UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

PROTÓTIPO DE LINÍGRAFO MICRO CONTROLADO

ANDRÉ ZIMMERMANN

BLUMENAU
2011

2011/2-02

ANDRÉ ZIMMERMANN

PROTÓTIPO DE LINIGRAFO MICRO CONTROLADO

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Miguel Alexandre Wisintainer, Mestre - Orientador

BLUMENAU
2011

2011/2-02

PROTÓTIPO DE LINÍGRAFO MICRO CONTROLADO

Por

ANDRÉ ZIMMERMANN

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Miguel Alexandre Wisintainer, Mestre – Orientador, FURB
Membro:	Prof. Antonio Carlos Tavares, Mestre – FURB
Membro:	Prof. Mauro Marcelo Mattos, Doutor – FURB

Blumenau, 12 de dezembro de 2011

Dedico este trabalho a todos que acreditaram na minha capacidade, mas principalmente, aqueles que sempre me apoiaram e tornaram possível a conclusão deste trabalho.

AGRADECIMENTOS

Agradeço imensamente aos meus pais, Jorge Zimmermann e Isa Zimmermann, por seu amor incondicional, nunca faltando com educação e afeto. A realização deste trabalho se deve primeiramente ao apoio deles.

Ao meu caro amigo Guilherme Faht, que tornou possível a conclusão deste trabalho.

Ao meu orientador, Miguel Alexandre Wisintainer, pelo profissionalismo e atenção durante a realização trabalho.

A minha companheira Garbareth Edianne de Mattos por seu apoio e atenção despendida que me motivou para a conclusão deste trabalho.

A empresa Windi Soluções por ter aprovado minha solicitação de férias, que sem isso, não conseguiria realizar este trabalho com tranquilidade.

RESUMO

Este trabalho descreve o desenvolvimento de um protótipo de linígrafo micro controlado que tem como objetivo principal efetuar leituras em um ponto de um curso d'água. O protótipo utiliza um sonar para adquirir os dados e os transmite com auxilio de um módulo Telit para um servidor, permitindo assim que as informações cadastradas possam ser exportadas e/ou manipuladas. Desta maneira é possível obter a vazão do curso d'água relacionando em uma equação chamada curva-chave, utilizado em estudos hidrológicos.

Palavras-chave: Linígrafo. Telit. Hidrologia. Sonar.

ABSTRACT

This work describes the development of a prototype of a micro controlled linigraph that has as main objective to perform reading a point in a water stream. The prototype uses a sonar acquire the data and transmits them with the aid of a Telit module to a server, allowing the identification information can be exported and / or manipulated. This way is possible to obtain the flow of the stream relating to an equation called turn-key, used in hydrological studies.

Key-words: Linigraph. Telit. Hydrology. Sonar

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Com a crescente preocupação do homem com o meio ambiente, os projetos de exploração dos recursos hídricos, sofreram alterações em seus objetivos. No passado, visavam o menor custo e o maior ganho para seus investidores e usuários, agora a atual tendência é o desenvolvimento com o aproveitamento racional e o mínimo dano ao meio ambiente (TUCCI, 2000, p. 26-27).

De acordo com Porto, Zahed Filho e Silva (2001, p. 5), a ciência que faz o estudo do comportamento dos recursos hídricos, chamada de Hidrologia, utiliza para estudos do ciclo hidrológico, séries históricas do escoamento superficial, importante indicador para avaliação da capacidade de navegação, estimativas de necessidades de irrigação, dimensionamento de sistemas de abastecimento de usinas hidrelétricas e drenagem, entre outras aplicações.

> O deslocamento da água na superfície da bacia, nos rios, canais, e reservatórios é uma das parcelas mais importantes do ciclo hidrológico. O escoamento é regido por leis físicas e representam quantitativamente por variáveis como vazão, profundidade e velocidade. (TUCCI et al., 1976 p. 373)

Com esta afirmação de Tucci é possível perceber a importância da vazão no processo de planejamento da utilização dos recursos hídricos. Segundo Porto, Zahed Filho e Silva (2001, p. 5), é possível relacionar as medições de vazão e nível em uma equação chamada de curva-chave. Esta equação desenvolvida por um hidrólogo, leva em consideração as características físicas e o nível da seção como variáveis da equação, sendo que as características físicas do rio são razoavelmente constantes, a única variável temporal é o nível da seção. O nível da seção é mensurado através de aparelhos chamados linígrafos (PORTO; ZAHED FILHO; SILVA, 2001, p. 5).

Atualmente o tipo de linígrafo mais utilizado no Brasil é o de bóia e contrapeso, que registra os dados em papel (SOUZA, 2003, p. 11). Este tipo de linígrafo eleva os custos da medição, pois requer papel especial e recursos humanos, já que se deve ir até o equipamento recolher e trocar a bobina de papel (RABELLO; CRUVINEL; DENARDIN, 1992, p.1).

Diante da necessidade da medição do nível da seção de forma eletrônica para transmissão dos dados para armazenamento, o principal objetivo deste trabalho é o desenvolvimento de um protótipo de linígrafo micro controlado que efetua as medições com sonar e possui telemetria por *Global System for Mobile communications* (GSM), utilizando um módulo Telit. Também será desenvolvido um *web-service* para o qual o linígrafo efetua o envio dos dados. O *web-service* permite curvas-chaves já previamente elaboradas para a

exportação da vazão, no formato *eXtensible Markup Language* (XML) de acordo com as medições já registradas.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é a construção de um protótipo de linígrafo micro controlado. Os objetivos específicos do trabalho são:

a) projetar e construir um protótipo de um linígrafo automatizado que utiliza sonar para a medição;
b) analisar a precisão da medição através de testes em campo;
c) disponibilizar um *web-service* para armazenar os dados, realizar a exportação através de um arquivo XML com um *layout* a ser definido e efetuar o cadastramento de curvas-chaves para o cálculo da vazão;
d) possibilitar o envio dos dados coletados para o *web-service* a ser implementado.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está organizado em quatro capítulos: introdução, fundamentação teórica, desenvolvimento e conclusões. O capítulo 2 apresenta a fundamentação teórica para o desenvolvimento do projeto. O capítulo 3 aborda o desenvolvimento presente do projeto, com especificações e diagramas detalhados. O capítulo 4 finaliza o trabalho apresentando as conclusões e sugestões para possíveis extensões.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda os conceitos relevantes para o desenvolvimento deste trabalho, os quais são: Linígrafos, curva-chave, módulo Telit, linguagem de programação Python, rede GSM, protocolo *Transmition Control Protocol/Internet Protocol* (TCP/IP), *Hypertext Transfer Protocol* (HTTP), software embarcado, sonar LV−MaxSonar-EZ, *web-services*, arquitetura *REpresentational State Transfer* (REST), *XML Schema*, *Java Architecture for XML Binding* (JAXB) e trabalhos correlatos.

2.1 LINÍGRAFOS

Segundo Souza (2003, p. 28), linígrafos são instrumentos que medem o nível do rio, fazendo o registro contínuo ou por amostragem. Os tipos mais comuns são os de bóia, de pressão por borbulhas, o de sensor de pressão submerso, existindo também outras formas de medição.

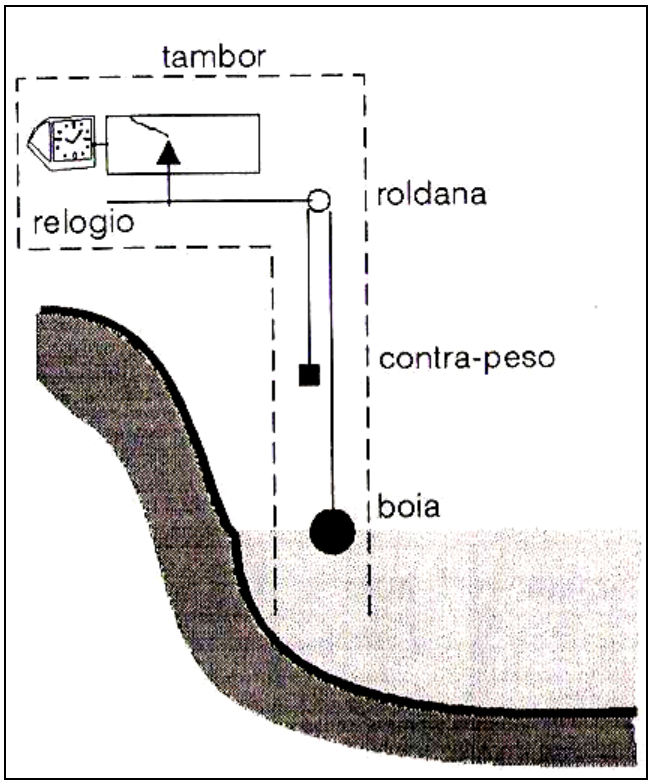
Souza (2003, p. 28) ainda descreve as réguas linimétricas, como outra forma de obter a medição da vazão; São colocadas em lances nas margens do rio, de forma que não inclinem com a correnteza (Figura 1).

Fonte: Souza (2003, p. 29).

Figura 1 - Réguas linimétricas dispostas de forma continua na margem do rio

O linígrafo de bóia, descrito por Souza (2003, p. 28-29) consiste em ter uma bóia que está ligada a uma polia e flutua acompanhando o nível do rio. Com a movimentação da bóia, a

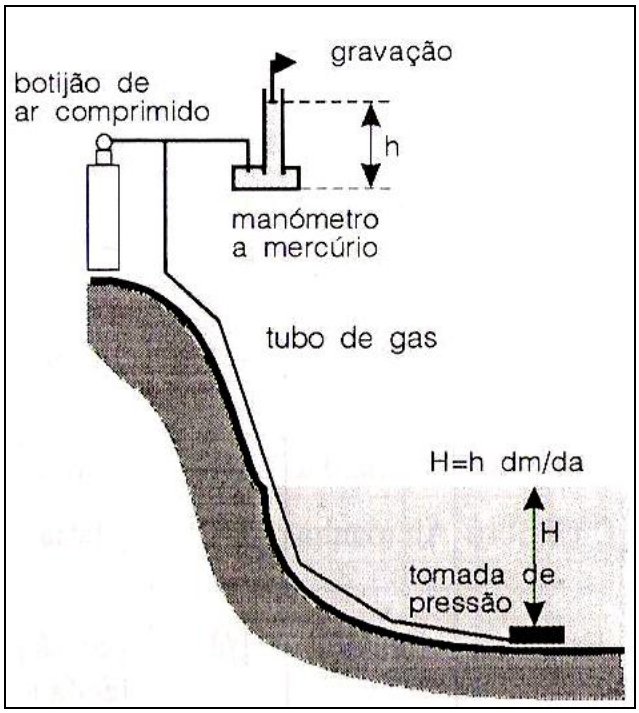
polia aciona um mecanismo que registra a nível atual em papel. Na Figura 2 pode-se observar o esquema de funcionamento deste linígrafo.



Fonte: Tucci (2000, p. 499).

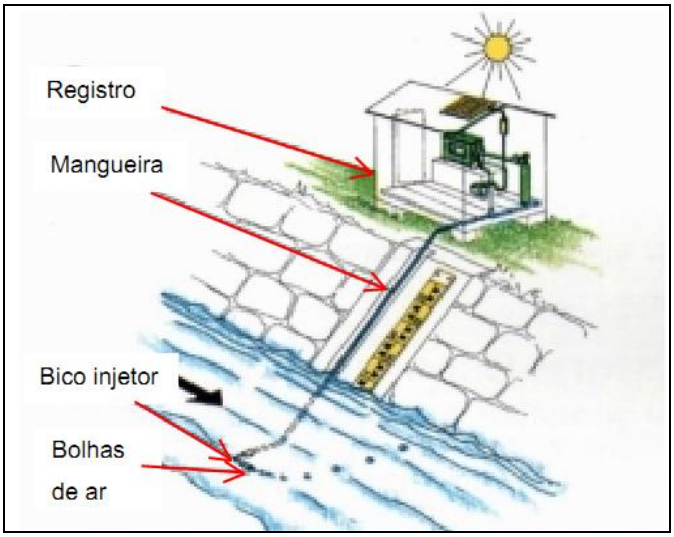
Figura 2 - Esquema do linígrafo de bóia

Porto, Zahed Filho e Silva (2000, p.30-31) afirmam que a medição do linígrafo por pressão a gás é efetuada a partir da medição da pressão no gás, alterada por uma membrana afixada no leito do rio. Essa membrana mensura a pressão da coluna d'água acima dela e reflete na pressão do gás. A pressão é mensurada através de um nanômetro de mercúrio que possibilita obter a altura da lâmina do corpo d'água (Figura 3).



Fonte: Tucci (2000, p. 499).
Figura 3 - Esquema do linígrafo com sensor de pressão a gás

Outro linígrafo descrito por Souza (2000, p. 32) é o de pressão por borbulhas. Ele consiste na medição de pressão necessária para injetar um gás abaixo do nível de água. Pela pressão mensurada é possível calcular o nível do rio, através da relação volume e pressão. Outro método de medição utiliza-se de sensores de pressão, um no fundo do rio e outro aberto a pressão atmosférica. Com esta diferença de pressão também é possível saber a altura da coluna da água. De acordo com Souza (2003, p. 34), este método é largamente utilizado na indústria para a medição de reservatórios (Figura 4).



Fonte: Porto, Zahed Filho e Silva (2000, p. 32).
Figura 4 - Esquema do linígrafo de pressão por borbulhas

Souza (2003, p. 35) ainda descreve outro método de medição, chamado de método

acústico. Um sensor que emite pulsos sonoros deve ser fixado acima do nível da superfície d'água, em altura conhecida. Para realizar a medição, inicialmente deve se conhecer a altura do nível d'água. A cada pulso é mensurado a distância entre o sensor e a superfície, com isso é possível efetuar o cálculo do nível relacionando a distância do sensor atual com o nível da água inicial.

2.2 CURVA-CHAVE

> Curva-chave é uma relação nível-vazão numa determinada seção do rio. Dado o nível do rio na seção para a qual a expressão foi desenvolvida, obtém-se a vazão. Não é apenas o nível da água que influencia a vazão: a declividade do rio, a forma da seção (mais estreita ou mais larga) também alteram a vazão, ainda que o nível seja o mesmo. (Porto, Zahed Filho e Silva, 2000, p. 5)

Porto, Zahed Filho e Silva (2000, p. 5) ainda observam que as características físicas sofrem poucas alterações ao longo do tempo, podendo ser consideradas, em uma determinada seção, constantes na curva-chave. Portanto a única variável que influencia no cálculo da vazão é o nível atual, obtido através de linígrafos, desta forma é possível relacionar a vazão e altura com a equação no Quadro 1 abaixo.

$$Q = a \cdot (H - H_0)^b$$

onde:

- a, b e H_0 são parâmetros de ajuste;
- H é o nível do rio;
- Q é a vazão;

Fonte: adaptado de Porto, Zahed Filho e Silva (2000, p. 38).

Quadro 1 - Equação da curva-chave, vazão x altura

Segundo Porto, Zahed Filho e Silva (2000, p. 39) o hidrólogo pode obter os parâmetros de ajuste descritos no quadro acima, processando um conjunto de medições de vazão em diversos níveis da seção em ferramentas iterativas.

2.3 MODULO TELIT

Segundo a Telit (2007) o módulo GE865-QUAD tem as funcionalidades necessárias para a construção de dispositivos micro controlados, que efetuam telemetria de sensores acoplados ao módulo, devido as suas capacidades de comunicação via GSM. As interfaces que existem para acoplar sensores e outros dispositivos, neste módulo, são duas interfaces seriais e pinos de *General Purpose Input/Output* (GPIO).

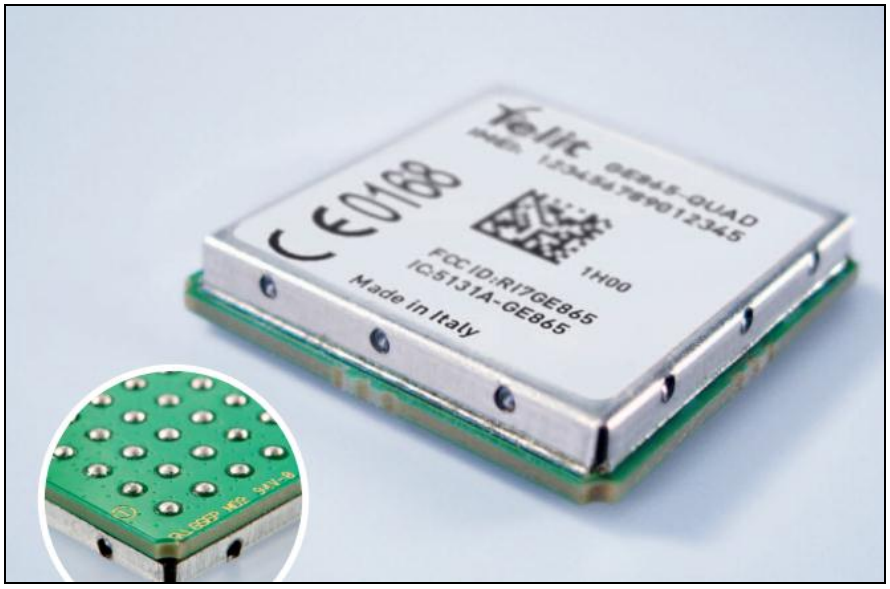
De acordo com a especificação do módulo, uma das portas seriais é compatível com RS-232 contando com os sete pinos para o controle de fluxo. As portas seriais são duas *Universal Asynchronous Receiver/Transmitter* (UART), que diferem do padrão RS-232 em nível, utilizando os níveis *Complementary Metal–Oxide–Semiconductor* (CMOS) e também diferem na polaridade, que é invertida. Com a interface principal é possível enviar *scripts* e comandos AT para o modem GSM no módulo, que são utilizados para a manipulação do modem. Este módulo ainda conta com 10 pinos de GPIO que são programáveis para leitura, escrita ou ainda uma função alternativa, dependendo do pino a ser utilizado. Com estes pinos também é possível configurar para o uso de protocolos como *Inter-Intergrated Circuit* (IIC) e *Serial Peripheral Interface* (SPI).

O módulo conta com a capacidade de interpretar *scripts* em Python. Ao programar para este módulo, pode-se contar com bibliotecas e rotinas prontas para acesso de funcionalidades deste módulo, como acessar o modem GSM ou utilizar a interface serial, garantindo maior facilidade em programar rotinas para o módulo, além de resultar em maior legibilidade ao código. Abaixo segue uma lista destas bibliotecas com uma descrição de sua funcionalidade:

a) MDM - possibilita o envio e recebimento de comandos AT para o modem interno;
b) SER - responsável pela comunicação através da porta serial do módulo. Pode ser utilizada para leitura, gerenciamento de dispositivos externos ou depuração em tempo de execução do programa;
c) GPIO - possibilita a manipulação dos pinos de entrada e saída para uso geral;
d) MOD - agrega algumas funções úteis, que manipulam o estado do módulo, para facilitar a programação.
e) IIC - utilizada para criar portas de comunicação com o protocolo I2C criadas a partir de pinos de GPIO;
f) SPI - similar a biblioteca IIC, é utilizada para criar portas de comunicação com o

protocolo SPI criadas a partir de pinos de GPIO.

O GE865-QUAD conta com as dimensões de 22mm por 22mm e 3mm de espessura, pesando apenas 3,2g, o que segundo a Telit Wireless Solutions (2010), é um dos menores módulos disponíveis no mercado. (Figura 5)



Fonte: adaptadado de Telit (2007).

Figura 5 - Módulo Telit GE865-QUAD

2.4 LINGUAGEM DE PROGRAMAÇÃO PYTHON

De acordo com Catunda (2001, p. 6), Python é uma linguagem de fácil aprendizado, que possui construções eficientes e um bom nível de abstração. É uma linguagem orientada a objetos, interativa e tipagem dinâmica. Lutz e Ascher (2001, p. 5-7) descrevem Python como uma linguagem poderosa, portável, fácil de usar, podendo ser integrada com C. Segundo Lutz e Ascher (2001, p. 5), ela fica entre linguagens interpretadas tradicionais como Perl, Tcl e Scheme por sua facilidade de uso e simplicidade e entre linguagens como C, C++ ou Java por ter características similares. O Quadro 2 demonstra a declaração de uma função que resulta na intersecção de duas listas em Python.

```
def intersect(seq1, seq2):
      res = [] # Inicia vazio
      for x in seq1: # varre seq1
            if x in seq2: # pertence a seq2?
            res.append(x) # adiciona ao resultado
      return res
```

Fonte: adaptado de Lutz e Ascher (2001, p. 306).

Quadro 2 - Declaração de uma função em Python que retorna a intersecção de duas listas

2.5 REDE GSM

Segundo Sverzut (2005, p. 59), na Europa, na década de 80, havia uma rede de celulares com um grande número de padrões analógicos incompatíveis e limitados. Com isto, as autoridades europeias sentiram a necessidade da criação e a implantação de um sistema digital de telefonia celular, surgia assim a tecnologia GSM. Sverzut (2005, p. 59) também afirma que GSM é uma tecnologia com rápida expansão. Em meados dos anos 2000 havia poucas empresas que trabalhavam com GSM, por volta dos anos 2005 havia centenas destas empresas e milhares de profissionais especializados para suprir a demanda.

Existe também o *General Packet Radio Service* (GPRS), uma extensão da rede GSM. Conforme Silva (2005, p. 50), o GPRS é um serviço de conexão a Internet, sem a necessidade de uma chamada de voz, disponibilizando assim o protocolo TCP/IP. Lopes (2008, p. 5) afirma que o GRPS torna qualquer celular ou dispositivo capaz de se conectar nesta rede através dos principais protocolos com a mesma disponibilidade que um computador convencional.

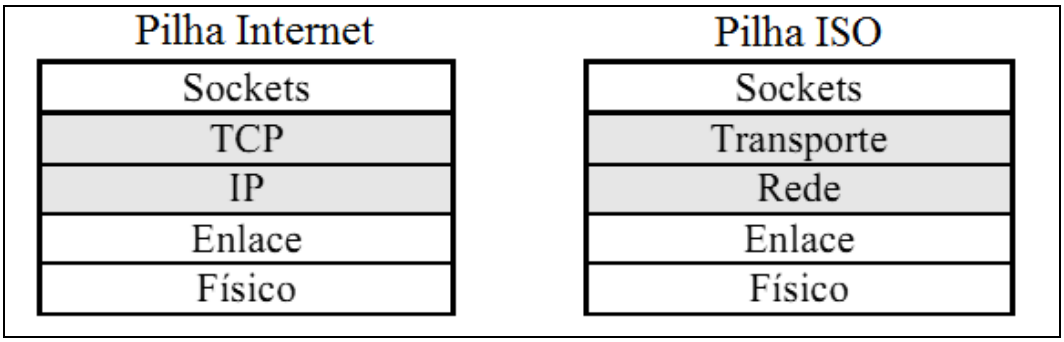
2.6 PROTOCOLO TCP/IP

"TCP/IP é na verdade o nome genérico para uma família de protocolos e utilidades também conhecido por *Internet Protocol Suite*, onde *Suite* designa uma pilha (*stack*) de protocolos" (SEIXAS FILHO, 2011, p. 2).

Parziale et al. (2006) afirmam que o principal objetivo do protocolo TCP/IP é interconectar redes de dispositivos heterogêneos, possibilitando a comunicação entre diferentes redes. O TCP/IP fornece serviços de comunicação que ficam entre a interface da

rede física e aplicação. Isto permite que as aplicações sejam independentes da camada física da rede, permitindo que os desenvolvedores da aplicação codifiquem sem a preocupação e características inerentes da camada física.

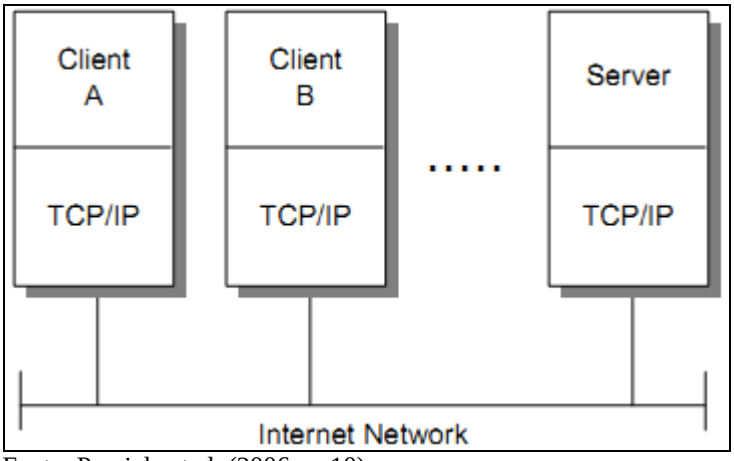
Seixas Filho (2011, p. 2) compara o TCP/IP com o padrão de protocolos *Open System Interconnection* (OSI) e afirma que os protocolos TCP e IP correspondem aos níveis de transporte e de rede. É possível observar abaixo na Figura 6, a comparação entre OSI e TCP/IP.



Fonte: adaptado de Seixas Filho (2011, p. 2).

Figura 6 - Comparação das camadas entre *Internet Protocol Suite* e OSI

Parziale et al. (2006) descrevem o protocolo TCP, responsável pela camada de transporte como sendo orientado a conexões, ponto a ponto, onde não existem relações de mestre/escravo. Entretanto as aplicações tipicamente utilizam um modelo de cliente/servidor para a comunicação (Figura 7).



Fonte: Parziale et al. (2006, p. 10).

Figura 7 - Esquema cliente/servidor sobre TCP/IP

De acordo com Parziale et al. (2006) o servidor é uma aplicação que oferece um serviço para os usuários da internet e um cliente é quem requisita estes serviços usando TCP/IP como um veículo de transporte. O servidor normalmente é capaz de processar e responder diversos clientes ao mesmo tempo.

2.7 PROTOCOLO HTTP

De acordo com Hilgenstieler (2003, p. 19), o protocolo HTTP é um protocolo do nível de aplicação que suporta sistemas de informação distribuídos, cooperativos e de hipermídia. As principais características são:

a) propicia busca de informações e atualizações;
b) envia mensagens em um formato similar aos utilizados pelo correio eletrônico da internet e pelo *Multipurpose Internet Mail Extensions* (MIME);
c) possui comunicação entre os agentes usuários e gateways, permitindo acesso a hipermídia e a diversos protocolos do mundo internet;
d) obedece ao paradigma de cliente-servidor, isto é, um cliente estabelece uma conexão com um servidor e envia um pedido, o qual o analisa e responde.

Os principais métodos disponíveis:

a) *GET*: busca as informações associadas com um recurso da rede;
b) *HEAD*: similar ao *GET*, mas não transfere dados referentes ao recurso utilizado para verificar a validade do recurso;
c) *POST*: envia dados para o servidor para processamento no corpo do comando;
d) *PUT*: cria uma nova ou atualiza uma entidade já existente do recurso especificado no pedido;
e) *DELETE*: solicita que o servidor apague o recurso da rede identificado no *Uniform Resource Identifier* (URI);
f) *LINK*: estabelece uma ou mais relações de links entre o recurso identificado pelo URI e outros recursos existentes, não permitindo que o corpo da entidade enviada seja subordinada ao recurso;
g) *UNLINK*: remove uma ou mais relações de links existentes entre o recurso identificado no URI.

2.8 SISTEMA EMBARCADO

Segundo Taurion (2007) sistema embarcado é um software embutido dentro de um dispositivo eletrônico, como o sistema de injeção eletrônica de um automóvel, permitindo que

este equipamento atue com maior funcionalidade e flexibilidade.

Nos últimos anos tem-se visto uma crescente utilização de softwares embarcados em praticamente todos os objetos eletrônicos construídos pelo homem. Antes apenas utilizados em sistemas complexos como sistemas industriais, aeronaves e navios, hoje já existem softwares embarcados em geladeiras, televisores e fornos de micro-ondas. Estes equipamentos tornam-se cada vez mais sofisticados, demandando maior complexidade no hardware e software embarcado (TAURION, 2007).

2.9 SONAR LV−MAXSONAR-EZ

A Acroname Robotics (2011) explica que o funcionamento destes sensores consiste em inicialmente enviar um pulso ultrassônico. Normalmente a frequência do pulso é de 40KHz, inaudível para o ouvido humano, que pode escutar sons entre 20Hz a 20KHz. Este pulso viaja a uma velocidade aproximada de 343,2 m/s e quando ele encontra um obstáculo, é refletido e volta para o sensor que, no momento, está esperando pelo pulso enviado. Com o tempo de voo do pulso e a velocidade dele é possível calcular a distância do sensor até o obstáculo.

O sensor LV-MaxSonar EZ1 utiliza sonar para efetuar as medições. Segundo Tato (2005), ele detecta obstáculos de 0 a 6.45m. É possível obter o valor da medição de três formas: largura de pulso, voltagem analógica e serial digital. O LV-MaxSonar EZ1 opera em na frequência de 42KHz, podendo efetuar medições a cada 50ms e requerendo uma alimentação entre 2.5V e 5.5V (Figura 8).

Fonte: MaxBotix (2011).

Figura 8 - Sensor sonar LV-MaxSonar da MaxBotix

2.10 WEB-SERVICE

> [...] podemos compreender os Web Services como uma combinação de protocolos e padrões que permitem a aplicações interagirem entre si através da Internet. Esta comunicação é facilitada através de uma descrição de como acessar a aplicação e das funcionalidades que a mesma oferece. (GARCIA, 2007, p. 14-15)

Para Vieira (2007, p. 14), *web-services* nada mais são que pedaços de programas oferecendo serviços, que recebem parâmetros e retornam dados processados, disponibilizados na internet. A aplicação cliente requisita o *web-service* através de uma *Universal Resource Locator* (URL), usando, sobre o *HyperText Transfer Protocol* (HTTP), o protocolo de comunicação *Simple Object Access Protocol* (SOAP), *Web Services Description Language* (WSDL) ou *Restful*.

2.10.1 Arquitetura *Restful*

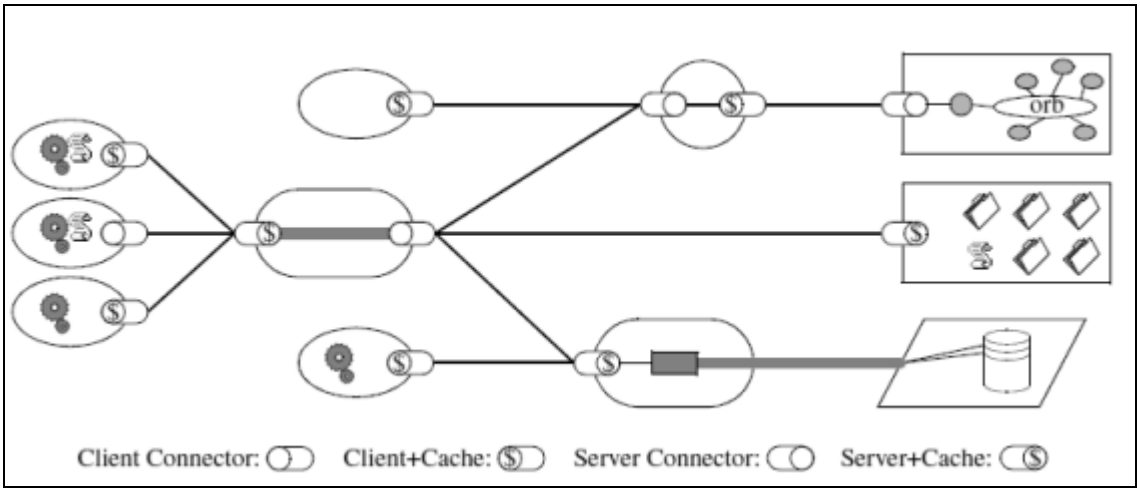
Segundo Feller (2010, p. 13), devido a complexidade apresentada pelos padrões SOAP e WSDL, foi desenvolvido o modelo REST, utilizando os padrões básicos como HTTP, URL e XML. Com este modelo se torna simples disponibilizar *web-services* que podem ser utilizados tanto por humanos quanto por máquinas.

De acordo com Feller (2010, p 14), este estilo arquitetônico possui restrições que permitem reforçam a escalabilidade, generalidade de interfaces, implantação independente de componentes, reforçar a segurança e encapsular sistemas legados, sendo as mais importantes citadas abaixo:

a) cliente-servidor: é possível melhorar a portabilidade da interface de usuário e melhorar a escalabilidade com a separação dos interesses;
b) comunicação sem estado (*stateless*): cada requisição do cliente para o servidor de deve contar toda a informação necessária para seu entendimento;
c) *cache*: para aumentar a eficiência do uso da rede, é possível efetuar o *cache* no cliente. Se uma resposta do servidor estiver marcada como "*cacheable*" o cliente tem o direito de reusá-la posteriormente para requisições equivalentes;
d) interface uniforme: através das aplicações do princípio às interfaces dos componentes, é possível simplificar a arquitetura do sistema;
e) sistema em camadas: permitindo que cada componente não veja além das camadas

com quem se comunica, promove independência de subsistemas e diminui a complexidade do sistema.

Feller (2010, p. 15) demonstra a aplicação destes elementos na Figura 9:



Fonte: Feller (2010, p. 15).

Figura 9 - Demonstração do estilo arquitetural REST

2.11 XML SCHEMA

De acordo com Oliveira (2006, p. 19), a *XML Schema*, for proposta pela *world wide Web Consortium* (W3C), em 1999. É uma linguagem XML usada para descrever e validar o conteúdo de documentos XML. Oliveira afirma que basicamente, a *XML Schema* utiliza uma sintaxe baseada em XML para descrever as possíveis estruturas e elementos com seus tipos de dados, presentes em um documento XML. A sua principal característica que diferencia de outras linguagens para descrever documentos XML, é que suporta conceitos de orientação objetos, como polimorfismo e herança.

Ruela (2008, p. 21) cita as principais características do *XML Schema*:

a) define elementos e atributos que podem aparecer em um documento;
b) define que elementos são elementos filhos;
c) define a ordem dos elementos filhos;
d) define tipos de dados para elementos e atributos;
e) define valores padrão e fixos para elementos e atributos.

Na linguagem *XML Schema*, se trabalha com dois tipos de elementos: elementos simples e elementos compostos. Elementos simples podem ser definidos com a utilização da

marcação *element* seguido dos atributos *name* e *type*, que representam o nome e o tipo do elemento. (OLIVEIRA, 2006, p. 20). Os elementos compostos são declarados com a marcação *complexType* e podem conter outros elementos e atributos. No Quadro 3 é possível visualizar um exemplo de código em *XML Schema*.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="cadastro" >
  <xs:complexType>
    <xs:sequence>
        <xs:element  name = "nome"  type ="xs:string"
      minOccurs ="1"/>
        <xs:element  name = "fone"  type ="xs:string"
      minOccurs ="0" maxOccurs="2"/>
        <xs:element  name = "empresa"  type ="xs:string"/>
        <xs:element  name = "cargo"  type ="xs:string"
      maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  </xs:element>
</xs:schema>
```

Fonte: adaptado de Oliveira (2006, p. 21).

Quadro 3 - Código em *XML Schema* demonstrando elementos simples e complexos

2.12 JAVA ARCHITECTURE FOR XML BINDING

JAXB disponibiliza uma API e uma ferramenta que permite mapeamento bidirecional entre documentos XML e objetos JAVA. A partir de um *XML Schema* o compilador JAXB pode gerar um conjunto de classes JAVA que permite desenvolvedores ler, manipular e recriar documentos XML sem escrever nenhuma lógica para estes processos. (LIN, 2002)

De acordo com Simion (2008), as principais vantagens de se trabalhar com JAXB são:

a) esconde os detalhes do processamento dos dados;
b) oferece uma modalidade de trabalhar orientado a objetos, com documentos XML;
c) geração de classes através de um compilador que analisa um ou mais *XML Schemas*.

Lin (2002) descreve os seguintes passos para ler, alterar e salvar um documento XML com JAXB:

a) descrever o documento XML em *XML Schema*;
b) processar esta *XML Schema* com o compilador JAXB;

c) desenvolver o código necessário para obter o documento XML;

d) utilizar a API da JAXB para efetuar o *unmarshalling;*

e) manipulação dos objetos instanciados pela API da JAXB;

f) utilizar a API da JAXB para efetuar o *marshalling.*

Durante o *marshal* e *unmarshalling* é possível utilizar a API da JAXB para validar a consistência de acordo com o *XML Schema*.

2.13 TRABALHOS CORRELATOS

Para Rabello, Cruvinel e Denardin (1994, p. 1), a aquisição e armazenamento de forma digital dos dados colhidos por linígrafos apresentam grandes vantagens em relação aos instrumentos analógicos. Os autores desenvolveram um linígrafo de bóia, micro-controlado, utilizando o 80C31, da família 8051 e programado em C. A obtenção dos dados era somente por uma interface serial, compatível o padrão RS-232C. O linígrafo tinha uma limitação muito grande para um linígrafo de bóia, o seu sensor, que capturava o movimento da boia, tinha uma faixa útil de zero até 255 milímetros.

A OTT Hydrometry líder de mercado em instrumentos para a hidrometria, oferece alguns produtos para medição do nível da água. Entre eles, destacam-se o Thalimedes, um linígrafo de bóia e o *Radar Level Sensor* (RLS), que utiliza radar para mensurar o nível da água. Segundo a OTT Hydrometry (1996, p. 2), Thalimedes possui armazenamento interno para até trinta mil medições. O sensor possui uma faixa útil de aproximadamente sessenta metros. O Thalimedes não possui modem GSM integrado, mas oferece suporte para integração através de duas interfaces seriais, uma conforme o padrão RS-232C e outra com o protocolo SDI-12. Conta também com a possibilidade de conexão com um computador através de infravermelho.

A OTT Hydrometry (2007, p. 2), descreve o método de medição do RSL como sendo acústico, utilizando pulsos de radar. Na Figura 10 há um exemplo do RLS em campo.

Fonte: OTT Hydrometry (2007).

Figura 10 - RSL instalado abaixo de uma ponte

De acordo com a OTT Hydrometry, este linígrafo não possui armazenamento interno e deve ser conectado a um *datalogger* por interface SDI-12 ou por RS-485. A OTT Hydrometry (2007, p. 3), cita como principal característica o baixo consumo de energia do RSL como uma grande vantagem do instrumento.

A Global Waters (2011, p. 5), também se destaca na venda instrumentos para hidrometria. O WL15 da Global Waters é um linígrafo com *datalogger*, com capacidade de registrar até 24400 registros. Ele utiliza uma interface serial, compatível com o padrão RS-232C, para se conectar a um computador e seu sensor é um transdutor de pressão que transforma a pressão em sinais eletrônicos, desta forma ela mensura o nível de água do sensor até a superfície. Com o software da Globar Waters (2011, p. 24), desenvolvido para o WL15 é possível calibrar o sensor para efetuar medições de até 6,35 metros e recuperar as informações contidas no *datalogger*.

3 DESENVOLVIMENTO

Neste capítulo são detalhados os requisitos do sistema, a especificação do hardware e software, as implementações realizadas e resultados obtidos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O protótipo de linígrafo deverá:

a) adquirir a informação da distância do protótipo até o nível de água e armazenar na memória interna (Requisito Funcional - RF);
b) enviar em intervalos de tempo previamente configurados, as informações das medições salvas na memória volátil para o *web-service* e apagar estes dados armazenados se a transferência for bem sucedida (RF);
c) permitir a configuração da quantidade de medições armazenadas para o envio e o intervalo de tempo entre as medições;
d) deve ser programado em Python (Requisito Não-Funcional - RNF);
e) deve usar sonar para efetuar a medição da distância (RNF);
f) deve ser identificado pelo *International Mobile Equipment Identity* (IMEI) (RNF).

Quanto ao *web-service*, ele deverá:

a) permitir o cadastro de linígrafos, com identificador único e local de instalação (RF);
b) receber as informações de medições enviadas pelos linígrafos (RF);
c) exportar, em formato XML, as informações de medições enviadas pelos linígrafos, referentes a um período (RF);
d) permitir o cadastro de várias curvas chaves para possibilitar o cálculo da vazão de diferentes corpos d’água (RF);
e) as informações recebidas devem ser armazenadas em um banco de dados MySQL, para manipulação e consulta das informações (RNF);
f) deve ser implementado em Java (RNF).

3.2 ESPECIFICAÇÃO

O trabalho desenvolvido se divide em duas partes distintas, a primeira parte a ser especificada é o hardware do protótipo. A segunda parte é dividida na especificação do software embarcado no Telit e da especificação do *web-service*, responsável por armazenar e processar os registros das medições.

3.2.1 Hardware

A principal funcionalidade do hardware é efetuar a leitura da distância do protótipo, até a superfície d'água, através de um sonar. As medições obtidas são armazenadas, para serem enviadas, através de uma conexão GPRS, ao *web-service* para posterior consulta e processamento. O módulo Telit é responsável por gerenciar, ler as informações de medição do sonar e efetuar a conexão GPRS para se comunicar com o *web-service*. Tanto o sonar, quanto o módulo Telit, são alimentados por uma bateria (Figura 11).

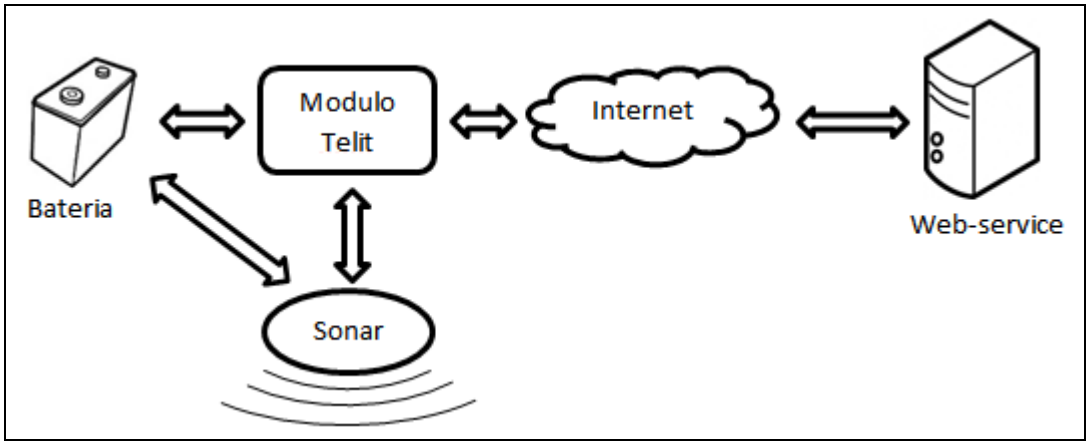


Figura 11 - Diagrama de ligação dos principais componentes do hardware

Para a construção do protótipo de linígrafo proposto foi utilizado o Telit GE865-QUAD, em uma placa de desenvolvimento específica para o módulo Telit, ambos fornecido pela empresa Sparkfun (2011) e um módulo de sonar MB1010 LV-MaxSonar-EZ1, fornecidos pela empresa MaxBotix (2011). Tanto o sonar, quanto a placa da Sparkfun, foram afixadas em uma placa universal, com 10cm x 10cm de tamanho, para facilitar a ligação entre o módulo Telit, módulo sonar e demais componentes necessários para o correto funcionamento do protótipo. Na Figura 12 pode-se visualizar o esquema eletrônico do protótipo.

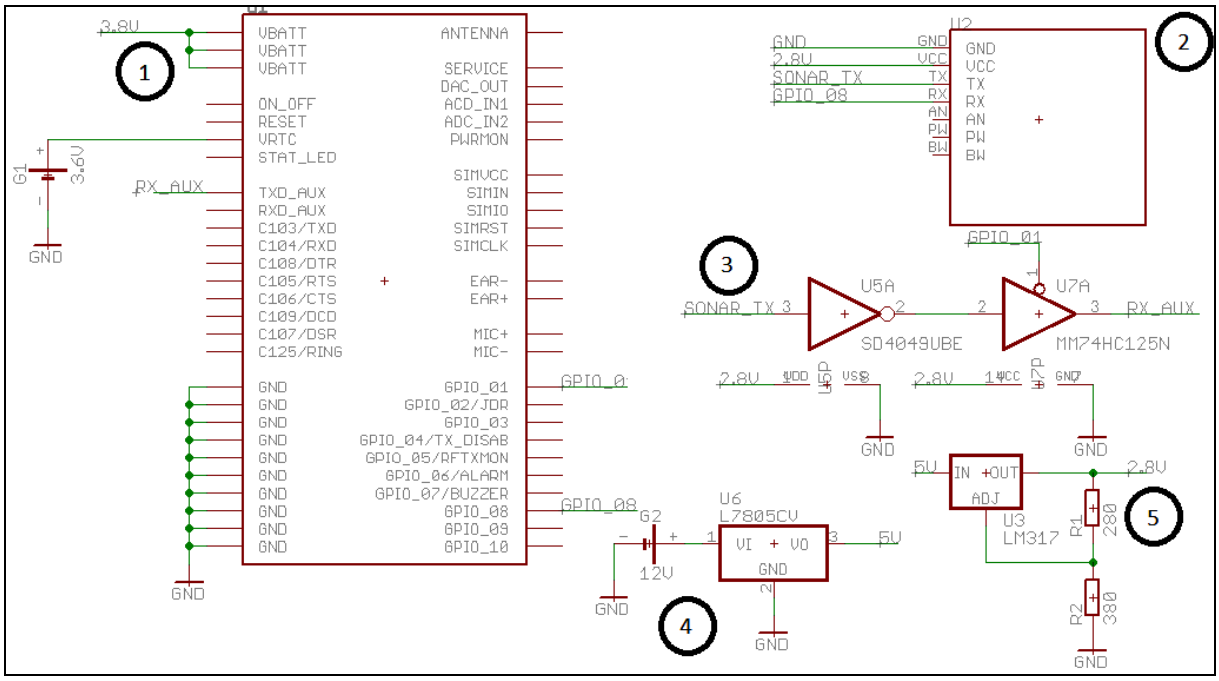


Figura 12 - Esquema eletrônico do protótipo

As numerações inseridas na Figura 12 são para a explanação dos circuitos integrados existentes e suas ligações. No número um é possível observar o GE865-QUAD. Sua alimentação é proveniente do regulador de tensão SPX29302, não demonstrado neste esquema eletrônico, pois está especificada na documentação da placa de desenvolvimento da Sparkfun. Os outros circuitos que fazem parte desta e que não foram ligados com outros componentes do protótipo, também foram abstraídos deste esquema eletrônico. Pode-se visualizar o módulo sonar MB1010 LV-MaxSonar-EZ1 (número dois), alimentado pelo regulador de tensão (número cinco).

O sonar MB1010 foi ligado com o módulo Telit através do circuito número três, este sendo necessário devido a uma peculiaridade descrita no manual do módulo Telit. O manual descreve que, nas seriais a polaridade de sinal é invertida, portanto para realizar a comunicação é inserida uma porta lógica de negação, proveniente do circuito integrado SD4049UBE. Entretanto, devido às características da porta lógica de negação e a existência de outra peculiaridade do módulo Telit, também descrita no seu manual, faz-se necessário o uso de um *tri-state*, presente no circuito integrado MM74HC125N. Esta peculiaridade este associado ao fato de que no momento do módulo ser ligado, ele estiver com um valor lógico alto em qualquer uma de suas GPIO, ele não liga. O último circuito a ser explanado é número quarto, é o regulador de tensão utilizado para alimentar o regulador de tensão SPX29302 e o circuito demonstrado no número 5, regulando os doze volts da bateria para os cinco volts.

3.2.2 Software

A especificação do software embarcado e *web-service*, foi realizada através de diagramas da Unified Modeling Language (UML), utilizando a ferramenta Enterprise Architect. As especificações são apresentadas nas subseções seguintes.

3.2.2.1 Software embarcado

Para a especificação do software embarcado, foram utilizados os diagramas de casos de uso, de atividades e de classes. Estes diagramas são descritos nas subseções abaixo.

3.2.2.1.1 Diagrama de casos de uso do software embarcado

A Figura 13 abaixo apresenta os casos de uso envolvidos com o software embarcado.

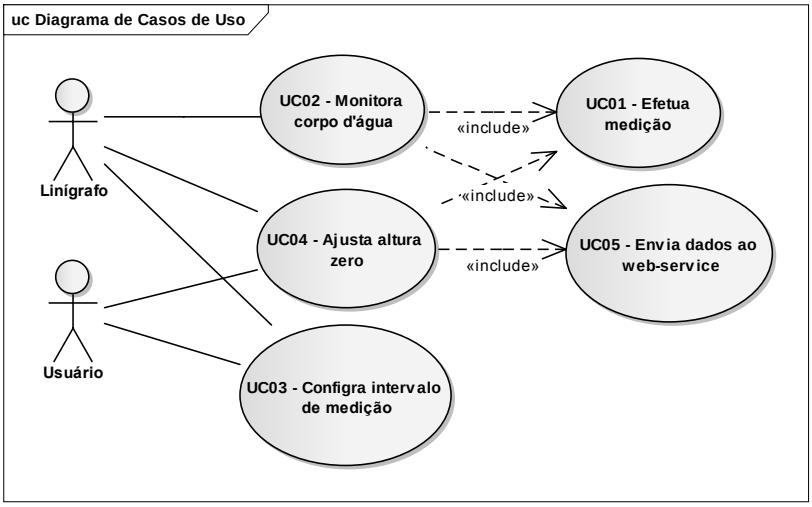


Figura 13 - Diagrama de casos de uso do software embarcado

Os casos de uso são descritos a seguir:

a) UC01 - efetua a medição: permite o protótipo efetuar a leitura da distância a partir de um sonar;
b) UC02 - monitora corpo d'água: protótipo em um intervalo de tempo determinado efetua uma medição e armazena o registro. O protótipo de línigrafo envia a medição após um determinado número de medições;

c) UC03 – ajusta altura zero: protótipo de línigrafo efetua uma medição e atualiza este valor no *web-service*;
d) UC04 - envia dados ao *web-service*: protótipo cria uma conexão com o *web-service* e envia dados pertinentes a requisição desejada;
e) UC05 - configura intervalo de medição: permite ao usuário configurar o intervalo de medição e o número de requisições armazenadas no protótipo.

3.2.2.1.2 Diagrama de atividades do software embarcado

O diagrama de atividades é responsável por representar os estados de uma aplicação.

A Figura 14 apresenta os passos disponíveis no software embarcado para que o monitoramento seja realizado.

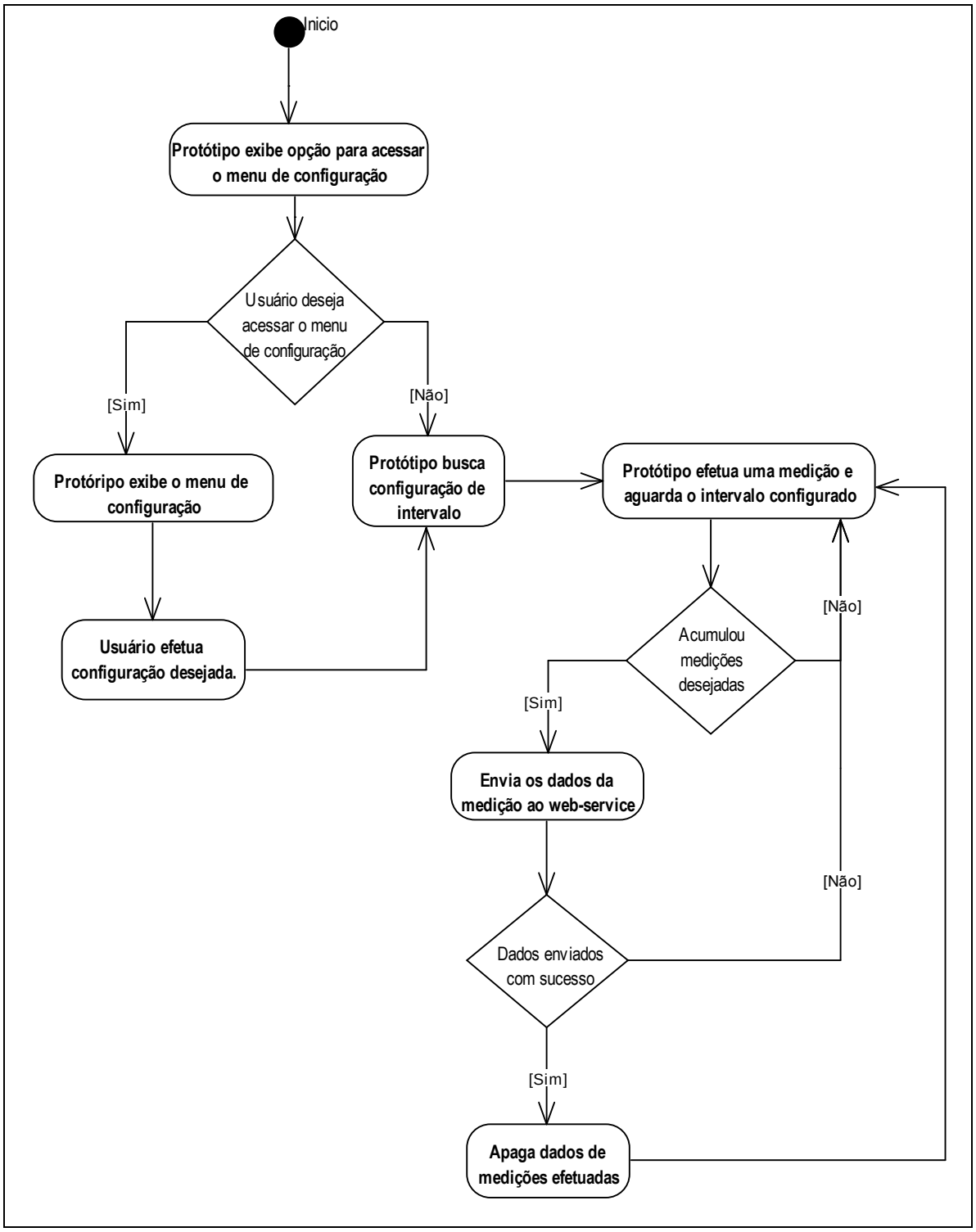


Figura 14 - Diagrama de atividades do software embarcado

Ao ligar o protótipo, é exibida para o usuário a opção para acessar o menu de configuração, se este estiver conectado ao protótipo através do *HyperTerminal*. Caso o usuário deseje acessar o menu, ele é exibido pelo protótipo, onde o usuário efetua as configurações necessárias. Ao finalizar as configurações ou se o usuário não desejar efetuar a configuração do protótipo, o protótipo busca as configurações de intervalo salva na memória interna do protótipo.

Após o processo inicial de configuração e inicialização das variáveis, o protótipo segue efetuando as medições e aguardando o intervalo configurado. Após este intervalo ele verifica se existem medições suficientes para o envio ao *web-service*. Em caso positivo, o protótipo envia as medições existentes ao *web-service*. O protótipo determina se houve sucesso ao enviar as medições. Se não ocorreram problemas durante o envio, ele apaga as medições enviadas de sua memória e segue com novas medições. Caso tenha ocorrido problemas durante o envio, ele mantem armazenadas na memória estes dados e segue novamente a efetuar as medições. O motivo de não ter um nodo final neste diagrama é porque não existem condições para o protótipo suspender o processo das medições, exceto a falta de bateria ou o desligamento do protótipo.

3.2.2.1.3 Diagrama de classes do software embarcado

A Figura 15 abaixo apresenta o diagrama de classes do software embarcado.

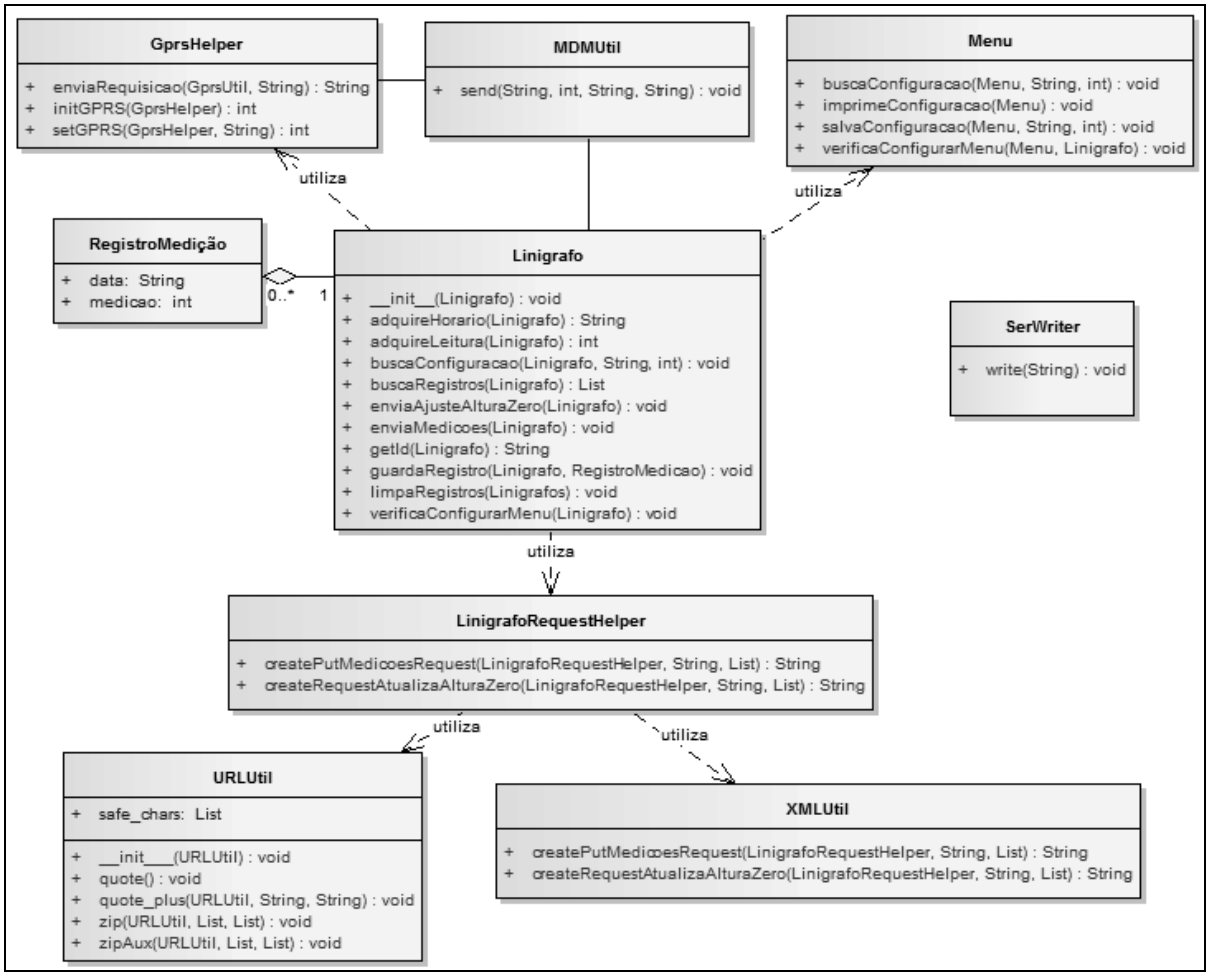


Figura 15 - Diagrama de classes do protótipo

A classe `Linígrafo`, modela o protótipo, coordenando o funcionamento do mesmo. Esta classe possui um relacionamento com a classe `RegistroMedição`, responsável por agrupar os dados da medição. Durante a inicialização do protótipo a classe `Linígrafo` instancia um objeto da classe `Menu`, responsável por exibir e controlar o menu de configurações. A classe `LinigrafoRequestHelper` tem a responsabilidade de montar as requisições HTTP para o envio ao *web-service*, utilizando as classes `URLUtil` e `XMLUtil`. Ambas as classes são utilitárias, sendo que a classe `URLUtil` é responsável por substituir caracteres inválidos das requisições HTTP. A classe `XMLUtil` é responsável por receber objetos da `RegistroMedição` e transformar em um documento XML que condiz com a especificação do *XMLSchema*, correspondente com a requisição desejada no momento.

A classe `GprsHelper` é a classe responsável por comandar e controlar a conexão com o *web-service*, enviando os dados necessários, solicitados pela classe `Linígrafo`. A classe `MDMHelper` e `SerWriter`, são classes voltadas para a manipulação do módulo Telit. A classe `SerWriter` é responsável por imprimir caracteres na serial principal do modulo, permitindo a depuração e comunicação com o usuário. A classe `MDMHelper` recebe comandos para o módulo e os envia, analisando sua resposta.

3.2.2.2 *Web-service*

Para a especificação do *web-service*, foram utilizados: diagrama de casos de uso; diagrama de pacotes; diagrama de classes; diagramas de atividades; diagramas de sequencia.

3.2.2.2.1 Diagrama de casos de uso do *web-service*

A Figura 16 abaixo apresenta os casos de uso relacionados ao *web-service*.

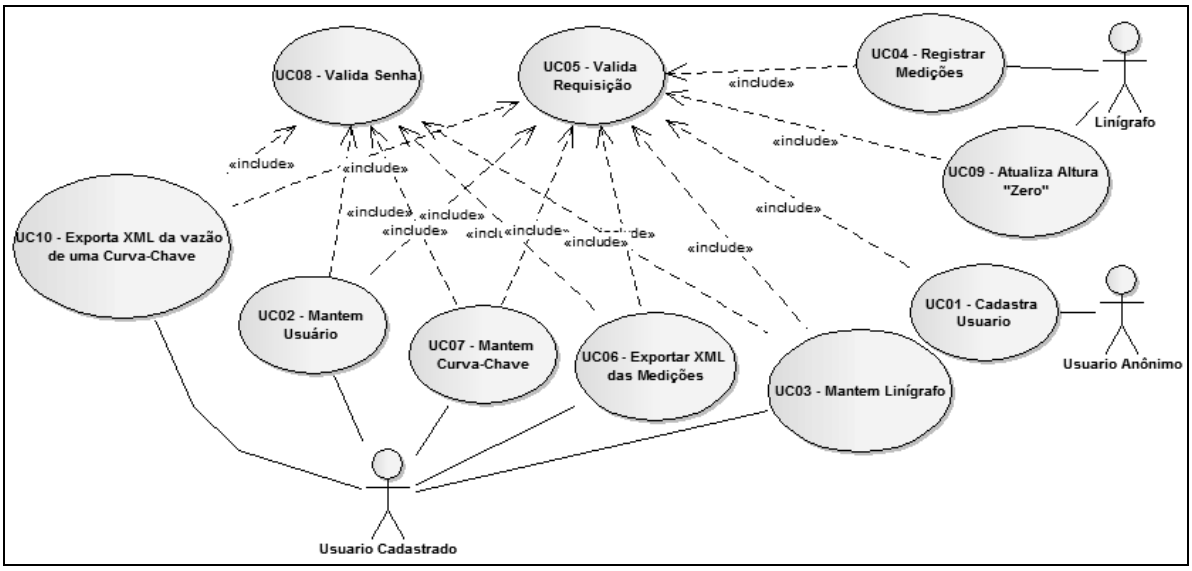


Figura 16 - Diagrama de casos de uso do *web-service*

Os casos de uso do *web-service* são descritos a seguir:

a) UC01 – cadastra usuário: permite ao *web-service* receber requisições para cadastrar um novo usuário;

b) UC02 – mantem usuário: permite ao *web-service* receber requisições para alterar os dados de um usuário já cadastrado;

c) UC03 – mantem linígrafo: permite ao *web-service* receber requisições para cadastrar um novo linígrafo ou atualizar um linígrafo previamente cadastrado;

d) UC04 – registrar medições: permite ao *web-service* receber requisições para registrar medições enviadas de um linígrafo previamente cadastrado no sistema;

e) UC05 – valida requisição: permite ao *web-service* validar as requisições enviadas;

f) UC06 – exportar XML das medições: permite ao *web-service* receber requisições para a exportação de documentos XML com nível do corpo d'água referente aos linígrafos e um período específico, determinados na requisição;

g) UC07 – mantem curva-chave: permite ao *web-service* receber requisições para cadastrar, atualizar ou remover uma curva-chave na base de dados;

h) UC08 – valida senha: permite ao *web-service* exigir autenticação durante o recebimento das requisições, validando com os usuários cadastrados;

i) UC09 – atualiza altura zero: permite ao *web-service* receber requisições de um linígrafo previamente cadastrado para o ajuste da altura zero, utilizada como referencial durante o cálculo do nível do corpo d'água;

j) UC10 - exporta XML da vazão de uma Curva-Chave: permite ao *web-service* receber requisições para a exportação de documentos XML com a vazão do corpo d´água a uma curva-chave, associando a um linígrafo e um determinado período.

3.2.2.2.2 Diagrama de pacotes do *web-service*

Descreve as dependências entre os pacotes que formam o modelo do sistema (Figura 17).

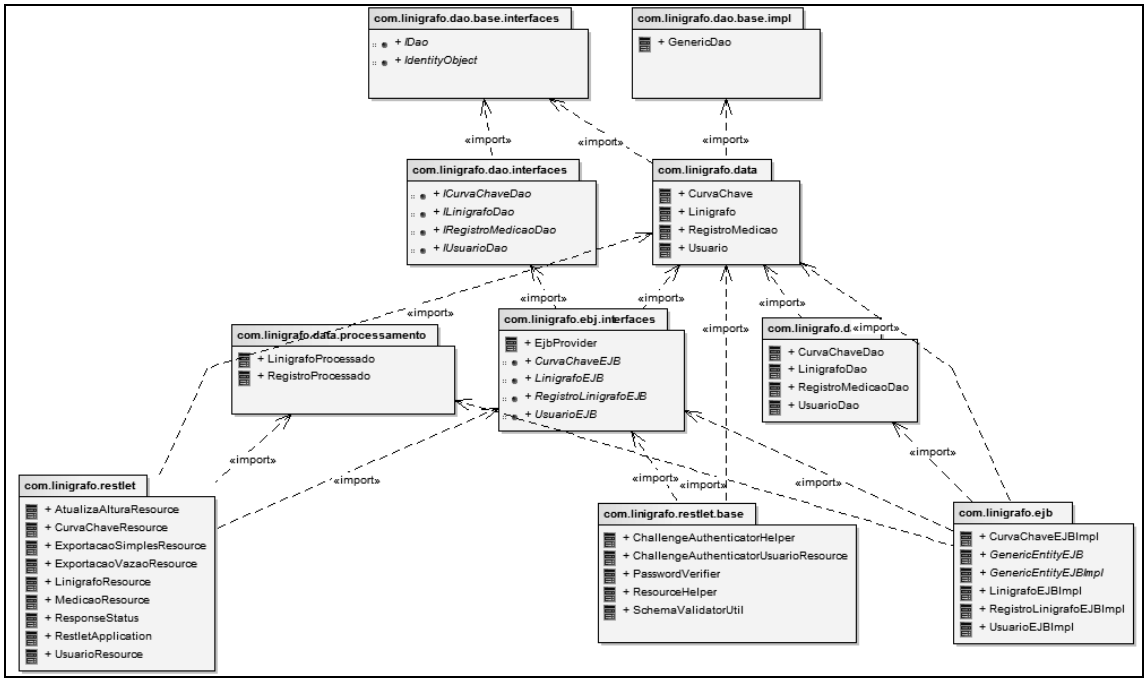


Figura 17 - Diagrama de pacotes do *web-service*

O pacote `com.linigrafo.restlet` é responsável por conter as classes que recebem e tratam as requisições, sendo que o pacote `com.linigrafo.restlet.base` contém classes que auxiliam e possuem rotinas em comum para o processamento destas requisições. O pacote `com.linigrafo.ejb.interfaces` contem as interfaces para a implementação de classes *Enterprise JavaBeans* (EJB). No pacote `com.linigrafo.ejb` se encontram as implementações destas interfaces. Os pacotes `com.linigrafo.dao` e `com.linigrafo.dao.interfaces` contém as classes e interfaces necessárias para acesso ao banco de dados. O pacote `com.linigrafo.dao.base.interfaces` possui as interfaces que ditam o funcionamento básico da persistência dos dados do *web-service*. O pacote `com.linigrafo.dao.base.impl` possui uma classe que é responsável por implementar e agrupar as funcionalidades básicas para o acesso ao banco de dados. Os pacotes `com.linigrafo.data` e `com.linigrafo.data.processamento` agrupam classes que modelam entidades utilizadas no sistema.

3.2.2.2.3 Diagramas de classes do *web-service*

Responsável por fornecer o panorama geral dos relacionamentos entre classes do *web-service*, o diagrama de classes foi particionado para facilitar o entendimento da estrutura construída. Na Figura 18, é possível visualizar um diagrama de classes que representa as classes necessárias para o recebimento de requisições e verificação da senha.

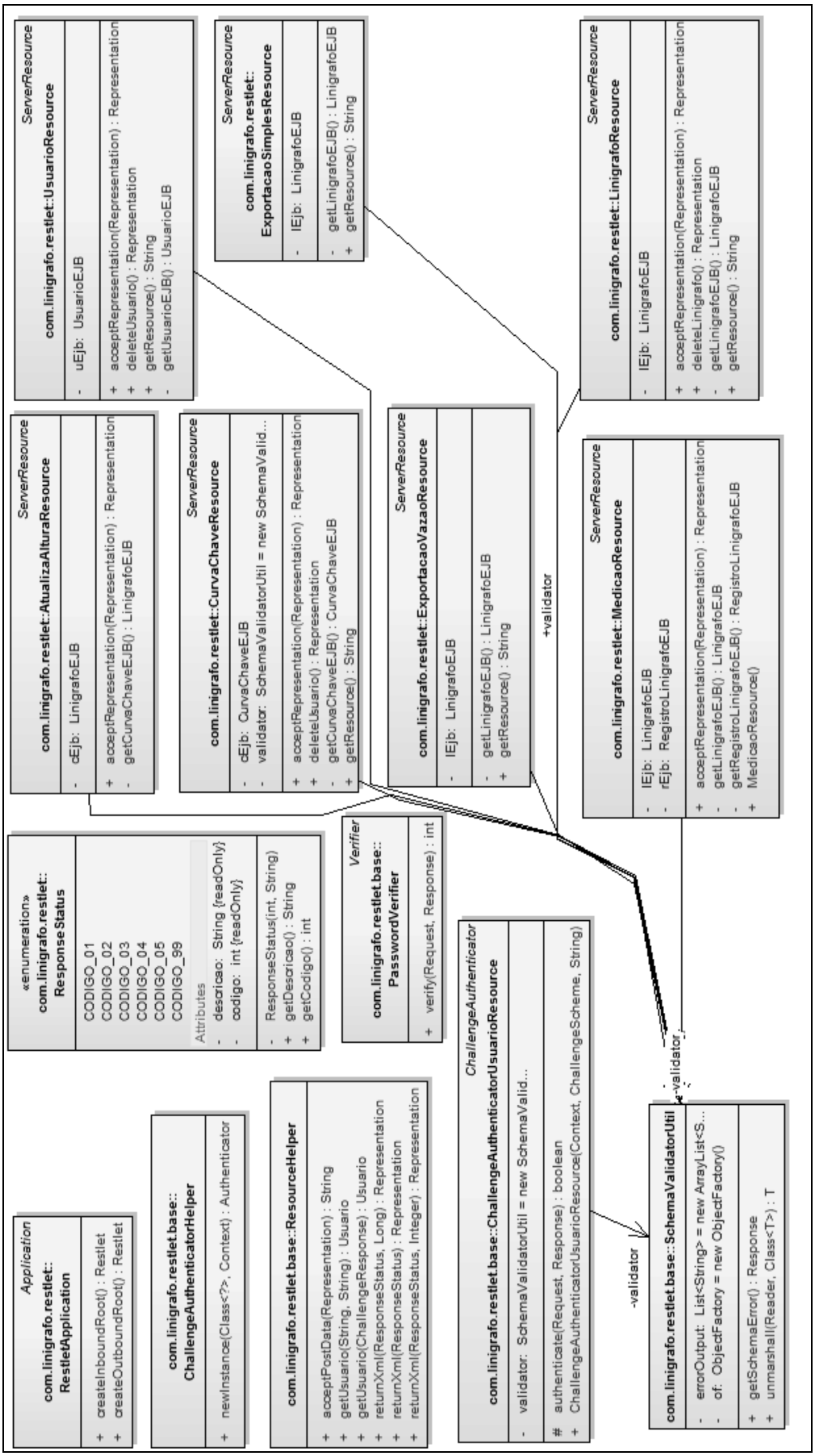


Figura 18 - Diagrama de classes para o recebimento das requisições

A classe `RestletApplication`, é a classe que trata as requisições, redirecionando-as para os devidos recursos. Os recurso são mapeados nas classes `UsuarioResource`, `LinigrafoResource`, `CurvaChaveResource`, `MedicaoResource`, `AtualizaAlturaResource`, `ExportacaoSimplesResource` e `ExportaVazaoResource`. Suas responsabilidades são descritas a seguir:

a) `UsuarioResource` - cadastrar, atualizar e deletar entidades da classe `Usuario` no sistema;
b) `LinígrafoResource` - cadastrar, atualizar e deletar entidades da classe `Linigrafo` no sistema;
c) `CurvaChaveResource` - cadastrar, atualizar e deletar entidades da classe `CurvaChave` no sistema;
d) `MedicaoResource` - cadastrar novas medições enviadas por um linígrafo previamente cadastrado no sistema;
e) `AtualizaAlturaResource` - ajustar a altura zero, usada como referência durante o cálculo do nível do corpo d'água;
f) `ExportacaoSimplesResource` - gerar um documento XML, com os níveis registrado por determinados linígrafos durante um determinado período;
g) `ExportaVazaoResource` - gerar um documento XML, com os níveis registrados por determinados linígrafos durante um determinado período.

A classe `ShemaValidatorUtil` é utilizada para auxiliar na validação dos documentos XML enviados nas requisições. A classe `ChallengeAuthenticatorHelper` auxilia no momento de instanciar classes que efetuam a segurança dos recursos do *web-service*. A classe `ChallengeAuthenticatorUsuarioResource` provê tratamento especial para o recurso mapeado pela classe `UsuarioResource`. A classe `ResponseStatus` é uma enumeração das possíveis respostas do *web-service*. A classe `PasswordVerifier` é responsável pela verificação da senha e login, enviadas nas requisições do *web-service*. A Figura 19 mostra as classes necessárias para a verificação das senhas enviadas nas requisições.

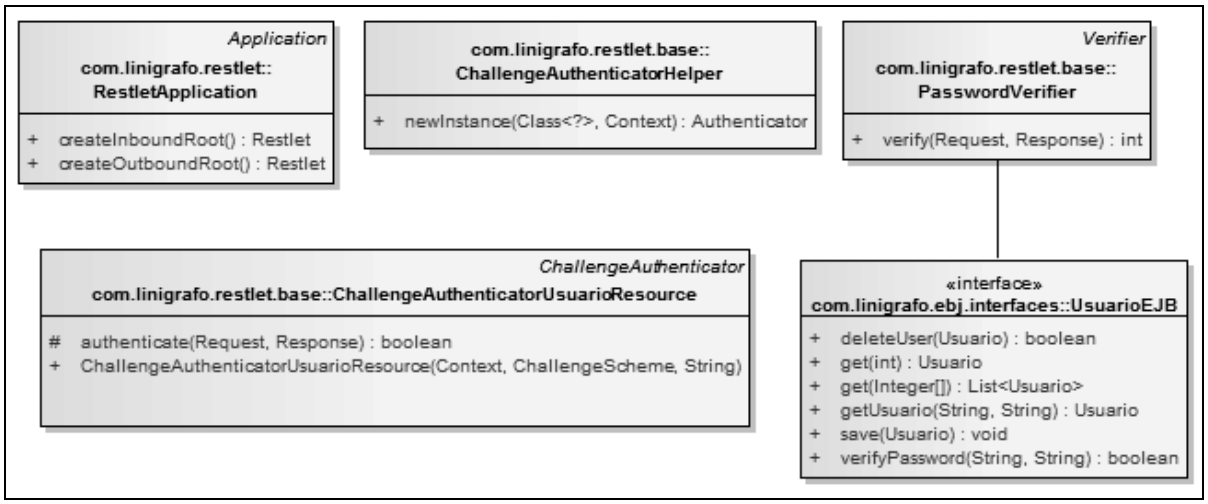


Figura 19 - Diagrama de classes para as verificações das senhas

A interface `UsuarioEJB` é responsável por especificar os métodos onde estão implementados as regras de negócio do sistema.

Na Figura 20, é possível visualizar todas as classes necessárias para o funcionamento da classe `UsuarioResource`.

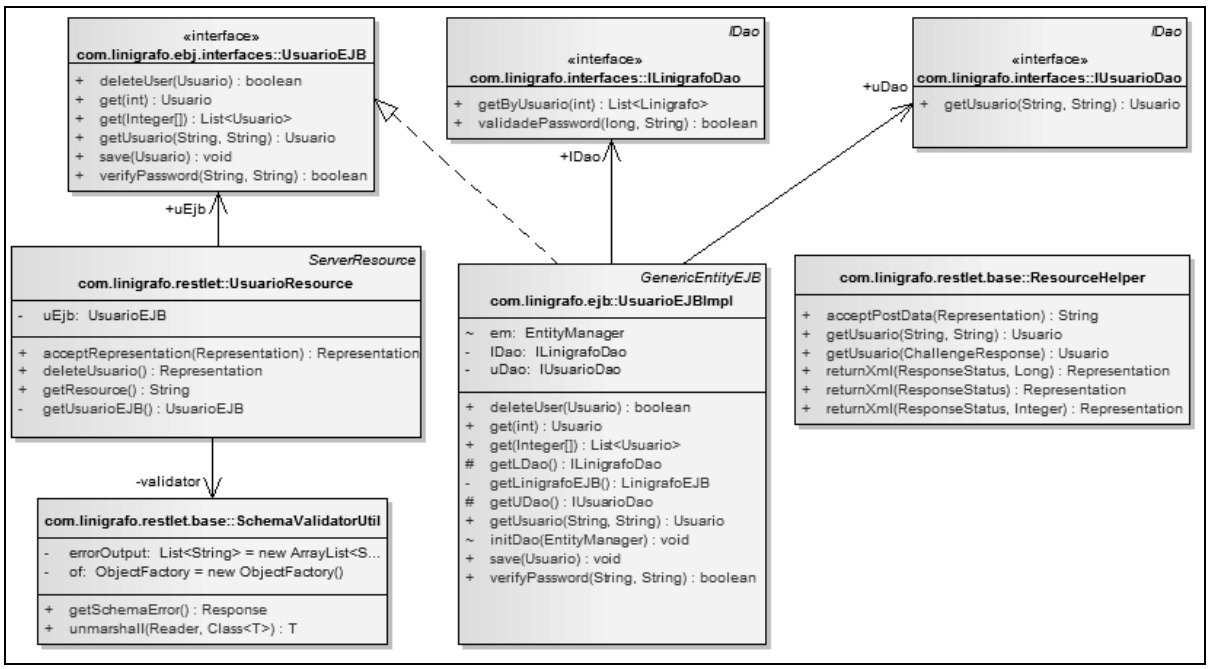


Figura 20 - Diagrama de classes relacionadas com o `UsuarioResource`

A classe `UsuarioResource`, possui uma instância da interface `UsuarioEJB`, implementada por `UsuarioEJBImpl`. A implementação da classe `UsuarioEJB` é principalmente, responsável por implementar as regras de negócios referentes a classe `UsuarioResource`. `UsuarioEJBImpl` gerencia o acesso ao banco, através das classes que implementam as interfaces `ILinigrafoDao`, `IUsuarioDao` e `IRegistroMedicaoDao`.

A Figura 21 abaixo, representa o diagrama de classes que mostra o relacionamento dos recursos do *web-service* com a interface `LinigrafoEJB`.

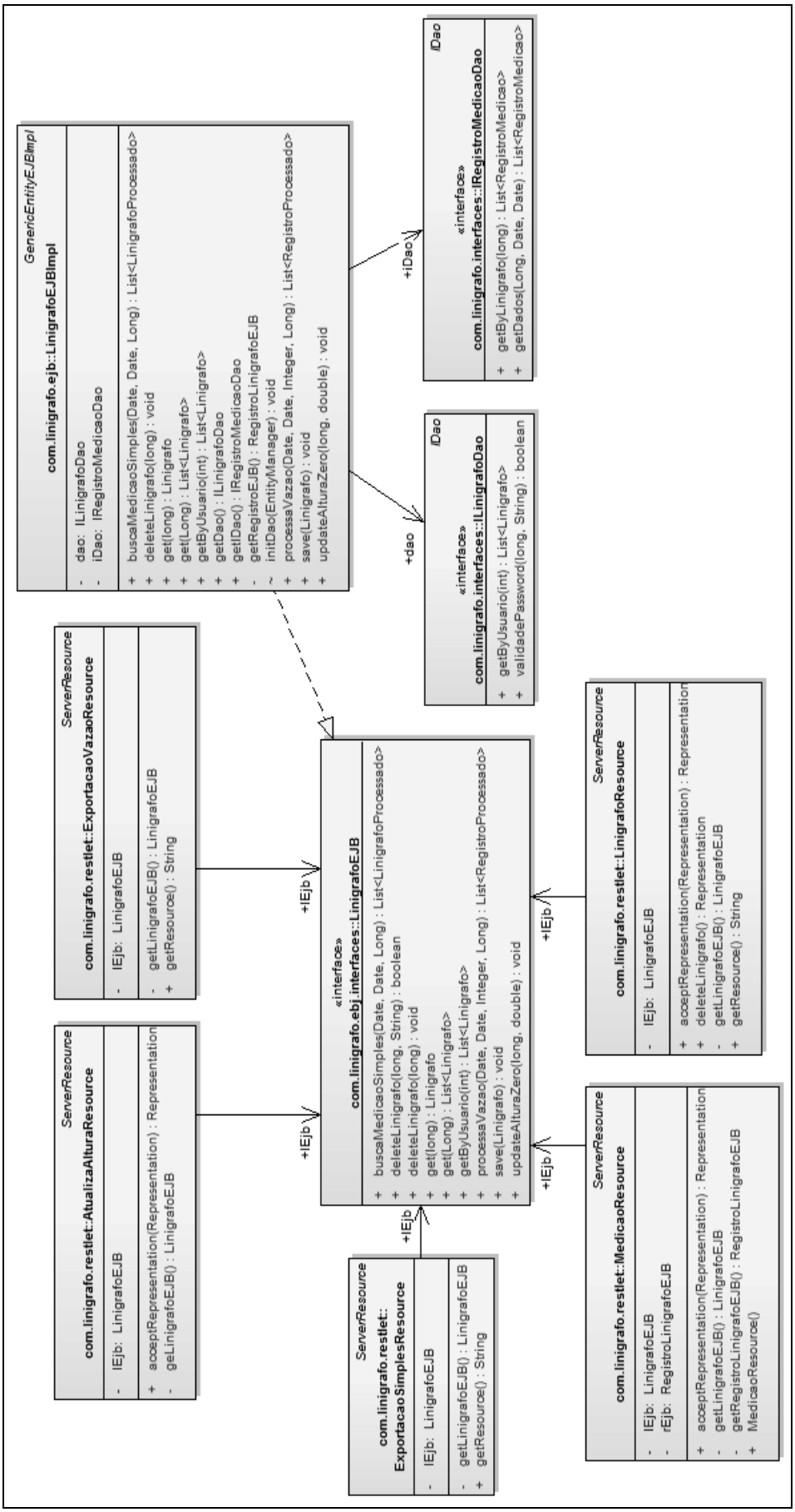


Figura 21 - Diagrama de classes relacionadas com a interface `LinigrafoEJB`

As classes `MediçãoResource`, `LinigrafoResorce`, `ExportacaoVazaoResource`, `AtualizaAlturaResource` e `ExportacaoSimplesResource`, possuem uma instancia da interface `LinigrafoEJB`, implementada por `LinigrafoEJBImpl`. Esta classe é responsável por agrupar implementações das regras de negócios referentes à exportação dos dados e ajuste da altura zero. `LinigrafoEJBImpl` gerencia o acessa ao banco, através das classes que implementam as interfaces `ILinigrafoDao` e `IRegistroMedicaoDao`.

Na Figura 22 é possível visualizar o relacionamento das interfaces `MedicaoEJB` e `CurvaChaveEJB` com outras classes do *web-service*.

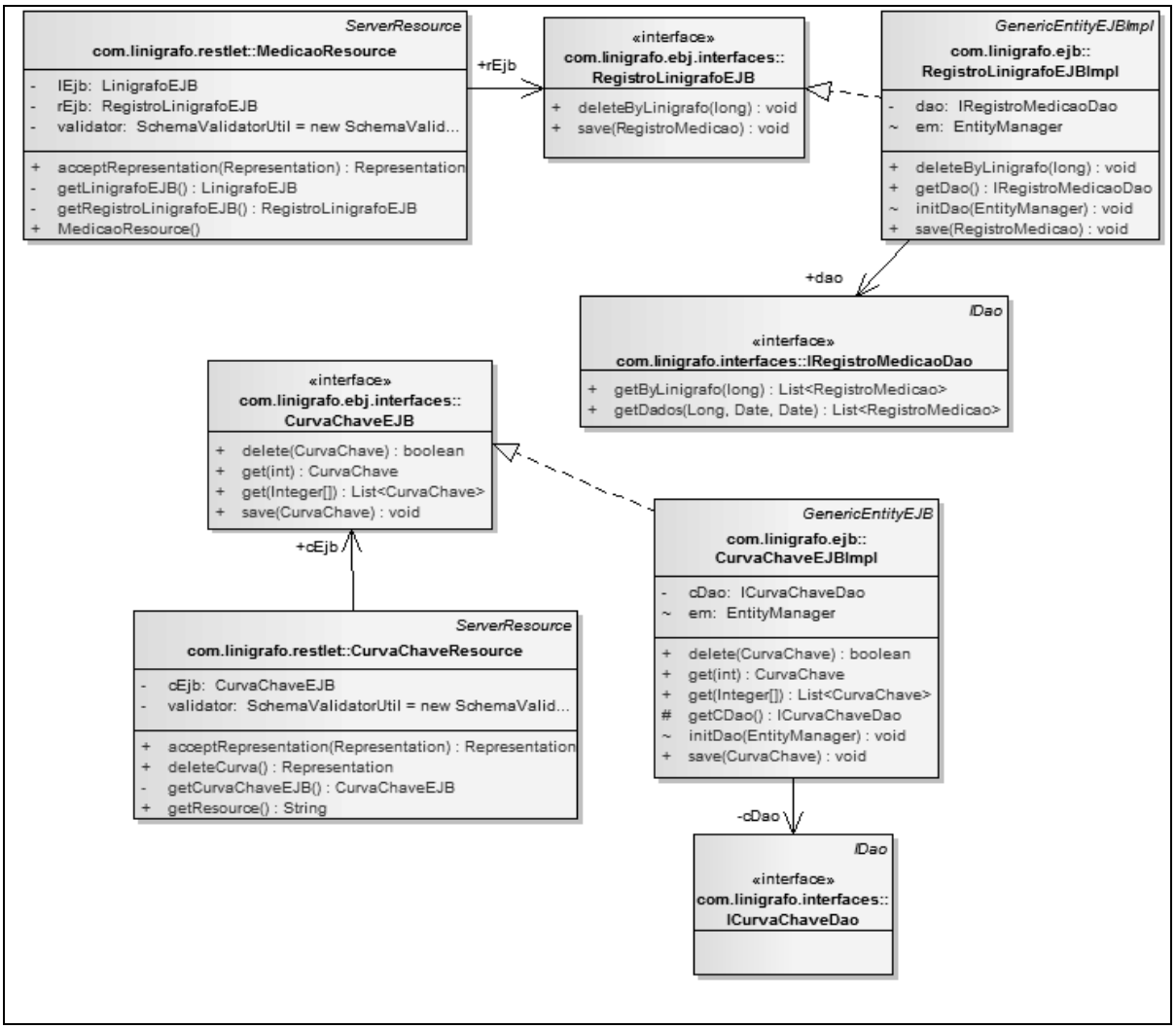


Figura 22 - Diagrama de classes relacionadas com as interfaces `RegistroLinigrafoEJB` e `CurvaChaveEJB`

A interface `RegistroLinigrafoEJB` é implementada pela classe `RegistroLinigrafoEJBImpl`, necessária para o correto funcionamento da classe `MedicaoResource`, devido a classe `RegistroLinigrafoEJBImpl` implementar as regras de negócio para o registro de uma medição enviada pelo linígrafo. A implementação da classe `IRegistroMedicaoDao` é responsável por efetuar o controle do acesso ao banco de dados. A

Classe `CurvaChaveEJBImpl` é responsável por controlar o cadastro, atualização e remoção de curva-chaves utilizando uma implementação da interface `ICurvaChaveDao`.

A Figura 23 apresenta as classes necessárias para o controle ao banco de dados.

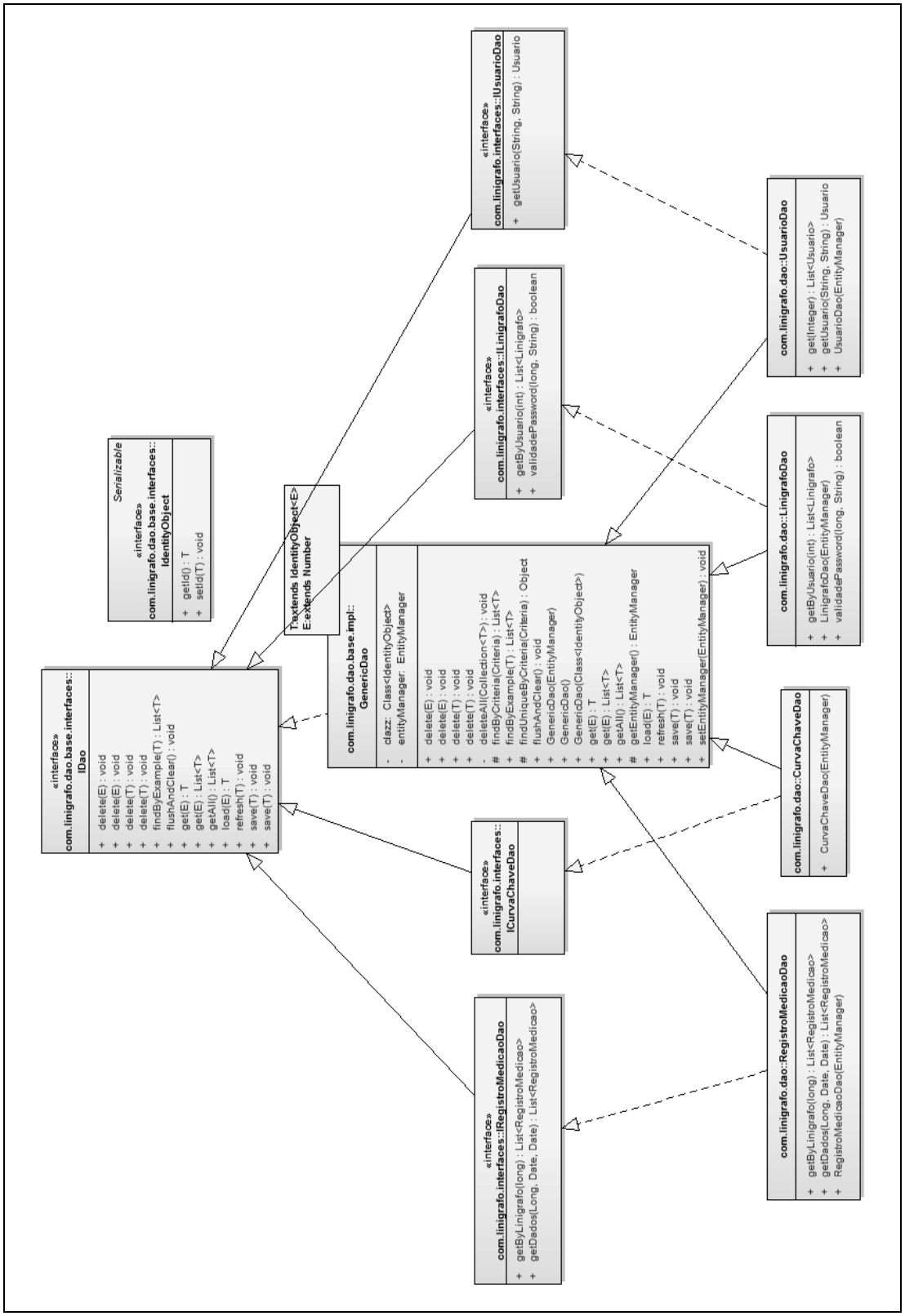


Figura 23 - Diagrama de classes para a persistência das entidades

As classes `RegistroMedicaoDao`, `CurvaChaveDao`, `UsuarioDao` e `LinigrafoDao` são

implementações das interfaces utilizadas nas classes EJB do *web-service*, que controlam o acesso ao banco de dados, efetuando a persistência das entidades. Estas interfaces herdam os métodos da interface `IDao`, que dita a funcionalidade básica para uma classe que controla a persistência das entidades. Esta interface é implementada pela classe `GenericDao` que possui os métodos comuns as interfaces que são filhas de `IDao`. A interface `IdentityObject`, é marca as classes que a implementam como processáveis pela interface `IDao`.

A Figura 24 apresenta as classes de modelo, que mapeiam os usuários do sistema, as curvas-chaves, o protótipo de linígrafo e suas medições.

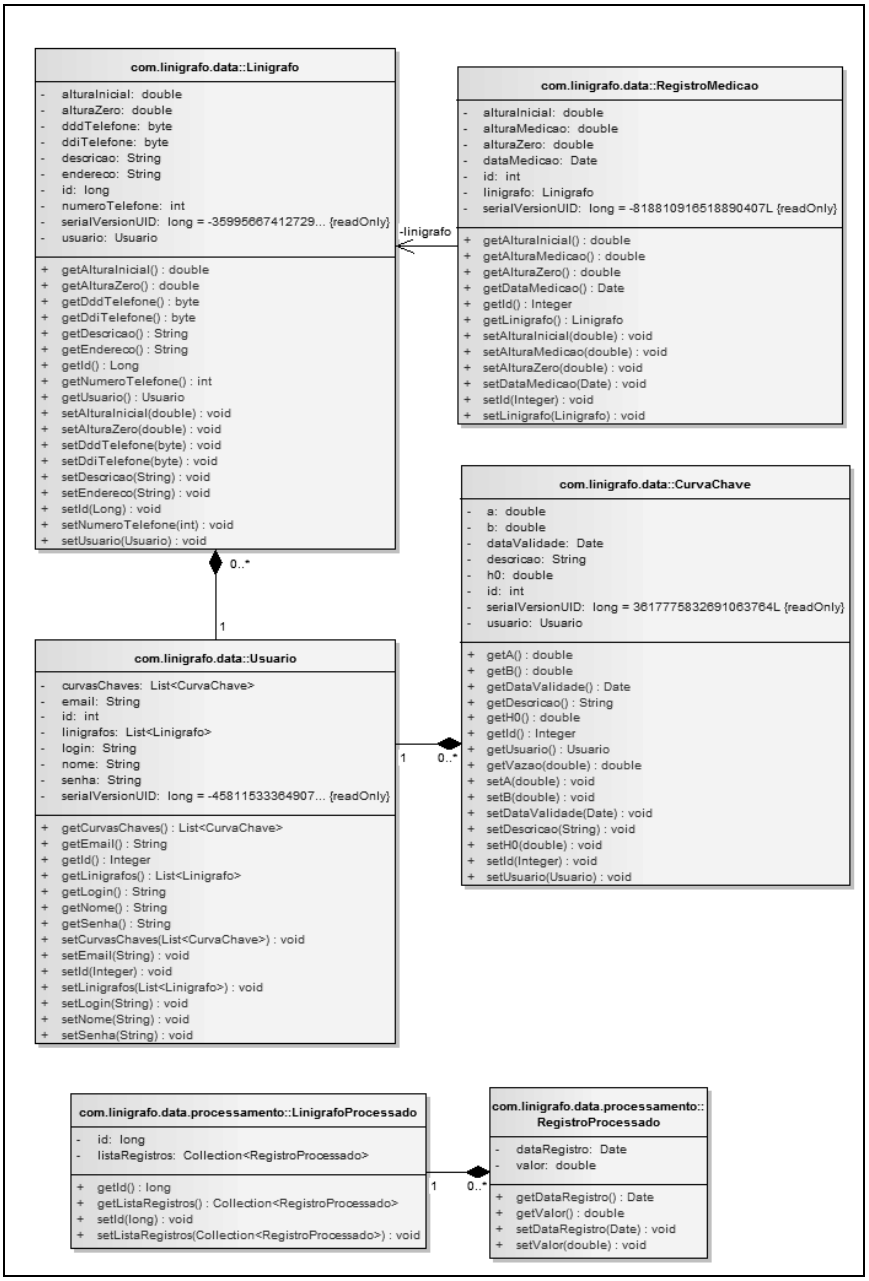


Figura 24 - Diagrama de classes de modelo

As classes `Usuário`, `Linigrafo`, `CurvaChave` e `RegistroMedicao` são classes que

implementam a interface `IdentityObject`, portanto são classes que pode serem persistidas através de classes que implementam a interface `IDao`. `LinigrafoProcessado` e `RegistroProcessado` são classes criadas para auxiliarem `LinigrafoEJB` durante o cálculo do nível ou processamento da vazão.

3.2.2.2.4 Diagramas de atividades do *web-service*

O diagrama de atividades apresenta o fluxo de atividades em um único processo. A Figura 25 representa o diagrama de atividades para a exportação de um documento XML da vazão, referente a uma curva-chave.

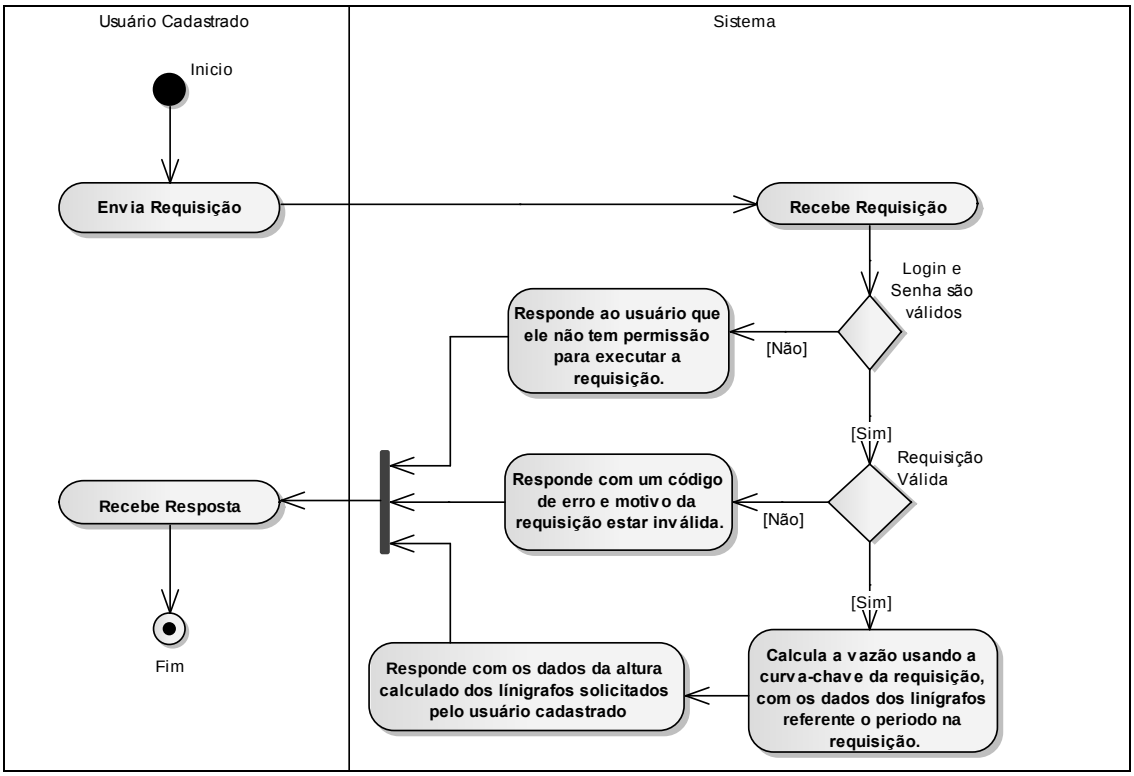


Figura 25 - Diagrama de atividades para a exportação de documentos XML de vazão

O processo de exportação se inicia quando um usuário cadastrado envia uma requisição ao *web-service*. No primeiro momento, o *web-service* verifica se o login e senha enviados na requisição são válidos. Caso não exista um usuário cadastrado com este login e senha, é respondido ao usuário que efetuou a requisição que ele não possui permissão. A próxima atividade do *web-service* é verificar a validade da requisição, ou seja, se todos os dados referentes a ela estão presentes e se os recursos utilizados pertencem ao usuário. Durante a verificação, se for encontrado algum problema, o *web-service* envia uma resposta ao usuário com um código e descrição correspondentes ao erro. Se a requisição não apresentar

problemas, o *web-service* calcula a vazão e responde ao usuário com os dados da vazão.

A Figura 26 corresponde a um diagrama de atividades para o registro de medições enviadas.

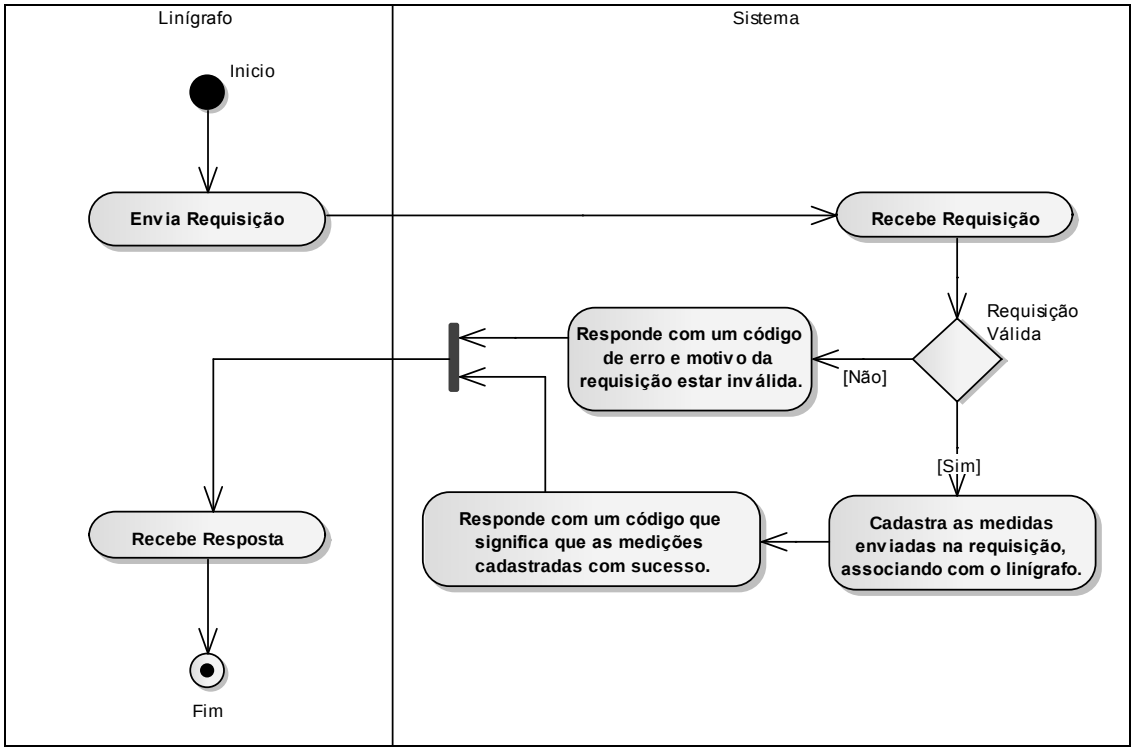


Figura 26 - Diagrama de atividades para o registro de medições

O *web-service* é responsável por processar as requisições enviadas pelos linígrafos cadastrados no sistema. Ao receber uma requisição, o *web-service* efetua a validação, caso exista algum problema com a mesma, é gerada uma resposta com um código e descrição correspondente. O passo seguinte é cadastrar as medidas enviadas e responder ao linígrafo com um código e descrição correspondente a situação de sucesso durante o cadastro.

3.2.2.2.5 Diagrama de sequência do *web-service*

Diagramas de sequência demonstram como os objetos colaboram durante a realização de um processo. Neles são registradas as interações entre as classes envolvidas em um caso de uso. Assim, na Figura 27 é possível observar o diagrama de sequência equivalente ao caso de uso UC10.

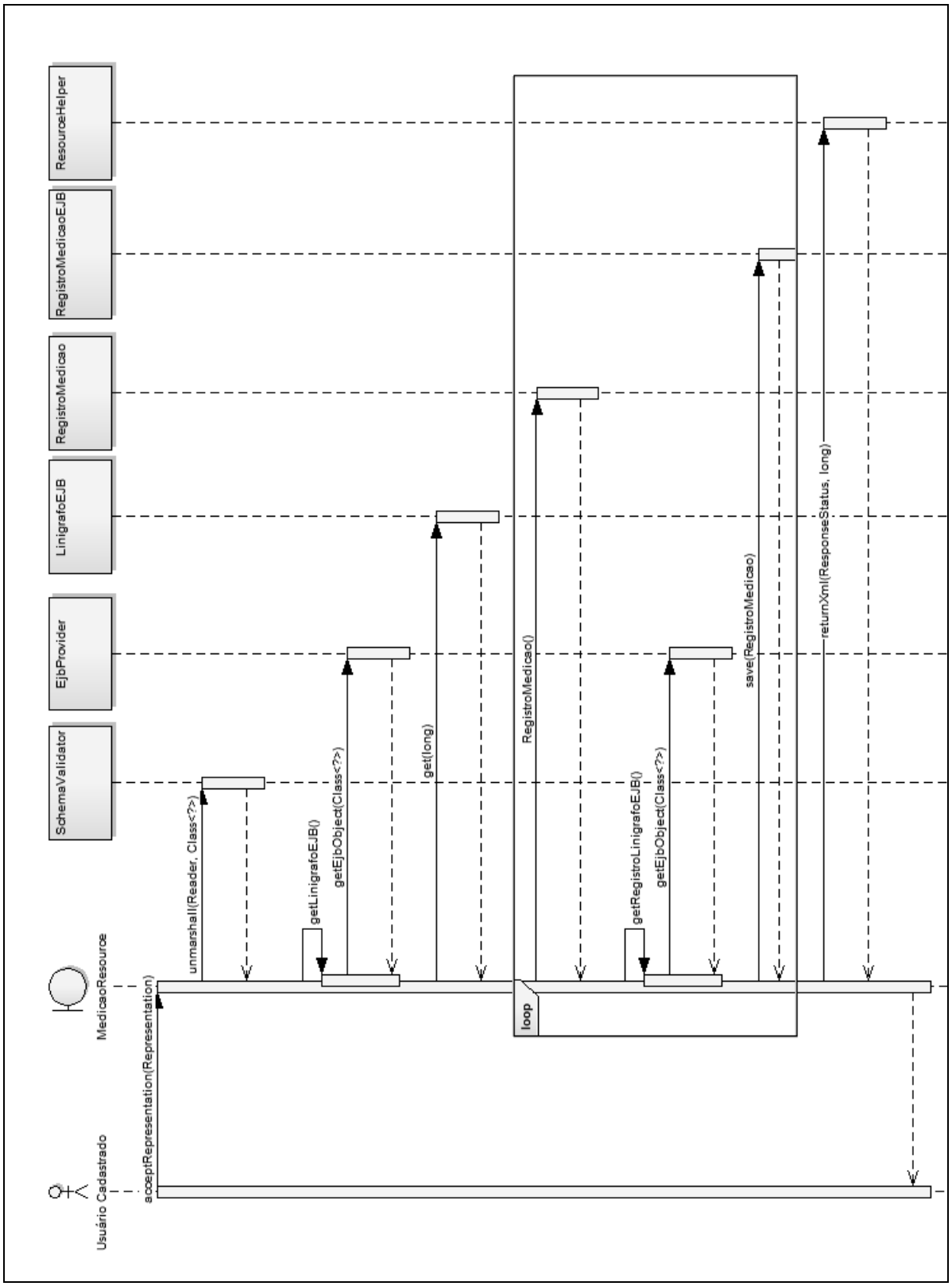


Figura 27 - Diagrama sequência referente ao registro de medições

O processo de registro das medições inicia-se com o envio das medições ao *web-service* efetuado pelo linígrafo. `MedicaoResource`, responsável por receber as requisições, verifica a validade da requisição com o auxilio da classe `SchemaValidator`, através do

método `unmarshall(Reader, Class<?>)`. Após a verificação, caso a requisição seja válida, é necessário buscar uma instância da classe `Linigrafo`, com o identificador único enviado na requisição. Esta busca se inicia com a instanciação de um objeto que implementa a interface `LinigrafoEJB`, feita pela classe `EjbProvider`. A seguir, para cada medição efetuada é instanciado um objeto da classe `RegistroMedicao`, onde é atribuído os dados da medição, associado com o linígrafo e então salvo na base, utilizando o método `save(RegistroMedicao)` da instância de uma classe que implementa `RegistroMedicaoEJB`, criada pelo `EjbProvider`, através do método `getEjbObject(Class<?>)`

3.3 IMPLEMENTAÇÃO

Esta seção descreve implementação, mostrando as técnicas e ferramentas utilizadas na confecção do hardware e software além da operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

Para a implementação do software embarcado no módulo Telit, foi utilizado a linguagem Python com o ambiente de desenvolvimento Eclipse, juntamente com o plugin Pydev. O software embarcado foi implementado seguindo a orientação a objetos, conforme especificação apresentada.

Com o intuito de facilitar o desenvolvimento, a ferramenta de comunicação HyperTerminal Round Solutions GSM Terminal foi utilizada, pois nela se encontram atalhos para efetuar a configuração e utilização de módulos Telit. O HyperTerminal foi configurado da seguinte forma: 115200 bits/s, 8 bits de dados, N de paridade, 1 para bits de parada e N para controle de fluxo desta forma foi possível a visualização de mensagens para a depuração de erros no código e enviar os *scripts* otimizados previamente pelo compilador do Python. Esta otimização reduz significativamente o tempo de inicialização do módulo Telit, considerando que o motor Pyhton no módulo Telit deve compilar e gerar um *script* otimizado, antes de executá-lo.

Na implementação do *web-service* também foi utilizado o ambiente de

desenvolvimento Eclipse, sendo Java a linguagem de programação escolhida. Para alavancar o desenvolvimento do *web-service*, seguindo os conceitos da arquitetura *Restful*, foi utilizado a *framework Restlet*, que possui todas as ferramentas necessárias para criar *web-services*. Para auxiliar e separar as regras de negócios foi utilizado EJB, tornando necessário o uso de um servidor de aplicação com capacidade de criar e gerenciar classes EJB. O servidor de aplicação escolhido foi o JBOSS AS 7. Para facilitar, organizar e padronizar a comunicação do cliente com o *web-service* foram utilizados documentos XML especificados utilizando *XML Schema*. Com o intuito de permitir a manipulação e validação dos documentos XML fez-se o uso da *framework* JAXB. O acesso ao banco de dados MySQL foi efetuado com classes que implementam o padrão de projeto Data Access Object (DAO), com o framework *Hibernate*, conforme a especificação *Java Persistence Api* (JPA).

3.3.2 Hardware

Para a implementação do hardware foi utilizado um kit de desenvolvimento da Sparkfun, com um módulo Telit GE865-QUAD, juntamente com o sonar LV-MaxSonar MB1010 fabricado por Maxbotix. O kit de desenvolvimento da Sparkfun atendeu a vários dos requisitos especificados, sendo um deles, o de disponibilizar uma interface serial para a comunicação com o sonar. Na Figura 28 é possível visualizar o módulo Telit GE865-QUAD, no kit de desenvolvimento da Sparkfun.

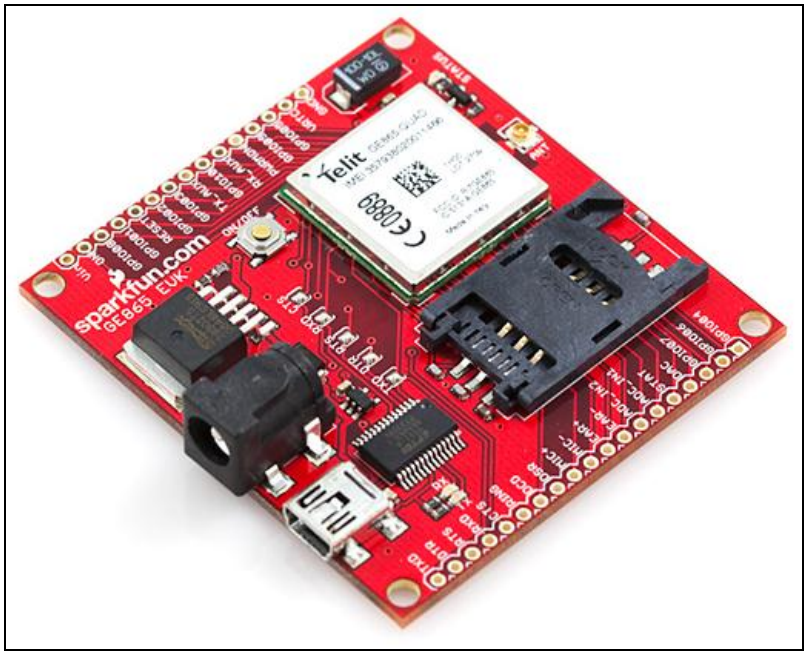


Figura 28 - Kit de desenvolvimento da Sparkfun

Foi utilizada uma placa universal para montagem padrão de circuitos, com a dimensão de 10 centímetros de comprimento por 10 centímetros de largura, para acoplar o módulo sonar e outros circuitos integrados, para o funcionamento do protótipo.

O módulo Telit é o principal componente, sendo ele que efetua a leitura das medidas diretamente do módulo sonar, armazena em memória estes dados, monta a requisição e envia através da rede GSM ao *web-service*. O sonar é responsável por efetuar a medição da distância do protótipo até a superfície da d'água. Ele está conectado ao módulo Telit com os pinos TX e RX. O pino TX esta ligado com o pino RX_AUX do módulo Telit e o pino RX do sonar esta conectado com o pino GPIO_08. Devido a uma característica peculiar do módulo Telit, descrita no seu manual, foi necessário utilizar dois circuitos integrados, o SD4049UBE e o MM74HC125N. Na Figura 12 é possível visualizar o esquema eletrônico mencionado.

Para alimentar o módulo Telit e o sonar, foi utilizada uma bateria de doze volts, regulando esta voltagem primeiramente para cinco volts para respeitar a recomendação da Sparkfun, sobre a alimentação externa do kit de desenvolvimento. O módulo Telit é alimentado com 3.8 volts, através de um regulador de tensão, presente neste kit de desenvolvimento. Este regulador de tensão é conectado pelo pino VIN, presente na placa da Sparkfun. Para alimentar o sonar e os circuitos integrados necessários para tornar a comunicação possível foi utilizado um LM317. Encontrado em diversos dispositivos eletrônicos, o LM317 foi regulado para 2.8 volts, respeitando a voltagem especificada no manual do módulo Telit e no manual do sonar, pois acima de 3.2 volts danifica o módulo Telit e abaixo de 2.5 volts o sonar não opera corretamente.

Para proteger o protótipo em campo e auxiliar na instalação do mesmo, a placa universal foi acoplada uma tampa de tubulação para esgoto. Na Figura 29 é possível visualizar o protótipo montado.

Figura 29 - Protótipo montado

3.3.3 Software

A implementação do software embarcado e *web-service* no desenvolvimento do protótipo são apresentados nas subseções seguintes, seguindo as especificações realizadas em capítulo específico.

3.3.3.1 Software embarcado

Para o software embarcado no protótipo, primeiramente foi necessário o estudo e utilização das bibliotecas nativas e disponíveis no módulo Telit. Essas bibliotecas permitem a manipulação dos recursos do módulo, bem como as portas de comunicação da placa de desenvolvimento Telit.

Inicialmente foi criada uma classe para facilitar a utilização da biblioteca `MOD`, responsável por efetuar manipulação do módulo Telit (Quadro 4).

```
class MDMUtil:
    def send(self,command, delay=20, win="OK", fail="ERROR"):
        print ("Enviando "+command)
        MDM.send(command + '\r', 5)#Envia o comando
        buffer = ''
        i = 0
        while( i < delay ):
            buffer = buffer + MDM.receive(1)# recebe o conteudo
            if buffer.find(win) > -1:# WIN (OK)?
                print ("Recebeu \n"+buffer)
                return buffer
            if buffer.find(fail) > -1:
                print ("Recebeu \n"+buffer)
                return 0
            i = i + 1;
        print ("TIMEOUT")
        return 0
```

Quadro 4 - Envio de comandos através da classe MDM

O programa principal, executado pelo módulo Telit, inicia instanciando um objeto da classe `Linigrafo`. Esta classe é responsável por controlar a execução do software embarcado. No Quadro 5 é possível visualizar a rotina principal e no Quadro 6 é possível visualizar a rotina de inicialização do protótipo.

```
from Classes import Linigrafo, RegistroMedicao
from Utils import SerWriter
import MOD
import sys
if __name__ == '__main__':
    sys.stdout = sys.stderr = SerWriter()
    l = Linigrafo()
    l.verificaConfigurarMenu()
    while 1:
        qtdMedicoes = int(l.buscaConfiguracao("numeroArmazenamento",5))
        qtdIntervalo = int(l.buscaConfiguracao("intervalo",1))
        a = 0
        while a < qtdMedicoes:
            l.guardaRegistro(RegistroMedicao(l.adquireLeitura(),
                                        l.adquireHorario()))
            a = a + 1
            MOD.powerSaving(qtdIntervalo)

        l.enviaMedicoes();
```

Quadro 5 - Rotina principal do software embarcado

```
class Linigrafo:
    def __init__(self):
        self.MDMSender = MDMUtil()
        self.GPRSSender = GprsHelper()
        self.listaMedicao = []
        self.MDMSender.send("AT#CMUXSCR=0")
        SER.set_speed('115200', '8N1')
        SER2.set_speed('9600', '8N1')
        self.MDMSender.send("AT#NITZ=1,0")
              self.MDMSender.send("AT+CMEE=1")
```

Quadro 6 - Rotina de inicialização do protótipo

Ao inicializar o software embarcado é exibido um menu para efetuar as configurações. Este acesso e controle ficou implementado na classe Menu, chamado pelo método `verificaConfigurarMenu` da classe `Linigrafo` (Quadro 7).

```
    def verificaConfigurarMenu(self): # Método da classe Linígrafo
        m = Menu.Menu()
        m.verificaConfigurarMenu(self)

    def verificaConfigurarMenu(self,linigrafo): # Método da classe Menu
        print "digite * para entrar no menu de configuracao"
        if(SER.receive(80).find('*') != -1):#Configura!
            while1 = 1
            while while1:
                print "Menu de configuracao do linigrafo"
                print "1 - Ajustar intervalo de medicao"
                print "2 - Ajustar numero de medicoes que sao armazenadas
na memoria antes de \r\n serem enviadas"
                print "3 - Visualizar configuracoes salvas"
                print "4 - Efetuar calibracao inicial"
                print "5 - Sair"
                while1 = self.tratarEscolhaMenu(linigrafo)
        else:
            print "Não configura"
```

Quadro 7 - Rotina do software embarcado para configuração do menu

Após a configuração são buscados na memória interna os valores para atribuir o intervalo de medições e quantidade de medições para armazenar antes de enviar. Esta rotina também é implementada na classe `Menu`, chamada pela classe `Linigrafo`, através do programa principal (Quadro 8).

```
def buscaConfiguracao(self,propriedade,valorDefault): #Classe Menu
        try:
            f = open("config.conf","r")            #Abre Arquivo
            read = f.readlines()                   #Carrega em uma lista
            f.close()                              #Fecha arquivo
            for line in read:#Itera pelas linhas do arquivo
                splited = line.split('=')
                if(splited[0] == propriedade):  # Achou?
                    return splited[1];          # retorna valor

            return valorDefault         #Não achou o valor ou
        except IOError:                 #Arquivo não existe
            print "IOError - Nao foi possível ler configuracao"
            return valor default
```

Quadro 8 - Leitura de configuração na memória interna do software embarcado

Para criar um registro de medição é necessário buscar o horário atual, fornecido pelo relógio interno do módulo Telit e a medição proveniente do sonar. Foi criado um método que busca o horário atual, processa a resposta e monta um literal que representa o horário conforme a especificação da *XML Schema*, para o envio ao *web-service*. No Quadro 9 é

possível visualizar a rotina presente na classe `Linigrafo`.

```
    def adquireHorario(self):
        horaEntrada = self.MDMSender.send("AT+CCLK?",60)
        indexString = horaEntrada.find("CCLK")
        if(indexString > -1):
            dia = horaEntrada[indexString+13:indexString+15]
            mes = horaEntrada[indexString+10:indexString+12]
            ano = horaEntrada[indexString+7:indexString+9]
            hora = horaEntrada[indexString+16:indexString+18]
            minuto = horaEntrada[indexString+19:indexString+21]
            segundo = horaEntrada[indexString+22:indexString+24]
            if(len(horaEntrada) == 36):
                tz = "-03"#nao achei o TZ, joga GMT-3
            else:
                tz = horaEntrada[indexString+24:indexString+27]

            return "20"+ano+"-"+mes+"-"+dia+"T"+hora+":"+
                                        minuto+":"+segundo + tz +":00"
     return "0";
```

Quadro 9 - Busca do horário no software embarcado

Com o horário já processado a última pendência para criar um registro de medição é a aquisição da leitura com sonar. Para acionar o sonar é necessário ligar o valor lógico alto no pino GPIO_08 do módulo Telit e ativar o MM74HC125N jogando um valor lógico baixo no GPIO_01. Foi verificado que podem ocorrer diferenças durante medições da mesma distância do corpo d'agua, por este motivo, durante a aquisição é aguardado 1 segundo e feito uma média aritmética com as medições que foram recebidas no módulo Telit. No Quadro 10 é possível visualizar a rotina de aquisição dos dados.

```
def adquireLeitura(self):
        print "Buscando medição"
        GPIO.setIOvalue(8, 1)#Ao jogar alto, inicia aquisição de dados
        GPIO.setIOvalue(1, 0)#Ao jogar baixo, ativa o tri-state
        MOD.sleep(2)#Tempo necessário para a calibragem do sonar
        #Tudo certo por aqui, posso começar a buscar o valor da medição
        SER2.read()#Tira o lixo da serial
        MOD.sleep(10)#Espera 1000ms
        GPIO.setIOvalue(8, 0)
        GPIO.setIOvalue(1, 1)
        readedString = SER2.read()#Le os valores
        readedString = readedString.replace(chr(13), '')
        listaMedicoes = readedString.split("R")
        del readedString
        valorMedio = 0
        numeroErros = 0;
        for i in range(len(listaMedicoes)):
            if (len(listaMedicoes[i]) == 3):# Se o número tem 3 dígitos
                # equanto tiver zeros na frente
                while(listaMedicoes[i].startswith("0")):
                     #Remove o primeiro zero a esquerda
                     listaMedicoes[i] = listaMedicoes[i][1:]
                if(listaMedicoes[i] == ""):
                    numeroErros = numeroErros + 1
                    continue
                try:
                    #Converte aqui para inteiros
                    valorMedio = valorMedio + int(listaMedicoes[i])
                except ValueError:
                    numeroErros = numeroErros + 1
            else:# Se não tiver 3 dígitos, é inválido
                numeroErros = numeroErros + 1

        ret = valorMedio / (len(listaMedicoes) - numeroErros
        print "Valor adquirido: "+ str(ret) + " inches"
        return ret
```

Quadro 10 - Processamento da medição efetuada pelo sonar no software embarcado

Com o número necessário de medições efetuadas, é necessário envia-las ao *web-service*. A classe `GPRSHelper` efetua a conexão à rede GPRS. Esta classe é responsável por inicializar os parâmetros de configuração referentes à operadora (neste caso a TIM) e enviar os comandos necessários, através da classe `MDMUtil`, para abrir conexões com o *web-service*, para o envio de requisições HTTP. O número de medições processadas e enviadas é restrito devido a uma limitação do hardware. No Quadro 11 é possível visualizar a rotina para conectar a rede GPRS e no Quadro 12 é possível visualizar a conexão ao *web-service*.

```
From Utils import MDMUtil
class GprsHelper:
    def __init__(self):
        self.MDMSender = MDMUtil()

    def initGPRS(self):
        if(not self.MDMSender.send('AT+CGDCONT=1,"IP","tim.br"')):
            return 0
        if(not self.MDMSender.send('AT#USERID="tim"')):
            return 0
        if(not self.MDMSender.send('AT#PASSW="tim"')):
            return 0
        return 1

    def setGPRS(self,switch):
        switch = str(switch)
        cgatt = self.MDMSender.send('AT+CGATT?',1000)
        if(switch == "1"):
            #SE CGATT estiver ligado, verificar se GPRS esta ligado
            #SE CGATT estiver ligado, e GPRS não, ligar
            if(cgatt.find("1") != -1):#LIGADO?
                gprs = self.MDMSender.send('AT#GPRS?',1000)
                if(gprs != 0 and gprs.find("1") == -1):
                    #Demora, depende a qualidade do sinal
                    self.MDMSender.send('AT#GPRS=1',1000)
            #SE CGATT estiver desligado, ligar os 2
            else:#CGATT Desligado
                self.MDMSender.send('AT+CGATT=1',1000)
                self.MDMSender.send('AT#GPRS=1',1000)
        elif (switch == "0"):
            #Manda desabilitar mesmo assim, se der ERROR não importa
            self.MDMSender.send('AT#GPRS=0',1000)
            self.MDMSender.send('AT+CGATT=0',1000)

        gprs = self.MDMSender.send('AT#GPRS?')
        if(gprs != 0 and gprs.find(switch) == -1):
            return 0
        return 1
```

Quadro 11 - Configuração para a conexão GPRS

```
    def enviaRequisicao(self,request):
        if(not self.MDMSender.send(
             'AT#SKTD=0,8080,"sapus.servebeer.com",0,0',1000,
                  "CONNECT")):
            self.setGPRS(0)#Deu ERROR, desativar GPRS
            return 0
        response = self.MDMSender.send(request,2000,"NO CARRIER")
        if(not response):
            return 0
        return response
```

Quadro 12 - Conexão do software embarcado ao *web-service*

Para preparar a requisição é necessário montar o documento XML com as requisições, converter caracteres inválidos neste documento XML e preencher o cabeçalho HTTP com as informações relevantes sobre a requisição. A classe `XMLUtil`, responsável por esta montagem, gera os documentos XML de acordo com a especificação dos *XMLSchema* criados para esta comunicação. No Quadro 13 é possível verificar a montagem do documento XML para as

duas requisições possíveis efetuadas pelo software embarcado.

```
class XMLUtil:
    def createRequestPutMedicoes(self,linigrafoId,listaMedicoes):
        xml = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
        xml = xml + "<medicao-put-request>"
        xml = xml + "<linigrafo-id>" + linigrafoId + "</linigrafo-id>"
        for i in listaMedicoes:
            registroMedicao = i
            xml = xml + "<medicao><data>"+ registroMedicao.data
            xml = xml + "</data><valor>"
            xml = xml + str(registroMedicao.medicao) + </valor></medicao>"
        xml = xml + "</medicao-put-request>"
        return xml

    def createRequestAtualizaAlturaZero(self,linigrafoId,valor):
        xml = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
        xml = xml + "<altura-zero-update-request>"
        xml = xml + "<linigrafo-id>" + str(linigrafoId) + "</linigrafo-
                                                                id>"
        xml = xml + "<valor>" + str(valor) + "</valor>"
        xml = xml + "</altura-zero-update-request>"
        return xml
```

Quadro 13 - Montagem do documento XML para envio do software embarcado ao *web-service*

Seguindo a especificação HTTP, é necessário remover caracteres inválidos, substituindo-os por caracteres de porcentagem e concatenando com o valor hexadecimal do caractere inválido. A classe `UrlUtil` é responsável por efetuar esta conversão, e a sua implementação pode ser visualizada no Quadro 14.

```
class URLUtil:
    def zip(self,a,b):
        if(len(a) > len(b)):
            return self.zipAux(a, b)
        else:
            return self.zipAux(b, a)
    def zipAux(self,a,b):
        list = []
        for i in range(len(a)):
            list.append((a[i],b[i]))
        return list
    def __init__(self):
        always_safe = ('ABCDEFGHIJKLMNOPQRSTUVWXYZ'
               'abcdefghijklmnopqrstuvwxyz'
               '0123456789' '_.-')
        self._safe_map = {}
        listaChar = []
        for z in range(256):
            listaChar.append(chr(z))
        lista = self.zip(range(256), listaChar)
        for i, c in lista:
            if(c < 128 and i in always_safe):
                self._safe_map[i] = i
            else:
                self._safe_map[i] = '%'+('%02X' % c)
    def quote(self,s, safe=''):
        #    quote('abc def') -> 'abc%20def'
        for c in safe:
            self._safe_map[c] = c
        print ("quoting ASCII chars ...")
        for i in self._safe_map.keys():
            s = s.replace(str(i), str(self._safe_map[i]))
        print ("DONE!")
              return s
          def quote_plus(self,s, safe=''):
              if ' ' in s:
                  s = self.quote(s, safe + ' ')
                  return s.replace(' ', '+')
              return self.quote(s, safe)
```

Quadro 14 - Conversão de caracteres inválidos no software embarcado

Esta implementação foi uma conversão da função presente na biblioteca urllib2 do Python a qual não estava disponível no Python 1.5.2+ que executa os *scripts* no módulo Telit. As funções da biblioteca urllib2 utilizavam várias construções presentes somente na versão 2.5 ou superior do Python.

Efetuada a conversão dos caracteres inválidos, é necessário criar a requisição HTTP e enviá-la ao *web-service*. A classe que cria a requisição HTTP é a `URLUtil`, já mencionada anteriormente. No Quadro 15 é possível visualizar o método responsável por gerar a requisição HTTP com seus cabeçalhos.

```
    def createPostRequest(self,xmlData,resource):
        safeData = self.quote_plus(xmlData)
        safeData = "xml-data=" + safeData
        req = "POST "+ resource +" HTTP/1.1\r\n"
        req = req + "Host: sapus.servebeer.com\r\n"
        req = req + "User-Agent: Linigrafo\r\n"
        req = req + "Cache-Control: no-cache\r\n"
        req = req + "Connection: close\r\n"
        req = req + "Accept: application/x-www-form-
                                              urlencoded\r\n"
        req = req + "Content-Type: application/x-www-form-
                                              urlencoded\r\n"
        req = req + 'Content-Length: %d\r\n' % (len(safeData))
        req = req + ("\r\n%s\r\n\r\n" % safeData)
              return req
```

Quadro 15 - Criação da requisição HTTP do software embarcado ao *web-service*

Após enviada a requisição é necessário analisar a resposta recebida para tomar a ação correspondente. No caso do envio das medições é necessário apagar da memória as requisições já enviadas. A ação correspondente ao ajuste da altura zero, é a impressão de seu resultado no console do HyperTerminal ao usuário. No Quadro 16 é possível visualizar os métodos que fazem esta análise.

```
def enviaMedicoes(self):
        if(not self.GPRSSender.setGPRS(1)):
            return 0
        helper = LinigrafoRequestHelper()
        while (len(self.listaMedicao) != 0):
            postMe = helper.createPutMedicoesRequest(self.getId(),
                                                 self.buscaRegistros());
            response = self.GPRSSender.enviaRequisicao(postMe);
            if(not response):
                break;#Tentar na próxima medicao
            if(response and response.find("<codigo>1</codigo>") != -1):
                self.limpaRegistros()#apagar lista
        self.GPRSSender.setGPRS(0)

    def enviaAjusteAlturaZero(self):
        if(not self.GPRSSender.setGPRS(1)):
            return 0
        helper = LinigrafoRequestHelper()
        postMe = helper.createRequestAtualizaAlturaZero(self.getId(),
                                              self.adquireLeitura());
        response = self.GPRSSender.enviaRequisicao(postMe);
        if(response and response.find("<codigo>1</codigo>") != -1):
            print "Sucesso!"
        self.GPRSSender.setGPRS(0)
```

Quadro 16 - Envio de requisições HTTP do software embarcado

Após efetuar as medições e efetuar o envio das mesmas ao *web-service*, o módulo repete esta ação por tempo indeterminado.

3.3.3.2 *Web-service*

Para utilizar o framework Restlet para desenvolvimento de *web-services restful* em Java, é necessário inicialmente configurar um *servlet* para receber todas as requisições enviadas à aplicação executando no servidor de aplicação. Para é isto, deve-se alterar o arquivo `web.xml`, responsável por configurar a aplicação web (Quadro 17).

```
<context-param>
      <param-name>org.restlet.application</param-name>
      <param-value>
         com.linigrafo.restlet.RestletApplication
      </param-value>
</context-param>

<servlet>
      <servlet-name>RestletServlet</servlet-name>
      <servlet-class>
         org.restlet.ext.servlet.ServerServlet
      </servlet-class>
</servlet>
<servlet-mapping>
      <servlet-name>RestletServlet</servlet-name>
      <url-pattern>/*</url-pattern>
</servlet-mapping>
```

Quadro 17 - Configuração do *servlet* para o *web-service*

Uma vez configurado, o *servlet* ao iniciar busca a classe `RestletApplication`, que mapeia os recursos do sistema. Para o *servlet* verificar permissões do usuário durante o processamento das requisições é necessário utilizar um mecanismo do *framework* Restlet que implementa esta funcionalidade. No Quadro 18 abaixo é possível visualizar o mapeamento dos recursos e a utilização deste mecanismo.

```
public synchronized Restlet createInboundRoot() {
      Router router = new Router(getContext());
      getContext().setDefaultVerifier(new PasswordVerifier());
      router.attach("/linigrafo", ChallengeAuthenticatorHelper.
            newInstance(LinigrafoResource.class, getContext()));
      router.attach("/linigrafo/medicao", MedicaoResource.class);
      router.attach("/linigrafo/medicao/ajuste",
            AtualizaAlturaResource.class);
      router.attach("/linigrafo/medicao/simples",
            ChallengeAuthenticatorHelper.newInstance(
            ExportacaoSimplesResource.class,
            getContext()));
      router.attach("/linigrafo/medicao/vazao",
            ChallengeAuthenticatorHelper.
            newInstance(ExportacaoVazaoResource.class,
            getContext()));
      router.attach("/curva-chave", ChallengeAuthenticatorHelper.
            newInstance(CurvaChaveResource.class, getContext()));
      router.attach("/usuario",
            ChallengeAuthenticatorHelper.newInstance(
            UsuarioResource.class, getContext()));
      return router;
}
```

Quadro 18 - Mapeamento dos recursos do *web-service*

As classes `PasswordVerifier` e `ChallengeAuthenticatorHelper` são utilizadas para efetuar a segurança dos recursos. A classe `PasswordVerifier` recebe as informações do usuário e verifica a validade e a `ChallengeAuthenticatorHelper` instancia classes que implementam a classe abstrata `Authenticator`, proveniente da *framework* Restlet, a qual decide se é necessário efetuar a verificação de usuário e senha. Com a utilização destas classes no mapeamento dos recursos, foi possível garantir que não ocorram acessos indevidos aos recursos do *web-service*. O Quadro 19 apresenta a implementação da classe `PassowordVerifier` e a implementação da classe `ChallengeAuthenticatorHelper`.

```
public class PasswordVerifier implements Verifier {

	@Override
	public int verify(Request req, Response resp) {
		try {

			String user =
					req.getChallengeResponse().getIdentifier();
			//Secret é um array de chars, converter para String
			String senha = String.valueOf(
					req.getChallengeResponse().getSecret());

			UsuarioEJB uEjb = EjbProvider.getEjbObject(
					UsuarioEJB.class);

			if(uEjb.verifyPassword(user,senha)){
				return Verifier.RESULT_VALID;
			}else{
				return Verifier.RESULT_INVALID;
			}
		} catch (Exception e) {
			return Verifier.RESULT_INVALID;
		}

	}
}

public class ChallengeAuthenticatorHelper {

	public static Authenticator newInstance(Class<?> clazz, Context
			context) {
		Authenticator chAuth;

		if (clazz == UsuarioResource.class) {
			chAuth = new ChallengeAuthenticatorUsuarioResource(
			   context, ChallengeScheme.HTTP_BASIC, "linigrafo-
			web");
		} else {
			chAuth = new ChallengeAuthenticator(context,
					ChallengeScheme.HTTP_BASIC, "linigrafo-
			web");
		}
		chAuth.setNext(clazz);
		return chAuth;
	}

}
```

Quadro 19 - Verificação de permissão para acesso a um recurso

Com os recursos devidamente protegidos, é necessário implementar cada atividade do *web-service*. Primeiramente é necessário cadastrar um usuário para obter permissão para cadastrar um linígrafo. No Quadro 20 é possível visualizar um trecho da implementação da classe `UsuarioResource`, que recebe uma requisição para cadastrar ou atualizar um usuário da base de dados.

```
	@Post
	public Representation acceptRepresentation(Representation rep){

		String xmlValue = ResourceHelper.acceptPostData(rep);
		//Unmarshal
		StringReader stringReader = new StringReader(xmlValue);
		UsuarioPutRequest putRequest  =
					validator.unmarshall(stringReader,
						UsuarioPutRequest.class);
		if(putRequest == null){
			StringWriter xml = new StringWriter();
			JAXB.marshal(validator.getSchemaError(), xml);
			return new StringRepresentation(xml.toString());
		}

		Usuario u = new Usuario();
		u.setId(putRequest.getId());
		u.setNome(putRequest.getNome());
		u.setSenha(putRequest.getSenha());
		u.setLogin(putRequest.getLogin());
		u.setEmail(putRequest.getEmail());
		getUsuarioEJB().save(u);
		if(getUsuarioEJB().save(u)){
			putRequest.setId(u.getId());
		return ResourceHelper.returnXml(
					ResponseStatus.CODIGO_01,u.getId());
		}else{
			return ResourceHelper.returnXml(
						ResponseStatus.CODIGO_06);
		}
	}
```

Quadro 20 - Rotina para cadastro ou atualização de um usuário no *web-service*

Nesta rotina pode-se visualizar o funcionamento padrão de quase todos os recursos do *web-service*, ou seja, é receber a requisição, verificar a validade de acordo com o *XML Schema* correspondente, caso tenha problemas, responder com um código e descrição de erro, gerados pelo JAXB na classe `SchemaValidatorUtil`. Se a validação da requisição não tiver problemas, tratar a requisição através das classes EJB implementadas, com auxílio dos objetos instanciados pela `ObjectFactory`, gerada pela API do JAXB. No Quadro 21 esta a implementação da classe `SchemaValidatorUtil`, utilizada em todos os recursos do *web-service*, e no Quadro 22 abaixo, esta o trecho do código da classe `UsuarioEJBImpl` e da classe `UsuarioDao` que efetuam a persistência dos usuários na base de dados.

```
public class SchemaValidatorUtil {

	private List<String> errorOutput = new ArrayList<String>();
	private ObjectFactory of = new ObjectFactory();

	public  <T> T unmarshall(Reader input,Class<T> clazz){
		try {
			JAXBContext context = JAXBContext.newInstance(clazz);
			Unmarshaller unmarshaller = context.createUnmarshaller();
			ValidationEventCollector validationCollector= new
                                        ValidationEventCollector();
			unmarshaller.setEventHandler( validationCollector );;
			errorOutput.clear();
			try {
				return (T) unmarshaller.unmarshal(input);
			} catch (Exception e) {
				for( ValidationEvent event:
                                validationCollector.getEvents() ){
					StringBuilder stringBuilder = new
                                                StringBuilder();
					String msg = event.getMessage();
					ValidationEventLocator locator =
                                                event.getLocator();
					int line = locator.getLineNumber();
					int column = locator.getColumnNumber();
					stringBuilder.append("[Error at line ");
					stringBuilder.append(line);
					stringBuilder.append(" column ");
					stringBuilder.append(column);
					stringBuilder.append("] ");
					stringBuilder.append(msg);
					errorOutput.add(stringBuilder.toString());
					System.out.println(errorOutput);
				}
			}
		} catch (JAXBException e) {
			e.printStackTrace();
		}
		return null;
	}


	public Response getSchemaError(){
		if(errorOutput.size() == 0) return null;
		Response schemaError = of.createResponse();

		StringBuffer sb = new StringBuffer();

		for (int i = 0; i < errorOutput.size(); i++) {
			if( i != 0 )
				sb.append("|");
			sb.append(errorOutput.get(i));
		}
		schemaError.setCodigo(ResponseStatus.CODIGO_99.getCodigo());
		schemaError.setDescricao(sb.toString());
		return schemaError;
	}
}
```

Quadro 21 - Validação das requisições enviadas ao *web-service*

```
// Método da classe UsuarioEJBImpl chamado pelo UsuarioResource
@Override
public boolean save(Usuario usuario) {
      // Se for um usuário novo, verificar na
      // base se já existe um usuário com o mesmo login
      if(usuario.getId() == 0 &&
         getUDao().verifyLogin(usuario.getLogin())){
            return false;// Se tiver, não permitir
      }
      getUDao().save(usuario);
      return true;
}

// Método da classe UsuarioDao para verificar duplicidade do login
// ao cadastrar um usuário novo
@Override
public boolean verifyLogin(String login) {
      Criteria forClass = Criteria.forClass(Usuario.class);
      forClass.add(Restrictions.eq("login",login));
      try {
            Usuario usuario = (Usuario) forClass.uniqueResult(
                                          getEntityManager());
            if(usuario == null ) return false;
            else return true;
      } catch (Exception e) {
            System.err.println("ERROR: "+e.getLocalizedMessage());
            return true;
      }
}

//Implementação padrão do GenericDao, chamado atravez do UsuarioDao
 @Override
 public void save(final T object) {
      if (!object.getId().equals(0)) {
            entityManager.merge(object);
      } else {
            entityManager.persist(object);
      }
 }
```

Quadro 22 - Persistência de um usuário na base de dados do *web-service*

A funcionalidade mais importante para o *web-service*, é o envio das medições registradas pelo protótipo de linígrafo, efetuada através da classe `MedicaoResource`. É necessário converter a unidade de medida enviada do protótipo de linígrafo, de polegadas para centímetros. Para salvar o registro de medição, é utiliza a implementação padrão do `GenericDao`. Ação para receber a requisição da classe `MedicaoResource` pode ser visualizada no Quadro 23. É possível observar que padrão de desenvolvimento previamente citado nesta subseção se repete.

```
@Post
public Representation acceptRepresentation(Representation rep) {
      String xmlValue = ResourceHelper.acceptPostData(rep);
      StringReader stringReader = new StringReader(xmlValue);
      MedicaoPutRequest putRequest =
            validator.unmarshall(stringReader,
            MedicaoPutRequest.class);
      if (putRequest == null) {
            StringWriter xml = new StringWriter();
            JAXB.marshal(validator.getSchemaError(), xml);
            return new StringRepresentation(xml.toString());
      }
      long linigrafoId = putRequest.getLinigrafoId();
      Linigrafo linigrafo = getLinigrafoEJB().get(linigrafoId);
      if (linigrafo == null) {
            return ResourceHelper.returnXml(
            ResponseStatus.CODIGO_04,linigrafoId);
      } else {
            for (Medicao medicao : putRequest.getMedicao()) {
                  RegistroMedicao registroMedicao = new
                        RegistroMedicao();
                  registroMedicao.setAlturaZero(
                        linigrafo.getAlturaZero());
                  registroMedicao.setAlturaMedicao(
                        medicao.getValor() * 2.54);
                  registroMedicao.setDataMedicao(
                        medicao.getData().toGregorianCalendar().
                        getTime());
                  registroMedicao.setLinigrafo(linigrafo);
                  registroMedicao.setAlturaInicial(
                        linigrafo.getAlturaInicial());
                  getRegistroLinigrafoEJB().save(
                        registroMedicao);
            }
            return ResourceHelper.returnXml(
                  ResponseStatus.CODIGO_01,linigrafoId);
      }
}
```

Quadro 23 - Persistência dos registros de medição enviados ao *web-service*

Outra funcionalidade importante do *web-service* é a exportação dos dados enviados para um documento XML. Este documento respeita o *XML Schema* correspondente e é gerado através da API do JAXB. O código para a exportação dos dados enviados e o código para a exportação da vazão relacionando uma curva-chave previamente cadastrada e as medições de um linígrafo são similares, mas diferem no resultado final, efetuar o cálculo da vazão com o nível e curva-chave ao invés de simplesmente calcular o nível. No Quadro 24 apresenta a rotina que exporta o documento XML e o Quadro 25 apresenta a rotina que busca os dados do armazenados, preparando-os para a exportação.

```
@Get
public String getResource() {
      String xmlValue = RestletUtil.getXmlValue(getQuery());
      // Unmarshal
      StringReader stringReader = new StringReader(xmlValue);
      MedicoesSimplesFetchRequest fetchData =
                                          validator.unmarshall(
                                                stringReader,
                                 MedicoesSimplesFetchRequest.class);
      if (fetchData == null) {
            StringWriter xml = new StringWriter();
            JAXB.marshal(validator.getSchemaError(), xml);
            return xml.toString();
      }

      // Busca os Dados na Base
      Date inicial =
                  fetchData.getDataInicial().toGregorianCalendar()
                                                      .getTime();
      Date dtFinal =
         fetchData.getDataFinal().toGregorianCalendar().getTime();
      Long[] array = fetchData.getLinigrafos().getId().toArray(new
                                                      Long[0]);
      List<LinigrafoProcessado> buscaMedicaoSimples =
            getLinigrafoEJB()
                  .buscaMedicaoSimples(inicial, dtFinal, array);
      // List<Usuario> list =
      // getUsuarioEJB().get(fetchData.getId().toArray(new
                                                      Integer[0]));
      // Começa a montar resposta da requisição
      ObjectFactory of = new ObjectFactory();
      MedicoesSimplesFetchResponse resp = of
                  .createMedicoesSimplesFetchResponse();

      for (LinigrafoProcessado lp : buscaMedicaoSimples) {
            LinigrafoMedicao lm = of.createLinigrafoMedicao();
            lm.setId(lp.getId());

            for (RegistroProcessado rp : lp.getListaRegistros()) {
                  DadosMedicao createDados =
                                          of.createDadosMedicao();

                  createDados.setData(XMLGregorianCalendarConverter
                 .asXMLGregorianCalendar(rp.getDataRegistro()));
                 createDados.setValor(rp.getValor());
                 lm.getDadosMedicao().add(createDados);
            }
            resp.getLinigrafoMedicao().add(lm);
      }

      StringWriter xml = new StringWriter();
      JAXB.marshal(resp, xml);
      return xml.toString();
}
```

Quadro 24 – Rotina de exportação dos dados para um documento XM L

```
	@Override
	public List<LinigrafoProcessado> buscaMedicaoSimples(Date inicial,
			Date dtFinal, Long... linigrafos) {

		List<LinigrafoProcessado> listaRetorno = new
					ArrayList<LinigrafoProcessado>();
		// Busca para cada linígrafo
		for (Long integer : linigrafos) {
			LinigrafoProcessado lP = new LinigrafoProcessado();

			lP.setId(integer);
			List<RegistroMedicao> dados = iDao.getDados(integer,
					inicial,dtFinal);

			for (RegistroMedicao registroMedicao : dados) {
				RegistroProcessado registroProcessado = new
					RegistroProcessado();
				// Altura do Linígrafo em Relação a Água h0
				// Altura da água no momento da Instalação hI
				// Distância do linígrafo (Medição) hM
				// Altura atual é a (h0 - hM) + hI

				double altura = (registroMedicao.getAlturaZero()
					- registroMedicao.getAlturaMedicao())+
					registroMedicao.getAlturaInicial();

				registroProcessado.setValor(altura);

				registroProcessado.setDataRegistro(
					registroMedicao.getDataMedicao());
				lP.getListaRegistros().add(registroProcessado);
			}
			listaRetorno.add(lP);
		}
		return listaRetorno;
	}
```

Quadro 25 - Cálculo do nível do corpo d'água pelo *web-service*

As outras rotinas são muito similares, praticamente todas precisam validar a identidade do usuário, validar o documento XML presente na requisição com a API do JAXB com a classe `SchemaValidatorUtil`, processar a regra de negócio dentro da implementação da interface EJB correspondente ao recurso utilizado e retornar um XML com a devida resposta a quem solicitou. As rotinas que não validam o usuário são as que cadastram um usuário e que as requisições enviadas pelo linígrafo.

3.3.4 Operacionalidade da implementação

Um hidrólogo que deseja realizar o monitoramento de uma pequena bacia, primeiramente deve contar com o desenvolvimento de um software cliente que consome o *web-service* para o seu cadastro no sistema e o cadastro do linígrafo. O w*eb-service* permite

ao desenvolvedor cadastrar e manter as informações do usuário, dos linígrafos e curva-chaves.

O web-service proporciona facilidade no desenvolvimento para realizar a exportação de informações dos registros de medições efetuas e a relação vazão e nível das curvas-chaves cadastradas.

3.3.4.1 Instalação

O protótipo deve ser instalado em campo protegido da ação do clima, com auxílio de uma estaca ou coluna de ponte e deve ser encaixado através da tampa acoplada ao protótipo. Para criar uma área de remanso e direcionar o sonar para a superfície do corpo d'água, sensor deve estar centralizado e formando um ângulo reto com a superfície do corpo d´água, isso garantirá medições com maior exatidão. A bateria deve ficar também protegida contra a ação do clima. Na Figura 30 é possível visualizar a instalação do protótipo em campo, seguindo estas recomendações.

Figura 30 - Instalação em campo do protótipo

Para o *web-service* funcionar corretamente é necessário a instalação de uma base de dados MySQL e o servidor de aplicação JBOSS AS 7. Deve-se destacar que o *web-service*

funcionará em qualquer outro que esteja configurado adequadamente e que possua suporte para EJB 3. Se desejado é possível substituir o banco de dados por outro que seja compatível com o *framework* Hibernate ou ainda, substituir o Hibernate por outro *framework* que implemente a especificação JPA.

3.3.4.2 Monitoramento e controle

Antes da utilização do protótipo de linígrafo para o registro das medições, é necessário cadastrá-lo no *web-service*, portanto, também é necessário registrar um usuário. A implementação necessária para o consumo do *web-service* requer os *XML Schemas*, que padronizam a comunicação entre cliente e o *web-service*. Tendo cadastrado o protótipo de linígrafo, é necessário ajustar no *web-service*, o nível atual do corpo d'água, para que esta medida seja utilizada como referência para as demais medições. Com o protótipo devidamente cadastrado, não há restrições quanto ao seu uso. Durante sua inicialização é possível configurar os intervalos de medições e ajustar a altura zero, também utilizada como referência para o cálculo do nível do corpo d'agua.

Com o intuito de testar a precisão das medidas do protótipo, ele foi afixado em uma caixa d'água, simulando a medição de um corpo d'água. Pode-se visualizar na Figura 31, o protótipo, desacoplado da tampa de encanamento e instalado acima de uma caixa d'água.

. 

Figura 31 - Protótipo instalado acima de caixa d'água

Para cadastrar o protótipo ao *web-service*, foi utilizada uma classe do *framework* Restlet, que auxilia na conexão e troca de informações com aplicações desenvolvidas com mesmo. O Quadro 26 apresenta o código utilizado para cadastrar um usuário no web-service e seus dados utilizados no cadastramento.

```
            Request requisicao = new Request();//Cria requisição
            Reference reference = new Reference("http://sapus.no-
ip.org:8080/linigrafo-web/usuario");
            requisicao.setResourceRef(reference);
            requisicao.setMethod(Method.POST);
            ObjectFactory of = new ObjectFactory();
            //Criar XML da requisição utilizando JAXB
            //Criar objeto que reflete o schema, com a ObjectFactory
            UsuarioPutRequest putRequest =
of.createUsuarioPutRequest();
            putRequest.setEmail("email@email.com");
            putRequest.setNome("André Zimmermann");
            putRequest.setLogin("andrez");
            putRequest.setSenha("123senha4");
            putRequest.setId(0);//Novo usuário
            Writer writer = new StringWriter();
            JAXB.marshal(putRequest, writer);//Gera o XML no writer
            //Gera uma representação do recurso
            StringRepresentation s = new StringRepresentation(
writer.toString());
            requisicao.setEntity(s);//Atribui a representação
(entidade) na requisição
            //Envia requisição
            Client client  = new Client(Protocol.HTTP);
            Response resposta = client.handle(requisicao);
            // Retorna um objeto response para tratamento
```

Quadro 26- Exemplo para cadastramento de um usuário na base de dados do *web-service*

Em seguida, deve-se cadastrar o linígrafo, enviando também as informações necessárias para a identificação de qual usuário esta efetuando a requisição. No Quadro 27 esta o trecho do código necessário para efetuar o cadastramento do protótipo do linígrafo utilizado para registrar a medição do nível d'água.

```
            Request requisicao = new Request();//Cria requisição
            Reference reference = new Reference("http://sapus.no-
ip.org:8080/linigrafo-web/linigrafo");
            requisicao.setChallengeResponse(new ChallengeResponse(
                        ChallengeScheme.HTTP_BASIC, "andrez",
                                                    "123senha4"));
            requisicao.setResourceRef(reference);
            requisicao.setMethod(Method.POST);
            ObjectFactory of = new ObjectFactory();
            //Criar XML da requisição utilizando JAXB
            //Criar objeto que reflete o schema, com a ObjectFactory
            LinigrafoPutRequest putRequest =
of.createLinigrafoPutRequest();
            putRequest.setAlturaInicial(55.88);
            putRequest.setDdd(47);
            putRequest.setDdi(55);
            putRequest.setAlturaZero(1);
            putRequest.setNumeroTelefone(99999999);
            putRequest.setDescricao("Linígrafo de Teste");
            putRequest.setEndereco("Caixa D'água - 01");
            putRequest.setId(357460036695315l);//Imei do módulo Telit
            Writer writer = new StringWriter();
            JAXB.marshal(putRequest, writer);//Gera o XML no writer
            //Gera uma representação do recurso
            StringRepresentation s = new StringRepresentation(
writer.toString());
            requisicao.setEntity(s);//Atribui a representação (entidade)
na requisição
            //Envia requisição
            Client client  = new Client(Protocol.HTTP);
            Response resposta = client.handle(requisicao);
            // Retorna um objeto response para tratamento
```

Quadro 27 - Exemplo para cadastramento de um linígrafo na base de dados do *web-service*

Após realizar o cadastramento e configuração do linígrafo é preciso aguardar o protótipo enviar os dados. No Quadro 28 pode-se verificar uma requisição enviada pelo protótipo ao *web-service*.

```
POST /linigrafo-web/linigrafo/medicao HTTP/1.1
Host: sapus.no-ip.org
User-Agent: Linigrafo
Cache-Control: no-cache
Connection: close
Accept: text/xml; charset="utf-8"
Content-Type: application/x-www-form-urlencoded
Content-Length: 1278

xml-data=%3C%3Fxml+version%253D%221.0%22+encoding%253D%22UTF-8%22%3F
%3E%3Cmedicao-put-request%3E%3Clinigrafo-id%3E357460036695315%3C%2Fl
inigrafo-id%3E%3Cmedicao%3E%3Cdata%3E2011-10-29T09%3A36%3A20-03%3A00
%3C%2Fdata%3E%3Cvalor%3E16%3C%2Fvalor%3E%3C%2Fmedicao%3E%3Cmedicao%3
E%3Cdata%3E2011-10-29T09%3A37%3A22-03%3A00%3C%2Fdata%3E%3Cvalor%%3A2
6-03%3A00%3C%2Fdata%3E%3Cvalor%3E16%3C%2Fvalor%3E%3C%2Fmedicao%3E%3C
medicao%3E%3Cdata%3E2011-10-29T09%3A39%3A29-03%3A00%3C%2Fdata%3E%3Cv
alor%3E16%3C%2Fvalor%3E%3C%2Fmedicao%3E%3Cmedicao%3E%3Cdata%3E2011-1
0-29T09%3A40%3A32-03%3A00%3C%2Fdata%3E%3Cvalor%3E16%3C%2Fvalor%3E%3C
%2Fdata%3E%3Cvalor%3E16%3C%2Fvalor%3E%3C%2Fmedicao%3E%3Cmedicao%3E%3
Cdata%3E2011-10-29T09%3A42%3A40-03%3A00%3C%2Fdata%3E%3Cvalor%3E16%3C
%2Fvalor%3E%3C%2Fmedicao%3E%3Cmedicao%3E%3Cdata%3E2011-10-29T09%3A43
%3A44-03%3A00%3C%2Fdata%3E%3Cvalor%3E16%3C%2Fvalor%3E%3C%2Fmedicao%3
E%3Cmedicao%3E%3Cdata%3E2011-10-29T09%3A44%3A48-03%3A00%3C%2Fdata%3E
%3Cvalor%3E16%3C%2Fvalor%3E%3C%2Fmedicao%3E%3Cmedicao%3E%3Cdata%3E20
11-10-29T09%3A45%3A52-03%3A00%3C%2Fdata%3E%3Cvalor%3E16%3C%2Fvalor%3
E%3C%2Fmedicao%3E%3C%2Fmedicao-put-request%3E
```

Quadro 28 - Exemplo de requisição para registro de medições enviadas pelo protótipo

À medida que as medições são registradas pelo *web-service* é possível exportar um documento XML destas medições. No Quadro 29 esta um trecho de código que solicita ao *web-service* esta exportação e no Quadro 30, o gráfico plotado desta exportação.

```
ObjectFactory of = new ObjectFactory();
Request request = new Request();
Reference reference = new Reference(
      "http://localhost:8080/linigrafo-web/linigrafo/medicao/simples");
request.setResourceRef(reference);
request.setChallengeResponse(new ChallengeResponse(
                        ChallengeScheme.HTTP_BASIC, "andrez", "123"));
request.setMethod(Method.GET);
MedicoesSimplesFetchRequest msfr = of
                        .createMedicoesSimplesFetchRequest();
Linigrafos createLinigrafos = of.createLinigrafos();
createLinigrafos.getId().add(3574600366953151);
msfr.setLinigrafos(createLinigrafos);
msfr.setDataInicial(XMLGregorianCalendarConverter
                        .asXMLGregorianCalendar(getDataInicial()));
msfr.setDataFinal(XMLGregorianCalendarConverter
                        .asXMLGregorianCalendar(getDataFinal()));
Writer writer = new StringWriter();
JAXB.marshal(msfr, writer);
reference.addQueryParameter("xml-data", writer.toString());
Client client = new Client(Protocol.HTTP);
Response resposta = client.handle(request);
MedicoesSimplesFetchResponse mFetch = JAXB.unmarshal(new
                        StringReader(resposta.getEntityAsText()),
                        MedicoesSimplesFetchResponse.class);
```

Quadro 29 - Exemplo de exportação de documento XML dos níveis registrados

Quadro 30 - Gráfico plotado com medições registradas pelo protótipo de linígrafo

Para a utilização dos recursos para o cálculo da vazão, é necessário cadastrar uma curva-chave e seguir os mesmos passos da exportação de documentos XML do nível registrado por um determinado linígrafo.

3.4 RESULTADOS E DISCUSSÃO

Para validar as medições efetuadas pelo protótipo, o mesmo foi instalado na bacia do Ribeirão Concórdia. Esta bacia está situada no município de Lontras, vertente atlântica do Estado de Santa Catarina. Ela está inserida na bacia do Rio Itajaí, sendo uma das sete bacias monitoradas pelo Projeto de Recuperação Ambiental e de Apoio ao Pequeno Produtor Rural (PRAPEM/MICROBACIAS), desenvolvido pela Secretaria de Estado da Agricultura e Desenvolvimento Rural de Santa Catarina. Ela faz parte do projeto “Rede de pesquisa em bacias representativas e experimentais no bioma da mata atlântica, na região sul do Brasil”, denominado MATASUL, financiado pelo MCT/FINEP/CT-Hidro-CNPq, sob a coordenação da UFRGS/IPH, com a participação da UFSM, UFPR e FURB. Na Figura 32 é possível observar a bacia e localizar a instalação do línigrafo.

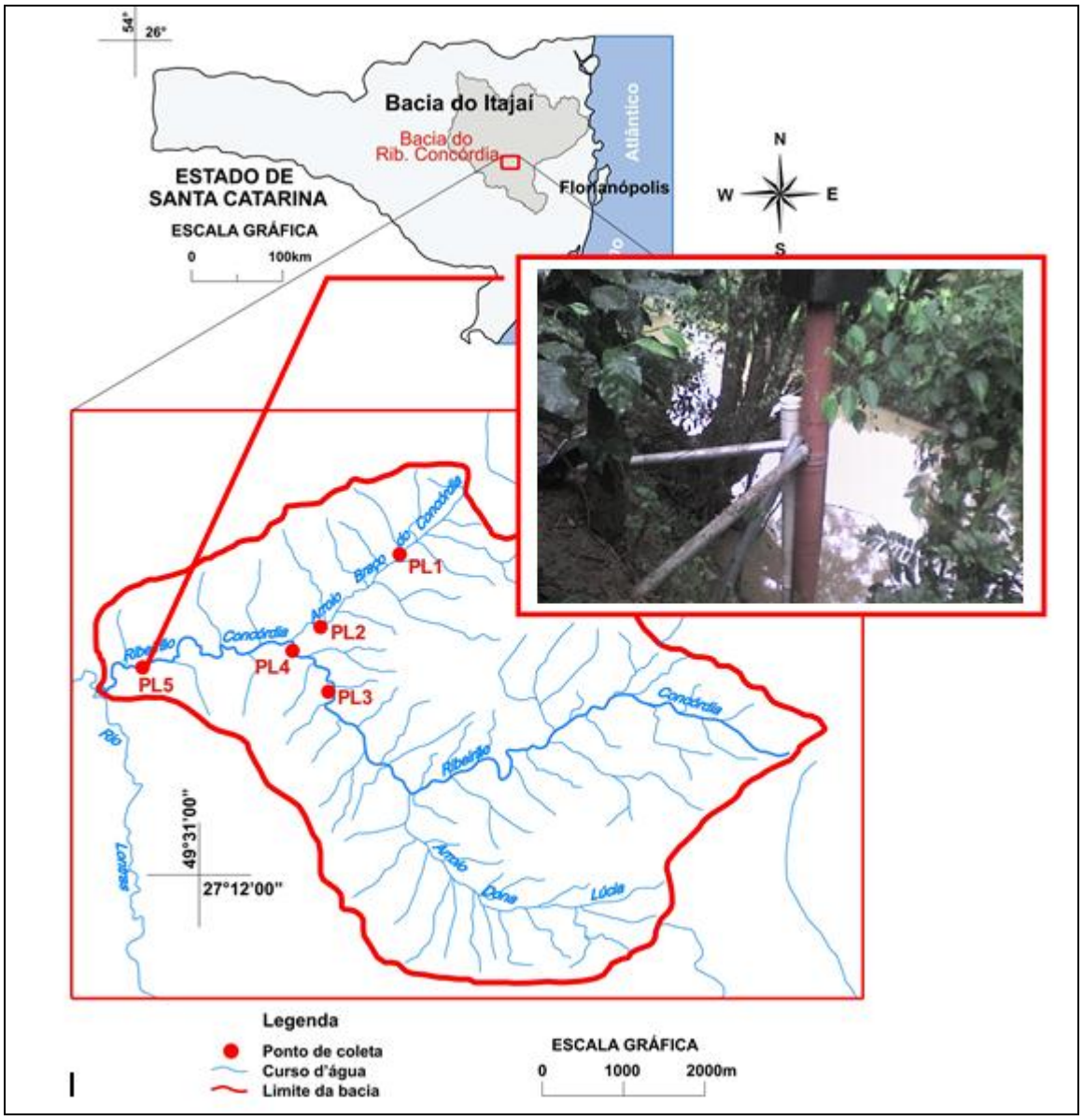


Figura 32 - Local de instalação do protótipo

Neste teste, os resultados das medições deveriam ser comparados com outro linígrafo, o WL-15 da Global Waters, descrito como um trabalho correlato. Embora o linígrafo tenha efetuado as medições corretamente, não foi possível envia-las ao *web-service* para o devido registro e análise. Embora houvesse sinal das operadoras de celular, o protótipo não obteve sucesso em captar este sinal e se registrar na rede. Foram realizados testes com diferentes antenas, de potência similar, na perspectiva de eliminar a antena como causa do problema. Contudo não se obteve sucesso. Restando apenas a placa a hipótese é que a placa da Sparkfun não tenha sido desenvolvida para captar sinais da rede GSM muito fracos. Como este problema é possível somente buscando outra placa com menor interferência ou desenvolvendo uma placa própria para tal, foi buscada outra opção para a validação deste protótipo.

O protótipo foi instalado para monitorar o nível de uma caixa d'água, visualizado na Figura 31. Durante os testes o módulo não apresentou nenhum problema para o envio dos dados ao *web-service* e suas medições foram precisas, dentro da margem de erro do sonar, que é aproximadamente 2.54 centímetros, para mais ou para menos. O *web-service* implementado se comportou da maneira esperada, sendo de fácil consumo, pode receber diversas requisições por segundo para as suas diversas operações.

É possível relacionar o protótipo com os trabalhos correlatos, podendo ser visualizado no Quadro 31.

Trabalho	Protótipo desenvolvido	Rabello, Cruvinel e Denardin	Thalimedes	RLS	WL-15
Tipo de Medição	Sem contato , por sonar	Boia - micro controlado	Boia - micro controlado	Sem contato por radar	Sensor de pressão
Faixa útil	15,24 cm até 6,45 m	0cm até 2,55 cm	0cm até 60m	80,00 cm até 35 metros	0cm até 6.35m
Armazenamento dos dados	3.000	Não especificado	30.000	0	24.400
Telemetria	GSM	-	-	-	-

Quadro 31 - Características dos trabalhos desenvolvidos e trabalhos correlatos.

Quanto ao armazenamento dos dados na memória interna, é importante ressaltar que embora a capacidade interna do protótipo desenvolvido seja menor que o WL-15 ou o Thalimedes ela é esvaziada regularmente devido a sua comunicação com o *web-service*.

4 CONCLUSÕES

Os objetivos traçados foram atendidos, embora que no teste de campo o protótipo não tenha se conectado a rede GSM. As limitações conhecidas sobre o uso da rede GSM também se aplicam ao protótipo, tornando a sua utilização inviável em locais com sinal fraco ou inexistente da rede GSM. O principal objetivo de efetuar o monitoramento do nível de corpos d'água foi atendido e protótipo apresentou uma faixa útil similar ao WL-15, um linígrafo comercial produzido pela Global Waters.

Quanto ao *web-service*, todos os objetivos foram atendidos. Acrescentou-se uma funcionalidade adicional, de cadastrar usuários, embora não tenha sido implementado um canal seguro para a permuta dos dados, oque pode comprometer a segurança das informações. A utilização de *XML Schema* com JAXB facilitou a validação dos dados enviados ao *web-service* e mostrou-se eficiente e simples de usar. Devido a natureza deste web-service, é possível que qualquer dispositivo que implemente a requisição HTTP necessária possa registrar medições, podendo assim, criar dispositivos com telemetria para integrar com *dataloggers* existentes.

Durante a implementação do software embarcado foi possível verificar as características e peculiaridades da programação para micro controladores, devidos as limitações dos recursos no hardware, como memória, velocidade de processamento ou ainda baixa capacidade para depuração. A utilização da arquitetura Restful demonstrou-se eficiente ao longo do trabalho, pois facilitou a comunicação entre o protótipo montado e o *web-service.*

Uma dificuldade adicional refere-se a qualidade da documentação relacionada à programação do módulo Telit e a fraca capacidade de depuração do código. Embora tenha sido fácil conectar HyperTerminal ao kit de desenvolvimento da Sparkfun, não haviam instruções ou tutoriais específicos para o módulo Telit GE865-QUAD. Outra limitação que provocou a necessidade de alteração do código esta relacionado do programa Round Solutions GSM Terminal que não permite o envio de *scripts* Python maior que seis kilobytes.

4.1 EXTENSÕES

Algumas sugestões de extensões deste trabalho são listadas a seguir:

a) melhorar a recepção do sinal GSM ou utilizar outro método para efetuar a transmissão;
b) aumentar a faixa útil do sensor utilizado;
c) permitir ao usuário obter informações sobre o módulo através do *web-service*;
d) permitir ao sistema efetuar o cálculo da curva-chave por métodos iterativos a partir de amostragens da vazão e nível do corpo d'água;
e) implementar um canal seguro para a permuta dos dados;
f) utilizar certificados digitais para garantir a identidade do protótipo.

REFERÊNCIAS BIBLIOGRÁFICAS

ACRONAME ROBOTICS. **Sonar ranging primer**. [S.l.], 2011. Disponível em: <http://www.acroname.com/robotics/info/articles/sonar/sonar.html>. Acesso em: 10 set. 2011.

CATUNDA, Marco. **Python**: guia de consulta rápida. São Paulo: Novatec, 2001.

FELLER, Nadjia J. **Estendendo Rest-Unit:** geração baseada em U2TP de *drivers* e dados de teste para *RESTful Web Services*. 2010. 77 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Curso de Ciência da Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre. Disponível em: <http://www.lume.ufrgs.br/bitstream/handle/10183/26353/000757814.pdf?sequence=1>. Acesso em: 10 set. 2011.

GARCIA, Karla M. **Detecção e recuperação de falhas em web services**. 2007. 72 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis. Disponível em: <http://www.projetos.inf.ufsc.br/arquivos_projetos/projeto_581/karla_tcc_vs2.doc>. Acesso em: 12 set. 2011.

GLOBAL WATERS. **WL15 water level logger user guide**. [S.l.], 2011. Disponível em: <www.globalw.com/downloads/WL15/WL15manual.pdf>. Acesso em: 15 str. 2011.

HILGENSTIELER, Fernando. **Protótipo de software para monitoração do cabeçalho do protocolo HTTP em uma rede TCP/IP.** 2003. 50 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de ciências exatas e naturais, Fundação Universidade Regional de Blumenau, Blumenau. Disponível em: <http://campeche.inf.furb.br/tccs/2003-II/2003-2fernandohilgenstielervf.pdf>. Acesso em: 12 set. 2011.

LIN, Tai-Wei. **Architecture for XML Binding (JAXB)**: a primer. [S.l.], 2002. Disponível em: <http://java.sun.com/developer/technicalArticles/xml/jaxb>. Acesso em: 9 set. 2011.

LOPES, Wagner L. **Sistema de telemetria e automação remota usando a rede GSM/GPRS**. 2008. 64 f. Trabalho de Conclusão de Curso (Graduação em Ciências da Computação) – Curso de Ciência da Computação, Universidade Luterana do Brasil, Gravataí. Disponível em: <https://gravatai.ulbra.tche.br/tcc/obtem_documento.php?documento=258>. Acesso em: 12 set. 2011.

LUTZ, Mark; ASCHER, David. **Programming Python**. 2nd ed. Sebastopol: O´Reilly & Associates, 2001.

OLIVEIRA, Lauro A. G. **Aplicação para gerenciamento de arquivos XML como suporte à interoperabilidade entre sistemas proprietários**. 2006. 112 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis. Disponível em: <http://www.pergamum.udesc.br/dados-bu/000000/000000000005/00000526.pdf >. Acesso em: 10 set. 2011.

OTT HYDROMETRY. **Thalimedes**. [S.l.], [1996?]. Disponível em: <http://meteo-tech.co.il/e-Catalog/pdf/thalimedes.pdf>. Acesso em: 13 set. 2011.

_____ **RLS**. [S.l.], 2007. Disponível em: <http://www.aqualab.com.au/files/95/brochure.pdf>. Acesso em: 13 set. 2011.

PARZIALE, Lydia et al. **TCP/IP tutorial and technical overview**. 8th. ed. [Estados Unidos?]: IBM International Technical Support Organization, 2006. 974 p.

PORTO, Rubem L. L.; ZAHED FILHO, Kamel; SILVA, Ricardo M. **Hidrologia aplicada**: medição de vazão e curva - chave. São Paulo: Departamento de Engenharia Hidráulica e Ambiental da USP, 2001. 48 p.

RABELLO, Ladislau M.; CRUVINEL, Paulo E.; DENARDIN, José E. **Linígrafo automatizado com microcontrolador**. São Carlos: EMPRAPA, 1994.

RUELA, Pedro H. O. **Solução web services**. 2008. 66 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Curso de Ciência da Computação, Faculdade de Jaguariúna, Jaguariúna. Disponível em: <http://bibdig.poliseducacional.com.br/document/?down=165>. Acesso em: 15 set. 2011.

SEIXAS FILHO, Constantino. **Comunicação através de sockets sobre TCP/IP**. [Belo Horizonte], [2002?]. Disponível em: <http://www.cpdee.ufmg.br/~seixas/PaginaSDA/Download/SDADownload.htm>. Acesso em: 12 set. 2011.

SIMION, Dănuţ-Octavian. Java facilities in processing XML files: JAXB and generating PDF reports. **Informatica Economică**, Roménia, v. 47, n. 3, p.109-113, 10 out. 2008. Disponível em: <http://revistaie.ase.ro/content/47/20simion.pdf>. Acesso em: 10 out. 2011.

SOUZA, Paulo K. **Produtos e equipamentos**: documento final. Porto Alegre: CGEE, 2003.

SPARKFUN. **GE865 evaluation board**. [S.l.], 2011. Disponível em: <http://www.sparkfun.com/products/9342>. Acesso em: 19 set. 2011.

SVERZUT, José U. **Redes GSM, GPRS, EDGE e UMTS**: evolução a caminho da terceira geração (3G). São Paulo: Érica, 2005.

TATO. **MaxSonar EZ-1**. [S.l.], 2005. Disponível em: <http://www.tato.ind.br/files/EZ1.pdf>. Acesso em: 13 out. 2011.

TAURION, Cézar. **Linux em tempo real**. [S.l.], 2008. Disponível em: <http://www2.eletronica.org/artigos/eletronica-digital/linux-em-tempo-real>. Acesso em: 13 set. 2011.

TELIT WIRELESS SOLUTIONS. **GE865-QUAD product description**. [S.l.], 2010. Disponível em: <http://www.telit.com/module/infopool/download.php?id=1591>. Acesso em: 11 ago. 2011.

TUCCI, Carlos E. M. et al. (Org.). **Hidrologia**: Ciência e Aplicação. 2. ed. Porto Alegre: Editora da Universidade/UFRGS, 2000. 943 p.

VIEIRA, Marco L. O. **Estudo e construção de uma solução orientada à serviços aplicado ao caso de uso de automação de postos de combustíveis**. 2007. 74 f. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) - Curso de Sistemas de Informação, Faculdade Cenecista Nossa Senhora dos Anjos, Gravataí. Disponível em: <http://postomobile.googlecode.com/files/Versão%20Boa.pdf>. Acesso em: 12 set. 2011.