UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA

CAIO ARCE NISHIBE
LUÍS GUILHERME BERGAMINI MENDES
RENATO GIRARDI GASOTO

SMGR: SISTEMA DE MONITORAMENTO E GESTÃO REMOTOS DE MAQUINÁRIO EM CANTEIROS DE OBRAS

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2012

CAIO ARCE NISHIBE
LUÍS GUILHERME BERGAMINI MENDES
RENATO GIRARDI GASOTO

SMGR: SISTEMA DE MONITORAMENTO E GESTÃO REMOTOS DE MAQUINÁRIO EM CANTEIROS DE OBRAS

Dissertação apresentada à disciplina de Trabalho de Conclusão de Curso 2, do Curso Superior de Engenharia de Computação do Departamento Acadêmico de Informática - DAINF - e do Departamento Acadêmico de Eletrônica - DAELN - da Universidade Tecnológica Federal do Paraná - UTFPR, como requisito parcial para obtenção do título de Engenheiro.

Orientador: Profº. Dr. Paulo Cézar Stadzisz

CURITIBA
2012

AGRADECIMENTOS

Agradecemos primeiramente a Deus pela força espiritual para a realização desse trabalho.

À toda comunidade da Universidade Tecnológica Federal do Paraná (UTFPR) pelo apoio durante o curso.

Aos colegas que deram apoio nos momentos difíceis e estavam juntos para comemorar as conquistas durante essa longa trajetória.

Ao Professor Doutor Paulo Cézar Stadzisz pela orientação deste trabalho e pelos momentos de aprendizado.

Ao Professor Doutor Dario Eduardo Amaral Dergint pelos momentos de aprendizado e, principalmente, por ter permitido que usássemos o laboratório para desenvolver este Trabalho de Conclusão de Curso durante a paralisação que ocorreu na instituição.

Por último, mas não menos importante, agradecemos as nossas famílias, porque, sem o apoio delas, certamente não conseguiríamos vencer esse desafio.

RESUMO

NISHIBE, Caio A.; MENDES, Luís G. B.; GASOTO, Renato G.. SMGR: Sistema de Monitoramento e Gestão Remotos de Maquinário em Canteiros de Obras. 113 f. Trabalho de Conclusão de Curso - Curso Superior de Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Curitiba 2012.

A gestão e o monitoramento preventivo de maquinário em obras sempre foi considerado importante, porém não crítico. Com o surgimento de tecnologias de comunicação sem fio e a Internet, está ocorrendo uma mudança de paradigma na indústria. Através de monitoramento contínuo, é possível que as empresas sejam mais eficientes, mais produtivas e ofereçam maior segurança para seus funcionários. Esse trabalho propõe um sistema baseado em arquitetura cliente/servidor para o monitoramento e gestão remotos de maquinário em canteiros de grandes obras. Um módulo embarcado, instalado em cada uma das máquinas, envia informações de sensores diversos para um servidor central, que guarda os dados em um banco de dados. Através de uma interface web, os gestores terão fácil acesso a todas as informações sobre o maquinário, permitindo um maior foco em análise e planejamento do que na coleta custosa dos dados.

Palavras-chave: Monitoramento. Gestão. Canteiro de Obras. Windows Embedded. Sistema Embarcado.

ABSTRACT

NISHIBE, Caio A.; MENDES, Luís G. B.; GASOTO, Renato G.. SMGR: Remote Monitoring and Management System for Construction Machinery in Construction Sites. 113 f. Trabalho de Conclusão de Curso - Curso Superior de Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Curitiba 2012.

The management and predictive monitoring of machinery in construction sites were always considered to be important, but not critical. With the increased availability of wireless communication technologies and the Internet, a paradigm shift is being observed throughout the industry. By means of a continuous monitoring, it is possible for companies to be more efficient, more productive, and provide better safety for its employees. This work proposes a system based on a client/server architecture for the management and monitoring of machinery in large construction sites. An embedded module installed in each machine sends information about a variety of sensors to a central server, which in turn stores all the data in a database. Through a web interface, managers will have easy access to all information about the machinery, allowing them to focus on analysis and planning rather than the costly process of collecting the data.

Keywords: Monitoring. Management. Construction Sites. Windows Embedded. Embedded System.

LISTA DE FIGURAS

1 Rover Mars Curiosity . . . . . p. 22

2 Arquitetura de Sistemas Embarcados . . . . . p. 23

3 Tipos de Microcontroladores mais Utilizados . . . . . p. 24

4 Tipos de Sistemas Operacionais Empregados em Aplicações Embarcadas p. 25

5 Modelos de Sistemas Operacionais para Sistemas Embarcados . . . . . p. 26

6 Descrição dos conectores frontais do eBox 3310A . . . . . p. 30

7 Descrição dos conectores traseiros do eBox 3310A . . . . . p. 30

8 Esquemático do eBox 3310A . . . . . p. 31

9 Captura de tela da BIOS do eBox 3310A . . . . . p. 32

10 Módulo Wi-Fi™ USB Wyse VT6656 . . . . . p. 32

11 Funcionamento básico do GPS utilizando três satélites . . . . . p. 35

12 Planos orbitais da constelação de GPS . . . . . p. 36

13 Segmentos do GPS . . . . . p. 36

14 Estações do segmento de controle do GPS . . . . . p. 37

15 Módulo GPS EM-411 . . . . . p. 38

16 Esquemático do EM-411 . . . . . p. 39

17 Elementos de um Microcontrolador . . . . . p. 42

18 Pacote PDIP do ATmega 328P . . . . . p. 42

19 Arduino . . . . . p. 43

20 Visão Geral da CLI . . . . . p. 49

21 Diagrama de Casos de Uso do SMGR . . . . . p. 54

22 Diagrama de Classe do Módulo Embarcado SMGR . . . . . p. 59

23 Diagrama de Sequência da Classe *Application* . . . . . p. 60
24 Diagrama ER do Banco de Dados . . . . . p. 62
25 Fluxo Geral da Interface da Aplicação Web (camada de usuário) . . . . . p. 63
26 Instalação do CE 6.0 . . . . . p. 66
27 Assistente de Criação de SO . . . . . p. 67
28 Catálogo de componentes . . . . . p. 68
29 Gerente Configurações . . . . . p. 70
30 Imagem sendo compilada . . . . . p. 71
31 Desktop padrão do Windows CE 6.0 . . . . . p. 71
32 Propriedades do SDK . . . . . p. 72
33 Captura de tela da ferramenta DiskPrep . . . . . p. 74
34 Esquemático de circuito do *driver* MAX-232 para o GPS . . . . . p. 77
35 Advantech 4711A . . . . . p. 78
36 MAX232 . . . . . p. 79
37 *Driver* Módulo de Aquisição . . . . . p. 79
38 Diagrama das Operações do *Web Service* . . . . . p. 81
39 Esquemático da Placa . . . . . p. 87
40 Desenho da Placa . . . . . p. 88
41 Visão Superior da Placa . . . . . p. 88
42 Visão Lateral da Placa Conectada ao eBox . . . . . p. 89
43 Módulo Embarcado SMGR - Visão Interna . . . . . p. 89
44 Detalhe Módulo Embarcado SMGR - Visão Interna . . . . . p. 90
45 Módulo Embarcado SMGR . . . . . p. 91
46 Página de Login . . . . . p. 91
47 Dashboard . . . . . p. 92
48 Página de Dispositivos . . . . . p. 92

49 Página de Visualização da Posição, Sensores e Propriedades de um Dispositivo . . . . . p. 93

50 Página do Mapa . . . . . p. 93

51 Página de Usuários . . . . . p. 94

52 Diagrama de Sequência da Classe `GPSReader` . . . . . p. 102

53 Diagrama de Sequência da Classe `SensorsReader` . . . . . p. 103

54 Diagrama de Sequência do Método `GetClientID` da Classe `InformationDatabase` . . . . . p. 104

55 Diagrama de Sequência do Método `GetLastValidGPSLatitude` da Classe `InformationDatabase` . . . . . p. 105

56 Diagrama de Sequência do Método `GetLastValidGPSLongitude` da Classe `InformationDatabase` . . . . . p. 106

57 Diagrama de Sequência do Método `GetPollingInterval` da Classe `InformationDatabase` . . . . . p. 107

58 Diagrama de Sequência do Método `GetServerAddress` da Classe `InformationDatabase` . . . . . p. 108

59 Diagrama de Sequência do Método `SetClientID` da Classe `InformationDatabase` . . . . . p. 109

60 Diagrama de Sequência do Método `SetLastValidGPSLatitude` da Classe `InformationDatabase` . . . . . p. 110

61 Diagrama de Sequência do Método `SetLastValidGPSLongitude` da Classe `InformationDatabase` . . . . . p. 111

62 Diagrama de Sequência do Método `SetPollingInterval` da Classe `InformationDatabase` . . . . . p. 112

63 Diagrama de Sequência do Método `SetServerAddress` da Classe `InformationDatabase` . . . . . p. 113

LISTA DE TABELAS

1 Explicação de mensagem GPGGA . . . . . p. 40

2 Componentes de um Descritor WSDL . . . . . p. 44

LISTA DE SIGLAS E ABREVIATURAS

AES	Advanced Encryption Standard
API	Application Programming Interface
BIOS	Basic Input/Output System
BSP	Board Support Package
CE	Compact Edition
CI	Circuito Integrado
CIL	Common Intermediate Language
CLS	Common Language Specification
CTS	Common Types System
DAO	Data Access Object
DBML	Database Management Language
DDR2	Double Data Rate 2
DSPs	Digital Signal Processor
ER	Entidade Relacionamento
ERP	Enterprise Resource Planning
FOC	Full Operational Capacity
FPGA	Field Programmable Gate Array
GPS	Global Positioning System
IDE	Integrated Drive Electronics
IIS	Internet Information Services

IOC	Initial Operacional Capability
LAN	Local Area Network
LCD	Liquid Crystal Display
LED	Light-emitting diode
MCS	Master Control Station
NASSL	Network Application Service Specification Language
NMEA	National Marine Electronics Association
PA	Power Amplifier
PC	Personal Computer
PDF	Portable Document Format
PDIP	Plastic Dual Inline Package
PPS	Precise Positioning System
PXE	Preboot Execution Environment
RAM	Random Access Memory
RTOS	Real-Time Operating System
SD	Secure Digital
SDL	Service Description Language
SMGR	Sistema de Monitoramento e Gestão Remotos de Maquinário em Canteiros de Obras
SOAP	Simple Object Access Protocol
SPS	Standard Positioning System
SQL	Structured Query Language
TI	Tecnologia da Informação
TKIP	Temporal Key Integrity Protocol

TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VES	Virtual Execution System
VESA	Video Electronics Standards Association
VGA	Video Graphics Array
W3C	World Wide Web Consortium
WEP	Wired Equivalent Privacy
WPA	Wi-Fi™ Protected Access
WSDL	Web Service Description Language
XML	eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO p. 15

1.1 Contexto . . . . . p. 15

1.2 Objetivos . . . . . p. 16

1.2.1 Objetivo Geral . . . . . p. 16

1.2.2 Objetivos Específicos . . . . . p. 16

1.3 Justificativa . . . . . p. 17

1.4 Metodologia . . . . . p. 19

1.5 Estrutura do Documento . . . . . p. 19

2 FUNDAMENTAÇÃO TEÓRICA p. 21

2.1 Sistemas Embarcados . . . . . p. 21

2.2 Sistemas Operacionais de Tempo Real (RTOS) . . . . . p. 24

2.3 Microsoft Windows Embedded . . . . . p. 27

2.4 eBox 3310A . . . . . p. 28

2.4.1 Características físicas . . . . . p. 29

2.4.2 BIOS e opções de *Boot* . . . . . p. 31

2.5 Módulo Wi-Fi™ USB Wyse VT6656 . . . . . p. 32

2.6 Módulo GPS . . . . . p. 34

2.6.1 Visão geral do sistema de GPS . . . . . p. 34

2.6.2 Módulo de GPS EM-411 . . . . . p. 37

2.7 Módulo de Aquisição de Sinais . . . . . p. 40

2.8 *Web Service* . . . . . p. 43

2.8.1 WSDL . . . . . p. 43
2.8.2 SOAP . . . . . p. 45
2.9 Banco de Dados . . . . . p. 45
2.9.1 Modelo relacional . . . . . p. 46
2.10 .NET Framework . . . . . p. 47
2.10.1 Principais características do .NET Framework . . . . . p. 47
2.10.1.1 Interoperabilidade . . . . . p. 47
2.10.1.2 CLR . . . . . p. 47
2.10.1.3 Independência de linguagem . . . . . p. 48
2.10.1.4 Portabilidade . . . . . p. 48
2.11 CLI - Common Language Infrastructure . . . . . p. 48
2.12 Considerações sobre o capítulo . . . . . p. 50

3 ESPECIFICAÇÃO p. 51
3.1 Especificação do Sistema . . . . . p. 51
3.1.1 Análise de Requisitos . . . . . p. 51
3.1.1.1 Requisitos Funcionais . . . . . p. 51
3.1.1.2 Requisitos Não Funcionais . . . . . p. 52
3.1.2 Diagrama de Casos de Uso . . . . . p. 53
3.2 Especificação do Módulo Embarcado . . . . . p. 53
3.2.1 Microsoft Windows Embedded . . . . . p. 54
3.2.2 Especificação de *Hardware* . . . . . p. 56
3.2.3 Especificação de *Software* . . . . . p. 56
3.2.3.1 Diagrama de Classes . . . . . p. 56
3.2.3.2 Diagramas de Sequência . . . . . p. 58
3.3 Especificação do Banco de Dados . . . . . p. 58
3.4 Especificação da Aplicação Web . . . . . p. 61

3.5 Considerações sobre o capítulo . . . . . p. 61

4 DESENVOLVIMENTO DO SISTEMA p. 64

4.1 Customização do Windows CE . . . . . p. 64

4.1.1 Ambiente de desenvolvimento . . . . . p. 64

4.1.1.1 Instalação das ferramentas de desenvolvimento . . . . . p. 65

4.1.2 Desenvolvimento da imagem do Windows CE 6.0 R3 . . . . . p. 67

4.1.3 Criando um SDK customizado . . . . . p. 71

4.1.4 Processo de *boot* pelo cartão *Compact Flash* . . . . . p. 72

4.2 Módulo Wi-Fi™ USB Wyse VT6656 . . . . . p. 73

4.3 Módulo EM-411 . . . . . p. 76

4.4 Módulo de Aquisição de Sinais . . . . . p. 77

4.5 *Web Service* . . . . . p. 80

4.6 *Software* para o Módulo Embarcado . . . . . p. 81

4.7 Aplicação Web . . . . . p. 82

4.7.1 API Google Maps . . . . . p. 85

4.7.1.1 API Google Maps Javascript V3 . . . . . p. 85

4.7.1.2 Reimers Google Maps .NET Control . . . . . p. 86

4.8 Considerações sobre o capítulo . . . . . p. 86

5 RESULTADOS p. 87

5.1 Módulo Embarcado . . . . . p. 87

5.2 Aplicação Web . . . . . p. 90

6 CONCLUSÕES p. 95

Referências p. 98

Anexo A – Diagramas Módulo Embarcado p. 102

1 INTRODUÇÃO

Este capítulo apresenta em que contexto está inserido este Trabalho de Conclusão de Curso, bem como seus objetivos, justificativa, metodologia e de que forma está estruturado este documento.

1.1 Contexto

O acompanhamento de maquinário em canteiros de grandes obras é um problema de alto impacto para empresas (VELOSO, 2009). Quanto maior a obra, maior é a complexidade de manter controle sobre a utilização destes recursos. Geralmente são enviadas equipes a regiões de difícil acesso para percorrer longas distâncias, como é típico em obras de porte extenso como dutos de gás, ao longo de centenas de quilômetros. Tal atividade demanda uma grande quantidade de tempo, gasta recursos e causa desgaste de todos os funcionários envolvidos no processo (BAXTER; JESUS, 2006).

Observa-se que, geralmente, esse processo ocorre de forma manual, quase sem nenhum tipo de acompanhamento de tecnologias da informação. Entretanto, a informatização das mais diversas atividades vem crescendo com a popularização dos computadores e disponibilidade de conexões à Internet com custo reduzido. Muitas áreas já fazem uso dessas tecnologias para automatizar e facilitar os processos envolvidos, porém percebe-se que a área de construção ainda não aderiu a esses recursos.

Uma forma de solucionar o problema é a gestão remota das máquinas (TUCKER, 2005). Sensores de localização GPS, de direção, de quantidade de combustível no tanque e de quilometragem, entre outros, podem ser instalados em cada um dos equipamentos do canteiro de obras. Tais sensores enviam seus dados a um módulo instalado na máquina, que por sua vez processa as informações e as envia a um servidor central através de uma rede Wi-Fi™ industrial de alta velocidade presente no local. Uma vez no servidor, as informações podem ser acessadas por uma enorme gama de dispositivos, indo desde computadores pessoais aos *smartphones* e *tablets*, conforme aponta a tendência de mercado

atual.

Através de um sistema com essas características, os responsáveis pelo controle de obras em empresas podem ter fácil acesso ao estado de todos os equipamentos distribuídos pelo canteiro, em qualquer lugar com disponibilidade de Internet e em qualquer dispositivo que conecte-se a ela. As repercussões disso incluem economia de capital, agilidade na tomada de decisões e redução do tempo de vistorias, que, consequentemente, permite um melhor uso dos funcionários para atividades de maior urgência. De forma geral, a empresa será mais produtiva e com isso mais lucrativa.

Existem no mercado soluções similares voltadas principalmente para a gestão de equipamentos agrícolas. Um exemplo é a solução "MODULO Máquina" da TI-AGRO (2012), que consiste em um monitor de recursos de máquinas agrícolas. É capaz de coletar dados sobre o consumo de combustível e atividades diárias, bem como gerar relatórios e manter um controle sobre o estado de manutenção dos equipamentos. Na área de construção, como é a proposta desse trabalho, não foi encontrada nenhuma solução com as características propostas, apenas sistemas baseados no modelo ERP com inserção manual de dados.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral deste Trabalho de Conclusão de Curso é conceber um sistema de gestão e monitoramento remotos de maquinário em canteiros de obras.

1.2.2 Objetivos Específicos

Os objetivos específicos deste Trabalho de Conclusão de Curso são:

- Aprofundar conhecimentos sobre as tecnologias Microsoft Windows Embedded.
- Implementar um sistema embarcado capaz de enviar dados de sensores para um servidor central através de uma rede Wi-Fi™ industrial.
- Implementar uma arquitetura em nuvem para o acesso remoto dos dados coletados em campo.

- Implementar um serviço web (*Web Service*) capaz de exibir as informações coletadas pelo módulo em campo, acessível através de uma grande variedade de dispositivos (computadores *desktop*, *notebooks*, *smartphones*, *tablets*, etc).

1.3 Justificativa

Durante décadas, a manutenção preventiva de equipamentos teve uma existência cíclica na indústria. Reorganizações na estrutura das companhias e cortes de pessoal causam o seu parcial esquecimento, para logo após uma falha crítica voltar a ser colocada em prática. A manutenção preventiva sempre foi considerada importante, porém não essencial. A curto prazo não apresenta grandes impactos, porém, a medida em que o tempo avança, o uso de maquinário e sua natural degradação causa grandes gastos. Dessa forma, observa-se que as companhias adotam posturas mais imediatas em relação ao acompanhamento de seu maquinário, ao invés de realizar um monitoramento a longo prazo (BAXTER; JESUS, 2006).

Com o surgimento de novas tecnologias e a Internet, um novo tipo de manutenção preventiva está surgindo no mercado: monitoramento e gestão remotos. Tal método elimina a tarefa manual, cansativa e custosa de coletar dados sobre maquinário, permitindo que os trabalhadores concentrem seus esforços em analisar dados, ao invés de coletá-los. Com acesso a Internet, os funcionários têm acesso aos dados em qualquer lugar e em qualquer horário. O monitoramento remoto também permite que os dados possam ser analisados com mais facilidade por terceiros, em locais distantes do maquinário em si. Os benefícios do monitoramento e gestão remotos tornam-se ainda mais evidentes quando combinados com questões críticas da atualidade (BAXTER; JESUS, 2006), como:

- Aumento de competitividade: empresas ao redor do mundo buscam formas de produzir mais e reduzir custos operacionais, de forma a obter mais lucro.
- Aumento de custos: o preço de combustíveis para se percorrer e avaliar uma grande obra, por exemplo, impacta drasticamente no lucro final.
- Diminuição da força de trabalho: um grande percentual de trabalhadores não realiza mais funções básicas de manutenção, preferindo trabalhos mais especializados, portanto o custo de tais serviços sobe.

Todos esses fatores estão trazendo uma mudança de paradigma para a indústria, baseada no princípio de que a empresa será mais eficiente se for mais inteligente na forma

de monitoramento de seu maquinário. Ao invés de se avaliar o equipamento de acordo com um calendário, por exemplo mensalmente, avalia-se de forma contínua através de um sistema computacional. Sem considerar esse novo paradigma, essas tarefas são difíceis e custosas. Equipamentos críticos são geralmente de difícil acesso, exigindo precauções de segurança em suas adjacências. Além disso, muitas companhias não possuem pessoal suficiente para realizar o trabalho de manutenção, contratando empresas terceirizadas e gastando com viagens até o local da obra. Com o novo paradigma, as métricas são obtidas com a frequência que se desejar (BAXTER; JESUS, 2006).

A chave para essa mudança está nas tecnologias de comunicação sem fio. Com dispositivos *wireless* e a Internet, agora é possível coletar dados remotamente e acessá-los em qualquer local (BAXTER; JESUS, 2006). Em resumo, as principais vantagens de tecnologias sem fio são:

- Disponibilidade comercial.
- Baixo custo.
- Elimina a necessidade de cabeamento longo.
- Requer pouca mudança na estrutura de TI.
- Permite acesso aos dados em qualquer lugar.

A medida em que a manutenção preventiva muda de paradigma, diversos benefícios tornam-se visíveis. Funcionários das empresas podem passar mais tempo analisando dados do que coletando-os; Informações podem ser facilmente coletadas, mesmo em dispositivos de difícil acesso; Tem-se um aumento na segurança dos funcionários, pois expõe os trabalhadores a menos riscos em campo; A coleta de dados torna-se automática, incluindo posição, combustível, velocidade, pressão, temperatura, etc; Um banco de dados torna possível armazenar informações sobre toda a estrutura da obra (BAXTER; JESUS, 2006).

Com isso, percebe-se que o monitoramento e gestão remotos de maquinário em obras faz com que empresas sejam muito mais eficientes e tenham um maior controle sobre seu equipamento, podendo focar esforços em análise de dados e planejamento para obter mais produtividade.

1.4 Metodologia

Para alcançar os objetivos propostos, este Trabalho de Conclusão de Curso se desenvolveu nas etapas descritas a seguir:

- revisão da literatura e levantamento de material teórico necessário.
- aquisição de componentes de *hardware*, a partir da determinação da estrutura de *hardware* necessária.
- aquisição de ferramentas de *software*, tais como ambientes de desenvolvimento, sistema operacional para o módulo embarcado, servidores web e bancos de dados.
- implementação do sistema embarcado responsável pela captura das variáveis mensuradas nos sensores e posterior envio ao servidor central da obra.
- implementação da arquitetura em nuvem, responsável pelo recebimento e armazenamento das informações coletadas no módulo localizado em campo.
- implementação de um serviço web (*Web Service*) para permitir a visualização remota das informações armazenadas no servidor.
- testes de *hardware*, etapa em que foram realizadas calibrações e ajustes finais no módulo de coleta de dados.
- testes de *software*, etapa em que foram realizados testes locais e globais na arquitetura de armazenamento das informações, bem como nos dispositivos que acessaram o sistema através do serviço web.
- análise dos resultados e conclusões, etapa em que foi redigida a monografia em sua versão final englobando os resultados obtidos, bem como detalhes de desenvolvimento e dificuldades encontradas durante a execução deste Trabalho de Conclusão de Curso.

1.5 Estrutura do Documento

O Capítulo 2 apresenta os conceitos presentes na literatura que serviram como base na análise e justifcativa das decisões deste Trabalho de Conclusão de Curso. Neste capítulo há uma breve introdução a sistemas embarcados, sistemas operacionais de tempo real, ao

Microsoft Windows Embedded e aos demais conceitos necessários para a realização deste Trabalho de Conclusão de Curso.

O Capítulo 3 contém a especificação do SMGR. São apresentadas a especificação do sistema, do módulo embarcado, da aplicação web e do banco de dados.

O Capítulo 4 descreve como este Trabalho de Conclusão de Curso foi desenvolvido.

O Capítulo 5 apresenta os resultados obtidos.

O Capítulo 6 contém as conclusões e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta uma breve introdução a Sistemas Embarcados, Sistemas Operacionais de Tempo Real, ao Microsoft Windows Embedded e aos demais conceitos necessários para o desenvolvimento deste trabalho.

2.1 Sistemas Embarcados

Segundo Stadzisz e Renaux (2008), um sistema embarcado é uma combinação de elementos de *hardware* e *software* e, eventualmente, componentes mecânicos, dedicados a uma tarefa muito específica. Na maioria das vezes, o usuário final nem sequer nota a presença do elemento computacional. Com a queda de preços e o desenvolvimento da eletrônica, sistemas embarcados hoje são facilmente encontrados na maioria dos produtos. Para citar alguns exemplos desses sistemas, basta olhar ao redor: telefones celulares, tocadores de música, controladores de ar-condicionado em carros, microcontroladores em escovas de dentes, o veículo de exploração *Mars Curiosity* (Figura 1); todos possuem *hardware* e *software* dedicados para uma tarefa que, geralmente, contém requisitos muito específicos e necessidade de atendimento em tempo real. Os sistemas embarcados são os maiores consumidores de microprocessadores do planeta, ao contrário da crença popular em computadores *desktop* ou *notebooks*. Todo ano são produzidos em torno de nove bilhões de microprocessadores e somente 150 milhões são utilizados em computadores. Os outros 99 % são destinados a sistemas embarcados (BARR; MASSA, 1999).

Construir um sistema embarcado é uma tarefa que representa um grande desafio. Sistemas desse tipo devem ser extremamente confiáveis, não podendo apresentar falhas durante sua execução. Caso alguma ocorra, os custos são geralmente muito altos, indo de bilhões de dólares no caso de uma pane do veículo em Marte, a uma vida humana em um sistema médico. Tais dispositivos possuem requisitos críticos de performance, tempo e energia. Geralmente a memória é muito limitada, assim como o poder de processamento. Dessa forma, projetar um sistema embarcado envolve uma cuidadosa engenharia

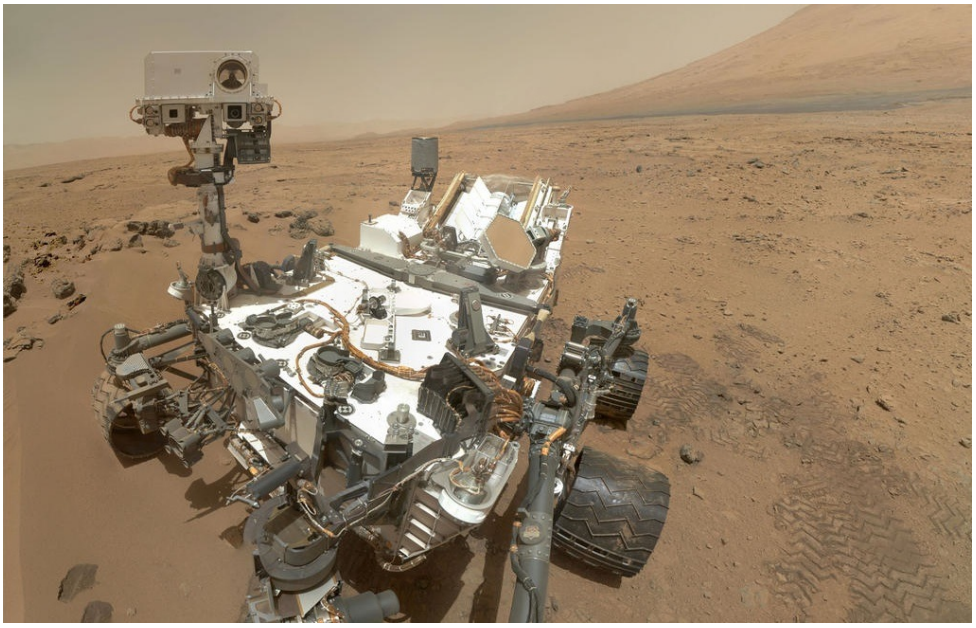

Figura 1: Rover Mars Curiosity
Fonte: `http://mars.jpl.nasa.gov/msl/multimedia/images`

de *software*.

Em contraste, um computador convencional atende propósitos gerais e não possui garantia de tempos de execução (BARR; MASSA, 1999). Ao desenvolver um *software* para um sistema operacional comum, o programador pode partir de certos pressupostos que serão verdadeiros (DEDICATED SYSTEMS EXPERTS, 2009). É seguro considerar, nos dias atuais, que o computador em que o *software* será executado é equipado com um monitor VGA com resolução mínima de 1024x768 pixels, um teclado, um mouse, memória RAM em torno de 512 MB, disco rígido com capacidade suficiente e processador com *clock* em torno de 1GHz. Tais considerações não são válidas para sistemas embarcados. Nesses casos, geralmente, não há monitor ou este é muito pequeno e com baixa resolução, a memória é limitada, não há teclado ou mouse, o processador é mais lento para diminuir custos, etc.

De acordo com Stadzisz e Renaux (2008), os sistemas embarcados apresentam grande heterogeneidade, envolvendo na maioria dos casos a customização de um *hardware* para um *software* específico. Envolvem uma grande variedade de microprocessadores, indo de 8 a 64 bits, além de FPGAs e DSP. Ainda há variações de *clock* em cada uma das principais famílias de arquiteturas, como ARM, x86, MIPS, etc. Tal heterogeneidade é facilmente explicável pela diversidade de necessidades, tecnologias e restrições dos sistemas embarcados.

O projeto de um sistema embarcado envolve uma série de restrições, em contraste com

computadores convencionais (STADZISZ; RENAUX, 2008). Dentre tais restrições, pode-se citar a questão de custos. É muito comum encontrar sistemas embarcados sendo produzidos em larga escala na forma de eletrônicos de consumo, portanto deve-se projetar o sistema para possuir a menor quantidade possível de componentes de *hardware* e *software*, desencadeando restrições em mais áreas. Já as restrições dimensionais envolvem tamanho e peso, uma questão muito importante na fabricação de um telefone celular, por exemplo. Restrições de consumo e autonomia aparecem pois muitos desses sistemas são alimentados por baterias, devido a necessidade de mobilidade, como um aparelho de GPS. Restrições de recursos são geralmente associadas à pouca memória disponível, impactando na forma que o *software* deve ser escrito. Por fim, pode-se citar restrições temporais, já que muitas operações devem ser executadas dentro de limites de tempo bem definidos (STADZISZ; RENAUX, 2008).

Um modelo utilizado para descrever a arquitetura de sistemas embarcados foi apresentado por Renaux (2005) (Figura 2). Cada camada representa um nível de abstração, sendo o desenvolvimento do sistema cada vez mais ágil e confiável à medida em que os níveis são construidos sobre o *hardware*.

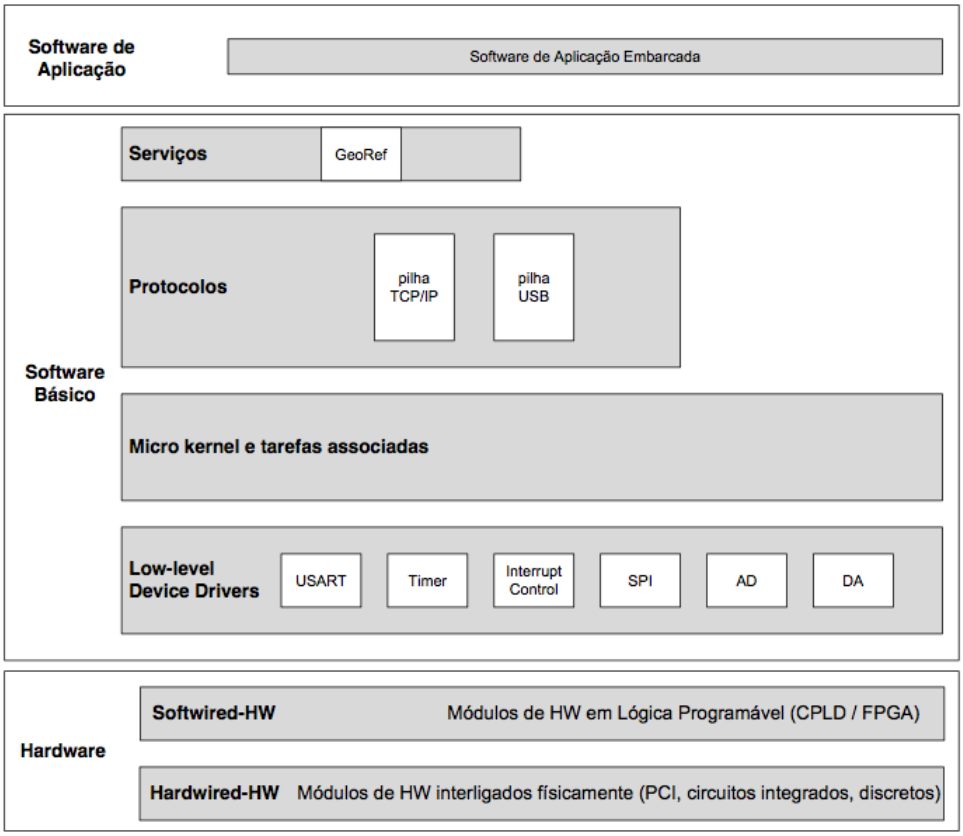

Figura 2: Arquitetura de Sistemas Embarcados
Fonte: Renaux (2005)

Um estudo realizado em 2006 durante a Embedded System Conference, nos Estados Unidos, mostra os tipos de microcontroladores mais utilizados em sistemas embarcados (Figura 3) (GANSSLE, 2006) .

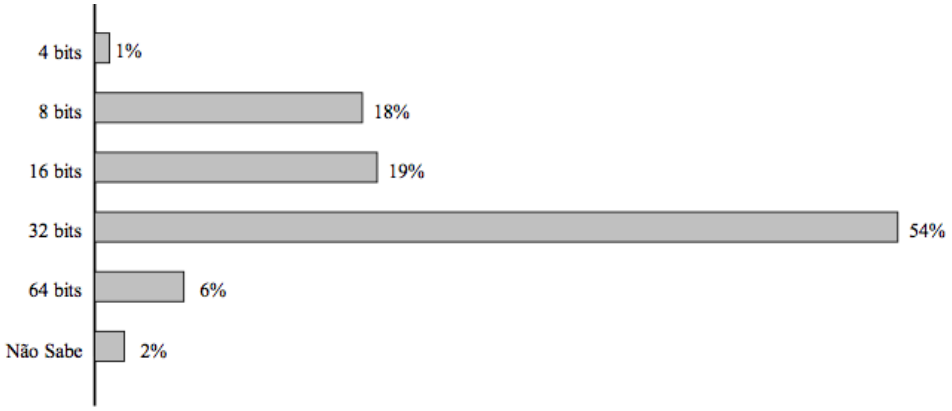

Figura 3: Tipos de Microcontroladores mais Utilizados
Fonte: Ganssle (2006)

Percebe-se uma predominância de processadores de 32 bits, porém ainda com uma grande participação de 8 e 16 bits, devido a certos tipos de projetos serem extremamente sensíveis a custos.

O desenvolvimento de um sistema embarcado ocorre de acordo com as características de cada projeto, não havendo um método aplicável a todos os casos. Inicialmente, faz-se a Concepção do Produto, fase em que o mercado é analisado assim como planos de negócio e viabilidade técnica. Feito isso, segue-se para a Engenharia de Requisitos, em que elabora-se uma especificação completa do sistema, contando inclusive com a construção de protótipos funcionais. O próximo passo da cadeia de desenvolvimento é a Engenharia de Sistema, fase em que o produto é tratado como um todo e tem cada componente estudado, assim como as interdependências. A partir daí parte-se para o desenvolvimento do *software* e do *hardware*, passando pela integração do sistema, testes, documentação e empacotamento final (STADZISZ; RENAUX, 2008). Com isso, percebe-se que há um grande número de atividades e complexidade no desenvolvimento de um produto.

2.2 Sistemas Operacionais de Tempo Real (RTOS)

Projetos de sistemas embarcados geralmente fazem uso de algum tipo de sistema operacional para auxílio no desenvolvimento de aplicações (STADZISZ; RENAUX, 2008). Programar um *software* diretamente em linguagem de máquina (Assembly) pode gerar um código altamente otimizado, porém apresenta um nível de dificuldade muito elevado

e não é produtivo. Um sistema operacional suporta programação em um nível mais alto, traduzindo-se em maior facilidade e produtividade, além de apresentar elevada performance devido aos avançados compiladores existentes. Linguagens de programação geralmente utilizadas são C, C++, Java, além de frameworks como o .NET, da Microsoft. A Figura 4 representa os tipos de sistemas operacionais geralmente empregados em aplicações embarcadas (TURLEY, 2006).

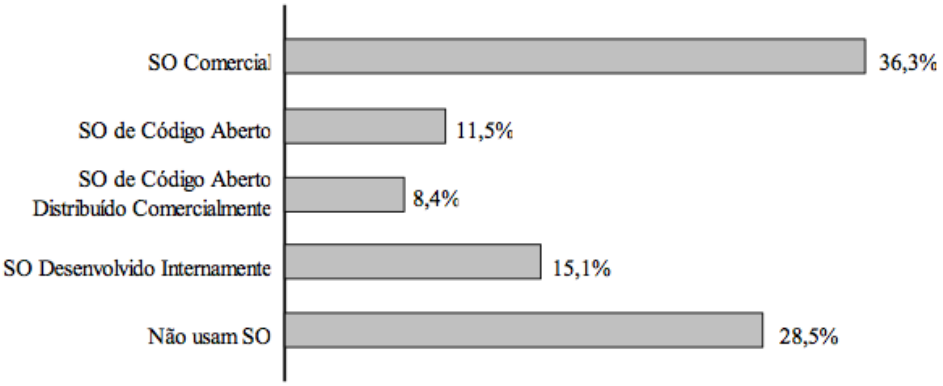

Figura 4: Tipos de Sistemas Operacionais Empregados em Aplicações Embarcadas
Fonte: Turley (2006)

Além disso, a pesquisa apresentada por Turley (2006) mostrou os principais sistemas operacionais próprios para sistemas embarcados (Figura 5).

Um tipo importante de SO geralmente associado a sistemas embarcados são os sistemas operacionais de tempo real, ou *Real-Time Operating Systems* (RTOS). Tais sistemas possuem uma característica fundamental, que é a garantia de execução de uma dada tarefa em tempo preciso (TANENBAUM, 2009). Há certos tipos de sistema que exigem tal comportamento, como por exemplo o acionamento de dispositivos de segurança em uma colisão de veículos. Um sistema operacional comum, sem qualquer garantia de tempos de execução, não poderia ser utilizado para essa finalidade. Geralmente, caracterizam-se os RTOS em dois tipos (TANENBAUM, 2009):

- Tempo Real Crítico: existem prazos absolutos que devem ser cumpridos.
- Tempo Real Não Crítico: o não cumprimento eventual de um prazo é indesejável, porém tolerável.

No centro de um RTOS existe um algoritmo de escalonamento (*scheduling*) mais avançado que em sistemas operacionais comuns, além das garantias de tempo mínimo de latência para o tratamento de interrupções e para a troca de contexto em *threads* ou

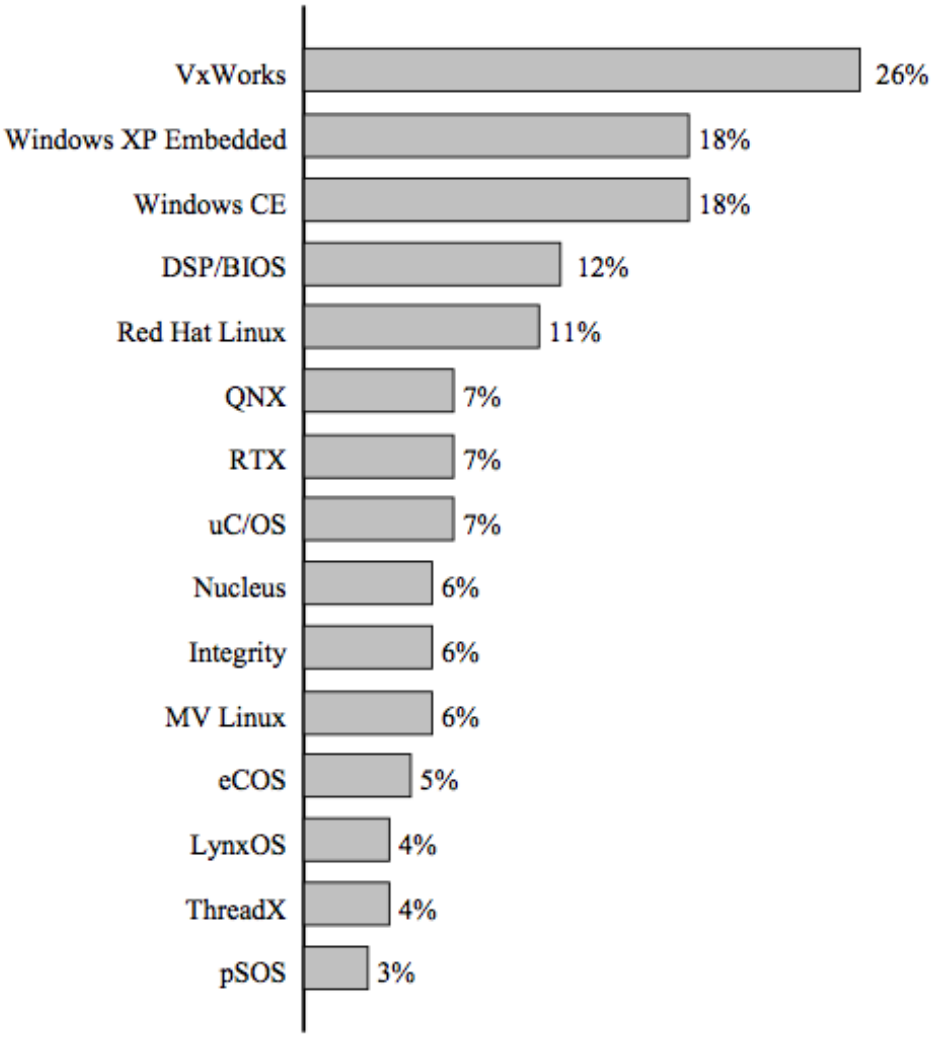

Figura 5: Modelos de Sistemas Operacionais para Sistemas Embarcados
Fonte: Turley (2006)

processos. Vale ressaltar que em um RTOS geralmente executa-se uma única aplicação dedicada, como é característico em sistemas embarcados. Para uma aplicação envolvendo múltiplos aplicativos em constante troca de contexto, um sistema operacional embarcado de maior complexidade, como o Windows Compact 7, seria necessário.

Atualmente, existem diversos fabricantes produzindo sistemas operacionais de tempo real amplamente utilizados no mercado, dentre eles:

- Microsoft, com o Windows Embedded (MICROSOFT, 2012c).
- QNX, com o RTOS Neutrino (QNX, 2012).
- Wind River Systems, com o VxWorks (WIND RIVERS, 2012).

Os sistemas operacionais são utilizados para o gerenciamento de tarefas e fornecer

interfaces para comunicação com o *hardware*. Um elemento muito comum em um SO é o Processo, que é uma unidade de execução definida por um código e conjunto de dados, os quais são processados e produzem uma saída (STADZISZ; RENAUX, 2008). Caso haja a necessidade de um sistema embarcado executar diversos processos, como por exemplo um celular que deve atender chamadas telefônicas e registrar nomes em sua agenda de contatos, o SO deve ser capaz de compartilhar o uso de CPU, uma técnica denominada compartilhamento de tempo (*time sharing*). Com o *time sharing*, cada tarefa ganha um tempo para executar suas atividades e depois fica em um estado de espera para que outra tarefa também possa ser executada, em uma falsa ilusão de simultaneidade. Tal processo é chamado de escalonamento. Sistemas operacionais comuns não fornecem nenhuma garantia de tempo para o escalonamento de seus processos, enquanto em um RTOS isso deve ser previsível.

2.3 Microsoft Windows Embedded

Windows Embedded é uma família de sistemas operacionais desenvolvidos pela Microsoft para uso em sistemas embarcados. As principais versões são:

- Windows Embedded Compact.
- Windows XP/7 Embedded.
- Windows Embedded Point of Service (POS).

O Windows XP/7 Embedded permite que o desenvolvedor customize o sistema operacional, selecionando quais módulos farão parte da imagem final que será distribuída aos usuários (MICROSOFT, 2012c). Essa possibilidade de customização é bastante interessante para organizações que desejam utilizar um Windows personalizado a suas necessidades, porém não se trata de um RTOS, apesar de fazer parte da família Embedded. O Windows Embedded POS é uma versão destinada a pontos de acesso ao usuário, tipicamente cabines de atendimento e máquinas de vendas. Inclui customizações típicas de tais sistemas, como *drivers* para telas sensíveis a toque. Por fim, o Windows Embedded Compact é um RTOS completo e é disponibilizado de forma totalmente modular, sendo possível customizá-lo para uma grande variedade de processadores e usos específicos. Para fins desse projeto, a versão de sistema operacional utilizada deverá ser o Windows Embedded Compact.

No caso do Windows Embedded Compact, o sistema operacional não é simplesmente uma versão reduzida do Windows convencional. Apesar de compartilhar a maioria das ferramentas de desenvolvimento, o *kernel* do Windows Embedded Compact é bastante diferente e não é compatível com as mesmas aplicações que são executadas no Windows para PC (PHUNG, 2008).

A primeira versão do Windows Embedded Compact (CE 1.0) foi lançada em 1996 e marcou o primeiro lançamento de um SO da Microsoft para dispositivos diferentes de computadores pessoais (PAVLOV; BELEVSKY, 2008). Na época, o objetivo era ganhar o mercado de PDAs que estava em crescimento, com diversas empresas criando modelos para atrair consumidores. No ano seguinte foi lançado o Windows CE 2.0, suportando uma maior gama de dispositivos e modelos de processadores. Três anos depois, em 2000, o Windows CE 3.0 trouxe pela primeira vez suporte a tecnologias gráficas avançadas, como o DirectDraw, DirectShow e o Windows Media Player, aos dispositivos embarcados. A versão 4.0, lançada em 2001, expandiu o processamento gráfico ao Direct3D e incluiu ferramentas de gerenciamento de energia, o protocolo SOAP (*Simple Object Access Protocol*), o banco de dados SQL Server CE, entre outros. Atualizações para o CE 4.0 incluiram Bluetooth, IPv6 e telefonia VoIP, além da primeira versão do .NET Compact Framework. Em 2005 a Microsoft anunciou o CE 5.0, com suporte a USB 2.0, Windows Media Player 9, Internet Explorer 6 e melhorias para customização do sistema operacional para aplicações específicas, incluindo um melhor modelo de BSP. A versão 6.0, lançada em 2006, teve a maior quantidade de mudanças desde a primeira versão. A memória virtual de cada processo, antes de 32 MB, foi expandida para 2 GB. O número de processos simultâneos passou de 32 para 32 mil. O *kernel* foi reestruturado de forma a acelerar chamadas de sua API e permitir que novos *drivers* fossem adicionados. Em 2011 foi lançada a versão CE 7.0 do Windows Embedded e, atualmente, está sendo lançada versão 8.0. Foram adicionados suporte a processadores multicore, memória RAM de até 3 GB, tecnologias Microsoft Silverlight, Flash, Internet Explorer 7, visualizadores de PDF, entre outros recursos.

2.4 eBox 3310A

Esta seção trata do eBox 3310A, dispositivo embarcado em que são executados o Windows Embedded Compact 6.0 R3 e o *software* do SMGR. Além de suas características físicas, são discutidas as configurações de sua BIOS.

2.4.1 Características físicas

O kit de desenvolvimento eBox 3310A, fabricado pela ICOP Technology, consiste em um dispositivo computacional compacto destinado a sistemas embarcados e aplicações em que o espaço físico é limitado. Pode ser parafusado em qualquer superfície que respeite o padrão VESA, como é normalmente encontrado na parte traseira de monitores LCD. O eBox 3310A possui um encapsulamento de alumínio que serve como dissipador de calor, eliminando a necessidade de um *cooler*. Dessa forma, a operação do eBox 3310A é completamente silenciosa e menos suscetível a falhas devido a problemas mecânicos com ventilação (PHUNG, 2010).

Equipado com o processador Vortex86DX, que integra periféricos de I/O e 512 MB de memória RAM DDR2, o eBox 3310A fornece poder computacional suficiente para executar o Windows Embedded e outros sistemas operacionais embarcados. Possui portas USB 2.0 que provêm diversas possibilidades de expansão e interfaceamento, além de uma porta Ethernet 10/100M com suporte a *boot* via rede (Figuras 6 e 7) (PHUNG, 2010). O eBox 3310A possui BSPs para todas as versões do Windows CE a partir da 5.0, tornando-o uma plataforma ideal para o desenvolvimento desse projeto.

A seguir encontra-se uma lista com as principais características do dispositivo:

- Design "*fan-less*", sem *cooler*.
- Suporte a fixação em superfícies com padrão VESA.
- Processador Vortex86DX, com clock de 1 GHz.
- Adaptador gráfico XGI Z9s, com suporte a diversas resoluções e frequências de vídeo VGA.
- 512 MB de memória RAM DDR2.
- Conectividade LAN 10/100Mbps R6040.
- Slot para cartão *Compact Flash* e Micro SD.
- 3 portas USB 2.0.
- 2 portas seriais RS-232.
- Entrada (Mic-in) e saída (Line-out) de áudio.

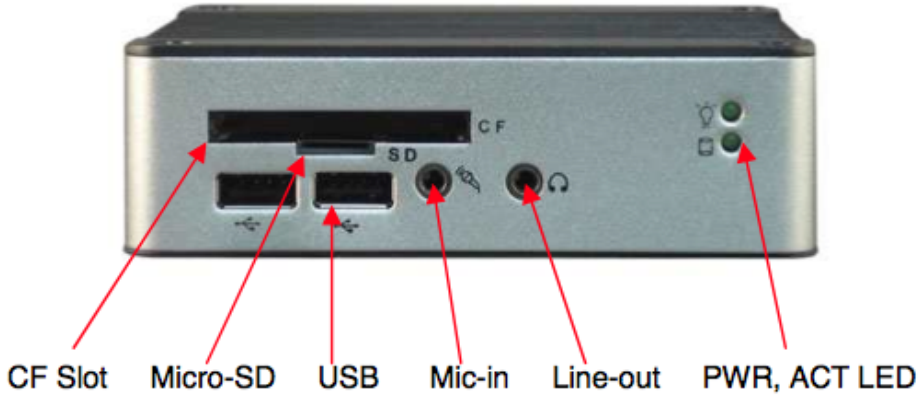

Figura 6: Descrição dos conectores frontais do eBox 3310A
Fonte: Phung (2010)

Figura 7: Descrição dos conectores traseiros do eBox 3310A
Fonte: Phung (2010)

- Suporte ao Windows CE a partir do 5.0.

O eBox 3310A pesa 505 gramas e necessita de uma alimentação de 5V/2A. Opera em faixas de temperatura de 5 a 50 graus Celsius. Seu tamanho físico é de 115x115x35 milímetros (Figura 8).

No ano de 2011, a equipe deste projeto participou da Microsoft Imagine Cup, uma competição para estudantes universitários com propostas para resolver problemas de alto impacto. Através dessa participação, na categoria de sistemas embarcados, a equipe recebeu da Microsoft de forma gratuita o kit de desenvolvimento contendo o eBox 3310A. Dessa forma, foi possível reaproveitar o equipamento para o projeto do SMGR.

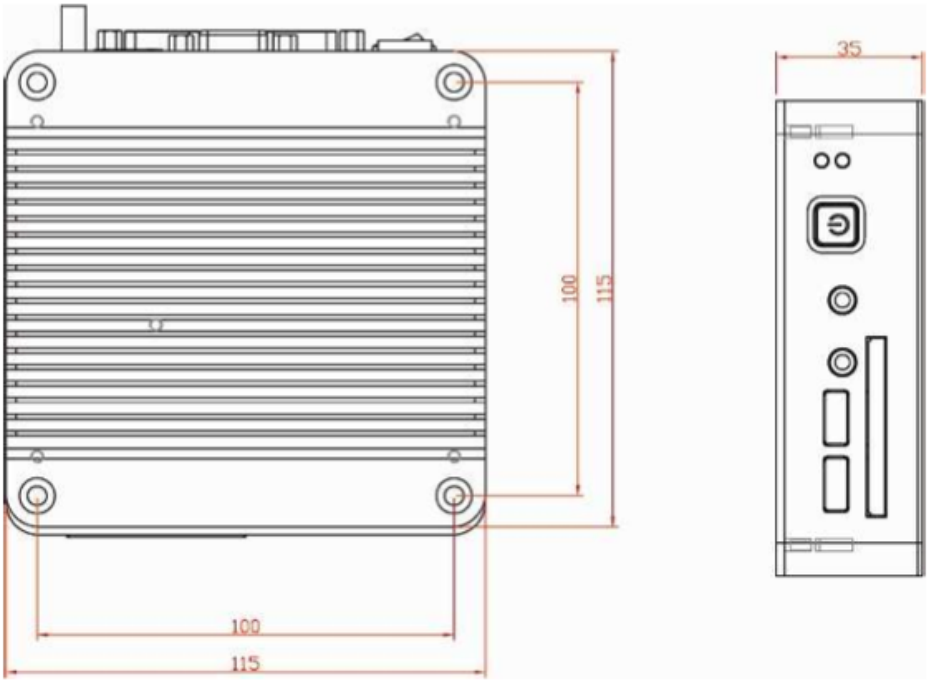

Figura 8: Esquemático do eBox 3310A
Fonte: Phung (2010)

2.4.2 BIOS e opções de *Boot*

O eBox 3310A possui uma BIOS AMI, que pode ser acessada pressionando-se a tecla `Enter` durante o procedimento de *boot*. Dentro da BIOS pode-se configurar informações básicas tais como a data e hora padrão do dispositivo, como também informações avançadas incluindo opções de dispositivos, processador e prioridade de *boot* (Figura 9) (PHUNG, 2010).

Segundo Phung (2010), há diversas opções de inicialização para o eBox 3310A. A lista a seguir representa as principais delas:

- Armazenamento IDE interno.
- Cartão Compact Flash.
- Cartão Micro SD.
- Floppy via USB.
- Armazenamento flash.
- Drive de CDROM via USB.

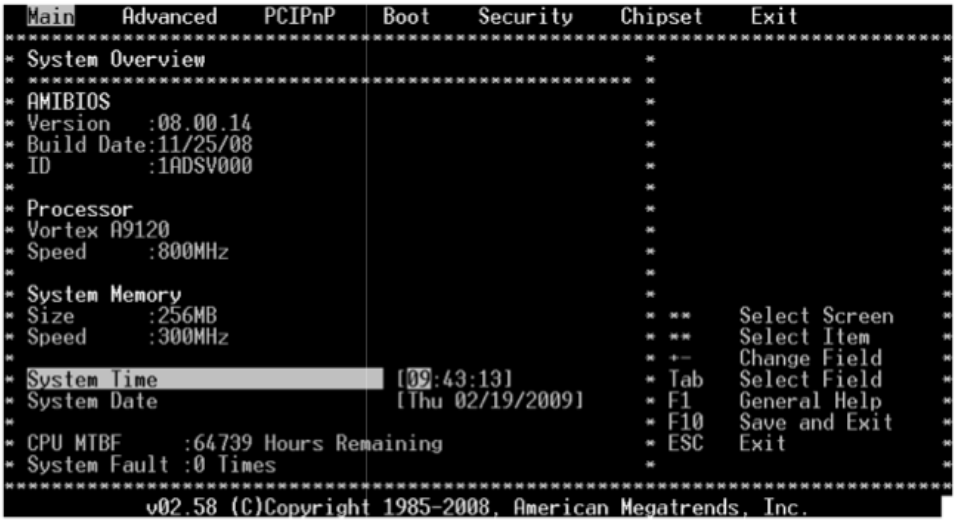

Figura 9: Captura de tela da BIOS do eBox 3310A
Fonte: Phung (2010)

- Via rede através de PXE .
- Via rede através de uma estação de desenvolvimento executando o Visual Studio e o Platform Builder.

Por padrão, o *boot* ocorre pelo dispositivo de armazenamento conectado na IDE.

2.5 Módulo Wi-Fi™ USB Wyse VT6656

O Wyse VT6656 (Figura 10) é um módulo Wi-Fi™ USB fabricado pela Wyse Technology. O fabricante, líder mundial em *Cloud Client Computing*, foi recentemente adquirido pela Dell (DELL, 2012) para estender os produtos de virtualização da companhia.

Figura 10: Módulo Wi-Fi™ USB Wyse VT6656
Fonte: Amazon (2012)

O módulo Wi-Fi™ via USB é utilizado para adicionar suporte a conexões de rede sem-fio no SMGR, já que o eBox 3310A não inclui essa opção por padrão, apenas através de

cabo Ethernet padrão RJ45. O SMGR possui uma arquitetura cliente/servidor em que cada cliente envia suas informações a um servidor central, obrigatoriamente através de uma rede sem fio. Já que cada cliente está instalado em uma máquina espalhada pelo canteiro de obras, que pode se estender por muitos quilômetros, a utilização de cabos fica totalmente descartada. Em ambientes desse porte, é possível a instalação de uma rede Wi-Fi™ de padrão industrial, com o alcance necessário para que os clientes não fiquem em zonas de sombra. É preferível utilizar Wi-Fi™ ao invés de outras tecnologias de conexão sem-fio, como Bluetooth ou ZigBee. O Bluetooth é uma tecnologia de comunicações de curto alcance (BLUETOOTH CONTRIBUTORS GROUP, 2010), portanto não atende aos requisitos do SMGR. Da mesma forma, o ZigBee, baseado no padrão 802.15.4 (IEEE COMPUTER SOCIETY, 2009), também é considerado uma tecnologia para pequenas distâncias, geralmente utilizado em eletrônicos de consumo, como controles remotos para aparelhos de TV. Além disso, a utilização de Wi-Fi™ permite criar serviços baseados no protocolo TCP/IP, que inclui o *Web Service* utilizado no desenvolvimento do SMGR.

O Wyse VT6656 é compatível com as tecnologias Wi-Fi™ A/B/G, oferecendo conexão confiável e de alta velocidade, chegando em até 54 Mbps (VIA NETWORKING TECHNOLOGIES, 2012). Além disso, suporta diversas opções de autenticação e criptografia, como por exemplo as chaves WPA/WPA2 de 128 bits. Seu formato é compacto, com apenas 8.9 x 5 x 1.2 centímetros, pesando 17 gramas.

O chip central do dispositivo é o VT6656, fabricado pela VIA Networking Technologies. Trata-se de uma solução baseada em USB 2.0, compatível também com USB 1.1. Por ser compatível com os padrões 802.11a/b/g, é bastante flexível e garante que a maioria das infraestruturas de rede sejam compatíveis com seu funcionamento. O VT6656 é certificado pela Wi-Fi™ Alliance, a organização global que fiscaliza dispositivos de comunicação Wi-Fi™. O baixo consumo de energia do VT6656 é atingido devido ao *Power Amplifier* (PA) and *Detector*, que monitora e otimiza a utilização de energia de forma a garantir uma transmissão eficiente. Tal característica é muito benéfica em sistemas que possuem requisitos de energia críticos, como é o caso do SMGR. Além disso, o VT6656 opera na faixa de 2.4Ghz, que minimiza interferências com outros dispositivos de comunicação sem fio (VIA NETWORKING TECHNOLOGIES, 2012). A seguir encontra-se uma lista com as principais características do VT6656:

- Compatível com os padrões 802.11a/b/g.
- Interface USB 1.1 ou 2.0.

- Tamanho reduzido.
- Baixo consumo de energia.
- Taxas de transmissão configuráveis: 6, 9, 12, 18, 24, 36, 48, 54 Mbps.
- Suporta os modos Infrastructure e Adhoc.
- Protocolos de segurança WPA, WPA 2.0, WEP, TKIP e AES.

O Wyse VT6656 foi adquirido pela equipe usando recursos próprios. A compra foi feita pelo site de compras coletivas eBay, já que se trata de um produto que não é mais vendido pelo fabricante. Há poucos modelos no mercado que oferecem compatibilidade com o Windows Embedded através de interface USB, pois geralmente os fabricantes de kits para desenvolvimento de sistemas embarcados já incluem módulos Wi-Fi™ integrados. Dentre as poucas opções disponíveis, o Wyse VT6656 ofereceu o menor custo e maior facilidade de acesso.

2.6 Módulo GPS

Nessa seção será introduzido o sistema de GPS e seus componentes, assim como o módulo de GPS EM-411 e suas características técnicas.

2.6.1 Visão geral do sistema de GPS

O Sistema de Posicionamento Global (Global Positioning System - GPS) foi criado pelo Departamento de Defesa (*Department of Defense* - DoD) do governo dos Estados Unidos nos anos 70. Inicialmente foi desenvolvido para atender as necessidades militares do país, porém, atualmente, está disponível para uso civil. O GPS provê informações contínuas de localização e temporização, sob quaisquer condições climáticas. Por servir uma quantidade ilimitada de usuários e por questões de segurança, o GPS é um sistema passivo, ou seja, é somente possível fazer a leitura de seu sinal (EL-RABBANY, 2002).

A ideia central ao GPS é muito simples. Supondo que as distâncias de um ponto na superfície terrestre a três satélites em órbita são conhecidas, então a localização do ponto pode ser determinada. Para encontrar essa distância, uma arquitetura global foi projetada. Teoricamente, apenas três satélites são necessários para a tarefa (Figura 11), porém, na prática, um quarto satélite torna-se necessário, pois há uma diferença entre a

passagem de tempo na superfície e na órbita do planeta. Atualmente, a posição pode ser determinada com 95% de precisão, um erro intencional é inserido na coordenada entregue pelos satélites para evitar problemas de segurança. Esse nível de operação é chamado Serviço de Posicionamento Padrão (*Standard Positioning System* - SPS). Usuários com acesso ao nível de Serviço de Posicionamento Preciso (*Precise Positioning System* - PPS), como os militares, possuem uma precisão muito maior (EL-RABBANY, 2002).

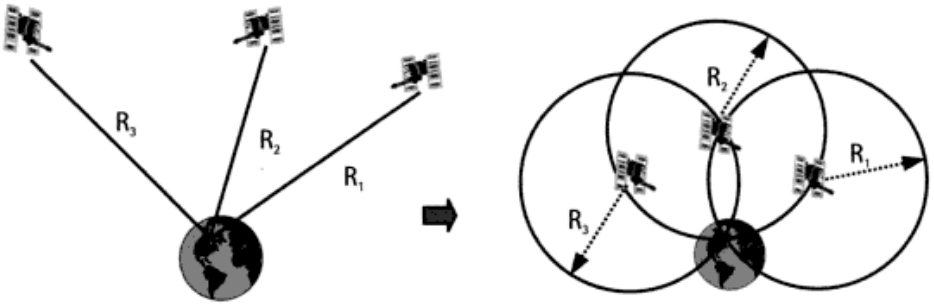

Figura 11: Funcionamento básico do GPS utilizando três satélites
Fonte: El-Rabbany (2002)

O GPS é composto por uma constelação de 24 satélites operacionais. Essa constelação, conhecida como Capacidade Inicial de Operação (Initial Operacional Capability - IOC), foi completada em Julho de 1993. Para garantir uma cobertura verdadeiramente global, os satélites que compõem o GPS são posicionados em 6 planos orbitais, com 4 satélites em cada um. Devido a essa geometria, 4 a 10 satélites estarão sempre visíveis em qualquer local do globo. As órbitas dos satélites são praticamente circulares, com uma excentricidade elíptica em torno de 0.01. Ficam localizados a uma altitude de aproximadamente 20,200 km da superfície terrestre, com um período orbital de 11 horas e 58 minutos. Em Julho de 1995 o GPS atingiu sua Completa Capacidade Operacional (*Full Operational Capacity* - FOC) e, desde então, o número de satélites na constelação é superior aos 24 iniciais (EL-RABBANY, 2002).

O GPS é composto por três segmentos: espacial, de controle e de usuário (Figura 13). O segmento espacial consiste na constelação de satélites em órbita. Cada satélite transmite um sinal com vários componentes: duas ondas senóides (conhecidas como frequências portadoras - *carrier frequencies*), dois códigos digitais e uma mensagem de navegação. As portadoras e os códigos são utilizados, principalmente, para determinar a distância do usuário aos satélites (EL-RABBANY, 2002).

A mensagem de navegação contém, além de outras informações, as coordenadas dos satélites em função do tempo. Os sinais transmitidos são controlados por um preciso

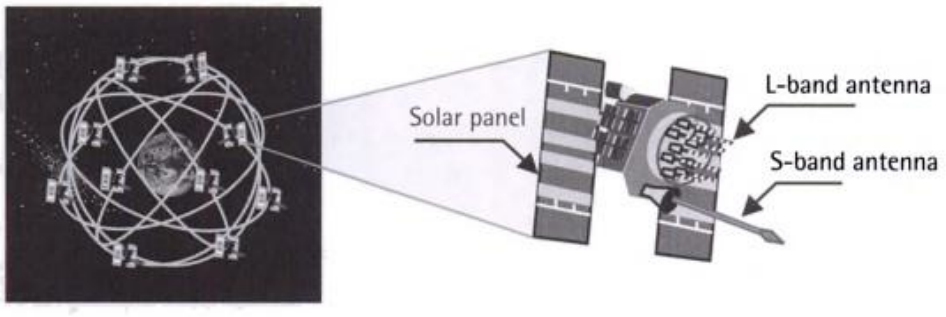

Figura 12: Planos orbitais da constelação de GPS
Fonte: El-Rabbany (2002)

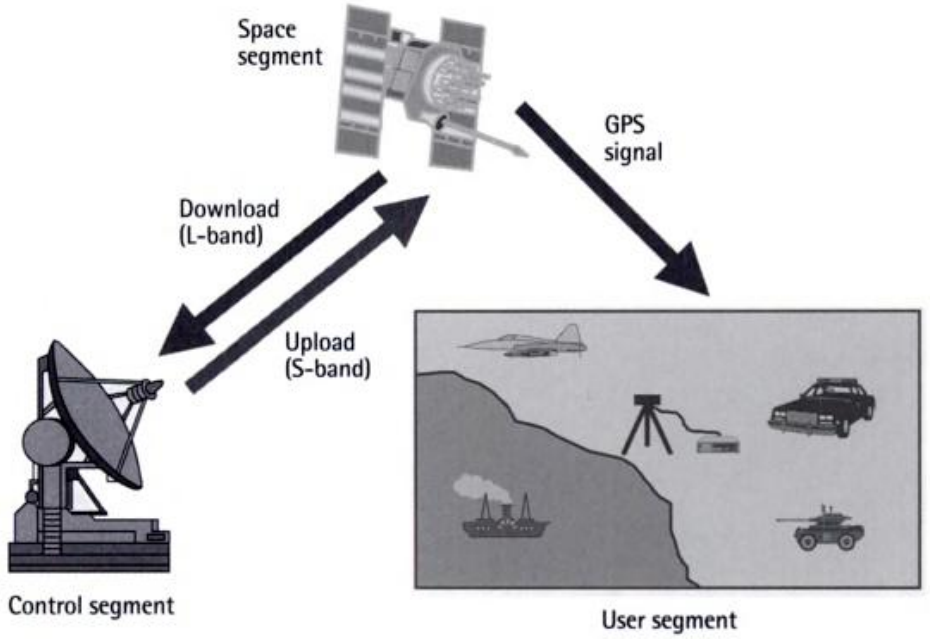

Figura 13: Segmentos do GPS
Fonte: El-Rabbany (2002)

relógio atômico situado em cada satélite. O segmento de controle consiste em uma rede global de estações de rastreamento (*tracking*), com uma estação central (*Master Control Station* - MCS) em Colorado Springs, Colorado, nos Estados Unidos. A tarefa principal do segmento de controle é rastrear os satélites para determinar posições, integridade do sistema, comportamento dos relógios atômicos, dados atmosféricos, etc. Essas informações são empacotadas e transmitidas a cada satélite através da banda-S, que apenas o segmento de controle tem acesso. Vale ressaltar que usuários do GPS não podem enviar sinais, apenas recebê-los. Outras estações do segmento de controle encontram-se no Havaí, Kwajalein (ilha a nordeste da Austrália), Diego Garcia (ilha no centro do Oceano Índico) e Ilha de Ascensão (ilha sob domínio britânico no centro do Oceano Atlântico, entre a América do Sul e a África) (EL-RABBANY, 2002).

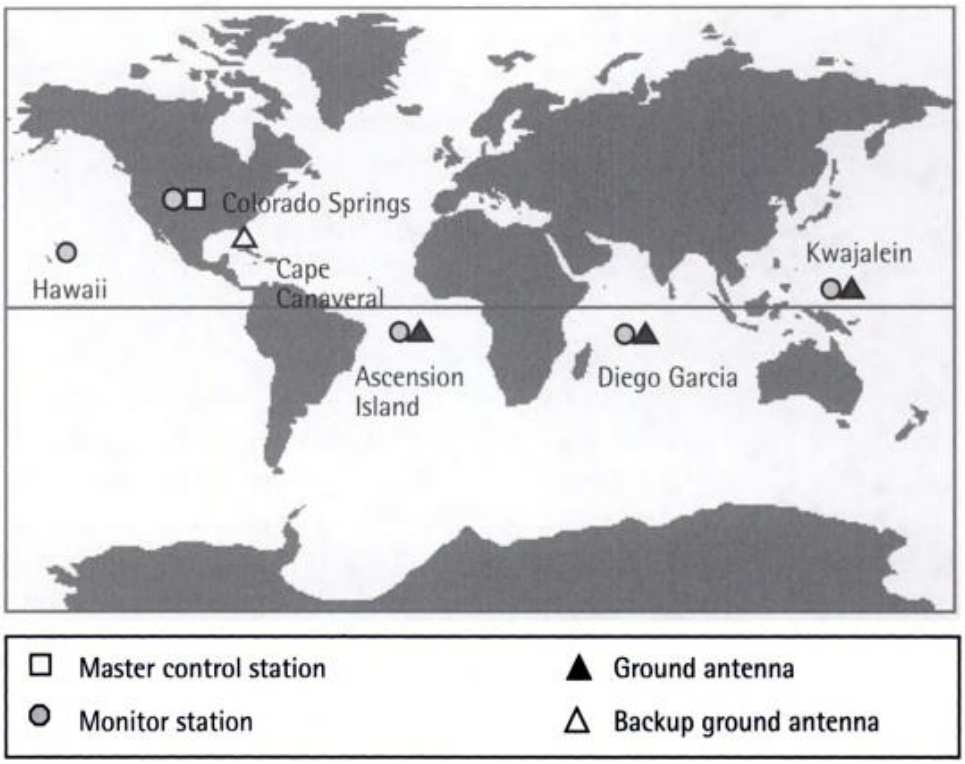

Figura 14: Estações do segmento de controle do GPS
Fonte: El-Rabbany (2002)

Por fim, o segmento de usuário consiste todos os usuários, tanto civis como militares. Um usuário equipado com um receptor de GPS pode saber sua posição em qualquer lugar do globo, sem qualquer custo direto (EL-RABBANY, 2002).

2.6.2 Módulo de GPS EM-411

O EM-411 é um módulo GPS com antena interna desenvolvido para projetos pequenos (GLOBALSAT TECHNOLOGY CORPORATION, 2009). Possui baixo consumo de energia e é ideal para sistemas de navegação, medidas de distâncias, monitoramento de veículos, etc. É de fácil instalação em qualquer superfície e possui tamanho compacto. Todas os conectores presentes operam em nível TTL, facilitando o interfaceamento com microcontroladores e outros dispositivos. Através de sua interface serial são enviadas uma série de mensagens que seguem o padrão NMEA 0183 (NATIONAL MARINE ELECTRONICS ASSOCIATION, 2012), as quais incluem dados de posição, informações sobre satélites, tempo, entre outros. O padrão NMEA 0183 define requisitos elétricos para os sinais de módulos GPS, o protocolo de transmissão de dados, e especifica o formato das mensagens ASCII em uma comunicação serial com *baud-rate* igual a 4800. Cada canal de comunicação pode possuir múltiplos ouvintes, porém apenas um transmissor.

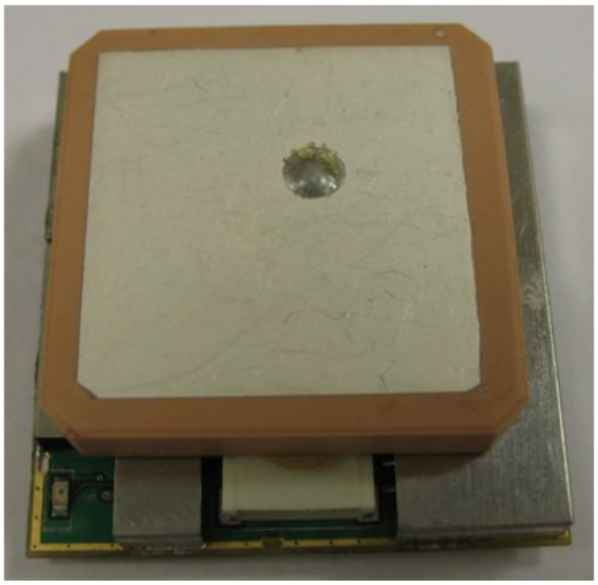

Figura 15: Módulo GPS EM-411
Fonte: (GLOBALSAT TECHNOLOGY CORPORATION, 2009)

Em seu centro, o EM-411 possui um microcontrolador de GPS de alta sensitividade, conhecido como SiRF Start III (GLOBALSAT TECHNOLOGY CORPORATION, 2009). Esse componente é responsável por interpretar o sinal de GPS recebido dos satélites e enviar as informações ao usuário através da porta serial seguindo o protocolo NMEA 0183.

A lista a seguir resume as principais características técnicas do EM-411:

- Alimentação de 5V e 60mA.
- Antena interna.
- Tempo de aquisição de sinal de 42 segundos.
- Saídas em nível TTL.
- Tamanho compacto: 30x30x10 milímetros.
- Suporte ao padrão NMEA 0183.
- Chipset SiRF Star III de alta sensitividade.
- Fácil integração com microcontroladores.

Ao lado do módulo EM-411 há um LED que indica o estado de sua operação. Com o LED desligado, o módulo está sem energia; LED ligado sinaliza que o módulo está

buscando um sinal; LED piscando sinaliza que os satélites foram localizados e a posição está sendo obtida normalmente (GLOBALSAT TECHNOLOGY CORPORATION, 2009).

O EM-411 possui uma alta precisão, em torno de 10 metros. Consegue medir a velocidade de movimentação, em um mínimo de 0,1 m/s e máximo de 515 m/s, além da altura em relação ao nível do mar, em um máximo de 18 km. Opera em temperaturas de -40 a 85 graus Celsius (GLOBALSAT TECHNOLOGY CORPORATION, 2009).

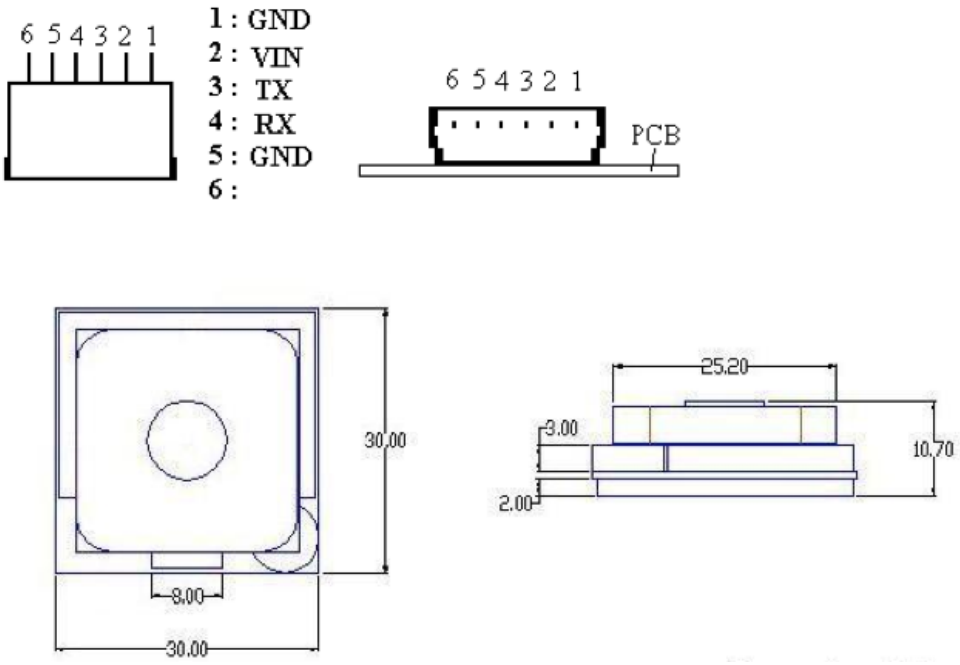

Figura 16: Esquemático do EM-411
Fonte: GlobalSat Technology Corporation (2009)

Há 4 pinos principais no EM-411:

- VIN: entrada de energia DC de 4.5 a 6.5 V.
- TX: pino de transmissão via interface serial.
- RX: pino de recebimento de dados, para configuração do módulo caso alguma opção especial seja necessária.
- GND.

Através do padrão NMEA 0183, diversas mensagens são enviadas através do pino TX. Em especial, é interessante observar a mensagem que inicia com o prefixo `GPGGA`. Essa mensagem traz consigo dados de localização GPS, como no exemplo do Código 2.1.

Código 2.1: Mensagem GPS

```
$GPGGA,161229.487,3723.2475,N,12158.3416,
                                    W,1,07,1.0,9.0,M,,,,0000*18
```

Tabela 1: Explicação de mensagem GPGGA

Nome	**Exemplo**	**Unidades**	**Descrição**
MessageID	$GPGGA		Cabeçalho do protocolo GGA
UTC Time	161229.487		hhmmss.sss
Latitude	3723.2475		ddmm.mmmm
N/S Indicator	N		N=norte ou S=sul
Longitude	12158.3416		ddmm.mmmm
E/W Indicator	W		E=leste ou W=oeste
Position Fix Indicator	1		
Satellites Used	07		De 0 a 12
HDOP	1.0		Horizontal Dilution of Precision
MSL Altitude	9.0	metros	
Units	M	metros	
Geoid Separation		metros	
Units	M	metros	
Age of Diff. Corr.		metros	Nulo quando DPGS não é usado
Diff. Ref. Station ID	0000	segundos	
Checksum	*18		
<CR><LF>			Fim da mensagem

Fonte: Adaptado de GlobalSat Technology Corporation (2009)

2.7 Módulo de Aquisição de Sinais

O SMGR possui como um de seus requisitos a capacidade de monitorar o maquinário em grandes obras. Para que tal monitoramento seja possível, é necessária a instalação de sensores em cada um dos equipamentos, capazes de obter informações sobre grandezas diversas. Uma vez instalados os sensores, o *software* do SMGR executando no eBox 3310A sobre o Windows Embedded CE 6.0 fará a leitura dos valores e os enviará ao servidor central, através do *Web Service* projetado. Basicamente, é possível utilizar sensores que coletam dados de grandezas analógicas e digitais.

Grandezas analógicas podem assumir valores contínuos em um determinado intervalo, enquanto digitais assumem apenas dois estados possíveis (aberto / fechado, ligado / desligado, aceso / apagado, entre outros). Os valores analógicos geralmente representam grandezas físicas, como a quantidade de combustível em um tanque, e isso geralmente traduz-se em uma variação de tensão, que posteriormente passa por um conversor analógico/digital para possuir uma representação numérica utilizável pelo computador. Sua precisão depende da resolução do conversor, que, em modelos comerciais comumente encontrados, variam de 10 a 12 bits.

Para ler dados de sensores através de um sistema computacional pode-se utilizar um microcontrolador. Trata-se de um componente que incorpora em um único *chip* uma série de componentes, como CPU e memória RAM (KAMAL, 2007). É muito comum confundir microcontrolador e microprocessador, porém há uma série de diferenças que valem ser ressaltadas. Um microcontrolador (Figura 17) incorpora um microprocessador (CPU) em sua composição, pois requer toda a capacidade de executar e processar instruções segundo um *clock* bem definido. Possui RAM, ROM, dispositivos de E/S (entrada e saída), *timers* e contadores, ao contrário do microprocessador, que requer a instalação individual de cada um desses itens. Além disso, um microcontrolador gasta menos tempo para a E/S de dados, já que todos os componentes de memória estão localizados em um mesmo *chip*. Requer menos *hardware* para um funcionamento completo, porém é menos flexível, pois executa código específico para sua arquitetura (GODSE; GODSE, 2008).

Específico ao projeto do SMGR, foi utilizado o ATmega 328P, um microcontrolador de 8 bits produzido pela Atmel. Possui baixo consumo de energia, uma arquitetura de instruções RISC, *clock* de 16 MHz, tensão de operação de 5V, 14 pinos de entrada digitais, 6 pinos de entrada analógicos, 32 kB de memória flash, 2 kB de SRAM e 1 kB de memória EEPROM. Além disso, inclui UART (*Universal Asynchronous Receiver/Transmitter*) para comunicação serial. Trata-se de um chip com 28 pinos, no formato PDIP (Plastic Dual Inline Package), operando em temperaturas de -40 a 85 graus Celsius (ATMEGA, 2012).

Cada canal analógico do ATmega 328P possui uma resolução de 10 bits, ou seja, é possível haver 1024 valores possíveis para se especificar um intervalo contínuo de alguma grandeza física (ATMEGA, 2012).

O ATmega 328P está disponível com o Arduino Uno (ARDUINO, 2012), uma placa de desenvolvimento para fins pessoais ou comerciais (CREATIVE COMMONS, 2012). O Arduino Uno disponibiliza um pacote para facilitar o uso e programação do ATmega

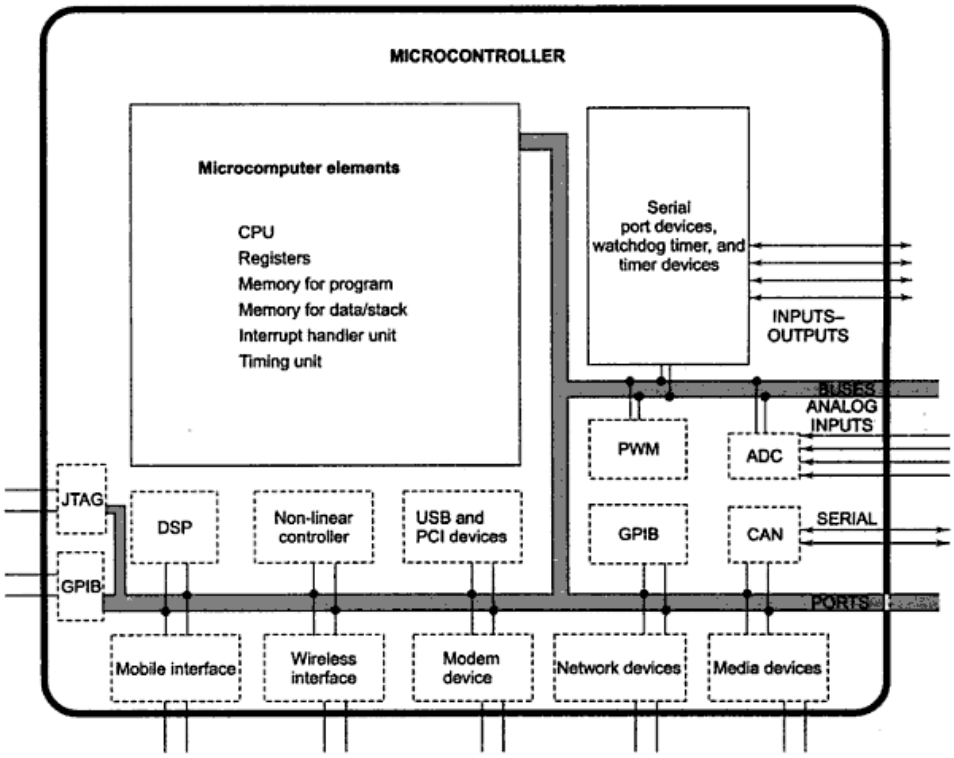

Figura 17: Elementos de um Microcontrolador
Fonte: Kamal (2007)

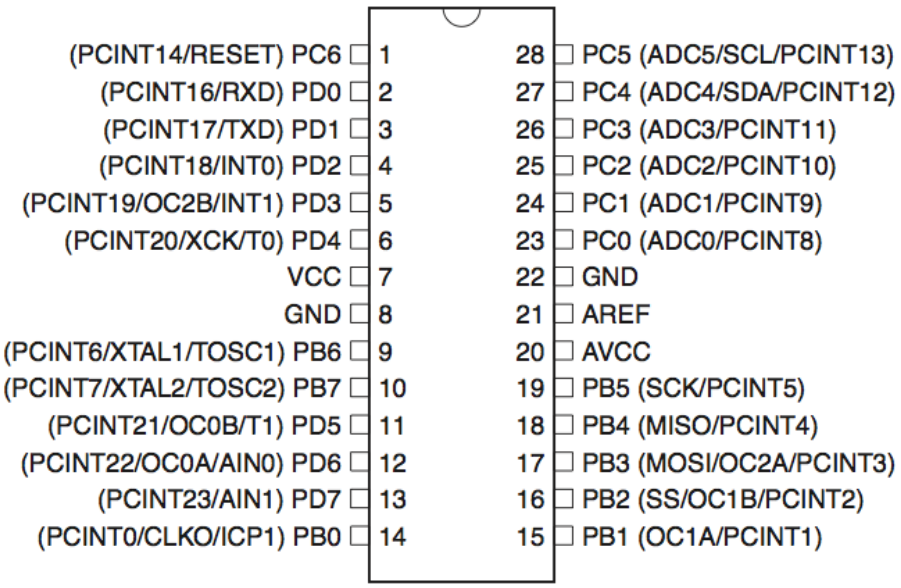

Figura 18: Pacote PDIP do ATmega 328P
Fonte: ATmega (2012)

328P, com alimentação através de porta USB e programação através de ambiente de desenvolvimento de código-livre. Além disso, o Arduino fornece uma barra de pinos para facilitar a interface com os sensores analógicos e digitais que serão necessários ao projeto.

Ainda mais, o Arduino possui 2 pinos para transmissão e recepção de dados via serial (UART), tornando-o uma alternativa que atende a requisitos técnicos de disponibilidade e custo para o projeto do SMGR. O Arduino foi adquirido pela equipe com capital próprio, através do site de compras eBay.

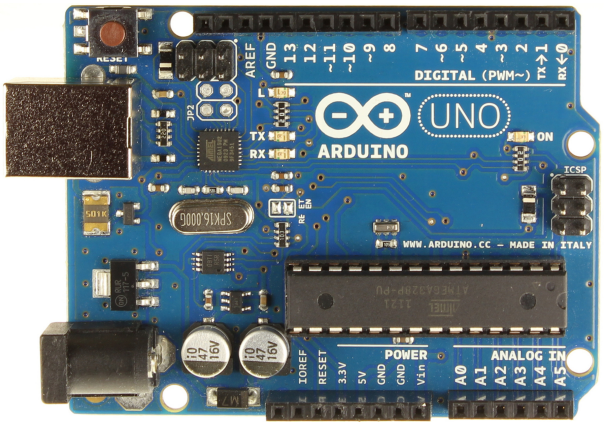

Figura 19: Arduino
Fonte: Arduino (2012)

2.8 *Web Service*

Segundo Berners-Lee (2009), um *Web Service* é um sistema de *software* destinado a suportar interação entre *softwares* distintos sobre uma rede. Sua interface é descrita por uma linguagem aberta baseada em XML (BRAY et al., 2008), a WSDL - *Web Service Description Language*. Uma descrição WSDL (CHRISTENSEN et al., 2000) de um web service é a responsável pela demonstração das operações e métodos disponíveis, como e onde acessá-los, em uma linguagem reconhecível pelo computador. As mensagens entre clientes e provedor de serviços é feita através do protocolo SOAP - *Simple Object Access Protocol* (GUDGIN et al., 2007), um protocolo de troca de informação estruturada, também baseado em XML.

2.8.1 WSDL

WSDL é uma linguagem criada por um empreendimento conjunto de Ariba, Microsoft e IBM, pela junção das tecnologias NASSL (Network Application Service Specification Language) da IBM e SDL (Service Description Language) da Microsoft. A versão inicial do WSDL foi enviada para a W3C em 2000 (CHRISTENSEN et al., 2000) e formalizada na versão 1.1 em 2001 (CHRISTENSEN et al., 2001). Atualmente, a WSDL está na versão 2.0, e é uma recomendação da W3C (BOOTH; LIU, 2007) sempre utilizar esta versão.

Como as definições abstratas de interfaces e mensagens são separadas das instâncias e do uso real, estas definições podem ser aproveitadas em diversas partes do código, como a definição de uma classe em orientação a objetos.

A Tabela 2 apresenta os componentes de um descritor WSDL, mostrando as diferenças entre as versões 1.1 e 2.0.

Tabela 2: Componentes de um Descritor WSDL

WSDL 1.1	**WSDL 2.0**	**Descrição**
Service	Service	Contém o conjunto de funções disponiveis para os protocolos de rede.
Port	Endpoint	Define o endereço ou ponto de conexão ao *Web Service*. Geralmente é representado por uma URL HTTP.
Binding	Binding	Especifica a interface e o estilo de conexão SOAP (RPC/documento) e o transporte pelo protocolo SOAP. A seção de binding também define as operações.
PortType	Interface	Define um serviço, as operações que podem ser realizadas e as mensagens utilizadas para tal operação.
Operation	Operation	Define as ações SOAP e o metodo de codificação. Por exemplo "literal". O equivalente a um método/função em uma linguagem de programação convencional.
Message	n/a	Tipicamente, uma mensagem corresponde a uma operação. A mensagem contém a informação necessária para executar a operação, cada mensagem é feita de um ou mais objetos lógicos. Cada objeto com um atributo de tipificação. Cada mensagem deve possuir um nome único, assim como o nome de um objeto da mensagem deve ser único entre os objetos da mensagem. No WSDL 2.0 as *messages* foram removidas, pois o XML Schema é o responsável por definir o corpo de entradas, saídas e falhas de maneira mais simples e direta.
Types	Types	Descreve os dados. Para tal, utiliza a linguagem XML Schema (XSD).

Fonte: Adaptado de Booth e Liu (2007)

2.8.2 SOAP

SOAP - *Simple Object Access Protocol* é um protocolo leve cujo propósito é a troca de informação estruturada em um ambiente descentralizado e distribuído. Sua estrutura é definida em XML, o que permite a transmissão de seu *framework* em uma vasta gama de protocolos. O *framework* de SOAP foi construído com a intenção de ser independente de qualquer linguagem de programação ou de semânticas específicas de alguma linguagem (GUDGIN et al., 2007).

As principais metas do protocolo SOAP são flexibilidade e extensibilidade. Para atingir estas metas, foram excluídas do *framework* de mensagens certas características comumente presentes em sistemas distribuídos, como confiabilidade, segurança e correlação.

O protocolo SOAP possui um conjunto básico de comunicação que permite a construção de Web Services. O protocolo HTML pode ser utilizado como camada de transporte para o SOAP (GUDGIN et al., 2007).

2.9 Banco de Dados

Muitas empresas atualmente dependem de seu amplo conjunto de informações para realizar operações diárias e, com o crescimento contínuo das trocas de informação, tudo o que é inserido como dado precisa ser completo, preciso e organizado de maneira que seja facilmente acessível quando necessário e no formato requisitado.

Para armazenar estas informaçoes, alguém poderia simplesmente utilizar um sistema de arquivos comum em um computador, porém isto não atenderia os requisitos previamente citados pois:

- A estrutura dos registros seria definida pela aplicação. Para trocar a estrutura dos dados, o programa deveria ser reescrito. Além disso, dados antigos deveriam ser convertidos ao novo modelo.
- Os registros seriam projetados para uso em uma aplicação específica. Se novas aplicações tem novos modelos estes geralmente são incompatíveis e, se quiserem utilizar este recurso elas devem ser adaptadas a este modelo.
- Quanto maior a complexidade dos dados, maior a dificuldade de manutenção de aplicativos que utilizam os mesmos dados.

Para solucionar estes problemas o conceito de banco de dados surgiu, com a proposta de considerar a informação como prioridade no projeto de sistemas, colocando dados como centro de interesse em aplicações.

Banco de dados é o nome dado a uma coleção de informações agrupadas em uma estrutura organizada em um conjunto de tabelas formalmente definidas, das quais as informações podem ser facilmente recuperadas. Existem diversos modelos de banco de dados, mas o mais utilizado é o modelo Relacional (CODD, 1970).

2.9.1 Modelo relacional

O modelo relacional foi criado por Codd (1970) em 1970. Trata-se de um modelo de banco de dados baseado em lógica predicativa de primeira ordem. No modelo relacional, todos os dados são representados na forma de tuplas, agrupados em relações.

O propósito do modelo relacional é providenciar uma maneira de representar dados e buscas de maneira declarativa, na qual o usuário determina qual informação o banco contém e o que ele quer extraír do banco. A partir daí, o sistema de gerenciamento do banco de dados cuida da maneira como armazenar a informação e como recuperá-la posteriormente (CODD, 1970).

A maioria dos sistemas de gerenciamento de banco de dados relacionais atualmente utilizam o modelo de definição de dados e buscas SQL - *Structured Query Language* - no português: linguagem de pesquisas estruturadas, em tradução livre. SQL é uma linguagem criada para manipular dados no modelo relacional. Apesar de inicialmente ter sido definida como linguagem padrão do modelo relacional, ela não adere completamente aos conceitos propostos por Codd (1970).

O tipo de operação em SQL mais comum é o de projeção, na qual se utiliza a sentença `SELECT` para extrair informação das tabelas disponíveis do banco. Um exemplo de projeção feita em SQL está ilustrado no Código 2.2. Esta *query* retorna a cor da casa e nome do proprietário que possua idade igual a 35 anos e a casa seja localizada na Rua dos Mafagafos.

Código 2.2: Exemplo de projeção em SQL

```
SELECT Casa.Cor, Proprietario.Nome
FROM Casa INNER JOIN Proprietario ON
                        Casa.ProprietarioID = Proprietario.ID
WHERE Proprietario.Idade = 35 AND Casa.Rua = 'Rua dos Mafagafos'
```

Existem diversas maneiras para integrar consultas feitas em SQL a sistemas. No .NET Framework 3.5, foi introduzida uma ferramenta de linguagem chamada LINQ, que permite construir sentenças SQL diretamente na linguagem de programação escolhida.

2.10 .NET Framework

.NET Framework é o nome do conjunto de ferramentas de programação disponibilizado pela Microsoft e feito para rodar primariamente em Windows (MICROSOFT, 2012d). Dentre suas principais vantagens está a interoperabilidade entre linguagens do pacote e uma vasta biblioteca de estruturas.

Programas feitos em .NET Framework rodam sobre uma máquina virtual chamada *Common Language Runtime* - CLR. Esta provê ao serviço ferramentas como Gerenciamento de memória, tratamento de exceções e camadas de segurança.

O desenvolvimento do .NET Framework começou no início dos anos 1990 sob o codinome Next Generation Windows Services (NGWS) - Serviços Windows da Futura Geração, em tradução livre. Sua primeira versão comercial foi distribuída no ano 2000 já com o nome .NET Framework 1.0. A partir da versão do Windows Server 2008 o .NET Framework passou a ser distribuido juntamente com as distribuições do Windows (MICROSOFT, 2012d).

2.10.1 Principais características do .NET Framework

2.10.1.1 Interoperabilidade

Uma necessidade comum entre *softwares* é a interação entre aplicações novas com aplicações mais antigas. o .NET Framework provê meios para que programas acessem funcionalidades implementadas em programas executados fora do ambiente a partir da interface disponível em `System.Runtime.InteropService` ou usando o recurso P/Invoke (MICROSOFT, 2003).

2.10.1.2 CLR

O CLR funciona como motor de funcionamento do .NET Framework. Como todas as aplicações .NET rodam sobre o motor algumas características são garantidas entre todas as aplicações desenvolvidas sobre o pacote (MICROSOFT, 2012a).

2.10.1.3 Independência de linguagem

O .NET introduziu consigo o *Common Type System* - CTS, que define todas as estruturas e tipos de dado suportados pelo CLR e determina como eles podem ou não interagir. Isto permite a troca de informação entre bibliotecas e programas construídos em linguagens diferentes suportadas pelo .NET Framework, dentre elas C#, Visual Basic e Visual C++ (MICROSOFT, 2012b).

2.10.1.4 Portabilidade

Apesar da Microsoft não ter implementado o .NET Framework integralmente em nenhuma plataforma além do Microsoft Windows, este foi projetado para ser independente de plataforma. Existe o .NET Compact Framework para ambientes como o Windows Embedded, e o projeto independente Mono, que disponibiliza o .NET Framework para outras plataformas, como Linux, iOS e Android (MONO, 2012).

2.11 CLI - Common Language Infrastructure

A CLI é uma especificação aberta criada pela Microsoft e certificada pela ISO e ECMA (ECMA INTERNATIONAL, 2012) que descreve o código executável e ambiente de execução do núcleo principal do .NET Framework e também das implementações livres Mono (MONO, 2012) e Portable.NET (DOTGNU PORTABLE.NET, 2012). Esta define um ambiente que permite o uso de diversas linguagens de alto nivel em diferentes plataformas computacionais sem haver necessidade de reescrever código para arquiteturas específicas.

Dentre outras características, a CLI possui quatro aspectos fundamentais:

- *Common Types System* (CTS): conjunto de tipos de dados e operações que todas as linguagens compativeis com o CTS devem possuir.
- Metadados: informações sobre a estrutura do programa sao independentes da linguagem, de maneira que possam ser referenciados em todas as linguagens disponiveis, em vista de facilitar o trabalho com código feito em uma linguagem desconhecida ao programador.
- *Common Language Specification* (CLS): um conjunto de regras de base que toda linguagem que busca conformidade com a CLI deve seguir para que possa interoperar com as outras linguagens da CLI. A CLS é um subconjunto da CTS.

- *Virtual Execution System* (VES): O VES carrega e executa programas compatíveis com a CLI, utilizando os metadados para interligar trechos de código gerados separadamente em tempo de execução.

Todas as linguagens são compiladas em um uma linguagem intermediária comum, ou CIL - *Common Intermediate Language*, ainda abstrata ao modulo de *hardware*. Quando o código é executado, a VES específica do *hardware* traduz o código gerado em CIL para o *hardware* e sistema operacional específico. Uma visão geral do CLI pode ser encontrada na Figura 20.

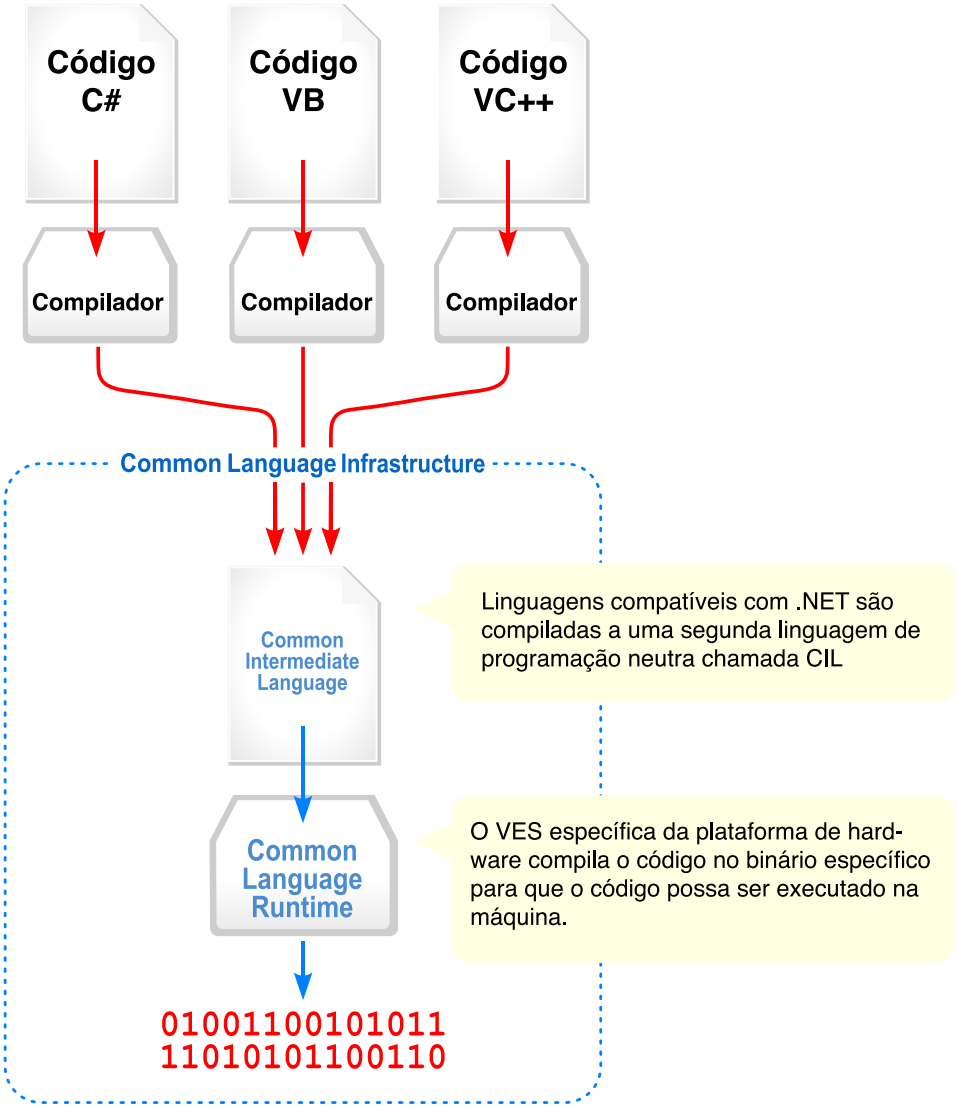

Figura 20: Visão Geral da CLI

2.12 Considerações sobre o capítulo

Este capítulo apresentou os fundamentos teóricos dos diversos conceitos envolvidos no projeto do SMGR. Iniciou-se tratando de sistemas embarcados, tema fundamental para a implementação da estrutura de *hardware* e *software* que faz a coleta de dados. O tema de sistemas operacionais em tempo real (RTOS) serve como base para a importante questão da plataforma que fornece suporte à programação do *software* embarcado executando sobre o *hardware* dedicado. A seguir, o item que aborda o Windows Embedded descreve o RTOS da Microsoft, que foi utilizado como solução específica ao projeto do SMGR. A seguir, o item eBox 3310A abordou as características do kit de desenvolvimento utilizado. Além disso, foram descritos os módulos Wi-Fi™(Wyse VT6656), GPS (EM-411) e de aquisição de sinais (ATmega 328P, disponível no Arduino). Os itens seguintes, web service e banco de dados, fornecem base teórica para o desenvolvimento da aplicação web que é responsável pelo *link* entre clientes e o servidor, além da interface de visualização dos dados. Por fim, o tópico .NET Framework abordou o *framework* de programação que foi utilizado em todos os módulos do sistema.

3 ESPECIFICAÇÃO

Neste capítulo serão apresentados a análise de requisitos do SMGR, bem como as especificações de arquitetura, *hardware* e *software*.

3.1 Especificação do Sistema

3.1.1 Análise de Requisitos

Segundo Sommerville (2007), possivelmente o maior problema a ser enfrentado no desenvolvimento de sistemas de *software* é o da engenharia de requisitos. Ela está ligada com a definição do que o sistema deverá fazer, suas propriedades essenciais e desejáveis e as restrições de desenvolvimento e operação do sistema.

Sommerville (2007) afirma ainda que os requisitos de um sistema são exigências sobre os serviços que devem ser oferecidos pelo sistema e suas restrições de operação. Esta seção descreve os requisitos do SMGR que foram levantados durante a fase de estudo e planejamento.

3.1.1.1 Requisitos Funcionais

Os requisitos funcionais de um sistema são declarações de serviços que o sistema deverá prover e como deverá se comportar em determinadas situações, ou seja, detalham o que o sistema deverá fazer (SOMMERVILLE, 2007). Abaixo são apresentados os requisitos funcionais do SMGR:

RF001: O sistema deve monitorar máquinas em canteiros de obras de maneira automática.

RF002: O sistema deve permitir a entrada manual de recursos que não são monitorados automaticamente.

RF003: O sistema deve permitir a atualização manual de informações.

RF004: O sistema deve gerenciar múltiplos módulos embarcados.

RF005: O sistema deve permitir o controle de acesso.

RF006: O sistema deve permitir a visualização do posicionamento das máquinas monitoradas em um mapa.

3.1.1.2 Requisitos Não Funcionais

Os requisitos não funcionais de um sistema são restrições sobre os serviços e funções oferecidos pelo sistema (SOMMERVILLE, 2007). Abaixo são apresentados os requisitos não funcionais do SMGR:

RN001: O sistema deve possuir um módulo cliente para cada máquina do canteiro de obras.

RN002: O módulo cliente deve ser alimentado por uma fonte de energia de 5V.

RN003: O módulo cliente deve ser provido de um Sistema Operacional Windows CE.

RN004: O módulo cliente deve anexar um aquisitor de sinais analógicos e digitais.

RN005: O módulo cliente deve ser capaz de receber dados de sensores instalados na máquina.

RN006: O módulo cliente deve possuir um conjunto de sensores adaptável para cada tipo de máquina.

RN007: O módulo cliente deve possuir uma interface para coleta de posicionamento.

RN008: O módulo cliente deve se comunicar com o servidor através de uma interface Wi-Fi™industrial.

RN009: O servidor deve manter as informações em um banco de dados Microsoft SQL Server Express.

RN010: O servidor deve prover um web service para uso dos módulos embarcados.

RN011: O servidor deve possuir um servidor HTTP para exibir as informações através de um aplicativo web.

RN012: O servidor HTTP deve ser capaz de receber conexões vindas da Internet.

RN013: O servidor HTTP deve obter informações a partir do web service.

RN014: O aplicativo web deve ser acessível a partir de computadores pessoais

RN015: O aplicativo web deve ser acessível a partir de plataformas móveis

3.1.2 Diagrama de Casos de Uso

Diagramas de caso de uso são diagramas utilizados para descrever um conjunto de ações, ou casos de uso, que algum sistema deve ou pode realizar de forma colaborativa com uma ou mais entidades externas ao sistema, os atores (FAKHROUTDINOV, 2012). A Figura 21 apresenta o diagrama de casos de uso para o sistema SMGR.

O caso de uso `View Data` permite que atores do tipo `User` possam visualizar as informações que foram coletadas pelos módulos embarcados ou cadastradas manualmente por um ator do tipo `Administrator`.

O caso de uso `Manage Users` permite que atores do tipo `Administrator` possam visualizar, adicionar e excluir usuários do sistema.

O caso de uso `Set Device Parameters` permite que atores do tipo `Administrator` possam editar as configurações dos módulos embarcados, como intervalo de tempo entre coleta de dados e nome do dispositivo.

O caso de uso `Insert Data` permite que atores do tipo `Administrator` possam inserir manualmente informações de dispositivos não gerenciados, isto é, que não possuem módulo embarcado instalado.

O caso de uso `Manage Devices` permite que atores do tipo `Adminstrator` possam visualizar e editar os dispositivos cadastrados.

O caso de uso `Get Attributes` permite que atores do tipo `Embedded Module` possam solicitar ao ator `Database` as configurações do módulo embarcado.

O caso de uso `Insert GPS Measurements` permite que atores do tipo `Embedded Module` possam enviar dados de posicionamento GPS ao ator `Database`.

O caso de uso `Insert Sensors Measurements` permite que atores do tipo `Embedded Module` possam enviar dados coletados dos sensores analógicos e digitais ao ator `Database`.

3.2 Especificação do Módulo Embarcado

Esta seção apresentará as especificações de *software* e *hardware* do módulo embarcado do SMGR.

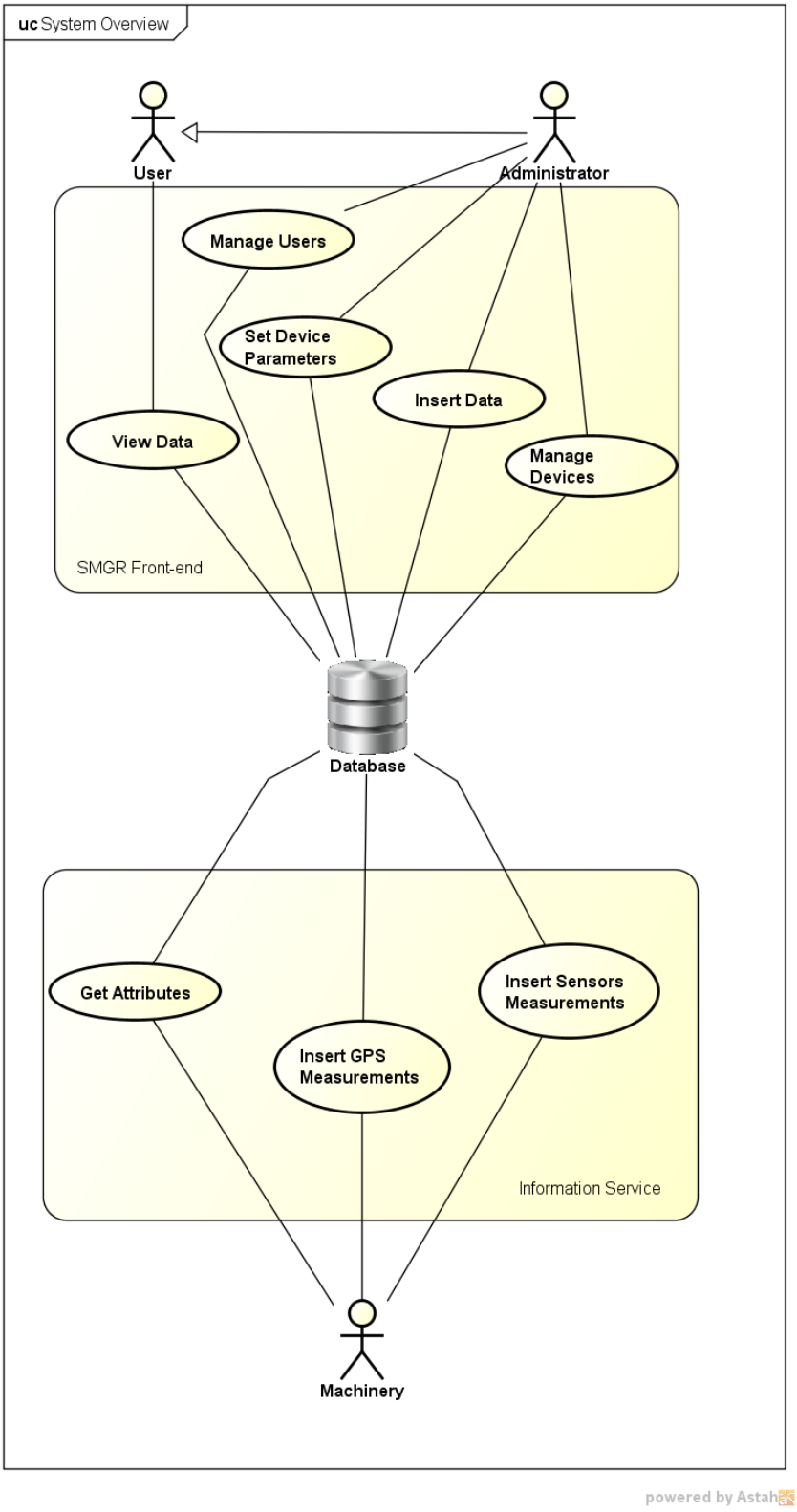

Figura 21: Diagrama de Casos de Uso do SMGR

3.2.1 Microsoft Windows Embedded

Para o desenvolvimento desse projeto, é absolutamente necessário que o sistema seja projetado na forma de um sistema embarcado, pelos seguintes motivos:

- A tarefa que o *hardware* e o *software* devem executar é bastante particular, não havendo outras aplicações em paralelo.
- O equipamento será totalmente dedicado ao propósito especificado.
- Os custos de cada equipamento devem ser baixos em comparação com um computador convencional.
- O sistema deve possuir baixo consumo de energia e deve ter pequenas dimensões.

O projeto descrito nesse documento deve utilizar um RTOS ao invés de um sistema operacional comum pois deve ser compatível com uma plataforma de *hardware* embarcada. Além de serviços básicos, como comunicações via rede e *drivers* para dispositivos como o GPS, a única aplicação executada será o sistema de monitoramento e gestão remota de obras, fazendo uso completo do *hardware*. Essa aplicação deve atender a requisitos de tempo para pedidos do operador. Além disso, deve ser possível customizar o SO para o *hardware*, de forma que, quando ligado, execute exatamente as funções requeridas pelo projeto.

Sistemas desse tipo geralmente possuem um custo muito elevado. Devido ao convênio da Universidade Tecnológica Federal do Paraná com o Microsoft DreamSpark, programa que disponibiliza gratuitamente *softwares* da Microsoft para estudantes, o Windows Embedded foi escolhido para o desenvolvimento do projeto em questão. Em análise feita em (DEDICATED SYSTEMS, 2009), o Windows Embedded foi testado em diversos critérios para ser validado como um RTOS comparável a outros no mercado. Em uma escala de 0 a 10, os avaliadores do RTOS da Microsoft chegaram aos seguintes resultados: Arquitetura do RTOS (8), Documentação (7), Configuração do SO (7), Componentes de Internet (9), Ferramentas de Desenvolvimento (9), Instalação e BSP - *Board Support Package* (8), Resultados dos testes (7), Suporte (8). Com tais resultados, o Windows Embedded é aceitável como escolha para o desenvolvimento desse projeto, tanto em questões técnicas quanto custo.

Apesar da última versão do Windows Embedded ser a 7.0, esse projeto fará uso da versão 6.0. A maior parte das mudanças da versão 7.0 ocorreram na camada de usuário, não havendo ocorrido mudanças significativas no *kernel*. O módulo Wi-Fi™ utilizado para comunicações do SMGR não possui *drivers* disponíveis para a versão 7, apenas para a 6, logo foi necessário utilizar a versão anterior. Isso não trará prejuízos ao projeto, uma vez que os dois sistemas possuem as mesmas funcionalidades básicas.

3.2.2 Especificação de *Hardware*

O módulo embarcado do SMGR deverá utilizar como base o eBox 3310A. Adicionalmente deverão ser incorporados o módulo Wi-Fi™ Wyse VT6656 e o módulo GPS EM-411. Este protótipo utilizará como módulo aquisitor de sinais o Arduino Uno.

3.2.3 Especificação de *Software*

O módulo embarcado do SMGR é responsável pela coleta dos dados de um conjunto de sensores analógicos e digitais, e a posição GPS da máquina que está sendo monitorada. Após coletadas, essas informações são enviadas para o servidor localizado na obra. Esta seção detalhará a especificação de *software* do módulo embarcado.

3.2.3.1 Diagrama de Classes

O diagrama de classes é aquele que apresenta a estrutura do sistema que está sendo modelado no nível de classes e interfaces, além de mostrar seus atributos, restrições e relações (FAKHROUTDINOV, 2012). A Figura 22 apresenta o diagrama de classes do módulo embarcado SMGR.

A classe `InformationDatabase` é responsável pela comunicação com o registro do Windows, onde são salvos o ID do cliente, o tempo entre envio de informações ao servidor, as últimas coordenadas GPS, e o endereço do servidor. A tarefa principal dessa classe é criar funções para fácil acesso ao `RegistryKey` do .NET Framework, que manipula os tipos de dados do registro. Os métodos `Open` e `CloseRegistryKey` são chamados, respectivamente, no início e final de cada operação com o registro. De forma geral, o registro do Windows salva informações em dois tipos primitivos de dados: inteiros (`DWORD`, ou `Double Word`, por questões históricas do Windows) e `strings` (sequências de caracteres). Todas as informações do SMGR são guardadas em um desses dois tipos de dados. A seguir encontra-se a descrição de cada método da classe `InformationDatabase`:

- `OpenRegistryKey`: abre uma chave de registro para alterações.
- `CloseRegistryKey`: fecha a chave após o uso.
- `SetStringValue`: define um valor do tipo `string` para uma chave do registro.
- `SetDWordValue`: define um valor do tipo inteiro para uma chave do registro.

- `SetLastValidGPSLatitude`: faz uso da função `SetStringValue` para definir o valor de última latitude recebida do GPS. Guarda-se a latitude como `string` pois não há um tipo de dado para números de ponto flutuante, como é o caso de coordenadas.
- `SetLastValidGPSLongitude`: idem ao método anterior, porém aplicado à última longitude recebida do GPS.
- `SetServerAddress`: define o endereço do servidor, na forma de `string`. Geralmente esse valor é configurado manualmente no registro, pois faz parte do *setup* inicial da aplicação.
- `SetClientID`: define o ID do cliente, na forma de um inteiro. Tal método é chamado após a primeira conexão com o servidor, quando um ID é fornecido ao cliente.
- `SetPollingInterval`: define o tempo, em segundos, entre o envio de informações ao servidor. Esse método é chamado no início de cada iteração do *loop* do SMGR.

Além desses métodos para definir os valores, há também métodos para obtê-los.

A classe `SerialReader` define métodos para comunicação com a porta serial, tanto no caso do GPS quando do módulo de aquisição de sinais. Os membros dessa classe são: número máximo de tentativas de leitura (já que conexões na porta serial podem falhar), nome da porta serial (por exemplo COM1, COM2), *baud rate* (número de pulsos por segundo, no caso 4800 para o GPS e 9600 para os sensores), bit de paridade (utilizado para detecção de erros na transmissão), bits de dados (número de bits para cada caracter), bits de parada (bit enviado ao fim de cada caracter para fins de sincronização), e, por fim, o objeto da classe `SerialPort` do .NET Framework, que engloba todas as informações anteriores. A classe `SerialReader` fornece métodos para se obter e se definir cada um dos membros da classe, além de abertura e fechamento de conexão. O método `ReadInformation`, que lê uma linha da porta serial, é definido como abstrato e deve ser implementado em classes que especializam o `SerialReader`: `GPSReader` e `SensorsReader`.

A classe `GPSReader` é uma especialização da classe `SerialReader`, logo faz uso de toda sua estrutura. Porém, adiciona métodos que só fazem sentido para a comunicação com o módulo de GPS. O método abstrato `ReadInformation` da classe pai é implementado de forma a aguardar a primeira linha contendo a `string` "`$GPGGA`", que indica o início das coordenadas GPS. Já o método `ParseInformation` extrai da linha recebida as coordenadas de latitude e longitude, caso o módulo esteja conectado aos satélites. A classe `GPSReader` guarda uma estrutura (`struct`) que contém a latitude e a longitude na

forma de números double, chamada `TCoordinates`. Essa estrutura é preenchida a cada obtenção bem-sucedida de coordenadas, e é utilizada para enviar os dados ao servidor.

A classe `SensorsReader` também trata-se de uma especialização da classe `SerialReader`, porém com métodos específicos ao módulo de aquisição de sinais. De forma similar ao `GPSReader`, implementa o método abstrato `ReadInformation` da classe pai, `SerialReader`, de forma a aguardar o recebimento da linha contendo o valor `0x41` em hexadecimal, que representa o início dos valores de sensores vindos do microcontrolador. Ao final do recebimento, guarda os valores dos sensores em uma `string`, em que cada valor é separado por vírgulas. São 17 valores ao total: 11 dos sensores digitais e 6 dos analógicos. Quando o recebimento é bem-sucedido, os valores são enviados ao servidor.

Por fim, a classe `Application` engloba o *loop* principal da aplicação, sendo responsável por instanciar objetos das demais classes e chamar seus métodos. A classe `Application` contém os seguintes objetos:

- `webService`, da classe `TCCWS`, criada automaticamente quando adiciona-se uma *Web Reference* ao projeto.
- `gpsReader`, da classe `GPSReader`.
- `sensorsReader`, da classe `SensorsReader`.
- `offlineMode`, *flag* booleana (verdadeiro/falso) que indica se o cliente está em executando em modo *offline*, caso a conexão com o servidor não tenha sido feita.

3.2.3.2 Diagramas de Sequência

O diagrama de sequência é o tipo mais comum de diagrama de interação. Ele é focado na troca de mensagens entre os objetos (FAKHROUTDINOV, 2012). A Figura 23 apresenta o diagrama de sequência da classe `Application`. Para maiores detalhes, consulte o Anexo A.

3.3 Especificação do Banco de Dados

No Modelo proposto para o banco, há cinco entidades:

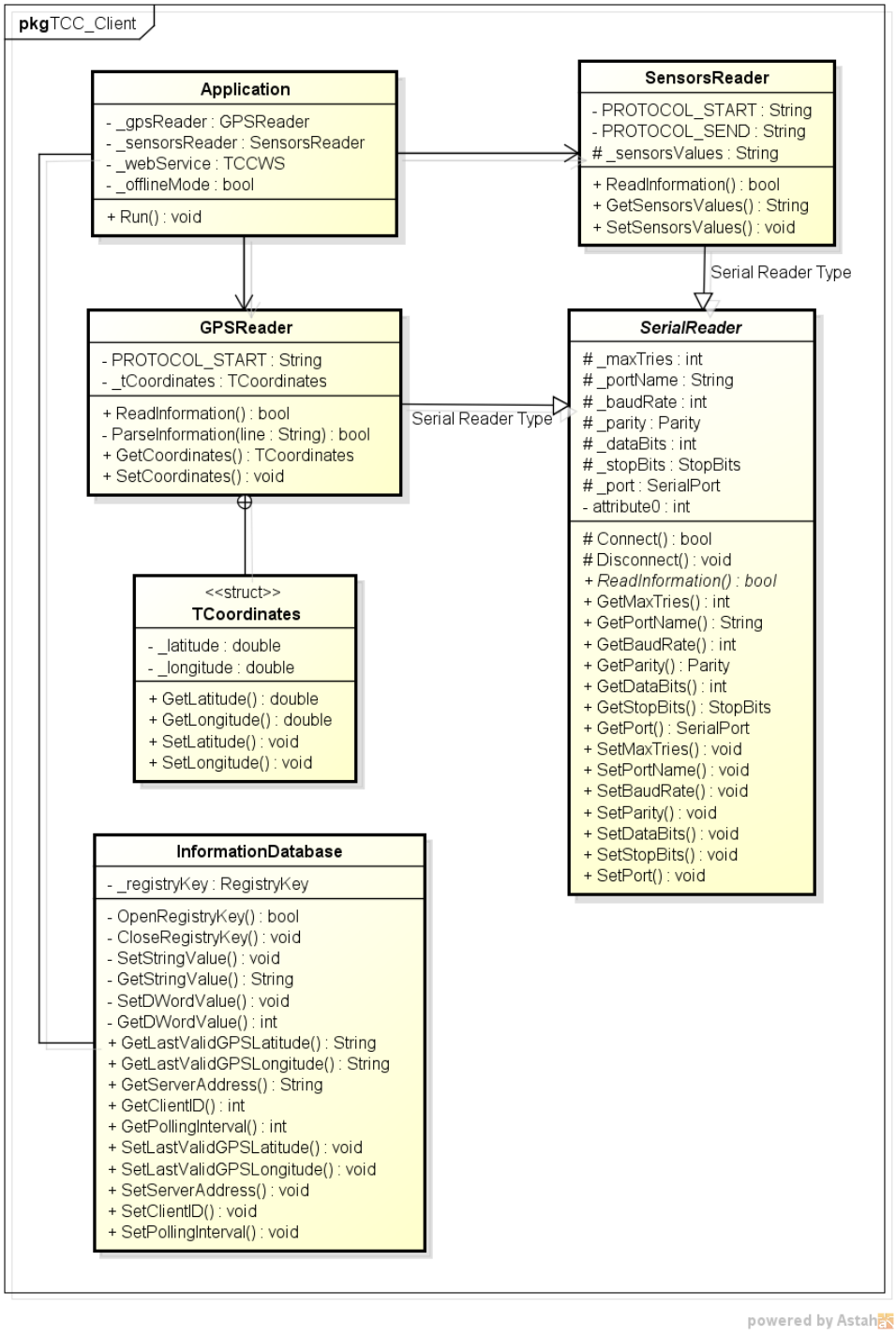

Figura 22: Diagrama de Classe do Módulo Embarcado SMGR

- `Client`: Refere-se ao módulo de aquisição de dados. Possui um identificador único, a data em que foi adicionado ao sistema, nome, intervalo entre novas aquisições de dados, e um sinalizador para indicar se é um dispositivo gerenciado ou não.

- `Properties`: Tabela para inserir propriedades adicionais ao cliente. Possui uma chave de identificação, uma chave para relacionar ao modulo, nome da propriedade

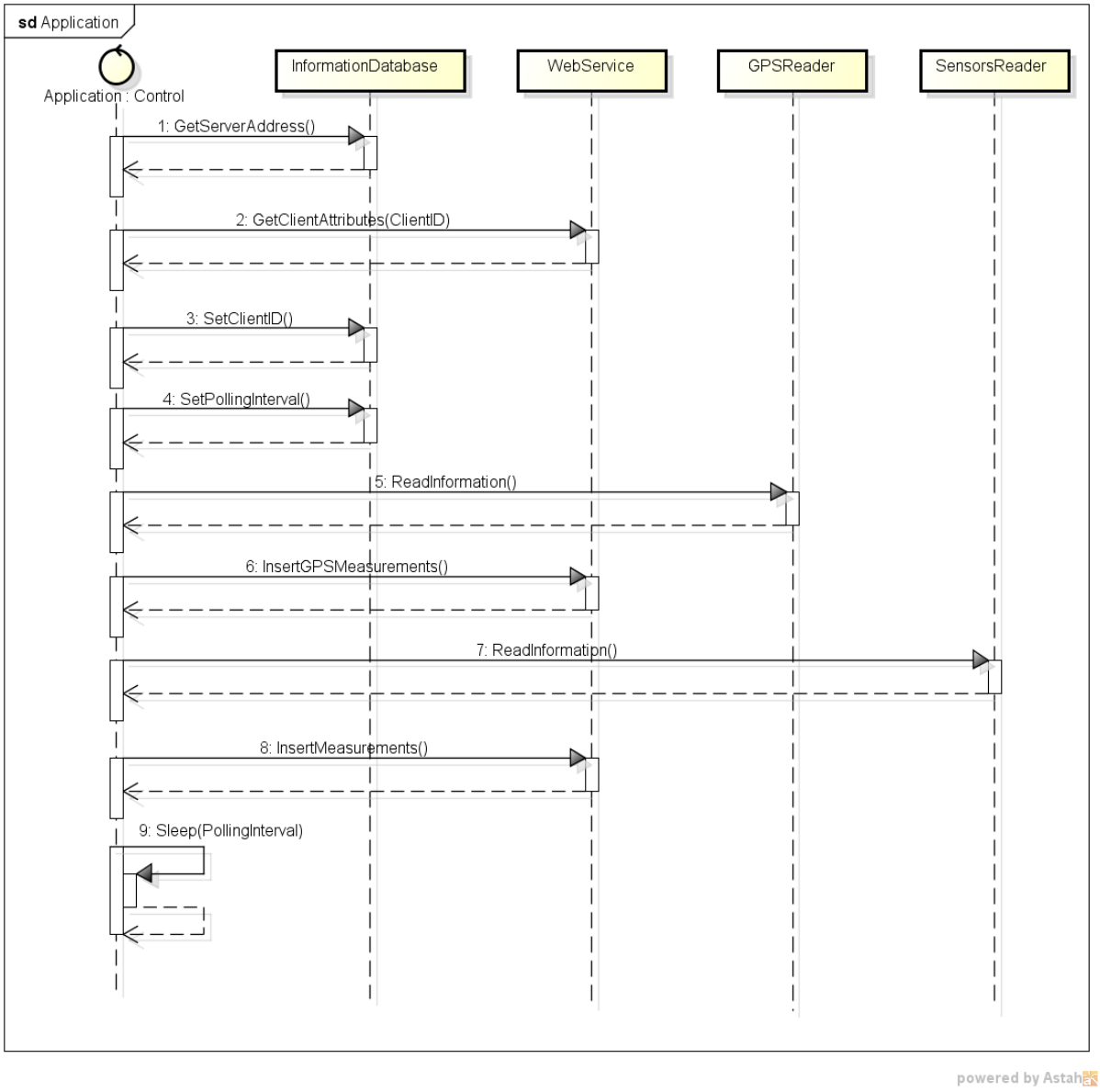

Figura 23: Diagrama de Sequência da Classe *Application*

e valor.

- `Sensor`: Armazena características de cada sensor ligado ao aquisitor de sinais, possui uma chave única de identificação, uma chave para relacionar ao seu módulo, nome, valores máximo e mínimo que podem ser medidos (utilizado para conversão de escala, no caso de sensores analógicos), índice na porta do aquisitor de sinais onde o módulo está instalado, um sinalizador pra indicar se é um sensor digital ou analógico e o texto de formato para apresentação na tela.

- `Measurement`: Armazena medidas obtidas por um sensor. Possui uma chave de identificação, chaves para relacionar a medida a um módulo e ao sensor, data de

aquisição e o valor medido.

- `GPSMeasurement`: Similar ao Measurement, porém, armazena as informações de latitude e longitude coletadas pelo GPS e a data da coleta. Possui um identificador único e para associar com o módulo que adquiriu o sinal possui uma chave de relacionamento.

Este modelo foi criado seguindo a terceira forma normal para sistemas de banco de dados, na qual não há nenhum atributo na tabela que não seja transitivamente dependente da chave candidata (HEUSER, 2009). Na Figura 24 apresenta-se o diagrama do banco de dados, com as relações entre as tabelas e os tipos de dado.

3.4 Especificação da Aplicação Web

A aplicação web deverá ser implementada utilizando a tecnologia ASP .NET. A Figura 25 apresenta o fluxo geral da aplicação.

Logo após o usuário efetuar o login ele deverá ser redirecionado para a página principal da aplicação, onde ele poderá acessar as funcionalidades básicas do sistema: Usuários, Dispositivos e Mapa.

Ao clicar sobre a funcionalidade Usuários, o usuário administrador poderá visualizar, cadastrar e excluir usuários do sistema.

Clicando sobre a funcionalidade Dispositivos, o usuário deverá visualizar os dispositivos cadastrados. Se este for um administrador, deverá adicionalmente ter a capacidade de alterar configurações, editar sensores e propriedades e adicionar dispositivos não gerenciados.

Ao clicar sobre a funcionalidade Mapa, o usuário deverá ser redirecionado para outra página onde ele poderá visualizar a posição atual de todos os dipositivos cadastrados no sistema e que possuam o módulo embarcado.

3.5 Considerações sobre o capítulo

O presente capítulo tratou da especificação formal do sistema e dos módulos que compõem o SMGR. Em primeiro lugar foi feito o levantamento dos requisitos funcionais e não funcionais, que definiram as propriedades essenciais, desejáveis e restrições. Além

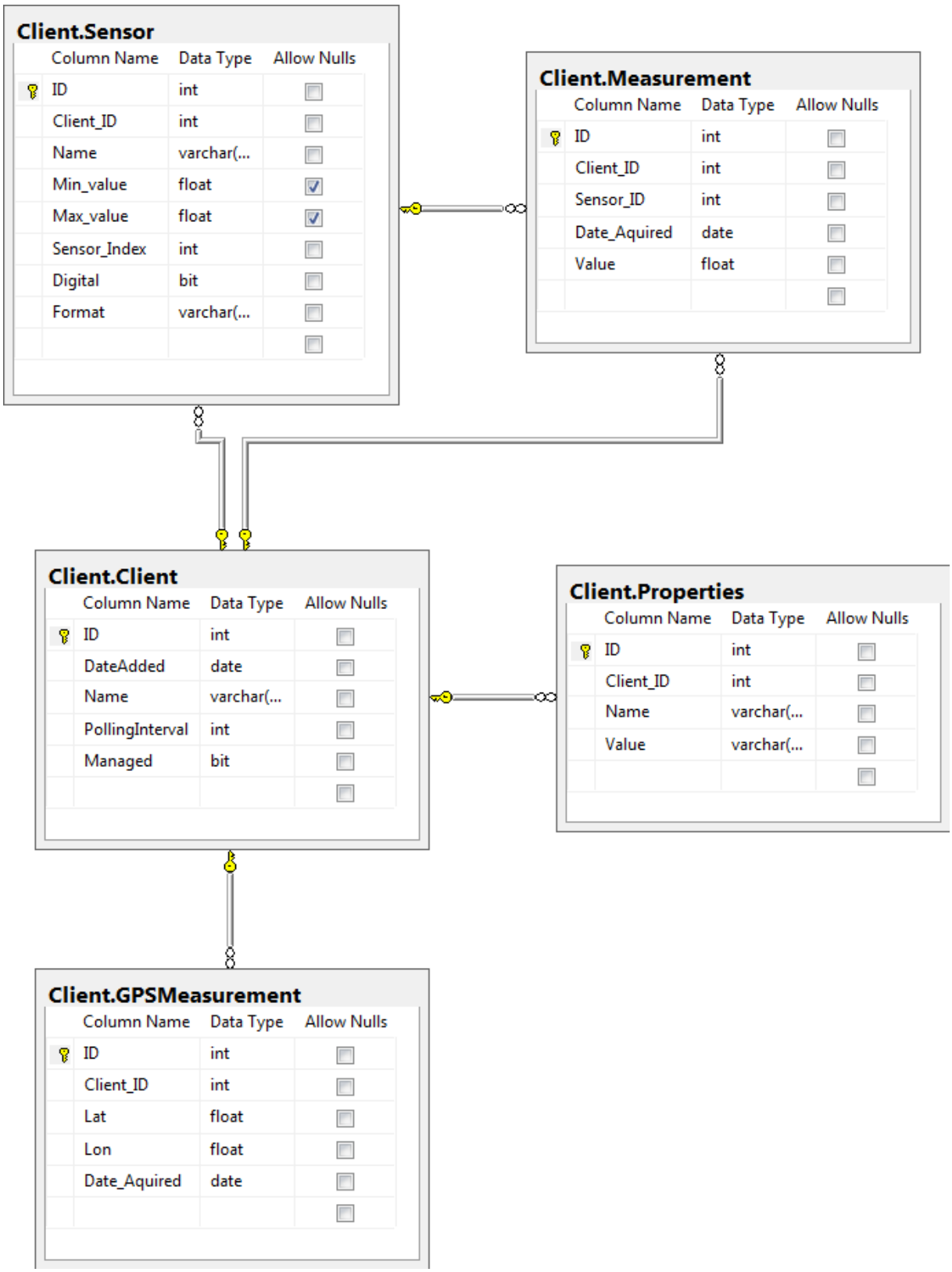

Figura 24: Diagrama ER do Banco de Dados

dos requisitos, especificou-se um diagrama de casos de uso, que define as ações de cada item do sistema com elementos externos, chamados atores. Em seguida, foi especificado

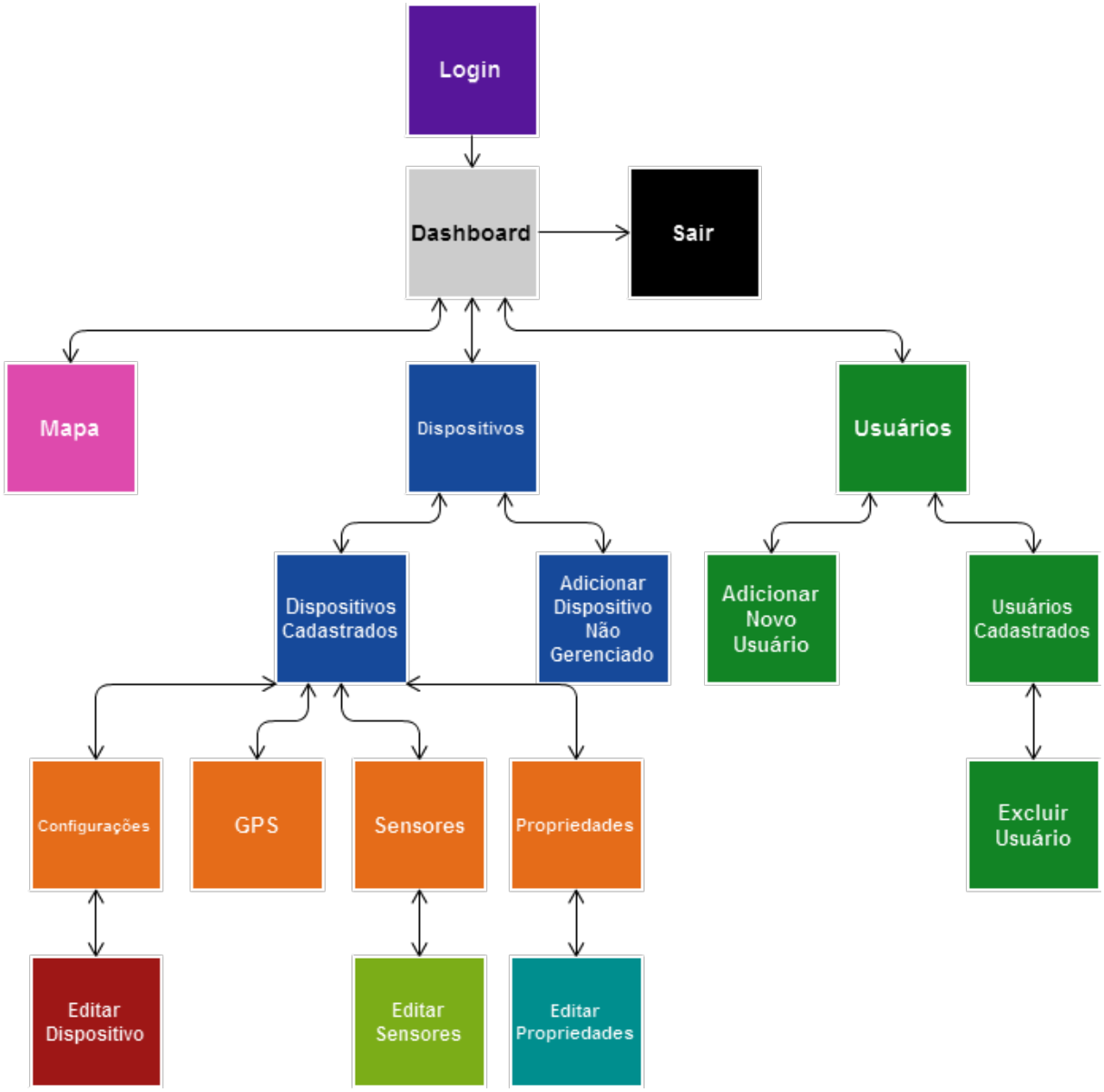

Figura 25: Fluxo Geral da Interface da Aplicação Web (camada de usuário)

o sistema operacional utilizado, o Windows Embedded Compact, amplamente utilizado pela indústria e validado em critérios técnicos por entidades de controle de qualidade. O *hardware* utilizado também foi especificado, no caso, o kit de desenvolvimento eBox 3310A, o módulo GPS EM-411, o módulo Wi-Fi™ Wyse VT6656 e o módulo de aquisição de sinais baseado no microcontrolador ATmega 328P. Quanto ao *software* embarcado, foram feitas especificações incluindo diagramas de classe e de sequência. A seguir foi feita a descrição formal do banco de dados, incluindo todas as tabelas e as respectivas colunas. Por fim, especificou-se a aplicação web, incluindo seu comportamento esperado e as seções de navegação.

4 DESENVOLVIMENTO DO SISTEMA

Este capítulo descreve algumas das etapas que foram realizadas para a concepção do SMGR, desde a criação da imagem do Windows Embedded do módulo embarcado até a implementação da aplicação web para visualização dos dados coletados.

4.1 Customização do Windows CE

Esta seção, escrita com base em Phung (2010), tratará do Platform Builder e do processo de criação de uma imagem customizada do RTOS Windows Embedded CE 6.0 R3. Serão mostradas as ferramentas utilizadas e suas devidas configurações para que o sistema operacional possa ser executado no dispositivo utilizado no projeto.

4.1.1 Ambiente de desenvolvimento

O Windows Embedded CE não é distribuído na forma de um sistema operacional pronto, como é comum em outras versões do Windows. Cabe ao desenvolvedor de sistemas embarcados definir quais requisitos o RTOS deverá atender para suas necessidades e, por fim, gerar a imagem do SO para executar no dispositivo. Define-se por imagem o arquivo binário gerado pelo processo de customização do sistema operacional (PHUNG, 2010). Tal processo envolve uma série de ferramentas que, no caso do CE 6.0, são instaladas em uma máquina de desenvolvimento equipada com o Windows XP ou Windows 7.

De forma geral, o processo de customização da imagem do SO é feita através do Platform Builder, um *plugin* para o Microsoft Visual Studio 2005. Nessa fase, o desenvolvedor escolhe a partir de um catálogo quais componentes deseja incluir em seu projeto, como por exemplo o BSP (*Board Support Package*), *drivers* para *hardware* específico, funcionalidades do sistema operacional, tais como suporte de conectividade a redes TCP/IP, etc. Componentes muito específicos que não estão disponíveis no catálogo podem ser obtidos junto ao fabricante e posteriormente incluídos de forma manual, através de arquivos DLL

adicionados ao *kernel* e diretivas no registro do Windows. De fato, a maior parte das configurações do Windows CE 6.0 são armazenadas em seu registro. Erros em sua especificação podem fazer com que o sistema não consiga inicializar-se. Recomenda-se utilizar o Platform Builder para o CE 6.0 no Windows XP e Visual Studio 2005, ao contrário da versão do CE 7.0, que executa no Visual Studio 2008 e Windows posteriores ao Vista. De forma resumida, os passos para se criar um sistema baseado no Windows Embedded são os seguintes:

- Desenvolver um BSP, ou utilizar uma versão fornecida pelo fabricante ou terceiros.
- Customizar uma imagem do sistema operacional.
- Gerar a imagem para o *hardware* em questão.
- Testar a imagem.
- Gerar um kit de desenvolvimento de *software* (SDK) específico do sistema operacional recém criado.
- Desenvolver a aplicação em linguagem nativa ou gerenciada, utilizando o SDK.
- Testar a aplicação.
- Implantar a solução final no dispositivo.

Para uso nesse projeto, as ferramentas de desenvolvimento foram obtidas através do DreamSpark, programa educacional da Microsoft que disponibiliza *software* a alunos do Departamento de Informática (DAINF) e outras instituições de ensino ao redor do mundo. Através do portal do DreamSpark, a equipe teve acesso de forma gratuita ao Windows XP, o Visual Studio 2005 e o Windows CE 6.0, que traduz-se na instalação do Platform Builder e suas opções de catálogo. Vale ressaltar que as versões de *software* disponíveis no DreamSpark são completas, não apenas demonstrações com tempo limitado.

4.1.1.1 Instalação das ferramentas de desenvolvimento

Primeiramente, deve-se instalar o Microsoft Windows XP, preferencialmente com o último Service Pack (versão 3). Para esse projeto, foi utilizada a versão de 32 bits (x86) em uma máquina virtual. Instalado o sistema operacional, deve-se instalar o Microsoft Visual Studio 2005. Para que o Platform Builder funcione corretamente, é necessário instalar o Service Pack 1 para o Visual Studio 2005. Em seguida, instala-se o pacote base

do Windows Embedded CE 6.0. Como o Windows Embedded teve diversas atualizações até o lançamento do CE 7.0, instala-se o Service Pack 1 seguido do CE 6.0 *Release* 2 (R2) e *Release* 3 (R3). Com isso, o ambiente de desenvolvimento básico já está configurado.

Após a instalação das ferramentas, certos componentes adicionais são necessários para o correto funcionamento do projeto. O primeiro deles é o BSP para a eBox 3300, *hardware* no qual o RTOS e o *software* embarcado serão executados (Figura 26). O BSP consiste em um conjunto de *drivers* específicos ao *hardware* em questão, como por exemplo suporte a seu processador x86, vídeo, rede e uma camada de adaptação para o CE 6.0. Caso esse BSP não seja instalado, o Platform Builder suportará somente o processador ARMV4I.

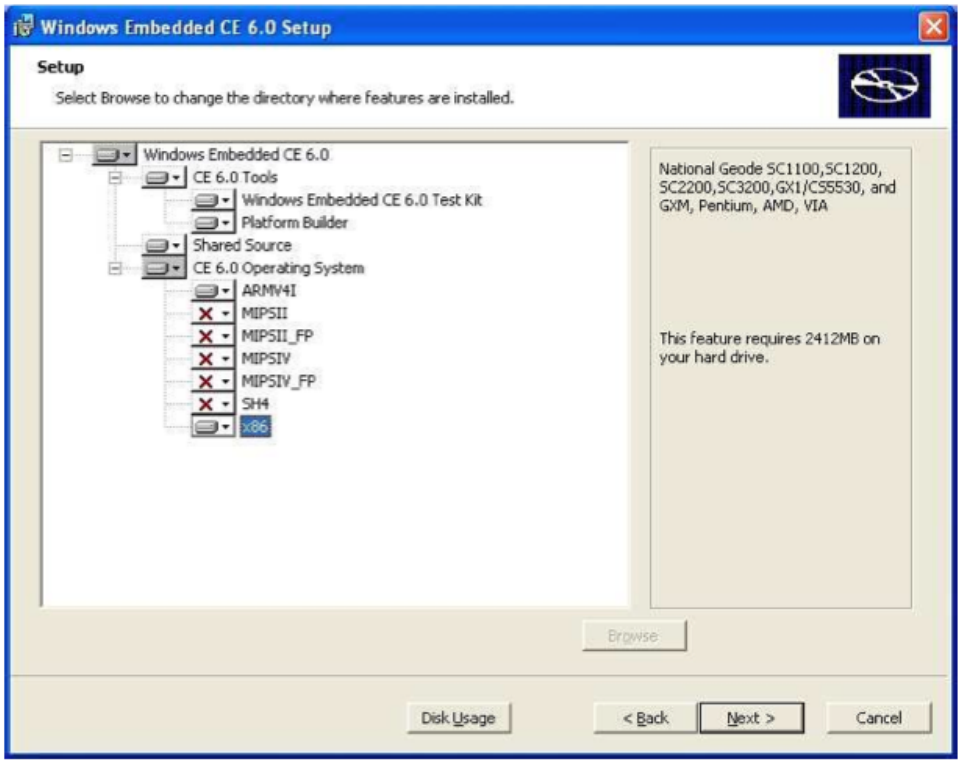

Figura 26: Instalação do CE 6.0

O segundo componente adicional necessário é o CoreCon, ou *Connectivity Framework*. Através do CoreCon será possível conectar o Visual Studio ao *hardware* através de uma rede TCP/IP padrão, facilitando o trabalho de depuração e transferência de arquivos ao sistema embarcado. Por fim, instala-se o AutoLaunch, componente responsável por iniciar automaticamente o *software* do SMGR assim que o *hardware* for ligado. Também pode-se utilizar o AutoLaunch para executar o CE Remote Display, aplicativo que permite visualizar remotamente a tela do Windows CE, de forma semelhante ao protocolo VNC. Para que o AutoLaunch funcione corretamente, deve ser configurado através de entradas no registro do Windows. As entradas apresentadas no Código 4.1 definem a execução

automática do CE Remote Display e do CoreCon, para fins de exemplo.

Código 4.1: Entradas no registro do Windows

```
[HKEY_LOCAL_MACHINE\Startup]
 ''Process0''=''cerdisp -c''
 ''Process0Delay''=dword:00001388 ; 5000 ms em hexadecimal
 ''Process1''=''ConmanClient2.ex''
 ''Process1Delay''=dword:00002710
```

4.1.2 Desenvolvimento da imagem do Windows CE 6.0 R3

Segundo Phung (2010), para o desenvolvimento da imagem do Windows CE, cria-se um projeto do tipo OS Design, encontrado na categoria Platform Builder for CE 6.0 do Visual Studio 2005. Nesse momento, um *wizard* será mostrado para auxiliar na configuração inicial da imagem do CE (Figura 27).

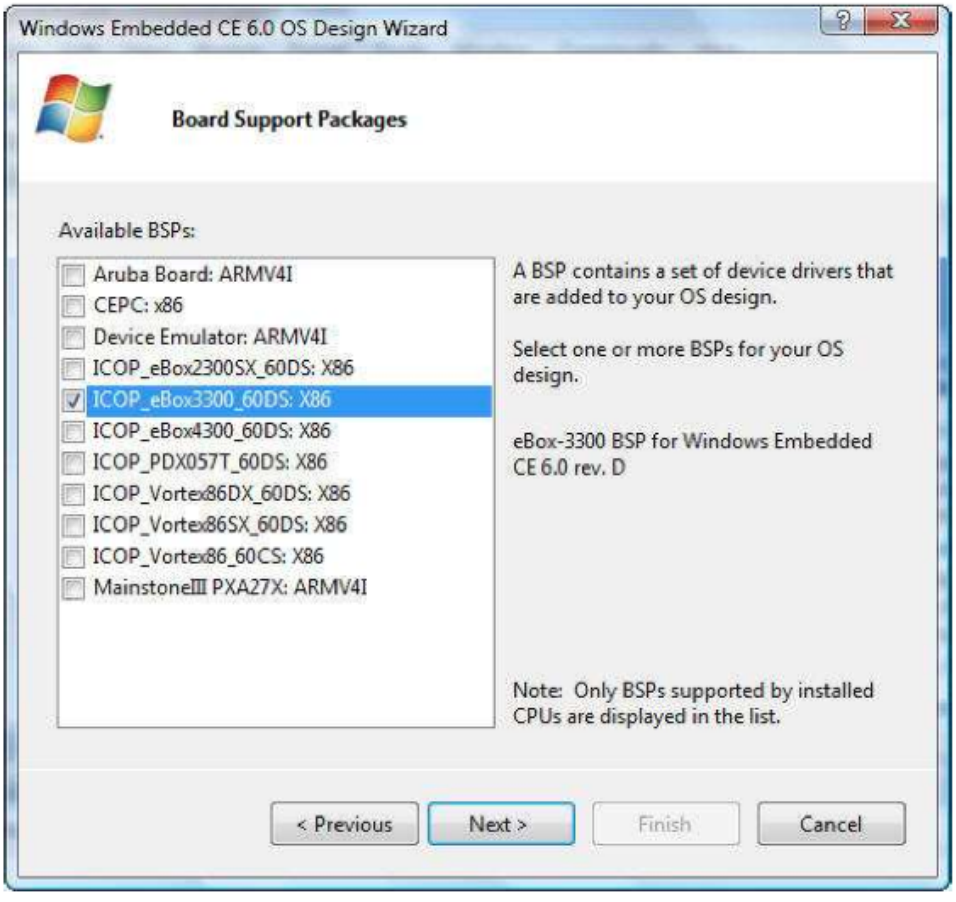

Figura 27: Assistente de Criação de SO

O primeiro passo consiste em selecionar o BSP, que contém os *device drivers* e a

camada de adaptação do *hardware* para uso do CE. Em seguida, deve-se escolher um *template* inicial do tipo de dispositivo que será utilizado. Para esse projeto, o *template* de *design* escolhido foi de dispositivo industrial. O próximo passo permite a escolha de componentes tais como o .NET Compact Framework 3.5 (Figura 28). Por fim, escolhem-se opções de conectividade, como suporte ao protocolo TCP/IP v4, v6, etc. Feito isso, o painel de catálogo será aberto, permitindo uma customização em maiores detalhes.

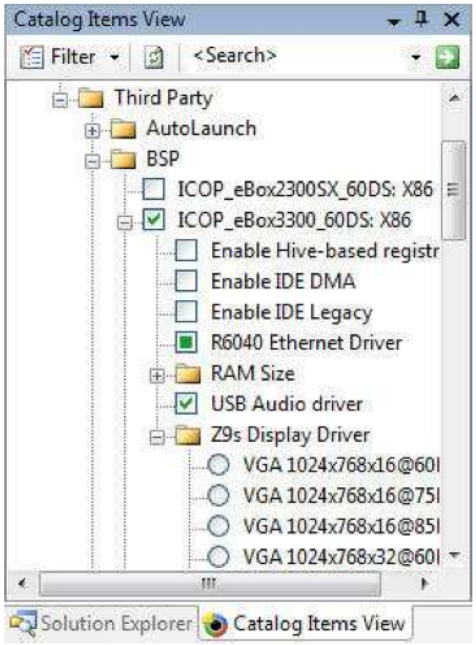

Figura 28: Catálogo de componentes

A lista a seguir mostra os principais componentes necessários ao SMGR, selecionados a partir do catálogo:

- *AutoLaunch* v200 x86
- BSP ICOP eBox 3310A x86
- ATAPI *Storage Driver*
- *Boot from IDE or Compact Flash*
- *Hive-based Registry Support* (responsável por manter o registro de forma permanente, já que por padrão é armazenado em RAM)
- USB *Driver*
- VGA 640x480x16@60 *Driver*

- *Serial Ports* COM1/COM2
- 512 MB RAM *Support*
- CoreCon v200 x86
- VIA VNUWLC6 USB *Wireless* LAN
- .NET Compact Framework 3.5
- XML *Core Services*
- Core *OS Services*

Todos os ítens da lista são vitais para o SMGR. Em especial, o .NET Compact Framework 3.5 é uma versão do .NET Framework especialmente criada para sistemas embarcados. Esse item permite o desenvolvimento de aplicações em código gerenciado, ao contrário de código nativo como é o caso das linguagens C/C++. O .NET Framework permite o desenvolvimento de aplicações com performance equivalente ao código nativo, porém fornece APIs (*Application Programming Interfaces*) e poderosas ferramentas ao programador. O SMGR foi escrito em C#, uma das linguagens do .NET Framework. Outro importante item da lista é o *Hive-based Registry Support*. Por padrão, sempre que o Windows CE é desligado, todas as alterações feitas em seu registro são destruídas. Isso se traduz em uma perda de configurações, o que não é aceitável para o projeto. Por esse motivo, a opção de *Hive-based Registry* é selecionada. Internamente, cada alteração feita no registro é salva em um arquivo chamado "`hive`" na memória permanente, no caso, o cartão de memória *Compact Flash* onde também estão o *software* do SMGR e o Windows CE.

Após configurados os ítens do catálogo, pode-se abrir o gerente de configurações (Configurations Manager) para definir se a imagem será gerada em modo *Debug* ou *Release*. A seguir, serão explicadas as principais diferenças entre cada um.

- **Debug:** a imagem *Debug* é um arquivo binário com um tamanho maior que a versão *Release*, já que inclui informações detalhadas sobre o estado do sistema além de suas atividades durante a inicialização. É interessante gerar uma imagem nesse modo para testar se todos os *drivers* estão sendo carregados corretamente, além de se poder observar os locais exatos de erros caso algum ocorra. Geralmente, faz-se o debug de uma imagem através de um cabo serial. Durante a fase de *boot*, o RTOS irá

enviar uma série de mensagens a um hiper-terminal conectado, permitindo analisar precisamente o estado do sistema.

- **Release:** a imagem *Release*, em contraste com a *Debug*, possui um tamanho bastante reduzido, justamente por ser a versão final que irá executar no dispositivo para a qual foi customizada. Possui uma série de otimizações que a torna mais rápida para uso convencional.

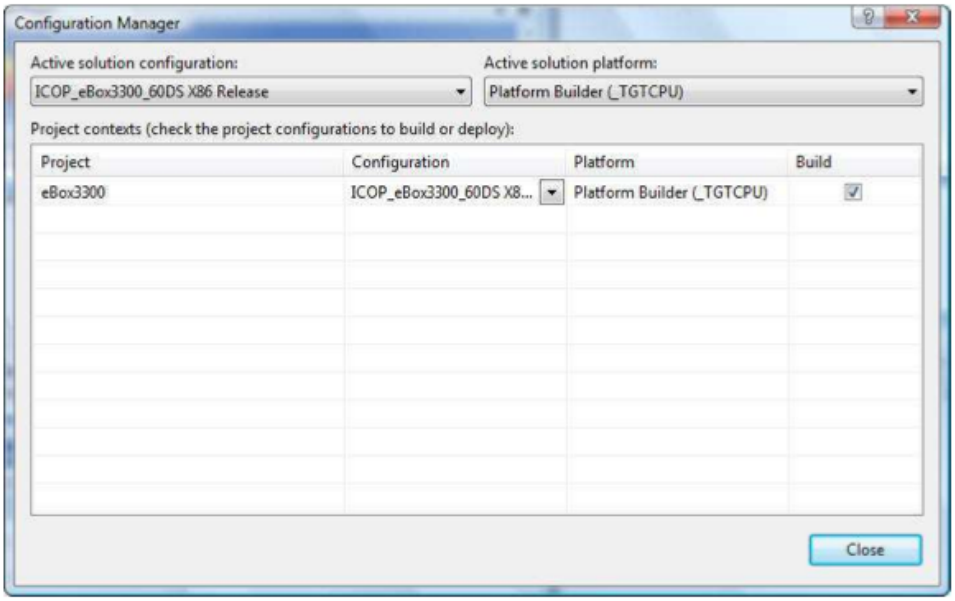

Figura 29: Gerente Configurações

Com a configuração terminada, a imagem do CE está pronta para ser compilada. Ao selecionar a opção *Build Solution* (Compilar Solução) no menu *Build* (Compilar), o processo terá início e pode levar vários minutos. No caso do SMGR, a compilação leva em torno de 30 minutos. É importante notar a aba de Saída (*Output*), em que qualquer erro será reportado juntamente com possíveis medidas corretivas (Figura 30).

Caso a imagem tenha sido gerada no modo *Debug*, pode-se utilizar a opção Executar (*Run*) do Visual Studio para fazer download via Ethernet e verificar a inicialização da imagem com a possibilidade de depurar o código fonte de *drivers*. Em modo *Release* não há essa opção de depuração e, geralmente, a imagem é copiada diretamente para o cartão *Compact Flash*. Ao final do processo de compilação, a pasta de projeto conterá o arquivo `NK.BIN` (geralmente em torno de 50 MB), que contém o Windows Embedded CE 6.0 R3 completo. Um arquivo de inicialização é configurado dentro do Compact Flash indicando o caminho da imagem, que é carregada e iniciada assim que o *hardware* é ligado. A Figura 31 ilustra o *desktop* padrão do Windows Embedded.

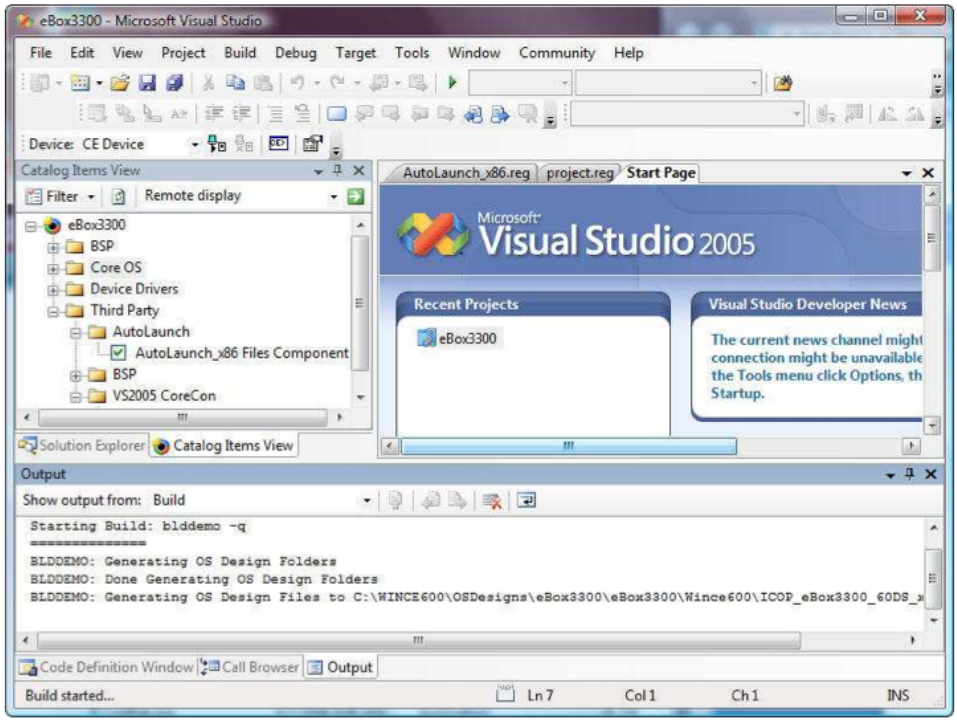

Figura 30: Imagem sendo compilada

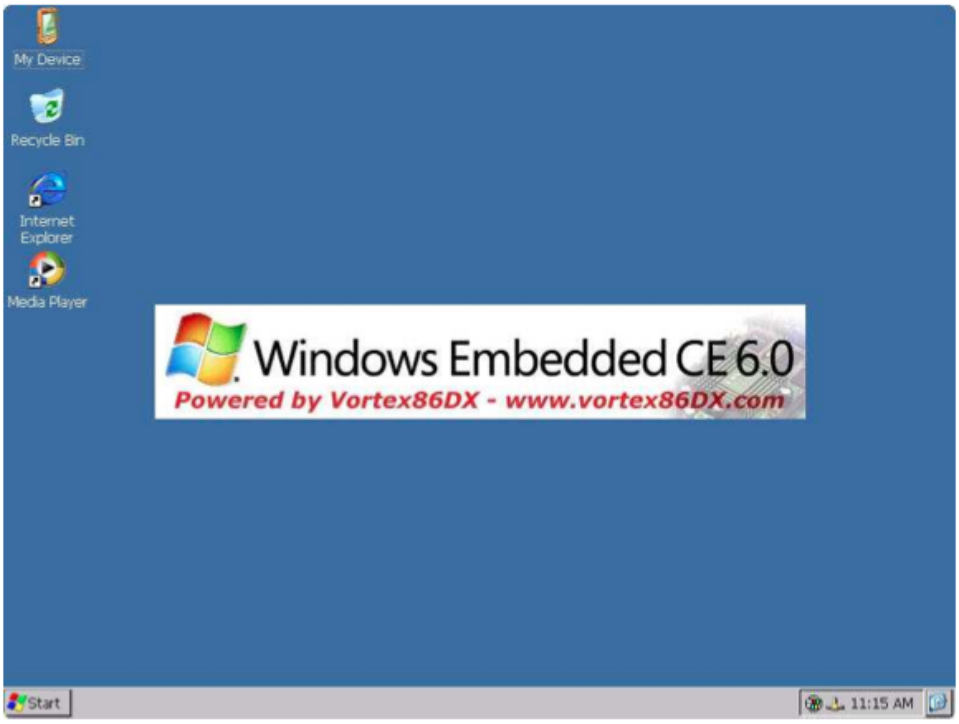

Figura 31: Desktop padrão do Windows CE 6.0

4.1.3 Criando um SDK customizado

Após compilada a imagem final do Windows Embedded, deve-se criar um SDK, ou *Software Development Kit*. Sem um SDK, não é possível desenvolver aplicações para o

sistema operacional recém criado, uma vez que não haveria maneira de saber quais funções o SO suporta. Através do SDK, a aplicação sabe exatamente quais funcionalidades estão disponíveis no dispositivo em que irá executar, para que, assim, possa ser compilado um código compatível. O SGMR consiste em uma aplicação escrita em C#, do .NET Framework, e requer o SDK do sistema operacional para poder ser compilado e executar na imagem customizada.

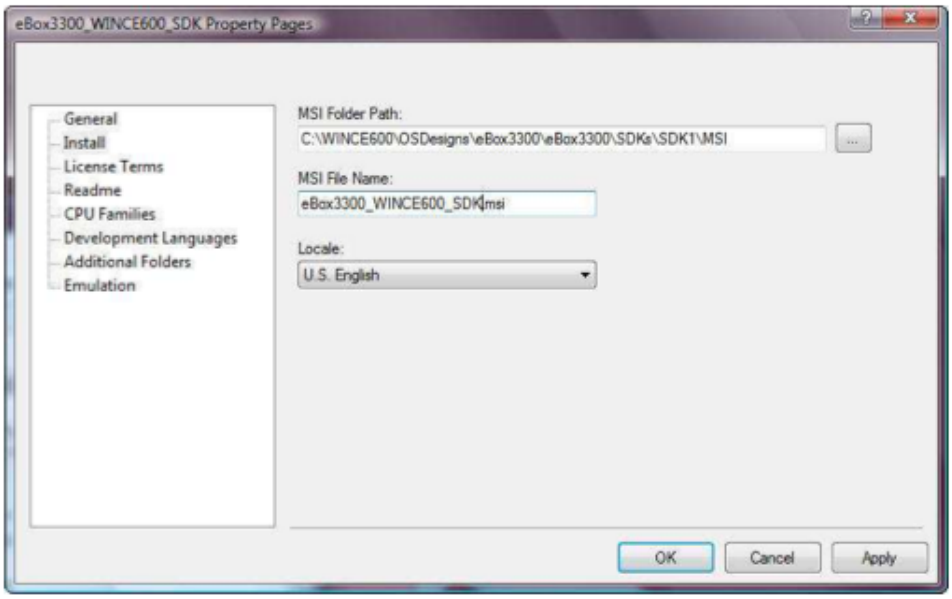

Figura 32: Propriedades do SDK

A criação do SDK é feita através da opção Adicionar Novo SDK (Add New SDK), localizada no menu Projeto (*Project*). No diálogo de configuração do SDK (Figura 32), pode-se personalizar diversas informações sobre a plataforma, como por exemplo o nome do desenvolvedor, um arquivo Leia-me, entre outros. É importante garantir que o suporte a código gerenciado esteja marcado, uma vez que o SMGR não funciona sem ele. Ao final do processo, tem-se um arquivo de instalação `*.MSI` que pode ser executado em qualquer máquina contendo o Visual Studio 2005. Por meio de um assistente de instalação gráfico, qualquer computador pode se tornar uma estação de desenvolvimento para o Windows Embedded CE 6.0 customizado.

4.1.4 Processo de *boot* pelo cartão *Compact Flash*

Após a compilação da imagem do Windows Embedded, tem-se um arquivo chamado `NK.BIN` que deve ser transferido para o eBox, de forma que possa ser inicializado e o sistema operacional entre em execução. Esse processo de inicialização, chamado *boot*, será feito utilizando-se um cartão do tipo *Compact Flash*.

Para configurar o cartão *Compact Flash*, utiliza-se a ferramenta DiskPrep, desenvolvida por Kennett (2010). O DiskPrep copia uma série de arquivos para o cartão, de forma que o `NK.BIN` seja colocado na memória e tenha seu primeiro endereço de instrução chamado. Um dos arquivos copiados é o `BIOSLoader`, uma ferramenta capaz de carregar uma imagem do Windows Embedded no formato `BIN`, ou fazer *boot* através de um servidor configurado via rede. No caso do SMGR, o sistema operacional será carregado a partir de uma imagem local, também armazenada no cartão CF.

Uma vez obtido o DiskPrep(Figura 33) de Kennett (2010), insere-se o cartão *Compact Flash* através de um adaptador USB, caso necessário. No DiskPrep, seleciona-se o *drive* em que se encontra o CF (PHUNG, 2010). Deve-se ter o cuidado de copiar quaisquer arquivos existentes no cartão, uma vez que ele será formatado utilizando o sistema de arquivos `FAT32`. Uma opção disponível no DiskPrep é a customização da logomarca que aparecerá no monitor, caso algum esteja conectado na saída VGA, durante o procedimento de *boot*. Para o SMGR, que não requer um monitor externo ligado, será exibida apenas a logomarca padrão do Windows CE 6.0. Pode-se criar, opcionalmente, um arquivo de configuração chamado `BOOT.INI`, que contém configurações tais como o nome do arquivo de imagem do Windows CE e a resolução de vídeo. Criou-se um arquivo especificando a imagem `NK.BIN` e uma resolução de vídeo 640x480x16. É importante notar que essa resolução é necessária caso se queira utilizar o CE Remote Desktop, que permite a visualização remota da tela do sistema operacional. É muito útil utilizar o Remote Desktop, uma vez que elimina a necessidade de se conectar um monitor, mouse e teclado no eBox 3310A.

Feitas essas configurações, o cartão *Compact Flash* é formatado e os arquivos de configuração são copiados, deixando-o pronto para uso com o eBox e *boot* para o Windows CE 6.0 customizado.

4.2 Módulo Wi-Fi™ USB Wyse VT6656

O Wyse VT6656 possui *driver* compatível somente com o Windows Embedded CE 6.0. Foi devido a essa limitação que a equipe optou por não utilizar a versão mais recente do RTOS da Microsoft, o CE 7.0. Foram feitos testes na tentativa de utilizar o *driver* na última versão, porém com resultados negativos. Dessa forma, a versão 6.0 foi escolhida e executou o *driver* sem maiores dificuldades.

No website da VIA Networking Technologies (2012) há um link para download do *driver* compatível com o Windows CE 6.0. Trata-se de um arquivo de instalação MSI,

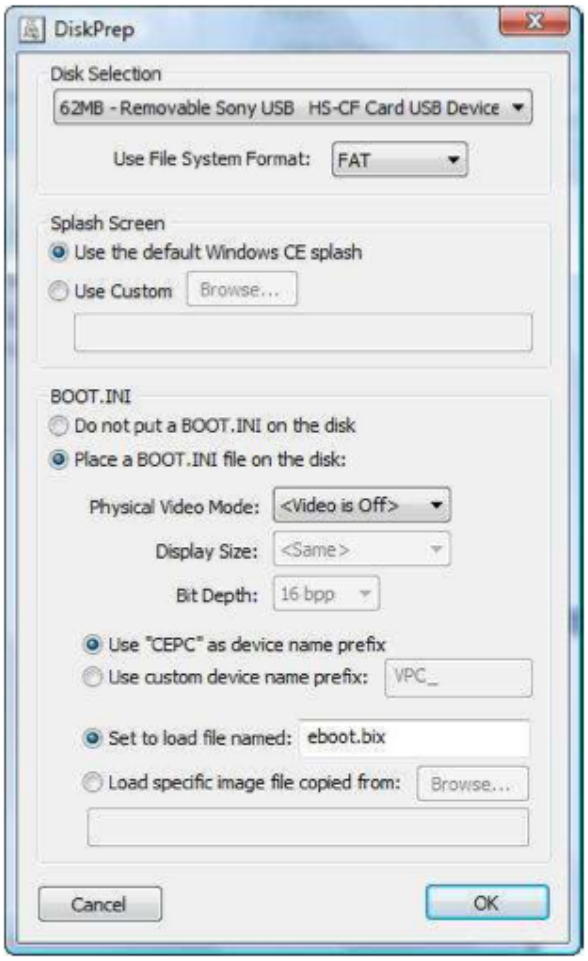

Figura 33: Captura de tela da ferramenta DiskPrep

que, quando instalado, adiciona um novo componente no catálogo do Platform Builder. Deve-se, portanto, instalar o *driver* na mesma máquina em que será customizada a imagem do sistema operacional.

Com o *driver* instalado, basta selecioná-lo no catálogo do Platform Builder para que seja adicionado no processo de compilação da imagem. O caminho do catálogo que contém o *driver* é "`Catalog\Third Party\Device Drivers\VNT VNWLC6 USB Wireless LAN`" (VIA TECHNOLOGIES, 2007). Essa operação automática realiza três passos fundamentais:

- Inclui uma série de entradas no registro do Windows, configurando a operação do módulo Wi-Fi™.

```
[HKEY_LOCAL_MACHINE\Drivers\USB\ClientDrivers\VNUWLC6]
  ""iniport"=""VNUWLC6"
  ""Prefix"=""NDS"
```

```
  ''Dll"=''NDIS.dll"

[HKEY_LOCAL_MACHINE\Drivers\USB\LoadClients\5642_12676\
                                        Default\Default\VNUWLC6]
  ''Dll"=''VNUWLC6.dll"

[HKEY_LOCAL_MACHINE\Comm\VNUWLC6]
  ''DisplayName"=''VNT VT6656 Wireless LAN"
  ''Group"=''NDIS"
  ''ImagePath"=''VNUWLC6.dll"

[HKEY_LOCAL_MACHINE\Comm\VNUWLC61\Parms]
  ''BusType"=dword:0
  ''BusNumber"=dword:0
  ''AdapterCFID"=dword:31841106
  ''NetworkAddress"=""
  ''AdapterType"=dword:0
  ''TransmitBuffers"=dword:00000020
  ''ReceiveBuffers"=dword:00000020
  ''ConnectionRate"=dword:0c
  ''Channel"=dword:06
  ''PreambleType"=dword:1
  ''RTSThreshold"=dword:092b
  ''FragThreshold"=dword:092a
  ''OperationMode"=dword:0
  ''BeaconInterval"=dword:64
  ''DesiredSSID"=""
  ''RadioControl"=dword:0
  ''AntennaDiversity"=dword:1
  ''IBSSMode"=dword:1
  ''SelectiveSuspend"=dword:0
  ''Roaming"=dword:1
  ''RoamingsignalStrength"=dword:3C
  ''RoamingInterval"=dword:5
  ''RoamingChannels"=""
```

```
    ''CountryRegion11BGBand"=dword:2
    ''CountryRegion11ABand"=dword:63

[HKEY_LOCAL_MACHINE\Comm\VNUWLC61\Parms\TcpIp]
    ''EnableDHCP"=dword:1
    ''DefaultGateway"=''"
    ''UseZeroBroadcast"=dword:0
    ''IpAddress"=''"
    ''Subnetmask"=''"
    ''DNS"=''"
    ''WINS"=''"
```

- Adiciona o arquivo DLL do *driver* na lista de módulos a serem carregados pelo *kernel*. Isso geralmente é feito em um arquivo com a extensão `BIB`:

```
VNUWLC6.dll  $(_FLATRELEASEDIR)\VNUWLC6.dll  NK SHK
```

- Copia os arquivos do *driver* para a pasta `\Windows` da imagem final. Esse procedimento é executado com scripts que são executados ao longo do processo de compilação.

4.3 Módulo EM-411

O SMGR possui como um de seus requisitos a necessidade de monitorar equipamentos situados em grandes canteiros de obras. Para que isso seja possível, é necessária a instalação de um módulo de GPS em cada uma das máquinas. Tal módulo deve possuir uma interface de comunicação padrão, baixo consumo de energia e ser compacto. Atendendo a tais características, além de preço e disponibilidade, a equipe optou por utilizar o EM-411, um módulo GPS fabricado pela empresa taiwandesa GlobalSat (GLOBALSAT TECHNOLOGY CORPORATION, 2009). No Brasil, não foram encontradas opções de venda do módulo em questão, ou outro modelo similar. Sendo assim, foi necessário localizar um distribuidor com opções de envio ao Brasil. A Futurlec, uma loja localizada na cidade de Nova Iorque, nos Estados Unidos, foi contactada e o módulo foi comprado.

O EM-411 foi conectado ao kit de desenvolvimento eBox 3310A através de sua porta serial RS-232. Como o EM-411 possui um pino de saída TX em nível TTL, deve-se criar um *driver* para comunicação via serial. Para essa tarefa, foi utilizado o MAX232 (TEXAS

INSTRUMENTS, 2004). O esquemático de circuito na Figura 34 mostra a interface do EM-411 com a porta serial através do MAX232.

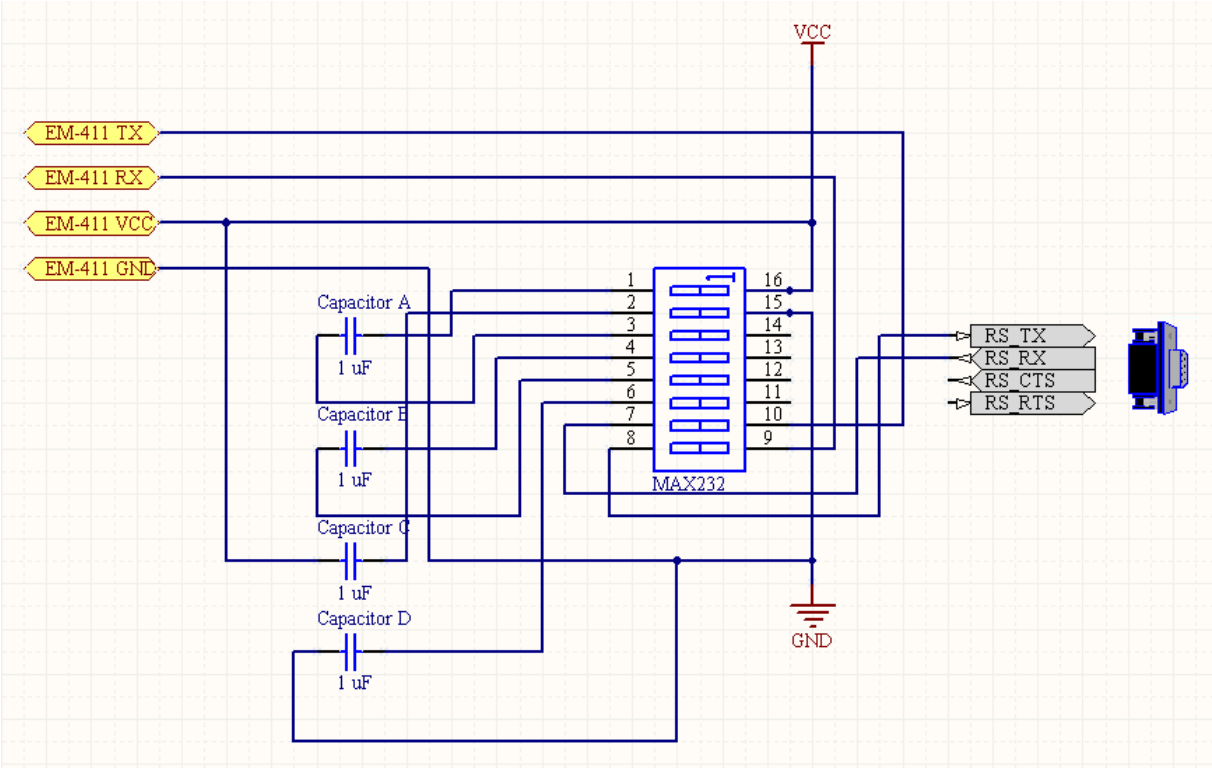

Figura 34: Esquemático de circuito do *driver* MAX-232 para o GPS

Constantemente, o EM-411 envia mensagens através da porta serial. O *software* do SMGR espera uma mensagem que inicia com `GPGGA`, que corresponde à mensagem que contém as coordenadas GPS, e ao recebê-la, faz a separação da latitude e longitude. Esses dados são salvos no registro como última posição encontrada e são enviados ao *Web Service*. Quando o usuário da interface web consultar informações sobre a máquina que enviou as coordenadas, os dados serão mostrados em conjunto com o Google Maps. Um mapa é mostrado contendo uma linha que liga as coordenadas enviadas pela máquina, com um marcador na última posição. Dessa forma, o gerente do sistema pode observar o trajeto percorrido e a localização atual daquele equipamento. Detalhes da API do Google Maps e do procedimento de captura de dados via serial serão tratados em seções específicas.

4.4 Módulo de Aquisição de Sinais

Para coletar dados dos sensores digitais e analógicos que serão utilizados em um equipamento de grandes obras, optou-se por um microcontrolador. Tais dispositivos atendem

aos requisitos de medição de sensores pois oferecem pinos específicos para a entrada de grandezas analógicas e digitais, possuem baixo custo e podem ser conectados aos demais módulos do sistema. Poderiam ser utilizadas outras soluções para o módulo de aquisição de sinais, como por exemplo o Advantech 4711A (Figura 35) (ADVANTECH, 2012). Trata-se de um módulo USB com diversas entradas para sensores e elevadas taxas de resolução, com *drivers* para Windows Embedded. Porém, devido ao seu elevado preço (em torno de mil e trezentos reais), foi descartado para uso no projeto.

Figura 35: Advantech 4711A
Fonte: Advantech (2012)

Há diversos modelos de microcontroladores no mercado, muitos com a capacidade de solucionar a questão de sensoriamento. Por disponibilidade e experiência da equipe, optou-se por utilizar o ATmega 328 (ATMEGA, 2012). Além disso, ferramentas de desenvolvimento para esse microcontrolador são disponibilizadas de forma gratuita, incluindo compiladores e *debuggers*.

Para o desenvolvimento do módulo de aquisição de sinais, será utilizado o microcontrolador ATmega 328P. Para conectá-lo ao kit de desenvolvimento eBox 3310A, não é possível fazer ligação direta com os pinos de UART disponíveis. Para isso, é necessário construir um *driver* de conversão para o padrão serial RS-232. Como solução, foi utilizado o MAX232 (Figura 36), um CI (circuito integrado) que fornece níveis de tensão corretos para comunicação RS-232 (TEXAS INSTRUMENTS, 2004).

O diagrama esquemático da Figura 37 mostra o *driver* construído para interfacear o módulo de aquisição com o eBox, através do padrão serial.

O microcontrolador envia seus dados para a porta serial através de uma comunicação com *baud rate* igual a 9700 pulsos por segundo. Os bytes enviados são lidos pelo *software* do SMGR, executando no Windows Embedded, para serem em seguida processados e

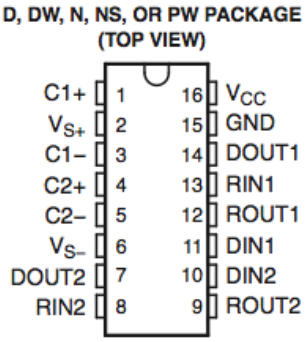

Figura 36: MAX232
Fonte: Texas Instruments (2004)

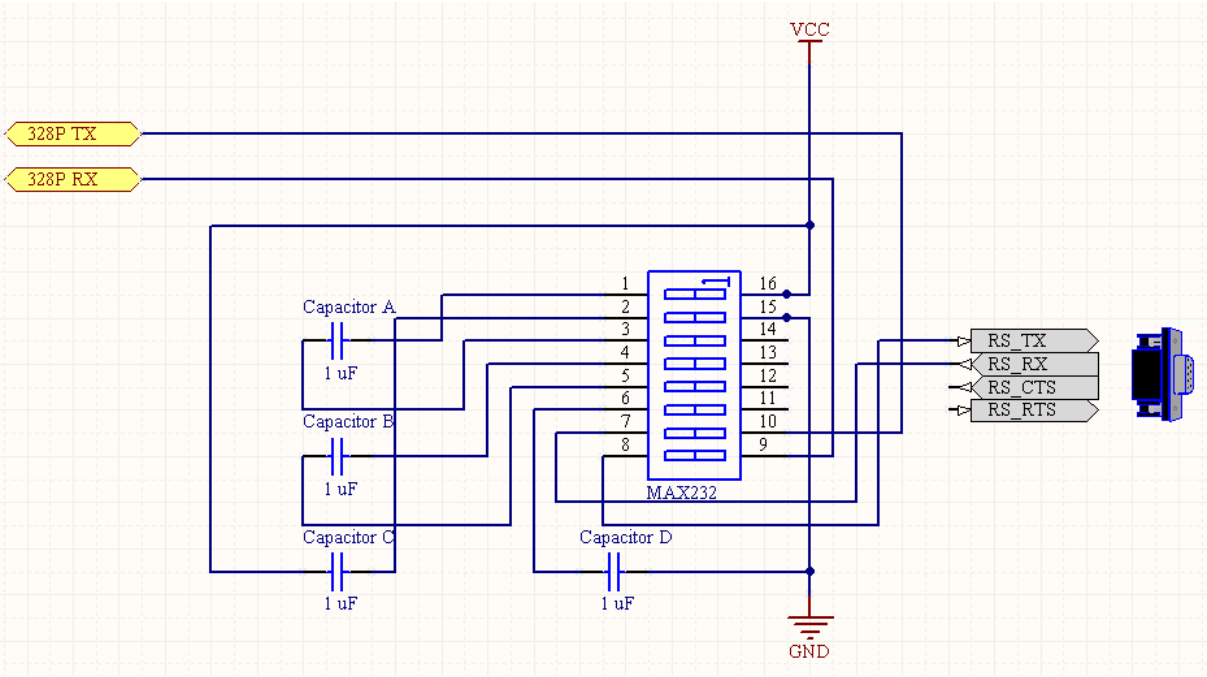

Figura 37: *Driver* Módulo de Aquisição

enviados ao servidor. Todos os dados são enviados na forma de um *array*, uma sequência numérica contendo os valores dos sensores digitais (0 ou 1) e dos analógicos (0 a 1023). Caso o usuário do SMGR tenha configurado o acompanhamento de algum desses sensores, o valor será inserido na base de dados e poderá ser visualizado na interface web.

Para demonstração do sistema, foram instalados dois sensores analógicos representados por potenciômetros e duas chaves para exemplificar os sensores digitais.

4.5 *Web Service*

Por ser interoperável, o *Web Service* pode ser acessado de diversas plataformas, permitindo maleabilidade no desenvolvimento do sistema de coleta de dados de maneira independente do restante do ecossistema.

Atualmente, existem diversas ferramentas que permitem agilizar o desenvolvimento de *Web Services*, cabendo ao programador somente desenvolver os métodos de que o sistema necessita, ficando a cargo das ferramentas gerar o *code-behind* que forma o *Web Service*, tornando seu desenvolvimento simples e transparente.

Neste trabalho foi utilizado o Visual Studio 2010 para desenvolvimento do *Web Service*, utilizando .NET Framework 4.0 para publicação do servidor que armazena o *Web Service*.

No lado do cliente do *Web Service*, o módulo que será instalado nos equipamentos a serem monitorados, foi utilizado .NET Compact Framework 2.0 para comunicação com o *Web Service*. Para isto, foi adicionada a referência ao endereço de publicação do *Web Service* e, com isso, todas as definições de métodos disponiveis via *web service* foram carregadas no código, permitindo seu uso no módulo.

Os métodos do *Web Service* (Figura 38) são responsáveis por receber as informações coletadas pelo módulo e salvá-los no banco de dados.

Para coletar tais informações foram criadas três operações no *Web Service*, `getClientAttributes`, que solicita do módulo um identificador e retorna ao módulo o valor de seu identificador e o intervalo entre o envio de informações para o *web service*. `InsertGPSMeasurements`, que solicita o identificador do módulo e os valores da latitude e longitude da coordenada GPS, e retorna -1 caso haja algum erro, ou 0 se insere os valores com sucesso. Similar ao anterior, há a operação `InsertMeasurements`, que solicita o identificador do módulo, e uma sequência de valores referente às medições do aquisitor de sinais, tanto digitais quanto analógicas. A quantidade de valores digitais e analógicos a serem inseridos dependem do *hardware* do módulo instalado.

Os dados coletados pelos módulos são armazenados em um banco de dados relacional SQL Server Express 2012.

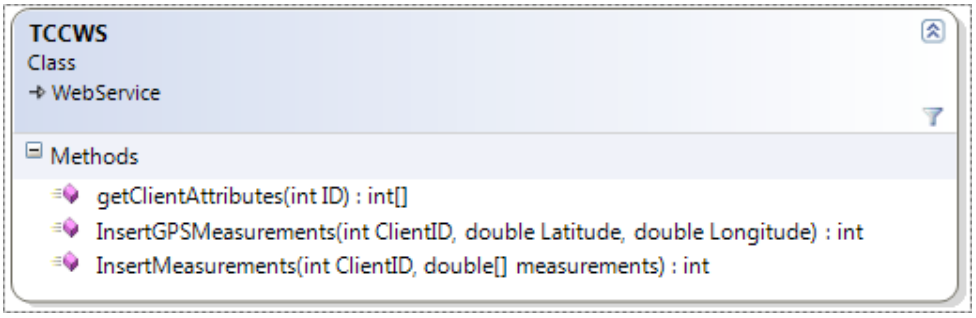

Figura 38: Diagrama das Operações do *Web Service*

4.6 *Software* para o Módulo Embarcado

Uma das partes do sistema embarcado, instalado em cada uma das máquinas do canteiro de obras, consiste no *software* embarcado. Esse *software* é responsável por capturar os dados dos sensores diversos, empacotá-los e enviá-los ao servidor do SMGR.

O Windows Embedded fornece ao programador poderosas ferramentas de desenvolvimento e debug, em um nível mais alto que código de máquina (Assembly) porém ainda assim com alto desempenho. Para se escrever o *software* do SMGR, foi utilizado o .NET Compact Framework 2.0, uma versão do popular .NET Framework, da Microsoft, feito especialmente para sistemas embarcados. O .NET Compact Framework não oferece todas as funções disponíveis em sua contraparte para o Windows convencional, porém fornece funções suficientes para a codificação desse projeto. É possível utilizar-se várias linguagens de programação com o .NET Framework, como por exemplo Visual Basic, C++ e C#. Para o SMGR, a linguagem escolhida foi C#. Para desenvolvimento foi utilizado o Microsoft Visual Studio 2005 Professional Edition, obtido pela equipe através do Microsoft DreamSpark, em convênio com a UTFPR, de forma gratuita.

O *software* do SMGR consiste em um *loop* que monitora as diversas entradas conectadas ao eBox 3310A. No início da aplicação, o SMGR verifica se existe no registro do Windows uma chave que informa o endereço do servidor. Caso essa chave exista, configura-se o web service com o endereço encontrado. Caso não exista, o SMGR trabalha em modo offline, em que irá apenas mostrar no console as informações lidas. Após esse *setup* inicial, o SMGR entra em um *loop*. No início do *loop*, tenta-se invocar o método `getClientAttributes` do servidor, passando como parâmetro o ID fornecido ao módulo embarcado. Caso esse ID ainda não exista, no caso da primeira execução, envia-se o valor -1 e o servidor retorna um ID válido, que deve ser salvo no registro para futuras conexões. Se essa conexão inicial envolvendo o ID falhar, o *software* trabalhará no mesmo modo offline descrito anteriormente. Ao final da chamada ao método, o servidor irá retornar

uma cópia do ID, de forma a confirmar a conexão, e o tempo em segundos entre o envio de cada pacote de informações, conforme configurado na interface web do SMGR.

Seguindo o fluxo do *loop*, primeiramente o SMGR tentará ler a porta serial COM2, em que está conectado o módulo de GPS EM-411 através do *driver* feito com o MAX232. Como podem ocorrer falhas na comunicação, são feitas 10 tentativas de leitura. No caso do GPS, busca-se uma linha começando com `$GPGGA`, que corresponde ao dado do GPS que indica as coordenadas atuais. Uma vez encontrada essa linha, faz-se a separação dos dados para extrair a latitude e a longitude, caso o módulo tenha conseguido conectar-se aos satélites. Se o módulo não conseguiu uma conexão com os satélites, nenhuma coordenada será obtida. Feita a extração da latitude e longitude, os dados são salvos no registro na forma de últimas coordenadas obtidas, e posteriormente enviados ao servidor. Há um método especial no web service para se postar os dados de GPS, chamado `InsertGPS-Measurements`. O cliente do SMGR invoca o método passando seu ID e as coordenadas obtidas.

O próximo passo do *loop* é obter os dados dos sensores, ligados ao módulo de aquisição de sinais feito com o microcontrolador ATmega 328P. De forma similar, o SMGR busca uma linha de texto vinda da porta serial COM1 começando com o valor hexadecimal `0x41`. Esse valor foi estipulado pela equipe e sinaliza ao *software* que a linha recebida contém informações de sensores. Uma vez identificado o valor 0x41, serão lidos 17 valores separados por vírgula. Os 11 primeiros correspondem aos valores dos sensores digitais, e os 6 seguintes correspondem às medidas dos sensores analógicos. De posse dos valores, o SMGR invoca o método `InsertMeasurements` do web service, informando seu ID e o vetor contendo os 17 valores.

Ao final do *loop*, o SMGR irá aguardar o tempo configurado inicialmente até repetir o processo e fazer um novo envio de informações. Vale ressaltar que o valor de tempo pode ter sido alterado ao longo da operação, e será atualizado no início do *loop* quando o método `getClientAttributes` for novamente chamado.

4.7 Aplicação Web

O SMGR consiste em um sistema baseado na arquitetura cliente/servidor, em que os clientes coletam dados de sensores e os enviam ao servidor central para inserção em seu banco de dados. Sem algum tipo de interface para visualização dos dados, seria necessário a execução de *queries* SQL diretamente no banco para a extração das informações, o que

não seria adequado. Logo, é necessária a criação de uma interface, de forma que os gestores tenham fácil acesso a todas as informações enviadas pelos clientes, além de poderem cadastrar informações de forma manual que não são possíveis de se obter automaticamente. Decidiu-se por uma interface via web, pois permite o acesso ao sistema a partir de qualquer dispositivo conectado à Internet.

A aplicação web do SMGR foi desenvolvida no Microsoft Visual Studio 2010 Professional, obtido através do DreamSpark, convênio da UTFPR e Microsoft para disponibilizar *software* de forma gratuita aos estudantes. A programação foi feita em ASP.NET (*Active Server Pages*), tecnologia sucessora do ASP. De forma geral, uma página em ASP.NET é conhecida como Web Form, e possui um código executado no servidor que é responsável por gerar conteúdo de forma dinâmica e apresentá-lo ao usuário final. Para executar a aplicação web, também é necessário utilizar um servidor web, que processa os *web forms* e os disponibiliza para visualização em *browsers*, usando o protocolo HTTP. Por possuir integração total com o ASP.NET, foi utilizado o servidor web Microsoft IIS (*Internet Information Services*).

A aplicação web foi separada em dois componentes principais: um DAO (*Data Access Object*) e uma *Web Service*. O DAO é responsável por criar uma interface abstrata para acesso ao banco de dados, enquanto o *Web Service* é a interface pública para que os clientes postem informações no banco de dados.

O DAO possui 6 classes principais que facilitam o acesso aos dados:

- `ClientAccessLayer`: camada de interface com a tabela `Client`. Através do `ClientAccessLayer`, é possível enumerar todos os clientes do sistema, criar um novo cliente com sensores (*managed*), criar um novo cliente sem sensores (*unmanaged*), encontrar um cliente dado seu ID, atualizar um cliente e remover um cliente.

- `DashboardAccessLayer`: camada de interface que fornece dados para o *Dashboard* do SMGR. Trata-se de um resumo do sistema que é apresentado na página inicial, e contém dados tais como o número de dispositivos cadastrados, além do número total de sensores, propriedades e coordenadas GPS.

- `GPSMeasurementAccessLayer`: camada de interface com a tabela de medições de GPS. Fornece métodos para retornar todas as medições de um determinado cliente, retornar a última medição de um cliente, deletar uma medição e deletar um intervalo de medições entre uma data inicial e final.

- `MeasurementAccessLayer`: camada de interface com a tabela de medidas dos sensores. Permite consultar as medidas de um determinado cliente em uma determinada porta (índice do sensor conectado ao microcontrolador), adicionar uma única medida, adicionar várias medidas ao mesmo tempo, e encontrar a última medida de um determinado cliente.

- `PropertyAccessLayer`: camada de interface com a tabela de propriedades. Permite listar, adicionar, editar e remover uma propriedade, assim como encontrar uma propriedade dado seu ID.

- `SensorAccessLayer`: camada de interface com a tabela de sensores. Permite listar todos os sensores de um cliente, listar apenas sensores digitais, apenas sensores analógicos, inserir, editar e remover um sensor, e obter um sensor dado seu ID.

O DAO mapeia o banco de dados em objetos através de um arquivo do tipo DBML (*Database Management Language*), que utiliza uma string de conexão com o banco e cria classes para cada tabela encontrada. Dessa forma, pode-se manipular tabelas através de propriedades de objetos.

O *Web Service* do SMGR é responsável por fornecer uma interface de acesso aos clientes e apresentar os dados na forma de uma página web, publicada na Internet. O web service fornece três métodos aos clientes:

- `GetClientAttributes`: cria um ID para o cliente, caso seja o primeiro acesso, ou retorna o ID e o tempo de envio entre informações de sensores e GPS.

- `InsertMeasurements`: dado o ID de um cliente, insere no banco o vetor de valores dos sensores analógicos e digitais.

- `InsertGPSMeasurements`: idem ao método anterior, porém insere coordenadas do GPS.

  A página web que exibe as informações é dividida em uma série de seções. O passo inicial para se obter acesso é fazer um *login*, que pode ser de dois tipos:

- Usuário comum: pode apenas visualizar as informações, sem nenhuma permissão de alteração.

- Usuário administrador: pode visualizar, inserir, modificar e excluir informações, além de criar novos usuários comuns ou administradores.

Feito o *login*, o usuário tem acesso a uma página inicial contendo o *Dashboard*, que mostra informações gerais sobre o sistema e links para acesso às demais seções. No link de Dispositivos o usuário pode inserir um dispositivo não gerenciado (*unmanaged*), que contém apenas propriedades, sem sensores e GPS. Essa função é útil para dispositivos mais simples que não são monitorados de forma automática, mas ainda assim podem ser cadastrados para fins de gestão.

É também possível acessar um dispositivo específico, para ver um mapa contendo seus pontos de GPS, as últimas medidas dos sensores, e as propriedades. Pode-se configurar os sensores e as propriedades, adicionando ou removendo-as conforme necessário. Um dispositivo pode ser renomeado ou removido. É importante renomear um dispositivo gerenciado, pois esse tipo de dispositivo é cadastrado automaticamente e recebe como nome a data de inserção. A interface do dispositivo mostrará somente informações de sensores que foram cadastrados, portanto deve-se acessar a página de Sensores para fazê-lo. Um sensor pode ser cadastrado como analógico ou digital, possui um índice no microcontrolador e uma unidade de medida.

Através do *dashboard* também é possível acessar a página de Mapas, que mostra as coordenadas de todos os dispositivos monitorados do sistema. Isso permite uma visão geral, através da API do Google Maps.

Há também a opção de Usuários, onde administradores podem visualizar e cadastrar novos usuários.

4.7.1 API Google Maps

Segundo Google (2012a), o Google Maps provê várias APIs que permitem ao desenvolvedor incorporar as funcionalidades do Google Maps em seus aplicativos e sites, bem como adicionar conteúdo customizado sobre os mapas. A API do Google Maps é um serviço gratuito e disponibilizado para qualquer *site* desde que não sejam excedidos os limites de uso. Para um site comercial, é permitido que o mapa seja carregado 25 000 vezes por dia utilizando a API Google Maps Javascript V3. Os acessos que excederem esses limites serão taxados (GOOGLE, 2012b).

4.7.1.1 API Google Maps Javascript V3

A API do Google Maps Javascript permite ao desenvolvedor adicionar as funcionalidades do Google Maps em suas páginas web. Esta API provê vários utilitários para a

manipulação dos mapas e adição de conteúdo, tornando possível a criação de aplicativos robustos em sites. Na versão 3 desta API, ocorreram várias alterações para que ela seja mais rápida e mais compatível com navegadores atuais e dispositivos móveis (GOOGLE, 2012a).

4.7.1.2 Reimers Google Maps .NET Control

A aplicação web desenvolvida neste trabalho foi implementada utilizando a tecnologia ASP.NET. Para facilitar a integração entre as páginas web e a API do Google Maps, decidiu-se utilizar um controle .NET que encapsulasse as funcionalidades desta API. O Reimers Google Maps .NET Control é um controle .NET gratuito e open source que segue a API do Google Maps V3 e provê as funcionalidades essenciais da API do Google, incluindo geocoding (REIMERS, 2012).

4.8 Considerações sobre o capítulo

Este capítulo apresentou os processos utilizados na etapa de desenvolvimento do SMGR. A primeira parte tratou da configuração do Windows Embedded, em que selecionaram-se os componentes necessários para o funcionamento do SMGR no kit de desenvolvimento eBox. Com o sistema configurado, foi criado o SDK customizado para uso no desenvolvimento do *software* embarcado. A seguir, o eBox 3310A foi preparado para *boot* do sistema operacional recém-criado. *Drivers* para o módulo Wi-Fi™ foram adicionados no RTOS da Microsoft, e uma placa foi projetada para abrigar o módulo GPS e de aquisição de sinais, juntamente com um *driver* para comunicação serial. O web service foi programado, criando-se métodos para que os clientes tenham acesso ao banco de dados do servidor. O *software* embarcado foi escrito e teve seu funcionamento testado juntamente com o web service. A aplicação web foi programada, incluindo as camadas de acesso a banco e a interface em si. Por fim, a API do Google Maps foi utilizada para visualização dos dados de GPS, através do componente Reimers Google Maps Control, específico para a plataforma .NET.

5 RESULTADOS

Este capítulo apresenta os resultados obtidos, detalhando o estado atual do módulo embarcado e a aplicação web.

5.1 Módulo Embarcado

Um dos produtos resultantes deste trabalho é o Módulo Embarcado SMGR. As Figuras 39 e 40 apresentam o esquemático e o desenho da placa confeccionada respectivamente.

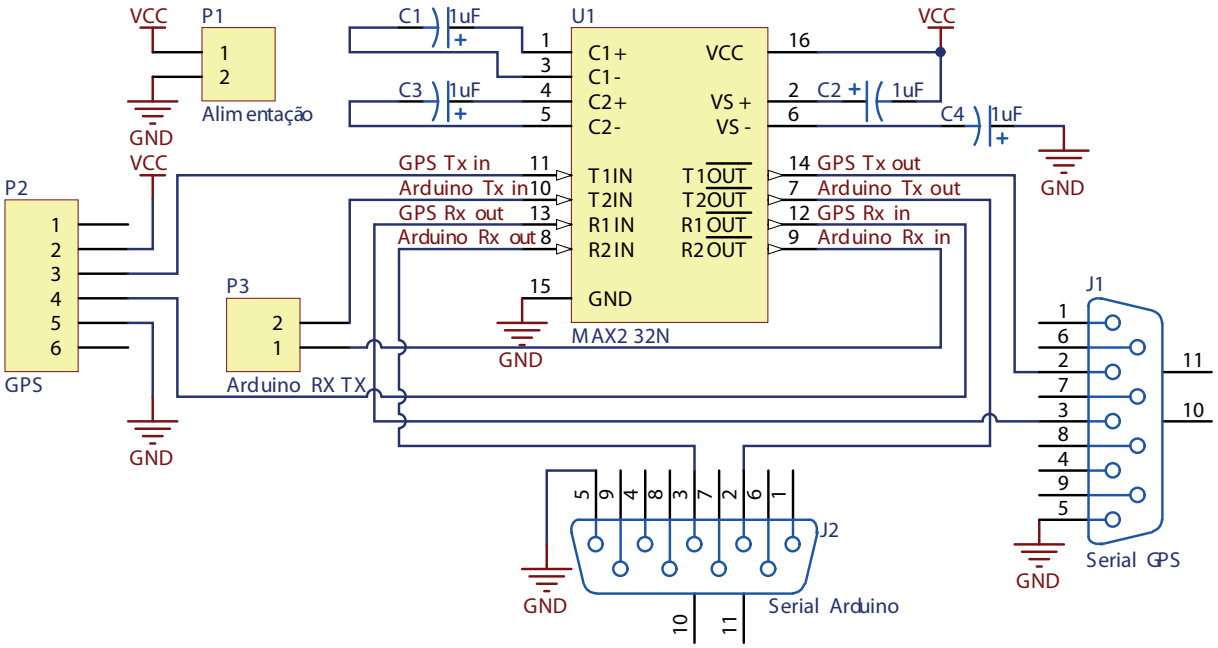

Figura 39: Esquemático da Placa

Após confeccionada a placa de circuito impresso, os componentes foram montados e soldados (Figura 41). A placa, então, foi fixada na eBox tornando-se assim um produto compacto (Figura 42).

Para proteção dos componentes eletrônicos e por questões estéticas, o conjunto produzido (placa e eBox) foi acomodado em uma caixa plástica como pode ser observado nas

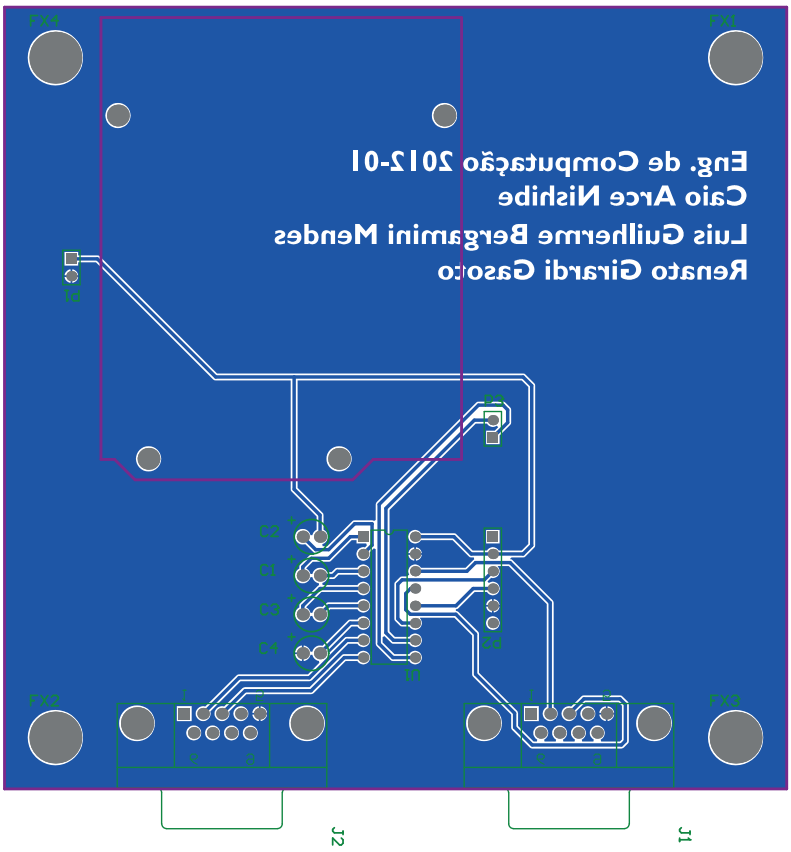

Figura 40: Desenho da Placa

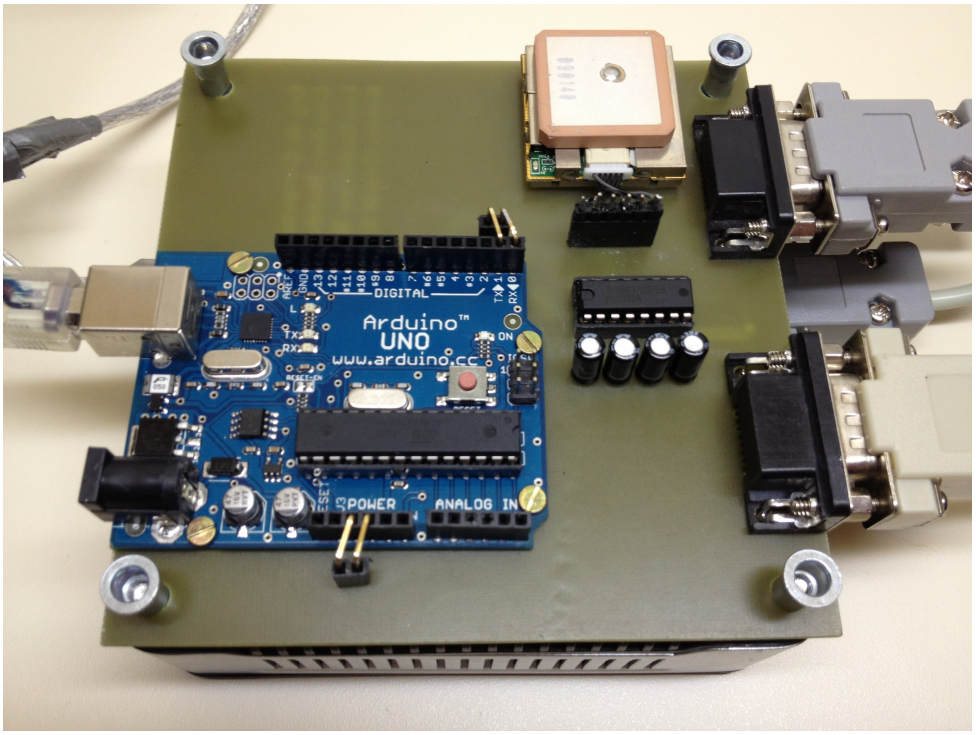

Figura 41: Visão Superior da Placa

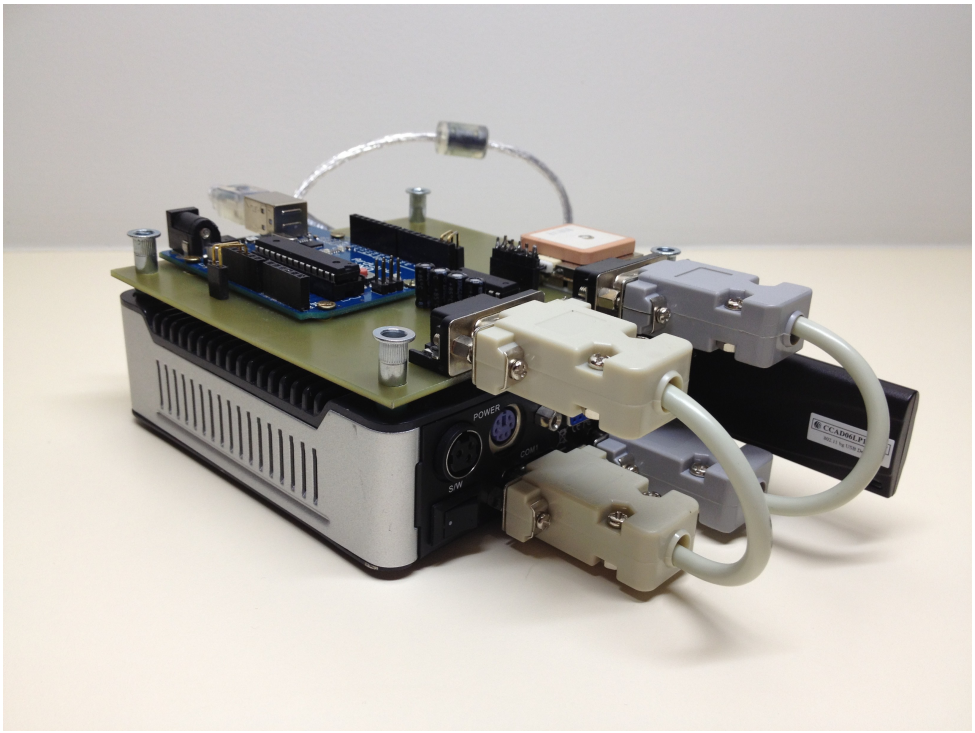

Figura 42: Visão Lateral da Placa Conectada ao eBox

Figuras 43 e 44.

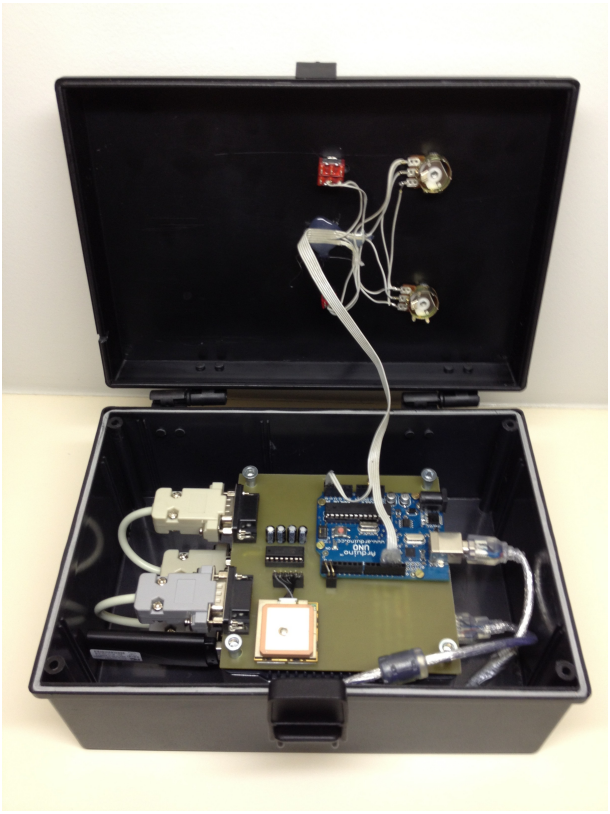

Figura 43: Módulo Embarcado SMGR - Visão Interna

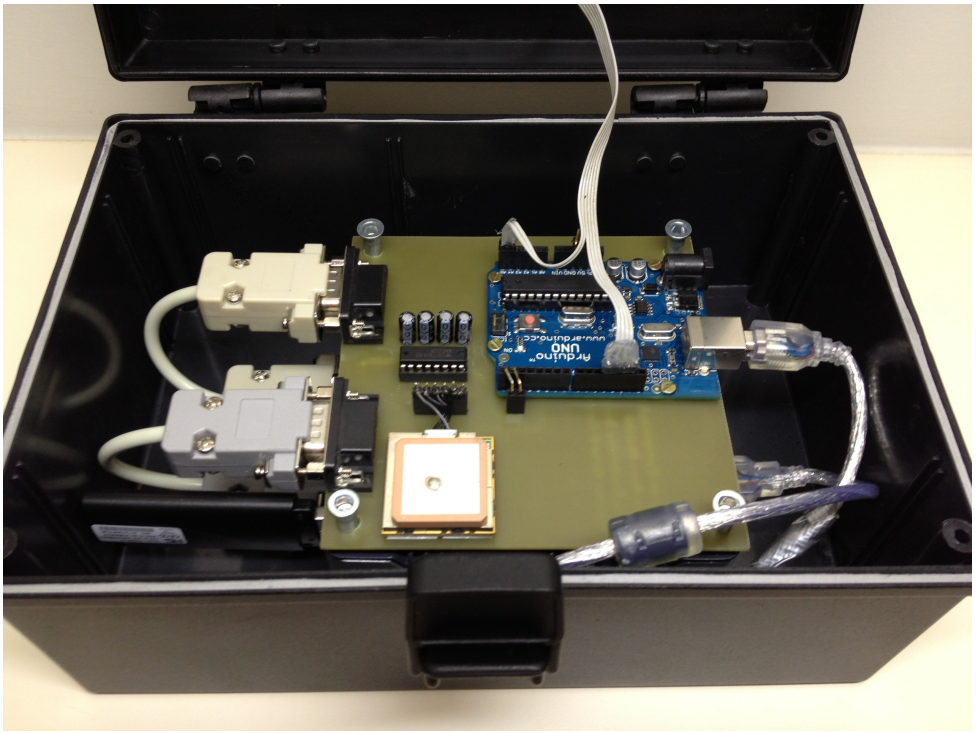

Figura 44: Detalhe Módulo Embarcado SMGR - Visão Interna

A Figura 45 mostra o Módulo Embarcado SMGR. Foram instalados dois potenciômetros e duas chaves para simular sensores analógicos e digitais que podem ser conectados com o módulo.

Foram efetuadas rotinas de testes de *hardware* e *software* para garantir que o Módulo Embarcado SMGR cumpre os requisitos propostos e que seja suficientemente robusto para ser instalado em uma máquina que opera em um canteiro de obras.

5.2 Aplicação Web

A aplicação web possui uma interface gráfica que segue a *Microsoft Design Languague*, antes conhecida como *design* Metro. As Figuras 46, 47, 48, 49, 50 e 51 apresentam alguns exemplos da aparência das páginas web concebidas.

Figura 45: Módulo Embarcado SMGR

Figura 46: Página de Login

Figura 47: Dashboard

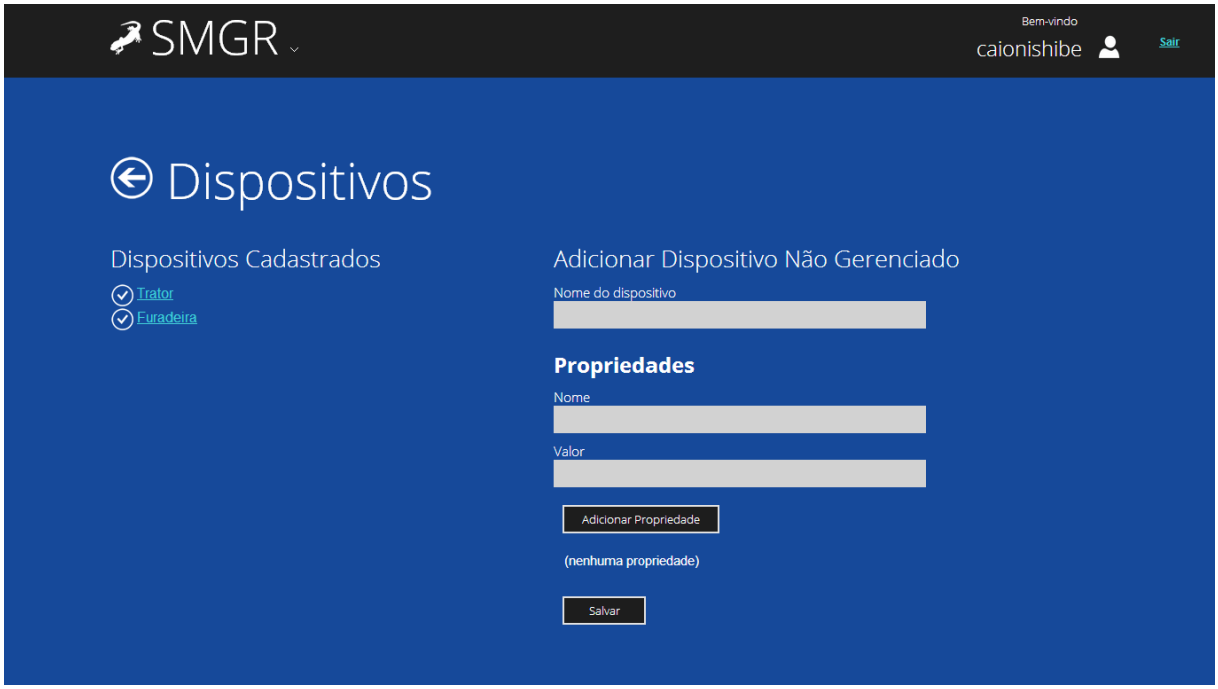

Figura 48: Página de Dispositivos

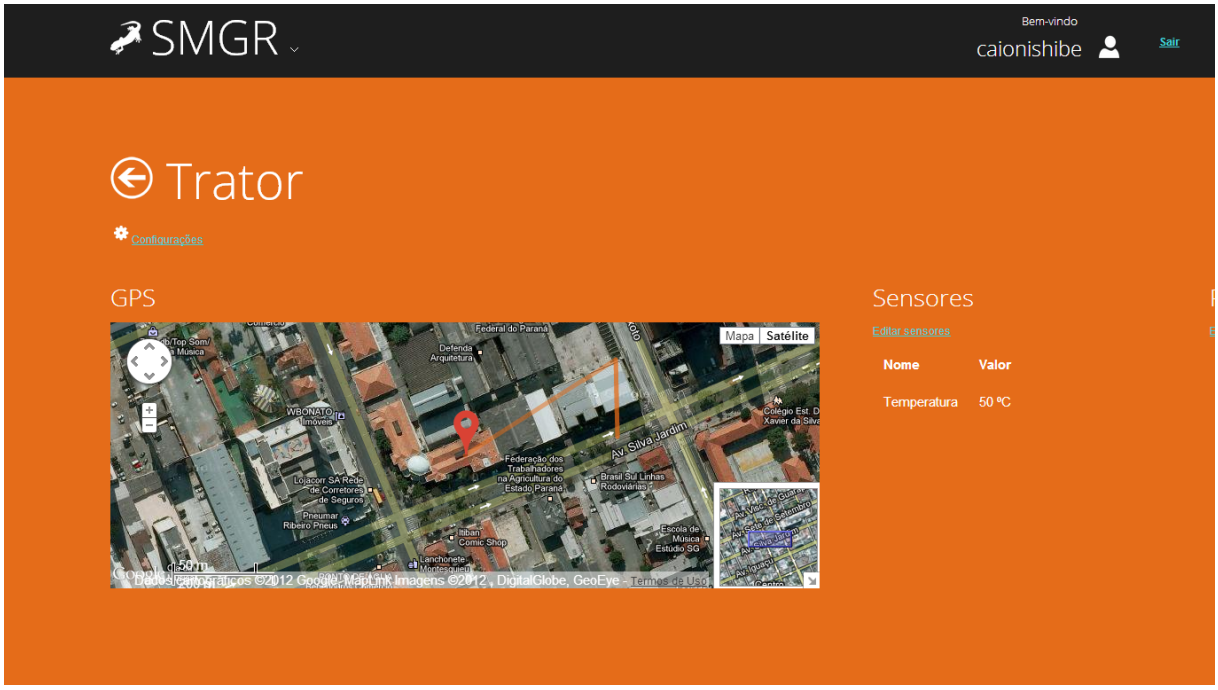

Figura 49: Página de Visualização da Posição, Sensores e Propriedades de um Dispositivo

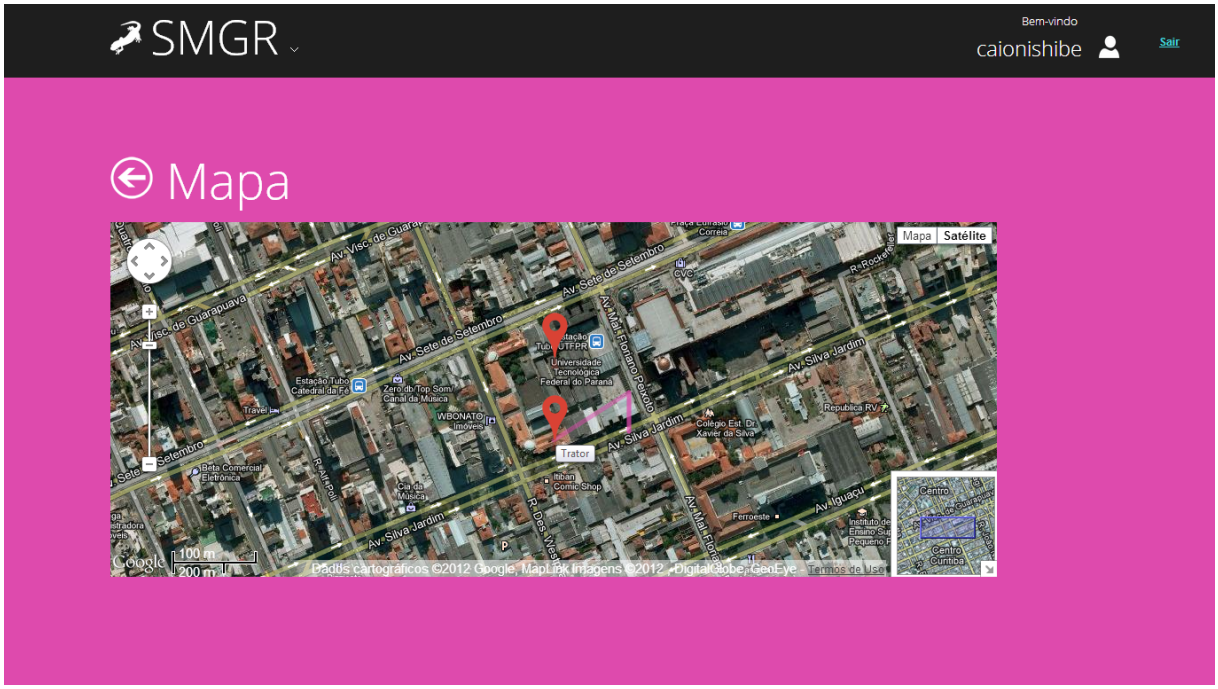

Figura 50: Página do Mapa

Figura 51: Página de Usuários

6 CONCLUSÕES

Sabe-se que o monitoramento e gestão remotos de maquinário são tarefas importantes, porém a indústria sempre os considerou como não críticos. Com o surgimento de tecnologias *wireless* e em especial a Internet, atualmente está havendo uma mudança de paradigma. Empresas estão percebendo que podem ser mais eficientes, produtivas, fornecer maior segurança a seus funcionários e, consequentemente, ser mais lucrativa com o uso de técnicas de monitoramento remoto e automático. Através desses métodos, pode-se focar em analisar dados e traçar planos de desenvolvimento e crescimento ao invés de dedicar tempo e dinheiro na coleta manual de informações.

Esse trabalho de conclusão de curso propôs-se a conceber um sistema de gestão e monitoramento remotos de maquinário em canteiros de grandes obras. Isso incluiu as tarefas de se projetar um sistema embarcado capaz de enviar dados de sensores a um servidor central através de uma rede Wi-Fi™ de padrão industrial e implementar uma arquitetura em nuvem para acesso aos dados coletados. Para isso, a equipe fez uma revisão da literatura disponível sobre os assuntos abordados, adquiriu componentes de *hardware* e *software* necessários para a construção de um protótipo, implementou o sistema embarcado e a aplicação web propostos, aplicou testes de *hardware* e *software* para validar o funcionamento do sistema, e por fim fez a análise dos resultados.

Primeiramente buscou-se conhecimento sobre sistemas embarcados, que são aqueles que envolvem *hardware*, *software* e mecânica dedicados a uma tarefa específica. Após isso estudou-se sobre sistemas operacionais de tempo real (RTOS), que, ao contrário do senso comum, não executam tarefas de forma rápida, mas sim com garantias de se cumprir um prazo máximo ou até mesmo um tempo determinístico. Em seguida a equipe estudou o Windows Embedded, um RTOS da Microsoft validado pela indústria e disponível de forma gratuita, através de convênio com a UTFPR, para suportar o desenvolvimento do *software* embarcado. Uma plataforma de *hardware*, compatível com o Windows Embedded, foi estudada e adaptada para esse trabalho. Foram adquiridos módulos para comunicação *wireless* via Wi-Fi™, GPS e aquisição de sinais, de forma a atender todos os

requisitos do projeto. Buscou-se também material sobre bancos de dados, web services e o .NET Framework, para fornecer embasamento durante o desenvolvimento da aplicação web utilizada para armazenar e exibir as informações coletas em campo.

De posse da fundamentação teórica necessária, a equipe partiu para a especificação da solução proposta utilizando a metodologia necessária. Primeiramente foi feita a especificação do sistema, envolvendo análise de requisitos e o estudo de cada caso de uso. Após isso, cada módulo foi individualmente especificado, começando pelo *hardware* e RTOS, passando para o *software* embarcado, em seguida todas as tabelas do banco de dados, até finalmente o fluxo da interface web.

Com o sistema totalmente especificado, partiu-se para sua implementação. Uma imagem do Windows Embedded foi customizada para a plataforma de *hardware*; um *software* embarcado foi escrito utilizando-se o .NET Framework, de forma a coletar as informações de GPS e sensores para depois enviá-los ao servidor; um web service foi projetado, de forma a servir de interface para que os clientes pudessem inserir as informações no banco através da rede Wi-Fi™ local ou até mesmo via Internet; a interface de acesso aos dados foi escrita em ASP.NET, fornecendo fácil acesso às informações e configuração do sistema.

Ao final do desenvolvimento, obteve-se como resultado um protótipo do sistema proposto inicialmente, cumprindo os requisitos e as especificações. Foi projetada uma placa para abrigar o GPS e o módulo de aquisição de sinais, juntamente com um *driver* para comunicação via serial. O *software* embarcado teve sucesso ao ler as informações enviadas, assim como em postá-las no servidor segundo o protocolo do web service. A interface web foi posta em funcionamento e pode ser utilizada para visualizar as informações e configurar o sistema de acordo com as necessidades do gestor. O sistema foi validado segundo testes em rede local e via Internet, confirmando o funcionamento de cada módulo.

Com um projeto desse tipo, a indústria pode de fato se benificar e ser mais eficiente. As novas tecnologias de comunicação sem fio estão se difundindo cada dia mais, e com custos progressivamente menores. Essa mudança de paradigma já está sendo observada em áreas agrícolas, porém a equipe não encontrou soluções similares na área de construção.

Ao longo desse trabalho de conclusão de curso, diversas dificuldades foram encontradas. O módulo GPS, assim como o adaptador Wi-Fi™ compatível com o Windows Embedded, não foram encontrados em lojas nacionais, portanto tiveram que ser adquiridos através de importação. O EM-411 apresentou dificuldades em obter sinal dos satélites, devido a sua limitada antena interna. O Windows Embedded, inicialmente escolhido em sua versão 7 para uso no projeto, teve que ser substituído pela versão anterior devido

a problemas de incompatibilidade com o módulo Wi-Fi™ Wyse VT6656. O módulo embarcado necessitou ser testado em uma rede distinta da local e postar os dados através da Internet, de forma a simular as distâncias encontradas em grandes canteiros de obras. Uma rede doméstica simples, conforme utilizada nos testes iniciais da equipe, não era suficiente para comprovar a flexibilidade do sistema.

Há pontos para a continuidade deste trabalho, como inserção de novos casos de uso na interface web, por exemplo visualização do histórico de dados de cada dispositivo, sumarização destes em um relatório. No módulo embarcado, é necessária a substituição do Arduino por um sistema de coleta de dados mais robusto e apropriado para o ambiente industrial.

Em conclusão, desenvolver esse projeto, batizado de SMGR, foi uma tarefa gratificante para a equipe e que envolveu diversos conceitos estudados durante o curso de Engenharia de Computação: programação, bancos de dados, estruturas de dados, algoritmos, eletrônica, sistemas embarcados, desenvolvimento para ambientes web, engenharia de *software*, redes. Todas as áreas puderam ser integradas em um projeto com impacto real para a indústria, potencialmente trazendo muitos benefícios a toda a área de construção civil.

REFERÊNCIAS

ADVANTECH. *Advantech USB 4711A DAQ Module*. 2012. Disponível em: <http://www.advantech.com.br/products/USB-4711A/mod_16CB666D-62EE-4C15-BF5F-EA49F61093CF.aspx>.

AMAZON. *Wyse Wyse VT6656 802.11b/g Wireless USB LAN Network Adapter*. 2012. Disponível em: <http://ecx.images-amazon.com/images/I/21z63JLDSBL.jpg>.

ARDUINO. *Arduino Uno*. 2012. Disponível em: <http://arduino.cc/en/Main/ArduinoBoardUno>.

ATMEGA. *8-bit Microcontroller with In-System Programmable Flash ATmega 328*. 2012. Disponível em: <http://www.atmel.com/Images/doc8161.pdf>.

BARR, M.; MASSA, A. *Programming Embedded Systems*. Sebastopol: O'Reilly, 1999.

BAXTER, N.; JESUS, H. D. *Remote Machine Monitoring: A Developing Industry*. 2006. Disponível em: <http://rtcmagazine.com/articles/view/100396>.

BERNERS-LEE, T. *Web Services: Program Integration across Application and Organization boundaries*. 2009. Disponível em: <http://www.w3.org/DesignIssues/WebServices.html>.

BLUETOOTH CONTRIBUTORS GROUP. *Bluetooth Master Table of Contents & Compliance Requirements*. 2010. Disponível em: <http://bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=229737>.

BOOTH, D.; LIU, C. K. *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*. 2007. Disponível em: <http://www.w3.org/TR/wsdl20-primer/>.

BRAY, T. et al. *Extensible Markup Language (XML) 1.0*. 2008. Disponível em: <http://www.w3.org/TR/xml/>.

CHRISTENSEN, E. et al. *Web Services Description Language (WSDL) 1.0*. 2000. Disponível em: <http://xml.coverpages.org/wsdl20000929.html>.

CHRISTENSEN, E. et al. *Web Services Description Language (WSDL) 1.1*. 2001. Disponível em: <http://www.w3.org/TR/wsdl>.

CODD, E. (Ed.). *A Relational Model of Data for Large Shared Data Banks*. 1970.

CREATIVE COMMONS. *Creative Commons Attribution-ShareAlike 2.5 Generic (CC BY-SA 2.5)*. 2012. Disponível em: <http://creativecommons.org/licenses/by-sa/2.5/>.

DEDICATED SYSTEMS. *RTOS Evaluation Program (Windows Embedded CE6.0 R2 on an x86 platform)*. 2009. Disponível em: <http://download.dedicated-systems.com/>.

DEDICATED SYSTEMS EXPERTS. *RTOS Evaluation Program (Windows Embedded CE6.0 R2 on an x86 platform)*. 2009.

DELL. *Dell Announces Intent to Acquire Wyse Technology*. 2012. Disponível em: <http://content.dell.com/us/en/corp/d/secure/2012-04-02-dell-acquisition-wyse-technology.aspx>.

DOTGNU PORTABLE.NET. *DotGNU Portable.NET*. 2012. Disponível em: <http://www.gnu.org/software/dotgnu/pnet.html>.

ECMA INTERNATIONAL. *Common Language Infrastructure (CLI) Partitions I to VI*. 2012.

EL-RABBANY, A. *Introduction to GPS - The Global Positioning System*. [S.l.]: Artech House, 2002.

FAKHROUTDINOV, K. *UML Diagrams*. 2012. Disponível em: <http://www.uml-diagrams.org/>.

GANSSLE, J. *What processor is in your product?* 2006. Disponível em: <http://www.embedded.com>.

GLOBALSAT TECHNOLOGY CORPORATION. *GPS Receiver Engine Board EM-411 Product User Manual*. [S.l.], 2009.

GODSE, A. P.; GODSE, D. A. *Microcontrollers*. [S.l.]: Technical Publications Pune, 2008.

GOOGLE. *Google Maps API*. 2012. Disponível em: <https://developers.google.com/maps>.

GOOGLE. *Google Maps Javascript API V3*. 2012. Disponível em: <https://developers.google.com/maps/documentation/javascript/usage>.

GUDGIN, M. et al. *SOAP Version 1.2 Part 1: Messaging Framework*. 2007. Disponível em: <http://www.w3.org/TR/soap12-part1/>.

HEUSER, C. A. *Projeto de Banco de Dados*. Porto Alegre: Bookman, 2009.

IEEE COMPUTER SOCIETY. *IEEE Standard 802.15.4d*. 2009.

KAMAL, R. *Microcontrollers Architecture: Programming, Interfacing and System Design*. [S.l.]: Pearson Education, 2007.

KENNETT, K. *Windows Embedded CE DiskPrep PowerToy*. 2010. Disponível em: <http://archive.msdn.microsoft.com/DiskPrep>.

MICROSOFT. *An Introduction to P/Invoke and Marshaling on the Microsoft .NET Compact Framework*. 2003. Disponível em: <http://msdn.microsoft.com/en-us/library/aa446536.aspx>.

MICROSOFT. *Common Language Runtime (CLR)*. 2012. Disponível em: <http://msdn.microsoft.com/en-us/library/8bs2ecf4.aspx>.

MICROSOFT. *Cross-Language Interoperability*. 2012. Disponível em: <http://msdn.microsoft.com/en-us/library/a2c7tshk.aspx>.

MICROSOFT. *Microsoft Windows Embedded Website*. 2012. Disponível em: <http://www.microsoft.com/windowsembedded/en-us/windows-embedded.aspx>.

MICROSOFT. *.NET Framework*. 2012. Disponível em: <http://msdn.microsoft.com/en-us/vstudio/aa496123.aspx>.

MONO. *Cross platform, open source .NET development framework*. 2012. Disponível em: <http://www.mono-project.com/Main_Page>.

NATIONAL MARINE ELECTRONICS ASSOCIATION. *NMEA 0183 Standard*. 2012. Disponível em: <http://www.nmea.org/content/nmea_standards/nmea_0183_v_410.asp>.

PAVLOV, S.; BELEVSKY, P. *Windows Embedded CE 6.0 Fundamentals*. [S.l.]: Microsoft Learning, 2008.

PHUNG, S. *Professional Windows Embedded CE 6.0*. [S.l.]: Wrox Programmer to Programmer, 2008.

PHUNG, S. *eBox 3300 MSJK Windows Embedded CE 6.0 R3 JumpStart Guide*. [S.l.]: ICOP Technology Inc, 2010.

QNX. 2012. Disponível em: <http://www.qnx.com/products/neutrino-rtos/index.html>.

REIMERS, J. *Google Maps .NET Control*. 2012. Disponível em: <http://www.reimers.dk/>.

RENAUX, D. P. B. Design of an object-oriented operating system for embedded systems. 2005.

SOMMERVILLE, I. *Engenharia de Software*. 8. ed. São Paulo: Pearson Addison Wesley, 2007.

STADZISZ, P. C.; RENAUX, D. P. B. *Software Embarcado*. 2008. Disponível em: <http://www.pessoal.utfpr.edu.br/douglasrenaux/Stad_Renaux_Software_Embarcado.pdf>.

TANENBAUM, A. S. *Sistemas Operacionais Modernos*. 3. ed. São Paulo: Pearson Prentice Hall, 2009.

TEXAS INSTRUMENTS. *MAX232 Dual EIA-242 Drivers/Receivers*. [S.l.], 2004.

TI-AGRO. *Módulo Máquinas*. 2012. Disponível em: <http://www.agrisoft.com.br/index.php?option=com_content&view=article&id=77&Itemid=192>.

TUCKER, N. *Using PDAs and Wireless Technology in Remote Machine Monitoring*. 2005. Disponível em: <http://rtcmagazine.com/articles/view/100396>.

TURLEY, J. *Operating Systems on the Rise*. 2006. Disponível em: <http://www.embedded.com>.

VELOSO, N. *Gerenciamento e Manutenção de Equipamentos Móveis*. 1. ed. São Paulo: Sobratema, 2009.

VIA NETWORKING TECHNOLOGIES. *VIA Solomon VT6656 - 802.11a/b/g Wireless LAN Controller with Integrated Baseband Processor-USB 2.0 bus interface*. 2012. Disponível em: <`http://www.via.com.tw/en/products/networking/wireless/vt6656/`>.

VIA TECHNOLOGIES. *VIA Networking VT6656 USB-Wireless LAN Adapter NDIS Driver for Windows CE 6.x*. 2007. Disponível em: <http://www.via.com.tw/en/support/drivers.jsp>.

WIND RIVERS. 2012. Disponível em: <`http://www.windriver.com/products/vxworks/`>.

ANEXO A – DIAGRAMAS MÓDULO EMBARCADO

Esta seção apresenta os demais diagramas produzidos durante a fase de especificação do *software* do módulo embarcado SMGR.

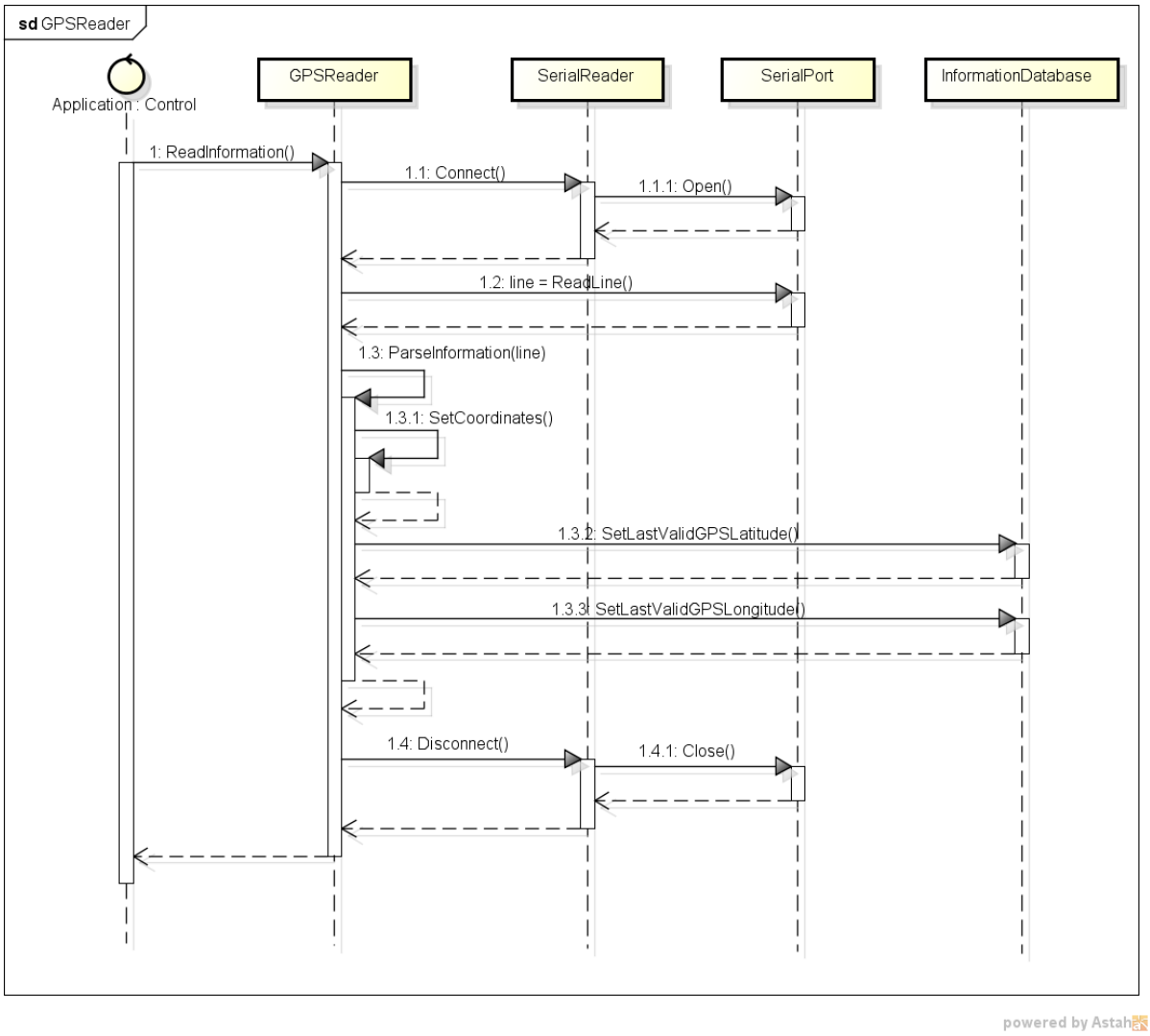

Figura 52: Diagrama de Sequência da Classe `GPSReader`

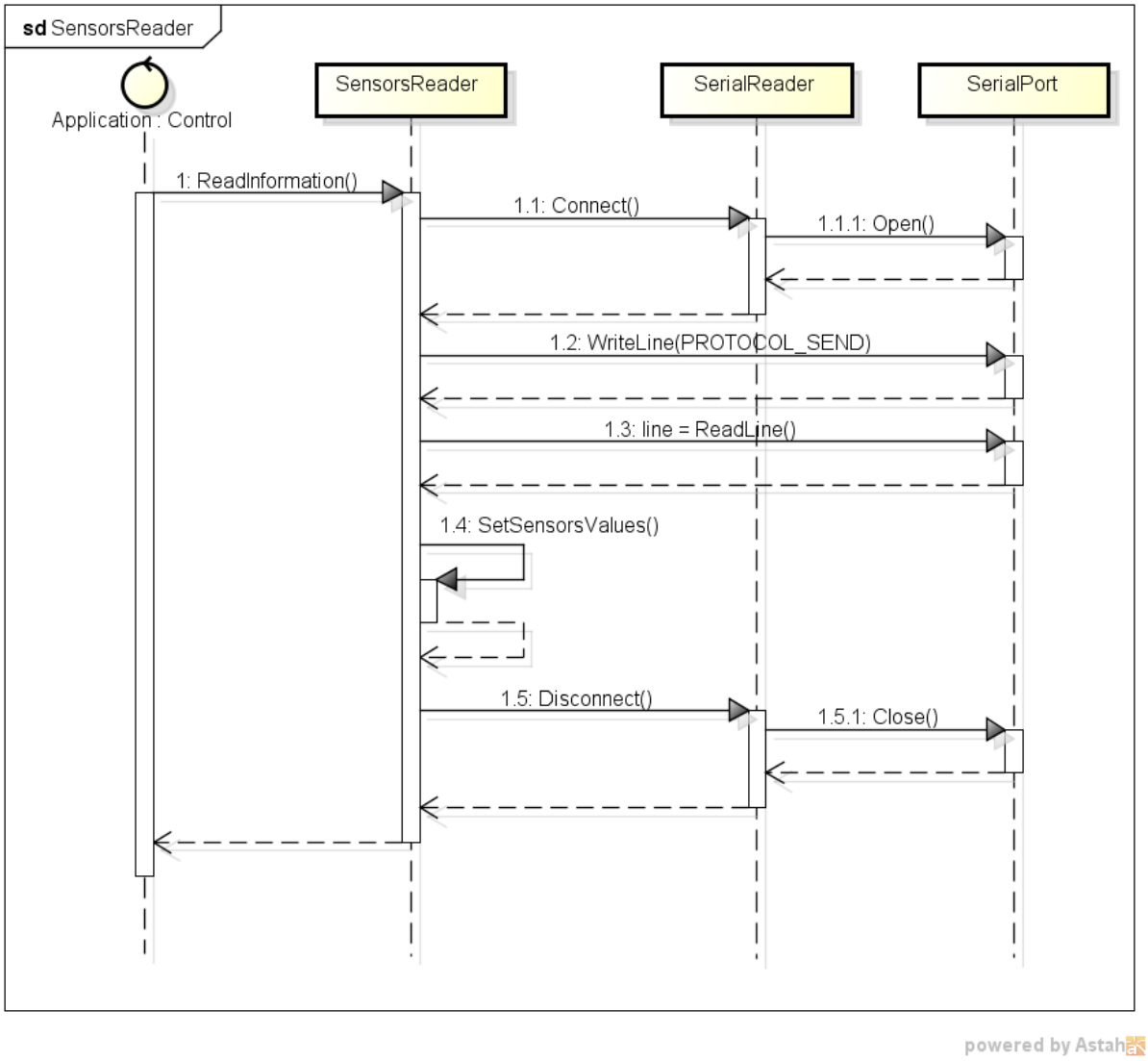

Figura 53: Diagrama de Sequência da Classe `SensorsReader`

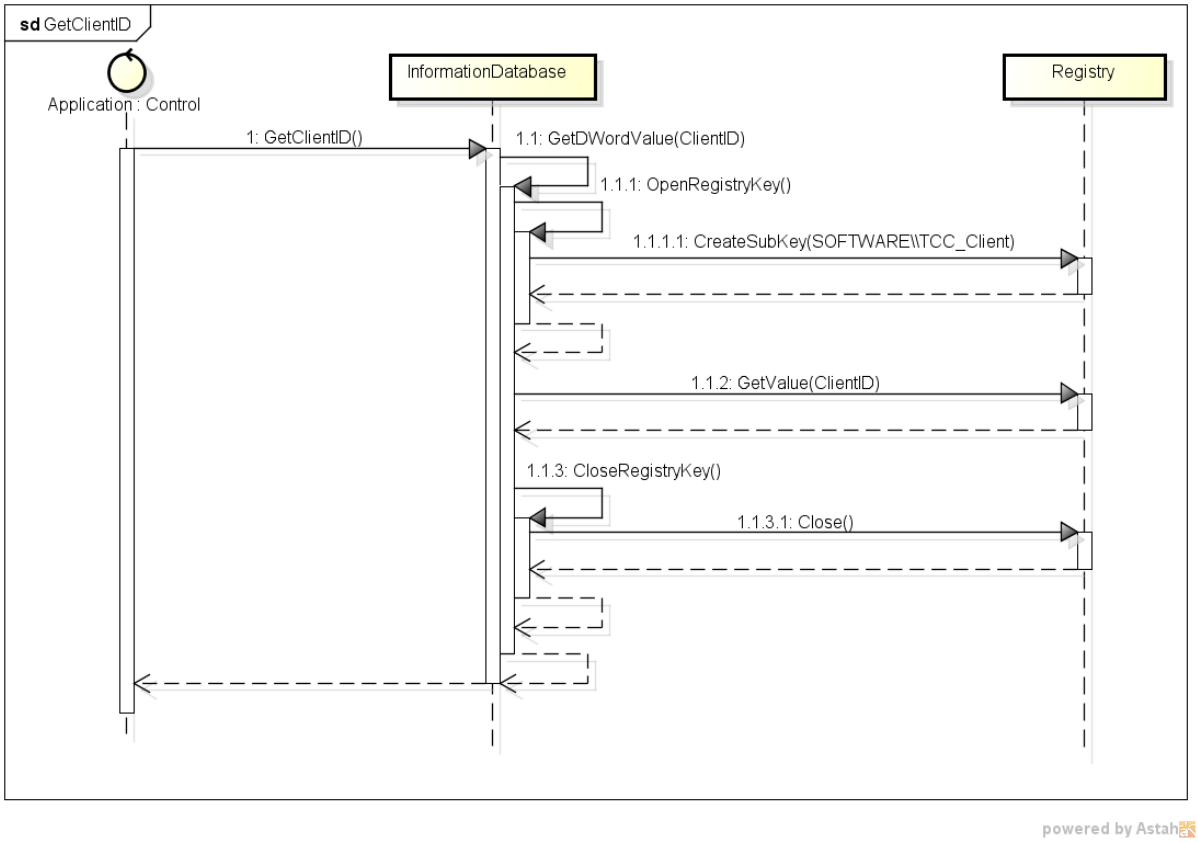

Figura 54: Diagrama de Sequência do Método `GetClientID` da Classe `InformationDatabase`

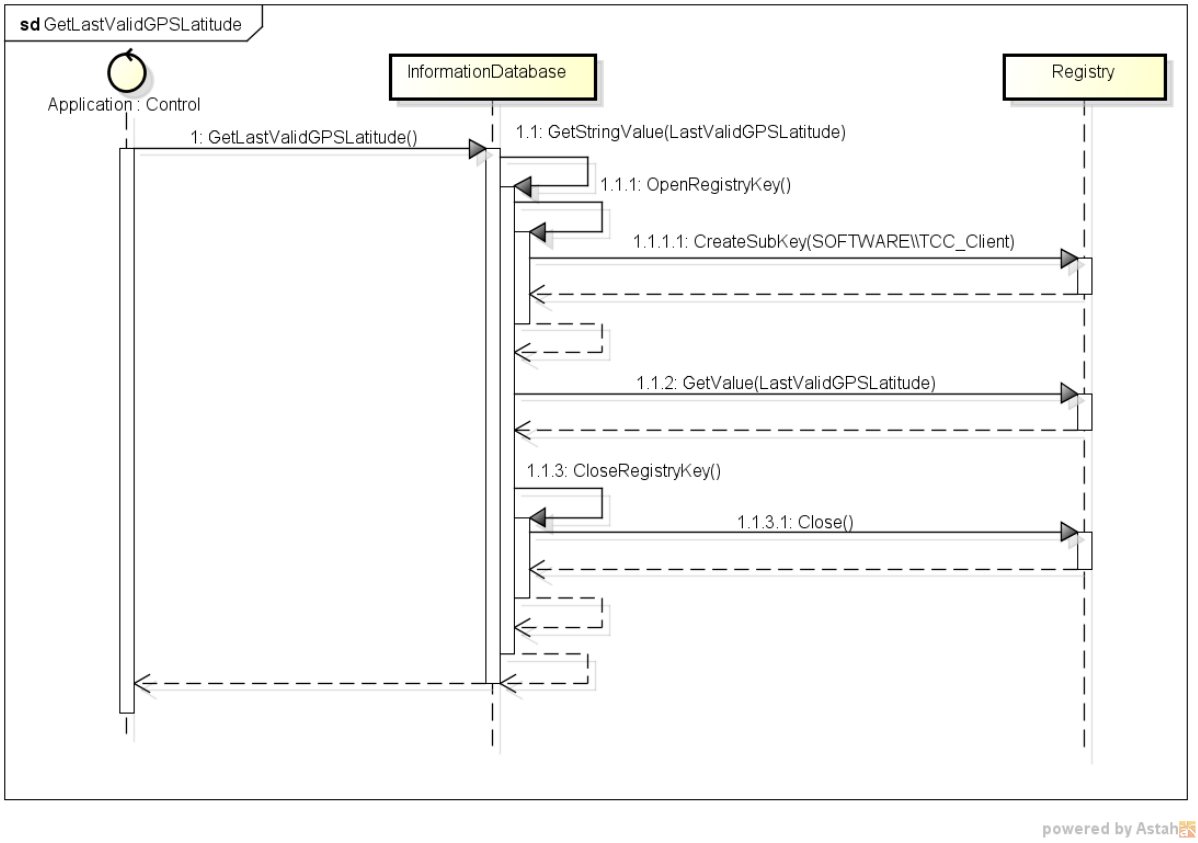

Figura 55: Diagrama de Sequência do Método `GetLastValidGPSLatitude` da Classe `InformationDatabase`

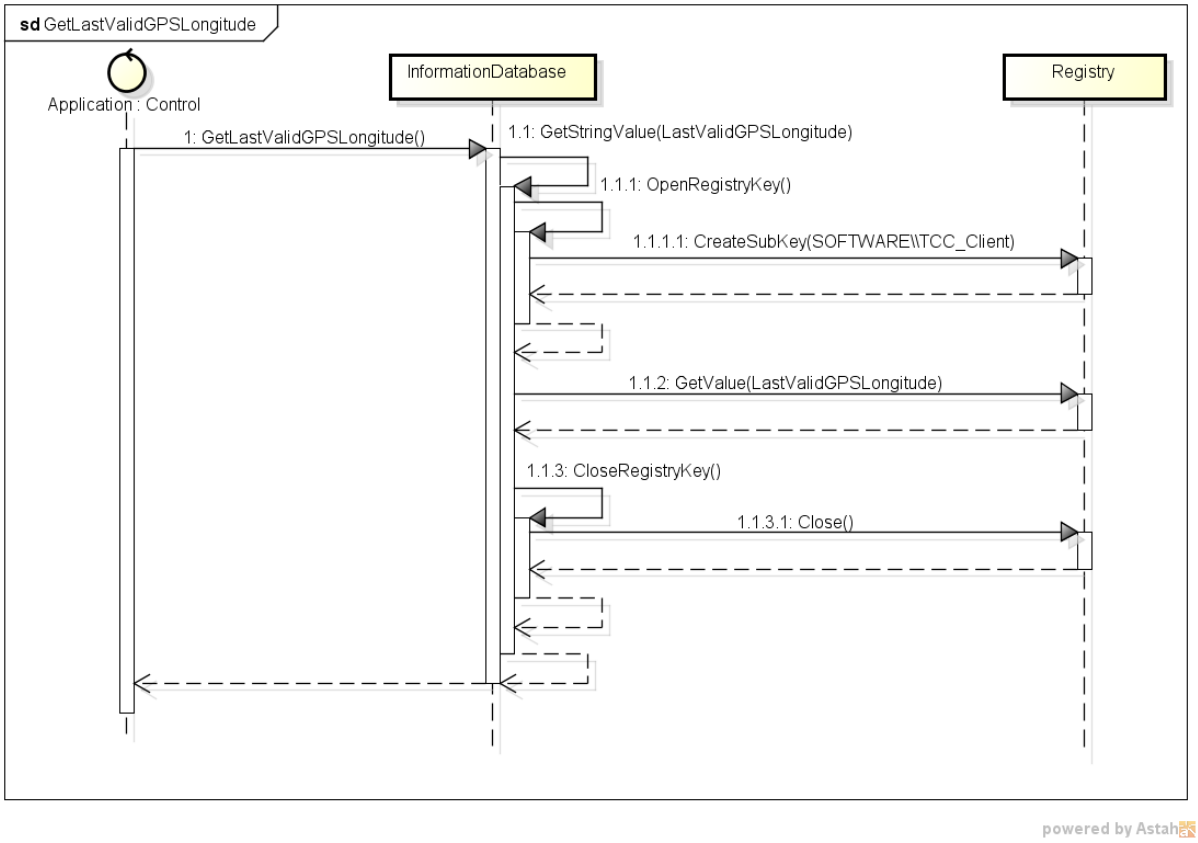

Figura 56: Diagrama de Sequência do Método `GetLastValidGPSLongitude` da Classe `InformationDatabase`

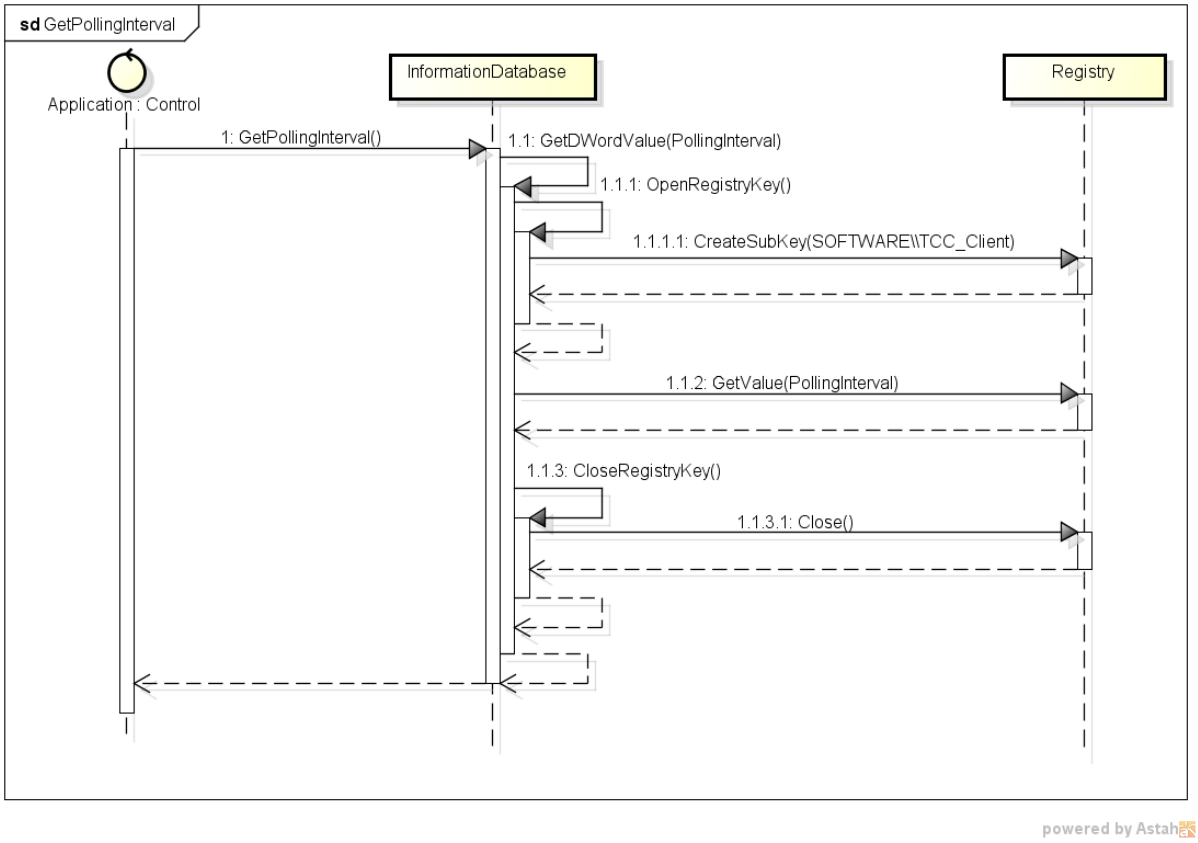

Figura 57: Diagrama de Sequência do Método `GetPollingInterval` da Classe `InformationDatabase`

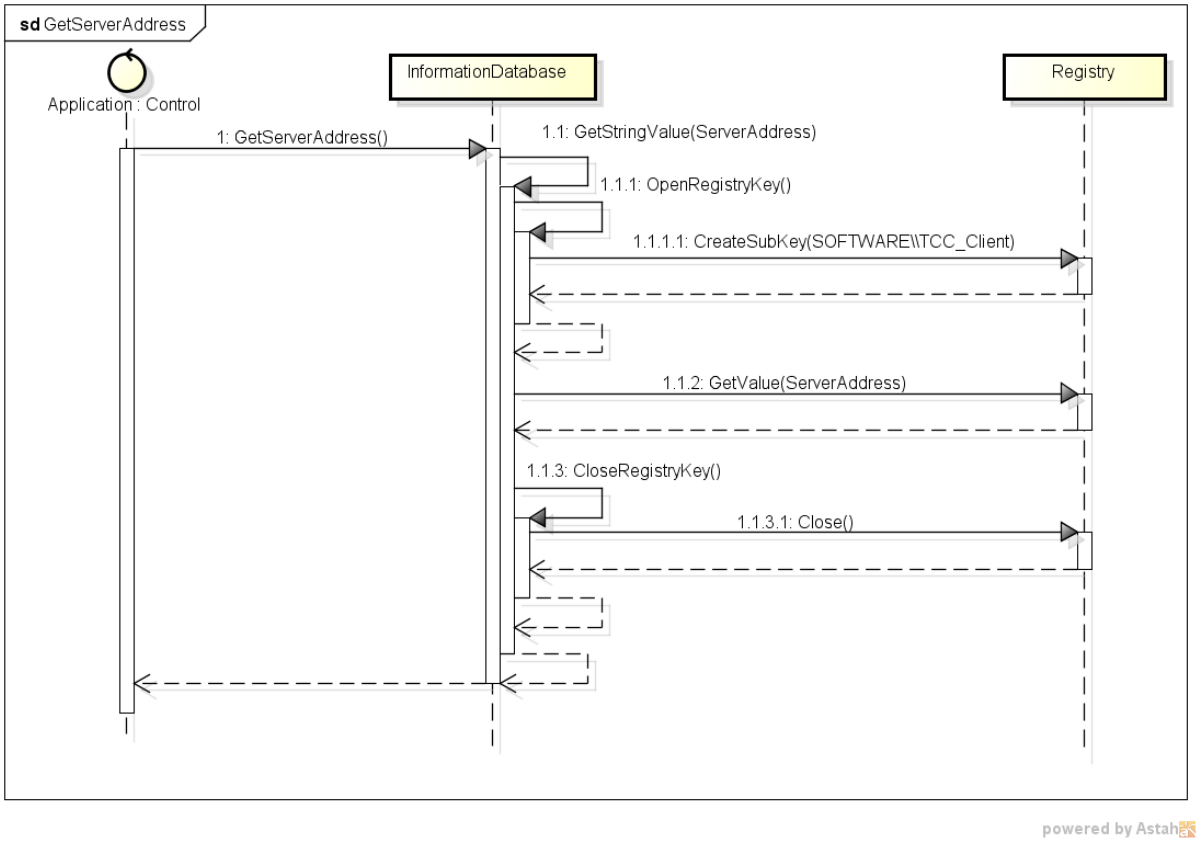

Figura 58: Diagrama de Sequência do Método `GetServerAddress` da Classe `InformationDatabase`

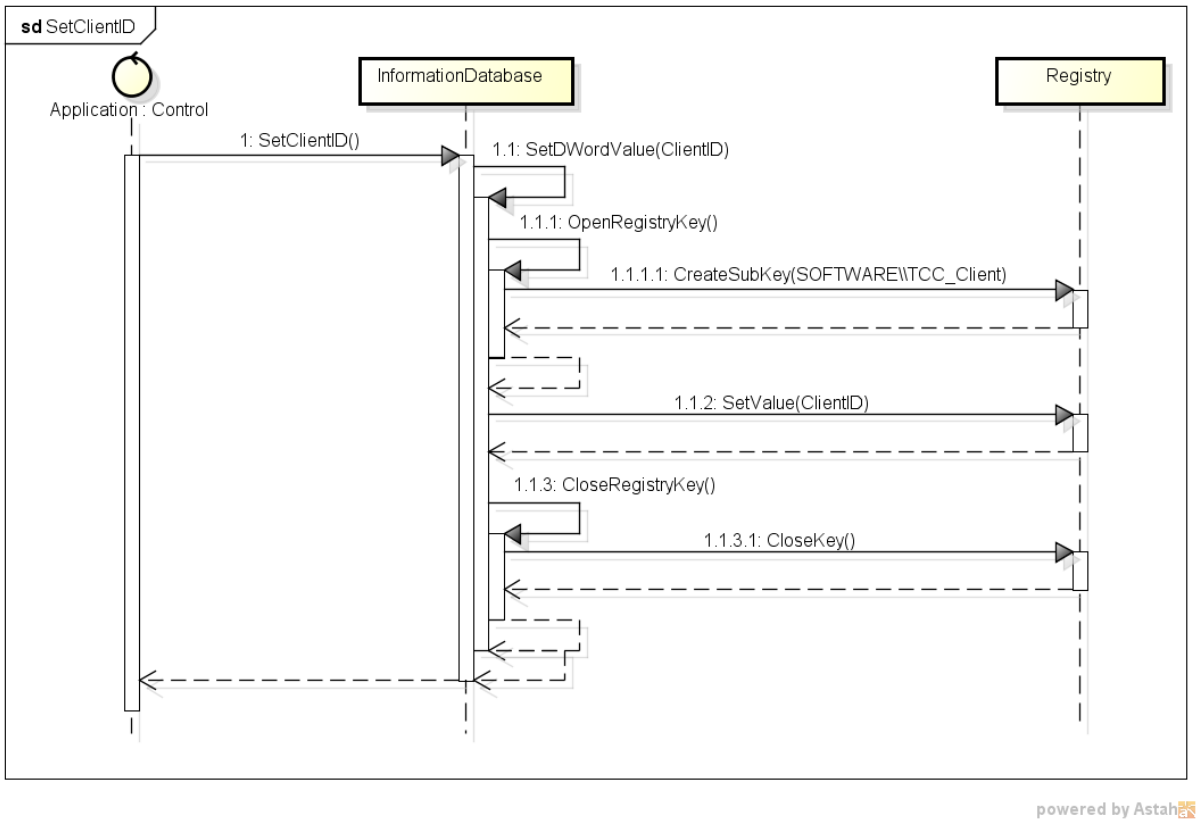

Figura 59: Diagrama de Sequência do Método `SetClientID` da Classe `InformationDatabase`

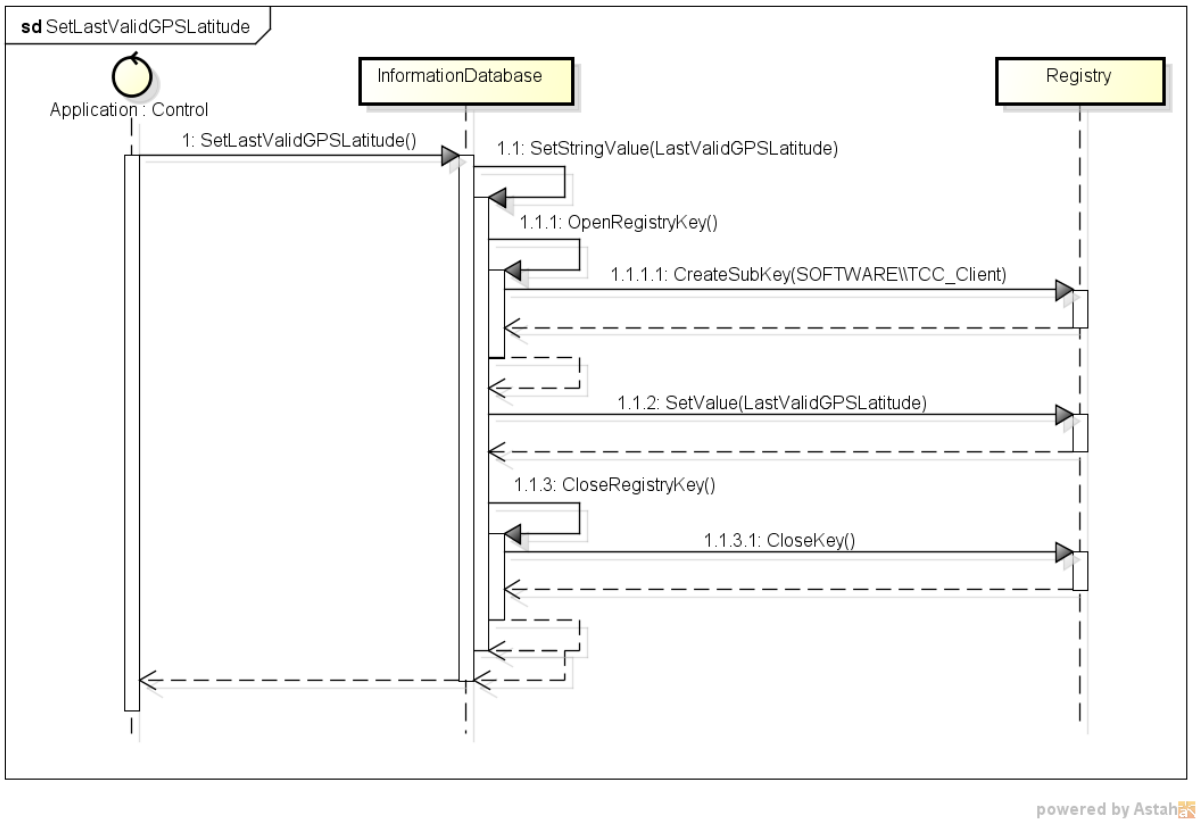

Figura 60: Diagrama de Sequência do Método `SetLastValidGPSLatitude` da Classe `InformationDatabase`

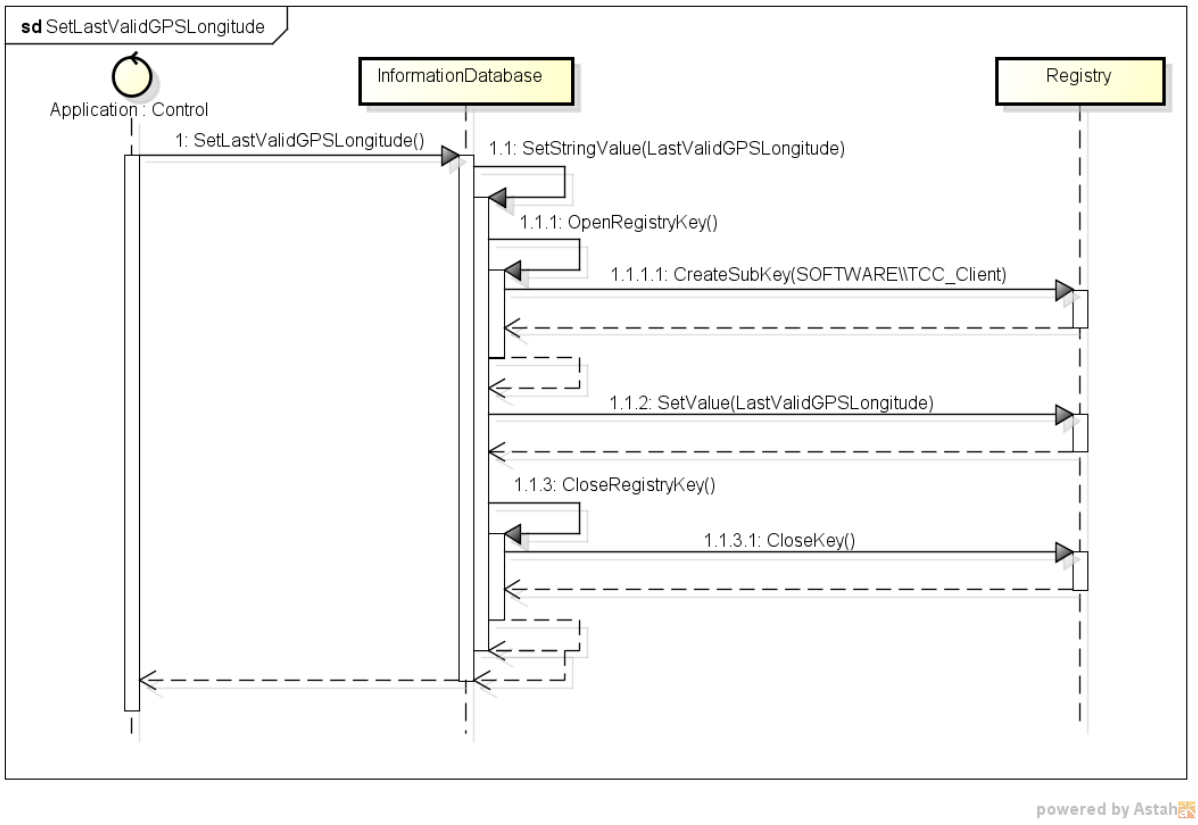

Figura 61: Diagrama de Sequência do Método `SetLastValidGPSLongitude` da Classe `InformationDatabase`

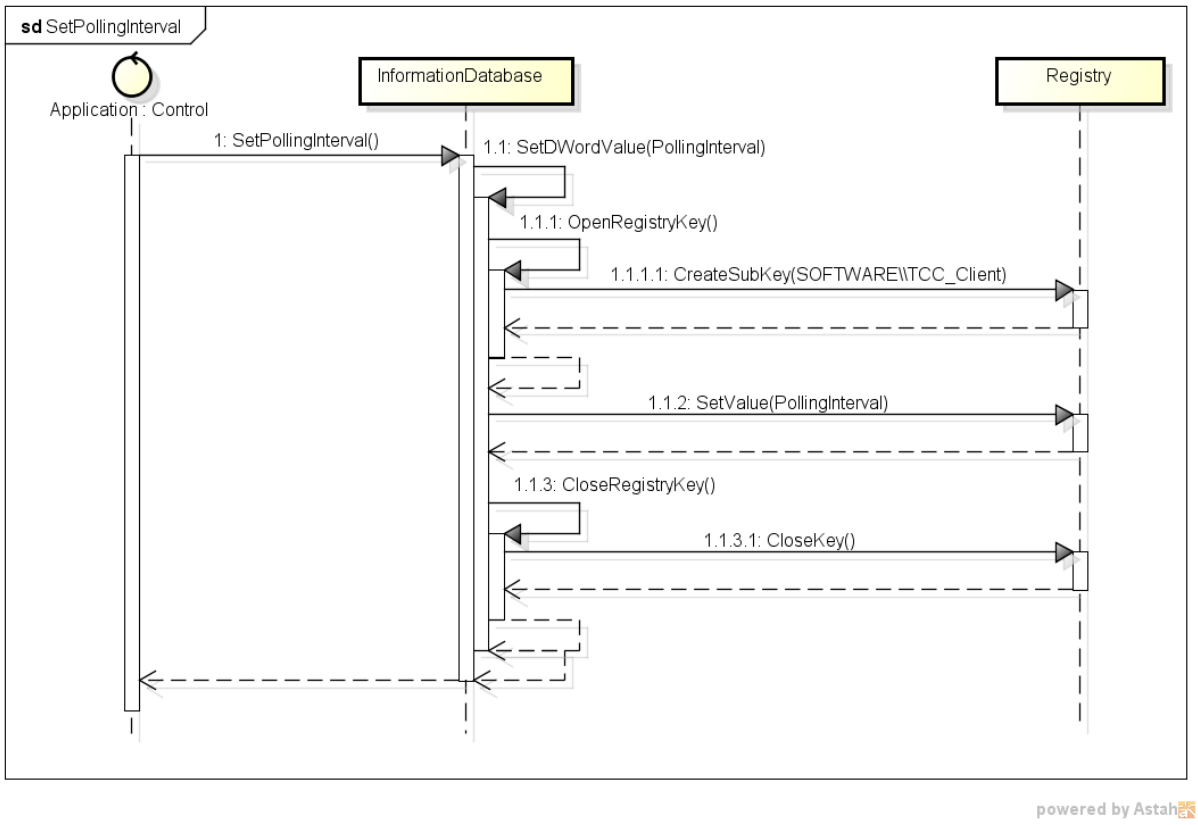

Figura 62: Diagrama de Sequência do Método `SetPollingInterval` da Classe `InformationDatabase`

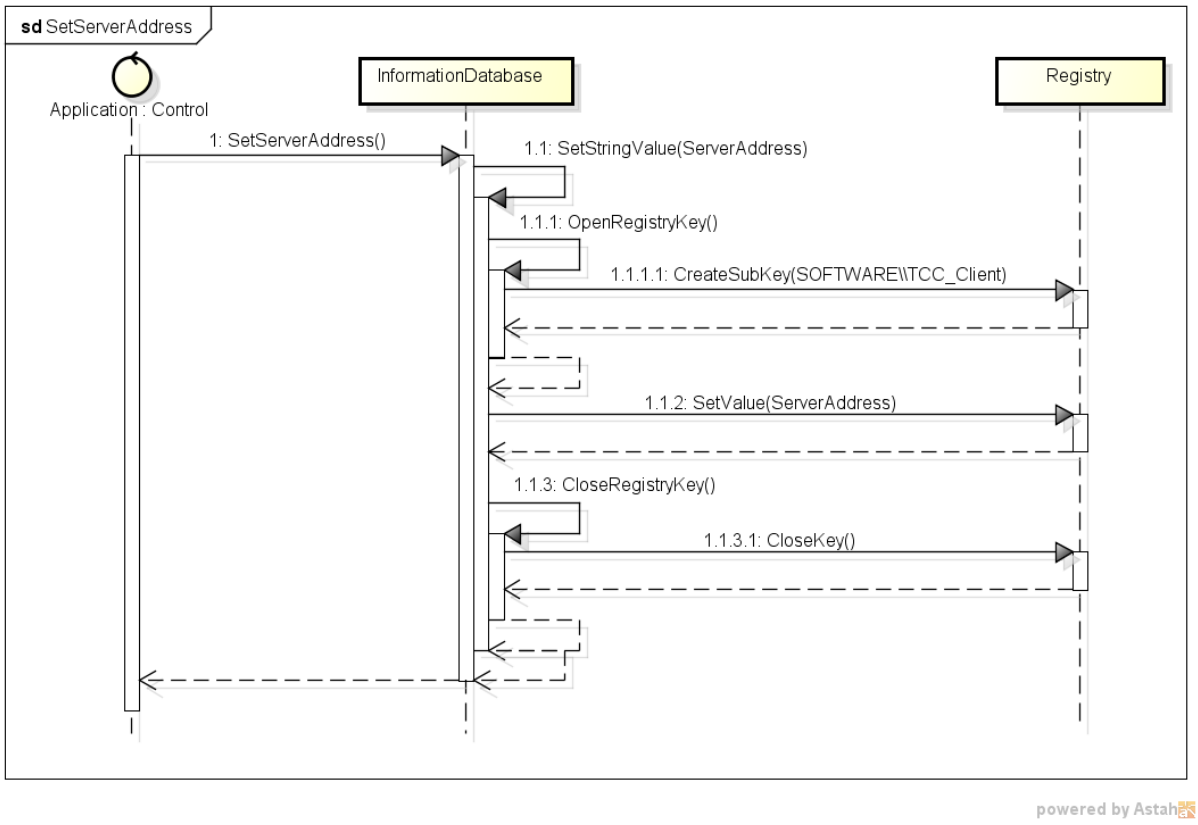

Figura 63: Diagrama de Sequência do Método `SetServerAddress` da Classe `InformationDatabase`