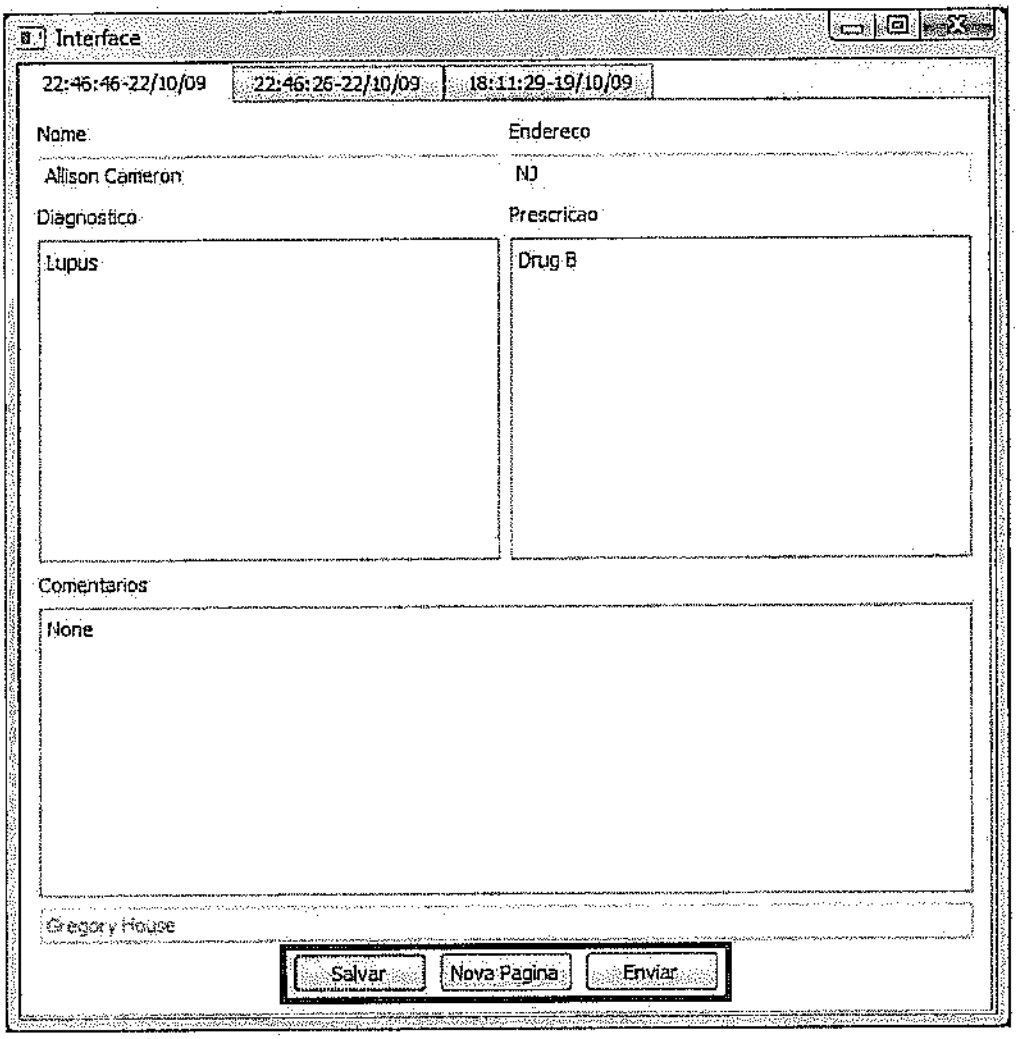


Figura 7 - Interface com os botões destacados

Por último, é possível observar a existência de botões na interface. Esses botões permitem que o usuário crie uma nova página de prontuário, salve as suas modificações na página e as envie para o Módulo de Dados. Devido à flexibilidade do design utilizado no programa, que será mostrada adiante, é possível criar novos botões com facilidade para atender a novos requisitos que os usuários possam ter.

Conforme foi explicado, a interface gráfica atende a diversos requisitos de projeto, muitos deles de segurança. Além de atender a esses requisitos, foi também considerada fundamental a facilidade de uso. Essa característica é muito importante

para não só possibilitar, mas também tornar mais rápida a adoção do sistema em hospitais e clínicas.

Dessa forma, a interface gráfica foi criada procurando-se apresentar ao usuário um ambiente simples e familiar. O visual da tela alcança esse objetivo ao ser consistente com o visual do sistema operacional do usuário: conforme será explicado adiante, a interface gráfica é multiplataforma e consistente com sistemas operacionais Windows, Mac OS X e Linux, todos em suas diversas versões.

Já o layout dos campos de informações foi criado com a simplicidade como objetivo principal. Os campos são estruturados de forma clara e agrupados de acordo com a sua categoria. Além disso, como mencionado anteriormente, o layout pode ser configurado pelos hospitais e clínicas em que o sistema for utilizado. Dessa forma, peculiaridades dos seus ambientes podem ser levadas em conta e melhorias podem ser realizadas com muita facilidade.

Gerenciador de componentes

Por ser responsável por toda a interação com os usuários, a Interface também é responsável por gerenciar a execução dos outros componentes do sistema. A abertura inicial da tela de prontuário, por exemplo, precisa das informações do prontuário para exibi-las na tela. Essas informações estão, porém, no Módulo de Dados, comprimidas e criptografadas.

Dessa forma, uma das primeiras ações executadas pela Interface é que execução do componente de Comunicação Bluetooth, em modo de recepção. Esse componente se encarrega, então, de realizar a comunicação com o Módulo de Dados através do Bluetooth e, de acordo com o protocolo definido, obter os dados do prontuário.

Esses dados estão, porém, criptografados e comprimidos (conforme explicado na seção Criptografia e Compressão). Assim, a segunda ação realizada pela Interface é a execução do componente de Criptografia, em modo de decifragem. Após a decifragem, os dados precisam ser, então, expandidos. Desse modo, a Interface executa o componente de Compressão, em expansão. Ao final de todos esses passos, os dados estão prontos para serem interpretados pela Interface e apresentados ao usuário.

De forma semelhante, também é necessário converter os dados quando o usuário desejar enviá-los de volta ao Módulo de Dados. Nessa etapa, os componentes de Criptografia e Compressão são executados novamente e, em seguida, o componente de Comunicação Bluetooth é executado em modo de envio.

Como a Interface é responsável por gerenciar a execução de diversos componentes, ela é responsável por encaminhar a saída da execução de um componente para a

entrada de outro, garantindo também que arquivos temporários sejam excluídos. Esse processo é semelhante ao processo de *piping* presente em sistemas operacionais.

IO

A terceira função realizada pela interface é de IO, ou seja, manipulação de dados no sistema de arquivos do sistema operacional. Essa operação, apesar de muito relacionada com a apresentação da interface gráfica, é um processo distinto.

Conforme será explicado adiante, os dados de cada página do prontuário são armazenados em um formato XML (*Extensible Markup Language*) simplificado. Dessa forma, a obtenção de informações a partir de arquivos e o seu armazenamento utilizam conversores para transformarem as informações presentas na tela em partes de arquivos XML.

A operação de IO também leva em conta a organização dos arquivos de dados e de configurações. Além disso, são também consideradas formas de reduzir o espaço ocupado pelos arquivos de dados através da criação de adaptadores de dados. Todas essas características são explicadas adiante.

4.1.3 Linguagem

A linguagem de programação escolhida para a implementação da Interface foi C++. Essa escolha foi devida a diversos motivos, como orientação a objetos, flexibilidade, velocidade, disponibilidade de bibliotecas, disponibilidade de toolkits gráficos e familiaridade.

A escolha de uma linguagem orientada a objetos foi fundamental. Essa característica facilita muito a organização do programa e a reutilização de componentes. Além disso, as interações entre as diversas partes do sistema são simplificadas consideravelmente quando os conceitos de classes e objetos são introduzidos.

A flexibilidade de C++ foi um dos principais fatores na escolha pela linguagem. Das linguagens existentes atualmente, C++ é uma das mais flexíveis. Além de apresentar características comuns a linguagens orientadas a objeto, como classes, objetos e templates, C++ também permite operações mais avançadas como herança múltipla e *overload* de operadores, entre outros. Essa flexibilidade amplia as possiblidades de design e deve ser considerada não apenas para o programa desenvolvido neste projeto, mas também para futuras extensões.

A velocidade de C++ é devida ao seu nível relativamente baixo. Seria possível utilizar, por exemplo, uma linguagem de *scripting* de alto nível para desenvolver o programa. O

protótipo inicial da interface, por exemplo, foi feito utilizando a linguagem AutoIt 3. Porém, devido à sua natureza, programas gerados através dessa linguagem são mais lentos. Como a sensação de uso foi uma das prioridades no projeto, a velocidade de execução foi fundamental.

A disponibilidade de bibliotecas foi importante devido ao tempo disponível para o projeto. Como C++ apresenta ótimos componentes de IO na biblioteca padrão, como iostream e fstream, a linguagem passou nesse requisito. Além disso, C++ também conta com as bibliotecas Boost, um conjunto de bibliotecas avaliadas por profissionais e consideradas de altíssimo nível. Essas bibliotecas podem ser muito úteis para facilitar, por exemplo, o acesso ao sistema de arquivos (Boost.Filesystem) e a utilização de threads (Boost.Thread) em extensões do sistema. Porém, no sistema apresentado no projeto, como o seu uso seria reduzido apenas à porções da funcionalidade de IO, o seu uso não seria compensado, de forma que elas não foram utilizadas.

Como uma das principais funcionalidades da Interface é a apresentação da interface gráfica, a existência de toolkits gráficos compatíveis com a linguagem escolhida foi essencial. C++ também apresentou diversas opções nesse requisito, como Qt e GTK+.

Finalmente, C++ teve uma grande vantagem por ser a linguagem orientada a objetos com a qual os dois integrantes do projeto tem mais familiaridade. Isso permitiu que menos tempo fosse utilizado com estudo sobre a linguagem e foi decisivo na escolha.

4.1.4 Toolkit gráfico

A interface gráfica é um dos principais componentes do projeto. Dessa forma, um dos primeiros passos no planejamento e na implementação dos componentes foi a procura de um toolkit gráfico.

Um toolkit gráfico é um conjunto de bibliotecas que disponibiliza componentes para a construção de interfaces gráficas. Esses componentes em geral são classes representando componentes para a construção de uma tela, como caixas de textos e botões, e mecanismos para gerenciar a interação entre o usuário, os componentes e o resto do programa.

Escolha de um toolkit

Existem diversos toolkits gráficos para C++. Alguns deles são específicos para determinadas plataformas, como as partes gráficas da Windows API, para Windows, ou do framework Cocoa, para Mac OS X. Por serem específicos para determinadas plataformas, esses toolkits têm a vantagem de utilizarem todo o potencial gráfico do sistema operacional. Por outro lado, o código gerado utilizando-os não é multiplataforma.

Como uma das principais e maiores partes do código desenvolvido no projeto é dedicada à interface gráfica, foi desejado que o programa desenvolvido fosse multiplataforma para que o trabalho investido nele gerasse mais resultados. Dessa forma, toolkits para sistemas operacionais específicos como os mencionados anteriormente foram descartados em favor de toolkits multiplataforma.

Existem também muitos toolkits multiplataforma em C++. Dois dos mais populares são o GTK+ e o Qt. O primeiro é um toolkit open source utilizado em projetos como o GNOME. O Qt, também tornado open source recentemente, é um toolkit da Nokia utilizado em aplicações como o Google Earth e o KDE. Entre esses dois toolkits, a decisão foi utilizar o Qt devido ao seu estado mais avançado e à sua documentação mais completa.

Nokia Qt

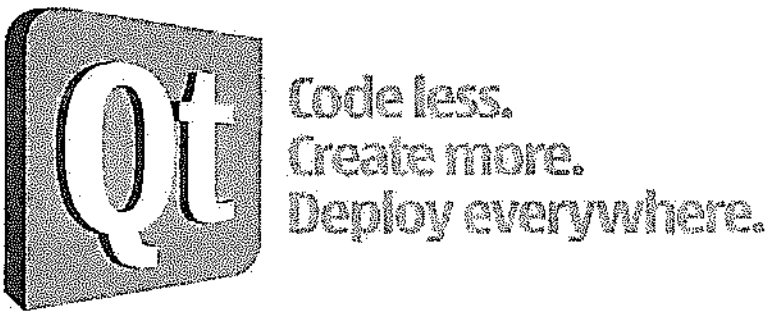


Figura 8 - Qt, o toolkit gráfico utilizado no projeto

O Qt é disponibilizado pela Nokia, a partir da versão 4.5, com a licença LGPL. Dessa forma, é permitido que o código do projeto usando o toolkit seja mantido proprietário caso seja desejado.

O Qt disponibiliza ao desenvolvedor uma série de classes representando objetos da interface gráfica. Existem componentes representando janelas, abas, botões, caixas de texto, entre muitos outros. Todos esses componentes tem um visual consistente com o do sistema operacional em que o programa é compilado. Essa é um grande vantagem

do toolkit, já que o mesmo código pode gerar programas consistentes com o visual do Windows, Mac OS X e Linux.

Além desses componentes visuais, o Qt também disponibiliza mecanismos para que usuário possa interagir com objetos na tela e para que essas interações possam se propagar para diferentes partes do programa. Um desses mecanismos é o sistema de Slots e Sinais. Utilizando esse sistema é possíveis adicionar os chamados slots a classes do programa, associados a diferentes sinais. Quando um desses sinais é obtido por uma classe (o que pode acontecer quando um usuário clica em um botão na tela, por exemplo), as classes contendo os slots correspondentes são sinalizadas. Dessa forma, o programa pode responder a ações do usuário de uma forma simples e clara. Esse sistema é semelhante ao padrão de design Observer.

Utilizando o Qt foi possível construir toda a interface de usuário do sistema. Através de testes apresentados em seções posteriores foi também possível verificar o seu funcionamento em diferentes plataformas.

4.1.5 Funcionamento

Após a decisão dos requisitos de projeto, da linguagem de programação e do toolkit gráfico, foi projetado o funcionamento do sistema. Ele é apresentado nessa seção e a estrutura do programa para a sua implementação é apresentada na seção seguinte.

Configurações

A execução do programa pelas configurações dos arquivos utilizados durante a operação. Os locais desses arquivos são lidos de um arquivo de configuração, config.txt.

Essa configuração é interessante por permitir que os arquivos de entrada e de saída de dados utilizados pelo programa possam ser modificados sem recompilação. Dessa forma, é fácil modificar a organização dos arquivos do programa.

Após a etapa de configuração, o programa já tem conhecimento sobre onde estão os arquivos a serem manipulados no sistema de arquivos do computador do usuário.

Conversão de dados

A próxima etapa na execução do programa é obter e converter os dados do prontuário, que nesse ponto estão criptografados e comprimidos no Módulo de Dados. Dessa forma, o programa executa a conversão de dados.

Em primeiro lugar, é executado o programa de Comunicação Bluetooth, e os dados são obtidos. Em seguida, os programas de Criptografia e Compressão são executados, de forma que os dados resultantes estão de acordo com o formato esperado pela Interface. Após a conversão os arquivos temporários são apagados do sistema de arquivos.

Adaptador de dados

Para reduzir o espaço ocupado na memória pelos dados do prontuário, foi criado um processo de união de todos os arquivos de prontuário antes do seu envio ao Módulo de Dados. Dessa forma, é preciso convertê-los de volta para múltiplos arquivos antes que a Interface possa prosseguir com a sua execução. Essa etapa foi chamada de Adaptação de Dados.

Criação da janela

Nesse momento da execução, a Interface tem à sua disposição todos os arquivos de prontuário e de configuração nos diretórios esperados, conforme a configuração inicial. O próximo passo é a criação da janela que será vista pelo usuário.

A janela é inicialmente criada vazia. Os componentes, como caixas de texto, são adicionados posteriormente.

Criação de abas

Em seguida, o programa inicia a criação das abas representando as páginas do prontuário. Para isso, o arquivo "pages.txt", vindo do Módulo de Dados, é lido; nesse arquivo existe uma lista contendo os nomes de todas as abas (que representam os seus tempos de criação) e os nomes dos arquivos onde estão os seus conteúdos. Para cada aba da lista, então, é criada uma nova aba na janela.

Cada objeto representando uma aba é configurado com o nome do arquivo onde está o seu conteúdo. Essa configuração é necessária para que o conteúdo das abas possa ser preenchido nas próximas etapas.

Configuração de layout

Nesse estágio da execução, o programa já tem uma janela criada contendo diversas abas vazias representando páginas do prontuário atual do paciente. O próximo passo é o preenchimento das abas com os componentes visuais, como caixas de texto e botões, chamados *widgets*.

Conforme mencionado anteriormente, o layout dos widgets pode ser configurado pelo usuário. Dessa forma, o programa lê um arquivo contendo essa configuração, chamado layout.txt, para quais widgets serão criados e onde eles serão posicionados.

Criação de layout

Após a determinação do layout, o programa começa a criar os widgets e adicioná-los à janela. Cada widget é associado a um arquivo onde os seus conteúdos devem ser lidos e escritos. Esses arquivos são os arquivos correspondentes às abas em que cada widget está.

Adição de botões

Após a adição dos widgets, são adicionados os botões às janelas. Por padrão, são adicionados três botões: Nova Página, Salvar e Enviar. Esses botões são posicionados no parte inferior da janela, após todos os widgets configurados pelo usuário.

Leitura de conteúdo

Após os passos anteriores, o programa apresenta uma janela preenchida com abas contendo os diversos widgets configurados pelo usuário. O último passo, então, é preencher o conteúdo dos widgets.

Isso é realizado através de uma iteração envolvendo todos os widgets, onde cada widget é requisitado a obter o seu conteúdo do seu arquivo e apresentá-lo na tela. Após essa etapa, todas as informações do prontuário podem ser visualizadas na tela.

Loop para atender o usuário

Nesse momento, o programa está pronto a apresentar a interface gráfica para o usuário e entrar em um loop aguardando pelos seus comandos. Os comandos podem ser a escrita de dados na tela, o acionamento de botões ou o fechamento da janela.

Botão "Nova Página"

Quando o botão Nova Página é ativado, é criada uma nova aba na tela correspondente ao momento de criação. Essa aba é então preenchida com widgets seguindo a configuração de layout. Na aba é adionado também o nome do usuário que a criou. A nova aba, representando uma nova folha no prontuário, pode ser então manipulada.

Botão "Salvar"

Quando o botão Salvar é ativado, é feita uma iteração por todos os widgets da aba atual. Cada widget executa grava, então, os seus conteúdos atualizados no arquivo correspondente à aba.

Botão "Enviar"

Quando o botão Enviar é ativado, os arquivos de dados do prontuário são unificados em um único arquivo contendo todo o conteúdo, para reduzir o espaço ocupado. Em seguida, os dados são comprimidos, criptografados e enviados ao Módulo de Dados.

4.1.6 Estrutura

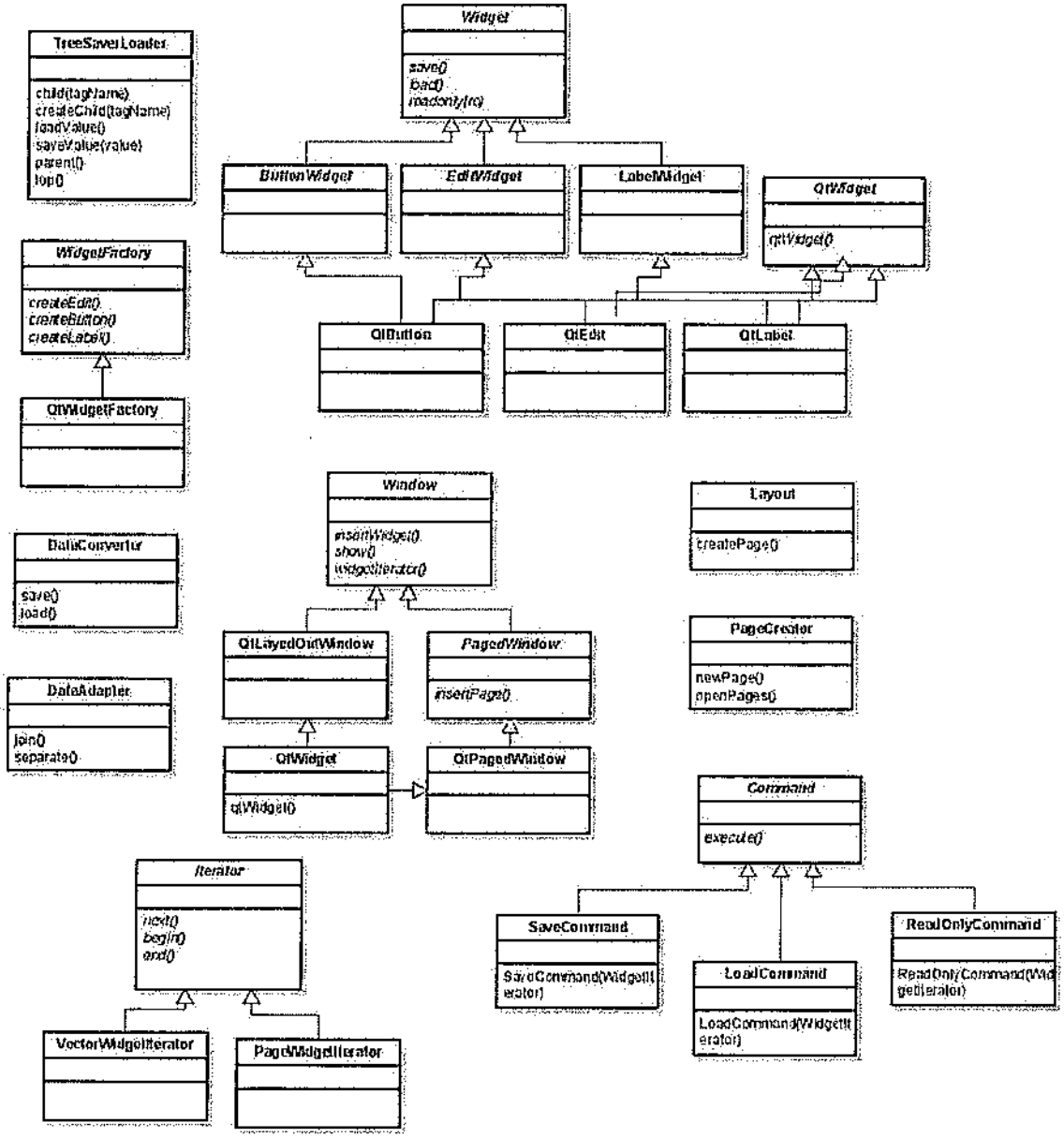


Figura 9 - Diagrama de classes completo da Interface

Será apresentado nesse item o design criado para implementar o funcionamento descrito anteriormente. A estrutura pode ser divida em oito partes: conversão de dados, widgets, layout, iteradores, commands, IO, decodificação e main.

Conversão de dados

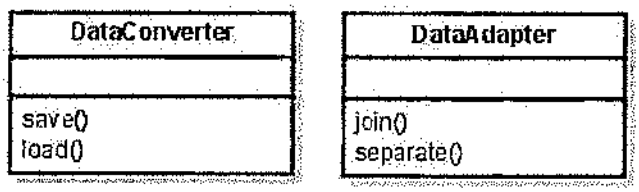


Figura 10 - DataConverter e DataAdapter

A conversão de dados é realizada por duas classes. A primeira, chamada DataConverter, é responsável por executar os programas de Criptografia, Compressão e Comunicação Bluetooth, além de criar um objeto DataAdapter.

A classe DataConverter apresenta duas operações: save e load. A operação save executa os comandos necessários para realizar a concatenação dos arquivos, compressão, criptografia e em seguida o envio dos dados ao Módulo de Dados. Já a operação load executa o comando de recepção de dados do Módulo de Dados e em seguida realiza a sua decifragem e expansão. Por último, é realiza a separação do conteúdo recebido em diversos arquivos representando páginas no prontuário, conforme esperado pelo resto do programa.

Os passos mencionados são executados de forma que a saída de um programa seja a entrada do programa seguinte. A execução desses diversos componentes gera arquivos temporários. Esses arquivos devem ser removidos tanto por questões de segurança quanto para reduzir o espaço utilizado em disco; esse trabalho também é responsabilidade da classe DataConverter.

A classe DataConverter contém a única parte do código de todo o sistema que não é multiplataforma. Isso ocorre porque a execução dos componentes mencionados anteriormente é realizada através da função "system()". Porém, como essa classe é pequena em comparação ao resto do programa e as suas operações são extremamente simples, seriam necessárias poucas modificações para tornar o programa multiplataforma.

A classe DataAdapter, utilizada pela classe DataConverter, é responsável por reduzir o espaço ocupado pelos dados no Módulo de Dados. Para isso, o conteúdo de todos os arquivos representando as páginas do prontuário é unificado em um único arquivo com tamanho reduzido. De forma complementar, essa classe também é capaz de gerar o múltiplos arquivos a partir do arquivo contendo todos os dados.

Dessa forma, a classe DataAdapter adapta os dados enviados pelo Módulo de Dados (um único arquivo) para o formato esperado pela Interface (múltiplos arquivos

nomeados de acordo com a data de criação de cada página do prontuário e um arquivo contendo uma lista de todas as páginas existentes).

Widgets

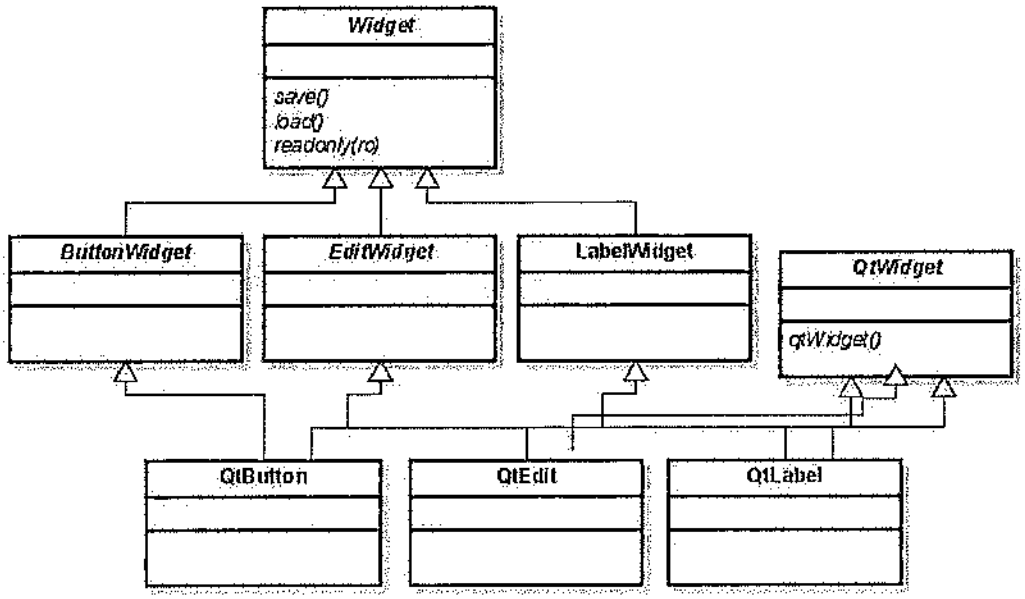


Figura 11 - Diagrama classes de Widgets

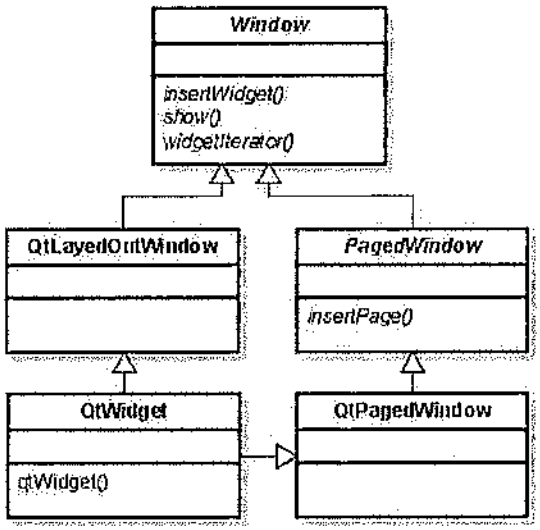


Figura 12 - Diagrama de classes de Window

A interface gráfica é composta por componentes visuais chamados de *widgets*. Os widgets podem ser desde botões e caixas de texto até abas e as janelas em si.

Conforme explicado anteriormente, o toolkit Qt oferece diversos tipos de widgets. Esses widgets poderiam ser utilizados diretamente para criar a interface gráfica.

Porém, foi considerado no design do programa que pode ser desejado, eventualmente, substituir o Qt por outro toolkit, ou até por outra versão dele mesmo.

Dessa forma, seguindo um dos princípios fundamentais de programação orientada a objetos, a dependência ao Qt foi encapsulada. O processo de encapsulamento envolve reunir todas as dependências ao Qt presentes no código e isolá-las através da criação de interfaces. O resto do programa passa a utilizar, então, apenas essas interfaces para acessar os componentes do Qt. Se um dia for desejado trocar o toolkit, bastará modificar a parte do programa que foi encapsulada, já que o resto do programa não está fortemente acoplado a ela.

Inicialmente foram criadas duas abstrações: Widget e Window. A inteface Widget tem o objetivo de representar objetos na tela como botões, textos e caixas de texto. Ela apresenta três funções principais, que devem ser implementadas pelas suas subclasses. A primeira é a operação "save", que deve salvar o conteúdo do componente em um arquivo. A segunda é a operação "load", que deve carregar o conteúdo de um arquivo e apresentá-lo na tela. A última é a operação "read-only", que muda o estado do componente entre "leitura e escrita" e apenas "leitura".

Foram ainda criadas subclasses de Widget, representando componentes específicos. Algumas delas são ButtonWidget, representando botões, EditWidget, representando caixas de texto, e LabelWidget, representando textos.

A segunda abstração, Window, representa tanto abas de uma janela como a própria janela. A sua função mais importante é "insertWidget". Através dessa função é possível adicionar widgets à uma janela ou aba. Foi ainda criada uma outra interface, que herda de Window, chamada PagedWindow. Essa interface representa uma janela com abas, e a sua única operação é "insertPage". Através dela é possível adicionar Windows à uma PagedWindow, cujo resultado visual é uma janela (PagedWindow) contendo abas (Windows).

A partir dessas interfaces foram definidos os componentes visuais do sistema as suas interações. Como pode ser observado, nenhum desses componentes apresenta operações dependentes da utilização do toolkit Qt. Dessa forma, o próximo passo foi a criação das implementações dessas diversas interfaces, utilizando os componentes fornecidos pelo Qt.

```
class QtEdit : public EditWidget, public QtWidget
{
public:
    ~QtEdit();
    QtEdit(std::string name, std::string filename);
    QWidget* qtWidget();
    void save();
    void load();
      void readonly(bool ro);
private:
    std::string _name;
    std::string _filename;
    QPlainTextEdit* _qtWidget;
};
```

Figura 13 - Código de QtEdit, representando como devem ser implementados Widgets.

As implementações dos Widgets são QtButton (que implementa ButtonWidget), QtEdit (que implementa EditWidget) e QtLabel (que implementa LabelWidget). Cada uma dessas implementações utiliza internamente um componente Qt, através de composição, para os quais as operações visuais são delegadas. São ainda adicionadas as operações necessárias para ler e escrever os conteúdos em arquivos. Para QtEdit, por exemplo, existem operações para escrever o seu texto em um arquivo. Já para QtButton, que não tem conteúdo, não é efetuada operação nenhuma.

A implementação de PagedWindow se chama QtPagedWindow. Associada a ela, foi criada a classe QtLayedOutWindow, que implementa Window. O objetivo dessa classe é representar as abas de uma janela, que serão adicionadas a QtPagedWindow. A vantagem dessa classe é que ela organiza automaticamente Widgets adicionados a ela em uma grade.

Com a criação das interfaces Widget e Window, e as suas implementações, é possível agora criar uma janela (interface PagedWindow, implementação QtPagedWindow) contendo abas (interface Window, implementação QtLayedOutWindow), onde cada aba contém widgets (interfaces ButtonWidget, EditWidget e LabelWidget, implementações QtButton, QtEdit e QtLabel). As outras partes do programa acessarão esses componentes apenas pelas suas interfaces, deixando de depender do Qt.

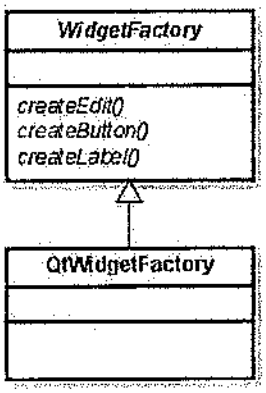


Figura 14 - Diagrama de classes de WidgetFactory

Por último, para que o programa dependa o mínimo possível do Qt, foi necessário abstrair a criação de Widgets. Para isso, foi criada a interface WidgetFactory, que apresenta funções como createButton, createEdit e createLabel. Foi então criada a implementação QtWidgetFactory, que retorna QtButtons, QtEdits e QtLabels.

Dessa forma, ao invés de instanciar objetos Widget diretamente (Widget* button = new QtButton), é possível realizar a operação utilizando a factory (Widget* button = widgetFactory.createButton()). Assim, caso seja eventualmente desejado trocar o Qt por outro toolkit, basta trocar a QtWidgetFactory por outra implementação que retorne Widgets de outra família correspondente ao novo toolkit.

Por último, foi criada a interface QtWidget, que deve ser implementada por todos os Widgets que usam Qt. Essa interface foi necessária para possibilitar a integração dos diversos componentes do toolkit.

O design mostrado nessa seção foi um dos fundamentos da Interface e resolveu diversos problemas encontrados. A criação da classe Widget permite que botões, caixas de texto e outros componentes sejam manipulados sem a necessidade de conhecer qual componente específico eles representam. Dessa forma, quando for necessário salvar o conteúdo de uma aba, por exemplo, é possível apenas realizar a operação "save" em todos os Widgets que ela contém; em Widgets como QtEdit, que contém informações a serem salvas, o processo necessário para a gravação será realizado; já em Widgets como QtButton, que não contém informação nenhum, nada será feito. Esse tipo de abstração torna o código muito mais simples.

Já a separação entre abas e janelas também facilita a sua criação. É possível criar as abas separadamente, inserindo Widgets, e depois simplesmente adicioná-las à janela.

Layout

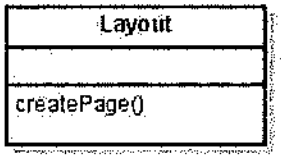


Figura 15 - Layout

O item anterior explicou como a criação de janelas, abas e widgets está estruturada. Porém, como um dos objetivos do projeto foi permitir que o posicionamento dos widgets em uma aba fosse configurável, o layout dos widgets nas abas não pode ser definido diretamente no código do programa. Ele deve ser lido de um arquivo de configuração e criado dinamicamente.

Para que isso fosse possível, foram definidos uma classe, chamada Layout, e um arquivo de configuração criado pelo usuário, chamado "layout.txt". Nesse arquivo, que tem um formato semelhante a XML, o usuário deve descrever o tipo, a posição e o conteúdo de cada widget a ser adicionado na tela.

```
<widget0>
        <type>label</type>
        <text>Diagnostico</text>
        <x>2</x>
        <y>0</y>
</widget0>
<widget1>
        <type>label</type>
        <text>Prescricao</text>
        <x>2</x>
        <y>1</y>
</widget1>
<widget2>
        <type>edit</type>
        <x>3</x>
        <y>0</y>
</widget2>
<widget3>
        <type>edit</type>
        <x>3</x>
        <y>1</y>
</widget3>
```

Figura 16 - Exemplo de layout.xml, mostrando como é simples a criação de um layout com dois labels, "Diagnóstico" e "Prescrição", e duas caixas de texto correspondentes.

O programa deve ser, então, capaz de ler o arquivo, criar os widgets definidos pelo usuário e inseri-los nas posições desejadas em cada aba; essa tarefa é realizada pela classe Layout. Essa classe, que apresenta apenas uma operação, "createPage", é capaz de ler o arquivo layout.txt e determinar todos os Widgets serem criados. A criação de cada Widget é feita, conforme mostrado no item anterior, utilizando uma WidgetFactory.

Em seguida, é criada uma nova aba onde os Widgets são inseridos nas posições definidas pelo usuário. As posições devem ser especificadas no formato de um par de coordenadas, representando as posições de cada widget na grade invisível presente em cada aba.

No final de sua execução, a aba criada pela classe Layout é retornada. Ela está então pronta para ser inserida na janela.

Iteradores

A partir das classes explicadas anteriormente, o programa é capaz de criar janelas contendo abas e widgets nas posições específicas pelo usuário através do arquivo de configuração layout.txt. Falta, então, adicionar um meio de manipular esses diversos

componentes. Isso é necessário para que seja possível, por exemplo, executar as operação "load" em todos os componentes de uma janela.

Para resolver esse problema, foi utilizado o padrão de design Iterator. Esse padrão define uma interface através da qual é possível acessar todos os itens de uma coleção de objetos, sem a necessidade de se saber como esses objetos estão organizados.

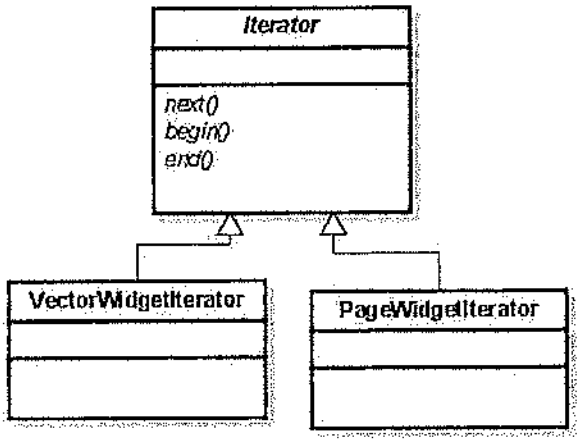


Figura 17 - Diagrama de classes de WidgetIterator

Inicialmente foi criado um WidgetIterator, contendo função como "begin", que retorna o primeiro Widget da coleção, e "operator++", que retorna o próximo Widget da coleção (operator++ é um exemplo de overload de operador, uma das vantagens de C++ descritas anteriormente). Assim, utilizando objetos dessa classe, é possível facilmente manipular todos os Widgets de uma dada coleção.

Foi então adicionada a operação "widgetIterator()" à classe Window. Essa operação deve retornar um WidgetIterator para todos os Widgets de uma dada Window. Além disso, foi adicionada a mesma função à classe PagedWindow. Nessa classe, a função deve retornar um WidgetIterator que manipula todos os Widgets de todas as abas de uma PagedWindow.

Dessa forma, qualquer parte do programa que tenha acesso a uma janela PagedWindow é capaz de manipular com facilidade todos os Widgets presentes em todas as abas dessa janela. Não é necessário saber quantas abas existem ou as suas posições. A aplicação de funções em Widgets é, então, consideravelmente simplificada.

Commands

Após a definição de um modo simples de acessar os Widgets, através dos WidgetIterators, foi possível implementar as diversas funcionalidades necessárias, como salvar os conteúdos da tela ou carregá-los de arquivos e apresentá-los. Deve ser

possível acessar essas funcionalidades de diferentes maneiras no programa. Por exemplo, o usuário deve poder acessar a funcionalidade de salvar através do botão "Salvar" presente na tela. Por outro lado, essa funcionalidade também deve ser acessível pelo código do programa.

Essa necessidade de acessar a mesma funcionalidade de modos diferentes pode ser satisfeita através de outro padrão de design, chamado Command. Essa padrão envolve o encapulamento da funcionalidade a ser executada em objetos que podem ser movidos ou recriados em diferentes partes do programa.

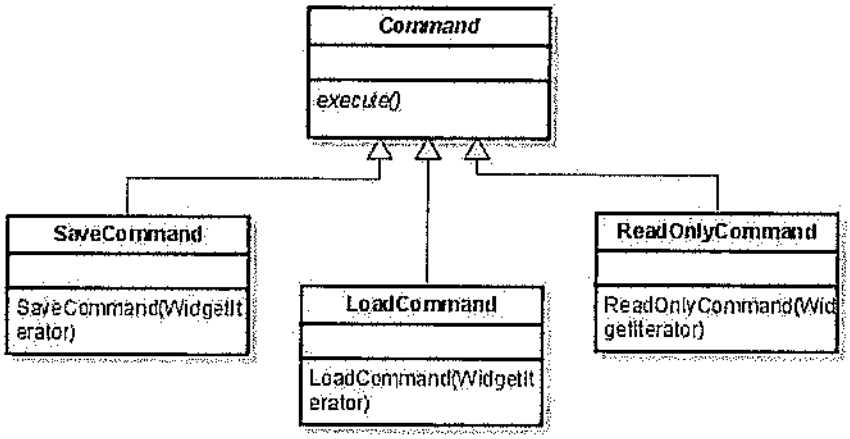


Figura 18 - Diagrama de classes de Command

Foi criada, então, a interface Command, contendo apenas uma função, "execute()". Quando essa função é executada, o objeto responsável pelo comando realiza a sua operação. Além disso, foram criadas três implementações de Command: SaveCommand, LoadCommand e ReadOnlyCommand.

```
class SaveCommand : public Command {
public:
    ~SaveCommand(){ delete _it; _it = 0; std::cerr << "Deleted
SaveCommand\n";}
    SaveCommand(WidgetIterator* it){ _it = it; }
    void execute() {
        _it->begin();
        while(!_it->end()) {
            _it->current()->save();
            ++(*_it);
        }
    }
private:
    WidgetIterator* _it;
};
```

Figura 19 - Código de SaveCommand, mostrando como é simples a implementação de um Command.

O primeiro, SaveCommand, é construído recebendo um WidgetIterator. A sua operação "execute" é responsável por invocar a função "save" em cada Widget

acessível pelo WidgetIterator. Dessa forma, pode ser criado um SaveCommand que recebe um WidgetIterator para todos os Widgets da janela e é capaz, então, de salvar toda a informação da janela com única e simples operação.

LoadCommand e ReadOnlyCommand têm funções muito parecidas. O primeiro é responsável por executar "load" em todos os Widgets, enquanto que o segundo é deve mudar o estado de todos os Widgets entre "leitura e escrita" e "somente leitura". Dessa forma, o sistema passa a ser capaz de carregar todas as informações da tela e de tranformar abas antigas em read-only.

Esses comandos são, então, distribuídos pelo programa. Quando a tela é aberta, por exemplo, é executado o LoadCommand para carregar as informações do prontuário. Além disso, é adicionado ao botão "Salvar" um SaveCommand. Cada aba tem também um ReadOnlyCommand, para mudar o seu estado.

Essa estrutura permite, de forma muito simples e rápida, a criação de novas funcionalidades no sistema que podem ser acessíveis de diversas formas diferentes. Seria possível criar, por exemplo, um menu na tela centralizando as operações de todos os comandos através da simples criação de mais instâncias dos comandos existentes.

IO

O design e as implementações descritas até o momento tornam o sistema capaz de mostrar a sua interface gráfica e executar operações de cada Widget da tela. Não foi explicado ainda, porém, como cada Widget realiza as suas interações com os arquivos para salvar e carregar o seu conteúdo.

Conforme foi explicado, cada Widget tem funções "save" e "load". Essas operações são capazes, através de um decodificador explicado no próximo item, de salvar o conteúdo do Widget e carregá-lo novamente. Porém, para que isso seja possível, cada Widget precisa saber onde o seu conteúdo deve ser salvo.

Esse requisito é resolvido com a configuração de cada aba da tela com um determinado arquivo no sistema de arquivos. Assim, quando uma determinada aba é criada e os seus Widgets são inseridos, eles são configurados, através da WidgetFactory, para o arquivo correspondente à sua aba.

Desse modo, quando um comando como SaveCommand for executado, cada Widget já conhece o arquivo onde o seu conteúdo deve ser salvo. Esse tipo de estrutura torna o programa muito mais simples do que se o próprio SaveCommand precisasse conhecer o arquivo de todos os Widgets da tela. Esse tipo de design é uma das grandes vantagens de programas utilizando objetos.

Decodificação

Todas as operações envolvendo arquivos realizadas no sistema são feitas utilizando uma versão simplificada de XML definida pelo grupo. As configurações do sistema, como o arquivo layout.txt, para especificar a aparência da tela, por exemplo, seguem esse formato. Além disso, a forma como o conteúdo do prontuário é salva também segue esse formato.

```
<widget0>
	<type>label</type>
	<text>Diagnostico</text>
	<x>2</x>
	<y>0</y>
</widget0>
<widget1>
	<type>label</type>
	<text>Prescricao</text>
	<x>2</x>
	<y>1</y>
</widget1>
<widget2>
	<type>edit</type>
	<x>3</x>
	<y>0</y>
</widget2>
<widget3>
	<type>edit</type>
	<x>3</x>
	<y>1</y>
</widget3>
```

Figura 20 - Parte de layout.xml, cujas informações podem ser decodigicadas por um objeto TreeSaverLoader.

A vantagem do formato é que ele permite a organização de informações de forma hierárquica. Por exemplo, no arquivo layout.txt existem diversas tags chamadas "widget". Dentro de cada uma dessas tags existem diversas outras tags, especificando o tipo de widget e a sua posição, por exemplo. Desse modo, quando o programa decodificar o arquivo layout.txt, será muito simples encontrar as informações referentes a cada widget. Da mesma forma, a identificação do conteúdo de cada widget será muito simples através da decodificação dos arquivos de conteúdo do prontuário.

A utilização desse formato precisa, porém, de um decodificador. Apesar de existirem diversas bibliotecas que realizam essa função, elas apresentam operações muito mais complexas do que as necessárias no projeto. Assim, para atender a necessidade do projeto de forma eficaz, foi criada um implementação simples de decodificador chamada TreeSaverLoader.

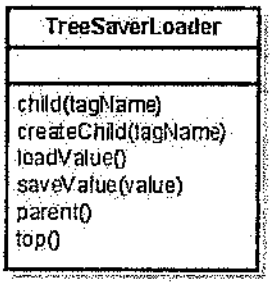


Figura 21 - TreeSaverLoader

O nome dessa classe é devido à forma como a decodificação é implementada. Quando um objeto TreeSaverLoader é criado, ele transforma o conteúdo do arquivo pseudo-XML em uma árvore de objetos contendo o conteúdo de cada tag do arquivo. Desse modo, a navegação pelo conteúdo do arquivo pode ser feita da mesma forma que a navegação em uma árvore.

A função "top()", por exemplo, faz com que seja visto o topo da árvore (todo o conteúdo do arquivo). A função "child(tagName)" muda o escopo para apenas o conteúdo da tag com nome "tagName". Já as funções "loadValue()" e "saveValue(value)" permitem a leitura ou modificação do conteúdo de uma tag. Existe ainda a função "createChild(tagName)", que possibilita a criação de uma nova tag no arquivo, e a função "parent()", que muda a visualização para a tag pai da atual. Por último, existe a função "hasChild(tagName)", que determina se o escopo atual contém a tag "tagName".

Através desses comandos é possível, de forma muito simples, localizar uma tag, mesmo que ela esteja dentro de outras, e carregar ou mudar o seu conteúdo, conforme mostrado na figura a seguir.

```
Arquivo.xml:

<parent>
      <child>
            value
      </child>
</parent>

Código:

TreeSaverLoader tree("Arquivo.xml");
std::string oldValue = tree.top().child("parent).child
("child").loadValue();
tree.top().child("parent).child("child").saveValue("newValue");
```

Figura 22 - Exemplo de arquivo XML simplificado e o código para acessar o seu conteúdo através de um TreeSaverLoader.

A criação e utilização dessa classe facilitou muito a implementação do sistema por definir um formato padronizado para gravação e leitura de dados. Desse modo, os

mecanismos de leitura e escrita só precisarão ser implementados uma vez e foram reutilizados muitas vezes ao longo de todo o sistema. Isso tornou o desenvolvimento não só mais rápido, mas também mais flexível, já que se for desejado o formato do arquivo pode ser mudado com facilidade.

Main

Após a criação de todas as partes do sistema mencionadas, faltou apenas definir o código principal. Esse código, representado na função main, lê as configurações do usuário utilizando um objeto TreeSaverLoader. Em seguida, objetos DataConverter e DataAdapter são utilizados para obter e converter os dados prontuário, vindos do Módulo de Dados. Após essa etapa, é criada uma QtPagedWindow e uma QtWidgetFactory.

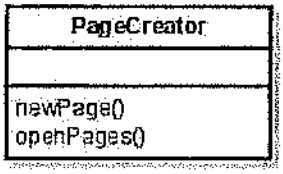


Figura 23 – PageCreator

Esses dois objetos são passados então para uma última classe, chamada PageCreator. Esse objeto é responsável por criar todas as abas da tela de acordo com o layout definido pelo usuário. Além disso, ele também é responsável por marcar o tempo de criação de cada nova aba, utilizando funções da biblioteca "ctime" de C++. Os tempos de criação são gravados nos arquivos das abas para que possam ser recuperados quando o sistema for iniciado novamente.

Finalmente, após a criação da janela pelo objeto PageCreator, o sistema entra em um loop aguardando pela interação do usuário, que é feita através da modificação do prontuário ou da ativação de botões (que por usa vez executam os Commands mencionados anteriormente).

4.2 CRIPTOGRAFIA E COMPRESSÃO

4.2.1 Objetivos

Um dos compromissos firmados ao início deste projeto foi em relação à segurança e ao sigilo dos dados armazenados e transmitidos pelo sistema. Deve-se impossibilitar que

terceiros não autorizados consigam ler as informações sigilosas a respeito do diagnóstico e do tratamento dados ao paciente.

Desta forma, o sistema de criptografia deve ser seguro e confiável, com a função de criptografar os dados que serão transmitidos e armazenados no módulo de dados na hora de sua gravação e decifrá-los na hora da sua leitura.

Somente profissionais autorizados têm acesso ao algoritmo de criptografia utilizado, impossibilitando a outros que não o conhecem, a sua decifragem.

Além disso, como os dados a serem armazenados são criados dentro de pastas contendo diversos arquivos, é necessária a existência de um sistema de compressão de arquivos, de modo que essa pasta seja substituída por apenas um arquivo compresso, diminuindo o seu tamanho em memória e facilitando as tarefas de criptografia e decifragem.

4.2.2 O Programa de Criptografia

O programa utilizado para a criptografia dos arquivos é baseado no GnuPG. O GNU Privacy Guard é a implementação completa e livre do padrão OpenPGP definido pelo RFC4880. O OpenPGP é baseado em uma família de softwares desenvolvida por Philip R. Zimmermann. A sigla PGP significa *Pretty Good Privacy*.

O GnuPG permite a criptografia e a assinatura de dados, por meio de um sistema de manuseio de chaves públicas e privadas. É um programa completamente executável por meio de linha de comando, permitindo sua fácil integração com outros aplicativos.

Basicamente, o programa funciona de seguinte maneira:

Cada usuário que deseje ler arquivos criptografados possui uma chave privada, que é pessoal e nenhuma outra pessoa tem acesso. Cada um também possui uma chave pública, que é única e está diretamente atrelada à chave privada. Essa chave, como o próprio nome diz, pode ser distribuída a todos os outros usuários que desejem criptografar arquivos a esse destinatário.

Na execução da criptografia, o remetente, de posse da chave pública do destinatário, cria o arquivo criptografado de acordo com ela. Ao receber o arquivo criptografado, o destinatário só poderá decifrá-la caso possua a chave privada diretamente relacionada à chave pública utilizada pelo remetente. Caso sejam diferentes, é impossível a sua decifragem.

O programa utilizado neste projeto é baseado em apenas uma chave pública e uma privada que só podem ser acessadas através do programa que as chama. Como

somente usuários autenticados podem ter acesso a ele, a segurança do sistema fica garantida.

O funcionamento do programa de criptografia pode ser observado no diagrama de blocos disposto abaixo:

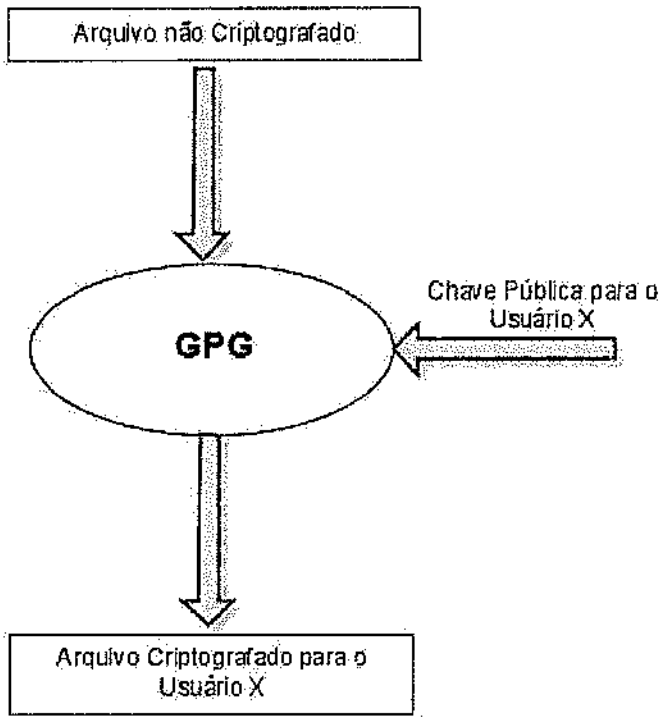


Figura 24 - Funcionamento da Criptografia

E o funcionamento do programa de decifragem pode ser observado no diagrama de blocos abaixo:

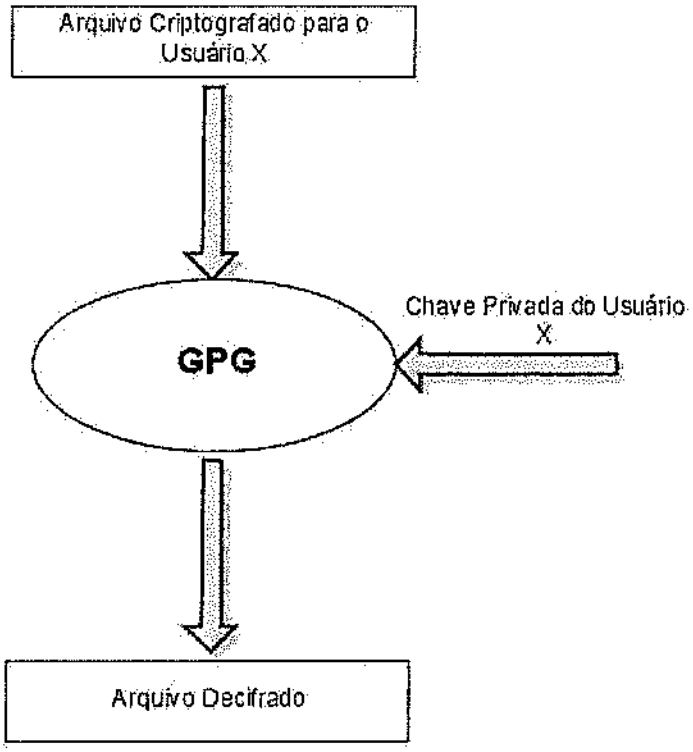


Figura 25 - Funcionamento da Decifragem

4.2.3 O Programa de Compressão

O Programa utilizado para a compressão é baseado no ZIP 3.0 e no UNZIP 6.0. Ambos são portáteis, *open source* e largamente difundidos ao redor do mundo. São completamente executáveis pela linha de comando, o que facilita muito a integração ao programa utilizado.

O aplicativo desenvolvido simplesmente informa os parâmetros necessários aos dois programas citados para as tarefas necessárias (compressão ou descompressão), comprimindo a pasta contendo os arquivos de informações do paciente antes de criptografá-la e enviá-la ao módulo de dados ou descomprimindo o arquivo decifrado que veio do módulo.

4.2.4 Problemas Encontrados

Até chegar ao produto final, muitos problemas tiveram de ser superados. Entre eles, destaca-se:

- Escolha do algoritmo de criptografia: Muitos algoritmos diferentes foram estudados. Desde os mais complexos até os mais simples. Foi cogitada a utilização do algoritmo Base 5, em que a representação dos caracteres é toda feita na base 5. Este algoritmo foi descartado, pois qualquer pessoa com o conhecimento de seu funcionamento poderia decifrá-lo, ao contrário do GPG, em que somente a pessoa com a chave privada correspondente consegue ler a mensagem.

- Escolha do programa de compressão e descompressão: Foi necessário encontrar um programa de compressão e descompressão que funcionasse por meio de linha de comando. Após muita procura, optou-se pelo Zip.exe e Unzip.exe para a realização dessas tarefas.

4.3 AUTENTICAÇÃO

Essa seção apresenta o primeiro contato do usuário com sistema, a autenticação.

4.3.1 Objetivo

O objetivo do componente de Autenticação é garantir que apenas usuários cadastrados tenham acesso ao sistema. Esse requisito é extremamente importante para evitar que dados de prontuários sejam visualizados por pessoas não autorizadas que, por algum motivo, tenham acesso aos computadores de usuários autorizados.

Dessa forma, o componente de Autenticação visa a criação de um método simples, confiável e rápido de verificação da permissão de acesso de um usuário ao sistema.

4.3.2 Funcionamento

O funcionamento do componente parte do princípio de que todos os usuários autorizados a utilizar o sistema tenham um "login" (nome de usuário) e uma senha. Todos esses usuários devem ser cadastrados no sistema a ser criado.

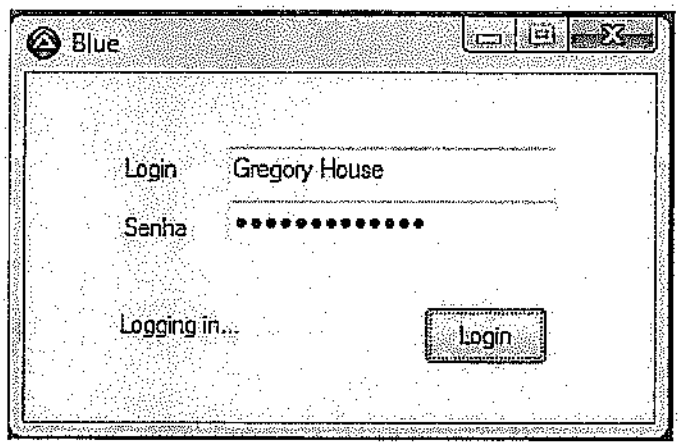


Figura 26 - Janela de autenticação de usuário

O programa de Autenticação é o primeiro contato de qualquer usuário com o sistema. Ele é iniciado quando o usuário clica no ícone do programa em seu computador. Em seguida, o componente de Autenticação exibe uma tela contendo duas caixas de texto, com os títulos Login e Senha, e um botão "Login".

O usuário deve, então, preencher os campos com o seu nome de usuário e senha. Como na maioria dos programas de autenticação, o campo de senha troca o texto sendo digitado por símbolos, para evitar que outras pessoas descubram a senha do usuário simplesmente observando a tela de seu computador.

Em seguida, quando o usuário pressiona o botão "Login", o sistema realiza a verificação da existência do usuário na lista de pessoas cadastradas no sistema. Como será explicado adiante, essa verificação leva em conta diversos requisitos de segurança.

Caso o usuário seja encontrado, a tela de Autenticação é fechada e o programa Interface é executado, exibindo as informações do prontuário. Caso contrário, o usuário tem a oportunidade de tentar novamente. Conforme será mostrado, essa possibilidade também leva em conta questões de segurança.

4.3.3 Implementação

Nessa seção será explicada a implementação do componente de Autenticação, cujo funcionamento foi descrito anteriormente. A implementação foi feita utilizando as linguagem de programação C++ e a linguagem de scripting AutoIt3.

Problemas

Como a autenticação é um dos principais métodos para garantir a segurança do sistema, existem diversas questões a serem analisadas. A primeira delas é relacionada ao método de armazenamento de logins e senhas no sistema de arquivos. Essas informações claramente não podem ser gravadas como arquivos de texto não criptografados, pois qualquer pessoa seria capaz de descobrir os logins de todos os usuários apenas observando o arquivo. Além disso, para tornar a segurança do sistema realmente consistente, esse arquivo de senhas não deve ser exposto em momento algum da execução do programa. O seu conteúdo deve ser indecifrável em qualquer instante de execução.

Um outro critério de segurança é garantir que apenas uma combinação correta de login e senha permita o acesso ao sistema. Um login e uma senha corretos, mas não associados, não devem ser aceitos, já que isso permitiria que um usuário modificasse o prontuário com o login de outro usuário.

Crypt

Parte da resolução dos problemas mencionados anteriormente foi a utilização da função "crypt", originalmente implementada em C. Essa é uma função que foi criada no sistema operacional Unix e é utilizada atualmente em sistemas operacionais Unix e Unix-like para autenticação.

Essa função utiliza criptografia para transformar uma string em uma sequência indecifrável. Não existem algoritmos conhecidos para reverter o processo de criptografia utilizado. Dessa forma, e considerando-se que essa função é utilizada a mais de 30 anos, ela pode ser considerada extremamente confiável.

A principal dificuldade para usar a função crypt foi a localização de uma implementação open source multiplataforma. Diversas implementações foram encontradas para Linux e Unix, mas, depois de muito esforço, foi encontrada apenas uma adequada para utilização também no Windows.

Essa procura foi, porém, recompensada. Como será mostrado a seguir, essa função é um dos fundamentos do processo de autenticação utilizado no projeto.

Cadastro de usuários

Inicialmente é preciso cadastrar os usuários no sistema. Esse processo é realizado fornecendo como argumentos ao programa de Autenticação o login e a senha do novo usuário a ser cadastrado, além da string "new".

Em primeiro lugar, o programa concatena os valores obtidos em uma nova string. Essa string é então criptografada utilizando a função "crypt", junto com um "sal". Esse processo gera uma nova string, contendo o resultado da criptografia.

Essa nova string é então armazenada no arquivo de senhas. É importante observar que em nenhum momento o arquivo de senhas foi decifrado nesse processo.

Autenticação

O primeiro passo na autenticação é a exibição da janela de login, criada com a linguagem de scripting AutoIt3. Essa janela contém uma caixa de texto comum, onde o usuário deverá digitar o seu login, e uma caixa de texto especial para senhas (que oculta o seu conteúdo através de símbolos), onde o usuário deverá digitar a sua senha. Quando o usuário clica no botão "Login", a execução passa para um programa escrito em C++ para realizar a autenticação a partir dos dados digitados.

O processo de autenticação criado foi baseado no processo utilizado em sistemas Unix. A primeira parte do programa inclui o header da biblioteca contendo a função crypt.

Quando a execução do programa é iniciada, o login e a senha que devem ser verificados são passados automaticamente como argumentos pela linha de comando. O programa realiza então a concatenação dessas duas informações em uma única string. Em seguida, essa nova string é criptografada pela função crypt, com a adição de um "sal". Essa função retorna o resultado da criptografia.

O resultado é então comparado com as entradas já existentes no arquivo de senhas. Caso seja encontrada uma entrada com o mesmo valor, é possível afirmar com segurança que o usuário está cadastrado no sistema. Dessa forma, a tela de login é fechada e o programa da Interface é executado, exibindo o prontuário do paciente.

Porém, caso não seja encontrada nenhuma entrada igual, é confirmado que as informações utilizadas para login estão incorretas. Nesse caso, é permitido que o usuário tente realizar o login novamente.

Existe um cuidado a ser tomado antes que seja permitida uma nova tentativa de login. Para evitar ataques de força bruta, em que um grande número de logins e senhas é tentado para realizar a autenticação no sistema, é forçada uma pausa de 3 segundos antes de cada nova tentativa. Desse modo, um ataque de força bruta demoraria um tempo muito grande, tornando-se inviável. Essa é uma prática comum em sistemas de autenticação modernos.

Desse modo, é possível perceber que o sistema descrito resolve todos os problemas mencionados inicialmente. Como os logins e senhas são concatenados antes da verificação, é impossível que valores corretos mas não associados sejam utilizados

para a autenticação. Além disso, como o arquivo de senhas nunca é decifrado, os logins e senhas de usuários cadastrados nunca são expostos.

4.4 COMUNICAÇÃO

Uma parte essencial deste projeto é a comunicação entre os módulos de dados e de usuário. Todo o desenvolvimento do projeto foi baseado na transmissão de dados sem fios e é isso o que o torna inovador frente às demais alternativas disponíveis para o preenchimento e o armazenamento de prontuários em hospitais.

Optou-se pela utilização do Bluetooth, que é uma tecnologia largamente utilizada e conhecida. Portanto, muita informação pôde ser encontrada na rede, auxiliando na implementação do sistema.

O hardware necessário será detalhado mais adiante em uma seção específica.

Basicamente, o desenvolvimento da etapa de comunicação pode ser dividido em três partes interdependentes: o protocolo, o programa do módulo de dados e o programa do módulo de usuário.

4.4.1 O Protocolo

De modo a efetuar a comunicação sem fios entre os módulos, foi desenvolvido um protocolo próprio para especificar as operações básicas do sistema.

Sua criação foi necessária para facilitar a execução das duas principais operações do sistema de comunicação: a transferência de dados do módulo de usuário para o módulo de dados e a operação inversa, do módulo de dados para o módulo de usuário.

A ausência de um protocolo específico inviabilizaria a execução dessas tarefas, que são essenciais para o funcionamento do sistema.

O protocolo é baseado na transmissão e no reconhecimento de algumas palavras chaves, que facilitam as operações de leitura e escrita dos arquivos.

Estas palavras são:

- **load**: Identifica a requisição feita pelo módulo de usuário para a leitura do arquivo de dados presente no módulo de dados.

- **save**: Identifica a notificação feita pelo módulo de usuário para a gravação do arquivo com as informações do paciente no módulo de dados.

- **start**: Identifica o início do arquivo com os dados do paciente.

- **end**: Identifica o fim do arquivo com os dados do paciente.

O funcionamento do protocolo pode ser descrito como segue:

- Envio do arquivo do módulo de usuário para o módulo de dados:

O módulo de usuário envia um sinal com a palavra "save" ao módulo de dados, que monitora os sinais enviados a ele. Em seguida, são enviados nesta ordem: a palavra "start", o arquivo com as informações sobre o paciente e a palavra "end". O módulo de dados, ao identificar o recebimento da palavra "save", espera e grava na memória a palavra "start", recebida. Nos endereços de memória subseqüentes, são gravados os bytes recebidos, correspondentes ao arquivo enviado pelo módulo de usuário. Ao identificar o recebimento da palavra "end", o módulo de dados a grava na memória e fica em estado ocioso, esperando o recebimento de outras palavras de comando.

O diagrama de blocos abaixo ilustra a operação descrita:

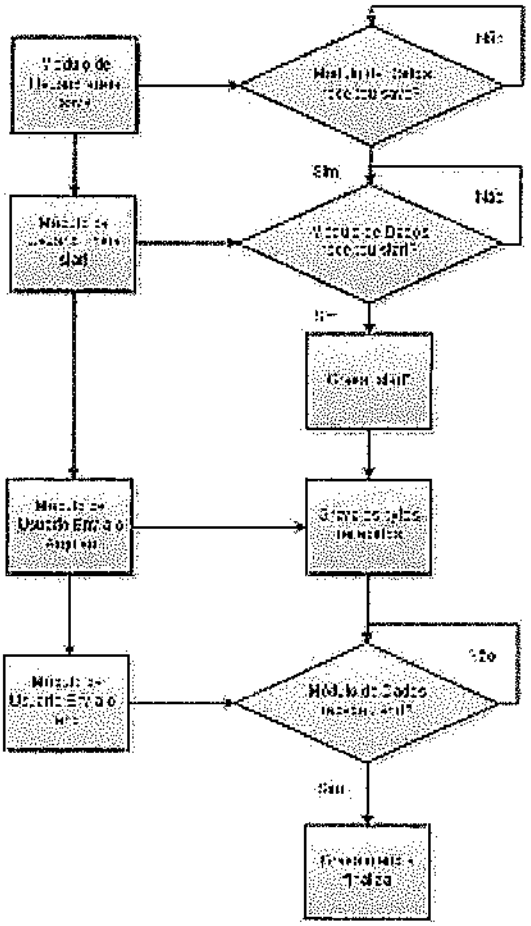


Figura 27 - Envio do arquivo do módulo de usuário ao módulo de dados

- Envio do arquivo do módulo de dados para o módulo de usuário:

O módulo de usuário envia um sinal com a palavra "load" ao módulo de dados. Este, ao recebê-lo, inicia a transmissão do arquivo presente em sua memória, desde a palavra "start" até a palavra "end". O módulo de usuário espera o recebimento da palavra "start" identificando o início do arquivo de dados. Então, grava em um buffer todos os bytes recebidos até identificar a palavra "end", verificando, assim, o final do documento.

O diagrama de blocos abaixo ilustra a operação descrita:

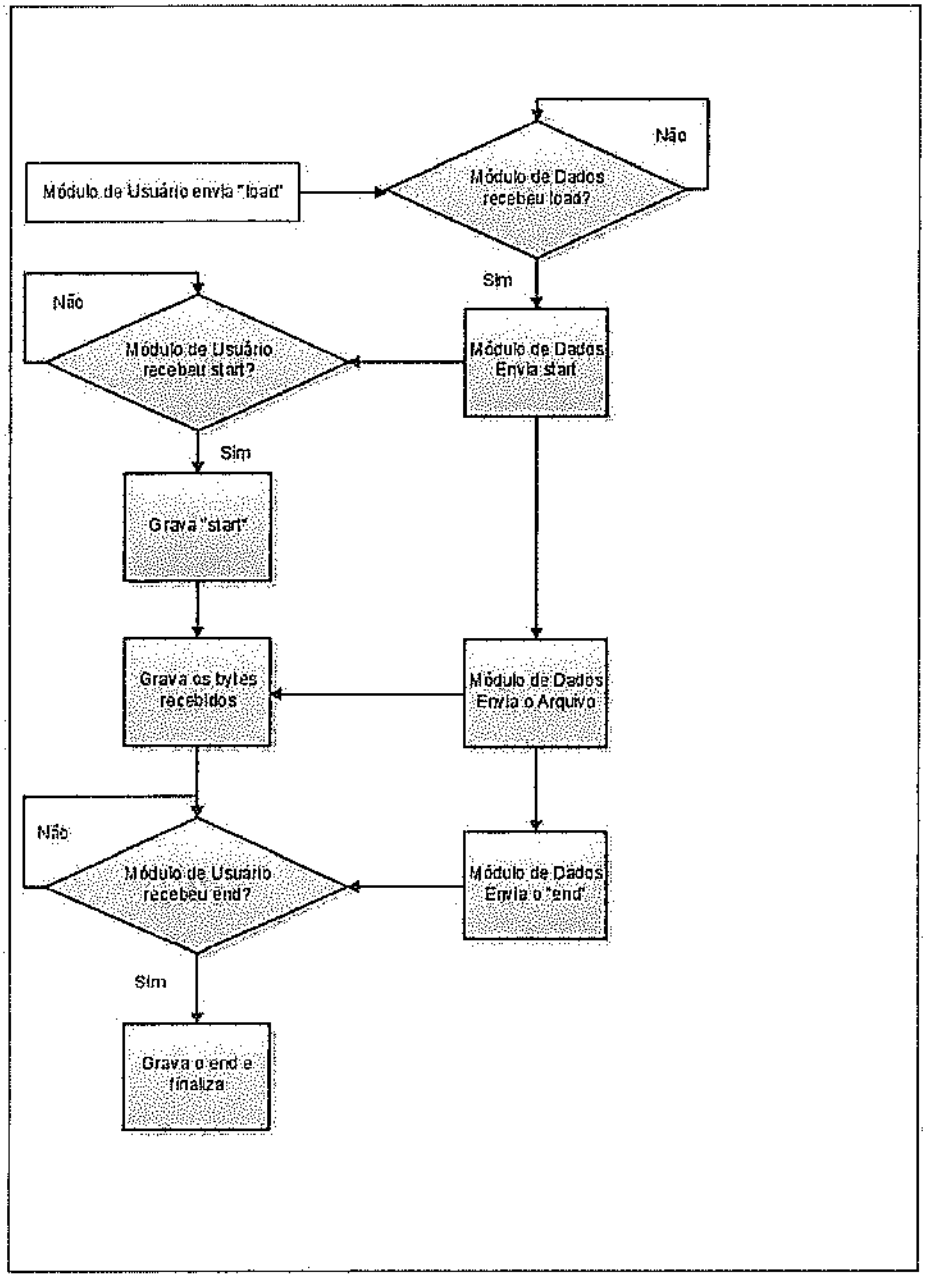


Figura 28 - Envio do arquivo do módulo de dados para o módulo de usuário

As palavras escolhidas para o funcionamento formam o vocabulário do protocolo do sistema de comunicação sem fios entre os módulos. Nota-se que utilizou-se o inglês, de modo a representar as operações: load e save para carregar e gravar, respectivamente e start e end para o começo e o fim do arquivo.

Problemas Encontrados

Na elaboração e desenvolvimento do protocolo utilizado, muitos problemas houveram de ser superados. Pode-se destacar:

- Tentativas de se efetuar comunicação sem protocolo: mostraram-se muito complicadas e pouco eficientes, pois os módulos não tinham a capacidade de

discriminar os bytes sendo recebidos e a hora certa de enviar ou receber dados. Tampouco era possível determinar o começo e o fim de um arquivo presente no módulo de dados.

- Vocabulário utilizado pelo protocolo: as letras das palavras utilizadas são enviadas caractere por caractere convertidos em bytes. Ocasionalmente, pode haver coincidência entre um byte do arquivo e um byte da letra utilizada pelo protocolo. Algumas tentativas com palavras menores ou simplesmente letras de sinalização mostraram-se ineficazes devido a essa coincidência, causando problemas como o encerramento da transmissão precocemente. De acordo com testes efetuados, as palavras finais escolhidas são pouco prováveis de causarem problemas de coincidência com os bytes do arquivo de dados do paciente.

4.4.2 O Programa do Módulo de Dados

O software programado na memória flash do microcontrolador ATMEGA168 da placa de desenvolvimento ARDUINO BT foi desenvolvido com o auxílio da biblioteca ARDUINO para *Processing*, que é uma *integrated development environment* (IDE) que utiliza uma versão simplificada da linguagem C++.

Ao alimentar a placa Arduino, esta era imediatamente encontrada pelo computador com suporte a Bluetooth, que criava uma porta serial para comunicação.

Carregando um firmware chamado *Firmata* para a placa Arduino, é possível utilizar a biblioteca descrita acima para a comunicação e a programação de diversas placas Arduinos utilizando o *Processing*. Desta maneira, para a programação do Microcontrolador no Arduino Bluetooth, não foi necessária qualquer conexão física ao computador, deveu-se apenas informar a porta serial de comunicação criada pelo sistema operacional e a gravação do programa ocorreu sem fios pelo Bluetooth.

O programa do módulo de dados basicamente utiliza o protocolo descrito no item acima para a comunicação e gerenciamento da memória do dispositivo. Para isso, foram criadas quatro funções básicas: receive(), send(), read(), write(); e duas funções que as utilizam para realizar as operações: save(), load(); além da função de configuração e a função principal do programa. Cada uma delas está descrita detalhadamente abaixo:

- **receive()**: Recebe o byte enviado pelo módulo de dados.

Espera o recebimento de um inteiro por meio de comunicação serial. A cada inteiro recebido, verifica se é válido, o converte para o byte correspondente e retorna este.

- **send()**: Envia um byte para o módulo de dados.

Recebe como parâmetro o byte a ser enviado e o envia por meio de comunicação serial.

- **read()**: Lê um byte na memória.

Recebe como parâmetro um número inteiro, que representa o endereço de memória que deve ser lido do Arduino. Caso seja menor que 512, que é o tamanho do buffer na memória RAM, retorna o byte contido no endereço dado do buffer. Caso seja maior que o tamanho do buffer na RAM e menor que a soma deste com o tamanho da memória EEPROM, que também é de 512 bytes, retorna o byte contido no endereço da EEPROM correspondente à subtração do número dado e do tamanho do buffer.

- **write()**: Grava um dado na memória.

Recebe como parâmetros o byte a ser escrito na memória e um número inteiro correspondente ao endereço de memória em que deve ser gravado. Caso seja menor que o tamanho do buffer na memória RAM, é gravado nesse endereço no buffer. Caso seja maior que ele e menor que sua soma com o tamanho da memória EEPROM, é gravado no endereço da EEPROM correspondente à subtração do número dado e do tamanho do buffer.

- **save()**: Grava o arquivo inteiro enviado pelo módulo de usuário na memória.

Descarta todos os bytes recebidos até que seja verificado o recebimento da palavra que indica o início do arquivo (*start*). Ao recebê-la, grava nas primeiras posições de seu buffer na memória RAM.

Grava todos os bytes válidos recebidos na seqüência, até a verificação do recebimento da palavra que indica o fim do arquivo (*end*). Ao recebê-la, grava nas posições seguintes ao arquivo recebido em sua memória e sai da função.

- **load()**: Envia o arquivo inteiro de sua memória para o módulo de usuário.

Lê os bytes desde o primeiro endereço do buffer de memória RAM e os envia para o módulo de usuário até que seja verificado o envio da palavra que indica o fim do arquivo (*end*), que também é enviada. Sai da função em seguida.

- **setup()**: Função de configuração da comunicação serial no Arduino.

Seta o *Baud Rate* do canal de comunicação serial em 115200 e chama uma função específica da biblioteca Arduino para essa tarefa.

- **loop()**: Função principal do Arduino.

Roda em um loop infinito recebendo todos os bytes válidos que chegam por meio de comunicação serial.

Ao receber a palavra de comando (*save*), que foi enviada pelo módulo de usuário indicando que este deseja gravar um arquivo na placa, o programa chama a função **save()**, que executa a operação.

Ao receber a palavra de comando (*load*), que foi enviada pelo módulo de usuário indicando que este deseja receber o arquivo gravado na memória da placa, o programa chama a função **load()**, que executa tal operação.

Para implementar as funções de escrita e leitura da memória EEPROM do microcontrolador, foi utilizada a biblioteca **EEPROM** que, basicamente, utiliza duas funções específicas para cada uma dessas duas operações:

- **EEPROM.read()**: Lê dado da EEPROM

Recebe como parâmetro um inteiro representando o endereço da EEPROM que se deseja pegar o valor. Retorna o byte correspondente a esse endereço.

- **EEPROM.write()**: Escreve dado na EEPROM

Recebe como parâmetro um inteiro representando o endereço da EEPROM que se deseja gravar o valor e o byte correspondente. Grava esse valor no endereço dado.

Para implementar as funções de comunicação pela porta serial, foram utilizadas funções da biblioteca Serial do próprio Arduino. As duas funções utilizadas foram:

- **Serial.read()**: Lê dado recebido.

Recebe o dado recebido pela porta serial de comunicação como um número inteiro.

- **Serial.print()**: Envia dado.

Recebe como parâmetro o byte a ser enviado e a denominação da variável a ser enviada, no caso *BYTE*. Envia esse byte pela porta de comunicação serial.

O funcionamento do programa pode ser observado no diagrama de blocos abaixo:

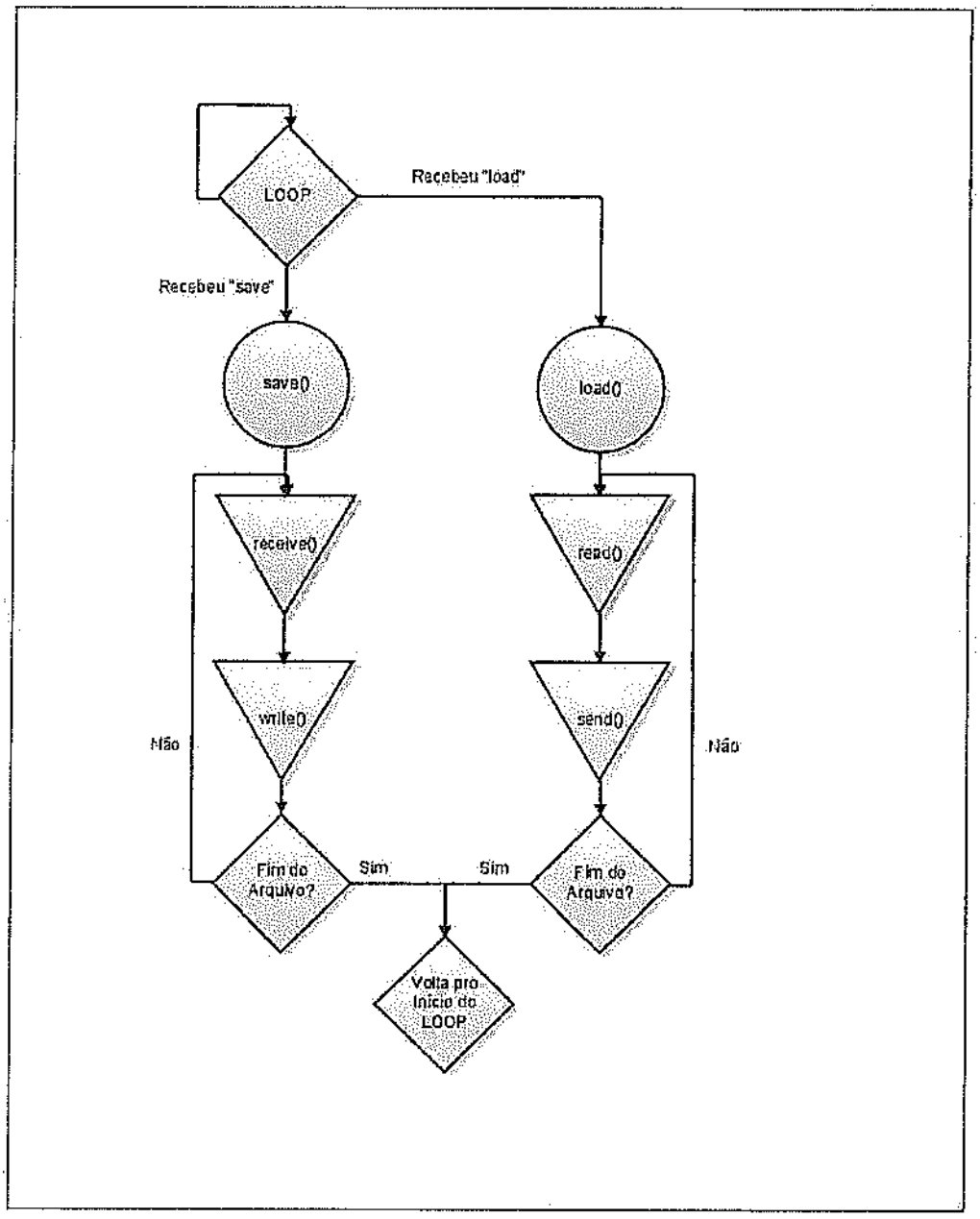


Figura 29 - Funcionamento do programa do módulo de dados

É possível perceber que as funções **read()** e **write()** foram desenvolvidas de maneira a utilizar no máximo 1 kilobyte de memória, utilizando 512 Bytes de memória RAM e 512 Bytes de memória EPROM. Isso foi feito, pois o tamanho máximo de memória no microcontrolador da placa Arduino BT disponível é de 1 kilobyte de RAM e 512 Bytes de EPROM.

Problemas Encontrados

Entre os principais problemas encontrados nesta etapa do desenvolvimento destacam-se:

- Problemas de Gravação do programa no microcontrolador:

Durante o período inicial de testes de programação do microcontrolador, foi constatado grande dificuldade na gravação do programa compilado no dispositivo. Na verdade, nenhuma tentativa de gravação havia obtido êxito.

Após a procura por soluções em fóruns especializados, chegou-se a solução de sempre que fosse necessária a gravação de um programa compilado no módulo de dados, era necessário resetar a placa através de um botão de reset presente na mesma antes de efetuar a operação de upload.

Assim, com a adoção desta solução, não houve mais problemas de gravação na placa.

- Limitação da memória para armazenamento no protótipo:

Como mencionado anteriormente, a memória disponível para armazenamento é de 512 Bytes na EEPROM do microcontrolador da placa e de 1 kilobyte em sua RAM. Como o microcontrolador utiliza a RAM para executar diversas operações durante seu funcionamento, utilizou-se metade da RAM disponível para armazenamento junto com a memória EEPROM.

O datasheet do ATMEGA 168 especifica o limite máximo de 100000 operações de gravação em sua memória EEPROM. Deste modo, optou-se pela utilização, primeiro de um buffer de 512 Bytes na memória RAM para, em seguida, armazenar o restante do arquivo na memória EEPROM.

Nos testes realizados, notou-se que o arquivo completo dificilmente ocuparia mais de 1 kilobyte na memória, que é o limite imposto pelo dispositivo e pelo programa. Portanto, concluiu-se não ser necessária qualquer atitude adicional para a expansão da capacidade de memória neste protótipo.

- Recebimento de bytes inválidos pela comunicação serial:

Nos primeiros testes, notou-se que muitos bytes recebidos eram inválidos, não representando qualquer parte do arquivo esperado nem das palavras de comando configuradas.

Na realidade, a função Serial.read(), que é utilizada para a leitura do sinal recebido pela comunicação serial, recebe um inteiro e, em seguida, dentro da função receive(), esse inteiro é transformado em byte. Constatou-se que os inteiros inválidos recebidos tinham valor -1. Portanto, após cada recebimento, foi feita a verificação de valor comparando-o ao valor -1, a fim de eliminar os dados inválidos.

4.4.3 Comunicação serial - Computador

Objetivo

O programa de comunicação serial que fica no computador do usuário é responsável por implementar a parte do Módulo de Usuário do protocolo descrito anteriormente. Esse programa deve levar em conta diversas considerações; as principais delas são como utilizar o dispositivo Bluetooth presente no computador do usuário e como garantir a consistência dos dados enviados e recebidos.

Funcionamento

O programa de comunicação serial no computador tem a sua operação decidida através um argumento passado pela linha de comando (o que é feito automaticamente pela Interface). As duas possibilidades são "save" e "load".

Caso seja passado o argumento "save", o programa entra em modo de envio de informações para o Módulo de Dados. Em primeiro lugar, ele abre o arquivo a ser enviado (cujo nome também é passado pela linha de comando). Em seguida, o conteúdo do arquivo é lido e armazenado.

O próximo passo executado pelo programa em modo "save" é o envio do conteúdo do arquivo seguindo o Protocolo definido anteriormente. Em outras palavras, é enviada a palavra "save" ao Módulo de Dados, para que este se prepare para a recepção. Em seguida, é enviada a palavra "start", o conteúdo do arquivo e a palavra "end". Como o protocolo Bluetooth garante o envio das informações, não são necessárias verificações adicionais.

Por outro lado, caso seja recebido o argumento "load" pela linha de comando, o programa entra em modo de recepção de dados. Ele se prepara, então, para receber do Módulo de Dados o conteúdo do arquivo, envolvido pelas palavras "start" e "end". O conteúdo do arquivo é então salvo na localização passada como segundo argumento pela linha de comando.

Esses dois modos de operação permitem que a Interface se comunique com o Módulo de Dados sem precisar ter conhecimento de como este armazena os dados e de que a comunicação é feita por Bluetooth.

A seguir serão descritos os problemas encontrados na implementação do funcionamento descrito acima e a estrutura do programa criada para resolvê-los.

Problemas encontrados

Devido ao contato próximo ao hardware tanto do dispositivo Bluetooth quando do Módulo de Dados, foram encontrados diversos problemas nesta parte do projeto. O primeiro deles está relacionado com a manipulação do dispositivo Bluetooth do computador do usuário. Como será explicado a seguir, o ideal seria que o modo de comunicação fosse feito através de abstrações.

Além disso, existiram problemas relacionados com a leitura e escrita realizada no Módulo de Dados e no Módulo de Usuário. Como esses dois módulos apresentam ambientes completamente diferentes, ocorreram diversos problemas de conversão de dados.

Biblioteca RXTX

O primeiro passo da implementação foi a procura por uma biblioteca que permitisse a manipulação de dispositivos Bluetooth. A melhor opção encontrada foi a biblioteca open source RXTX, implementada em Java. Apesar de que todo o resto do sistema é implementado predominantemente em C++, foi considerado vantajoso o uso dessa biblioteca devido à sua simplicidade.

A principal vantagem dessa biblioteca é que, exceto por configurações iniciais de Bluetooth, como a porta serial utilizada, todo o resto da comunicação é feito como se ela estivesse ocorrendo através de uma porta serial. Dessa forma, não é necessário o conhecimento do protocolo Bluetooth.

Estrutura

A estrutura do programa criado pode ser dividida em 5 partes, representadas por 5 classes: Main, Protocol, SerialStream, FileStream e Debug. Cada uma dessas classes realiza um papel bem definido no programa e será explicada a seguir.

```
public class FileStream {
	private Debug debug = new Debug(Debug.Mode.OFF);
	private InputStream in;
	private OutputStream out;
	private String inputFilename;
	private String outputFilename;

	private void writeByte(byte b);
	private int readByte();

	public FileStream(String inputFilename, String outputFilename);
	public void close();
	public void debugMode(Debug.Mode debugMode);

	public void write(byte[] bytes);
	public void write(Vector<Byte> bytes);
	public byte[] read();
}
```

Figura 30 - Definição de FileStream.

A primeira classe criada, FileStream, tem o objetivo de facilitar o acesso ao sistema de arquivos e realizar as conversões necessárias para que os dados enviados e recebidos do Módulo de Dados estejam corretos. Conforme pode ser visto na figura, essa classe apresenta um InputStream e um OutputStream. Esses streams representam arquivos de entrada e de saída de dados no sistema de arquivos.

Em seguida, foram definidas duas funções, "writeByte" e "readByte". Essas funções são as únicas duas no programa de comunicação serial que realizam diretamente o envio e a recepção de dados de um arquivo. Essas funções garantem a leitura e a escrita de exatamente um byte de um arquivo. Foi muito importante que essa quantidade exata fosse utilizada, por os dados lidos são, posteriormente no programa, enviados ao Módulo de Dados.

Foram criadas então funções "read" e "write", que utilizam as funções "readByte" e "writeByte". Essas funções servem para facilitar a escrita e a leitura de dados. A primeira função, "read", lê todo o conteúdo de um arquivo e o armazena em uma matriz de bytes. Já a função "write" escreve uma matriz ou um "vector" de bytes recebidos como argumento em um arquivo.

É possível notar ainda a presença de um construtor, que recebe os nomes dos arquivos manipulados, e de uma função "close", que fecha os streams. Essa última função é essencial, por os streams sempre devem ser fechados no término da execução do programa.

A partir da classe FileStream, o resto do programa é capaz de ler e escrever dados em arquivos do sistema de arquivos sem precisar conhecer o formato em que eles são lidos e escritos. Desse modo, a classe FileStream resolve todos os problemas relacionados ao sistema de arquivos.

```
import gnu.io.CommPort;
import gnu.io.CommPortIdentifier;
import gnu.io.SerialPort;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class SerialStream {
	private Debug debug = new Debug(Debug.Mode.OFF);
	private InputStream in;
	private OutputStream out;
	private CommPort commPort;
	private SerialPort serialPort;
	private boolean AUTOFLUSH = false;

	private void sendByte(byte b);
	private byte receiveByte();

	public SerialStream(String portName);
	void connect(String portName);
	public void close();
	public void debugMode(Debug.Mode debugMode);

	public void send(byte b);
	public void send(byte[] bytes);
	public void send(String s);
	public byte receive();
}
```

Figura 31 - Definição de SerialStream.

Em seguida, de forma análoga à classe FileStream, foi criada a classe SerialStream. Essa classe tem operações semelhantes às apresentadas por FileStream, mas ao invés de realizar input e output em arquivos, realiza essas operações com a porta serial representando o dispositivo Bluetooth do usuário.

Em primeiro lugar, é possível perceber a presença de dois streams, assim com na classe FileStream. As funções "sendByte" e "receiveByte" enviam e recebem exatamente um byte pelo Bluetooth, de forma semelhante às funções "writeByte" e "readByte" de FileStream. Já as funções "send" e "receive" são funções auxiliares para a recepção e o envio de múltiplos bytes por Bluetooth, de forma análoga às funções "read" e "write".

Por último, é possível notar a presença do construtor, da função "connect" e da função "close". O construtor recebe a porta serial pela qual a comunicação Bluetooth será realizada. Em seguida, ele executa a função connect, que realiza a conexão com o Bluetooth e abre cria os streams de comunicação. Finalmente, a função "close" é encarregada de fechar os streams.

A semelhança entre FileStream e SerialStream poderia ser levada mais adiante. Poderia ser projetada uma interface Stream, implementada pelas duas classes, que criaria uma abstração tanto para a escrita no sistema de arquivos quanto para o Bluetooth. Esse tipo de abstração, que trata dispositivos como arquivos, é um dos fundamentos de sistemas Unix.

```
import java.util.Vector;


public class Protocol {
	private Debug debug = new Debug(Debug.Mode.OFF);
	private String FILE_START_DELIM = "start";
	private String FILE_END_DELIM = "end";
	private String PROTOCOL_SAVE_COMMAND = "save";
	private String PROTOCOL_LOAD_COMMAND = "load";
	private String PROTOCOL_STATUS_COMMAND = "status";

	private FileStream fileStream;
	private SerialStream serialStream;
	public Protocol(Debug.Mode debugMode, String portName, String
inputFilename, String outputFilename);

	public void saveFile();
	public void loadFile();
}
```

Figura 32 - Definição de Protocol.

Após a criação dos Streams mencionados acima, foi criada a classe Protocol. Essa classe encapsula o protocolo explicado anteriormente neste texto.

Ela apresenta duas funções principais "saveFile" e "loadFile". A primeira lê um arquivo do sistema de arquivos e executa os passos definidos no protocolo para o enviar ao Módulo de Dados. Já a segunda executa os passos necessários para receber um arquivo do Módulo de Dados e em seguida o salva no sistema de arquivos.

A classe Protocol possui um objeto FileStream e um objeto SerialStream, de forma que as duas operações descritas anteriormente utilizam os Streams para realizarem toda a escrita e leitura e o envio e recepção de dados tanto para o sistema de arquivos quanto para o sistema operacional. O construtor de Protocol recebe tanto a porta serial quanto os nomes dos arquivos a serem manipulados, de forma que ele seja capaz de configurar os Streams. A estrutura criada por essas três classes é muito interessante, porque torna muito simples a mudança tanto do protocolo quando do formato de envio dos dados.

Finalmente, após a definição das três principais classes descritas acimas, foram criadas as classes Main e Debug. A primeira apenas cria um objeto da classe Protocol, configurando-o com a porta serial e os nomes dos arquivos corretos, recebidos pela linha de comando. Em seguida, através de outro parêmetro da linha de comando, que pode valer "save" ou "load", a operação "saveFile" ou "loadFile" de Protocol é executada, respectivamente.

A classe Debug foi usada apenas durante o desenvolvimento. Ela serve para configurar a quantidade de mensagens de Debug apresentadas pelo programa durante a sua execução. Apesar de simples, ela facilitou a detecção de erros.

O programa descrito resolve todos os problemas encontrados. A comunicação por Bluetooth é facilitada pela biblioteca RXTX. Os problemas de conversão são resolvidos

pelas classes FileStream e SerialStream. Já o protocolo é implementado de forma simples pela classe Protocol.

4.5 INTEGRAÇÃO DO SOFTWARE

Nas seções anteriores foram apresentados todos os componentes de software do sistema criado no projeto. Conforme pode ser observado, cada componente executa funções específicas, tendo um papel importante no sistema.

A integração de todos esses componentes é a última parte para tornar o sistema operacional. Como explicado anteriormente, grande parte da integração foi realizada no componente Interface.

Como esse componente tem um papel central na execução do sistema, ele determina a ordem de operação de diversos outros componentes. Eles são os de Criptografia, Compressão e Comunicação Serial.

Dessa forma, restou apenas a integração entre a Autenticação e a Interface. Ela foi implementada utilizando a linguagem de scripting AutoIt3. Como o primeiro contato do usuário é com a tela de Autenticação, é natural que esse componente execute, no seu término, a Interface.

Por último, sobra o componente de Comunicação Serial do Módulo de Dados. Ele é executado em paralelo a todos os outros componentes, no Módulo de Dados. Dessa forma, o papel do Módulo de Dados é semelhante ao de um servidor.

4.6 HARDWARE

4.6.1 A Placa Arduino BT

Para possibilitar a comunicação por meio da tecnologia Bluetooth e o armazenamento do arquivo com os dados do paciente em um dispositivo específico, são necessários um módulo de comunicação Bluetooth e um microcontrolador.

Optou-se pela utilização da placa de desenvolvimento Arduino BT, que é uma placa Arduino com conexão a um módulo Bluetooth, permitindo a comunicação sem fios.

Arduino é uma plataforma de prototipagem eletrônica livre, baseada na facilidade de utilização tanto do hardware quanto do software necessários.

A placa Arduino BT apresenta as seguintes características:

- O uso de um conversor DC-DC, permitindo que a placa seja alimentada com no mínimo 1,2 V e, no máximo 5,5 V.

- Um microcontrolador ATmega168.

- Módulo configurado somente para comunicação serial com Baud Rate de 115200 s^{-1}.

- O Módulo Bluetooth utilizado na placa é o Bluegiga WT11

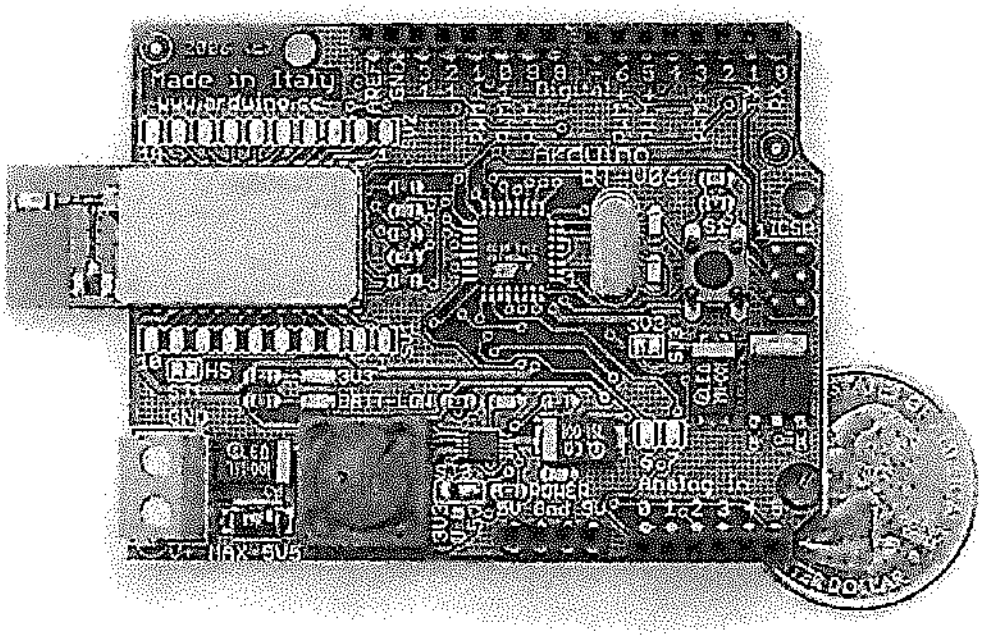


Figura 33 - A placa Arduino BT

O Bluegiga WT11 é um módulo de Bluetooth versão 2.1 + EDR (*Enhanced Data Rate*) de classe 1. Alcança transmissões de dados até três vezes mais rápidas que os módulos Bluetooth versão 1.2 consumindo menos energia. Suas principais características são:

- Bluetooth de Classe 1. Alcance de até 100 metros com potência máxima de transmissão de 100mW (20 dBm).

- Enhanced Data Rates (EDR), que utiliza diferentes combinações de modulação para alcançar throughput de 2 a 3 Mbps.

- Firmware iWRAP para o controle da tecnologia Bluetooth.

- Freqüência de 2.4 GHz.

- Tensão de alimentação de 3,3 V.

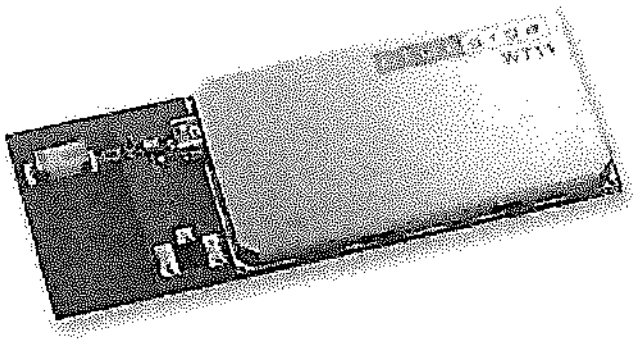


Figura 34 - O Módulo Bluegiga WT11

O ATmega 168 é um microcontrolador de 8 bits fabricado pela ATMEL. Apresenta como características principais:

- Memória de programação Flash de 16 Kbytes.

- Memória EEPROM de 0,5 Kbyte.

- Memória SRAM de 1,0 Kbyte.

- Freqüência máxima de 20 MHz.

- Tensão de Alimentação de 5,0 V.

Optou-se pela utilização da placa Arduino BT, pois se trata de uma placa já confeccionada que possui todo o hardware necessário para a comunicação, além do necessário para a programação. Constatou-se que a confecção de uma placa a partir dos componentes separadamente tomaria muito tempo com um trabalho apenas manual, desperdiçando tempo que poderia ser utilizado em etapas mais importantes como a idealização de um protocolo de comunicação, por exemplo.

4.6.2 Alimentação do Módulo de Dados

O módulo de dados deve ser alimentado idealmente por 3,3 V.

Para isso, são utilizadas duas pilhas AA que, em suas melhores condições totalizam pouco mais de 3,0 V em série, sendo suficientes para o funcionamento do dispositivo.

A própria placa Arduino tem um conversor DC/DC integrado que mantém a tensão em 5,0 V para o funcionamento do microcontrolador ATMEL e em 3,3 V para o funcionamento do módulo Bluetooth.

4.6.3 O Adaptador Bluetooth USB

De modo a habilitar o dispositivo utilizado como módulo de usuário a comunicar-se via Bluetooth, foi adquirido um mini adaptador Bluetooth Dongle 2.0, que foi ligado via porta serial USB ao laptop utilizado.

Este adaptador funciona de maneira *Plug and Play* e é compatível com o Windows Vista, que foi o principal sistema operacional utilizado para os testes com o protótipo.

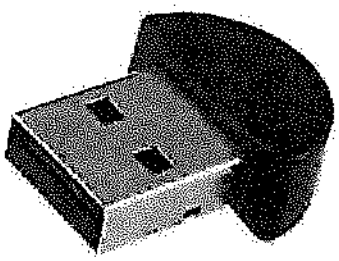


Figura 35 - Mini Adaptador Bluetooth USB

Algumas características do dispositivo são:

- Bluetooth Classe 3. Alcance de até 6 metros, com potência máxima de 1 mW (0 dBm).

- Compatível com Bluetooth versão 1 e versão 2 + EDR.

- Freqüência 2.4 GHz.

Esse adaptador foi escolhido, pois atende a todos os requisitos do projeto. Seu alcance máximo de 6 metros é ideal porque a interferência no sinal dos módulos de dados situados no leito de outros pacientes é, obviamente, indesejada. Seu tamanho reduzido também foi um fator importante. Porém, o seu preço unitário de R$ 15,00 foi um fator muito atraente para a escolha.

4.6.4 Compatibilidade entre os Módulos

Observa-se nas especificações dos componentes que o adaptador Bluetooth funciona com Bluetooth de Classe 3 enquanto o módulo Bluegiga WT11 funciona com Bluetooth de Classe 1.

A diferença básica entre as classe é a potência máxima de transmissão de cada uma. Enquanto a da Classe 1 é de até 100 mW, a da classe 3 chega até somente 1 mW.

Isso implica distância máxima de transmissão de até 100 metros para a primeira opção e de até 6 metros para a segunda.

Ao efetuar a comunicação entre dispositivos de classe diferente, prevalecem as especificações do de menor potência, ou seja. Ao utilizar um componente de classe 1 e outro de classe 3 para comunicação entre si, prevalecem as especificações da classe 3. Ou seja, a distância máxima de até 6 metros e até 1 mW de potência de transmissão.

4.6.5 A Conexão entre os Módulos

Para se efetuar a conexão entre os dispositivos de hardware, é necessário conectar o adaptador Bluetooth a uma entrada USB do laptop, que é imediatamente reconhecida pelo sistema operacional, no caso deste projeto, o Windows Vista.

Após o reconhecimento do adaptador, alimenta-se a placa Arduino BT e deve-se procurar por dispositivos Bluetooth através do computador para a placa ser automaticamente detectada. Deve-se então configurar uma conexão serial entre os hardwares, entrando com o pass code pedido pelo sistema operacional para conectar-se ao dispositivo. Esse pass code é padrão da placa e tem valor **12345**.

Por fim, após a inserção do pass code, o próprio sistema operacional designa duas portas de comunicação serial do sistema, as **COM**. Uma porta é a de entrada, utilizada para a programação do dispositivo e a outra porta de saída, utilizada para a comunicação com o hardware.

4.6.6 Problemas Encontrados

Entre os problemas encontrados, destacam-se:

- Problemas de conexão entre o Sistema Operacional e a placa Arduino BT:

Constatou-se que caso houvesse algum problema na programação do microcontrolador da placa, dependendo de sua gravidade, poderia afetar seriamente a configuração automática feita pelo Sistema Operacional ao detectá-la. Para solucionar esse problema, nos casos em que houve problemas na programação foi necessário resetar a placa imediatamente antes de efetuar a configuração com o sistema operacional.

- Problemas de detecção do dispositivo pelo laptop:

Neste caso, quando da utilização do próprio transceptor Bluetooth do laptop, não foi possível detectar a placa Arduino BT. Para solucionar tal problema, utilizou-se o adaptador Bluetooth USB anteriormente mencionado.

Constatou-se que o problema encontrado foi do próprio hardware com computador e nada relacionado à placa. Portanto, o erro relacionado dificilmente ocorreria em outro dispositivo habilitado para se comunicar através da tecnologia Bluetooth.

5. TESTES

5.1 INTERFACE

O funcionamento da Interface pode ser observado pelas figuras a seguir.

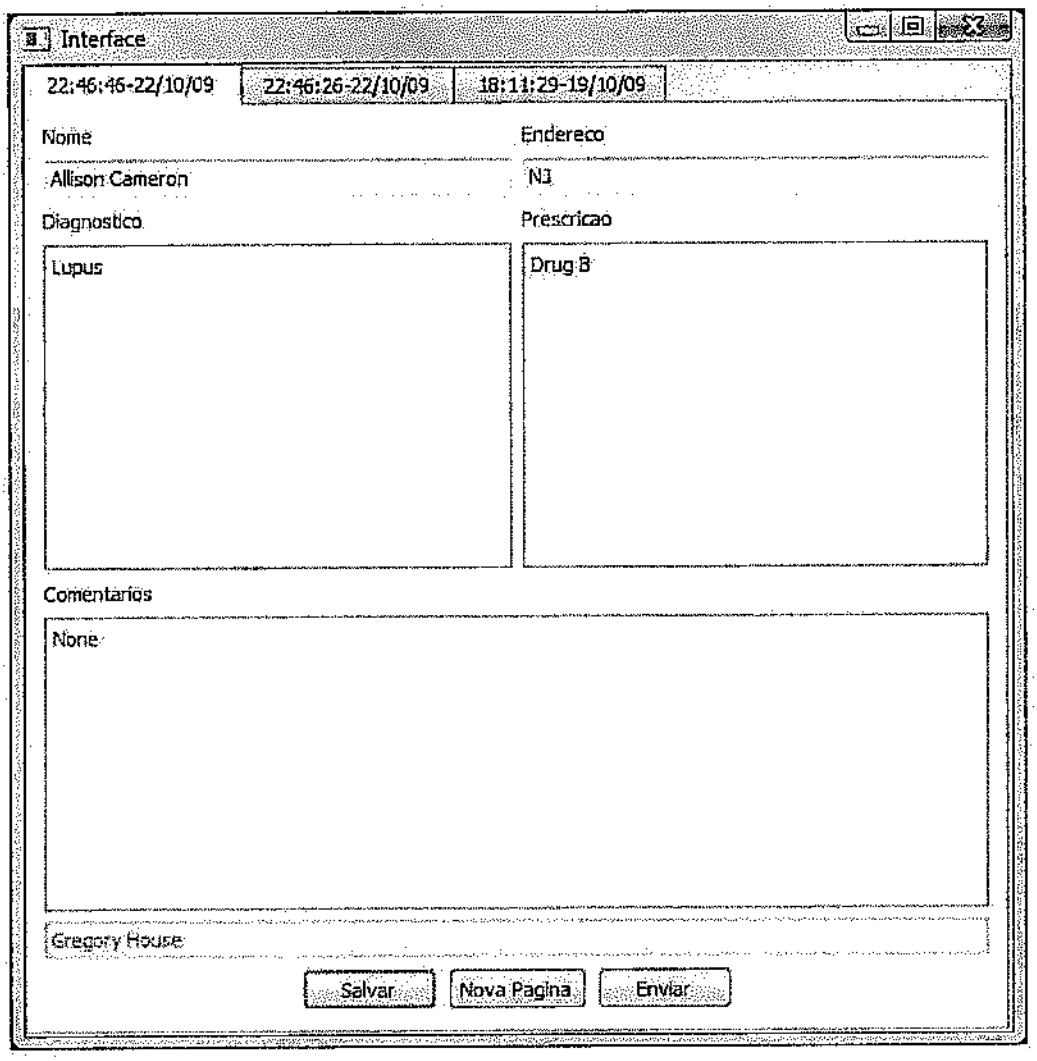
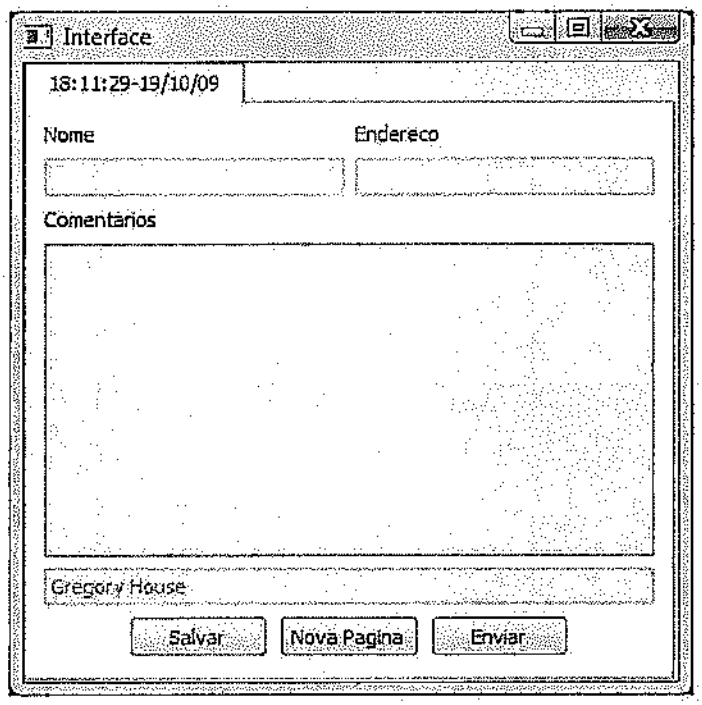


Figura 36 - Interface vista no Windows Vista.



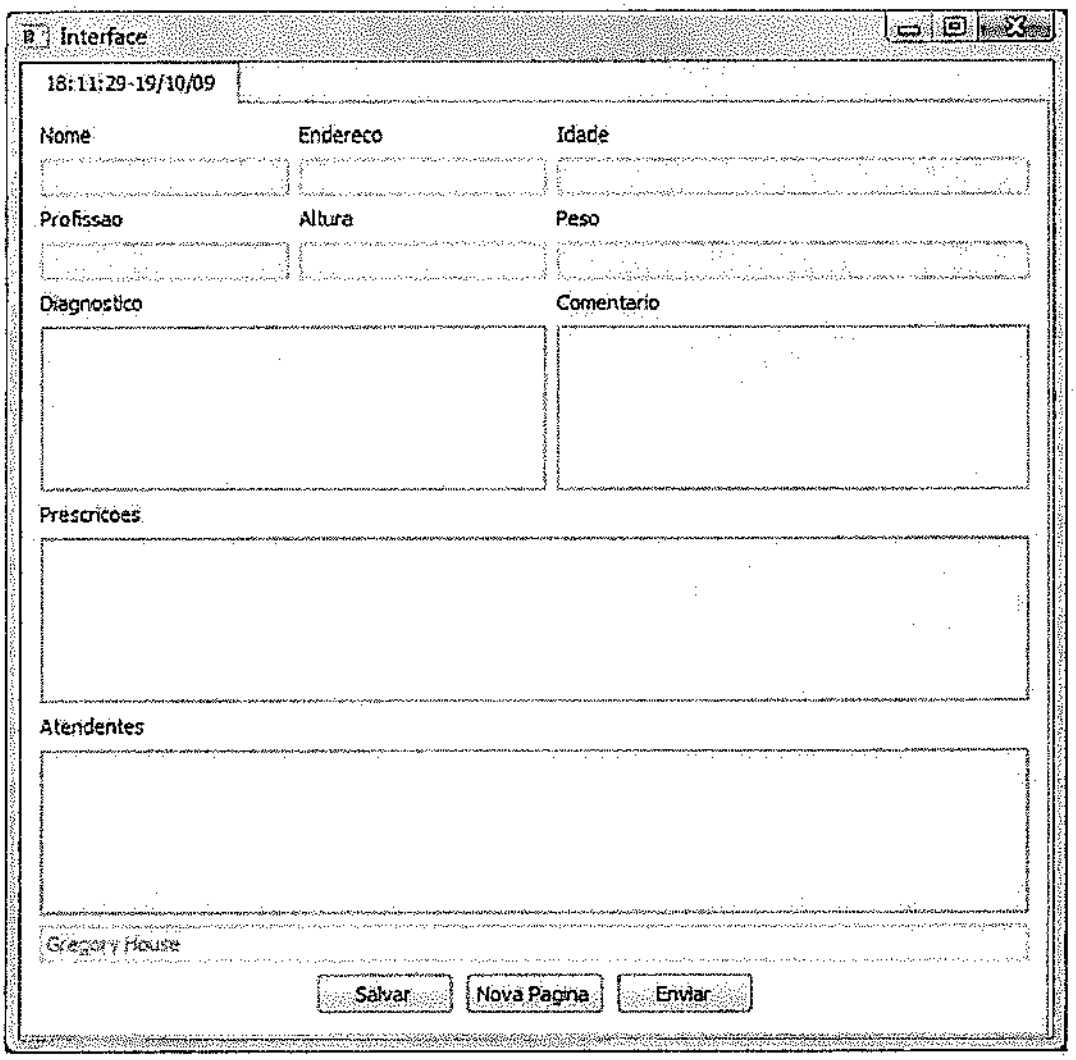


Figura 37 – Duas versões de layout configurados pelo usuário, mostrando a flexibilidade do sistema.

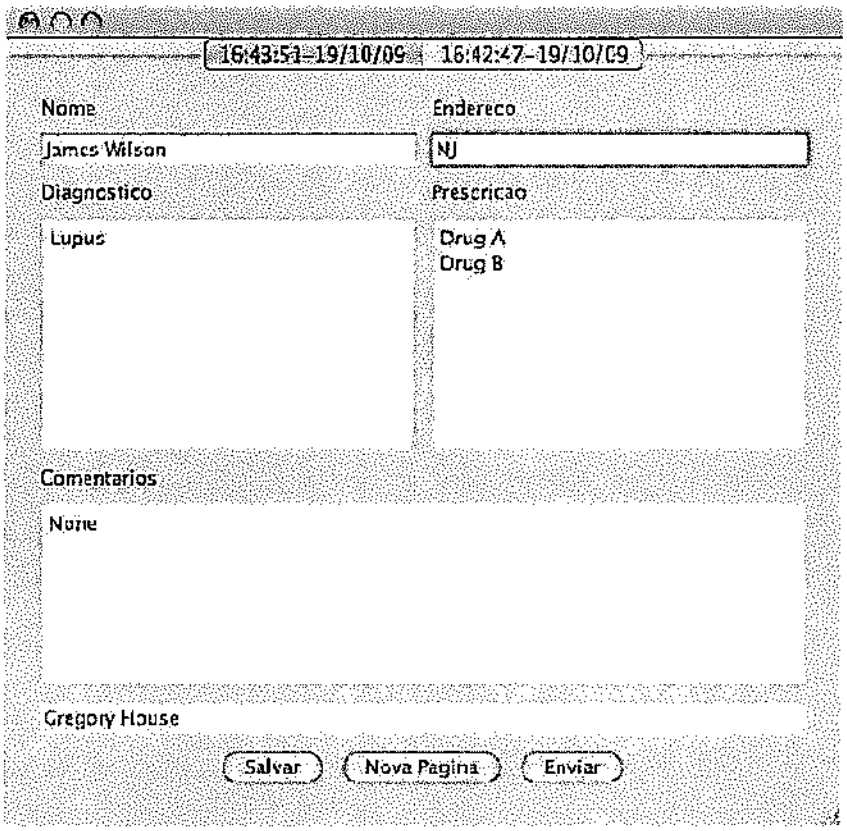


Figura 38 - Interface vista no Mac OS X, consistente com o visual do sistema operacional, mostrando que a Interface é multiplataforma.

Nas figuras pode ser observada a presença de diversas abas, cujos nomes são datas. Conforme explicado anteriormente, cada aba representa uma página do prontuário.

É possível observar também a presença de diversos campos de texto e botões. É através deles que o usuário observa e modifica as informações do prontuário e manipula o sistema. Existe também um campo de texto especial no canto inferior da tela, contendo o nome do usuário que criou a aba sendo observada.

Além disso, são mostradas abas com diferentes layouts. Eles mostram a capacidade do sistema de interpretar o layout desejado pelo usuário através do arquivo de configuração "layout.txt".

Finalmente, pode ser notado que foram mostradas figuras de sistemas operacionais diferentes: Windows Vista e Mac OS X. Essas figuras comprovam que a Interface é multiplataforma e tem um visual consistente com o sistema operacional onde é executada.

5.2 CRIPTOGRAFIA E COMPRESSÃO

Para demonstrar o funcionamento dos programas de criptografia e de compressão, serão utilizados como exemplos arquivos para demonstrar a diferença do conteúdo de um arquivo não criptografado e o mesmo arquivo criptografado e diferença de tamanho entre um arquivo compresso e outro não compresso.

O arquivo não criptografado utilizado foi o arquivo de texto teste.txt, cujo conteúdo pode ser visto abaixo:

Ouviram do Ipiranga as margens plácidas
De um povo heróico o brado retumbante,
E o sol da Liberdade, em raios fúlgidos,
Brilhou no céu da Pátria nesse instante.

Se o penhor dessa igualdade
Conseguimos conquistar com braço forte,
Em teu seio, ó Liberdade,
Desafia o nosso peito a própria morte!

Ó Pátria amada,
Idolatrada,
Salve! Salve!

Brasil, um sonho intenso, um raio vívido,
De amor e de esperança à terra desce,
Se em teu formoso céu, risonho e límpido,
A imagem do Cruzeiro resplandece.

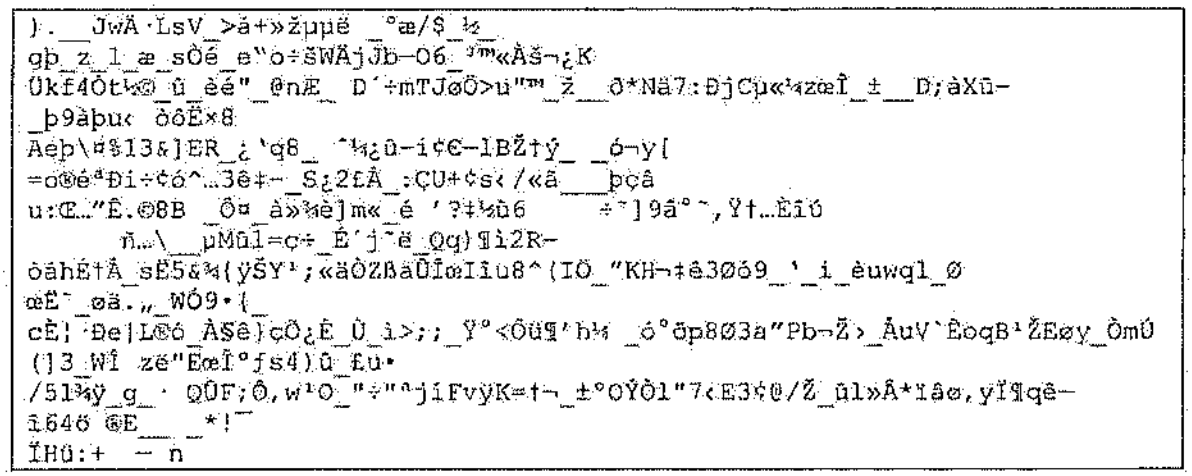
Gigante pela própria natureza,
És belo, és forte, impávido colosso,
E o teu futuro espelha essa grandeza.

Terra adorada
Entre outras mil
És tu, Brasil,
Ó Pátria amada!

Dos filhos deste solo
És mãe gentil,
Pátria amada,
Brasil!

O conteúdo do arquivo após sua criptografia pode ser observado abaixo:

```
…___-3Q]_5 ___Ž0«_Ò_-°¡bš:_Æë°Ä_we 7±T°TY£€ I¾«í› \öãÆŽ(_q_ñOmüc;~
ßüÄLÚ ÊÁ¾__^Vãù_ì\†½¿__™_ó_˜y|öÛ'Wä_[ìÓßMG¡KÛ
«/?ý_»2y¡¨¤_Ò ,°J&'¬SMcÜœz9_B_þ>vhÇÍì_þÒRc___ÉŸ„#¦Ãsœ_ÆMÃ÷ÙãÇ_|ÐE¼
  ë‹>  xì>R»c   pH  Vpê ;=`òqz,ç±ßÅZÀâ Âìì1ñè>PæÐ £—
```



Portanto, a partir do exemplo observado, conclui-se a funcionalidade do programa de criptografia utilizado.

Para a demonstração da funcionalidade do programa de compressão, foi compresso o diretório Teste contendo o arquivo monografiav2.0.doc, que é a monografia entregue ao fim da fase de planejamento do projeto de formatura.

O tamanho do arquivo de (1.210.368 bytes) pode ser observado na figura abaixo:

```
29/11/2009  23:17    <DIR>          .
29/11/2009  23:17    <DIR>          ..
06/10/2009  19:49         1.210.368 monografiav2.0.doc
               1 arquivo(s)      1.210.368 bytes
               2 pasta(s)   50.516.377.600 bytes disponíveis
```

Figura 39 - Tamanho do arquivo original

Após a compressão, pode-se observar diminuição considerável no tamanho do diretório compresso teste.zip (1.049.487 bytes):

```
29/11/2009  23:17    <DIR>          Teste
29/11/2009  23:29         1.049.487 teste.zip
29/08/2009  11:56           167.936 unzip.exe
11/03/2005  23:11           135.168 zip.exe
               6 arquivo(s)      1.833.582 bytes
               3 pasta(s)   50.516.967.424 bytes disponíveis
```

Figura 40 - Tamanho da pasta compressa

Portanto, com os testes efetuados, verificam-se as funcionalidades dos algoritmos de criptografia e de compressão utilizados.

5.3 AUTENTICAÇÃO

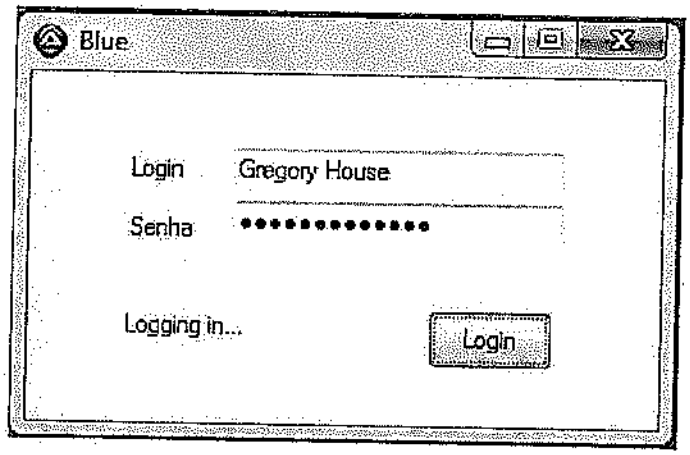


Figura 41 - Janela de Autenticação.

A figura mostrada acima representa a janela de autenticação, onde o usuário realiza o login no sistema. Nela pode ser observado um campo de texto superior, onde o nome de usuário deve ser digitado, e um campo de texto inferior , onde deve ser escrita a senha do usuário.

O usuário deve pressionar, então, o único botão presente na tela para que a sua autenticação seja realizada. Caso o usuário seja autenticado, a janela de autenticação é fechada e a Interface é aberta. Caso contrário, é mostrada uma mensagem de erro e é permitido que o usuário tente novamente após uma espera de 3 segundos.

5.4 COMUNICAÇÃO SEM FIOS

Os resultados demonstrados a seguir comprovam o funcionamento da comunicação sem fios via Bluetooth entre os módulo de usuário e de dados como um todo, utilizando o protocolo anteriormente descrito.

Logo no início da execução do programa, é enviado o sinal de load para o módulo de dados, que inicia a transmissão do arquivo para o módulo de usuário.

```
Stable Library
=========================================
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7
Receiving file

> Receiving file from serial...

> Done (772 bytes)

> Writing file to filesystem...

> Done (772 bytes)
Done
```

Figura 42 - Recebendo Arquivos

Ao fim da alteração do prontuário eletrônico, para o envio do arquivo ao módulo de dados é enviado o comando save seguido do arquivo e das palavras delimitadoras.

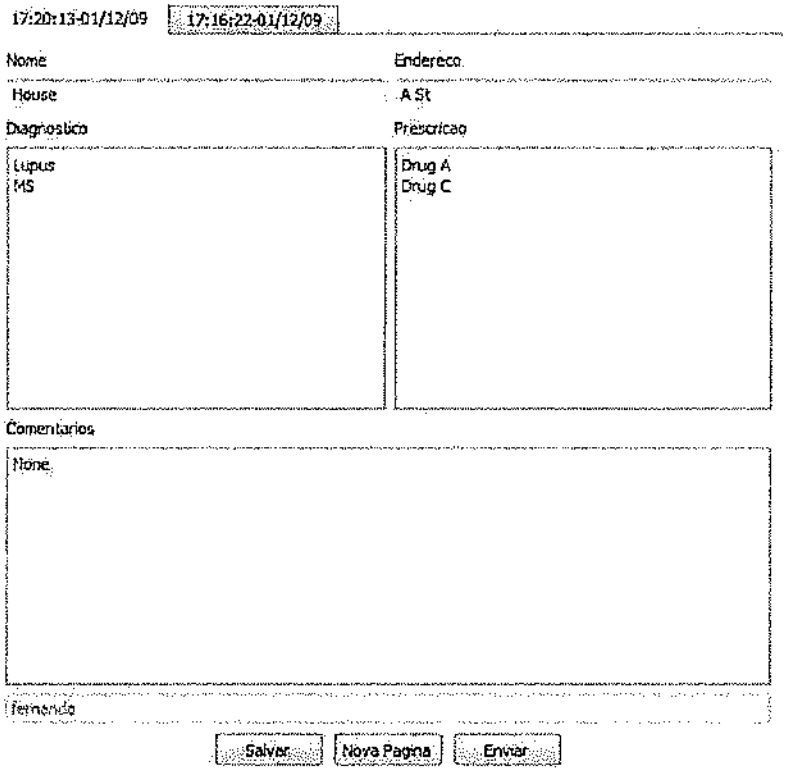


Figura 43 - Efetuando modificações no prontuário

Após o envio, o programa é executado novamente para verificar se o arquivo do módulo de dados contém as modificações feitas anteriormente.

```
Stable Library
=========================================
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7
Sending file

> Reading file from filesystem...

> Done (772 bytes)

> Sending file to serial...
```

Figura 44 - Enviando Arquivos

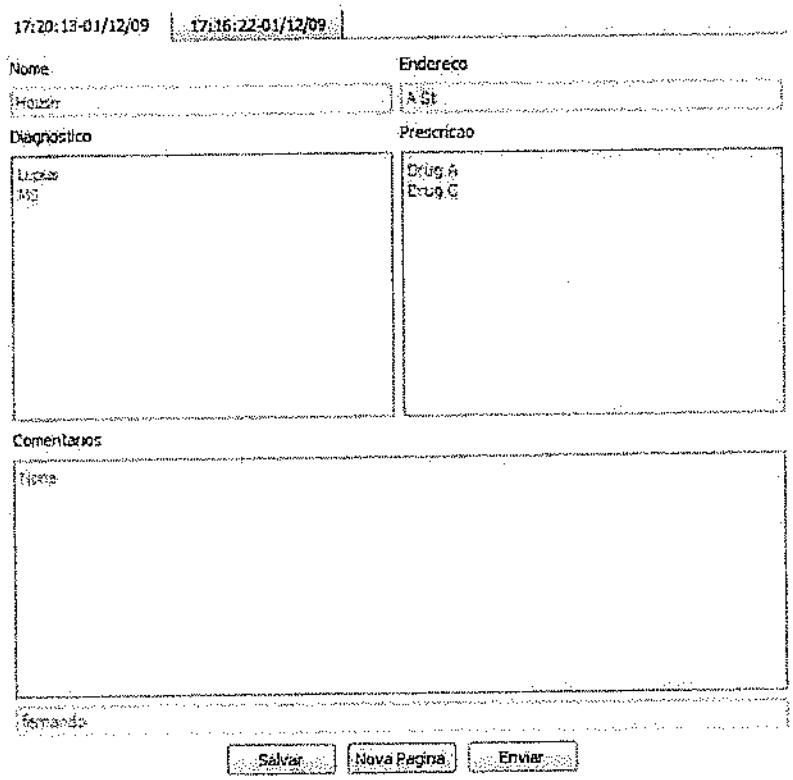


Figura 45 - Modificações Verificadas

Observando a figura acima, conclui-se que o sistema de comunicação sem fios e o protocolo estão funcionando corretamente.

Foi efetuado outro teste de verificação da funcionalidade da transmissão sem fios.

O teste foi efetuado no Laboratório de Microeletrônica. Com o auxílio de um analisador de espectro, foi possível medir e observar a potência do sinal recebido a diferentes distâncias.

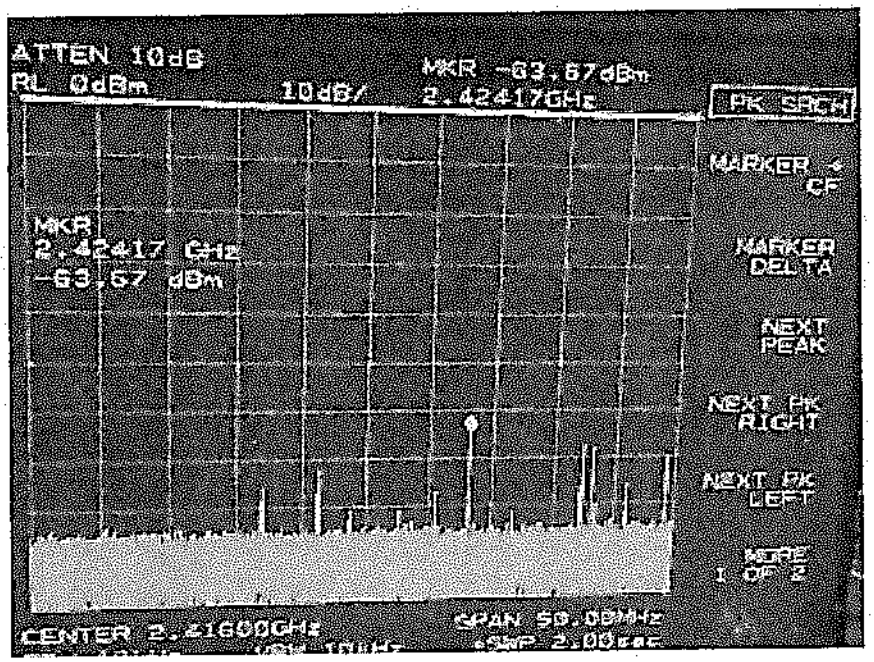


Figura 46 - Tela do Analisador de Espectro

A freqüência da portadora é de 2,424 GHz e as diferenças nas potências medidas em função da distância podem ser observadas na tabela abaixo:

Distância (m)	Potência do Sinal (dBm)
0,05	-59
3,00	-66
5,00	-69

Observa-se, portanto o correto funcionamento da comunicação sem fios.

5.1 INTEGRAÇÃO DO SOFTWARE

O teste de funcionamento da integração dos componentes de software pode ser feito através do uso normal do sistema. Em primeiro lugar, a janela de autenticação mostrada anteriormente é exibida. O usuário digita, então, o seu login e senha e pressiona o botão.

Supondo que essas informações estejam corretas, a janela da Interface é aberta. O seu programa executa automaticamente os componentes de Comunicação Serial do Computador, Criptografia e Compressão para obter os dados do Módulo de Usuário. Essa comunicação é realizada devido ao componente de Comunicação Serial do Arduino no Módulo de Dados, que, junto com o componente de Comunicação Serial do Computador no Módulo de Usuário, implementa o Protocolo de comunicação descrito anteriormente.

O usuário pode então ler e modificar as informações do prontuário pela Interface. Caso ele clique no botão Enviar, as informações do prontuário atualizadas são enviadas de volta ao Módulo de Dados.

Após a realização desses passos é possível comprar que o sistema criado funciona conforme o projeto e atende a todos os seus requisitos e compromissos com o cliente.

5.6 HARDWARE

O procedimento descrito a seguir mostra a funcionalidade do hardware, tanto do módulo de dados quanto do adaptador Bluetooth utilizado pelo módulo de usuário.

Quanto à alimentação, pode-se verificar o seu funcionamento pelo fato de a placa Arduino BT possuir leds que demonstrem a sua operação.

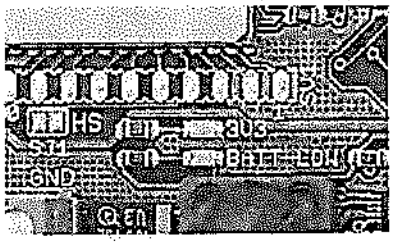


Figura 47 - LEDs do Arduino BT

Já quanto a funcionalidade do módulo de dados, é possível verificá-la através do sistema operacional. Para fazê-lo, basta procurar por dispositivos Bluetooth. Além de verificar a operação da placa ARDUINO BT, observa-se também a operação do adaptador Bluetooth USB.

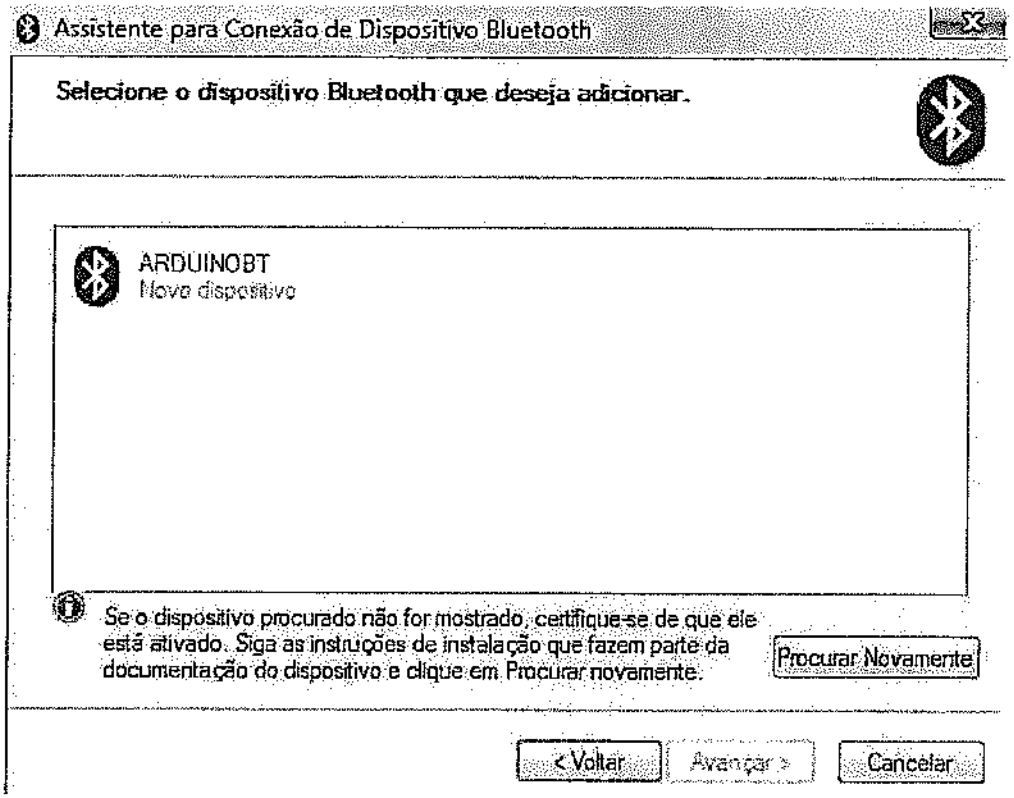


Figura 48 - Detecção da placa ARDUINO BT pelo Windows

Portanto, a partir dos dados apresentados pode-se concluir que o hardware funciona corretamente e da maneira esperada.

6 O RESULTADO

6.1 CUMPRIMENTO DOS COMPROMISSOS

Concluído o projeto, deve-se analisar se todos os compromissos firmados com o cliente foram devidamente cumpridos. Esta é uma das etapas mais importantes, pois demonstra o grau de comprometimento do desenvolvimento da proposta às obrigações documentadas antes do início dos trabalhos.

Caso o resultado final esteja aquém das expectativas do usuário final, decreta-se o fracasso do projeto, desperdiçando meses de trabalho.

6.1.1 Interface com o Usuário

A obrigação firmada era a de desenvolver uma interface gráfica para o módulo de usuário, de maneira a facilitar a visualização e a alteração do prontuário eletrônico.

Essa etapa foi realizada com muita perfeição e detalhamento, pois crê-se que a aparência e a simplicidade sejam de suma importância para a satisfação do usuário final.

Como é possível observar na figura abaixo, a interface desenvolvida é simples e sua utilização é trivial. Basta preencher os campos necessários, salvar as alterações e enviar o arquivo ao módulo de dados.

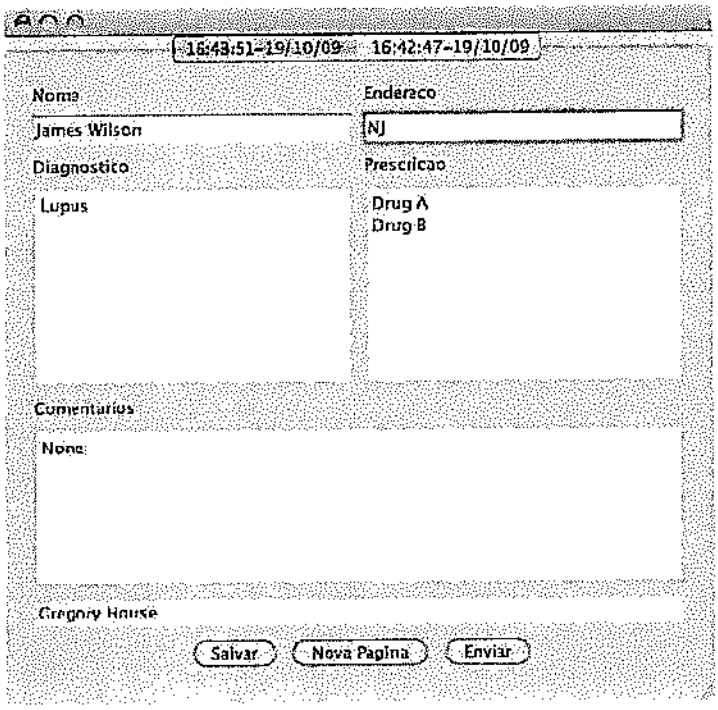


Figura 49 - Interface do Módulo de Usuário

Conclui-se, portanto, que o resultado atingido com essa interface foi favorável e atende ao compromisso firmado anteriormente.

6.1.2 Autenticação

Era esperado o desenvolvimento de um sistema de autenticação seguro, que possibilitasse o acesso aos prontuários somente de usuários autorizados.

Essa tarefa foi concluída logo ao início da fase de desenvolvimento e encontra-se em perfeito estado de funcionamento. Só é possível acessar a interface gráfica com a autenticação do usuário, por meio de login e senha. Além disso, apresenta utilização simples, porém um sistema eficiente por trás de sua operação. Na figura abaixo, pode-se observar a simplicidade de sua utilização pelo usuário leigo.

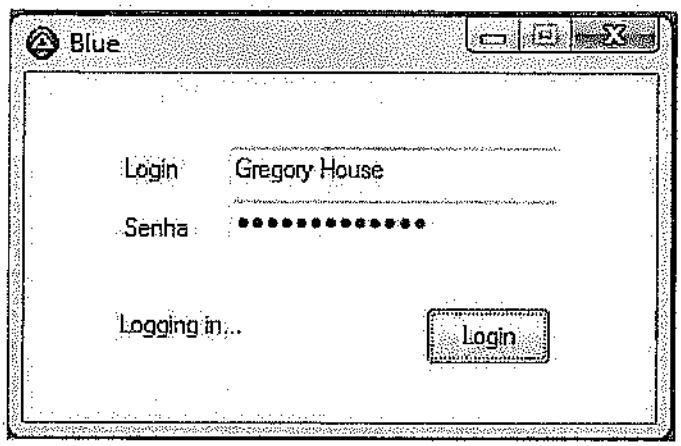


Figura 50 - Tela de Autenticação

Conclui-se, portanto, que a autenticação atendeu aos requisitos esperados documentados.

6.1.3 Criptografia

Com o objetivo de impedir que usuários mal intencionados alterem os dados dos pacientes armazenados nos módulos de dados, era necessária a criação de um sistema de criptografia seguro e robusto.

Optou-se pela utilização do programa de criptografia GPG, que trabalha com chaves de criptografia públicas e privadas. Essa escolha mostrou-se correta, pois o algoritmo utilizado para criptografia é mundialmente famoso por sua eficácia, simplicidade e robustez.

Portanto, conclui-se que o método de criptografia escolhido atende aos requisitos necessários para o correto funcionamento do sistema.

6.1.4 Hardware

Firmou-se o compromisso da criação de um protótipo funcional, com todas as operações devidamente prontas.

O protótipo pronto atende aos principais requisitos de software, que neste caso são: baixo consumo de energia, tamanho reduzido e funções finalizadas e implementadas. Na figura abaixo, é possível observar a comparação de seu tamanho a uma moeda de 25 centavos de dólar.

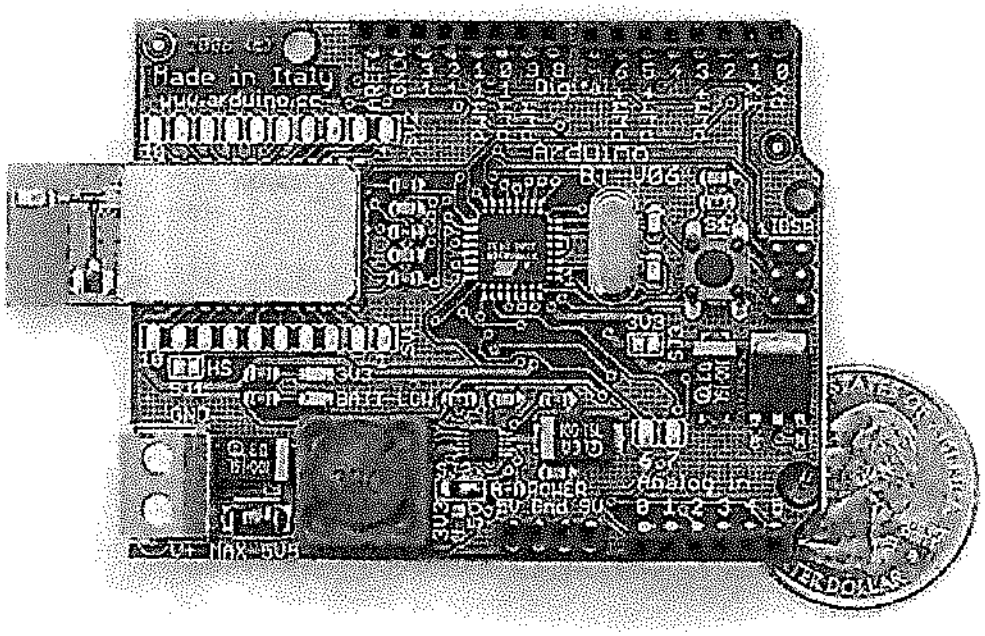


Figura 51 - Comparação do tamanho do Arduino BT

É possível concluir que as especificações de Hardware foram atendidas para a criação deste protótipo. Porém, é importante ressaltar que no caso da utilização em massa pelos hospitais, outras placas mais simples podem ser confeccionadas utilizando tecnologias que reduzam ainda mais o seu tamanho e eliminando componentes desnecessários para o caso final.

6.1.5 Comunicação Sem Fios

O principal diferencial da solução proposta é a possibilidade de comunicação sem fios para a alteração e visualização dos prontuários dos pacientes. Desta forma, é essencial que essa interação esteja funcionando perfeitamente, evitando transtornos para o usuário caso erros apareçam em horas não apropriadas.

Ao final do projeto, consideram-se atingidos os requisitos essenciais para a comunicação sem fios. Logicamente, pequenos erros de comunicação são passíveis de ocorrer. Porém, de acordo com os testes realizados, a freqüência de sua ocorrência é baixa. Assim como a possível ocorrência de interferências, que varia com o ambiente de utilização do dispositivo.

Nos casos de ocorrência de erros, é fácil a sua verificação, bastando somente efetuar a comunicação sem fios novamente para a correção do erro verificado.

6.2 CUMPRIMENTO DO CRONOGRAMA

Ao fim do primeiro semestre de 2009, foi elaborado um cronograma baseado no tempo disponível para a finalização do projeto, na alocação de tempo às tarefas de maior complexidade e na possibilidade de atraso no recebimento de alguns componentes necessários para a realização de outras etapas.

O cronograma elaborado pode ser observado a seguir:

Tabela 5 - Cronograma Planejado

	Julho	Agosto	Setembro	Outubro	Novembro
Interface com o Usuário					
Compra dos Componentes					
Criptografia					
Autenticação					
Montagem do Hardware					
Teste de Comunicação Sem Fio					
Implementação do Algoritmo					
Main do Módulo de Dados					
Main do Módulo de Usuário					
Montagem e Teste do Protótipo					
Relatório Final, Apresentação e Pôster					

Devido a fatores diversos como a pandemia de gripe suína que afetou a cidade durante os meses de inverno e, conseqüentemente, o adiamento do início das aulas do segundo semestre levando ao adiamento do fim das aulas deste mesmo período letivo, e da própria decisão de alteração na alocação do tempo a tarefas diferentes após verificado o seu grau de complexidade, o cronograma realizado apresenta algumas diferenças.

Tabela 6 - Cronograma Realizado

	Julho	Agosto	Setembro	Outubro	Novembro	Dezembro
Interface com o Usuário						
Compra dos Componentes						
Criptografia						
Autenticação						
Montagem do Hardware						
Teste de Comunicação Sem Fio						
Implementação do Algoritmo						
Main do Módulo de Dados						
Main do Módulo de Usuário						
Montagem e Teste do Protótipo						
Relatório Final, Apresentação e Pôster						

Verifica-se pouca diferença entre o cronograma planejado e o realizado. Com mais tempo para a entrega do produto final, foi possível alocar um tempo maior à finalização do protótipo caso ocorressem problemas.

Conclui-se, portanto, que o projeto foi realizado dentro do tempo hábil e o cronograma foi cumprido.

6.3 CUSTOS FINAIS

Estavam previstos custos totais no valor de R$ 640,50. O cálculo dos custos pode ser observado na tabela abaixo:

Tabela 7 - Custos Planejados

Componente	**Preço Unitário**	**Unidades Necessárias**	**Preço Total**
Equipamento Portátil (Módulo de Usuário)	-	1	-
Arduino Bluetooth (Módulo de Dados)	R$ 615,00	1	R$ 615,00
Pilhas AA	R$ 2,00	2	R$ 4,00
Adaptador Bluetooth USB	R$ 15,00	1	R$ 15,00
Suporte para Pilhas AA	R$ 2,00	1	R$ 2,00
Botão On/Off	R$ 5,50	1	R$ 5,50
		Total	R$ 640,50

Como à época da formulação da tabela acima, praticamente todos os componentes já haviam sido adquiridos, não houve diferenças significativas entre os custos planejados e os realizados.

Obviamente, não foram contabilizados custos diversos como o transporte às instalações de trabalho e às lojas de vendas de componentes, etc.

Portanto, pode-se considerar como custo final para a conclusão do projeto o valor de R$ 640,50. Exatamente igual ao valor planejado. Portanto, não houve discrepâncias para evidenciar a falta de controle do orçamento para a realização da proposta.

7 INFRAESTRUTURA

7.1 PROGRAMAS UTILIZADOS

Foram utilizados diversos programas ao longo do desenvolvimento do projeto. Isso ocorreu devido ao número considerável de linguagens de programação e sistemas operacionais utilizados.

SciTe

Editor de código usado para a linguagem de script AutoIt3. Usado em Windows XP, Vista e 7.

Xcode

IDE (Integrated Development Environment) criada pela Apple, utilizada para a edição e compilação de código em C++. Utilizada no sistema operacional Mac OS X.

Qt Creator

IDE da Nokia criada para a edição e compilação de projetos em C++ utilizando o toolkit Qt. Possui ferramentas especiais para facilitar o uso do toolkit, além de um compilador especializado. Multiplataforma.

Eclipse

IDE utilizada para a criação de código em Java. Multiplataforma.

Firefox

Navegador de internet utilizado para pesquisa e comunicação.

Windows XP, Vista e 7

Sistema operacional da Microsoft para PCs.

Mac OS X

Sistema operacional da Apple para Macs.

Git

Software de controle de versão. Utilizado na criação da Interface.

7.3 O REPOSITÓRIO DE ARQUIVOS

Utilizou-se o repositório online de arquivos Dropbox. O serviço permite que os usuários armazenem e sincronizem seus arquivos na rede e entre computadores, além de permitir o compartilhamento entre arquivos e diretórios com outros usuários.

O Dropbox disponibiliza um cliente multi-plataforma que pode ser instalado pelo usuário, permitindo que este atualize qualquer diretório diretamente da interface de seu computador. Cada modificação é imediatamente sincronizada e transferida para todos os usuários que compartilhem a mesma pasta.

O uso do Dropbox mostrou-se muito útil, pois este projeto não é um trabalho individual, e sim formado por dois integrantes. Portanto, seu uso facilitou muito todo o trabalho de compartilhamento de arquivos necessários.

7.4 A COMUNICAÇÃO ONLINE

Utilizou-se o sistema de e-mail Gmail da Google para a comunicação online. O serviço permite o envio e o recebimento de e-mails, além de uma interface fácil de ser utilizada, ótima organização das mensagens além de diversos outros serviços de interação entre os usuários.

O Gmail é um serviço de correio eletrônico grátis patrocinado por anúncios. Atualmente permite mais de 7 gigabytes em armazenamento, aumentando freqüentemente.

O seu uso mostrou-se essencial para o projeto. Desde seu início, muitas tarefas foram realizadas de maneira não presencial e separadamente. Hoje em dia, isso é possível devido à alta tecnologia e um dos fatores que possibilitaram isso foi o uso do Gmail como meio de comunicação.

7.5 ESTRUTURA FÍSICA

Quanto à estrutura física, foram utilizados laboratórios e salas de computadores da Escola Politécnica da Universidade de São Paulo.

A maior parte do desenvolvimento de hardware foi feita na sala C1-01, que é disponibilizada aos alunos para o desenvolvimento de seus projetos. Nesta sala podem

ser utilizados: computadores, osciloscópios, fontes de tensão, ferros de solda, furadora e prensa para confecção de placas de circuito impresso.

A parte de programação e testes de software foi realizada no Centro de Computação Eletrônica da Universidade de São Paulo (CCE), que disponibiliza diversos computadores com acesso à internet para a utilização pelo público uspiano em geral.

A etapa de mensuração da freqüência de transmissão da placa Arduino BT foi feita na sala do Laboratório de Microeletrônica, sob supervisão do professor José Kléber da Cunha Pinto, onde foi utilizado um analisador de espectro.

Utilizou-se material do almoxarifado do departamento para a realização de algumas etapas do desenvolvimento. Entre eles: chaves de fenda, ferros de solda, multímetros e alicates.

8 DESENVOLVIMENTOS FUTUROS E CONSIDERAÇÕES FINAIS

8.1 DESENVOLVIMENTOS FUTUROS

Nessa seção são descritas possíveis melhorias e outras aplicações possíveis para o sistema desenvolvido.

8.1.1 Melhorias

É possível notar diversas melhorias possíveis que, apesar de registradas durante o desenvolvimento do projeto, não puderam ser implementadas por questões de tempo e de dificuldade. Elas são enumeradas a seguir.

- Segurança dos dados do prontuário durante a execução da Interface, possivelmente decifrando-os por demanda. Dessa forma, eles nunca seriam armazenados decifrados.
- Implementação de persistência de diversos Módulos de Dados para armazenamento de prontuários em bases de dados centralizadas.
- Expansão da memória do Módulo de Dados através da adição de um cartão SD ao Arduino.
- Utilização de bibliotecas de compressão no lugar do programa Zip no componente de Compressão.
- Utilização de bibliotecas de criptografia no lugar do programa GPG no componente de Criptografia.
- Criação de logins e senhas mais seguros no componente de Autenticação.
- Nova implementação do componente de Autenticação usando C++, Qt e os componentes criados para o componente Interface.
- Criação de componente para a criação de layout visualmente. Pode ser facilmente implementado devido à estrutura do programa e à forma como o layout é configurado na Interface.
- Melhor visualização das páginas de prontuário na Interface, possivelmente através de uma lista contendo as datas de todas as páginas existentes. Quando o usuário clica em uma entrada da lista, a aba correspondente é aberta.
- Remoção das pequenas porções de código da Interface, Compressão e Criptografia que impedem a portabilidade completa do sistema.
- Recriação do componente de Comunicação Serial no Computador em C++ ao invés de Java, para manter a uniformidade de linguagens no projeto.
- Aumento da velocidade do Módulo de Dados através do uso de um microcontrolador mais rápido que o Atmega168 do Arduino. O tempo atual de

escrita na EEPROM, por exemplo, é de 3.3 ms por escrita, o que torna o envio de dados ao Módulo muito lento.
- Estudo de outras tecnologias, como WiFi, para transmissão de dados no lugar de Bluetooth.

8.1.2 Outras aplicações

O sistema criado não tem nenhuma característica que limita o seu uso a ambientes de tratamento de saúde. Ele pode ser utilizado em qualquer tipo de aplicação onde seja interessante o acesso a informações sem a utilização de fios através de dispositivos móveis.

Com a grande difusão de celulares atualmente, muitos deles com a tecnologia Bluetooth, o sistema projetado tem uma quantidade muito grande de aplicações muito interessantes. Algumas delas são mostradas a seguir.

- Acesso de informações de sistemas onde o contato próximo não é possível.
- Obtenção de informações em pontos turísticos.
- Download de anotações de professores em salas de aula.
- Acesso a cardápios em lanchonetes e restaurantes.
- Leitura de catálogos de livros em bibliotecas.
- Acesso a catálogos de produtos em lojas tanto por vendedores quanto por clientes

.

8.2 CONSIDERAÇÕES

Foi possível concluir o projeto no tempo e com os recursos disponíveis. Porém, alguns problemas tiveram de ser superados para o seu sucesso. Aqui serão expostos alguns dos problemas encontrados, assim como sugestões para a sua melhoria em atividades futuras.

8.2.1 Recursos Disponibilizados

Foi informada, ao início da fase de idealização do projeto a ser realizado, a existência de recursos no valor de dez mil reais para auxílio no desenvolvimento dos projetos da turma de sistemas eletrônicos.

Entretanto, o acesso a tais recursos mostrou-se muito difícil, levando a maioria dos grupos a bancar por conta própria os componentes e demais custos relacionados à confecção de seus protótipos.

Notou-se também que um projeto, favorecido pelos recursos da turma, apresentou enorme atraso no recebimento dos componentes pedidos, o que, com certeza, prejudicou gravemente o andamento da sua fase de desenvolvimento.

Fica como sugestão: a melhor distribuição dos recursos disponíveis para toda a turma, maior facilidade de obtenção de informação a respeito do andamento dos pedidos de recursos feitos e a diminuição das etapas burocráticas para a obtenção destes recursos para a aquisição de equipamentos que, ao final dos projetos, ficarão sob propriedade da universidade.

Crê-se que, com a adoção das sugestões feitas, os projetos futuros apresentarão melhora significativa da qualidade, inovação e menor dificuldade de conclusão, acrescentando positivamente não só aos formandos, mas também à própria Universidade e à disciplina de Projeto de Formatura.

8.2.2 Infraestrutura Disponível

O maior problema de infraestrutura encontrado pela maioria dos grupos de projeto de formatura é a falta de internet nos laboratórios disponibilizados assim como a falta de uma rede sem fios segura e de livre acesso para os estudantes da Engenharia Elétrica.

Muitos problemas encontrados pelos alunos podem ser facilmente resolvidos com a ajuda da rede mundial de computadores. Porém, não há uma rede WiFi no prédio de engenharia elétrica dedicada aos alunos. Deve-se optar por redes livres de outros laboratórios que apresentam instabilidade e falta de segurança nas transmissões.

Além disso, na sala de computadores exclusiva aos estudantes de Sistemas Eletrônicos o acesso à internet é totalmente restrito ao uso do Gmail e de sites da própria Universidade. Muitas manifestações foram feitas pelos alunos no sentido de eliminar tal restrição. Porém, foram todas ignoradas até o momento.

9 REFERÊNCIAS

BLUETOOTH.ORG. Especificação do Bluetooth. Disponível em:
<https://www.bluetooth.org/apps/content/> -.
Acesso em: 29 Nov. 2009.

ARDUINO. Informações gerais sobre o Arduino BT. Disponível em:
<http://www.arduino.cc/en/Guide/ArduinoBT> -.
Acesso em: 29 Nov. 2009.

NOKIA. Site sobre o Qt. Disponível em:

<http://qt.nokia.com/ >
Acesso em: 29 Nov. 2009.

DOFACTORY. Padrão de design Observer. Disponível em:
<http://www.dofactory.com/Patterns/PatternObserver.aspx >
Acesso em: 29 Nov. 2009.

LEPUS. Padrão de design Iterator. Disponível em:
<http://www.lepus.org.uk/ref/companion/Iterator.xml - >
Acesso em: 29 Nov. 2009.

24 BYTES. Padrão de design Command. Disponível em:
<http://www.patterns.24bytes.com/2009/04/command-design-pattern.html>
Acesso em: 29 Nov. 2009.

GOOGLE CODESEARCH. Implementação de crypt. Disponível em:
< http://www.google.com/codesearch/p#118goTAkg2o/usr/src/libc/gen/crypt.c >
Acesso em: 29 Nov. 2009.

RXTX.ORG. Biblioteca RXTX. Disponível em:
< http://rxtx.org/ >
Acesso em: 29 Nov. 2009.

SCINTILLA. Site sobre o SciTe. Disponível em:
< http://www.scintilla.org/SciTE.html>

Acesso em: 29 Nov. 2009.

APPLE. Site sobre Xcode. Disponível em:
< http://developer.apple.com/tools/xcode/>
Acesso em: 29 Nov. 2009.

NOKIA. Site sobre Qt Creator. Disponível em:
< http://qt.nokia.com/products/developer-tools>
Acesso em: 29 Nov. 2009.

ECLIPSE.ORG. Site sobre o Eclipse. Disponível em:
< http://www.eclipse.org/ >
Acesso em: 29 Nov. 2009.

MOZILLA FOUNDATION. Site sobre o Firefox. Disponível em:
< http://www.mozilla.com/en-US/firefox/personal.html>
Acesso em: 29 Nov. 2009.

MICROSOFT. Site sobre o Windows. Disponível em:
< http://www.microsoft.com/en/us/default.aspx >
Acesso em: 29 Nov. 2009.

APPLE. Site sobre o Mac OS X. Disponível em:
< http://www.apple.com/macosx/>
Acesso em: 29 Nov. 2009.

GIT-SCM. Site do Git. Disponível em:
< http://git-scm.com/ >
Acesso em: 29 Nov. 2009.

WORLD HEALTH ORGANIZATION. Electromagnetic Fields. Disponível em:
<http://www.who.int/topics/electromagnetic_fields/en/index.html>.
Acesso em: 28 Nov. 2009.

WIKIPEDIA. Microwave oven. Disponível em:
<http://en.wikipedia.org/wiki/Microwave_oven>
Acesso em: 28 Nov. 2009.

AGÊNCIA NACIONAL DE TELECOMUNICAÇÕES. Anexo à resolução nº 303 de 2 de Julho de 2002. Disponível em: <http://www.anatel.gov.br/Portal/verificaDocumentos/documento.asp?null&filtro=1&documentoPath=biblioteca/resolucao/2002/anexo_res_303_2002.pdf>. Acesso em: 28 Nov. 2009.

MATS K.; SAMSOM W. Evaluation of Bluetooth as a Replacement for Cables in Intensive Care and Surgery. **Anestesia & Analgesia Journal**, v.98, p. 763-767, 2004. Disponível em: <http://www.anesthesia-analgesia.org/cgi/content/full/98/3/763>. Acesso em: 28 Nov. 2009.

GNUPG. The GNU Privacy Guard. Disponível em: <http://www.gnupg.org/>. Acesso em: 7 Nov. 2009.

IETF. OpenPGP. Disponível em: <http://www.ietf.org/rfc/rfc4880.txt>. Acesso em: 7 Nov. 2009.

INFO-ZIP. Zip 3.0 e UnZip 6.0. Disponível em: <http://www.info-zip.org/>. Acesso em: 7 Nov. 2009.

BLUEGIGA. WT11 - Class 1 Bluetooth® 2.1+ EDR Module. Disponível em: <http://www.bluegiga.com/>. Acesso em: 28 Nov. 2009.

WIKIPEDIA. Bluetooth. Disponível em: <http://en.wikipedia.org/wiki/Enhanced_Data_Rate#Bluetooth_2.0>. Acesso em: 28 Nov. 2009.

ATMEL. ATmega168. Disponível em: <http://www.atmel.com/dyn/products/Product_card.asp?part_id=3303>. Acesso em: 28 Nov. 2009.

DROPBOX. Dropbox Features. Disponível em: <https://www.dropbox.com/features>. Acesso em: 29 Nov. 2009.

WIKIPEDIA. Dropbox (storage provider). Disponível em: <http://en.wikipedia.org/wiki/Dropbox_(storage_provider)>. Acesso em: 29 Nov. 2009.

WIKIPEDIA. Gmail. Disponível em: < http://en.wikipedia.org/wiki/Gmail>. Acesso em: 29 Nov. 2009.

APÊNDICE 1 - MANUAL DE UTILIZAÇÃO

1.1 OBJETIVO

O objetivo deste manual é ilustrar a utilização do produto final entregue ao usuário leigo. Foi elaborado de maneira a facilitar o uso das ferramentas criadas para atingir os objetivos desejados pelo usuário final.

1.2 PREMISSAS

Este manual foi elaborado para os usuários que utilizem computadores pessoais como módulo de usuário (laptop ou desktop) com o sistema operacional Windows Vista, com o adaptador Bluetooth USB devidamente funcional.

1.3 UTILIZAÇÃO

Em primeiro lugar, deve-se conectar o módulo de dados ao módulo de usuário. Para isso, com o módulo de dados ligado, deve-se utilizar a ferramenta de detecção de dispositivos Bluetooth do Windows para detectar a placa ARDUINO BT. A figura abaixo ilustra a operação:

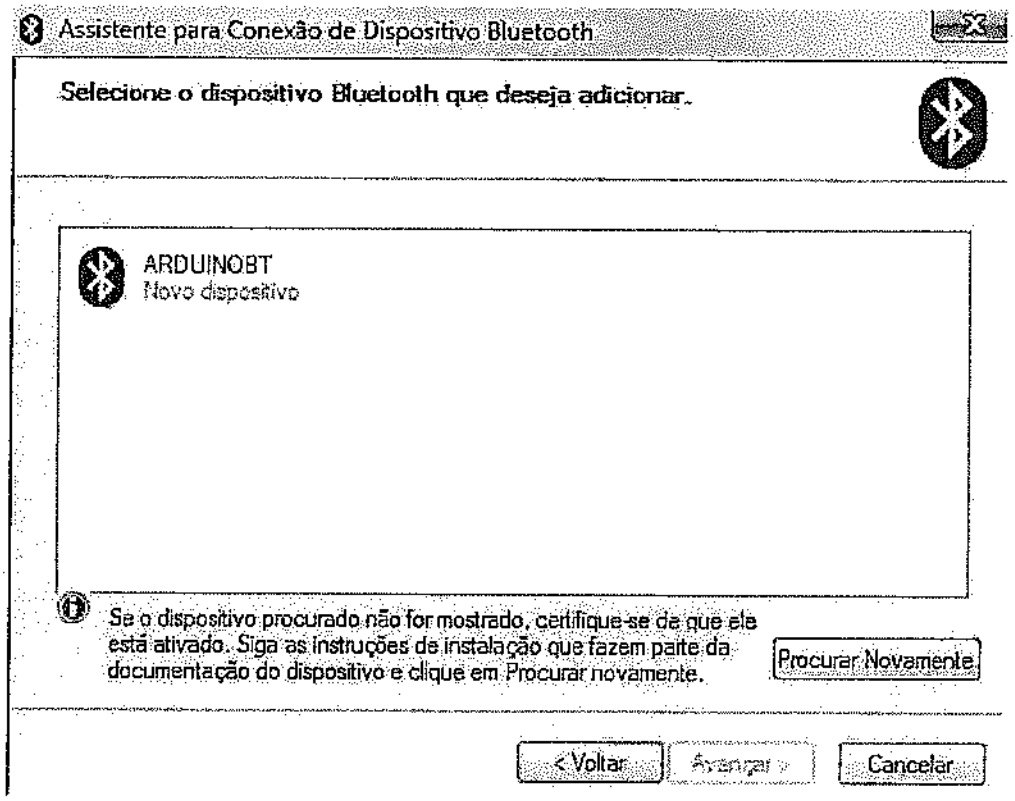


Figura 52 - Detecção do Módulo de Usuário

Após a detecção, insere-se a chave de conexão da placa **12345** e o Windows configurará uma porta de comunicação serial para o módulo de dados.

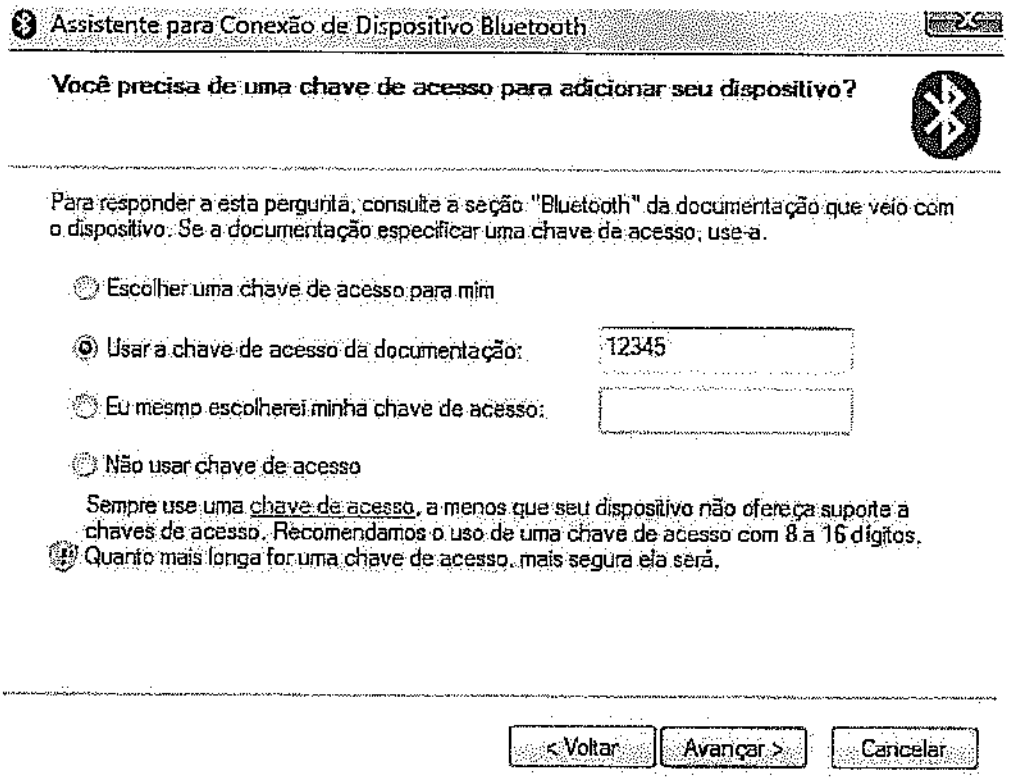


Figura 53 - Inserção da chave de acesso

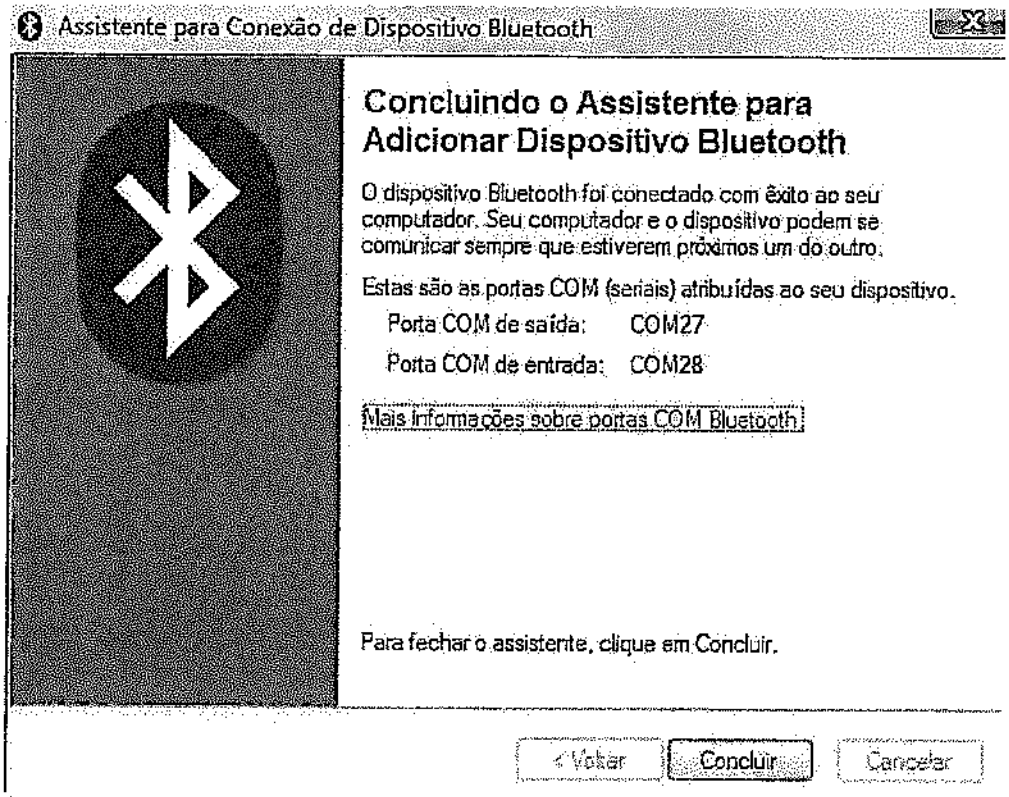


Figura 54 - Portas de comunicação serial criadas

Com a porta serial em mãos, deve-se abrir um arquivo de configuração e inserir essa porta, para que o programa tenha condições de comunicar-se com o módulo de dados através dela.

```
<port>COM27</port>
<newRecord>false</newRecord>
```

Arquivo – config.txt

No arquivo podem ser inseridos parâmetros diferentes dos padrões utilizados pelo programa.

Entre os delimitadores <port> pode ser inserida a porta de comunicação serial criada pelo Sistema Operacional.

Entre os delimitadores <newRecord> pode ser inserida a palavra "true" ou "false", que indica ao programa a intenção de se criar um prontuário novo (caso "true") ou utilizar o prontuário presente no módulo de dados (caso "false").

Após essa interação inicial, inicia-se o programa. Na tela de autenticação, devem ser inseridos o nome de usuário e a senha do profissional cadastrado no sistema. Somente usuários cadastrados podem ter acesso ao prontuário.

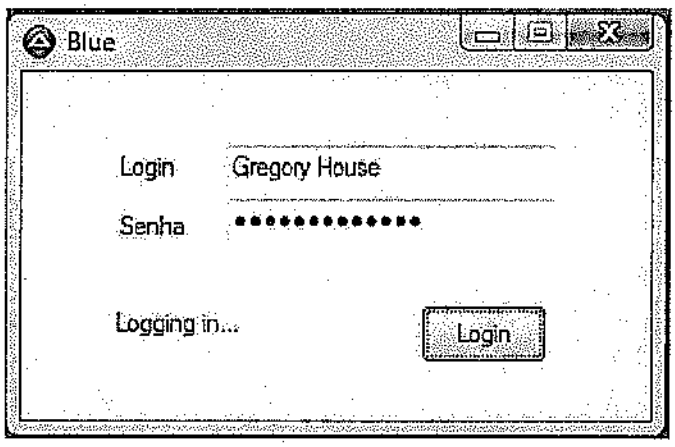


Figura 55 - Tela de Autenticação

Após a autenticação, o módulo de usuário baixa automaticamente o prontuário presente no módulo de dados e apresenta o seu conteúdo na interface do programa.

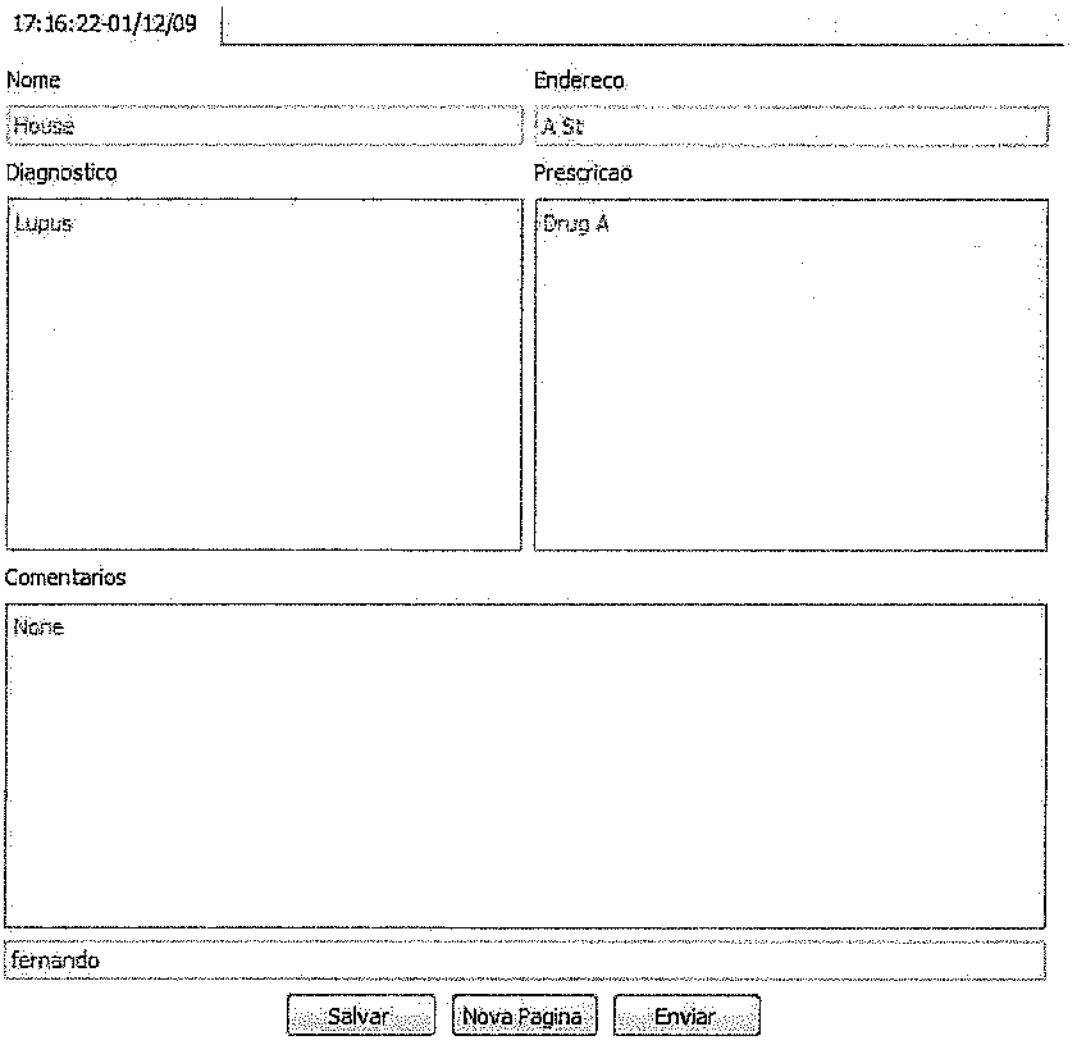


Figura 56 - Conteúdo do Prontuário

Nas abas de cima, podem ser vistas as páginas disponíveis do prontuário e a data e horário de sua criação.

Para adicionar uma nova página do prontuário, basta clicar no botão "Nova Página" e será aberta uma com os campos em branco para o preenchimento. O nome do usuário é automaticamente inserido.

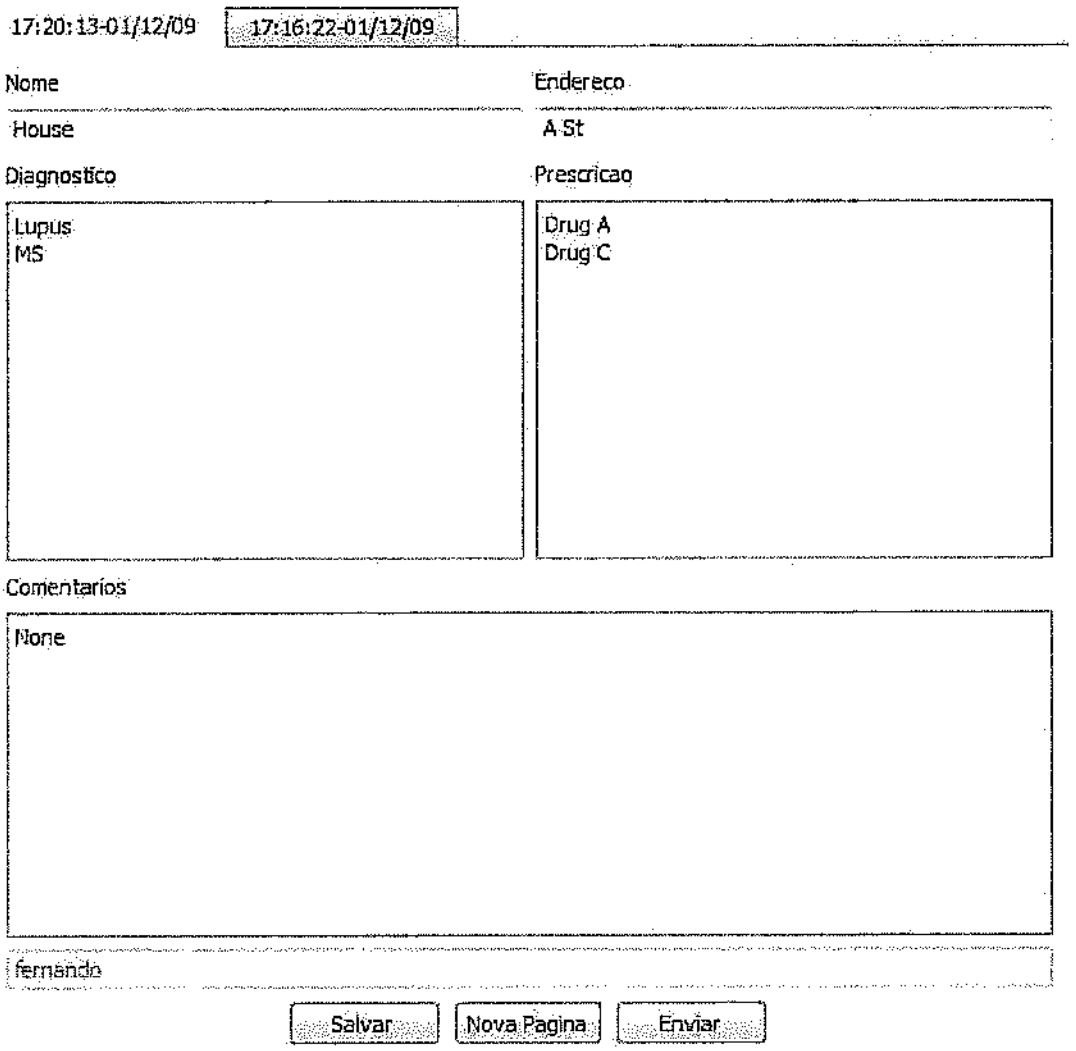


Figura 57 - Página nova com entradas

Após todas as modificações, para gravá-las é necessário clicar no botão salvar. E para enviar o arquivo modificado ao módulo de dados deve-se clicar no botão enviar. O envio pode demorar alguns segundos.

```
Stable Library
=========================================
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7
Sending file

> Reading file from filesystem...

> Done (772 bytes)

> Sending file to serial...
```

Figura 58 - Enviando o arquivo

Finalmente, após o envio do arquivo, fecha-se o programa e a execução está terminada.

APÊNDICE 2 - COMPATIBILIDADE ELETROMAGNÉTICA

2.1 EQUIPAMENTOS MÉDICOS TESTADOS

Encontra-se abaixo a lista de todos os equipamentos médicos testados no estudo de compatibilidade eletromagnética do Bluetooth em ambientes hospitalares publicado no artigo de Mats e Samsom (2004).

Type of equipment	Manufacturer	Name	OP/ICU	Purchasing year	Comments
Anesthesia machine	Dameca	Dameca	OP	–	
	Siemens	Servo 900C	OP	1987	
	Siemens	SC 9000 (KION)	OP	–	
Blood warmer	Biegler	BW 385 L	OP	1994	
Dialyzer	Hospal	Prisma CFM	ICU	1996	
Electro surgery	Valleylab	Force FX	OP	2000	
	Valleylab	SSE 2	OP	–	
Infusion pump	ICUC	P6002	OP	1999	
	Imed	960	OP	1982	
Monitor	Baxter	Vigilance	ICU	–	Cardiac output monitor
	Datex	AS/3	OP	1996	
	Datex	Cardiocap II	OP	1994	
	Datex	Capnomac ultima	OP	1994	ECG monitor
	Datex Engström	Light	ICU	–	
	Hellige	Servomed	OP	–	ECG monitor
	Protocol Systems	ProPaq 102	OP	1997	
Monitor module	Datex		OP	1996	ECG and SpO2
	Datex		OP	1996	gas module
Suction apparatus	Egnell		OP	1992	
Patient heater	Mallickrodt Medical	WarmTouch 5800	OP	1998	
Pulse oximeter	Datex	SatLite Trans	OP	1992	
Ultrasound	Acusan	Sequoia C256	ICU	–	
Ventilator	Siemens	Servo 300	ICU	1993	

Incubator	Dräger	Babytherm 8010	OP	1998	
Electro surgery	Valleylab	Force 10	OP	1993	
	Valleylab	Force 2	OP	–	
Infusion pump	Baxter	FloGuard 6201	ICU	–	
Laparoscopic equipment	Sony	PVM 1443 MD	OP	1991	TV
	Sony	PVM 2043 MD	OP	1992	TV
	Storz	Thermo-flator	OP	2001	Insufflator
	Storz	Xenon 300	OP	2001	Light source
	Storz	Tricam SL PAL	OP	2001	Endoscope
	Wolf	Lapro CO2 Pneu	OP	–	Insufflator
	Wolf	Auto LP	OP	–	Light source
	Wolf	3 CCD Endocam	OP	–	Endoscope
	Olympus	VH I	OP	–	Insufflator
Suction apparatus	Ameda	Master 45	OP	2000	
Radiograph	Picker	PQ 5000	OP	1996	CT
		Stenoscop	OP	–	
	Swemac Medical Appliances	Biplanar 400 E	OP	–	
	Siemens	Mobilett	ICU	–	
Other equipment	Olympus	CLK-4	ICU	–	Light source
	Fischer & Paykel	MR 600 S	ICU	1997	Humidifier
	Dräger	CF 800	ICU	1999	CPAP

OP = Operation Unit. Aparelho utilizado nas salas de operação.
ICU = Intensive Care Unit. Aparelho utilizado nas unidades de terapia intensiva.

APÊNDICE 3 – CÓDIGOS DOS PROGRAMAS

3.1 INTERFACE

3.1.1 Abort.h

```
#ifndef ABORT_H
#define ABORT_H

#include <exception>

class Abort {
public:
        static void execute(){ exit(1); }
};

#endif // ABORT_H
```

3.1.2 ButtonWidget.h

```
#ifndef BUTTONWIDGET_H
#define BUTTONWIDGET_H

#include "Widget.h"

class ButtonWidget : public Widget {
public:
    virtual ~ButtonWidget(){}
};

#endif // BUTTONWIDGET_H
```

3.1.3 Command.h

```
#ifndef COMMAND_H
#define COMMAND_H

class Command
{
public:
    virtual ~Command(){}
    virtual void execute() = 0;
};

#endif // COMMAND_H
```

3.1.4 CopyPageCommand.h

```
#ifndef COPYPAGECOMMAND_H
#define COPYPAGECOMMAND_H


#include "Command.h"
#include "PagedWindow.h"
#include "SaveCommand.h"
#include "LoadCommand.h"
#include "WidgetFactory.h"

#include <iostream>

class CopyPageCommand : public Command {
public:
    ~CopyPageCommand(){ std::cerr << "Deleted CopyPageCommand\n"; }
    CopyPageCommand(PagedWindow* window, WidgetFactory* factory){
_window = window; _factory = factory;}
    void execute() {
        PagedWindowPage* page2 = new PagedWindowPage();
        Command* newpage = new CopyPageCommand(_window, _factory);
        Command* save2 = new SaveCommand(page2->widgetIterator());
        Command* load2 = new LoadCommand(page2->widgetIterator());
        page2->insertWidget(0,0,_factory->createEdit());
        page2->insertWidget(0,1,_factory->createEdit());
        page2->insertWidget(1,0,_factory->createButton("Load",
load2));
        page2->insertWidget(1,1,_factory->createButton("Save",
save2));
        page2->insertWidget(1,2,_factory->createButton("New Page",
newpage));

        static char c = '1';
        std::string s = "eddie";
        _window->insertPage(s + (c++), page2);
    }
private:
    PagedWindow *_window;
    WidgetFactory* _factory;
);

#endif // COPYPAGECOMMAND_H
```

3.1.5 DataAdapter.cpp

```
#include "DataAdapter.h"

void DataAdapter::join(std::string pagesFilename, std::string outputFilename) {
  TreeSaverLoader pages(pagesFilename);
  TreeSaverLoader outputFile(outputFilename);
```

```
    int pageCount = atoi(pages.top().child("count").loadValue().c_str());
    clog << "pageCount: " << pageCount << "\n";

    for(;pageCount > 0; pageCount--) {
        clog << "Reading files\n";
        int count = pageCount - 1;
        std::string filename = pages.top().child(tag("p", count)).child("f").loadValue();
        clog << "Reading file: " << filename << "\n";
        std::ifstream page(filename.c_str());
        char c;
        std::string content = "";
        while(page >> c)
            content += c;
        outputFile.top().createChild(tag("F", count)).child(tag("F", count)).saveValue(filename);
        outputFile.top().createChild(tag("C", count)).child(tag("C", count)).saveValue(content);
        page.close();
    }
    clog << "done\n";
}

void DataAdapter::separate(std::string pagesFolder, std::string pagesFilename, std::string
outputFilename) {
    TreeSaverLoader joinedPages(outputFilename);
    TreeSaverLoader pages(pagesFilename);

    int count;
    for (count = 0;;count++) {
        if (!joinedPages.top().hasChild(tag("F", count))) {
            break;
        }
        std::string filename = joinedPages.top().child(tag("F", count)).loadValue();
        std::string content = joinedPages.top().child(tag("C", count)).loadValue();
        std::ofstream page(filename.c_str());
        page << content;
        page.close();

        pages.top().createChild(tag("p", count)).child(tag("p", count));
        pages.createChild("f").child("f").saveValue(filename);
        pages.top().child(tag("p",
count)).createChild("n").child("n").saveValue(filenameToDate(filename));
    }
    pages.top().createChild("count").child("count").saveValue(tag("", count));
    clog << "done\n";
}

std::string DataAdapter::filenameToDate (std::string fullfilename) {
    int start = fullfilename.find_last_of("//");
    if (start == std::string::npos) {
        start = -1;
    }
    start++;
```

```
    std::ostringstream ofilename;
    for (int i = start; i < fullfilename.length(); i++) {
        ofilename << fullfilename[i];
    }

    std::istringstream ifilename(ofilename.str());

    std::string sec;
    std::string min;
    std::string hour;
    std::string day;
    std::string mon;
    std::string year;
    std::string temp;

    getline(ifilename, temp, '-');
    getline(ifilename, mon, '-');
    getline(ifilename, day, '-');
    getline(ifilename, hour, '-');
    getline(ifilename, min, '-');
    getline(ifilename, sec, '-');

    year += temp[2];
    year += temp[3];

    return hour + ":" + min + ":" + sec + "-" + day + "/" + mon + "/" + year;
}
std::string DataAdapter::tag(std::string c, int n) {
    std::stringstream tag;
    tag << c << n;
    return tag.str();
}
```

3.1.6 DataAdapter.h

```
#ifndef DATAADAPTER_H
#define DATAADAPTER_H

#include <string>
#include <sstream>
#include <fstream>

#include "Log.h"
#include "TreeSaverLoader.h"

class DataAdapter {
public:
    static void join(std::string pagesFilename, std::string
outputFilename);
    static void separate(std::string pagesFolder, std::string
pagesFilename, std::string outputFilename);
private:
```

```
    static std::string filenameToDate (std::string fullfilename);
    static std::string tag(std::string c, int n);
};

#endif // DATAADAPTER_H
```

3.1.7 DataConverter.cpp

```
/*
 *  DataConverter.cpp
 *  Window
 *
 *  Created by Andre Rigon on 10/18/09.
 *  Copyright 2009 __MyCompanyName__. All rights reserved.
 *
 */

#include "DataConverter.h"

void DataConverter::folder(std::string commPort, std::string
pagesFolder, std::string pagesFilename, std::string contentFilename){
    _pagesFolder = pagesFolder;
    _pagesFilename = pagesFilename;
    _contentFilename = contentFilename;
    _commPort = commPort;
}

void DataConverter::load(bool newRecord, bool removeGpg, bool
removeZip, bool removeContentTxt) {
        std::string decryptCommand = "cripto.exe decrypt " +
_contentFilename + ".zip.gpg " + _contentFilename + ".zip";
        std::string unzipCommand = "compressao.exe decompress " +
_contentFilename + ".zip .";
        std::string createPagesFolderCommand = "mkdir " +
_pagesFolder;
        std::string loadCommand = "java -
Djava.library.path=SerialCommLib -jar Arduino.jar load "
                                  + _contentFilename + ".zip.gpg " +
_commPort;

        if (newRecord) {
            system(createPagesFolderCommand.c_str());
            return;
        }

        system(loadCommand.c_str());
      system(decryptCommand.c_str());
        if (removeGpg) deleteGpg();
        system(unzipCommand.c_str());
        if (removeZip) deleteZip();
        system(createPagesFolderCommand.c_str());
        DataAdapter::separate(_pagesFolder, _pagesFilename,
_contentFilename);
        if (removeContentTxt) deleteContentTxt();
}

void DataConverter::save(bool send, bool removePagesFolder, bool
removeZip, bool removeContentTxt, bool removeGpg) {
```

```
        std::string zipCommand = "compressao.exe compress " +
_contentFilename + " " + _contentFilename + ".zip";
        std::string encryptCommand = "cripto.exe encrypt " +
_contentFilename + ".zip " + _contentFilename + ".zip.gpg fernando";
        std::string saveCommand = "java -
Djava.library.path=SerialCommLib -jar Arduino.jar save "
                                  + _contentFilename + ".zip.gpg " +
_commPort;

        DataAdapter::join(_pagesFilename, _contentFilename);
        if (removePagesFolder) deletePagesFolder();
        system(zipCommand.c_str());
        if (removeContentTxt) deleteContentTxt();
      system(encryptCommand.c_str());
        if (removeZip) deleteZip();

        if (send)
            system(saveCommand.c_str());

        if(removeGpg) deleteGpg();
}

void DataConverter::deletePagesFolder() {
    std::string removeFolderCommand = "rd /S /Q " + _pagesFolder;
    system(removeFolderCommand.c_str());
}

void DataConverter::deleteGpg() {
    std::string removeGpgCommand = "del " + _contentFilename +
".zip.gpg";
    system(removeGpgCommand.c_str());
}

void DataConverter::deleteZip() {
    std::string removeZipCommand = "del " + _contentFilename + ".zip";
    system(removeZipCommand.c_str());
}

void DataConverter::deleteContentTxt() {
    std::string removeContentTxtCommand = "del " + _contentFilename;
    system(removeContentTxtCommand.c_str());
}

std::string DataConverter::_commPort = "";
std::string DataConverter::_pagesFolder = "";
std::string DataConverter::_pagesFilename = "";
std::string DataConverter::_contentFilename = "";
```

3.1.8 DataConverter.h

```
/*
 *  DataConverter.h
 *  Window
 *
 *  Created by Andre Rigon on 10/18/09.
 *  Copyright 2009 __MyCompanyName__. All rights reserved.
 *
 */
```

```
#ifndef DataConverter_H
#define DataConverter_H

#include "RunAppCommand.h"
#include "DataAdapter.h"
#include "Abort.h"

#include <string>
#include <fstream>

class DataConverter {
public:
        static void folder(std::string commPort, std::string
pagesFolder, std::string pagesFilename, std::string contentFilename);
        static void load(bool newRecord, bool removeGpg, bool
removeZip, bool removeContentTxt);
        static void save(bool send, bool removeFolder, bool removeZip,
bool removeContentTxt, bool removeGpg);
        static void deletePagesFolder();
        static void deleteGpg();
        static void deleteZip();
        static void deleteContentTxt();
private:
        static std::string _commPort;
        static std::string _pagesFolder;
        static std::string DataConverter::_pagesFilename;
        static std::string DataConverter::_contentFilename;
};


#endif // DataConverter_H
```

3.1.9 EditWidget.h

```
#ifndef EDITWIDGET_H
#define EDITWIDGET_H

#include "Widget.h"

class EditWidget : public Widget {
public:
    virtual ~EditWidget(){}
};

#endif // EDITWIDGET_H
```

3.1.10 LabelWidget.h

```
#ifndef LABELWIDGET_H
#define LABELWIDGET_H

#include "Widget.h"

class LabelWidget : public Widget {
```

```
public:
    virtual ~LabelWidget(){}
};

#endif // LABELWIDGET_H
```

3.1.11 Layout.cpp

```
#include "Layout.h"

Window* Layout::createPage(std::string usernameFilename, std::string
filename, std::string name, WidgetFactory* factory, Command* newpage)
{
      QtLayedOutWindow* page = new QtLayedOutWindow(name);
      TreeSaverLoader tree(filename.c_str());
      int count = atoi(tree.top().child("count").loadValue().c_str());

      int i = 0;
      int maxLine = 0;
      int maxCol = 0;
      while (i < count) {
            std::ostringstream tag;
            tag << "widget" << i;
            Widget* widget = factory-
>createWidget(tree.top().child(tag.str()).loadValue());
            if (tree.top().child(tag.str()).hasChild("x")) {
                  int x =
atoi(tree.top().child(tag.str()).child("x").loadValue().c_str());
                  int y =
atoi(tree.top().child(tag.str()).child("y").loadValue().c_str());
                  page->insertWidget(x, y, widget);
                  if (x > maxLine)
                        maxLine = x;
                  if (y > maxCol)
                        maxCol = y;
            }
            else {
                  int x0 =
atoi(tree.top().child(tag.str()).child("x0").loadValue().c_str());
                  int y0 =
atoi(tree.top().child(tag.str()).child("y0").loadValue().c_str());
                  int x1 =
atoi(tree.top().child(tag.str()).child("x1").loadValue().c_str());
                  int y1 =
atoi(tree.top().child(tag.str()).child("y1").loadValue().c_str());
                  page->insertWidget(x0, y0, x1, y1, widget);
                  if (x1 > maxLine)
                        maxLine = x1;
                  if (y1 > maxCol)
                        maxCol = y1;
            }
            ++i;
      }

      ++maxLine;
      page->insertWidget(maxLine, 0, maxLine, maxCol, factory-
>createUsernameEdit(usernameFilename));
```

```
      std::vector<Widget*>* buttons = new std::vector<Widget*>();
      Command* save = new SaveCommand(page->widgetIterator());
        Command* send = new
SendDataCommand(dynamic_cast<SaveCommand*>(save));
      buttons->push_back(factory->createButton("Salvar", save));
      buttons->push_back(factory->createButton("Nova Pagina",
newpage));
      buttons->push_back(factory->createButton("Enviar", send));

      ++maxLine;
      page->insertButtons(maxLine,0,maxLine,maxCol,buttons);

      return page;
}
```

3.1.12 Layout.h

```
#ifndef LAYOUT_H
#define LAYOUT_H

#include "EditWidget.h"
#include "QtPagedWindow.h"
#include "QtWidgetFactory.h"
#include "QtLayedOutWindow.h"
#include "SaveCommand.h"
#include "LoadCommand.h"
#include "ReadOnlyCommand.h"
#include "SendDataCommand.h"
#include "RunAppCommand.h"

#include <string>
#include <vector>
#include "treeparser.h"

class Layout {
public:
      static Window* createPage(std::string usernameFilename,
std::string filename, std::string name, WidgetFactory* factory,
Command* newpage);
};

#endif // LAYOUT_H
```

3.1.13 LineEditWidget.h

```
#ifndef LINEEDIT_H
#define LINEEDIT_H

#include "Widget.h"

class LineEditWidget : public Widget {
public:
    virtual ~LineEditWidget(){}
};
```

```
#endif // LINEEDIT_H
```

3.1.14 LoadCommand.h

```
#ifndef LOADCOMMAND_H
#define LOADCOMMAND_H

#include "Command.h"
#include "WidgetIterator.h"

#include <iostream>

class LoadCommand : public Command {
public:
    ~LoadCommand(){ delete _it; _it = 0; std::cerr << "Deleted
LoadCommand\n"; }
    LoadCommand(WidgetIterator* it){ _it = it;}
    void execute() {
        _it->begin();
        while(!_it->end()) {
            _it->current()->load();
            ++(*_it);
        }
    }
private:
    WidgetIterator* _it;
};


#endif // LOADCOMMAND_H
```

3.1.15 Log.h

```
#ifndef LOG_H
#define LOG_H

#include <fstream>

class Log {
public:
    Log(std::string filename) {
        stream = new std::ofstream(filename.c_str());
    }
    ~Log() {
        stream->close();
        delete stream;
        stream = 0;
    }
    template<class T> Log& operator<<(T t){
        *stream << t;
        stream->flush();
        return *this;
    }
private:
    std::ofstream* stream;
};

 extern Log clog;

#endif // LOG_H
```

3.1.16 main.cpp

```
#include "EditWidget.h"
#include <QApplication>
#include "QtPagedWindow.h"
#include "QtWidgetFactory.h"
#include "SaveCommand.h"
#include "LoadCommand.h"
#include "ReadOnlyCommand.h"
#include "QtLayedOutWindow.h"
#include "NewPageCommand.h"
#include "Layout.h"
#include "PageCreator.h"
#include "DataConverter.h"
#include "TreeSaverLoader.h"
#include "Log.h"

#include <iostream>
#include <string>
#include <sstream>
#include <fstream>

Log clog("C:\\Users\\Familia\\Desktop\\log.txt");

int main(int argc, char** argv) {
        bool newRecord = false;
        std::string commPort = "COM27";
        std::string contentFilename = "Content\\content.txt";
        std::string usernameFilename = "User\\username.txt";
        std::string layoutFilename = "Layout\\layout.txt";
        std::string pagesFilename = "Pages\\pages.txt";
        std::string pagesFolder = "Pages";

        std::ifstream configFile("Config/config.txt");
        if (configFile.is_open()) {
                clog << "Reading configurations\n";
                configFile.close();
                TreeSaverLoader config("Config\\config.txt");
                if (config.top().hasChild("usernameFile")) {
                        usernameFilename =
config.top().child("usernameFile").loadValue();
                }
                if (config.top().hasChild("layoutFile")) {
                        layoutFilename =
config.top().child("layoutFile").loadValue();
                }
                if (config.top().hasChild("pagesFile")) {
                        pagesFilename =
config.top().child("pagesFile").loadValue();
                }
                if (config.top().hasChild("pagesFolder")) {
                        pagesFolder =
config.top().child("pagesFolder").loadValue();
                }
                if (config.top().hasChild("contentFile")) {
```

```
                        contentFilename =
config.top().child("contentFile").loadValue();
                }
                if (config.top().hasChild("port")) {
                        commPort =
config.top().child("port").loadValue();
                }
                if (config.top().hasChild("newRecord")) {
                        std::string newRecordTmp;
                        newRecordTmp =
config.top().child("newRecord").loadValue();
                        if (newRecordTmp == "true")
                            newRecord = true;
                        else if (newRecordTmp == "false")
                            newRecord = false;
                }
        }


        DataConverter::folder(commPort, pagesFolder, pagesFilename,
contentFilename);
//          if (argv[1] == "new") {
//              newRecord = true;
//          }
        DataConverter::load(newRecord, true, true, true);

        QApplication app(argc, argv);
        QtPagedWindow* window = new QtPagedWindow("window");
        WidgetFactory* factory = new QtWidgetFactory("");

        PageCreator c(usernameFilename, layoutFilename, pagesFilename,
pagesFolder + "/", window, factory);
        c.openPages();
        window->show();

        app.exec();
        DataConverter::save(false, true, true, true, true);
    return 0;
}
```

3.1.17 NewPageCommand.h

```
#ifndef NEWPAGECOMMAND_H
#define NEWPAGECOMMAND_H

#include "Command.h"
#include "PagedWindow.h"

class NewPageCommand : public Command {
//public:
//    NewPageCommand(PagedWindow* window, Creator* creator) :
_window(window), _creator(creator) {}
//    void execute() {
//    }
//private:
//    PagedWindow*  window;
```

```
//    Creator* _creator;
};

#endif // NEWPAGECOMMAND_H
```

3.1.18 PageCreator.cpp

```
#include "PageCreator.h"

PageCreator::PageCreator(std::string usernameFilename, std::string
layoutFilename, std::string pagesFilename, std::string pagesFolder,
PagedWindow* window, WidgetFactory* factory)
      : _usernameFilename(usernameFilename),
_layoutFilename(layoutFilename), _pagesFilename(pagesFilename),
_pagesFolder(pagesFolder), _window(window), _factory(factory),
_newpage(new NewPageCommand(this)) {}

PageCreator::PageInfo* PageCreator::newPageInfo() {
    time_t rawtime;
    struct tm * timeinfo;
    time ( &rawtime );
    timeinfo = localtime ( &rawtime );

    char pagename[201];
    strftime(pagename, 200, "%H:%M:%S-%d/%m/%y", timeinfo);
      char filename[201];
    strftime(filename, 200, "%Y-%m-%d-%H-%M-%S", timeinfo);

      std::string* name = new std::string(pagename);
      std::string* file = new std::string(_pagesFolder);
      file->append(filename);

      return new PageInfo(*name, *file);
}

bool PageCreator::exists(std::string filename) {
      std::ifstream file(filename.c_str());
      bool result = file.is_open();
      file.close();
      return result;
}

void PageCreator::newPage() {
      PageInfo* pageInfo = newPageInfo();
      std::string name = pageInfo->name;
      std::string file = pageInfo->file;
      _factory->file(file);
      ReadOnlyCommand roCom(_window->pageAt(0)->widgetIterator());
      roCom.execute();
      _window->insertPage(Layout::createPage(_usernameFilename,
_layoutFilename, name, _factory, _newpage));

      TreeSaverLoader pages(_pagesFilename);
      std::string scount =
pages.top().child("count").loadValue().c_str();
      int count =
atoi(pages.top().child("count").loadValue().c_str());
```

```
        pages.top().createChild("p" + scount).child("p" + scount);
        pages.createChild("n").child("n").saveValue(name);
        pages.parent().createChild("f")
        .child("f").saveValue(file);

      std::ostringstream aux;
      aux << (++count);
      pages.top().child("count").saveValue(aux.str());
}

void PageCreator::openPages() {
      bool newpage = false;
      if(!exists(_pagesFilename)) {
            PageInfo* pageInfo = newPageInfo();
            std::string name = pageInfo->name;
            std::string file = pageInfo->file;
            TreeSaverLoader tree(_pagesFilename);

      tree.top().createChild("count").child("count").saveValue("1");
                  tree.top().createChild("p0").child("p0");
                  tree.createChild("n").child("n").saveValue(name);

tree.parent().createChild("f").child("f").saveValue(file);
                  newpage = true;
      }

      TreeSaverLoader pages(_pagesFilename);
      int count =
atoi(pages.top().child("count").loadValue().c_str());

      for (int i = 0; i < count; ++i) {
            std::ostringstream tag;
                  tag << "p" << i;
                  std::string name =
pages.top().child(tag.str()).child("n").loadValue();
                  std::string file =
pages.top().child(tag.str()).child("f").loadValue();

            _factory->file(file);
            Window* page = Layout::createPage(_usernameFilename,
_layoutFilename, name, _factory, _newpage);
            LoadCommand load(page->widgetIterator());
            load.execute();
            if (newpage == false || count != 1) {
                  ReadOnlyCommand roCom(page->widgetIterator());
                  roCom.execute();
            }
            _window->insertPage(page);
      }
}
```

3.1.19 PageCreator.h

```
#ifndef PAGECREATOR_H
#define PAGECREATOR_H

#include "QtPagedWindow.h"
#include "QtWidgetFactory.h"
```

```
#include "LoadCommand.h"
#include "ReadOnlyCommand.h"
#include "QtLayedOutWindow.h"
#include "Layout.h"

#include <string>
#include <sstream>
#include <fstream>
#include <ctime>
#include "TreeSaverLoader.h"

class PageCreator {
public:
	PageCreator(std::string usernameFilename, std::string
layoutFilename, std::string pagesFilename, std::string pagesFolder,
PagedWindow* window, WidgetFactory* factory);
	void newPage();
	void openPages();
private:
	class NewPageCommand;
	struct PageInfo;
	bool exists(std::string filename);
	PageInfo* newPageInfo();
	std::string _usernameFilename;
	std::string _layoutFilename;
	std::string _pagesFilename;
	std::string _pagesFolder;
	PagedWindow* _window;
	WidgetFactory* _factory;

	class NewPageCommand : public Command {
	public:
		NewPageCommand(PageCreator* PageCreator) :
_PageCreator(PageCreator){}
		void execute(){ _PageCreator->newPage(); }
	private:
		PageCreator* _PageCreator;
	}* _newpage;

	struct PageInfo {
		PageInfo(std::string _name, std::string _file) :
name(_name), file(_file){}
		std::string name;
		std::string file;
	};
};

#endif // PAGECREATOR_H
```

3.1.20 PagedWindow.h

```
#ifndef PAGEDWINDOW_H
#define PAGEDWINDOW_H

#include "Window.h"

class PagedWindow
{
```

```
public:
    virtual ~PagedWindow(){}
    virtual void insertPage(Window* page) = 0;
      virtual Window* pageAt(int index) = 0;
};

#endif // PAGEDWINDOW_H
```

3.1.21 PageWidgetIterator.cpp

```
#include "PageWidgetIterator.h"

PageWidgetIterator::~PageWidgetIterator() {
    delete _widgetIt; _widgetIt = 0;
    std::cerr << "Deleted PageWidgetIterator\n";
}

PageWidgetIterator::PageWidgetIterator(std::vector<Window*>* tabs){
    _tabs = tabs;
    _tabIt = _tabs->begin();
    _widgetIt = _tabs->front()->widgetIterator();
}

WidgetIterator* PageWidgetIterator::begin(){
    _tabIt = _tabs->begin();
    _widgetIt = _tabs->front()->widgetIterator();
    return this;
}

bool PageWidgetIterator::end(){
    return _widgetIt->end() && _tabIt == _tabs->end();
}

WidgetIterator* PageWidgetIterator::operator++(){
    ++(*_widgetIt);
    if (!_widgetIt->end()) {
        return this;
    }

    ++_tabIt;
    if (_tabIt != _tabs->end()) {
        delete _widgetIt;
        _widgetIt = 0;
        _widgetIt = (*_tabIt)->widgetIterator();
        return this;
    }
}
Widget* PageWidgetIterator::current() {
    if (end()) {
        std::cerr << "tried to access iterator past the end:
PageWidgetIterator::next\n";
        Abort::execute();
    }
    Widget* w = _widgetIt->current();
    return w;
}
```

3.1.22 PageWidgetIterator.h

```
#ifndef PAGEWIDGETITERATOR_H
#define PAGEWIDGETITERATOR_H

#include "WidgetIterator.h"
#include "Window.h"
#include "Widget.h"

#include <vector>
#include <iostream>
#include "Abort.h"

class PageWidgetIterator : public WidgetIterator
{
public:
    ~PageWidgetIterator();
    PageWidgetIterator(std::vector<Window*>* tabs);
    WidgetIterator* begin();
    WidgetIterator* operator++();
    bool end();
    Widget* current();
public:
    std::vector<Window*>* _tabs;
    std::vector<Window*>::iterator _tabIt;
    WidgetIterator* _widgetIt;
};

#endif // PAGEWIDGETITERATOR_H
```

3.1.23 QtButton.cpp

```
#include "QtButton.h"

QtButton::~QtButton() {
    delete _slot; _slot = 0;
    delete _qtWidget; _qtWidget = 0;
    std::cerr << "Deleted QtButton\n";
}

QtButton::QtButton(std::string label, Command* com) {
    _qtWidget = new QPushButton(label.c_str());
    _slot = new SimpleQtSlot(com);
    QObject::connect(_qtWidget, SIGNAL(clicked()), _slot,
SLOT(execute()));
}

QWidget* QtButton::qtWidget() {
    return _qtWidget;
}

void QtButton::readonly(bool ro) {
      _qtWidget->setEnabled(!ro);
}
```

3.1.24 QtButton.h

```
#ifndef QTBUTTON_H
#define QTBUTTON_H

#include "QtWidget.h"
#include "ButtonWidget.h"
#include "Command.h"

#include <QPushButton>
#include <QObject>
#include <iostream>

class SimpleQtSlot;

class QtButton : public ButtonWidget, public QtWidget
{
public:
    ~QtButton();
    QtButton(std::string label, Command* com);
    QWidget* qtWidget();
      void readonly(bool ro);
private:
    QWidget* _qtWidget;
    SimpleQtSlot* _slot;
};

class SimpleQtSlot : public QObject {
    Q_OBJECT
public:
    ~SimpleQtSlot(){ delete _com; _com = 0; std::cerr << "Deleted
SimpleQtSlot\n"; }
    SimpleQtSlot(Command* com){ _com = com; }
public slots:
    void execute() { _com->execute(); }
private:
    Command* _com;
};

#endif // QTBUTTON_H
```

3.1.25 QtEdit.cpp

```
#include "QtEdit.h"

QtEdit::~QtEdit() {
    std::cerr << "Deleted QtEdit\n";
}

QtEdit::QtEdit(std::string name, std::string filename) : _qtWidget(new
QPlainTextEdit()) {
    _name = name;
    _filename = filename;
}

QWidget* QtEdit::qtWidget() {
```

```
    return _qtWidget;
}

void QtEdit::save() {
    TreeSaverLoader t(_filename);
    if (!t.hasChild(_name))
        t.top().createChild(_name).child(_name);
    t.top().child(_name);
    t.saveValue(_qtWidget->toPlainText().toStdString());
}

void QtEdit::load() {
    TreeSaverLoader t(_filename);
    if (t.hasChild(_name)) {
        t.child(_name);
        _qtWidget->setPlainText(t.loadValue().c_str());
    }
}

void QtEdit::readonly(bool ro) {
      _qtWidget->setEnabled(!ro);
}
```

3.1.26 QtEdit.h

```
#ifndef QTEDIT_H
#define QTEDIT_H

#include "EditWidget.h"
#include "QtWidget.h"
#include "TreeSaverLoader.h"

#include <QWidget>
#include <QPlainTextEdit>
#include <string>
#include <iostream>
#include <fstream>

class QtEdit : public EditWidget, public QtWidget
{
public:
    ~QtEdit();
    QtEdit(std::string name, std::string filename);
    QWidget* qtWidget();
    void save();
    void load();
      void readonly(bool ro);
private:
    std::string _name;
    std::string _filename;
    QPlainTextEdit* _qtWidget;
};

#endif // QTEDIT_H
```

3.1.27 QtLabel.cpp

```
#include "QtLabel.h"

QtLabel::~QtLabel() {
	std::cerr << "Deleted QtLabel\n";
}
QtLabel::QtLabel(std::string text, std::string name, std::string
filename) : _name(name), _filename(filename){
	_qtWidget = new QLabel(text.c_str());

}
QWidget* QtLabel::qtWidget() {
	return _qtWidget;
}
```

3.1.28 QtLabel.h

```
#ifndef QTLABEL_H
#define QTLABEL_H

#include "LabelWidget.h"
#include "QtWidget.h"
#include "TreeSaverLoader.h"

#include <QWidget>
#include <QLabel>
#include <string>
#include <iostream>
#include <fstream>

class QtLabel : public LabelWidget, public QtWidget
{
public:
    ~QtLabel();
    QtLabel(std::string text, std::string name, std::string filename);
    QWidget* qtWidget();
private:
    std::string _name;
    std::string _filename;
    QLabel* _qtWidget;
);

#endif // QTLABEL_H
```

3.1.29 QtLayedOutWindow.cpp

```
#include "QtLayedOutWindow.h"

/* QtLayedOutWindow */
QtLayedOutWindow::~QtLayedOutWindow() {
    std::vector<Widget*>::iterator it;
    for (it = _widgets->begin(); it != _widgets->end(); ++it)
```

```
        delete (*it);
    delete _widgets; _widgets = 0;
    delete _gridLayout; _gridLayout = 0;
    delete _page; _page = 0;
    std::cerr << "Deleted QtLayedOutWindow\n";
}

QtLayedOutWindow::QtLayedOutWindow(std::string name) : _name(name) {
    _page = new QWidget;
    _gridLayout = new QGridLayout(_page);
    _widgets = new std::vector<Widget*>();
}

QWidget* QtLayedOutWindow::qtWidget() {
    return _page;
}

void QtLayedOutWindow::insertWidget(int x, int y, Widget* widget) {
    QtWidget* qw = dynamic_cast<QtWidget*>(widget);
    if (qw == 0) {
        std::cerr << "dynamic_cast failed:
QtLayedOutWindow::insertWidget\n";
        Abort::execute();
    }
    _gridLayout->addWidget(qw->qtWidget(), x, y);
    _widgets->push_back(widget);
}

void QtLayedOutWindow::insertWidget(int x0, int y0, int x1, int y1,
Widget* widget){
    QtWidget* qw = dynamic_cast<QtWidget*>(widget);
    if (qw == 0) {
        std::cerr << "dynamic_cast failed:
QtLayedOutWindow::insertWidget\n";
        Abort::execute();
    }
    _gridLayout->addWidget(qw->qtWidget(), x0, y0, x1-x0+1, y1-y0+1);
    _widgets->push_back(widget);
}

void QtLayedOutWindow::insertButtons(int x0, int y0, int x1, int y1,
std::vector<Widget*>* buttons){
      QBoxLayout *layout = new QBoxLayout(QBoxLayout::LeftToRight,
_page);

      std::vector<Widget*>::iterator it;
      for (it = buttons->begin(); it != buttons->end(); ++it) {
            QtWidget* qw = dynamic_cast<QtWidget*>(*it);
            if (qw == 0) {
                  std::cerr << "dynamic_cast failed:
QtLayedOutWindow::insertButtons\n";
                  Abort::execute();
            }
            layout->addWidget(qw->qtWidget());
      }

    _gridLayout->addLayout(layout, x0, y0, x1-x0+1, y1-y0+1,
Qt::AlignHCenter);

      delete buttons;
}
```

```
WidgetIterator* QtLayedOutWindow::widgetIterator() {
    WidgetIterator* it = new VectorWidgetIterator(_widgets);
    return it;
}

void QtLayedOutWindow::show() {
    _page->show();
}
```

3.1.30 QtLayedOutWindow.h

```
#ifndef QTLAYEDOUTWINDOW_H
#define QTLAYEDOUTWINDOW_H

#include "Window.h"
#include "WidgetIterator.h"
#include "VectorWidgetIterator.h"
#include "PagedWindow.h"
#include "QtWidget.h"

#include <QWidget>
#include <QGridLayout>
#include <QBoxLayout>
#include <vector>
#include <exception>
#include <iostream>
#include "Abort.h"

class QtLayedOutWindow : public Window, public QtWidget
{
public:
    ~QtLayedOutWindow();
    QtLayedOutWindow(std::string name);
    void insertWidget(int x, int y, Widget* widget);
      void insertWidget(int x0, int y0, int x1, int y1, Widget*
widget);
      void insertButtons(int x0, int y0, int x1, int y1,
std::vector<Widget*>* buttons);
    WidgetIterator* widgetIterator();
    void show();
    std::string name(){ return _name; }
    QWidget* qtWidget();
private:
    std::string _name;
    QWidget* _page;
    QGridLayout* _gridLayout;
    std::vector<Widget*>* _widgets;
};


#endif // QTLAYEDOUTWINDOW_H
```

3.1.31 QtLineEdit.cpp

```
#include "QtLineEdit.h"

QtLineEdit::~QtLineEdit() {
    std::cerr << "Deleted QtLineEdit\n";
}

QtLineEdit::QtLineEdit(std::string name, std::string filename) :
_qtWidget(new QLineEdit()) {
    _name = name;
    _filename = filename;
}

QWidget* QtLineEdit::qtWidget() {
    return _qtWidget;
}

void QtLineEdit::save() {
    TreeSaverLoader t(_filename);
    if (!t.hasChild(_name))
        t.top().createChild(_name).child(_name);
    t.top().child(_name);
    t.saveValue(_qtWidget->text().toStdString());
}

void QtLineEdit::load() {
    TreeSaverLoader t(_filename);
    if (t.hasChild(_name)) {
        t.child(_name);
        _qtWidget->setText(t.loadValue().c_str());
    }
}

void QtLineEdit::readonly(bool ro) {
      _qtWidget->setEnabled(!ro);
}
```

3.1.32 QtLineEdit.h

```
#ifndef QTLINEEDIT_h
#define QTLINEEDIT_h

#include "LineEditWidget.h"
#include "QtWidget.h"
#include "TreeSaverLoader.h"

#include <QWidget>
#include <QLineEdit>
#include <string>
#include <iostream>
#include <fstream>

class QtLineEdit : public LineEditWidget, public QtWidget
{
public:
    ~QtLineEdit();
    QtLineEdit(std::string name, std::string filename);
    QWidget* qtWidget();
    void save();
```

```
    void load();
      void readonly(bool ro);
private:
    std::string _name;
    std::string _filename;
    QLineEdit* _qtWidget;
};

#endif // QTLINEEDIT_h
```

3.1.33 QtPagedWindow.cpp

```
#include "QtPagedWindow.h"

QtPagedWindow::~QtPagedWindow() {
    std::vector<Window*>::iterator it;
    for (it = _tabs->begin(); it != _tabs->end(); ++it)
        delete (*it);
    delete _tabs; _tabs = 0;
    delete _tabWidget; _tabWidget = 0;
    delete _window; _window = 0;
    std::cerr << "Deleted QtPagedWindow\n";
}

QtPagedWindow::QtPagedWindow(std::string name) : _name(name) {
    _window = new QWidget;
    _tabWidget = new QTabWidget;
      _tabWidget->setUsesScrollButtons(1);
    _tabs = new std::vector<Window*>();
}

void QtPagedWindow::insertPage(Window* page) {
    QtWidget* qw = dynamic_cast<QtWidget*>(page);
    _tabWidget->insertTab(0, qw->qtWidget(), page->name().c_str());
    _tabWidget->setCurrentIndex(0);
    _tabs->push_back(page);
}

Window* QtPagedWindow::pageAt(int index) {
      return _tabs->at(_tabs->size()-1);
}

void QtPagedWindow::show() {
    _tabWidget->show();
}

WidgetIterator* QtPagedWindow::widgetIterator(){
    WidgetIterator* it = new PageWidgetIterator(_tabs);
    return it;
}
```

3.1.34 QtPagedWindow.h

```
#ifndef QTPAGEDWINDOW_H
#define QTPAGEDWINDOW_H
```

```
#include "Window.h"
#include "WidgetIterator.h"
#include "PagedWindow.h"
#include "QtWidget.h"
#include "PageWidgetIterator.h"

#include <QWidget>
#include <QTabWidget>
#include <vector>
#include <exception>
#include <iostream>

class QtPagedWindow : public PagedWindow
{
public:
    ~QtPagedWindow();
    QtPagedWindow(std::string name);
    void insertPage(Window* page);
      Window* pageAt(int index);
    WidgetIterator* widgetIterator();
    void show();
    std::string name(){ return _name; }
private:
    std::string _name;
    QWidget* _window;
    std::vector<Window*>* _tabs;
    QTabWidget* _tabWidget;
};

#endif // QTPAGEDWINDOW_H
```

3.1.35 QtUsernameEdit.cpp

```
/*
 *  QtUsernameEdit.cpp
 *  Window
 *
 *  Created by Andre Rigon on 10/18/09.
 *  Copyright 2009 __MyCompanyName__. All rights reserved.
 *
 */

#include "QtUsernameEdit.h"

QtUsernameEdit::~QtUsernameEdit() {
    std::cerr << "Deleted QtUsernameEdit\n";
}

QtUsernameEdit::QtUsernameEdit(std::string name, std::string filename,
std::string usernameFilename) : _qtWidget(new QLineEdit()) {
     _name = name;
     _filename = filename;
       _usernameFilename = usernameFilename;
       _qtWidget->setEnabled(false);
       load();
}
```

```
QWidget* QtUsernameEdit::qtWidget() {
    return _qtWidget;
}

void QtUsernameEdit::save() {
    TreeSaverLoader t(_filename);
    if (!t.hasChild(_name))
        t.top().createChild(_name).child(_name);
    t.top().child(_name);
    t.saveValue(_qtWidget->text().toStdString());
}

void QtUsernameEdit::load() {
      TreeSaverLoader t(_filename);
    if (t.hasChild(_name)) {
        t.child(_name);
        _qtWidget->setText(t.loadValue().c_str());
            return;
    }

      std::ifstream usernameFile(_usernameFilename.c_str());
      if (usernameFile.is_open()) {
            char username[100];
            usernameFile.getline(username, 99);
            std::string user(username);
            if (user != "") {
                  _qtWidget->setText(user.c_str());
                  usernameFile.close();
                  return;
            }
      }
      usernameFile.close();

      std::cerr << "QtUsernameEdit save: Username not found.\n";
      Abort::execute();
}

void QtUsernameEdit::readonly(bool ro) {}
```

3.1.36 QtUsernameEdit.h

```
/*
 *  QtUsernameEdit.h
 *  Window
 *
 *  Created by Andre Rigon on 10/18/09.
 *  Copyright 2009 __MyCompanyName__. All rights reserved.
 *
 */

#ifndef QtUsernameEdit_H
#define QtUsernameEdit_H

#include "EditWidget.h"
#include "QtWidget.h"
#include "TreeSaverLoader.h"

#include <QWidget>
```

```
#include <QLineEdit>
#include <string>
#include <iostream>
#include <fstream>
#include "Abort.h"

class QtUsernameEdit : public EditWidget, public QtWidget {
public:
    ~QtUsernameEdit();
    QtUsernameEdit(std::string name, std::string filename, std::string
usernameFilename);
    QWidget* qtWidget();
    void save();
    void load();
      void readonly(bool ro);
private:
    std::string _name;
    std::string _filename;
      std::string _usernameFilename;
    QLineEdit* _qtWidget;
};

#endif // QtUsernameEdit_H
```

3.1.37 QtWidget.h

```
#ifndef QTWIDGET_H
#define QTWIDGET_H

#include <QWidget>

class QtWidget
{
public:
    virtual ~QtWidget(){}
    virtual QWidget* qtWidget() = 0;
};

#endif // QTWIDGET_H
```

3.1.38 QtWidgetFactory.cpp

```
#include "QtWidgetFactory.h"

QtWidgetFactory::~QtWidgetFactory() {
    std::cerr << "Deleted QtWidgetFactory\n";
}

QtWidgetFactory::QtWidgetFactory(std::string filename) :
_filename(filename) {}

void QtWidgetFactory::file(std::string filename) {
      _filename = filename;
}
```

```
Widget* QtWidgetFactory::createButton(std::string label, Command* com)
{
    Widget* w = new QtButton(label, com);
    return w;
}

Widget* QtWidgetFactory::createEdit() {
    static int i = 0;
    std::ostringstream name;
    name << "e" <<i++;
    Widget* w = new QtEdit(name.str(), _filename);

    return w;
}

Widget* QtWidgetFactory::createLineEdit() {
    static int i = 0;
    std::ostringstream name;
    name << "l" << i++;
    Widget* w = new QtLineEdit(name.str(), _filename);

    return w;
}

Widget* QtWidgetFactory::createUsernameEdit(std::string
usernameFilename) {
        std::string name = "u";
    Widget* w = new QtUsernameEdit(name, _filename, usernameFilename);

    return w;
}

Widget* QtWidgetFactory::createLabel(std::string text) {
      static int i = 0;
    std::ostringstream name;
    name << "label" << i++;
    Widget* w = new QtLabel(text, name.str(), _filename);
    return w;
}

Widget* QtWidgetFactory::createWidget(std::string properties) {
      Widget* widget;
      TreeParser::Tree tree(properties);
      std::string type = tree.top().child("type").read_value();

      if (type == "edit") {
            widget = createEdit();
      }
      else if (type == "label") {
            widget =
createLabel(tree.top().child("text").read_value());
      }
      else if (type == "lineedit") {
            widget = createLineEdit();
      }

      return widget;
}
```

3.1.39 QtWidgetFactory.h

```
#ifndef QTWIDGETFACTORY_H
#define QTWIDGETFACTORY_H

#include "WidgetIterator.h"
#include "WidgetFactory.h"
#include "QtEdit.h"
#include "QtLineEdit.h"
#include "QtUsernameEdit.h"
#include "QtButton.h"
#include "QtLabel.h"
#include "Widget.h"
#include "TreeSaverLoader.h"

#include <string>
#include <fstream>
#include "treeparser.h"

class QtWidgetFactory : public WidgetFactory
{
public:
    ~QtWidgetFactory();
    QtWidgetFactory(std::string filename);
    Widget* createButton(std::string label, Command* com);
    Widget* createEdit();
      Widget* createLineEdit();
      Widget* createUsernameEdit(std::string usernameFilename);
    Widget* createLabel(std::string text);
      Widget* createWidget(std::string properties);
      void file(std::string filename);
private:
    std::string _filename;
};

#endif // QTWIDGETFACTORY_H
```

3.1.40 ReadOnlyCommand.h

```
#ifndef READONLYCOMMAND_H
#define READONLYCOMMAND_H

#include "Command.h"
#include "WidgetIterator.h"

#include <iostream>

class ReadOnlyCommand : public Command {
public:
    ~ReadOnlyCommand(){ delete _it; _it = 0; std::cerr << "Deleted
ReadOnlyCommand\n"; }
    ReadOnlyCommand(WidgetIterator* it){ _it = it;}
    void execute() {
        _it->begin();
        while(!_it->end()) {
            it->current()->readonly(true);
```

```
            ++(*_it);
        }
    }
private:
    WidgetIterator* _it;
};

#endif // READONLYCOMMAND_H
```

3.1.41 RunAppCommand.h

```
#ifndef RUNAPPCOMMAND_H
#define RUNAPPCOMMAND_H

#include "Command.h"

#include <string>

class RunAppCommand : public Command {
public:
      RunAppCommand(std::string systemCommand) :
_systemCommand(systemCommand){}
      void execute(){ system(_systemCommand.c_str()); }
private:
      std::string _systemCommand;
};

#endif //RUNAPPCOMMAND_H
```

3.1.42 SaveCommand.h

```
#ifndef SAVECOMMAND_H
#define SAVECOMMAND_H

#include "Command.h"
#include "WidgetIterator.h"

#include <iostream>

class SaveCommand : public Command {
public:
    ~SaveCommand(){ delete _it; _it = 0; std::cerr << "Deleted
SaveCommand\n";}
    SaveCommand(WidgetIterator* it){ _it = it; }
    void execute() {
        _it->begin();
        while(!_it->end()) {
            _it->current()->save();
            ++(*_it);
        }
    }
private:
    WidgetIterator* _it;
};
```

```
#endif // SAVECOMMAND_H
```

3.1.43 SaverLoader.h

```
#ifndef SAVERLOADER_H
#define SAVERLOADER_H

#include <string>

class SaverLoader
{
public:
    virtual ~SaverLoader(){}
    virtual SaverLoader& top() = 0;
    virtual SaverLoader& parent() = 0;
    virtual SaverLoader& child(std::string name) = 0;
    virtual SaverLoader& createChild(std::string name) = 0;
    virtual void saveValue(std::string value) = 0;
    virtual std::string loadValue() = 0;
      virtual bool hasChild(std::string name) = 0;
};

#endif // SAVERLOADER_H
```

3.1.44 SendDataCommand.h

```
/*
 *  SendDataCommand.h
 *  Window
 *
 *  Created by Andre Rigon on 10/19/09.
 *  Copyright 2009 __MyCompanyName__. All rights reserved.
 *
 */

#ifndef SendDataCommand_H
#define SendDataCommand_H

#include "Command.h"
#include "DataConverter.h"
#include "SaveCommand.h"

class SendDataCommand : public Command {
public:
    SendDataCommand(SaveCommand* save){_save = save;}
    void execute(){
        _save->execute();
        DataConverter::save(true, false, true, true, true);
    }
private:
    SaveCommand* _save;
};

#endif // SendDataCommand_H
```

3.1.45 treeparser.cpp

```
#include "treeparser.h"

TreeParser::Tree::Tree(std::string tree) {
    _stream = new std::stringstream(tree);
    _stream->seekg (0, std::ios::end);
    int length = _stream->tellg();
    _branch.push_back(new Node(0, length, "root"));
}

TreeParser::Tree::~Tree() {
    delete _stream;
    _stream = 0;
}

TreeParser::Tree& TreeParser::Tree::parent() {
    if (_branch.size() > 1)
        _branch.pop_back();
    return *this;
}

TreeParser::Tree& TreeParser::Tree::top() {
    while (_branch.size() > 1)
        _branch.pop_back();
    return *this;
}

TreeParser::Tree& TreeParser::Tree::child(std::string tag) {
    _stream->clear();
    _stream->seekg(_branch.back()->beg, std::ios::beg);

    int beg = find(begtag(tag), _stream->str(), _branch.back()->beg,
_branch.back()->end) + begtag(tag).size();
    int end = find(endtag(tag), _stream->str(), _branch.back()->beg,
_branch.back()->end);

    if (end < 0 || end >= _branch.back()->end) {
                clog << "TreeParser: A tag '" << tag << "' nÃ£o foi
encontrada dentro da tag '" << _branch.back()->label << "'.\n";
            Abort::execute();
      }

    _branch.push_back(new Node(beg, end, tag));

    return *this;
}

bool TreeParser::Tree::has_child(std::string tag) {
    _stream->clear();
    _stream->seekg(_branch.back()->beg, std::ios::beg);

    int beg = find(begtag(tag), _stream->str(), _branch.back()->beg,
_branch.back()->end) + begtag(tag).size();
    int end = find(endtag(tag), _stream->str(), _branch.back()->beg,
_branch.back()->end);
```

```
    if (end < 0 || end >= _branch.back()->end)
        return false;
    else
        return true;
}

TreeParser::Tree& TreeParser::Tree::create_child(std::string tag) {
    int beg = _branch.back()->beg;
    int end = _branch.front()->end;

    std::string file_content = get_content(beg, end, *_stream);
    file_content.insert(0, begtag(tag) + endtag(tag));

    _stream->seekp(beg, std::ios::beg);
    *_stream << file_content;

    update_branch();
    return *this;
}

std::string TreeParser::Tree::read_value(bool trim) {
    std::string value = get_content(_branch.back()->beg,
_branch.back()->end, *_stream);

    if (trim) {
        int b = 0;
        int e = value.length()-1;
        while (isspace(value[b])) ++b;
        while (isspace(value[e])) --e;

        std::string trimmed;
        while (b <= e)
            trimmed += value[b++];

        return trimmed;
    }

    return value;
}

void TreeParser::Tree::write_value(std::string value) {
    int beg = _branch.back()->beg;
    int end = _branch.back()->end;
    int beg_file = 0;
    int end_file = _branch.front()->end;

    std::string file_content_beg = get_content(beg_file, beg,
*_stream);
    std::string file_content_end = get_content(end, end_file,
*_stream);

    delete _stream;
    _stream = new std::stringstream();

    *_stream << file_content_beg + value + file_content_end;
    update_branch();
}

void TreeParser::Tree::update_branch() {
    std::list<Node*>::iterator it = _branch.begin();
```

```
    _stream->seekg (0, std::ios::end);
    int end = (*it)->end = _stream->tellg();
    int beg = 0;
    for (++it; it != _branch.end(); ++it) {
        std::string label = (*it)->label;
        _stream->seekg (0, std::ios::beg);
        (*it)->beg = find(begtag(label), _stream->str(), beg, end) +
begtag(label).size();
        (*it)->end = find(endtag(label), _stream->str(), beg, end);
        beg = (*it)->beg;
        end = (*it)->end;
    }
}

int TreeParser::find(std::string tag, std::istream& stream) {
    std::string w;
    while (stream >> w && w != tag);

    int pos = stream.tellg();
    return pos < 0 ? -1 : pos - tag.size();
}

int TreeParser::find(std::string tag, std::string s, int beg, int end)
{
    int pos = s.find(tag, beg);
    return (pos == std::string::npos || pos > end) ? -1 : pos;
}

std::string TreeParser::get_content(int beg, int end, std::istream&
stream) {
    std::string buffer;
    stream.seekg(beg, std::ios::beg);

    while (stream.good() && beg++ < end)
        buffer.push_back((char) stream.get());

    return buffer;
}

std::string TreeParser::Tree::print_branch() {
    std::list<Node*>::iterator it;
    std::stringstream result;

    for (it = _branch.begin(); it != _branch.end(); ++it) {
        result << (*it)->label << ":\t" << (*it)->beg << "\t" <<
(*it)->end << "\n";
    }

    return result.str();
}
```

3.1.46 treeparser.h

```
#ifndef TREEPARSER_H
#define TREEPARSER_H

#include <iostream>
#include <fstream>
```

```
#include <sstream>
#include <string>
#include <list>
#include "Abort.h"
#include "Log.h"

namespace TreeParser {
    class TreeParser_Exception{
    public:
        TreeParser_Exception(std::string msg){ clog << msg << "\n"; }
    };

    struct Node {
        Node(int b, int e, std::string l){ beg = b; end = e; label =
l; }
        int beg; // Beginning of content.
        int end; // One after end of content. First character ("<") of
closing tag.
        std::string label;
    };

    class Tree {
    public:
        static const bool TRIM = 1;
        static const bool ERASE = 1;

        // Creation
        Tree(std::string tree);
        ~Tree();

        // Movement
        Tree& parent();
        Tree& child(std::string tag);
        Tree& create_child(std::string tag);
        Tree& top();

        // Value
        std::string read_value(bool trim = 0);
        void write_value(std::string value);
        bool has_child(std::string tag);

        // Output
        std::string to_string(){ return _stream->str(); }
        std::string print_branch();

    private:
        void update_branch();
        std::string begtag(std::string name){ return "<" + name + ">";
}
        std::string endtag(std::string name){ return "</" + name +
">"; }
        std::stringstream *_stream;
        std::list<Node*> _branch;
    };

    // Utility functions
    int find(std::string tag, std::istream& stream);
    int find(std::string tag, std::string s, int beg, int end);
    std::string get_content(int beg, int end, std::istream& stream);
}
```

```
#endif // TREEPARSER_H
```

3.1.47 TreeSaverLoader.cpp

```
#include "TreeSaverLoader.h"

TreeSaverLoader::~TreeSaverLoader() {
    delete _tree;
    _tree = 0;
    clog << "Deleted TreeSaverLoader\n";
}

TreeSaverLoader::TreeSaverLoader(std::string filename) {
    _filename = filename;
    std::ofstream out(filename.c_str(), std::ios::out |
std::ios::app);
    out.close();

    std::ifstream file(filename.c_str());
    if (!file.is_open()) {
        clog << "couldn't open file " << filename << ":
TreeSaverLoader constructor\n";
        Abort::execute();
    }

    std::string content;
    while (file.good()) {
        char c;
        c = file.get();
        if (file.good())
            content.push_back(c);
    }
    file.close();

    _tree = new TreeParser::Tree(content);
}

SaverLoader& TreeSaverLoader::top() {
    _tree->top();
    return *this;
}

SaverLoader& TreeSaverLoader::parent() {
    _tree->parent();
    return *this;
}

SaverLoader& TreeSaverLoader::child(std::string name) {
      _tree->child(name);
      return *this;
}

bool TreeSaverLoader::hasChild(std::string name) {
    return _tree->has_child(name);
}

SaverLoader& TreeSaverLoader::createChild(std::string name) {
     tree->create_child(name);
```

```
    std::ofstream file(_filename.c_str(), std::ios::trunc |
std::ios::out);
    file << _tree->to_string();
    file.close();
    return *this;
}

void TreeSaverLoader::saveValue(std::string value) {
    _tree->write_value(value);
    std::ofstream file(_filename.c_str(), std::ios::trunc |
std::ios::out);
    file << _tree->to_string();
    file.close();
}

std::string TreeSaverLoader::loadValue() {
    return _tree->read_value();
}
```

3.1.48 TreeSaverLoader.h

```
#ifndef TREESAVERLOADER_H
#define TREESAVERLOADER_H

#include "SaverLoader.h"
#include "Log.h"

#include "treeparser.h"
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
#include "Abort.h"

class TreeSaverLoader : public SaverLoader
{
public:
    ~TreeSaverLoader();
    TreeSaverLoader(std::string filename);
    SaverLoader& top();
    SaverLoader& parent();
    bool hasChild(std::string name);
    SaverLoader& child(std::string name);
    SaverLoader& createChild(std::string name);
    void saveValue(std::string value);
    std::string loadValue();
private:
    TreeParser::Tree* _tree;
    std::string _filename;
};

#endif // TREESAVERLOADER_H
```

3.1.49 VectorWidgetIterator.cpp

```
#include "VectorWidgetIterator.h"

VectorWidgetIterator::~VectorWidgetIterator() {
    std::cerr << "Deleted VectorWidgetIterator\n";
}

VectorWidgetIterator::VectorWidgetIterator(std::vector<Widget*>*
children){
    _children = children;
    _it = _children->begin();
}

WidgetIterator* VectorWidgetIterator::begin(){
    _it = _children->begin();
    return this;
}

bool VectorWidgetIterator::end(){
    return _it == _children->end();
}

WidgetIterator* VectorWidgetIterator::operator++(){
    ++_it;
    return this;
}

Widget* VectorWidgetIterator::current() {
    if (_it == _children->end()) {
        std::cerr << "tried to access iterator past the end:
WidgetIterator::next";
        Abort::execute();
    }
    return *_it;
}
```

3.1.50 VectorWidgetIterator.h

```
#ifndef VECTORWIDGETITERATOR_H
#define VECTORWIDGETITERATOR_H

#include "WidgetIterator.h"

#include <vector>
#include <exception>
#include <iostream>
#include "Abort.h"

class VectorWidgetIterator : public WidgetIterator
{
public:
    ~VectorWidgetIterator();
    VectorWidgetIterator(std::vector<Widget*>* children);
    WidgetIterator* begin();
    WidgetIterator* operator++();
    bool end();
    Widget* current();
public:
    std::vector<Widget*>*  children;
```

```
    std::vector<Widget*>::iterator _it;
};


#endif // VECTORWIDGETITERATOR_H
```

3.1.51 Widget.h

```
#ifndef WIDGET_H
#define WIDGET_H

class Widget
{
public:
    virtual ~Widget(){}
    virtual void save(){}
    virtual void load(){}
    virtual void readonly(bool ro){}
};

#endif // WIDGET_H
```

3.1.52 WidgetFactory.h

```
#ifndef WIDGETFACTORY_H
#define WIDGETFACTORY_H

#include "Widget.h"
#include "WidgetIterator.h"
#include "Command.h"

#include <string>

class WidgetFactory
{
public:
    virtual ~WidgetFactory(){}
    virtual Widget* createButton(std::string label, Command* com) = 0;
    virtual Widget* createEdit() = 0;
    virtual Widget* createLineEdit() = 0;
      virtual Widget* createUsernameEdit(std::string usernameFilename)
= 0;
    virtual Widget* createLabel(std::string text) = 0;
      virtual Widget* createWidget(std::string properties) = 0;
      virtual void file(std::string filename) = 0;
};

#endif // WIDGETFACTORY_H
```

3.1.53 WidgetIterator.h

```
#ifndef WIDGETITERATOR_H
#define WIDGETITERATOR_H

#include "Widget.h"

class WidgetIterator
{
public:
    virtual ~WidgetIterator(){}
    virtual WidgetIterator* begin() = 0;
    virtual WidgetIterator* operator++() = 0;
    virtual bool end() = 0;
    virtual Widget* current() = 0;
};

#endif // WIDGETITERATOR_H
```

3.1.54 Window.h

```
#ifndef WINDOW_H
#define WINDOW_H

#include "Widget.h"
#include "WidgetIterator.h"

#include <string>

class Window
{
public:
    virtual ~Window(){}
    virtual void insertWidget(int x, int y, Widget* widget) = 0;
    virtual void insertWidget(int x0, int y0, int x1, int y1, Widget*
widget){}
    virtual WidgetIterator* widgetIterator() = 0;
    virtual void show() = 0;
    virtual std::string name() = 0;
};

#endif // WINDOW_H
```

3.2 CRIPTOGRAFIA

3.2.1 cripto.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <fstream>
#include <string>
#include <iostream>

using namespace std;

void encrypt (string dest, string arqin, string arqout)
```

```
{
     const char *out;
     string aux;
     aux = "gpg --homedir . --recipient \"" + dest + "\" --yes --output \"" + argout +
"\" --encrypt \"" + argin + "\"";
     out = aux.c_str();
     system(out);
}

void decrypt (string argin, string argout)
{
     const char *out;
     string aux;
     aux = "type passphrase.txt | gpg --homedir . --output \"" + argout + "\" --
passphrase-fd 0 --decrypt \"" + argin + "\"";
     out = aux.c_str();
     system(out);
}


int main (int numargs, char *args[])
{
     string option, argin;

     if (numargs < 2)
     {
          cout << "Options: encrypt or decrypt" << endl << "In case of Encryption:
<file> <output> <destiny>" << endl;
          cout << "In case of Decryption: <file> <output>" << endl;
          return 1;
     }

     option = args[1];

     if (option == "encrypt")
     {
          string argin, dest, argout;
          argin = args[2];
          dest = args[4];
          argout = args[3];
          encrypt(dest, argin, argout);
     }

     if (option == "decrypt")
     {
          string argin, argout;
          argin = args[2];
          argout = args[3];
          decrypt(argin, argout);
     }
     return 0;
}
```

3.3 COMPRESSÃO

3.3.1 compressao.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <fstream>
#include <string>
#include <iostream>

using namespace std;

void compress (string dest, string folder)
{
     const char *out;
     string aux;
     aux = "zip " + dest + " -q -r " + folder;
```

```
    out = aux.c_str();
    system(out);
}

void decompress (string dest, string file)
{
    const char *out;
    string aux;
    aux = "unzip -q -d " + dest + " " + file;
    out = aux.c_str();
    system(out);
}


int main(int numargs, char *args[])
{
    if (numargs < 4)
    {
        cout << "Options: <compress> <decompress>" << endl;
        cout << "In case of compression: <folder> <output>" << endl;
        cout << "In case of decompression: <file> <output>" << endl;
        return 1;
    }

    string option, folder, file;

    option = args[1];

    if (option == "compress")
    {
        folder = args[2];
        file = args[3];
        compress(file, folder);
    }

    if (option == "decompress")
    {
        file = args[2];
        folder = args[3];
        decompress(folder, file);
    }

    return 0;
}
```

3.4 AUTENTICAÇÃO

3.4.1 Login.au3

```
#cs ----------------------------------------------------------------

 AutoIt Version: 3.3.0.0
 Author:         André Gustavo Rigon

 Script Function:
        Template AutoIt script.

#ce ----------------------------------------------------------------

; Script Start - Add your code below here

#include "WindowSettings.au3"
#include <GUIConstantsEx.au3>
#include <EditConstants.au3>
```

```
Global $window
Global $current_height
Global $login
Global $password
Global $login_info_key = 0
Global $MAX_WAIT = 10
Global $delay = 1000

Global $filename = "Login\login.txt"
Global $confirmation_filename = "Login\loggedin.txt"
Global $interface_path = "Interface.exe"
Global $username_file = "Login\username.txt"
Global $authentication_filename = "crypt.exe"
Global $user_dir = "User"

main()

Func sheet()
	add_fields("Login", "Senha")
	add_buttons("","", "", "", "", $login_button, "")
EndFunc

Func main()
	calculate_window_height()
	$window_height = $current_height
	Opt("GUIOnEventMode", 1)
	$window  = GUICreate($window_name, $window_width, $window_height)
	GUISetOnEvent($GUI_EVENT_CLOSE, "Cancelar")
	$current_height = $std_vert_margin
	sheet()
	GUISetState()

	While 1
		Sleep(1000)
	WEnd
EndFunc

Func calculate_window_height()
	Opt("GUIOnEventMode", 1)
	$win1 = GUICreate("Rendering window...", $window_width, $window_height)
	GUISetOnEvent($GUI_EVENT_CLOSE, "Cancelar")
	$current_height = $std_vert_margin
	sheet()
	$current_height += $std_vert_margin*0.5
	GUIDelete($win1)
EndFunc

Func add_fields($label1, $label2)
	; Constantes
	Local $input_height = $std_height
	Local $label_height = $std_height
```

```
	Local $label_width = ($window_width - 6*$std_hor_margin)/4
	Local $input_width = $label_width*3

	GUICtrlCreateLabel($label1, $std_hor_margin*2 + $label_width/3,
$current_height*1.1, $label_width - $label_width/3, $label_height)

	$login = GUICtrlCreateInput("", 3*$std_hor_margin + $label_width, $current_height,
$input_width, $input_height, -1)
	$current_height += ($input_height + $std_offset*0.5)

	GUICtrlCreateLabel($label2, $std_hor_margin*2 + $label_width/3,
$current_height*1.1, $label_width - $label_width/3, $label_height)

	$password = GUICtrlCreateInput("", 3*$std_hor_margin + $label_width,
$current_height, $input_width, $input_height, $ES_password)
	$current_height += ($input_height + $std_offset*3)
EndFunc

Func add_buttons($label1, $label2, $label3, $label4, $label5, $label6, $label7)
	Local $button_height = $std_height * 1.4
	Local $button_width = $std_height * 3
	Local $current_margin = $window_width - $std_hor_margin - $button_width
	Local $distance = ($std_width - 7*$button_width)/6

	If $label7 <> "" Then
		Local $b7 = GUICtrlCreateButton($label7, $current_margin, $current_height,
$button_width, $button_height)
		$label7 = StringReplace($label7, " ", "_")
		GUICtrlSetOnEvent(-1, $label7)
	EndIf
	$current_margin -= ($button_width + $distance)

	If $label6 <> "" Then
		Local $b6 = GUICtrlCreateButton($label6, $current_margin, $current_height,
$button_width, $button_height)
		$label6 = StringReplace($label6, " ", "_")
		GUICtrlSetOnEvent(-1, $label6)
	EndIf
	$current_margin -= ($button_width + $distance)

	If $label5 <> "" Then
		Local $b5 = GUICtrlCreateButton($label5, $current_margin, $current_height,
$button_width, $button_height)
		$label5 = StringReplace($label5, " ", "_")
		GUICtrlSetOnEvent(-1, $label5)
	EndIf
	$current_margin -= ($button_width + $distance)

	If $label4 <> "" Then
		Local $b4 = GUICtrlCreateButton($label4, $current_margin, $current_height,
$button_width, $button_height)
		$label4 = StringReplace($label4, " ", "_")
```

```
                GUICtrlSetOnEvent(-1, $label4)
        EndIf
        $current_margin -= ($button_width + $distance)

        If $label3 <> "" Then
                Local $b3 = GUICtrlCreateButton($label3, $current_margin, $current_height,
$button_width, $button_height)
                $label3 = StringReplace($label3, " ", "_")
                GUICtrlSetOnEvent(-1, $label3)
        EndIf
        $current_margin -= ($button_width + $distance)

        If $label2 <> "" Then
                Local $b2 = GUICtrlCreateButton($label2, $current_margin, $current_height,
$button_width, $button_height)
                $label2 = StringReplace($label2, " ", "_")
                GUICtrlSetOnEvent(-1, $label2)
        EndIf
        $current_margin -= ($button_width + $distance)

        If $label1 <> "" Then
                Local $b1 = GUICtrlCreateButton($label1, $current_margin, $current_height,
$button_width, $button_height)
                $label1 = StringReplace($label1, " ", "_")
                GUICtrlSetOnEvent(-1, $label1)
        EndIf

        $current_margin -= ($button_width + $distance)
        $current_height += ($button_height + $std_offset)
EndFunc

Func execute_login($user_login, $user_password)
        Local $wait = 8
        Local $label_key
        Local $file_exists = 0

        If $login_info_key = 0 Then
                $login_info_key = GUICtrlCreateLabel($login_message, $std_hor_margin*3,
$current_height - $std_height * 1.3 - $std_offset)
        EndIf
                get_passwords_file()
                $user_login = StringReplace($user_login, " ", "")
                ShellExecute("crypt.exe", $user_login&" "&$user_password, @ScriptDir,
"open", @SW_HIDE)
                While $wait < $MAX_WAIT
                        GUICtrlSetData($login_info_key, $login_message)
                        Sleep($delay)
                        If FileExists($confirmation_filename) Then
                                $file_exists = 1
                                ExitLoop
                        EndIf
                        $wait += 1
```

```
                WEnd
                If $file_exists <> 1 Then
                        GUICtrlSetData($login_info_key, $wrong_login_message)
                        Sleep(1000)
                        GUICtrlSetData($login_info_key, "")
                        Return
                Else
                        GuiDelete($window)
                        ;MsgBox(0, $login_button, $logged_in_message, $msgbox_timeout)
                        open_interface()
                        Exit
                EndIf
EndFunc

Func get_passwords_file()
        ;FAZER
        ;Função que recebe do Arduino o arquivo de senhas password.txt
EndFunc

Func open_interface()
        Run($interface_path)
EndFunc

Func login()
        Local $user_login = GUICtrlRead($login)
        Local $user_password = GUICtrlRead($password)

        Local $handle = FileOpen($filename, 10)
        FileWriteLine($handle, $user_login)
        FileWriteLine($handle, $user_password)
        FileClose($handle)

        $handle = FileOpen($username_file, 10)
        FileWriteLine($handle, $user_login)
        FileClose($handle)

        DirCreate($user_dir)

        FileDelete($confirmation_filename)
        execute_login($user_login, $user_password)
EndFunc

Func Cancelar()
        Exit
EndFunc
```

3.4.2 WindowSettings.au3

```
#include-once
```

```
; Esses valores podem ser modificados

Global $window_name          = "Blue"
Global $window_height                = 700 ; A altura é calculada automaticamente. Não
adianta mudar esse valor, mas ele é necessário
Global $window_width         = 300 ; Largura da janela. A altura é calculada
automaticamente para que todos os componentes caibam
Global $std_width                    = $window_width * 0.9 ; Largura dos componentes
Global $std_height                   = 20 ; Componentes tem altura multipla desse valor
Global $std_offset                   = 10 ; Separação vertical entre os componentes
Global $std_hor_margin               = $window_width * 0.05 ; Margem horizontal dos
componentes em relação à borda esqueda da janela
Global $std_vert_margin      = $window_height * 0.05 ; Margem vertical do primeiro
componente em relação ao topo da janela
Global $tab_hor_margin               = $window_width * 0.025 ; Margem horizontal das
abas em relação à borda esqueda da janela
Global $tab_vert_margin      = $window_height * 0.025 ; Margem vertical das abas em
relação ao topo da janela
Global $MAX_YEAR                     = 2100
Global $data_folder                  = "Data" ; Pasta em que o prontuário é armazenado
Global $undo_button                  = "Desfazer"
Global $save_button                  = "Salvar"
Global $login_button         = "Login"
Global $login_message        = "Logging in..."
Global $logged_in_message    = "Abrindo prontuário..."
Global $wrong_login_message = "Login errado."
Global $save_msg                     = "Mudanças salvas."
Global $undo_msg                     = "Mudanças desfeitas."
Global $change_forbidden     = "Não é permitido modificar momentos anteriores"
Global $msgbox_timeout               = 0.6 ; Tempo de duração das mensagens, em
segundos. Se for 0, as mensagens só desaparecem quando o usuário clica em OK.


; COPIA DOS VALORES ORIGINAIS
;
;~ Global $window_name              = "Blue"
;~ Global $window_height            = 700 ; A altura é calculada automaticamente. Não
adianta mudar esse valor, mas ele é necessário
;~ Global $window_width             = 300 ; Largura da janela. A altura é calculada
automaticamente para que todos os componentes caibam
;~ Global $std_width                = $window_width * 0.9 ; Largura dos componentes
;~ Global $std_height               = 20 ; Componentes tem altura multipla desse valor
;~ Global $std_offset               = 10 ; Separação vertical entre os componentes
;~ Global $std_hor_margin           = $window_width * 0.05 ; Margem horizontal dos
componentes em relação à borda esqueda da janela
;~ Global $std_vert_margin  = $window_height * 0.05 ; Margem vertical do primeiro
componente em relação ao topo da janela
;~ Global $tab_hor_margin           = $window_width * 0.025 ; Margem horizontal das
abas em relação à borda esqueda da janela
;~ Global $tab_vert_margin  = $window_height * 0.025 ; Margem vertical das abas em
relação ao topo da janela
```

```
;~ Global $MAX_YEAR                    = 2100
;~ Global $data_folder                 = "Data" ; Pasta em que o prontuário é armazenado
;~ Global $undo_button                         = "Desfazer"
;~ Global $save_button                 = "Salvar"
;~ Global $login_button          = "Login"
;~ Global $login_message               = "Logging in..."
;~ Global $logged_in_message  = "Abrindo prontuário..."
;~ Global $wrong_login_message         = "Login errado."
;~ Global $save_msg                    = "Mudanças salvas."
;~ Global $undo_msg                    = "Mudanças desfeitas."
;~ Global $change_forbidden    = "Não é permitido modificar momentos anteriores"
;~ Global $msgbox_timeout              = 0.6 ; Tempo de duração das mensagens, em
segundos. Se for 0, as mensagens só desaparecem quando o usuário clica em OK.
```

3.5 MÓDULO DE DADOS

3.5.1 comm_arduino.pde

```
#include <EEPROM.h>
        byte PROTOCOL_SAVE_COMMAND[] = {'s', 'a', 'v', 'e'};
        int SAVE_COMMAND_SIZE = 4;
        byte PROTOCOL_LOAD_COMMAND[] = {'l', 'o', 'a', 'd'};
        int LOAD_COMMAND_SIZE = 4;
        byte PROTOCOL_STATUS_COMMAND[] = {'s', 't', 'a', 't', 'u', 's'};
        int STATUS_COMMAND_SIZE = 6;
        byte FILE_START_DELIM[] = {'s', 't', 'a', 'r', 't'};
        int START_DELIM_SIZE = 5;
        byte FILE_END_DELIM[] = {'e', 'n', 'd'};
        int END_DELIM_SIZE = 3;

      #define EEPROM_SIZE 512
      #define RAM_SIZE 512
      byte RAM[RAM_SIZE];

        byte receive() {
                int i;
                while ((i = Serial.read()) == -1);
                byte b = (byte) i;
                return b;
        }
        void send(byte b) {
                Serial.print(b, BYTE);
        }
        byte read(int address) {
                 byte b;
                 if (address < RAM_SIZE) {
                         b = RAM[address];
                 }
                 else if (address < RAM_SIZE + EEPROM_SIZE) {
                        b = EEPROM.read(address - RAM_SIZE);
                 }
                 else {
                         Serial.print("ABORT");
                 }
                 return b;
        }
        void write(byte b, int address) {
                 if (address < RAM_SIZE) {
                         RAM[address] = b;
                 }
                 else if (address < RAM_SIZE + EEPROM_SIZE) {
                        EEPROM.write(address - RAM_SIZE, b);
                 }
```

```
        else {
                Serial.print("ABORT");
        }
}

void save() {
        // Espera receber a FILE_START_DELIM
        int address = 0;
        int startDelimCount = 0;
        while(1) {
                byte b = receive();
                if (b == FILE_START_DELIM[startDelimCount]) {
                        ++startDelimCount;
                }
                else {
                        startDelimCount = 0;
                }
                if (startDelimCount == START_DELIM_SIZE) {
                        break;
                }
        }

        // Escreve FILE_START_DELIM na memória
        for (int i = 0; i < START_DELIM_SIZE; i++) {
                write(FILE_START_DELIM[i], address++);
        }

        // Até receber FILE_END_DELIM,
        // grava os bytes na memória (inclusive FILE_END_DELIM)
        int endDelimCount = 0;
        while(1) {
                byte b = receive();
                write(b, address++);
                if (b == FILE_END_DELIM[endDelimCount]) {
                        ++endDelimCount;
                }
                else {
                        endDelimCount = 0;
                }
                if (endDelimCount == END_DELIM_SIZE) {
                        break;
                }
        }
}

void load() {
        int address = 0;
        int endDelimCount = 0;
        while (1) {
                byte b = read(address++);
                send(b);
                if (b == FILE_END_DELIM[endDelimCount]) {
                        ++endDelimCount;
                }
                else {
                        endDelimCount = 0;
                }
                if (endDelimCount == END_DELIM_SIZE) {
                        break;
                }
        }

}

void setup() {
        Serial.begin(115200);
}

void loop() {
        int saveCommandCount = 0;
        int loadCommandCount = 0;
        while (1) {
                byte b = receive();

                if (b == PROTOCOL_SAVE_COMMAND[saveCommandCount]) {
                        ++saveCommandCount;
```

```
                    }
                    else {
                            saveCommandCount = 0;
                    }
                    if (saveCommandCount == SAVE_COMMAND_SIZE) {
                            saveCommandCount = 0;
                            save();
                    }

                    if (b == PROTOCOL_LOAD_COMMAND[loadCommandCount]) {
                            ++loadCommandCount;
                    }
                    else {
                            loadCommandCount = 0;
                    }
                    if (loadCommandCount == LOAD_COMMAND_SIZE) {
                            loadCommandCount = 0;
                            load();
                    }
            }
             delay(2000);
      }
```

3.6 COMUNICAÇÃO BLUETOOTH – MÓDULO DE USUÁRIO

3.6.1 Debug.java

```
      public class Debug {
            public enum Mode {
                  OFF, LOW, MEDIUM, HIGH;
            }
            public Mode debugMode = Mode.OFF;

            public Debug(Mode debugMode) {
                  this.debugMode = debugMode;

            }

            public void debugMessage(Mode requiredMode, String
message) {
                  if (debugMode.compareTo(requiredMode) >= 0)
                        System.err.print(message);
            }

            public void debugMessage(Mode requiredMode, char message)
{
                  if (debugMode.compareTo(requiredMode) >= 0)
                        System.err.print(message);
            }
      }
```

3.6.2 FileStream.java

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
```

```
import java.util.Vector;

public class FileStream {
	private Debug debug = new Debug(Debug.Mode.OFF);
	private InputStream in;
	private OutputStream out;
	private String inputFilename;
	private String outputFilename;

	private void writeByte(byte b) {
		try {
			out.write(b);
		} catch (IOException e) {
			e.printStackTrace();
		}
		debug.debugMessage(Debug.Mode.MEDIUM, (char) b);
		debug.debugMessage(Debug.Mode.HIGH, ": " + b + "\n");
	}

	private int readByte() {
		int i = -1;
		try {
			i = in.read();
		} catch (IOException e) {
			e.printStackTrace();
		}
		debug.debugMessage(Debug.Mode.MEDIUM, (char) i);
		debug.debugMessage(Debug.Mode.HIGH, ": " + (byte) i +
"\n");
		return i;
	}

	public FileStream(String inputFilename, String outputFilename) {
		this.inputFilename = inputFilename;
		this.outputFilename = outputFilename;
	}

	public void close() {
		try {
			if (in != null)
				in.close();
			if (out != null)
				out.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	public void debugMode(Debug.Mode debugMode) {
		debug = new Debug(debugMode);
	}

	public void write(byte[] bytes) {
		if (out == null)
			try {
				out = new FileOutputStream(outputFilename);
			} catch (FileNotFoundException e) {
				e.printStackTrace();
				System.exit(1);
			}
		for (byte b : bytes) {
```

```
                writeByte(b);
            }
        }

        public void write(Vector<Byte> bytes) {
            if (out == null)
                try {
                    out = new FileOutputStream(outputFilename);
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                    System.exit(1);
                }
            for (byte b : bytes) {
                writeByte(b);
            }
        }

        public byte[] read() {
            if (in == null)
                try {
                    in = new FileInputStream(inputFilename);
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                    System.exit(1);
                }
            int i;
            Vector<Byte> byteVector = new Vector<Byte>();
            while ((i = readByte()) != -1) {
                byte b = (byte) i;
                byteVector.add(b);
            }
            byte[] byteArray = new byte[byteVector.size()];
            int j = 0;
            for (Byte b : byteVector) {
                byteArray[j] = b;
                ++j;
            }
            return byteArray;
        }
}
```

3.6.3 Main.java

```
import java.io.FileInputStream;

public class Main {
    public static void main(String[] args) {
        if (args.length != 3) {
            System.out.println("[save/load] [file] [port]");
            System.exit(0);
        }

        String file = args[1];
        String port = args[2];

//      String in =
"C:\\Users\\Familia\\Desktop\\Files\\content.txt.zip.gpg";
```

```
//          String out =
"C:\\Users\\Familia\\Desktop\\Files\\content_out.txt.zip.gpg";
//          String port = "COM21";

            Protocol protocol = new Protocol(Debug.Mode.LOW, port,
file, file);

            if (args[0].equals("save")) {
                  System.out.println("Sending file");
                  protocol.saveFile();
            }
            if (args[0].equals("load")) {
                  System.out.println("Receiving file");
                  protocol.loadFile();
            }

            //compare(in, out);
            System.out.println("Done");
            protocol.close();
      }

      public static boolean compare(String in, String out) {
            StringBuffer inContent = new StringBuffer();
            StringBuffer outContent = new StringBuffer();

            try {
                  FileInputStream fileIn = new FileInputStream(in);
                  FileInputStream fileOut = new FileInputStream(out);

                  int i;
                  while ((i = fileIn.read()) != -1) {
                        inContent.append(i);
                  }
                  while ((i = fileOut.read()) != -1) {
                        outContent.append(i);
                  }
            } catch (Exception e) {
                  throw new RuntimeException("Problema no compare!");
            }

            boolean result = true;

            if (inContent.length() != outContent.length()) {
                  System.out.println("Tamanhos diferentes");
                  result = false;
            }

            for (int j = 0; j < inContent.length(); j++) {
                  if (inContent.charAt(j) != outContent.charAt(j)) {
                        System.out.println("Diferença: " + j);
                        result = false;
                  }
            }

            System.out.println("Compare: " + result);
            System.out.println("(Resultado equals: "
                        +
inContent.toString().equals(outContent.toString()) + ")");

            return result;
      }
```

```
}
```

3.6.4 Protocol.java

```
import java.util.Vector;


public class Protocol {
	private Debug debug = new Debug(Debug.Mode.OFF);
	private String FILE_START_DELIM = "start";
	private String FILE_END_DELIM = "end";
	private String PROTOCOL_SAVE_COMMAND = "save";
	private String PROTOCOL_LOAD_COMMAND = "load";
	private String PROTOCOL_STATUS_COMMAND = "status";

	private FileStream fileStream;
	private SerialStream serialStream;

	public static class Listener implements Runnable {

		private SerialStream serialStream;
		private Debug debug = new Debug(Debug.Mode.OFF);

		public Listener(Debug.Mode debugMode, SerialStream
serialStream) {
			this.serialStream = serialStream;
			debug = new Debug(debugMode);
		}

		@Override
		public void run() {
			debug.debugMessage(Debug.Mode.LOW, "Starting
listener thread...");
			while(true) {
				char c = (char) serialStream.receive();
				debug.debugMessage(Debug.Mode.LOW, c);
			}
		}
	}

	public Protocol(Debug.Mode debugMode, String portName, String
inputFilename, String outputFilename) {
		fileStream = new FileStream(inputFilename,
outputFilename);
		serialStream = new SerialStream(portName);
		debug = new Debug(debugMode);
		fileStream.debugMode(debugMode);
		serialStream.debugMode(debugMode);
	}

	public void close() {
		fileStream.close();
		serialStream.close();
		System.exit(0);
	}
	public void debugMode(Debug.Mode debugMode) {
		this.debug = new Debug(debugMode);
```

```
	}

	public void saveFile() {
		debug.debugMessage(Debug.Mode.LOW, "\n> Reading file from
filesystem...\n");
		byte[] fileContent = fileStream.read();
		debug.debugMessage(Debug.Mode.LOW, "\n> Done (" +
fileContent.length + " bytes)\n");
		debug.debugMessage(Debug.Mode.LOW, "\n> Sending file to
serial...\n");
		serialStream.send(PROTOCOL_SAVE_COMMAND);
		serialStream.send(FILE_START_DELIM);
		serialStream.send(fileContent);
		serialStream.send(FILE_END_DELIM);
		debug.debugMessage(Debug.Mode.LOW, "\n> Done (" +
fileContent.length + " bytes)\n");
	}
	public void loadFile() {
		debug.debugMessage(Debug.Mode.LOW, "\n> Receiving file
from serial...\n");
		serialStream.send(PROTOCOL_LOAD_COMMAND);
		for(int fileStartDelimCount = 0;;) {
			byte b = serialStream.receive();
			if (b == (byte)
FILE_START_DELIM.charAt(fileStartDelimCount)) {
				++fileStartDelimCount;
			}
			else {
				fileStartDelimCount = 0;
			}
			if (fileStartDelimCount ==
FILE_START_DELIM.length())
				break;
		}
		Vector<Byte> fileContent = new Vector<Byte>();
		for(int fileEndDelimCount = 0;;) {
			byte b = serialStream.receive();
			fileContent.add(b);

			if (b == (byte)
FILE_END_DELIM.charAt(fileEndDelimCount)) {
				++fileEndDelimCount;
			}
			else {
				fileEndDelimCount = 0;
			}
			if (fileEndDelimCount == FILE_END_DELIM.length()) {
				for(int count = 0; count <
FILE_END_DELIM.length(); count++) {
					fileContent.remove(fileContent.size()-
1);
				}
				break;
			}
		}
		debug.debugMessage(Debug.Mode.LOW, "\n> Done (" +
fileContent.size() + " bytes)\n");
		debug.debugMessage(Debug.Mode.LOW, "\n> Writing file to
filesystem...\n");
		fileStream.write(fileContent);
		debug.debugMessage(Debug.Mode.LOW, "\n> Done (" +
```

```
fileContent.size() + " bytes)\n");
        }
}
```

3.6.5 SerialStream.java

```
import gnu.io.CommPort;
import gnu.io.CommPortIdentifier;
import gnu.io.SerialPort;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class SerialStream {
        private Debug debug = new Debug(Debug.Mode.OFF);
        private InputStream in;
        private OutputStream out;
        private CommPort commPort;
        private SerialPort serialPort;
        private boolean AUTOFLUSH = false;

        private void sendByte(byte b) {
                try {
                        out.write(b);
                        if (AUTOFLUSH)
                                out.flush(); // REMOVER
                } catch (IOException e) {
                        e.printStackTrace();
                }
                debug.debugMessage(Debug.Mode.MEDIUM, (char) b);
                debug.debugMessage(Debug.Mode.HIGH, ": " + b + "\n");
        }

        private byte receiveByte() {
                int i = -1;
                try {
                        while ((i = in.read()) == -1)
                                ;
                } catch (IOException e) {
                        e.printStackTrace();
                }
                byte b = (byte) i;
                debug.debugMessage(Debug.Mode.MEDIUM, (char) b);
                debug.debugMessage(Debug.Mode.HIGH, ": " + b + "\n");
                return b;
        }

        public SerialStream(String portName) {
                try {
                        connect(portName);
                } catch (Exception e) {
                        e.printStackTrace();
                        throw new RuntimeException();
                }
        }
```

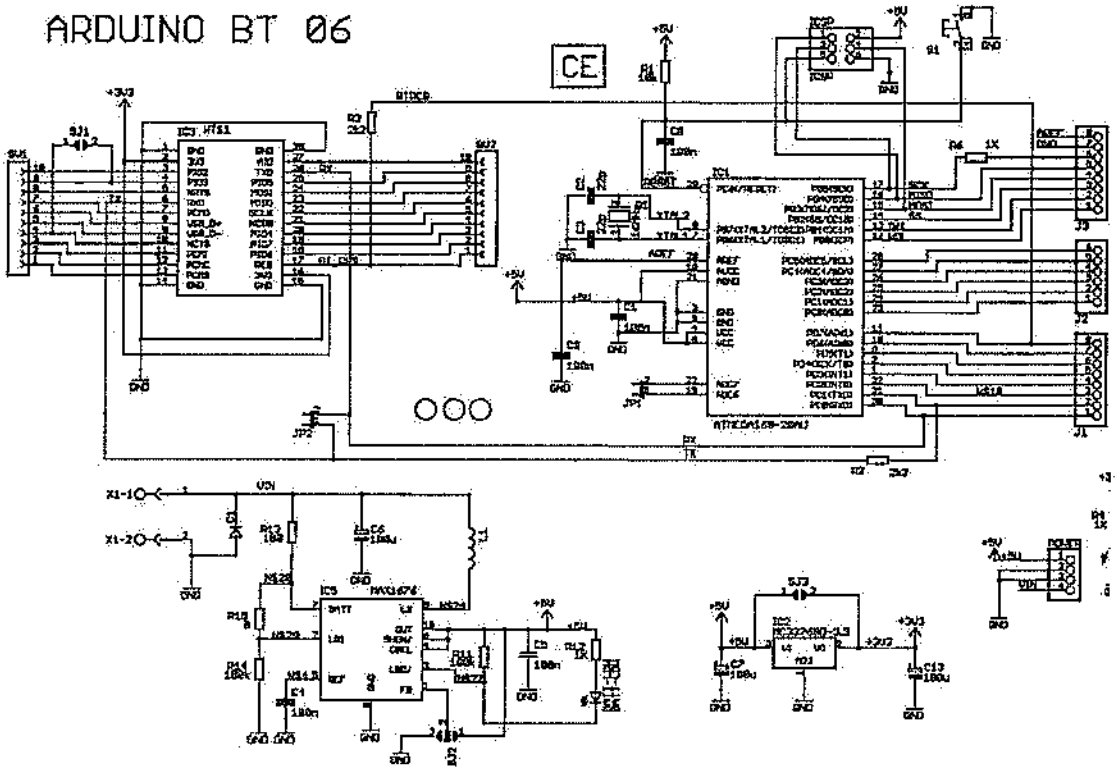
```
	void connect(String portName) throws Exception {
		CommPortIdentifier portIdentifier = CommPortIdentifier
				.getPortIdentifier(portName);
		if (portIdentifier.isCurrentlyOwned()) {
			System.out.println("Error: Port is currently in
use");
		} else {
			commPort =
portIdentifier.open(this.getClass().getName(), 2000);

			if (commPort instanceof SerialPort) {
				serialPort = (SerialPort) commPort;
				serialPort.setSerialPortParams(57600,
SerialPort.DATABITS_8,
						SerialPort.STOPBITS_1,
SerialPort.PARITY_NONE);

				in = serialPort.getInputStream();
				out = serialPort.getOutputStream();

			} else {
				System.out
						.println("Error: Only serial ports
are handled by this example.");
			}
		}
	}

	public void close() {
		try {
			in.close();
			out.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	public void debugMode(Debug.Mode debugMode) {
		debug = new Debug(debugMode);
	}

	public void send(byte b) {
		sendByte(b);
	}

	public void send(byte[] bytes) {
		for (byte b : bytes) {
			sendByte(b);
		}
	}

	public void send(String s) {
		char[] chars = s.toCharArray();
		for (char c : chars) {
			byte b = (byte) c;
			sendByte(b);
		}
	}

	public byte receive() {
		return receiveByte();
```

```
        }
}
```

APÊNDICE 4 – DOCUMENTOS TÉCNICOS

4.1 ARDUINO BT

4.1.1 Esquemático



4.2 ATMEGA168

4.2.1 Block Diagram and Descriptions

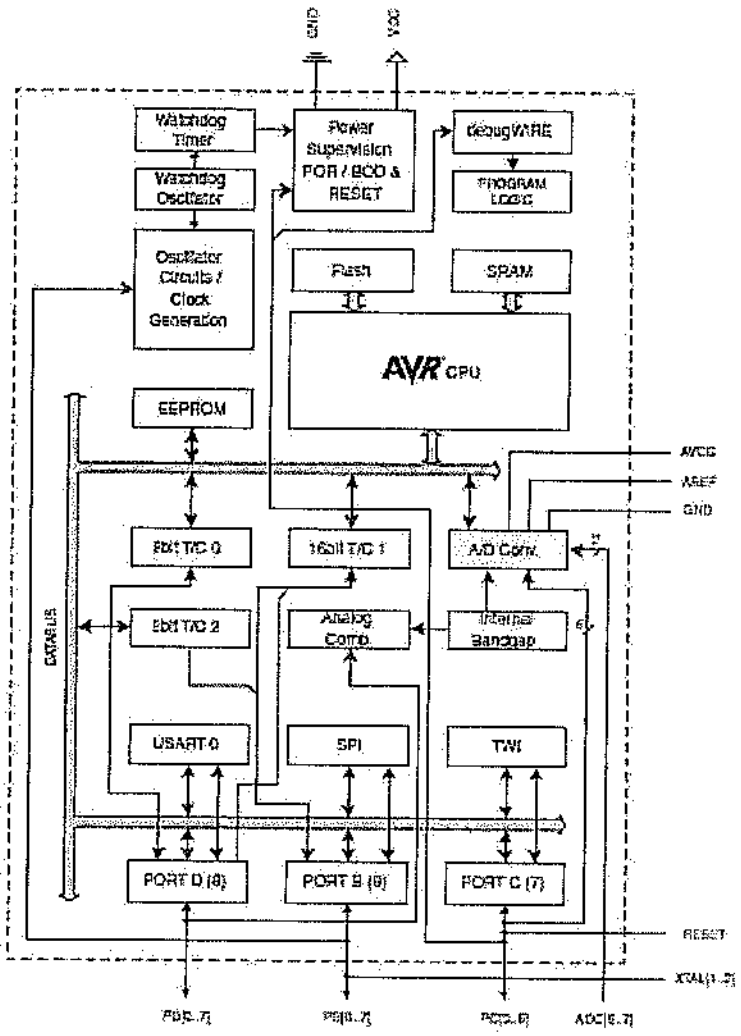


Figure 59 - Block Diagram of ATmega168

The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega48/88/168 provides the following features: 4K/8K/16K bytes of In-System Programmable Flash with Read-While-Write capabilities, 256/512/512 bytes EEPROM, 512/1K/1K bytes SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, a byte-oriented 2-wire Serial Interface, an SPI serial port, a 6-channel 10-bit ADC (8 channels in TQFP and QFN/MLF packages), a programmable Watchdog Timer with Internal Oscillator, and five software selectable

power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, USART, 2-wire Serial Interface, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or hardware reset.

In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption.

The device is manufactured using Atmel's high density non-volatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed In-System through an SPI serial interface, by a conventional non-volatile memory programmer, or by an On-chip Boot program running on the AVR core. The Boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega48/88/168 is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The ATmega48/88/168 AVR is supported with a full suite of program and system development tools including: C Compilers, Macro Assemblers, Program Debugger/Simulators, In-Circuit Emulators,and Evaluation kits.

Mais informações a respeito das características do componente podem ser encontradas em seu Data Sheet. O link encontra-se nas referências.

4.3 BLUEGIGA WT11

4.3.1 Block Diagram and Descriptions

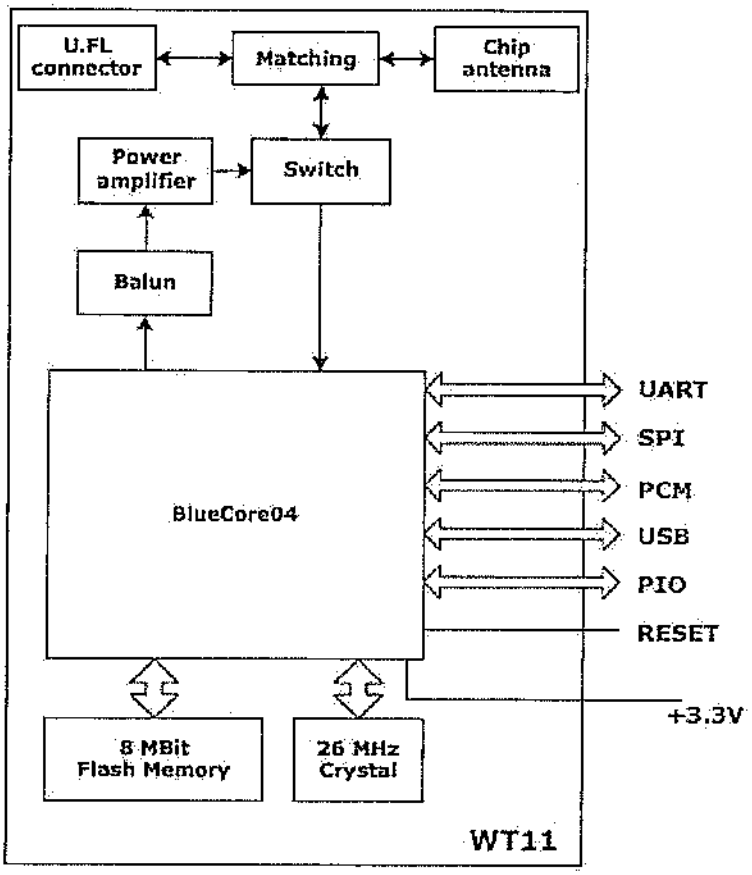


Figure 60 - Block Diagram of WT11

BlueCore04

BlueCore4 is a single chip Bluetooth solution which implements the Bluetooth radio transceiver and also an on chip microcontroller. BlueCore4 implements Bluetooth® 2.0+EDR (Enhanced Data Rate) and it can deliver data rates up to 3 Mbps.

The microcontroller (MCU) on BlueCore04 acts as interrupt controller and event timer run the Bluetooth software stack and control the radio and host interfaces. A 16-bit reduced instruction set computer (RISC) microcontroller is used for low power consumption and efficient use of memory.

BlueCore04 has 48Kbytes of on-chip RAM is provided to support the RISC MCU and is shared between the ring buffers used to hold voice/data for each active connection and the general purpose memory required by the Bluetooh stack.

Crystal

The crystal oscillates at 26 MHz.

Flash

Flash memory is used for storing the Bluetooth protocol stack and Virtual Machine applications. It can also be used as an optional external RAM for memory intensive applications.

Balun

Balun changes the balanced input/output signal of the module to unbalanced signal of the monopole antenna.

Power amplifier

Power amplifier is used to increase the output power to a level required by class 1 specification.

Switch

Switch is used to separate transmission and receiver modes.

Matching

Antenna matching components match the antenna to 50 Ohms and also selects between chip antenna and UFL connector.

Antenna

The antenna is ACX AT3216 chip antenna.

U.FL

This is a standard U.FL male connector for external antenna possibility.

USB

This is a full speed Universal Serial Bus (USB) interface for communicating with other compatible digital devices. WT11 acts as a USB peripheral, responding to requests from a Master host controller such as a PC.

Synchronous Serial Interface

This is a synchronous serial port interface (SPI) for interfacing with other digital devices. The SPI port can be used for system debugging. It can also be used for programming the Flash memory.

UART

This is a standard Universal Asynchronous Receiver Transmitter (UART) interface for communicating with other serial devices.

Audio PCM Interface

The audio pulse code modulation (PCM) interface supports continuous transmission and reception of PCM encoded audio data over Bluetooth.

Programmable I/O

WT11 has a total of 6 digital programmable I/O terminals. These are controlled by firmware running on the device.

Reset

This can be used to reset WT11.

Mais informações a respeito das características do componente podem ser encontradas em seu Data Sheet. O link encontra-se nas referências.