Conexão de um Módulo Eletrônico com Microcontrolador PIC em Rede GSM utilizando Arduino e Web Services RESTful

Trabalho de Conclusão do Curso de
Tecnologia em Sistemas para Internet

Tiago Gonzales Dermann
Orientador: Alex Dias Gonsales

[1]Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Campus Porto Alegre
Rua Cel. Vicente, 281, Porto Alegre – RS – Brasil

tgdbr@yahoo.com.br, alex.gonsales@poa.ifrs.edu.br

Resumo. *Este trabalho descreve a conexão entre um módulo eletrônico de aquisição de dados equipado com Microcontrolador PIC e a internet. O equipamento, de baixo custo, não foi idealizado originalmente com esta funcionalidade. Para realizar esta conexão, foi desenvolvida uma camada de comunicação envolvendo o uso da plataforma Arduino em conjunto com o Shield SIM900 para acessar a rede GSM e, através do serviço GPRS, publicar os dados do módulo por meio de requisições HTTP a Web Services.*

Palavras-chave: Arduino, Módulo Eletrônico, GPRS, GSM, PIC, RESTful, Web Services.

1. Introdução

O ensino experimental é uma técnica que propicia uma nova perspectiva na relação do aluno com a construção do conhecimento científico, sendo as atividades experimentais consideradas como um importante instrumento de auxílio à prática pedagógica [de Oliveira Barbosa et al. 1999].

Chinelli et al. [Chinelli et al. 2008] identificaram equipamentos utilizados no ensino de ciências, que com características de versatilidade e, em via de regra, baixo custo, podem cumprir o papel de incentivar uma forma de aprendizagem mais interessada e significativa. Quando se decide pelo desenvolvimento desses equipamentos está se fomentando a pesquisa e a busca por novos conhecimentos, os quais podem ser úteis a toda comunidade envolvida no processo que a executa, possibilitando inclusive a geração de outras pesquisas em áreas correlatas.

Sendo assim, no ano de 2012 teve início, no IFRS - Campus Porto Alegre, o projeto de pesquisa “Desenvolvimento de um Módulo de Monitoramento de Temperatura para uso em Ambiente Educacional". O Projeto teve como finalidade o desenvolvimento de um equipamento de baixo custo que permitisse o monitoramento e registro de temperatura, de forma a ser utilizado em ambiente educacional, seja em experimentos de sala de aula ou mesmo em projetos de pesquisa ou extensão.

A proposta demonstrou-se viável funcionalmente e economicamente, tendo sido desenvolvido um módulo portátil contendo quatro sensores de temperatura, com opções

de configuração do tempo de amostragem, da quantidade de sensores desejados, além de poder armazenar em sua memória interna até 100 valores monitorados. Possui, também, a funcionalidade de transmitir os dados armazenados para um computador PC via porta serial RS-232, sendo possível abrir este arquivo em planilha eletrônica e gerar gráficos.

O módulo desenvolvido caracteriza-se por ser facilmente modificável para a adição de outros tipos de sensores, tendo a pesquisa evoluído com a integração de sensores de gás da linha MQ (monóxido e dióxido de carbono, GLP, etanol, gás natural e gás hidrogênio) [Arduino 2015b] e radiação ultravioleta (UV-A, UV-B e UV-C).

No entanto, apesar das características descritas acima, o módulo não possui a funcionalidade específica para disponibilização dos dados na *web*, uma vez que o equipamento tem como base um microcontrolador PIC modelo 16F886, o qual dispõe apenas das interfaces seriais RS-232 e SPI *(Serial Peripheral Interface)* [Microchip 2007].

A conexão com a internet traz benefícios como uma maior facilidade e agilidade para a visualização dos dados salvos, a possibilidade de construção de consultas e oportuniza ao aluno a publicação direta dos dados de seus experimentos, dentro da ótica de utilização da *web* como meio de assistência ao processo de aprendizagem [Dias 2000].

Dessa forma, este trabalho toma como base de conhecimento o projeto de pesquisa desenvolvido nos anos anteriores, porém visando abordar a questão da disponibilidade imediata na internet dos dados adquiridos. O método escolhido foi a criação de uma camada de comunicação entre o módulo de sensores e a *web*. A solução utiliza uma placa Arduino UNO equipada com um *shield* GSM de forma a estabelecer uma conexão com um serviço remoto. A técnica abrange a comunicação serial do Arduino com os dispositivos, o conjunto de comandos AT do *shield* para o estabelecimento de conexões HTTP (*Hypertext Transfer Protocol*), a manipulação dos diferentes tipos de dados para a construção de objetos JSON (*JavaScript Object Notation*) e sua transmissão para um *Web Service*.

Este trabalho está organizado da seguinte forma: na seção 2 é feita uma revisão bibliográfica de trabalhados relacionados à aquisição e transmissão remota de dados. Na seção 3 é apresentada a proposta da solução desenvolvida, na seção 4 a metodologia empregada e na seção 5 o desenvolvimento. Por fim, na seção 6 são apresentadas as conclusões do trabalho.

2. Trabalhos Relacionados

Foram estudadas diversas tecnologias em aquisição de dados as quais utilizam redes GSM e seus serviços: GPRS (*General Packet Radio Service*) ou CSD *(Circuit Switched Data)* como forma de transmissão remota, sendo mostrados os trabalhos a seguir:

Colnago [Colnago 2009] apresenta o projeto de um medidor para aferir a qualidade da rede de energia elétrica. A arquitetura do sistema foi desenvolvida em nove diferentes blocos, compostos por circuitos dedicados. A variável monitorada é o sinal da rede de baixa tensão que, após passar pelas etapas de adequação, filtragem e conversão analógico-digital, é enviado por barramento SPI a um controlador dsPIC33, onde é feito o processamento e envio por interface serial a outro controlador de mesmo modelo. Este segundo controlador está conectado a um módulo Motorola G24 via interface RS-232. O

módulo G24 é programável em linguagem Java e é capaz de transmitir os dados via GPRS [Motorola 2008].

Machado et. al. [Machado et al. 2008] propõem uma arquitetura de sistema para monitoramento remoto por meio de dispositivos móveis e serviços *Web*. Um protótipo de cliente de aplicação para telefone celular foi desenvolvido em linguagem Java. Este sistema utiliza um monitor cardíaco, o qual coleta informações do paciente e as envia para um telefone celular por *bluetooth* ou infravermelho. O telefone celular deve ter conexão com a internet para poder fazer a requisição para o *Web Service*. A aplicação do servidor, por sua vez, atualiza as informações no banco de dados e, conforme necessário, as envia por *e-mail* ou as encaminha a um *Web Service* terceirizado para o envio de SMS (*Short Message Service*) para o médico responsável.

Tateoki [Tateoki 2007] demonstra um sistema de monitoramento, o qual se comunica com um servidor através de um modem GSM para, posteriormente, exibir os dados em página *Web*. No subsistema de comunicação via GPRS, foi feita a opção por um modem TC45 Siemens/Duodigit com a justificativa de usar a tecnologia GSM e de disponibilidade comercial do produto. Contudo, o autor ressalva a alta curva de aprendizagem para utilizar este *hardware*.

Gruber [Gruber 2007] aborda um protótipo para monitoramento de parâmetros de aerogeradores (equipamentos utilizados em energia eólica). Diferentemente dos outros trabalhos, este projeto opera com um modem GSM dedicado à transmissão e outro à recepção. O primeiro modem é ligado ao chamado Sistema de Monitoração e o outro à Placa de Aquisição de Dados. Durante a etapa de validação, os testes foram feitos em conexão CSD ao invés de GPRS. Isto significa que os dispositivos estão conectados ponto-a-ponto, sem conexão com a internet. No aspecto de *software*, é apresentado um supervisório desenvolvido em linguagem Delphi, para sistemas *Desktop*.

Vianna e Lima [Vianna and Lima 2014] desenvolveram uma plataforma de monitoramento ambiental baseada em Arduino utilizando o *Shield* SIM900 como meio de transmissão. O sistema, aplicado no monitoramento da poluição sonora, é equipado com um sensor de pressão sonora (decibelímetro) e conecta-se via GPRS a um servidor PHP, onde um *script* salva os dados em uma base MySQL. É possível, ainda, exportar os dados para uma planilha Excel, porém ainda não foi implementada uma forma de apresentação dos dados na *web*.

3. Solução Proposta

É utilizada a interface UART *(Universal Asynchronous Receiver/Transmitter)* do microcontrolador PIC existente no módulo de sensores para efetuar a comunicação serial com uma placa Arduino equipada com um *Shield* GSM SIM900 como forma de criar um meio de comunicação remota do módulo de sensores com um servidor *web*.

A conexão ocorre da seguinte maneira: O módulo de sensores efetua a leitura dos sensores e transmite os dados através de sua porta serial para a placa Arduino, a qual fica encarregada de gerenciar tanto a conexão com o módulo de sensores como a consistência da conexão com a rede GSM. A partir do momento em que receber a confirmação de que o serviço GPRS está ativado, o Arduino inicia uma conexão TCP remota e transmite os dados.

Através de comandos AT, enviados pelo Arduino ao SIM900, são realizadas requisições HTTP ao *Web Service*, o qual grava as informações em banco de dados. A partir disto, os dados podem ser exibidos em páginas *web* dinâmicas, acessíveis ao usuário normalmente via navegador.

A figura 1 mostra um diagrama onde estão representados os diversos dispositivos que fazem parte da proposta desenvolvida, bem como as interfaces de comunicação e protocolos utilizados.

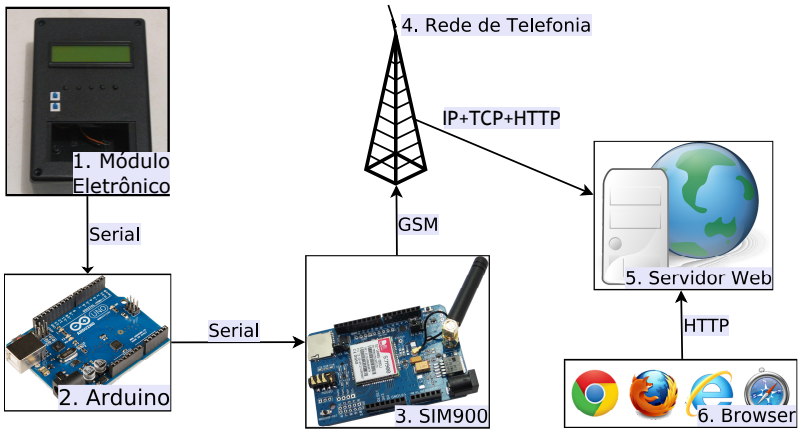

Figura 1. Proposta Geral

3.1. Justificativa

O *hardware* adicional necessário para a implementação desta solução consiste em uma placa Arduino e um *Shield* SIM900.

Arduino é uma plataforma de *hardware open-source* com o propósito de ser fácil de programar e possuir alta flexibilidade de *software e hardware*, ideal para o uso em protótipos [Sarik and Kymissis 2010].

Shields são placas externas que, conectadas à placa principal do Arduino, ampliam suas funcionalidades [Arduino 2015a].

O SIM900 é um *Shield* que proporciona acesso aos diversos serviços de telefonia GSM, como chamada de voz, mensagens SMS e serviço GPRS, também oferece suporte aos protocolos TCP/IP (*Transmission Control Protocol / Internet Protocol*) e HTTP através de comandos AT específicos [SIMCom 2010b].

No início da elaboração desta proposta, cogitou-se diferentes possibilidades de implementação desta solução, tais como o uso da Rede XBee, da Plataforma *Raspberry Pi* ou ainda a utilização do próprio microcontrolador PIC no gerenciamento da conexão de rede. Os fatores considerados na escolha da Rede GSM em detrimento da XBee foram o alcance (900 m) [XBee 2013] e a disponibilidade, pois a conexão GSM/GPRS está presente nos locais abrangidos pelas redes de telefonia celular sem a necessidade de repetidores.

A Plataforma Arduino foi escolhida avaliando-se os recursos computacionais requeridos e a complexidade de desenvolvimento, pois a programação do microcontrolador PIC é feita em nível mais baixo em relação ao *hardware*, enquanto

a *Raspberry Pi* oferece facilidades como diversas linguagens de alto nível e sistemas operacionais, porém esta plataforma foi considerada superdimensionada para os requisitos apresentados neste caso. Entre os modelos de Arduino oferecidos, optou-se pelo modelo UNO em função deste ser considerado o mais utilizado e melhor documentado, de acordo com a organização responsável pelo Projeto [Arduino 2016a]. Por outro lado, este modelo possui o *hardware* mais restrito entre todas as opções, porém suficiente para a proposta aqui apresentada.

No aspecto da programação *web*, escolheu-se o Java EE (*Enterprise Edition*), pois esta tecnologia proporciona facilidades para a programação de aplicações distribuídas, transacionais e portáteis, de maneira a reduzir o tempo de desenvolvimento, a complexidade da aplicação e aumentar o desempenho [Jendrock et al. 2014]. Além disso, torna-se facilitado o uso de boas práticas como os padrões de projeto (*design patterns*) e o MVC (*Model View Controller*).

A tecnologia Java EE permite a construção de aplicações em múltiplas camadas, implementadas pelo uso de diversos componentes. Para o desenvolvimento desta proposta foram utilizados os componentes JPA (*Java Persistence API*) e EJB (*Enterprise JavaBeans*).

O componente JPA é o responsável pelo mapeamento objeto-relacional das classes de entidade [Oracle 2015], abstraindo a camada de banco de dados em relação à aplicação em tempo de execução. A conexão com o banco de dados é criada através de uma *pool* instanciada pelo servidor e disponibilizada como um recurso (*resource*), o qual pode ser acessado pela Camada de Negócio (EJB).

As injeções de dependência são ativadas através de anotações no código Java e podem ser usadas pelo EJB ou nos clientes de aplicação. Esta prática promove o desacoplamento entre o código do cliente e do servidor, pois os objetos são instanciados no servidor, onde podem ou não conter informações de estado (*Stateful* ou *Stateless*). Isto é determinado pelo uso de anotações EJB.

Os serviços RESTful são um padrão de desenvolvimento para *Web Services* elaborado para a transferência de informações de forma *Stateless*, onde se utiliza tipicamente o protocolo HTTP [Jendrock et al. 2014]. O Padrão RESTful consegue simplificar o desenvolvimento e tornar a aplicação mais rápida.

3.2. Comparativo

A tabela 1 relaciona as principais características existentes nos sistemas pesquisados com uma breve descrição das escolhas feitas nesta proposta.

Nota-se que os sistemas pesquisados diferem em suas arquiteturas e áreas de atuação. Os parâmetros considerados foram: características de construção das interfaces de usuário; de implementação no servidor; no *hardware* principal utilizado; nos custos das soluções; nas formas de temporização e início da comunicação e nos dispositivos empregados para conexão à rede GSM.

As tabelas 2 e 3 exibem um comparativo entre os sistemas pesquisados e a solução proposta.

Tabela 1. Considerações sobre as características da solução

Característica	Descrição
Interface do usuário	A interface *Web* facilita a visualização dos dados tanto em computadores pessoais como em dispositivos móveis
Linguagem servidor	A linguagem Java com o uso de *Web Services* permite disponibilizar os métodos de envio remotamente para as aplicações
SGBD	O MySQL oferece um SGBD em *Software* Livre adequado às dimensões do projeto
Custo	Investimento no *hardware* GSM e na Placa Arduino
Tipo de transmissão	O módulo conta com a funcionalidade de comunicação serial RS-232 em periodicidade configurável pelo usuário (síncrona)
Dispositivo de comunicação	Os *shields* SIM900 são utilizados na Plataforma Arduino e contam com suporte da comunidade
Hardware Utilizado	O microcontrolador PIC16F886 do módulo existente em conjunto com a Plataforma Arduino

4. Metodologia

Conforme mencionado anteriormente, o módulo de sensores efetua o monitoramento da temperatura, permitindo a interação com o usuário por meio de teclado de botões e visor de cristal líquido (LCD). Também é capaz de salvar os dados em memória e transmiti-los via porta serial RS-232.

Para a implementação da proposta foi escolhida uma abordagem iniciando-se com o desenvolvimento do código do servidor web, pois isto facilita a realização de testes e o diagnóstico de problemas ao isolar as demais etapas.

A próxima parte é o desenvolvimento de um cliente *web* simples para acessar o *Web Service* disponível no servidor. Este cliente tem a finalidade de testar os métodos do serviço de envio de dados (valores dos sensores).

Em seguida, é implementado o código no Arduino para a conexão com o SIM900 e acesso ao serviço. Este código também resolve o tratamento da consistência da conexão.

Na sequência, é efetuada a recepção serial no Arduino para os dados enviados pelo módulo de sensores. A aplicação Arduino tem de manter estes dados na memória enquanto a conexão não estiver estabelecida.

5. Desenvolvimento

Primeiramente foi feita a modelagem entidade-relacionamento de maneira a refletir a estrutura de dados mínima envolvida no monitoramento dos sensores. Como primeira abordagem, suficiente para este trabalho, implementou-se o diagrama de classes com as entidades Leitura e Dado, conforme mostrado na figura 2.

Tabela 2. Comparativo das especificações dos sistemas pesquisados

Trabalho	Interface do usuário	Linguagem Servidor	SGBD	Custo
[Colnago 2009]	Desktop	Não informado	Oracle	Em torno de R$1000,00
[Machado et al. 2008]	Java *Mobile* e SMS	Java	MySQL	Não se aplica
[Tateoki 2007]	*Web*	PHP	MySQL	R$1037,00
[Gruber 2007]	Desktop (Delphi)	Não se aplica	Não se aplica	Não informado
[Vianna and Lima 2014]	Planilha	PHP	MySQL	Não informado
Proposta	*Web*	Java	MySQL	R$457,90

5.1. Servidor Web

Foram criadas duas tabelas (Dado e Leitura) em uma base MySQL. A Tabela Leitura representa uma sequência de monitoramento, ou seja, uma sequência de valores de temperatura dos sensores lidos do módulo. Essa tabela também pode representar um experimento sendo monitorado. Os dados sobre a conexão (data atual e IP), os parâmetros do módulo (quantidade de sensores e intervalo entre amostras) e um campo identificador (ID), o qual é a chave primária são armazenados e como campo opcional, foi prevista uma descrição sobre a conexão.

Na Tabela Dado ficam salvos todos os valores obtidos de cada um dos sensores em cada monitoramento efetuado. Nessa tabela são salvos o número do sensor (1 a 4), o valor de temperatura lido do sensor, a hora atual (obtida pelo servidor) e o cálculo para verificação de integridade (*hash*). O campo de identificação de Tabela Leitura é referenciado através de chave estrangeira (*foreign key*).

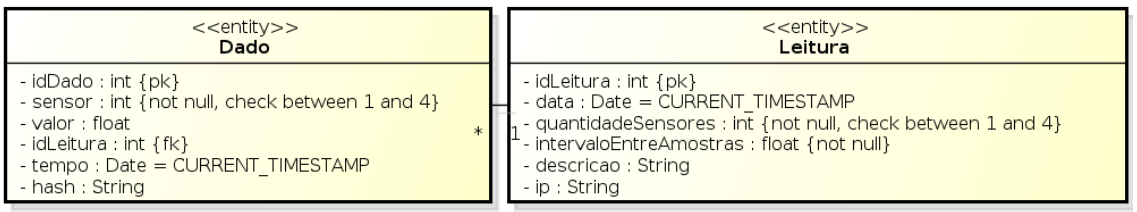

Figura 2. Diagrama de Classes

Para acesso às tabelas do banco de dados a partir do JPA é necessário criar uma *pool* de conexão e uma fonte de dados (*data source*) no servidor GlassFish.

Para cada uma das tabelas é criada uma classe Java de entidade, onde temos um modelo de objetos correspondentes aos registros de cada tabela. A partir destas classes de entidade, o JPA consegue fazer a persistência dos dados em banco.

Tabela 3. Comparativo dos sistemas pesquisados quanto à forma de comunicação

Trabalho	**Tipo de transmissão**	***Hardware* utilizado**	**Dispositivo de comunicação**
[Colnago 2009]	Medições iniciadas e configuradas via interface de usuário (assíncrona)	dsPIC33	Módulo Motorola G24
[Machado et al. 2008]	Medição síncrona iniciada ao ligar e transmissão assíncrona (somente em caso de alteração)	Monitor Médico	Telefone Celular (Bluetooth e GPRS)
[Tateoki 2007]	Medição síncrona iniciada ao ligar e transmissão assíncrona (presença de dados na porta serial)	PIC16F877A	Modem GSM Siemens
[Gruber 2007]	Medição síncrona iniciada ao ligar e transmissão assíncrona (somente em caso de alteração)	PIC16F877A	(2x) Modem GSM Siemens
[Vianna and Lima 2014]	Medição e transmissão síncronas (iniciadas ao ligar)	Arduino	*Shield* SIM900
Proposta	Medição e transmissão síncronas (iniciadas pelo usuário)	Arduino, PIC16F886	*Shield* SIM900

No EJB, as classes de entidade são acessadas através de *Façade*, onde cada classe *Façade* corresponde a um *web service* RESTful e os métodos destas classes podem ser acessados por comandos HTTP (POST, GET, PUT, DELETE).

Uma conexão com o métodos de envio da Entidade Leitura do *web service* inicia com um comando POST contendo as informações de quantidade de sensores e intervalo entre as amostras. A figura 3 mostra um exemplo dos cabeçalhos HTTP necessários para o envio de uma leitura via método POST.

A partir das informações recebidas, o servidor adiciona o endereço IP do cliente e a data atual em um objeto Leitura. Este objeto é persistido no banco de dados, onde um novo campo identificador (ID) é gerado. Este mesmo objeto Java, equivalente ao registro da tabela, também é retornado por HTTP na forma de um objeto JSON. Na figura 4 é exibido um exemplo da resposta ao envio das configurações para uma leitura.

```
POST /ServicoEnvioMysql/webresources/entidades.leitura HTTP/1.1
Host: 200.132.50.78
Content-Type: application/json
Content-Length: 56

{"intervaloEntreAmostras": 1080, "quantidadeSensores":1}
```

Figura 3. Comando POST para a Entidade Leitura do serviço

```
HTTP/1.1 200 OK
X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish Server Open Source Edition  4.0  J
ava/Oracle Corporation/1.8)
Server: GlassFish Server Open Source Edition  4.0
Content-Type: application/json
Date: Wed, 29 Jun 2016 06:22:07 GMT
Content-Length: 134

{"dataLeitura":"2016-06-29T03:22:07.567","idLeitura":1356,"intervaloEntreAmostr
as":1080.0,"ip":"191.247.229.4","quantidadeSensores":1}
```

Figura 4. Resposta HTTP contendo um objeto Leitura em formato JSON

Este objeto é incluído pela aplicação cliente na próxima etapa da conexão, ou seja, o envio dos dados lidos de cada sensor. O cliente deve calcular o *hash* de cada informação (sensor, valor) e enviar ao servidor, onde um novo objeto Dado é construído e o *hash* novamente é calculado. A referência ao objeto Leitura é conferida, a hora atual é adicionada e, caso o objeto seja válido, as informações são persistidas em banco e uma confirmação HTTP código 200 (OK) é enviada ao cliente.

5.2. Configurações de Rede do Servidor Web Services

Na estação de testes, o NAT (*Network Address Translation*) de um roteador ADSL (*Asymmetric Digital Subscriber Line*) foi configurado para direcionar as conexões vindas da internet para o endereço IP local do Servidor *Web Services*. O DHCP (*Dynamic Host Configuration Protocol*) deste roteador foi configurado de maneira a fornecer um endereço IP estático dentro da rede local para o Servidor Glassfish.

Também deve ser liberada a porta 8080 no *firewall* do Servidor *Web Services* para permitir o acesso de conexões de origem remota ao Servidor.

5.3. Cliente Web Services

Com a finalidade de testar os métodos disponibilizados pelos serviços de envio, foi criado um cliente HTML para o *web service* com o uso do *Framework* AngularJS.

5.4. Módulo Arduino-SIM900

Conforme previamente mencionado, o *hardware* escolhido para fazer a interface do módulo de sensores com o a rede GSM foi a placa Arduino UNO com o Shield GSM SIM900.

O desenvolvimento do *software* Arduino foi realizado com o IDE (*Integrated Development Environment*) disponível na página oficial do projeto [Arduino 2016b]. O

Arduino é conectado a um PC através de um cabo USB do tipo A/B. Essa conexão serve a dois propósitos: 1) transferência do software para a memória EEPROM (*Electrically-Erasable Programmable Read-Only Memory*) do Arduino; 2) comunicação com a interface serial RS-232 do Arduino através de um conversor USB/Serial existente na própria placa Arduino. Essa funcionalidade foi utilizada para depuração do código executado, através do envio de textos para a porta serial que são mostrados no monitor serial do IDE.

A placa Arduino UNO possui apenas uma interface RS-232 (pinos 0 e 1) sendo que esta foi utilizada para depuração do código, conforme descrito. Logo, não pôde ser utilizada para comunicação com o módulo de sensores. No entanto, o IDE de desenvolvimento disponibiliza a biblioteca SoftwareSerial [Arduino 2016c] que permite utilizar os demais pinos digitais da placa Arduino para emular em *software* o comportamento de portas seriais. Entretanto, esta biblioteca tem a limitação [Arduino 2016c] de não se poder utilizar a funcionalidade de *listening* (geração de interrupção ao receber um dado) em mais de uma porta ao mesmo tempo.

Dessa forma, utilizou-se a biblioteca SoftwareSerial para se emular duas portas seriais: uma para comunicação com o módulo de sensores (PIC) e outra para a comunicação com o *Shield* SIM900. A tabela 4 mostra como foram organizados os pinos digitais da placa Arduino UNO nesta solução.

Tabela 4. Disposição das interfaces seriais na Placa UNO

Pino	**Recepção/Transmissão**	***Hardware/Software***	**Dispositivo**
0	Recepção	Hardware	PC
1	Transmissão	Hardware	PC
2	Recepção	Software	PIC
3	Transmissão	Software	PIC
6	Recepção	Software	SIM900
7	Transmissão	Software	SIM900

A comunicação serial no Arduino funciona com um *buffer* padrão de recepção de 64 bytes. Se os dados não forem lidos antes deste espaço se esgotar, os próximos dados são perdidos. Por este motivo, implementou-se no código duas rotinas de leitura das portas seriais, uma para ler os dados do módulo de sensores e outra para ler as respostas vindas do SIM900. Essas rotinas armazenam os dados lidos em *buffers* próprios no código desenvolvido e a fim de garantir que os *buffers* seriais sejam mantidos dentro de seus limites de espaço, elas têm prioridade sobre a execução do restante do código.

Durante o desenvolvimento do trabalho, foi cogitado o uso de interrupções para o tratamento da recepção dos dados das portas seriais e controle destes *buffers*. Porém, verificou-se que isto não é necessário, pois o tratamento das interrupções correspondentes já é feito pelas bibliotecas HardwareSerial e SoftwareSerial. Como esta informação não consta na documentação, o código-fonte das bibliotecas teve de ser examinado.

Para testar a estabilidade da conexão do Arduino com o SIM900, foi feita a requisição de um array JSON ao servidor de *Webservices* na taxa de 19200 bps, porém ocorreu perda de dados, sendo então adotada uma conexão de 9600 bps, onde a leitura não

teve perdas. Na conexão do Arduino com o PIC, foi mantida a taxa de 2400bps, padrão do módulo de sensores.

Para o envio de requisições POST, foi preciso utilizar a conexão TCP ao invés de HTTP, pois a implementação do protocolo HTTP no SIM900 não permite especificar o cabeçalho "Content-Type: application/json", utilizado para o servidor interpretar corretamente o Objeto JSON enviado.

No gerenciamento da conexão do SIM900 implementou-se uma máquina de estados, conforme o comportamento definido pelo protocolo TCP do SIM900 (fig. 5).

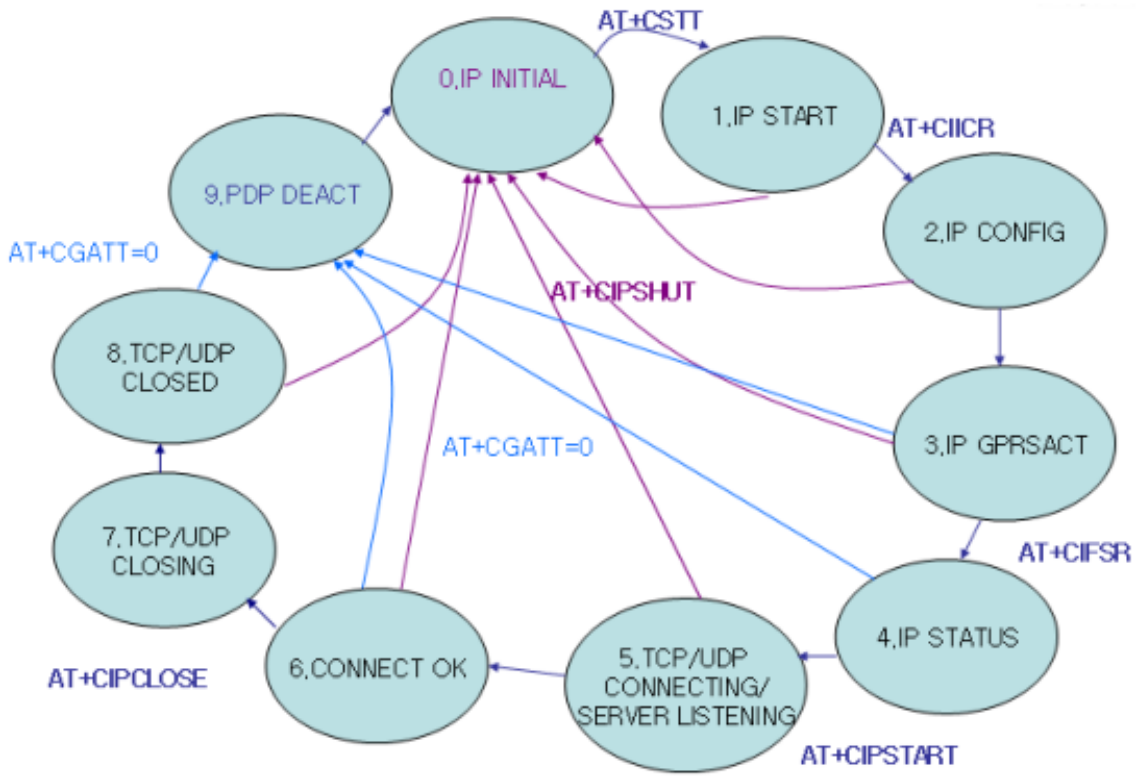

Figura 5. Estados TCP do SIM900 [SIMCom 2011]

Foi necessário o estudo do comportamento dos comandos AT do SIM900 [SIMCom 2010a] de forma a receber e tomar decisões conforme as respostas de cada comando. Inicialmente foi implementada uma solução com o uso da função *delay* e da separação das etapas da conexão por funções, porém como algumas respostas nem sempre chegavam no tempo estimado, a conexão sofria inconsistências. Ao seguir a abordagem com máquina de estados, o problema de conexão foi resolvido e, quando houver alguma falha, pode ser refeita em diferentes níveis durante a execução.

Os dados vindos do módulo de sensores são salvos no *buffer* interno do Arduino, convertidos em objetos JSON e uma conexão TCP com o Servidor Glassfish é iniciada. Dependendo da entidade a ser requisitada, a conexão com o serviço correspondente é feita e os dados enviados via método POST. Na primeira passagem pela máquina de estados, a requisição é feita para a Classe de Entidade Leitura. Após o término da requisição e do recebimento correto da resposta, as próximas requisições são feitas somente para a Classe Dado.

Na geração dos *arrays* JSON, foi testada a Biblioteca ArduinoJSON [Blanchon 2016], porém não foi possível utilizá-la no caso desta solução, onde tem-se um *array* encadeado e a passagem deste parâmetro para a função que implementa o *POST*. Ocorreu que na passagem de parâmetro, o valor chegava para a função como nulo. Foram tentados diferentes tamanhos de buffer para cada um dos *arrays* e mesmo assim o problema não foi solucionado. Concluiu-se, então, que o espaço de memória era sobrescrito durante a execução, pois a memória de dados não tinha espaço suficiente.

A abordagem para solucionar esta questão foi criar um Objeto JSON como *array* de caracteres e usar a função sprintf_P [Atmel 2016], derivada da função sprintf da biblioteca C padrão, porém com a constante de formato da *string* armazenada na memória de programa.

Esta opção tem como objetivo liberar espaço na memória de dados (restrita a 2.048 *bytes*) transferindo as constantes para a memória de programa, pois esta possui 32 KiB disponíveis [Arduino 2016a].

Visando a garantia de integridade dos valores de cada sensor enviados ao servidor, é calculada a *hash* MD5 pelo software no Arduino com o uso da Biblioteca ArduinoMD5 [Georgitzikis 2016]. Este *hash* também é calculado pelo servidor e caso estejam iguais, os dados são salvos.

Após a confirmação do servidor, a conexão é encerrada pelo cliente e o código retoma a leitura de dados do PIC.

5.5. Considerações sobre o comportamento do código durante a conexão

Para iniciar o serviço GPRS, é necessário informar o APN (*Access Point Name*) da operadora e, dependendo do caso, usuário e senha.

Após o início da conexão com o *Access Point* e da obtenção de um endereço IP local, a porta serial conectada ao módulo de sensores passa a ser lida e, após os valores serem recebidos, o fluxo da execução retorna para ler a porta serial onde está conectado o SIM900.

5.6. Módulo de Sensores

Um protótipo idêntico ao módulo de sensores foi montado em *protoboard* mas com apenas uma modificação: remoção do conversor de nível lógico MAX232, pois os microcontroladores ATmega e PIC podem ser ligados diretamente. Caso fosse utilizado o módulo de sensores original, seria necessário colocar mais um conversor de nível lógico (MAX232) também na entrada serial da placa Arduino.

Com relação ao *software*, foram efetuadas duas modificações simples para envio das configurações de taxa de amostragem e quantidade de sensores pela porta serial ao iniciar um monitoramento em cada vez que o Arduino for ligado e envio dos valores monitorados a cada leitura efetuada.

6. Conclusões

As vantagens obtidas pela integração em um projeto como este se explicam pelo aproveitamento da base de código existente e pela possibilidade de que ao se desenvolver

código para outros tipos de sensores, não seja necessária a reescrita tanto na parte de envio como na de publicação dos dados.

O método aqui apresentado poderá ser utilizado como base para elaboração de outras soluções similares, já que existem muitos dispositivos legados de aquisição de dados em que a única interface disponível é também a interface serial RS-232.

Como ponto de partida para eventuais projetos derivados, recomenda-se o uso da porta serial de *hardware* para a comunicação com o SIM900. No caso deste Projeto, esta interface teve como finalidade o monitoramento da comunicação com o propósito de depuração de código. Em novos projetos, pode ser utilizada uma serial por *software* e outra por *hardware*, evitando, assim, a perda de dados decorrente do chaveamento da leitura entre duas portas seriais por *software* se houver recepção simultânea.

As técnicas de desenvolvimento de código modularizado empregadas asseguram independência entre *hardware*, cliente de aplicação e servidor, facilitando, assim, a produção de testes e uma possível continuidade na produção de diferentes clientes de aplicação, com o uso de outros protocolos de rede ou na substituição do *hardware* intermediário. No caso de clientes de aplicação, pode-se citar como um exemplo a criação de um cliente Android também desenvolvido para visualização dos dados dos sensores.

No decorrer do trabalho ficaram evidentes as diferenças existentes nos diferentes paradigmas de programação encontrados. Desde a programação em mais baixo nível como no caso do microcontrolador PIC até os níveis mais altos de abstração como orientação a objeto e uso de *frameworks* nas linguagens Java e JavaScript.

Conclui-se que no caso das plataformas com maior capacidade de processamento, o uso de orientação a objeto e *frameworks* realmente facilitam a programação ao encapsular seu funcionamento interno e proporcionam vantagens como o maior reaproveitamento e, em teoria, uma curva de aprendizagem menos acentuada.

Porém, no caso principalmente da Plataforma Arduino, onde são implementados alguns dos princípios do paradigma orientado a objeto e uso de bibliotecas, o seu uso deve ser criterioso, pois, ao mesmo tempo em que a Plataforma é recomendada para iniciantes, em projetos com um nível de complexidade um pouco maior, estas facilidades nem sempre poderão ser utilizadas. Seja por maior uso de memória ou pela própria interação entre as bibliotecas, pois cada uma pode interferir diretamente em espaço de memória, interrupções, *timers*, etc.

Nos testes efetuados, a comunicação do Arduino com o SIM900 mostrou-se relativamente estável, com, no decorrer de algumas horas, o *shield* não retornando nenhuma resposta. Recomenda-se estipular uma verificação de tempo esgotado no envio de requisições HTTP por parte do Arduino nas situações em que isto se tornar um problema. Como este código implementa uma máquina de estados e as funções que leem as portas seriais não fazem travamento (*non-locking*), o código é capaz de retornar ao estado em que ocorreu a falha sem perder os dados que deveriam ser enviados.

Referências

Arduino (2015a). Arduino - arduinoshields. `https://www.arduino.cc/en/Main/ArduinoShields`. Acessado em 10 de outubro de 2015.

Arduino (2015b). Arduino playground - mqgassensors. http://playground.arduino.cc/Main/MQGasSensors. Acessado em 25 de novembro de 2015.

Arduino (2016a). Arduino - arduinoboarduno. https://www.arduino.cc/en/main/arduinoBoardUno. Acessado em 21 de maio de 2016.

Arduino (2016b). Arduino - software. https://www.arduino.cc/en/Main/Software. Acessado em 20 de maio de 2016.

Arduino (2016c). Arduino - softwareserial. https://www.arduino.cc/en/Reference/SoftwareSerial. Acessado em 21 de maio de 2016.

Atmel (2016). Function sprintf_p() - - avr libc reference manual. http://www.atmel.com/webdoc/AVRLibcReferenceManual/group__avr__stdio_1ga2b829d696b17dedbf181cd5dc4d7a31d.html. Acessado em 21 de maio de 2016.

Blanchon, B. (2016). bblanchon/arduinojson: An elegant and efficient json library for embedded systems. https://github.com/bblanchon/ArduinoJson. Acessado em 21 de maio de 2016.

Chinelli, M. V., Pereira, G. R., and Aguiar, L. d. (2008). Equipamentos interativos: uma contribuição dos centros e museus de ciências contemporâneos para a educação científica formal. *Revista Brasileira de Ensino de Física*, 30(4):4505–1.

Colnago, G. P. (2009). Desenvolvimento e implementação de um sistema de monitoramento em tempo real da tensão da rede com acesso remoto. Dissertação, Universidade Federal do Espírito Santo.

de Oliveira Barbosa, J., de Paulo, S. R., and Rinaldi, C. (1999). Investigação do papel da experimentação na construção de conceitos em eletricidade no ensino médio. *Caderno Brasileiro de Ensino de Física*, 16(1):105–122.

Dias, P. (2000). Hipertexto, hipermédia e media do conhecimento: representação distribuída e aprendizagens flexíveis e colaborativas na web.

Georgitzikis, V. (2016). Github - tzikis/arduinomd5: The first easily embeddable md5 library for arduino. https://github.com/tzikis/ArduinoMD5. Acessado em 21 de maio de 2016.

Gruber, V. (2007). Sistema de monitoramento remoto baseado em rede de celular gsm/gprs para gerenciamento de desgaste de pastilha de freio e vibração da torre em aerogeradores. Dissertação, Universidade Federal do Rio Grande do Sul.

Jendrock, E., Cervera-Navarro, R., Evans, I., Haase, K., and Markito, W. (2014). *The Java EE 7 Tutorial*, volume 1. Addison-Wesley Professional.

Machado, A., Padoin, E. L., Salvadori, F., Righi, L., Campos, d. M., Sausen, P. S., and Dill, S. L. (2008). Utilização de dispositivos móveis, web services e sotfware livre no monitoramento remoto de pacientes. In *CONGRESSO BRASILEIRO DE INFORMÁTICA NA SAÚDE, XI. Anais*.

Microchip (2007). *PIC16F882/883/884/886/887 Data Sheet*.

Motorola (2008). *Motorola G24 KJAVA User's Guide*.

Oracle (2015). Java persistence api. http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html. Acessado em 27 de outubro de 2015.

Sarik, J. and Kymissis, I. (2010). Lab kits using the arduino prototyping platform. In *Frontiers in Education Conference (FIE), 2010 IEEE*, pages T3C–1. IEEE.

SIMCom (2010a). *AT Commands Set SIM900_ATC_V1.00*.

SIMCom (2010b). *FTP HTTP ATCommands User Guide*.

SIMCom (2011). *SIM900 TCPIP Application Note V1.02*.

Tateoki, G. T. (2007). Monitoramento de dados via internet baseado em telefonia celular. Dissertação, Universidade Estadual Paulista.

Vianna, P. V. P. d. V. and Lima, E. A. P. d. (2014). Aplicação do arduíno no monitoramento ambiental.

XBee, D. (2013). Xbee-pro™ oem rf modules.“. *Product Manual v1. xAx-802.15*, 4.