Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

Desenvolvimento de software para controle, aquisição de dados e telemetria em uma boia meteo-oceanográfica

Autor:

Marcel Corrêa de Mello

Orientador:

Fabio Nascimento de Carvalho, D. Sc.

Orientador:

Prof. Aloysio de Castro Pinto Pedroza, D. Sc.

Examinador:

Prof. Heraldo Luís Silveira de Almeida, D. Sc.

DEL - UFRJ

Julho de 2013

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica - Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro - RJ CEP 21949-900

DEDICATÓRIA

Dedico este trabalho às pessoas que felizmente fazem parte de minha e às que infelizmente deixaram de fazer. Sem elas, este trabalho não seria possível e sequer faria sentido.

AGRADECIMENTO

Agradeço à minha família pela amor, carinho e apoio incondicional durante toda esta jornada chamada vida. Agradeço também à minha namorada Juliana pelo amor, compreensão e eficientes palavras de incentivo. Aos meus amigos e companheiros de batalha, agradeço por estarem ao meu lado.

Agradeço aos professores do Departamento de Eletrônica da Universidade Federal do Rio de Janeiro pela intensa torrente de desafios. Agradeço aos meus orientadores Fábio Nascimento de Carvalho e Aloysio de Castro Pinto Pedroza.

RESUMO

Informações meteorológicas e oceanográficas obtidas através de medições de campo são essenciais para a previsão de eventos climáticos extremos e gestão de operações *offshore*. Estas variáveis podem ser medidas utilizando sistemas dedicados de aquisição de dados *in situ*. Este trabalho descreve a iniciativa do Laboratório de Instrumentação Oceanográfica, da COPPE/UFRJ, como parte do projeto de nacionalização de componentes de uma boia meteo-oceanográfica, de desenvolvimento do software embarcado de medição.

Batizado de Sistema de Controle, Aquisição e Telemetria(SisCAT), o projeto utilizou técnicas de engenharia de software, desenvolvimento ágil e gestão distribuída da equipe para a construção do protótipo aqui descrito. A especificação elaborada balizou o desenvolvimento, servindo de referencia para a codificação e os testes automatizados. As ferramentas adotadas contribuíram para amenizar as dificuldades gerenciais e operacionais inerentes ao desenvolvimento de um projeto de inovação em ambiente acadêmico com equipe espacialmente distribuída.

Palavras-Chave: Engenharia de Software, Telemetria, Sistemas Embarcados, Engenharia Oceânica.

ABSTRACT

Meteorological and oceanographical information measured by *in situ* data acquisition systems are essential for predicting extreme weather events and managing offshore operations. This paper describes the initiative of the Oceanographic Instrumentation Laboratory, at COPPE/UFRJ, to develop the telemetry embedded software as part of the nationalization of the components of a metocean buoy.

Named SisCAT(portuguese acronym for Telemetry, Control and Data Acquisition System), the design used software engineering techniques, agile development and distributed team management tools. These tools softened the challenges associated with management and operations of a innovative project developed in academic environments.

Key-words: Software Engineering, Telemetry, Embedded Systems, Ocean Engineering.

SIGLAS

UFRJ - Universidade Federal do Rio de Janeiro

REMETEOO - Rede de Eletrônica Meteo-oceanográfica *Offshore*

BMO - Boia meteo-oceanográfica

LIOc - Laboratório de Instrumentação Oceanográfica

SisCAT - Sistema de controle, aquisição e telemetria

MPPT - *Maximum power point tracking*

ICC - *Inductive Cable Coupler*

IMM - *Inductive Modem Module*

SBC - *Single-Board Computer*

IDE - *Integrated Development Environment*

TSM - Temperatura da Superfície do Mar

ADCP - *Acoustic Doppler Current Profiler*

CTD - Condutividade, Temperatura e Profundidade(*depth*)

DCVS - *Distributed Concurrent Versions System*

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Tema

O objeto de estudo deste trabalho é o projeto do software da unidade controladora de uma boia meteo-oceanográfica(BMO) utilizada para aquisição, processamento e transmissão de dados ambientais. O projeto envolve técnicas de engenharia de software e verificação de resultados a partir de protótipos funcionais.

1.2 Delimitação

O sistema limita-se a permitir a aquisição de dados ambientais através de sensores e instrumentos microprocessados, assim como o controle de parâmetros necessários para tal atividade. Exclui-se do escopo o pós-processamento e análise dos dados coletados assim como especificação formal de sistemas ou subsistemas presentes na BMO que não pertençam à área de software.

1.3 Justificativa

Informações meteorológicas e oceanográficas obtidas através de medições de campo são essenciais para a previsão climática e de eventos extremos que, por sua vez, são de fundamental importância para a sociedade atual. Estas informações são obtidas através de dados observacionais coletados *in situ* por sistemas de aquisição de dados ou por sensoriamento remoto.

Figura 1.1: Bóia meteo-oceanográfica da empresa Axys

Estes sistemas de aquisição de dados podem ser realizados na forma de bóias meteo-oceanográficas(BMO)(Fig 1.1), contemplando funcionalidades de transmissão de dados, medição de parâmetros ambientais e operação em baixo consumo. Visto o crescimento da atividade *offshore* no Brasil devido ao pré-sal e o tamanho de sua costa, identifica-se uma demanda crescente por BMOs.

Neste contexto, o desenvolvimento de tecnologia nacional para criação dessas boias permite diminuir os custos associados a operação, aprimoramento e gerenciamento do sistema. Assim surgiu o projeto da Rede de Eletrônica Meteo-oceanográfica (REMETEOO), apoiado pela Financiadora de Estudos e Projetos(FINEP) e realizado em conjunto pela COPPE, Marinha do Brasil, através do Centro de Hidrografia da Marinha(CHM), Universidade do Vale do Itajaí(UNIVALI) e as empresas Holos Brasil e Ambidados.

Ao Laboratório de Instrumentação Oceanográfica(LIOc) da COPPE, onde este trabalho foi desenvolvido, reside a responsabilidade pelos sistemas eletro-eletrônicos e computacionais. Relevante para o estudo aqui descrito, o software designado para execução na BMO precisa atender diversos requisitos como robustez, confiabilidade, precisão, fácil manutenção e extensibilidade. O desenvolvimento deste tipo de software tem uma série de criticidades associadas que precisam ser devidamente tratadas. Dificuldades como complexidade do sistema, capacidade computacional limitada da plataforma e códigos que interagem diretamente com hardware são alguns

exemplos.

Visando isolar esses aspectos críticos em um pacote de software a ser mantido separadamente de funcionalidades especificas da aplicação, como *drivers* de sensores e algoritmos de processamento de dados, e permitir rápido desenvolvimento de novos módulos, uma abordagem de engenharia de software orientada a módulos independentes mostrou-se adequada.

1.4 Objetivos

O objetivo do trabalho é exibir uma visão geral de uma boia meteo-oceanográfica, identificar as dificuldades e os desafios para o desenvolvimento de sistemas computacionais embarcados e detalhar a solução de software desenvolvida no LIOc. É esperado que este trabalho sirva como fonte de informações para futuros colaboradores do projeto REMETEOO, além de documentação do *know-how* adquirido até o momento.

1.5 Metodologia

Para realizar o projeto, a seguinte separação de atividades foi estabelecida:

- Pesquisa tecnológica em sistemas embarcados;
- Especificação de requisitos de hardware e software;
- Avaliação de ferramentas de software *open-source* e familiarização com o *Single-Board Computer* - SBC (Fig1.2) utilizado para o protótipo inicial;
- Definição da arquitetura do software e implementação do protótipo;
- Análise dos resultados.

Do levantamento de requisitos originou-se o documento usado como referência para o desenvolvimento do projeto. Semanalmente, os requisitos foram discutidos e a documentação inicial iterativamente adaptada. Assim, mudanças detectadas inicialmente eram integradas sem maiores alterações.

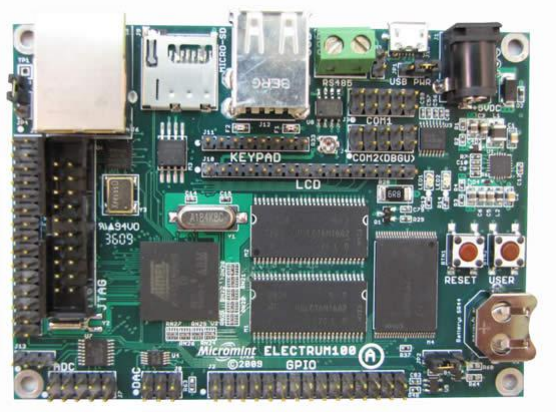

Figura 1.2: *Single-Board Computer* da empresa MicroMint

1.6 Descrição

No capítulo 2 será apresentada uma visão geral dos subsistemas que integram a boia meteo-oceanográfica do REMETEOO. Serão descritos os principais periféricos, interfaces de comunicação, modos de operação e outros aspectos importantes para a elaboração do software de controle.

No capítulo 3 o Sistema de Controle, Aquisição e Telemetria(SisCAT), software designado para ser executado na bóia meteo-oceanográfica, será alvo de detalhamento através de análises de requisitos, diagramas conceituais, representação com orientação a objetos e outros métodos que visam capturar o ideia do projeto.

O capítulo 4 entrará em detalhes sobre o desenvolvimento do software, características de implementação, ambiente de desenvolvimento e ferramentas. Também será detalhada a plataforma computacional utilizada no protótipo, suas interfaces e principais periféricos.

Os resultados obtidos com a implementação do software em um *Single-Board Computer* com alguns dos sensores previstos de incorporação à BMO são apresentados no capítulo 5 juntos de uma análise das dificuldades, vantagens e desvantagens da abordagem adotada. Por fim, tem-se sugestões de trabalhos futuros em relação ao REMETEOO e ao software desenvolvido, de forma a aprimorar e estender o conhecimento adquirido nesta empreitada.

Capítulo 2

A boia meteo-oceanográfica

A BMO do projeto REMETEOO conta com diversos subsistemas que, também pelo ponto de vista do desenvolvimento do software, precisam ser cuidadosamente analisados. Uma interação sadia entre os módulos e o software em execução é vital para o sucesso do projeto. Assim, para um melhor entendimento, elaborou-se um Diagrama de Visão (Fig 2.1).

2.1 O Diagrama de Visão

O diagrama de visão do projeto é uma metodologia da área de gerenciamento de projetos que permite uma rápida compreensão da BMO como um todo[1]. Utilizando-se de facilidades como a navegação(Fig 2.2) através de *links* entre módulos e especificações, é possível acessar informações relevantes entre os diversos componentes que compõem a solução proposta no projeto REMETEOO. Como exibido na Fig 2.1, diversas partes interagem diretamente com o software embarcado e precisam ser abstraídas de forma adequada quando forem computacionalmente tratadas.

Uma breve descrição dos principais módulos é necessária para a contextualização do problema.

2.2 Principais subsistemas da BMO

Como se pode observar no Diagrama de Visao Geral do projeto da BMO(Fig 2.1), este é composto pelos sistemas Boia, Fundeio, Recepção de Dados, Fundeios

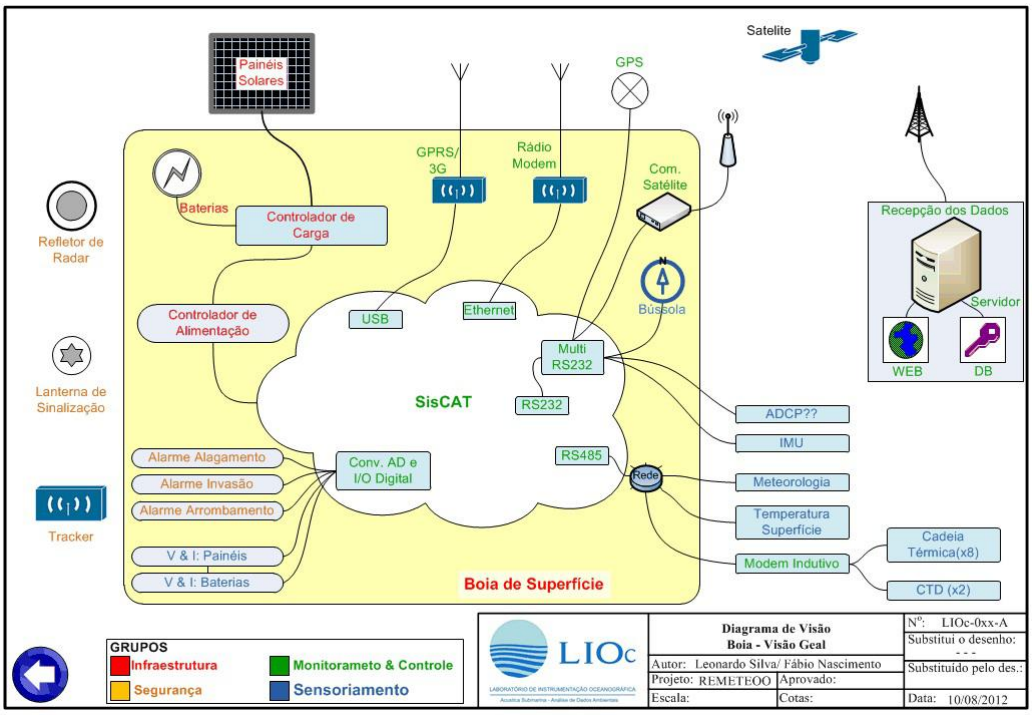

Figura 2.1: Diagrama de Visão da Boia e seus componentes diretos, elaborado pela equipe do LIOc

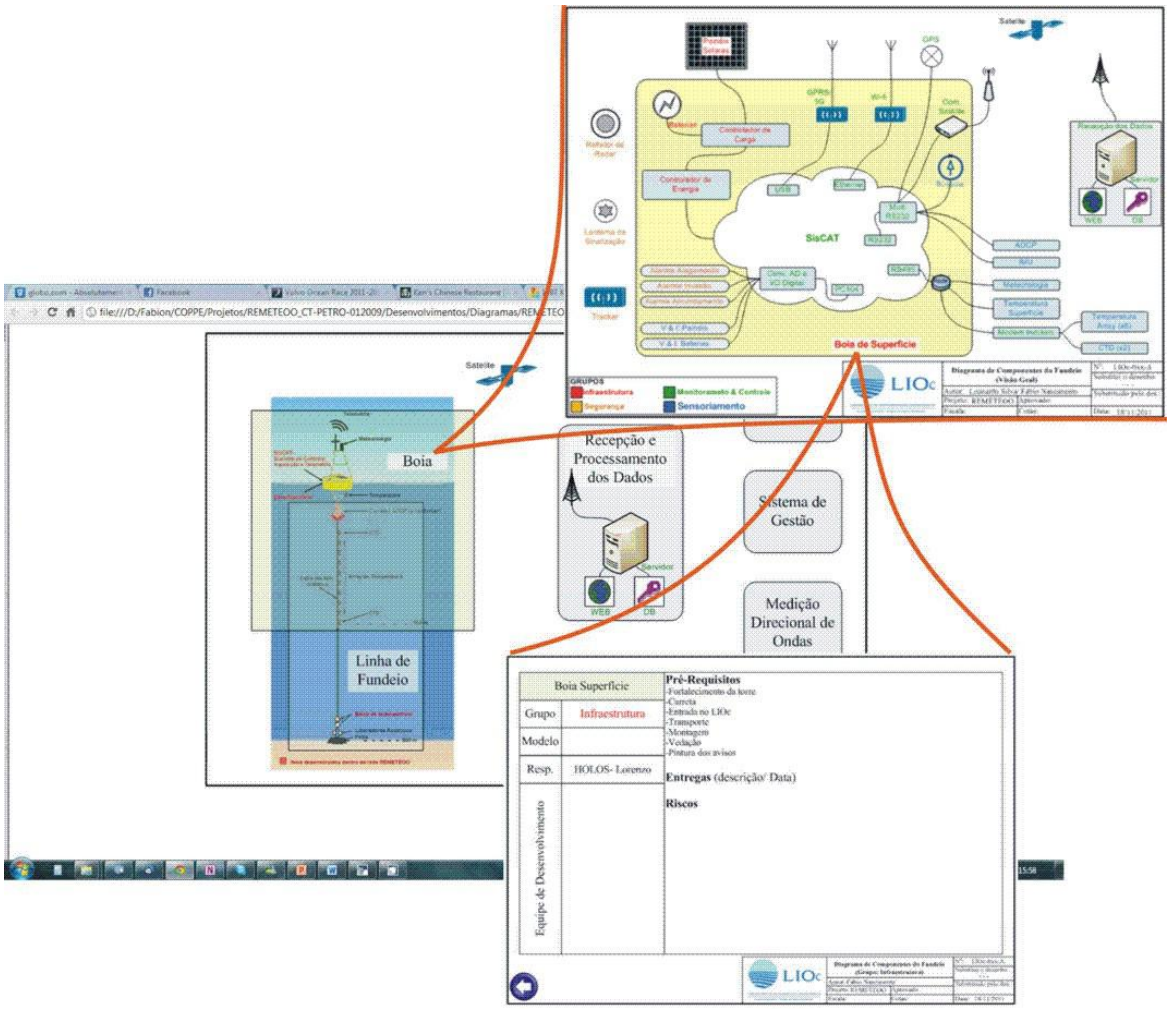

Figura 2.2: Exemplo de navegação: Visão Geral - BMO - Casco da BMO

no Mar e Gestão. Para o presente estudo, somente o primeiro é importante, pois os demais compreendem as parcelas referentes ao sistema de ancoragem da boia no mar, da recepção e processamento dos dados em terra, das operações com navios e do gerenciamento do projeto, respectivamente. Como estes sistemas não influenciam nos requisitos do projeto do software, não são parte do escopo deste trabalho.

Como se pode observar na Fig 2.1, existem 4 grupos nos quais os módulos se encaixam.

- Infraestrutura
- Segurança
- Monitoramento & Controle
- Sensoriamento

Uma descrição de cada um será realizada de forma a identificar a relação de cada módulo com o software desenvolvido. Assim, após delineados os aspectos relevantes de cada grupo(e seus componentes) ao software, o trabalho avançará para o projeto do Sistema de Controle, Aquisição e Telemetria da BMO, tratado a partir de agora pelo acrônimo SisCAT.

2.2.1 Infraestrutura

O conjunto de Infraestrutura da BMO é composto pelos sistemas gerenciamento de energia e pelo casco. As funções de gerenciamento de energia são executadas pelo controlador de carga, controlador de alimentação, painéis solares e baterias. Em conjunto, devem atuar a fim suprir por longos períodos - vários meses- os componentes da boia com energia elétrica de boa qualidade e também gerenciar o consumo energético com o objetivo de minimiza-lo. A outra parte da infraestrutura é o casco da BMO em si, que apesar da óbvia importância, não será detalhado por estar fora do escopo do software SisCAT.

A função primária do controlador de carga é gerenciar o trinômio: Fornecimento de energia para as cargas, carga/descarga das baterias e otimização da potência

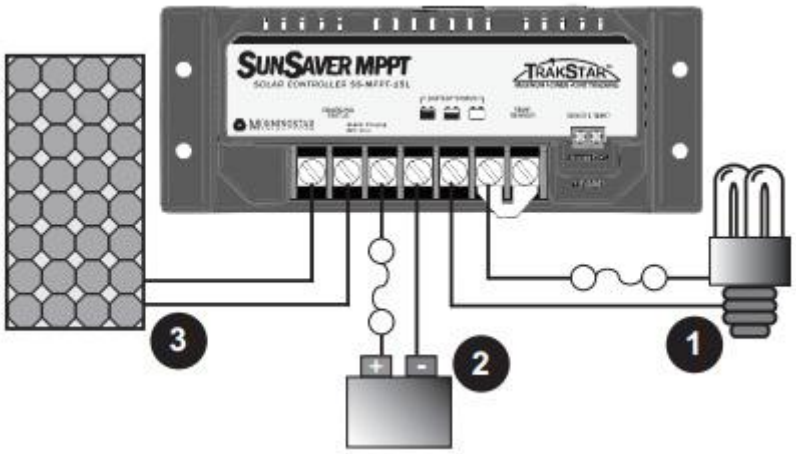

Figura 2.3: Exemplo de ligação entre baterias, painéis solares e controlador de carga. 1.Carga; 2.Bateria; 3.Painel Solar

gerada pelos painéis solares. O uso de um controlador de carga permite estender a vida útil das baterias empregadas na BMO, evitando intervenções precoces para troca de baterias e prolongando a estimativa de operação. Ao otimizar o uso das baterias, também é possível reduzir a quantidade delas, atendendo às restrições de espaço físico disponível e peso total embarcado.

A ligação entre o sistema de baterias, os painéis fotovoltaicos e o controlador de carga é feita de forma centralizada(Fig 2.3), delegando aos algoritmos de controle e balanceamento internos da controladora a responsabilidade pela regulação do fornecimento de energia e manutenção da vida útil das baterias. Também pode estar presente a aplicação da técnica *Maximum Power Point Tracking* - MPPT - para otimizar a coleta de energia dos painéis. O controlador de carga utilizado na boia é o SS-MPPT-15L (Fig 2.4) da empresa *MorningStar*. Este controlador possui funções de monitoramento remoto, *datalogging* e configuração via interface RS232, prevista na plataforma computacional. Estas funcionalidades são essenciais para o gerenciamento remoto de energia na BMO. Além disso, a possibilidade de *datalogging* permite estudos sobre o balanço energético ao longo dos períodos de operação.

O controlador de alimentação, subprojeto em andamento no LIOc, tem como objetivo permitir o chaveamento do fornecimento de energia para periféricos seleci-

Figura 2.4: Controlador de carga *SS-MPPT-15L*

onados. Esta funcionalidade permite um controle mais preciso sobre o consumo de energia da BMO. O módulo interfaceará com a plataforma computacional para receber os comandos necessários, comportando-se como um componente cujo controle será feito pelo software.

Do ponto de vista do software em execução, o monitoramento das baterias, dos painéis fotovoltaicos e da alimentação de periféricos é feito por parâmetros e regras estabelecidas pelo usuário do sistema através do arquivo de configuração. Por exemplo, para um valor abaixo de um nível pré-estabelecido de tensão nas baterias, o software pode tomar medidas necessárias para o prolongamento da operação ou até mesmo desligar-se para voltar a operar somente após uma recarga desejada.

2.2.2 Segurança

O subsistema de segurança engloba as partes da BMO referentes a itens de navegação obrigatórios, proteção e integridade física do sistema. A Marinha do Brasil estabelece diversas regras de segurança no mar que devem ser seguidas a fim de garantir concordância entre a operação da BMO e as práticas de segurança vigentes. O documento NORMAS DA AUTORIDADE MARÍTIMA PARA AUXÍLIOS À NAVEGAÇÃO NORMAM-17/DHN descreve os itens que devem ser levados em consideração em tal empreitada marítima. Da extensa lista de objetos associados a

Figura 2.5: Refletor de radar Echomax 230BR

auxílio à navegação, os itens previstos para a BMO são:

- Refletor de Radar
- Lanterna de Sinalização
- Rastreador de posição independente
- Alarmes de Arrombamento, Invasão e Alagamento

O Refletor de Radar(Fig 2.5) é um dispositivo cujo objetivo é refletir os pulsos emitidos pelos radares de outras embarcações de forma que a posição da BMO seja mais facilmente explicitada, evitando colisão e outros contra-tempos. Como o material utilizado para a fabricação do casco da BMO não reflete os sinais eletromagnéticos incidentes, este dispositivo resolve esta parte do problema de navegação.

Outro item de auxilio à navegação incluído na BMO é a Lanterna de Sinalização. A função deste item é auxiliar na visibilidade da boia durante o período noturno. O modelo escolhido conta com funções programáveis de lampejo e sua visibilidade chega a mais de 3 milhas náuticas na maioria das situações. A figura 2.6 exibe o modelo adotado.

Figura 2.6: Lanterna refletora Carmanah m650

Os alarmes serão desenvolvidos internamente pela equipe do LIOc em uma das fases finais da construção da BMO e integrados ao SisCAT. Estes deverão possibilitar ao sistema detectar algumas situações de vandalismo e alagamento e tomar decisões pré-programadas. O rastreador de posição, ou *Tracker*, consiste em um módulo com GPS, comunicação via satélite, antena e baterias. Estará localizado de forma dissimulada no casco da BMO e alertará automaticamente para casos de deriva ou alteração de posição.

2.2.3 Monitoramento & Controle

Esta seção descreverá brevemente os meios de comunicação utilizados na BMO, seja em ambiente de testes, operação plena ou nas redes de sensores. Pela necessidade de também poder operar longe da costa continental, a BMO precisa de sistemas de comunicação que permitam transmissão horária dos dados para terra, assim como, no sentido inverso, envio de parâmetros de controle ou manutenção da terra para a BMO. Assim, diversas tecnologias poderão ser utilizadas:

- Rádio Modem
- Rede 3G/GPRS
- Transmissão via satélite

O Rádio Modem é um dispositivo que permite conexão serial com a BMO para distâncias de até 30 quilômetros em visada direta. Deste modo, uma conexão direta com o computador embarcado pode ser estabelecida pelo usuário para execução de rotinas de manutenção, teste e gerenciamento. É importante notar que esta possibilidade diminui a necessidade de intervenção direta na BMO, o que é custoso e deve ser evitado quanto esta está no mar. O modelo utilizado neste projeto conta com interfaces RS232 e RS485 simultâneas, opera na frequência 902-928MHz e, caso necessário, permite criptografia dos dados. Vale ressaltar que o objetivo da BMO é operar a longas distancias da costa, de forma que a limitação de 30km para o uso do rádio modem faz com que sua aplicabilidade seja possível nas fases de teste e desenvolvimento somente, quando a BMO estará perto da costa. Ao final do desenvolvimento, ela poderá ser instalada em regiões com lâmina d'água superior a 2.000m e distantes centenas de quilômetros da costa, ficando o radio para controle a partir da embarcação de apoio.

Para os testes de campo, também é possível utilizar a rede de dados de telefonia 3G através de um modem USB. A velocidade de conexão desta rede permite transferências de dados em alta velocidade, o que já é uma grande vantagem em comparação ao rádio modem. Uma outra vantagem sólida do uso da conexão 3G é a inserção da plataforma computacional em uma rede IP, tornando possível acesso remoto por *Secure Shell*(SSH) e uso de outros protocolos mais amigáveis para compartilhamento de arquivos, como o *File Transfer Protocol*(FTP). O alcance desta solução, porém, limita-se a disponibilidade da cobertura da rede de telefonia.

O meio de comunicação, quando a BMO estará em plena operação, longe da costa, é a transmissão de dados por satélite através da rede Iridium de satélites. Este serviço permitirá a transferência de dados previamente compactados mesmo quando a BMO se encontrar em posições remotas do planeta. A rede Iridium é constituída por 66 satélites que orbitam a Terra a uma altura de aproximadamente 781km, na região *Low Earth Orbit*(LEO). Como pode-se imaginar, o uso de uma rede de satélites para transferência de dados incorre em altos custos para o projeto. Assim, o software contará com técnicas de compactação de dados e codificação de comandos para minimizar o uso da rede, que opera no padrão *Short Burst Data*(SBD),

possuindo a limitação de envio de mensagens de no máximo 320 bytes.

Para a comunicação entre os instrumentos microprocessados e o SBC, as seguintes interfaces estão disponíveis:

- Rede RS485
- Portas RS232
- Portas USB
- GPIO's (do inglês, *General Purpose Input/Output*)
- Modem Indutivo(Conectado a RS232 ou RS485)
- Porta Ethernet

Destas, cabe explicar o funcionamento do modem indutivo. Os sensores de temperatura, com datalogger's internos, serão distribuídos até a profundidade de 150m e se comunicarão com o SisCAT através de uma rede indutiva que faz fronteira com a rede RS485 da BMO(Fig 2.7). Esta rede indutiva, submersa na água e acoplada à linha de fundeio, utiliza como meio físico de transmissão o cabo de aço revestido com PVC ao qual os instrumentos estão ligados. A ligação entre os instrumentos e o cabo é feita através de um acoplador de cabo indutivo(*Inductive Cable Coupler*, ICC), que de forma semelhante a um transformador, propaga os sinais elétricos pela rede. A figura 2.8 ilustra a ligação de dois componentes na rede: um instrumento(*IM Instrument*) e o modem indutivo(*SIM - Surface Inductive Modem*). Internamente, o funcionamento da rede indutiva, ou seja, a troca de dados entre instrumentos e SIM, é regida por protocolos do fabricante, de forma que o acesso do SisCAT é feito através do SIM, módulo com o qual o software interage.

2.2.4 Sensoriamento

A seguir, serão descritos os sensores meteorológicos e oceanográficos instalados na BMO com o objetivo de obter dados ambientais para a análise de eventos extremos ou a melhoria de modelos numéricos de previsões de clima ou suporte a operações marítimas.

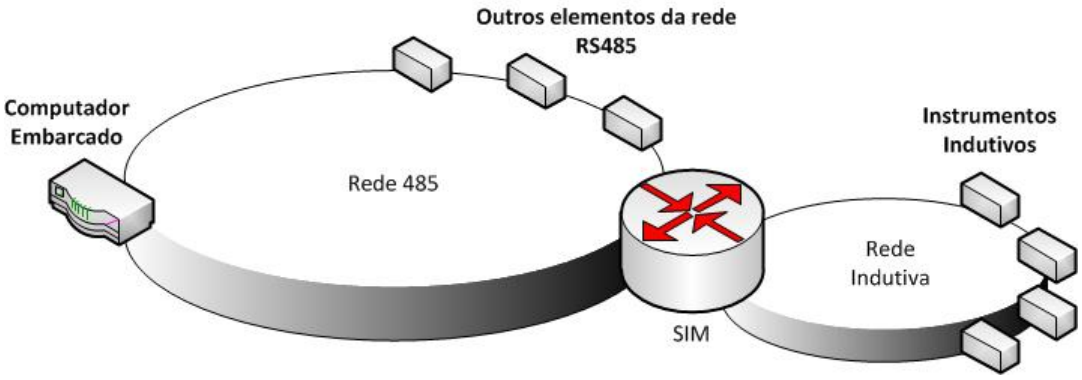

Figura 2.7: Esquema da rede RS485, o modem indutivo(SIM) e a rede indutiva Seabird

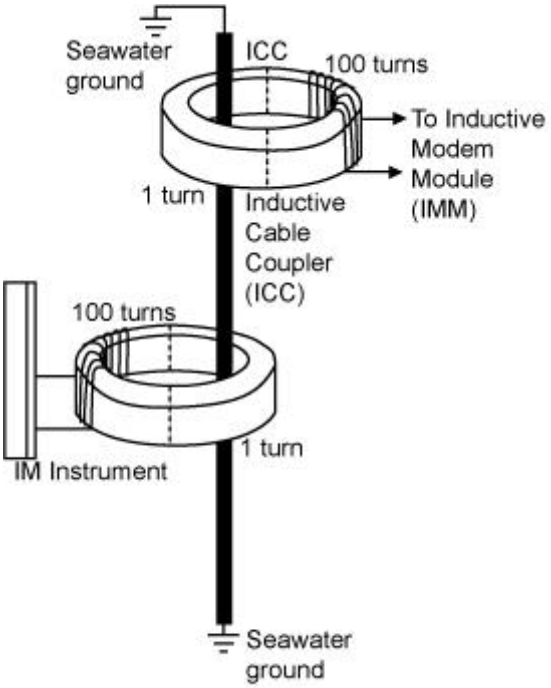

Figura 2.8: Ligação com ICC's entre um instrumento e o modem indutivo, tecnologia da empresa SeaBird

Figura 2.9: Estação Meteorológica Vaisala WXT520

2.2.4.1 Estação meteorológica

Composta por sensores de temperatura do ar, umidade relativa, pressão atmosférica, pluviosidade, direção e intensidade de ventos, possibilita a amostragem dos principais parâmetros meteorológicos para as previsões de tempo, clima e ondas no oceano. A estação meteorológica, ou sensor multiparamétrico, Vaisala WXT520(Fig 2.9), através de programação, permite a escolha de quais parâmetros devem ser enviados a controladora mas não possui armazenamento interno de dados.

2.2.4.2 Sensor de temperatura da superfície do mar - TSM

Termômetro de alta precisão que mede a temperatura da água à profundidade de 1 metro. Os dados obtidos são bastante utilizados para a comparação com informações de imagens de satélites. O sensor de TSM escolhido foi o Seabird SBE38 (Fig 2.10), que não possui armazenamento interno e utiliza interface RS-232 ou RS-485 para

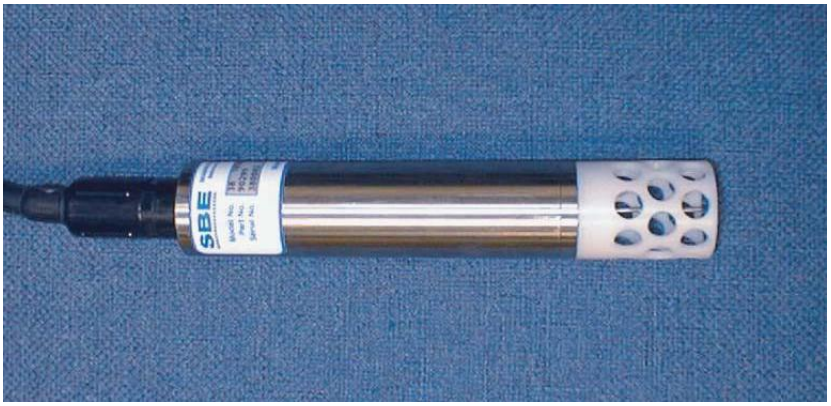

Figura 2.10: Sensor de Temperatura de Superfície do Mar(TSM) Seabird SBE38

comunicação.

2.2.4.3 Medidor de Correntes - ADCP

O medidor acústico de perfil de correntes por efeito Doppler, ou *Acoustic Doppler Current Profiler* - ADCP, tem como princípio de funcionamento a medição do valor do desvio da frequência ocorrido pela reflexão do sinal acústico nas partículas de água em movimento - o efeito Doppler. Este equipamento emite sinais acústicos e em seguida aguarda o retorno das componentes de sinal refletidas nas partículas suspensas nas massas d'água que compõem as correntes marinhas. O tempo de resposta e o desvio Doppler permitem saber a distância do ADCP para a partícula e sua velocidade. Com um ADCP de três feixes, utilizado nesta BMO, é possível resolver as correntes tridimensionalmente a até uma distância de 500 metros. O ADCP da BMO, Teledyne RDI LongRanger 75 KHz (Fig 2.11), possui armazenamento interno(datalogger) e não estará conectado à rede indutiva, não sendo, então, arguido pelo SisCAT.

2.2.4.4 CTD's

Os medidores de Condutividade, Temperatura e Profundidade(*depth*) - CTD - são fundamentais nos estudos oceanográficos. Por exemplo, no estudo de composição das massas d'água e de suas movimentações, a temperatura, pressão e salinidade influenciam diretamente a densidade da água, e com isso, a sua circulação vertical.

Figura 2.11: Medidor Acústico de Perfil de Correntes(ADCP) Teledyne RDI LongRanger 75 KHz

O CTD a ser usado, Seabird SBE37IM(Fig 2.12), possui datalogger interno e estará ligado à rede indutiva, podendo ser arguido pelo SisCAT.

2.2.4.5 *Array* de Temperatura

Esta cadeia de oito sensores de temperatura de alta precisão, Seabird SBE39IM(Fig 2.13), possibilitará o conhecimento, através das temperaturas dos primeiros 150 m de profundidade, dos deslocamentos verticais das massas d'água do oceano, região esta chamada de "Camada de Mistura". Cada elemento da cadeia terá alguns valores arguidos pelo SisCAT através da rede indutiva além de contarem com datalogger's internos.

Figura 2.12: Medidor de Condutividade, Temperatura e Profundidade(CTD) Seabird SBE37IM

Figura 2.13: Sensor de temperatura Seabird SBE39IM

Capítulo 3

Sistema de Controle, Aquisição e Telemetria - SisCAT

Esta seção é dedicada ao debate sobre requisitos, necessidades e características que o SisCAT deve possuir. Também será descrita a especificação utilizada como referência para a codificação do protótipo. Esta especificação se dá através de um documento de Especificação de Requisitos, uma modelagem por classes de domínio, com descrições do comportamento de cada classe, e por detalhamento do comportamento dinâmico através de diagramas de sequência.

3.1 Documentos do Projeto

O projeto foi documentado através de um conjunto de diagramas que procuram especificar sua estrutura estática e seu comportamento dinâmico. Os documentos elaborados durante o projeto do SisCAT foram:

- Especificação de Requisitos
- Diagrama de Classes de Domínio
- Diagrama Sequencial
- Diagrama de Classes de Implementação

A abordagem utilizada para registrar os conceitos levantados até agora foi a Modelagem por Classes de Domínio [2]. Esta modelagem coloca em um mesmo diagrama

conceitual os tópicos relacionados ao problema, descreve suas entidades, atributos, papéis e relacionamentos. Também permite exibir as restrições conhecidas no relacionamento entre as classes e reforçar o jargão adotado para o projeto. Neste trabalho, a Modelagem por Classes de Domínio é complementada por Diagramas de Sequência e a Documentação de Implementação, que é gerada dinamicamente através do código escrito. É critério de qualidade interno a concordância entre todos os documentos.

3.2 Análise de Requisitos

O Sistema de Controle, Aquisição de Telemetria(SisCAT) é o software designado para executar durante a operação da bóia meteo-oceanográfica(BMO) do projeto REMETEOO. Para defini-lo e especificá-lo corretamente, uma análise dos requisitos é adequada [3]. Pelo tipo de software em questão, diversos aspectos têm de ser avaliados e especificados com a melhor qualidade possível. O objetivo desta sessão é apresentar uma visão geral dos requisitos analisados. Entre os quais estão:

- Interfaces(I/O) do sistema;
- Interação com usuários e arquivos de configuração;
- Dados;
- Funcionalidades;
- Recursos disponíveis na BMO;

As interfaces identificadas como necessárias para o SisCAT são determinadas pelos instrumentos necessários para os tipos de medições que ocorrerão, pelos aparelhos de comunicação e de infraestrutura da BMO. De posse do inventário de sensores e periféricos externos que precisam ser operados e integrados, determinou-se que o SisCAT deve poder gerenciar as interfaces físicas RS232, RS485, USB, Ethernet e pinos GPIO(*General Purpose Input Output*). Esse gerenciamento prevê o controle de acesso entre instrumentos em rede para evitar problemas de concorrência em uma mesma interface física(ex.: RS485).

Os usuários-alvo do projeto REMETEOO são os pesquisadores da comunidade científica brasileira que atuam nas áreas de ciências do mar. É esperado que o SisCAT permita executar tarefas de medições de acordo com parâmetros configurados pelo operador. O perfil de uso da BMO pode variar com o tempo devido a novas necessidades científicas ou reconfiguração de manutenção, por exemplo. Entretanto, o local de operação da BMO torna inviável intervenções manuais frequentes. Dessa forma, também é esperado que a configuração feita pelo usuário possa ser modificada remotamente. Configurações gerenciais também precisam ser feitas e alteradas remotamente conforme necessário.

Os dados resultantes das aquisições devem ser armazenados mantendo todas as informações necessárias para recuperação do contexto no momento da medição. As informações que permitem esta reconstrução estão na tabela 3.1. Estes dados devem ser armazenados atendendo critérios de memória, custo computacional para leitura, escrita e compactação. A precisão dos valores é determinada pelo instrumento utilizado e não deve ser alterada pelo SisCAT.

Tabela 3.1: Informações de contexto da medição.

Informação	**Tipo de dado**
Identificação do Instrumento	Número inteiro(sequencial para mesma rede)
Modelo do Instrumento	Texto alfanumérico
Data da medição	Data
Hora da medição	Hora
Valor original da medição	Texto alfanumérico
Interface Física	Número inteiro(sequencial e configurado)

O SisCAT deve realizar todas as operações registradas no arquivo de configuração do usuário sob os limites de operação estabelecidos pela configuração interna. Também deve ser possível a utilização de algoritmos de processamento de dados objetivando cálculos relevantes ao domínio dos usuários-alvo, por exemplo, extração de informações espectrais de ondas através de séries temporais medidas. Estes algoritmos podem ter parâmetros configuráveis pelo usuário e novos algoritmos devem poder ser adicionados em novas versões com o menor impacto possível no desenvolvi-

mento. Todas as medições devem ser feitas com temporização precisa e controladas por um *timer* físico de alta resolução. O SisCAT deve gerenciar os recursos computacionais de forma a otimizar o uso de processamento, memória e armazenamento. É esperado que informações sobre disponibilidade energética estejam disponíveis para o SisCAT de forma que, se necessário for, o sistema entre em modo de baixíssimo consumo de energia. Uma relação completa das funcionalidades pode ser encontrada no Documento de Especificação de Requisitos.

Deve ser criado um mapeamento de possíveis erros no software e eventos de interesse devem ser armazenados em um log. Estas informações serão utilizadas para análise de problemas caso necessário. O intervalo de tempo armazenado no log deve ser configurável para otimizar o custo de armazenamento.

O sistema também deve ser construído para ser facilmente testável, uma vez que a criticidade do SisCAT é alta. Os testes servirão para validar a performance do sistema e verificar o pleno cumprimento dos requisitos, tanto funcionais como os não-funcionais.

3.2.1 Ciclo de Tarefas

Dentro dos diversos requisitos identificados durante o processo de análise, existe um cuja relevância para a abordagem de desenvolvimento faz com que seja tratado separadamente neste trabalho: A ciclicidade temporal que as medições devem obedecer. Dentro das atuais metodologias de medições oceanográficas, as diversas variáveis de interesse devem ser medidas de forma que o instante de medição seja correlacionado. Assim, os métodos numéricos que necessitam dos dados podem ser utilizados com maior precisão e confiança, principalmente no que tange a defasagem dos sinais.

Desta necessidade surgiu o conceito de *Ciclo de Tarefas*(também denominado Macrociclo). O ciclo de tarefas é uma forma de organizar criteriosamente o funcionamento das medições BMO. Nele, todas as atividades que serão executadas são posicionadas e configuradas a partir de informações do usuário e de configuração do sistema. Conforme explicitado em uma próxima sessão, o projeto definiu como

Tarefa(ou *Task*) o conjunto de atividades em questão e como *TaskCycle* a entidade descrita nesta seção. Na figura 3.1 é possivel observar uma ilustração de um ciclo de tarefas com duas tarefas(T1 e T2) e suas propriedades de posicionamento em relação às divisões temporais, microgerenciadas pelo SisCAT, do ciclo de tarefas. Estas propriedades estão descritas na tabela 3.2

Tabela 3.2: Propriedades de uma Tarefa.

Propriedade	**Descrição**
Offset	Deslocamento de tempo para o início da primeira execução
Período	Intervalo entre duas ocorrências da mesma Tarefa em um mesmo ciclo
Quantidade	Número de ocorrências de uma tarefa em um mesmo ciclo de tarefas

Assim, diversas tarefas podem figurar no mesmo Ciclo de Tarefas, uma vez que cada uma é configurada individualmente. Estes parâmetros devem ser programáveis de forma que os clientes utilizem o SisCAT conforme suas necessidades.

3.3 Conceitos Iniciais

Após discussões em torno do documento de requisitos, das funcionalidades que o SisCAT deve ter e familiarização com alguns instrumentos previstos para compor a BMO, alguns conceitos iniciais foram estabelecidos:

- Cada acesso efetivo a um instrumento da BMO é composto por uma *Operação* de envio, e possivelmente recebimento, de dados;
- Um conjunto de *Operações* correlacionadas é denominado *Tarefa*;
- O Ciclo de Tarefas é composto pelas tarefas registradas no Arquivo de Usuário e interpretado dinamicamente pelo sistema.
- Cada interface física é denominada por *Interface*;
- Um ou mais instrumentos podem ser conectados a uma *Interface*;

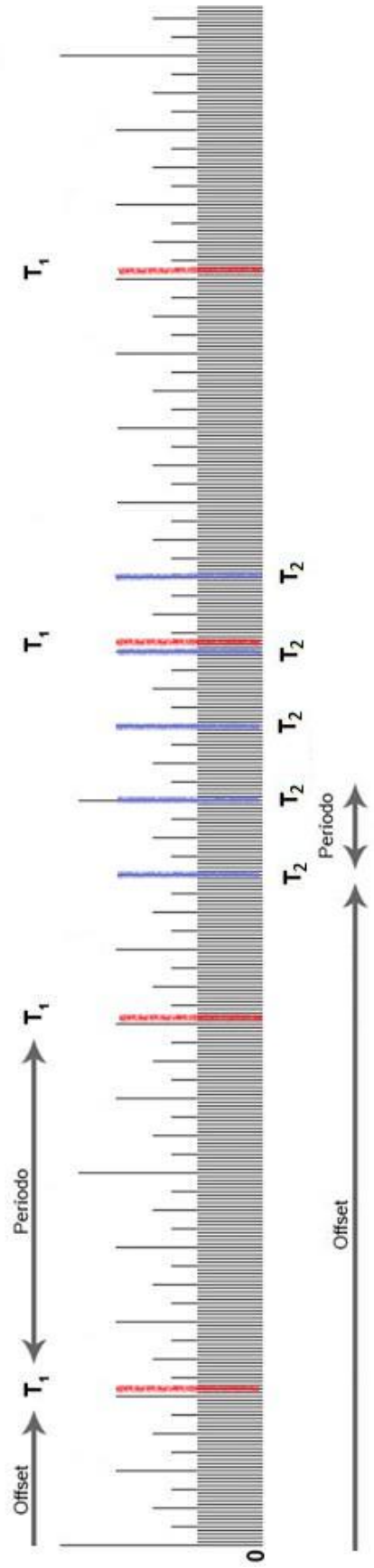

Figura 3.1: Exemplo de ciclo de tarefas(*TaskCycle*).

Com estes termos definidos e documentados, iniciou-se o processo de projeto do software usando uma alta abstração de como deve ser seu funcionamento e a relação entre as entidades identificadas. Estes conceitos permitiram o esboço do funcionamento de um protótipo que interage com instrumentos utilizando uma sequência estruturada de operações. Após esta experiência, e procurando satisfazer os requisitos de processamento de dados e compactação de dados, conceitos mais avançados foram derivados. Os conceitos definidos anteriormente foram alterados, estendidos ou repensados.

O conceito de *Interface* teve sua abstração elevada de forma a comportar entidades que gerenciam algoritmos do domínio do problema(ex.: Compactação de uma série temporal por informações de espectro) ou rotinas auxiliares, mantendo o sincronismo e o gerenciamento dos recursos adequadamente. Assim, conectados a uma *Interface* não estão somente instrumentos. O termo definido passou a ser *Componente*. Agora, tanto instrumentos quanto algoritmos específicos, desenvolvidos pela comunidade acadêmica ou pela equipe de desenvolvedores do LIOc, são organizados de forma homogênea, denominados *Componentes*, utilizados através de *Operações* e associados a alguma *Interface*. Para casos de algoritmos de processamento de dados, existe uma(ou mais) *Interface Lógica* enquanto as portas físicas denominam-se de acordo com sua natureza, tal qual *Interface Serial* ou *Interface USB*.

Para manipular todas as partes descritas, gerenciar a temporalidade e o funcionamento da BMO identificou-se a necessidade de uma entidade centralizadora de controle. O nome atribuído a tal entidade foi *Controlador*. A esta cabe o gerenciamento também do ciclo de tarefas, descrito anteriormente, e do *loop* principal de funcionamento.

Estas entidades são classes do domínio do SisCAT. A identificação dos conceitos permitiu o avanço para a fase de especificação da relação entre os mesmos. Utilizando técnicas de orientação a objetos como generalização e especialização, o Diagrama de Classes de Domínio(Fig 3.2) foi elaborado de forma a capturar e documentar esta análise. No diagrama, pode-se observar as relações[4] de Composição, Dependência, Agregação e Generalização estabelecidas entre as entidades do SisCAT.

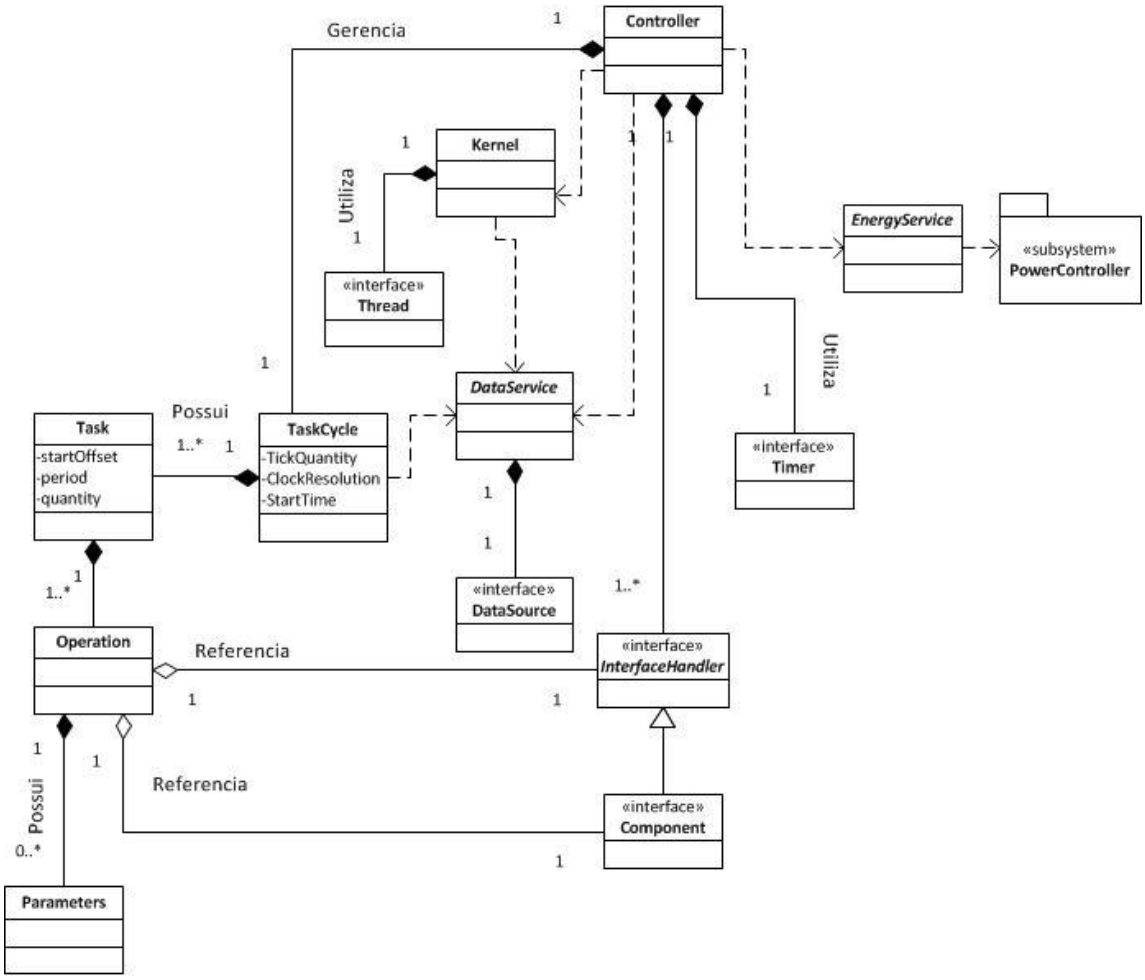

Figura 3.2: Diagrama de Classes de Domínio

3.4 Classes de domínio

Nesta seção, serão descritas as classes de domínio do diagrama(Fig 3.2) elaborado. É válido mencionar que a definição formal de um idioma especifico(neste caso, o inglês) para a modelagem evita incoerências no desenvolvimento.

3.4.1 *Controller*

Esta classe representa a entidade centralizadora de poder do SisCAT. Seu objetivo é interagir com todas as outras regulando a temporização, executando as atividades programadas e deliberando sobre fluxos alternativos. O *Controller* inicia a maior parte dos métodos chamados pois é seu papel realizar a tomada de decisão de acordo com as regras de negócio implementadas. Mantém referências para as instâncias de *InterfaceHandler's*, *TaskCycle* e *Timer*. Essa generalização permite um melhor encapsulamento das funcionalidades e também das complexidades de cada módulo com qual o *Controller* interage.

3.4.2 *Kernel*

O papel da classe *Kernel* é fazer a interface com módulo específicos da plataforma onde o SisCAT está sendo executado. No caso deste trabalho, com o sistema operacional Linux, funcionalidades que dependem de chamadas de sistema estão isoladas dentro do *Kernel*. Assim, concentra-se o acoplamento do SisCAT com o Linux em somente um ponto de contato. Alguns exemplos de chamadas de sistema encapsuladas pelo *Kernel* são funções do Linux como criação e gerenciamento de thread's ou acesso a informações computacionais como desempenho, recursos e identificação.

3.4.3 *Timer*

A interface *Timer* é acessada pelo *Controller* de forma que a temporização do SisCAT seja gerenciada com a precisão temporal que o sistema exige. Através desta interface, é possível criar, pausar, reiniciar e reprogramar o *Timer* que a implementa. O *Timer* deve ter resolução suficientemente pequena para que as subdivisões do *TaskCycle*(Ciclo de Tarefas) possam ser gerenciadas individualmente.

3.4.4 *TaskCycle*

A classe *TaskCycle* representa a organização de *Task's* configurada pelo usuário e pelo sistema. Além da distribuição de tarefas, informações sobre a resolução do intervalo mínimo e de um tempo reservado para atividades internas do SisCAT, como compactação de dados ou transmissão, estão presentes. Estas informações constam nos arquivos de configuração, seja do usuário ou de sistema, de forma que ambos são necessários para configurar a operação do SisCAT e programar o *TaskCycle*.

3.4.5 *Task*

Como detectado na fase de análise de requisitos, o conjunto de atividades é organizado e chamado de Tarefa. Estas atividades são divididas em instâncias de *Operation's* e são agrupadas, ordenadas e gerenciadas pela entidade *Task*(do inglês, Tarefa). Além das operações, uma *Task* precisa armazenar informações que possibilitem seu posicionamento adequado, incluindo as da tabela 3.2.1.

3.4.6 *Operation*

Dentro da modelagem atingida para o SisCAT, a classe *Operation* faz a ligação situacional entre um *Component*, um *InterfaceHandler* e uma instância de *Parameters*. Em outras palavras, a classe *Operation* atua como o conjunto de informações necessárias para uma utilização pontual do SisCAT. Seus principais atributos estão na tabela 3.3.

Tabela 3.3: Campos de um *Operation*

Campo	**Descrição**
InterfaceHandler	Referência para o *InterfaceHandler*
Component	Referência para o *Component* que executará a operação
Parameters	Parâmetros de execução da operação

3.4.7 *Parameters*

Esta classe representa a estruturação dos parâmetros a serem utilizados em uma dada operação(*Operation*). Sua implementação permite acessar tanto os valores

quanto os nomes dos parâmetros. Neste protótipo, esta classe é implementada contando com as facilidades da classe *std::map* pertencente à biblioteca padrão C++.

3.4.8 *InterfaceHandler*

A classe *InterfaceHandler* é a entidade responsável por gerenciar e controlar o acesso a recursos compartilhados entre os mesmos componentes. Estes recursos podem ser o acesso a uma porta ou a um GPIO. Esta organização é importante para evitar problemas de concorrência em que dois *Component's* acessam ao mesmo tempo uma determinada porta resultando em um comprometimento das informações trocadas. Retirar essa responsabilidade do sistema operacional diminui ainda mais a dependência e o acoplamento entre o SisCAT e o Linux.

3.4.9 *Component*

Dentro do domínio do SisCAT, cada módulo cuja funcionalidade é programável pelo usuário é representado de forma generalizada como um *Component*. Estes módulos podem ser instrumentos de medição, módulos de cálculos matemáticos ou qualquer funcionalidade relevante que venha a ser adicionada conforme necessidade. A interface enxuta da classe base *Component* garante a versatilidade necessária para permitir reuso destes módulos e o baixo acoplamento permite que eles sejam desenvolvidos em paralelo, otimizando o trabalho em equipe. Os *Component* são registrados em seus respectivos *InterfaceHandler's* através de um método virtual de inicialização.

3.4.10 *DataService*

O *DataService* é a entidade que gerencia toda a parte de armazenamento e sistema de arquivos(caso exista). O sufixo *Service* é utilizado para frisar que tal classe pode ser acessada em carater de "serviço", através de métodos estáticos, de forma que a gerência de instâncias é feita internamente. Entre as funções do *DataService* estão administrar o resultado das operações, salvando em memória não volátil quando solicitado pelo *Controller* e realizar back-up das amostras salvas. Seu principal atributo é a interface *DataSource*, explicado a seguir. Concentrar todas as funcionalidades de

dados e armazenamento nesta entidade permite a adição de novas funcionalidades com uma relativa facilidade, uma vez que, assim, há um desacoplamento explícito.

3.4.11 *DataSource*

Esta interface é utilizada para tornar o SisCAT independente de uma abordagem específica de armazenamento de dados. Desta forma, diferentes utilizações poderão contar com implementações específicas de sistemas de arquivo, armazenamento e controle de dados. Exemplos de abordagens podem ser a utilização de arquivos de texto para armazenamento de dados ou sistema de arquivos em rede(NFS) para armazenamento remoto.

3.4.12 *EnergyService*

Com a mesma filosofia do *DataService*, o *EnergyService* concentra todas as funcionalidades relacionadas ao gerenciamento de energia da BMO. Como a operação autônoma do SisCAT precisa realizar tomadas de decisão em relação à consumo, disponibilidade e planejamento energético, é natural uma entidade fornecer tais informações em uma interface padrão, auto-contida e especializada.

3.4.13 *Thread*

Uma Linha de execução, ou *thread* em inglês, é um conjunto de instruções independente cujo momento da execução é gerenciado por um escalonador, normalmente de um sistema operacional. Esta divisão se da através de algoritmos de multiplexação de tempo e permite atingir uma execução virtualmente paralela. Usando esta abordagem, o SisCAT aloca uma *thread* para cada *Task* sendo executada. A interface *Thread* do projeto permite isolar a interação com o escalonador, a configuração do algoritmo de multiplexação de tempo e a biblioteca de *threads* utilizada.

Esta interface, no sentido de orientação a objetos, permite ao SisCAT o gerenciamento de diversas *threads* necessárias ao seu funcionamento. Seus métodos virtuais contemplam operações para criação, deleção, cancelamento e sincronismo entre as *threads*.

3.4.13.1 Controlador de Alimentação

Uma das dependências do *EnergyService* é o Controlador de Alimentação. Como observado na figura 3.3, o *PowerController*(Controlador de Alimentação) é caracterizado como um subsistema a parte. Este é um projeto futuro do LIOc que visará atuar sobre a alimentação individual de cada instrumento ou periférico da BMO de forma a otimizar o consumo energético. Para esta versão do protótipo, é suficiente modelar considerando a existência de tal entidade.

3.5 Classes especializadas

O projeto do SisCAT utiliza o paradigma de orientação a objetos[4]. Com isso, o ferramental disponível para organização das entidades e das relações permite atingir uma especificação que atenda requisitos de portabilidade e reutilização. Assim, para a mesma infraestrutura descrita anteriormente, diversos instrumentos e sensores podem ser utilizados, bibliotecas de *thread's* ou implementações de armazenamento de dados podem ser testadas, tudo com o mínimo impacto nos algoritmos principais do SisCAT. Nesta seção serão descritas as implementações especializadas(Fig 3.3) utilizadas neste protótipo.

3.5.1 *InterfaceHandler* - SerialInterfaceHandler

Esta especialização torna possível o gerenciamento de portas seriais pelo SisCAT. Como os requisitos preveem a necessidade de diversas portas deste tipo, a possibilidade de gerenciá-las individualmente, dentro de uma arquitetura organizada, é muito útil. A um *SerialInterfaceHandler* podem ser associados instrumentos ambos da rede RS485 quanto de interface RS232. O controle de acesso à porta e feito com Mutexes (técnicas de exclusão mútua entre linhas de execução concorrentes) pela própria classe, garantindo o acesso organizado a seções críticas do código. Como explicado no capítulo 2, a grande maioria dos instrumentos utilizados na BMO possui interface serial como meio de comunicação, justificando a criação desta classe.

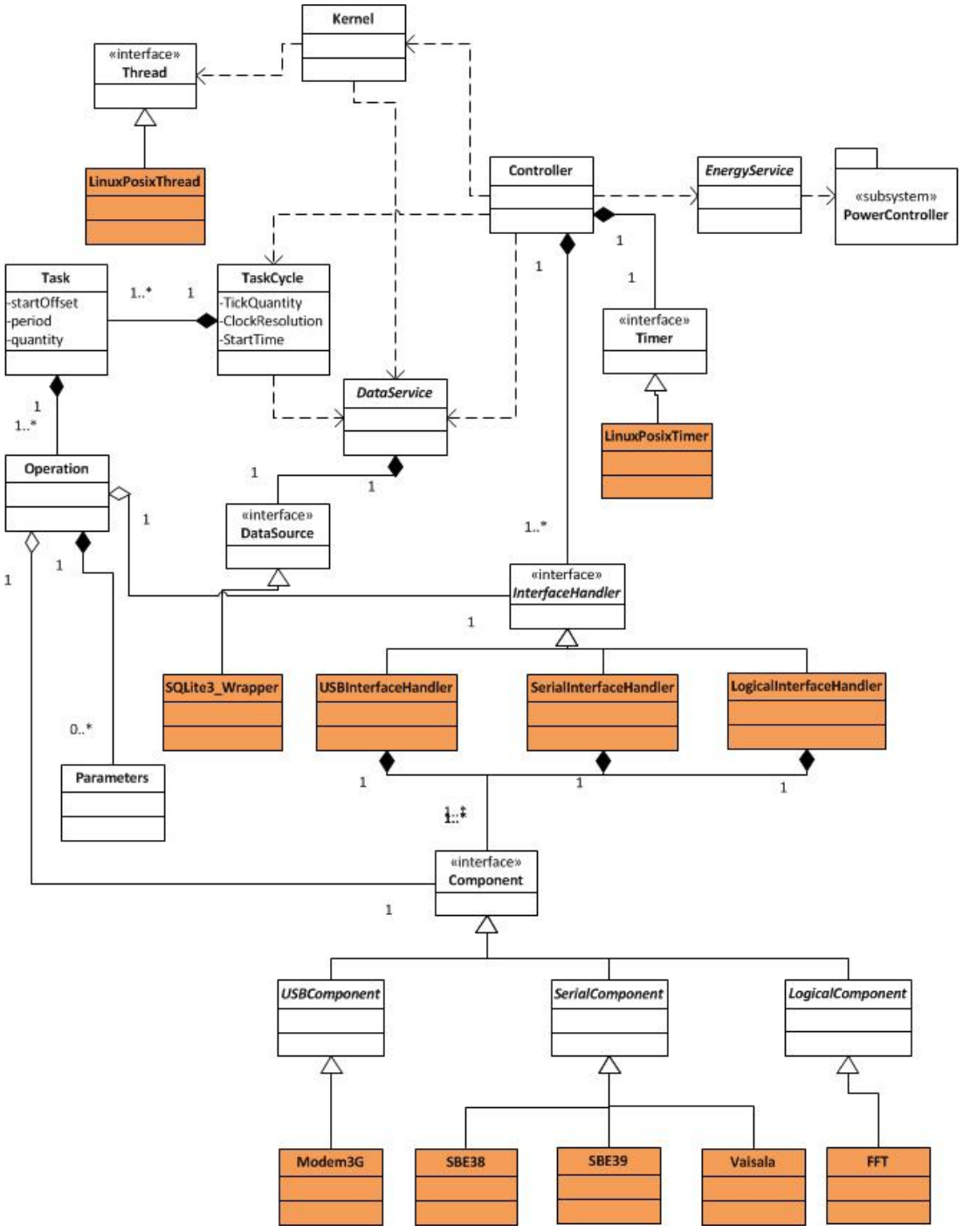

Figura 3.3: Diagrama de Classes de Domínio com implementações específicas

3.5.2 *InterfaceHandler* - LogicInterfaceHandler

Estão previstas atividades de cálculo computacional relativamente intenso (compactação de dados ou operações matemáticas) e o tempo de processamento dedicado a estas operações pode influenciar no consumo de energia e na performance do SisCAT como um todo, o *LogicInterfaceHandler* gerencia justamente este recurso. Assim, o usuário tem a possibilidade de programar estas atividades de acordo com suas necessidades.

3.5.3 *Component* - SerialComponent

Para utilizar instrumentos ligados às portas seriais, os parâmetros internos necessários para configuração e operação são encapsulados na especialização *SerialComponent*. Esta classe, no entanto, é abstrata. Rotinas de temporização interna, detecção de baud rate, configuração de *stop* bits e bits de paridade são implementadas já e unicamente nesta classe, deixando somente a tradução dos comandos seriais da estrutura interna de parâmetros do SisCAT para os comandos próprios de cada instrumento. A figura 3.4 exibe a árvore de especialização da versão atual do SisCAT, com ênfase nas especializações da interface *Component*.

3.5.4 *Component* - LogicComponent

Esta especialização da classe *Component* faz com que algoritmos específicos e rotinas relacionadas às medições sejam encapsuladas e executadas de acordo com a programação escolhida pelo usuário. Diferente do *SerialComponent*, esta classe atua apenas como uma fina camada que diferencia os componentes lógicos dos de outros tipos.

3.5.5 *DataSource* - SQLite3

Para o armazenamento não-volátil de dados, o SisCAT conta com uma banco de dados relacional enxuto e próprio para aplicações com poucos recursos computacionais[5]. O SQLite3[6] implementa toda a lógica transacional de um banco de dados comum através da sua biblioteca que, ligada ao executável pelo processo de *linkagem*, permite operações de leitura e escrita em um único arquivo. Atualmente é utilizado em

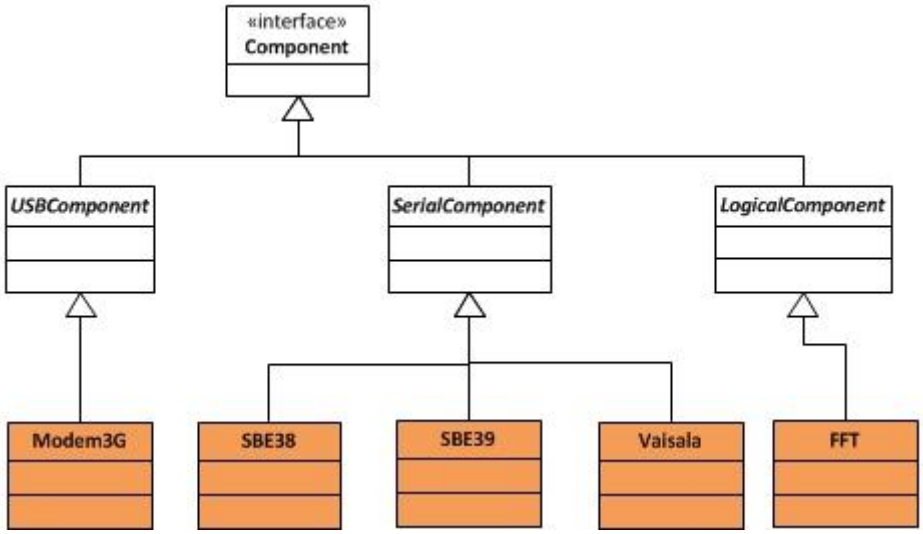

Figura 3.4: Especialização - Classe *Component* e exemplos de subclasses

telefones celulares, PDA's e aplicações de telemetria. O modelo de dados utilizado é simples e poderoso, pois mantém a relação entre as amostras, operações, interfaces etc. A figura 3.5 exibe o modelo utilizado para este trabalho.

3.5.6 *Thread* - Linux POSIX Thread

A implementação da interface *Thread* utilizada neste trabalho foi baseada na biblioteca *pthreads*[7](POSIX Threads). Seu uso é amplamente incentivado em ambientes Linux.

3.5.7 *Timer* - Linux POSIX Timer

Para esta versão do *Timer* do SisCAT, o classe especializada *LinuxPosixTimer* faz uso das funções do padrão POSIX que, assim como no caso das threads, está tradicionalmente presente em ambientes Linux[8]. Esta vantagem permite acelerar o desenvolvimento e alcançar resultados palpáveis mais rapidamente. No entanto, a implementação final do SisCAT prevê a utilização exclusiva dos temporizadores via hardware. Devido ao desacoplamento alcançado com a modelagem descrita, é possível intercambiar as implementações. Assim, uma vez especificado o temporizador via hardware a ser utilizado(independe do sistema operacional), o impacto da

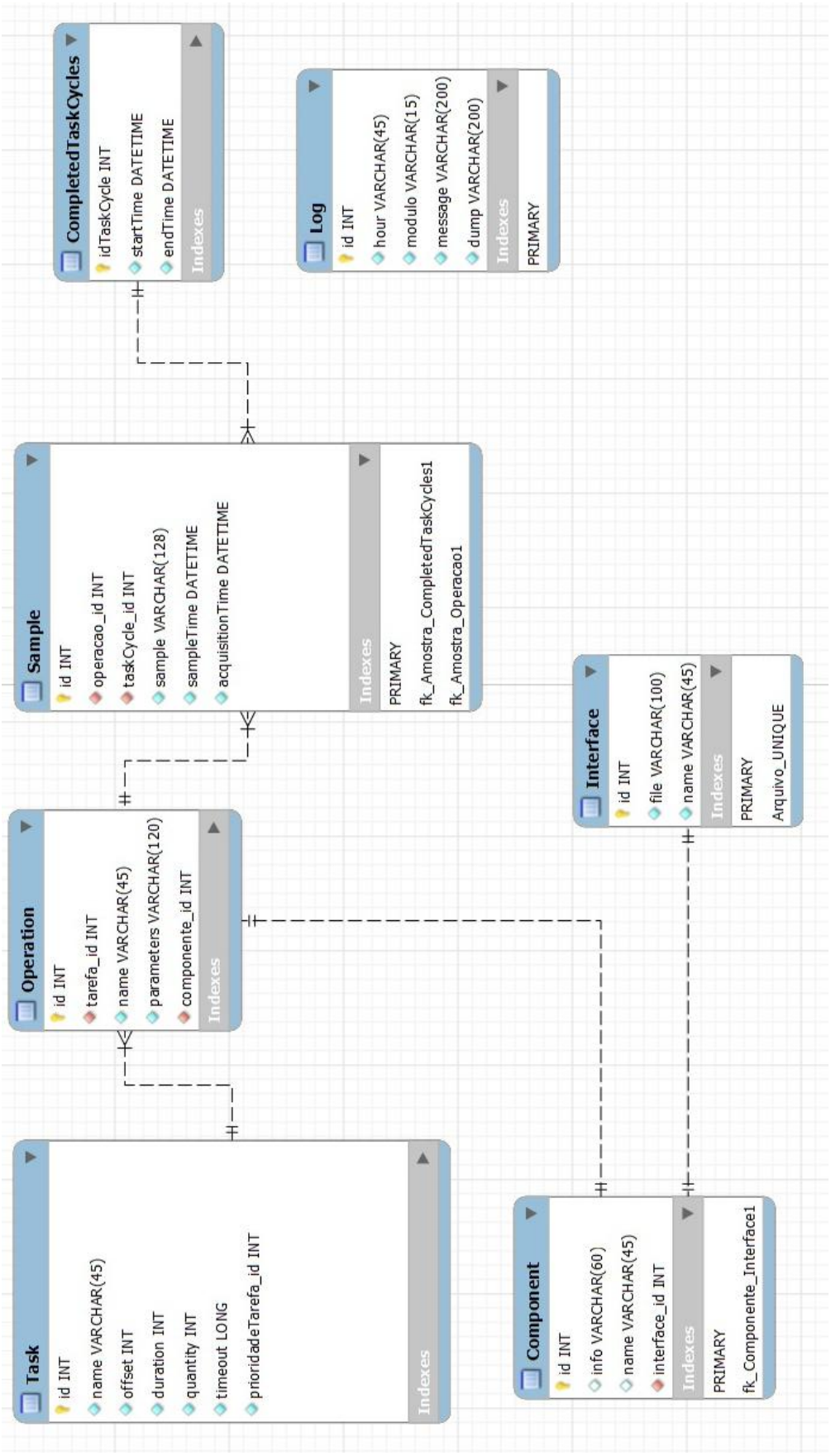

Figura 3.5: Modelo de dados SQLite3 para o SisCAT

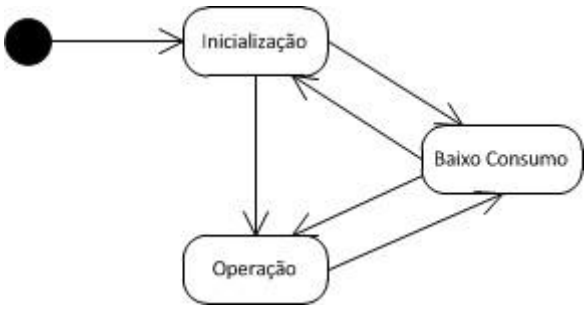

Figura 3.6: Estados do SisCAT

troca será minimizado.

3.6 Algoritmos de controle

O SisCAT opera em basicamente três estados(Fig 3.6): Inicialização, Operação e Baixo Consumo. O primeiro estado, *Inicialização*, é o estado o qual o sistema se encontra enquanto realiza a sequência de verificações de integridade de componentes, periféricos e disponibilidade mínima de energia. Após esta etapa, ocorre a transição para o estado *Operação*, que é quando ocorrem as atividades de medição, processamento, armazenamento, telemetria etc. Em ambos os estados, caso ocorra uma situação de escassez energética, o SisCAT programa-se para entrar no estado de *Baixo Consumo* até que haja energia disponível novamente.

3.6.1 Inicialização

O SisCAT inicia sua execução realizando a sequência de inicialização descrita adiante. Nesta etapa o sistema procura realizar uma verificação de suas principais partes, configurar-se e preparar-se para a operação.

3.6.1.1 Verificar disponibilidade de energia

A primeira etapa é a verificação da disponibilidade de energia. Este procedimento ocorre de imediato. A interação entre as classes é mostrada no diagrama da figura 3.7 para o caso de sucesso. Caso não haja energia suficiente para o início da operação,

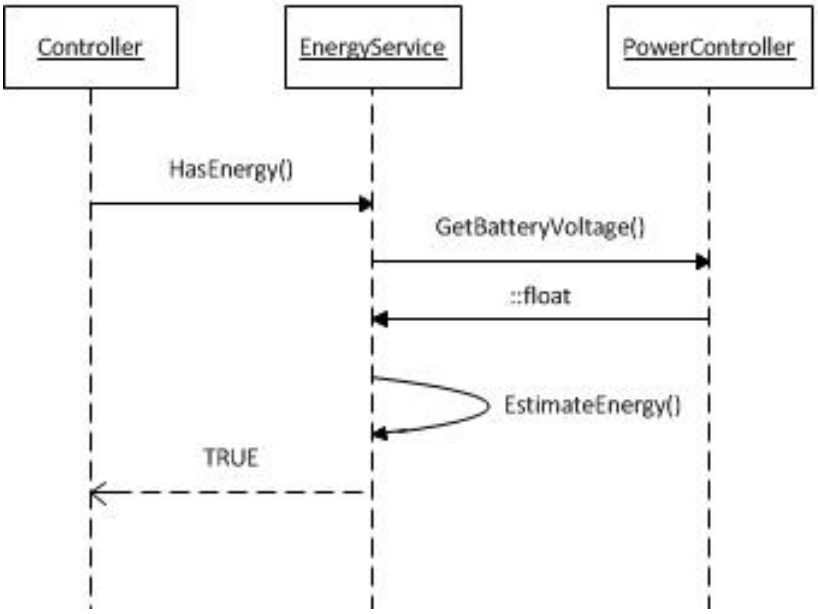

Figura 3.7: Verificação de energia

o SisCAT programa-se para hibernar até um momento futuro, quando novamente este procedimento será executado.

Como mencionado anteriormente, o subsistema *PowerController* atua sob caráter de simulação. A funcionalidade de verificar a tensão das baterias e dos painéis é implementada diretamente no *EnergyService*

3.6.1.2 Verificar integridade do armazenamento não-volátil

Nesta etapa, o *DataSource* é questionado sobre o estado do seu esquema de armazenamento não volátil. No caso deste trabalho, a utilização do SQLite3 como armazenamento permite acessar as próprias funções embutidas em sua biblioteca para este fim. O próprio motor de gerenciamento de banco de dados faz a verificação de inconsistências e de entradas inválidas. O trecho do diagrama sequencial que contempla tal atividade é mostrado na figura 3.8.

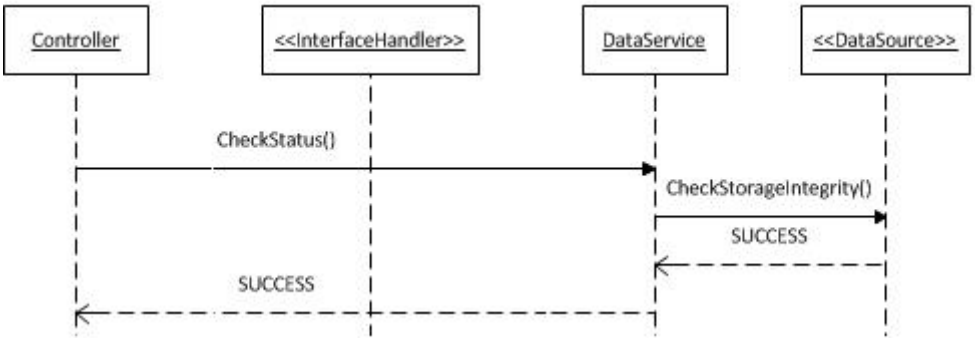

Figura 3.8: Verificação do armazenamento

3.6.1.3 Verificação dos *InterfaceHandler's* e dos *Component's*

Para verificar o estado de cada *InterfaceHandler* faz-se uma chamada para o método virtual *InterfaceHandler::checkStatus()*. Este método deve ser obrigatoriamente implementado pela classe especializada. Neste ponto, o comportamento varia de acordo com a especialização. Por exemplo, um *SerialInterfaceHandler* verifica a conectividade com a porta física associada, um *USBInterfaceHandler* verifica pacotes USB de status e o *LogicInterfaceHandler*, por ser um "organizador" puramente virtual, não demanda maior atenção.

Ainda nesta etapa, para cada *InterfaceHandler*, ocorre a execução do método virtual *Component::checkStatus*, referente a cada componente registrado, e sua resposta é analisada. Neste ponto o *Component* pode ser marcado como impróprio para operação ou desligado conforme o resultado da verificação. Neste caso, também, o comportamento é implementado de acordo com a natureza do *Component*: Um instrumento microprocessado(*SerialComponent*) pode contar com funções de autoverificação enquanto um módulo de cálculo de parâmetros direcionais de onda (*LogicComponent*) pode ter seus algoritmos numéricos testados para valores conhecidos. A figura 3.9 exibe os dois *loops* de execução para o caso de sucesso.

3.6.1.4 Importar arquivo de sistema e de usuário

Esta fase da inicialização faz a leitura e a interpretação(Fig 3.10) das informações encontradas no *Arquivo de Sistema*. Este arquivo, estruturado, representa a confi-

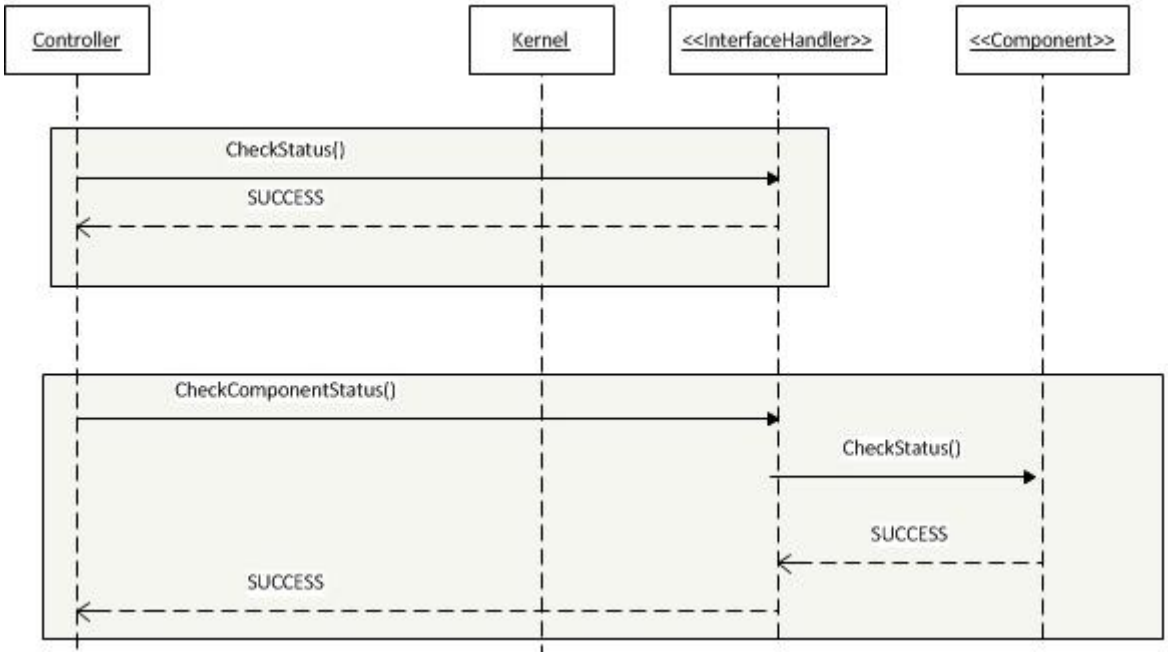

Figura 3.9: Verificação das interfaces e dos componentes

guração atual que a BMO deve ter. Nele constam informações de configuração das redes de sensores, como Baud Rate, ou parâmetros globais de operação do SisCAT, como tempo de *timeout* máximo para cada operação. Esta configuração, após lida, é validada pelo método *Kernel::ParseConfigurationFile*, onde os parâmetros serão individualmente analisados.

De forma semelhante, o *Arquivo de Usuário* conta com alguns parâmetros configuráveis pelo usuário além das informações do ciclo de tarefas, operações e parâmetros de medição. A leitura se dá de maneira praticamente igual(Fig 3.11), uma vez que os arquivos utilizam as mesmas regras de estruturação.

3.6.1.5 Montagem do *TaskCycle*

A última etapa da sequência de inicialização é a montagem do *TaskCycle*, o ciclo de tarefas. Conforme descrito, diversas *Task's*, que agrupam *Operation's* e *Parameter's*, são organizadas de forma a realizar todas as medições, e outras atividades, programadas pelo usuário. Estas informações constam também no *Arquivo de Usuário* e são convertidas para objetos das classes do SisCAT conforme leitura(Fig 3.12).

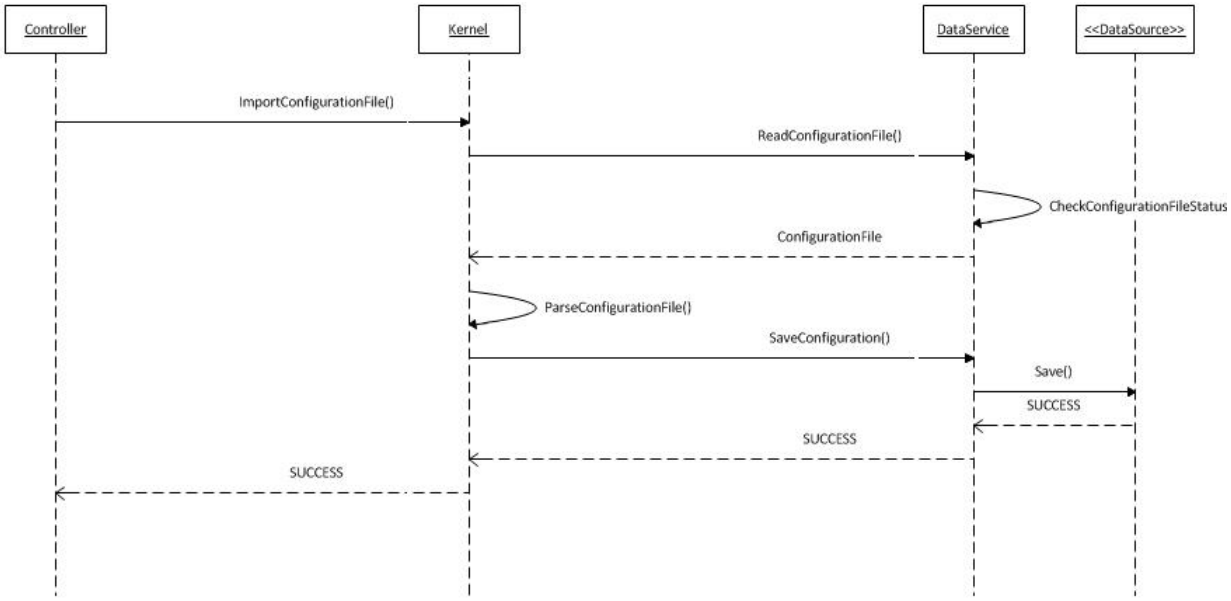

Figura 3.10: Leitura do arquivo de configuração

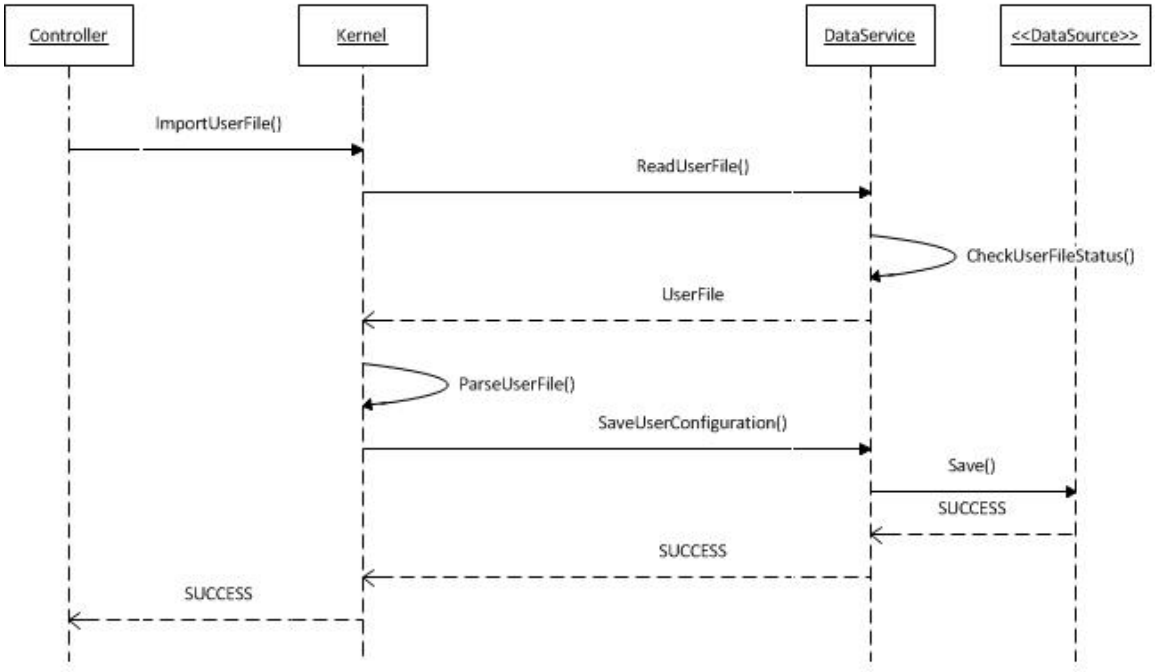

Figura 3.11: Leitura do arquivo de configuração

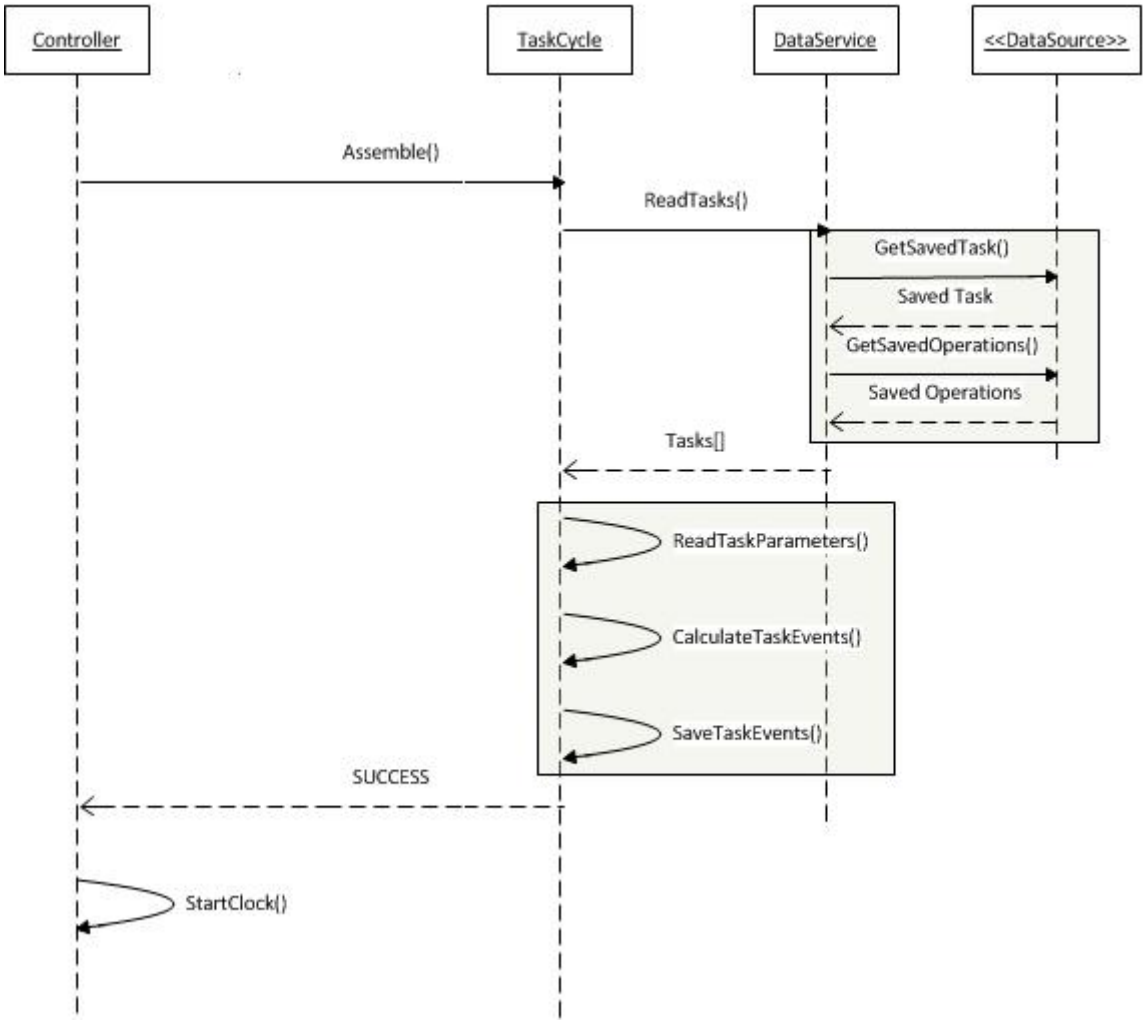

Figura 3.12: Montagem do ciclo de tarefas

Assim, com as informações que existem em cada objeto, monta-se a sequência de execução de forma dinâmica. Uma vez preparado o ciclo de tarefas, o SisCAT faz a transição para o estado de operação.

3.6.2 Operação

O estado de operação do SisCAT é composto pelo *loop* infinito do processo e pelas *Thread's* criadas para a execução das operações de cada tarefa, conforme temporização. Ao fim do ciclo de tarefas, uma janela de tempo é reservada para operações internas do SisCAT, como transmissão de dados, verificações de energia ou back-up's.

3.6.2.1 A temporização e o lançamento de uma *Thread*

Para ter um preciso domínio sobre a temporização de um *TaskCycle*, o *Controller* programa o *Timer* de acordo com o tamanho da subdivisão temporal deste ciclo. Este tamanho é calculado baseado nas informações do *Arquivo de Usuário*, dentro dos limites de operação especificados no *Arquivo de Configuração*. Uma vez programado e em execução, o *Timer* executará o método indicado pelo *Controller* na hora de sua criação, a cada expiração. Atualmente, trata-se de um método estático da classe *Controller* que é passado como ponteiro no construtor do *Timer*. A figura 3.13 ilustra os passos de busca por uma *Task* e lançamento de uma *Thread*.

3.6.2.2 A execução de uma *Task*

O trecho de código executado na *Thread* lançada para uma determinada *Task* percorre todas as *Operation's* na ordem especificada, executando-as uma a uma. Para cada *Operation*, o método *InterfaceHandler::Execute* é chamado tendo como argumentos a referência para o *Component* em questão e o objeto *Parameters* desta *Operation*. Com estas informações, a instancia do objeto *Component* que executará a *Operation* tem o método *Component::Execute* chamado. Assim, no caso de sucesso, o resultado é armazenado pelo *DataService*, que age de acordo com a política escolhida, podendo salvá-lo imediatamente em um meio não-volátil através do *DataSource* ou aguardar até o fim do *TaskCycle* para efetuar o armazenamento. A figura 3.14 exibe em detalhes a sequência de execução.

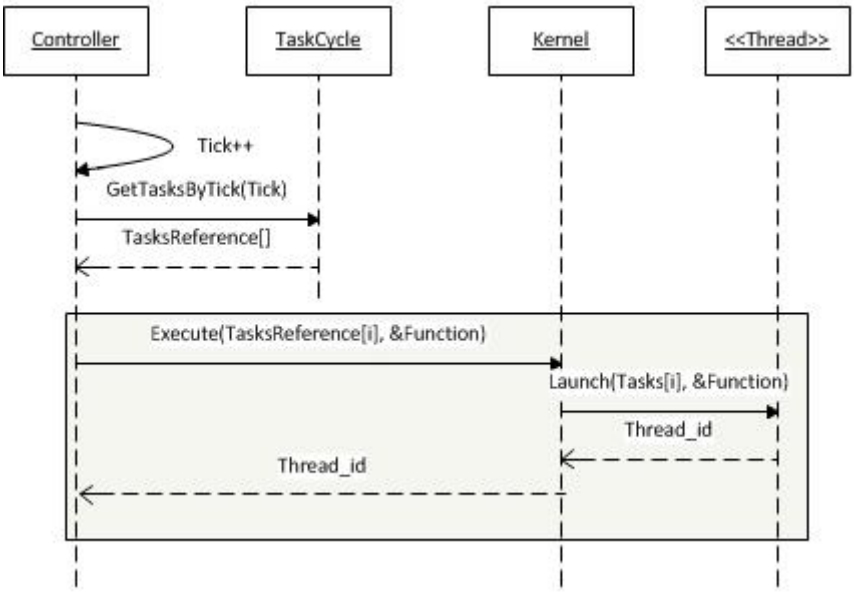

Figura 3.13: Lançamento de uma *Thread* para execução de uma *Task*

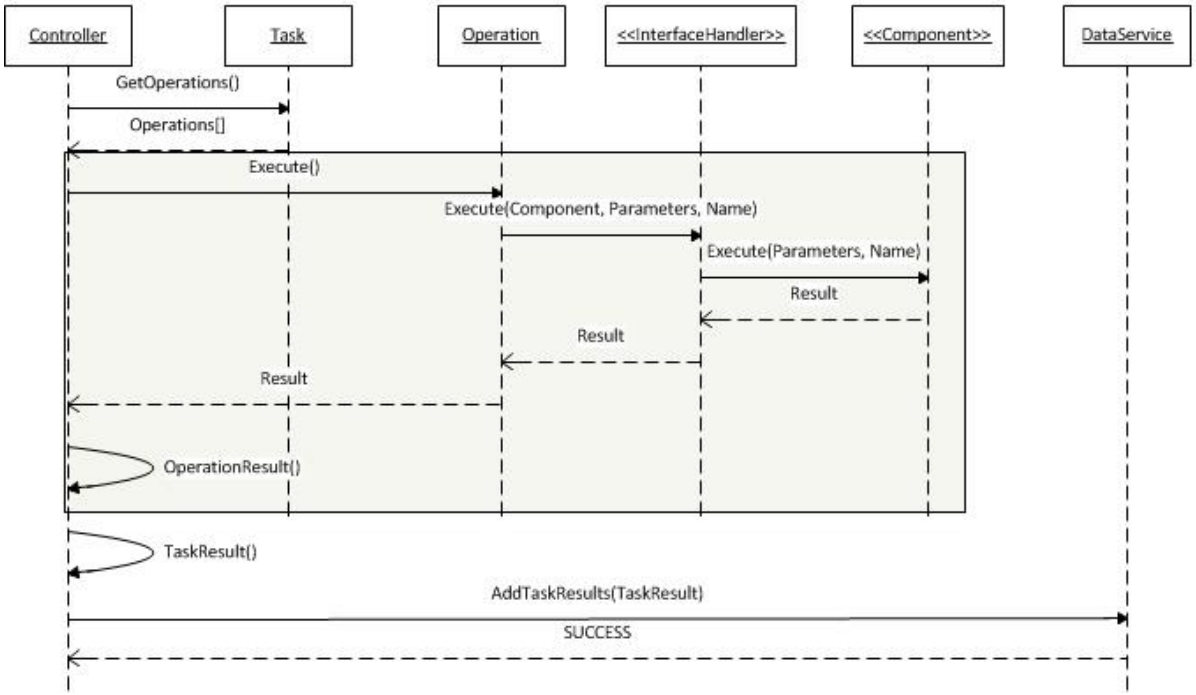

Figura 3.14: Execução de *Operation's* pertencentes à uma *Task*

Capítulo 4

A implementação do protótipo

Esta sessão descreverá os detalhes de implementação inerentes ao protótipo desenvolvido. Diferente da sessão que trata de modelagem e *design* do SisCAT, os tópicos aqui descritos procuram detalhar as opções de projeto e tecnologias adotadas para esta versão do protótipo.

4.1 Plataforma computacional de baixo consumo

Devido à limitada disponibilidade de energia na BMO, é necessário utilizar um módulo computacional eficiente e de baixo consumo. Na tendência contrária, o poder computacional esperado deve ser compatível com o software em execução. Após comparação de diversas opções, o computador escolhido para a fase inicial de prototipagem foi um *Single Board Computer*(SBC) com as seguintes características:

- Processador Atmel AT91SAM9G20
- 512 MB de memória Flash
- 64 MB de memória RAM
- 2 Portas RS232
- 1 Porta RS485
- 53 GPIO's
- USB, Ethernet, MicroSD

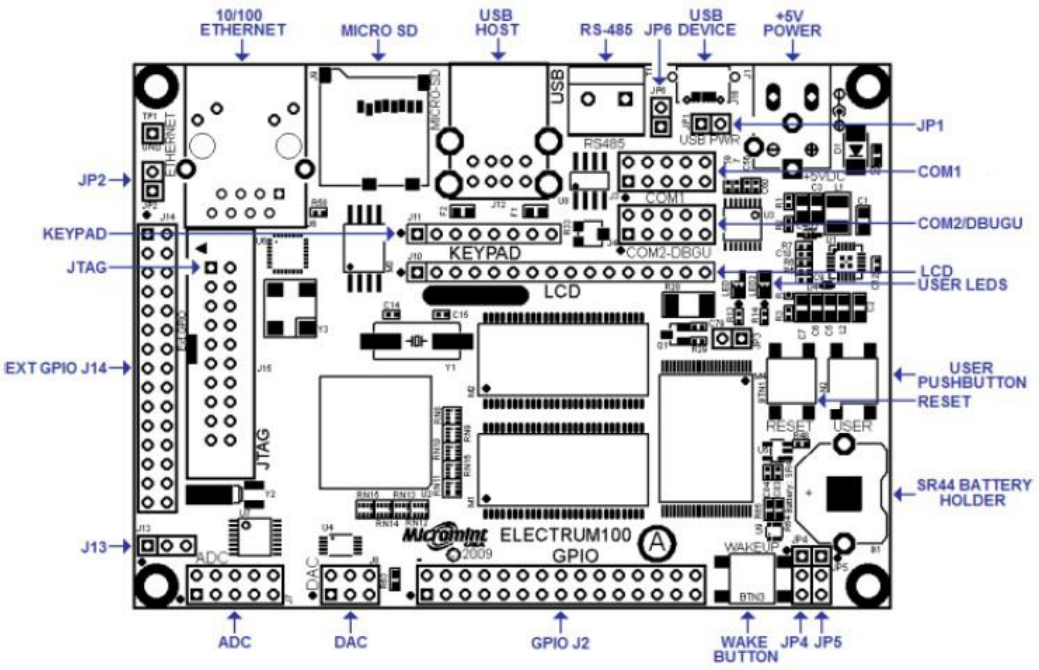

Figura 4.1: Detalhes de interface do *Single-Board Computer* MicroMint Electrum 100

O processador Atmel AT91SAM9G20[9] é um *System On a Chip*(SOC) baseado no núcleo ARM926 de 32bits, opera com uma frequência de 400MHz e possui uma extensa gama de periféricos(4.1), muitos dos quais se tem acesso através dos GPIO's programáveis.

O grande atrativo do SBC utilizado é o baixíssimo consumo em plena operação que, sendo menor que 1W, permite dedicar recursos energéticos da BMO para outros periféricos e subsistemas. Uma outra vantagem, descrita em detalhes em outra seção, é a disponibilidade do sistema operacional Debian(Linux). Contar com ferramentas consagradas de uma distribuição Linux para o protótipo adiciona um nível elevado de confiança além de inúmeras facilidades.

4.2 Linux embarcado

A função de um sistema operacional é gerenciar os recursos de hardware e software corretamente e de forma otimizada. Como um computador moderno é composto por diversos periféricos, cada um com seu protocolo e especificação técnica associada,

esta tarefa não é nada menos que árdua. No contexto deste trabalho, a plataforma computacional conta com periféricos que, necessários à implementação de todas as funcionalidades requisitadas e, pela criticidade da aplicação, devem operar livre de falhas e com elevada robustez. O desenvolvimento de firmware para interação com hardware é uma atividade com alta complexidade e um enorme risco associado. Faz-se necessário mencionar que, além do firmware citado anteriormente, a implementação de protocolos, como a pilha TCP/IP ou o acesso à memória *flash* por exemplo, tem complexidade igual ou superior.

Frente a este desafio, optou-se por utilizar o sistema operacional fornecido pelo fabricante do SBC: Uma distribuição de Linux, baseada em Debian, com *drivers* próprios para os periféricos. Vantagens de usar Linux embarcado, ou do inglês *Embedded Linux*, são contar com um sistema operacional maduro, de livre acesso ao código fonte e com excelente suporte da comunidade *open-source*[10]. Além destas vantagens, utilizar o Kernel desenvolvido e mantido pelo fabricante do SBC acelera o tempo de desenvolvimento do SisCAT, uma vez que todos os *drivers* dos periféricos do SBC estão implementados, testados e atualizados. E finalmente, a familiaridade da equipe de desenvolvimento com o Linux e a presença das principais ferramentas tanto no SBC quanto na estação de trabalho permitem um elevado grau de produtividade e confiança no desenvolvimento do SisCAT.

4.3 Ambiente de desenvolvimento

Desenvolver software para ser executado em um computador embarcado requer uma série de medidas e ferramentas. Primeiramente, deve-se explicitar que o código compilado para o computador principal, onde o desenvolvimento é feito e denominado *host*, pode não ser compatível com o SBC, denominado *target*. Isso ocorre quando o conjunto de instruções utilizado por uma arquitetura de CPU não é implementado pela outra. No caso deste protótipo, o computador *host* é baseado na arquitetura x86 enquanto o SBC dispõe de um processador ARM. Ambos contam com o sistemas operacional Linux e, portanto, existem as seguintes opções para o desenvolvimento:

- Transferir o código escrito no *host* para o *target* e compilá-lo utilizando suas próprias ferramentas.

- Escrever o código diretamente no *target*(pelo editor de texto VI, por exemplo) e compilá-lo, assim como da forma anterior, utilizando suas próprias ferramentas.

- Escrever o código no *host* e utilizar de ferramentas específicas(*toolchain*) para compilar este código para o *target*.

As duas primeiras opções são possíveis, porém, sofrem com a limitação de recursos computacionais do sistema embarcado. Utilizá-las implica em abrir mão de ferramentas de desenvolvimento e auxílio que somente o computador *host* dispõe, como um ambiente de desenvolvimento integrado(*Integrated Development Environment*-IDE), elevado poder computacional e rápida conexão com a internet, para nomear algumas.

Assim, a abordagem escolhida foi a terceira opção. Esta abordagem consiste em escrever o código em um computador *host*, utilizar ferramentas de compilação cruzada (*toolchain*) para gerar arquivos executáveis e transferi-los para o próprio SBC. Essa transferência pode ser feita por uma conexão serial, através da porta RS232, ou por sessões de SSH/Telnet, através da porta Ethernet. A figura 4.2 demonstra o esquema utilizado no projeto.

O *toolchain* utilizado foi o Sourcery CodeBench Lite[11], que é *open-source* por basear-se no *GNU C Compiler*(GCC). Este *toolchain* consiste em compiladores C e C++, *assembler* e *linker*, além de um *debugger*. Como tanto o SBC quanto o computador principal utilizam o sistema operacional Linux, é possível utilizar a maior parte do código em ambos os sistemas apenas "chaveando" o *toolchain* utilizado. Assim, pode-se desenvolver, testar ou realizar sessões de *debug* com praticamente a mesma base de código que será compilada para o computador *target*. As interações com sensores ou hardware podem ser simuladas devido ao baixo acoplamento atingido entre as classes. No entanto, apesar de útil, utilizar o computador *host* para testes não permite identificar os problemas externos à lógica do software como, por

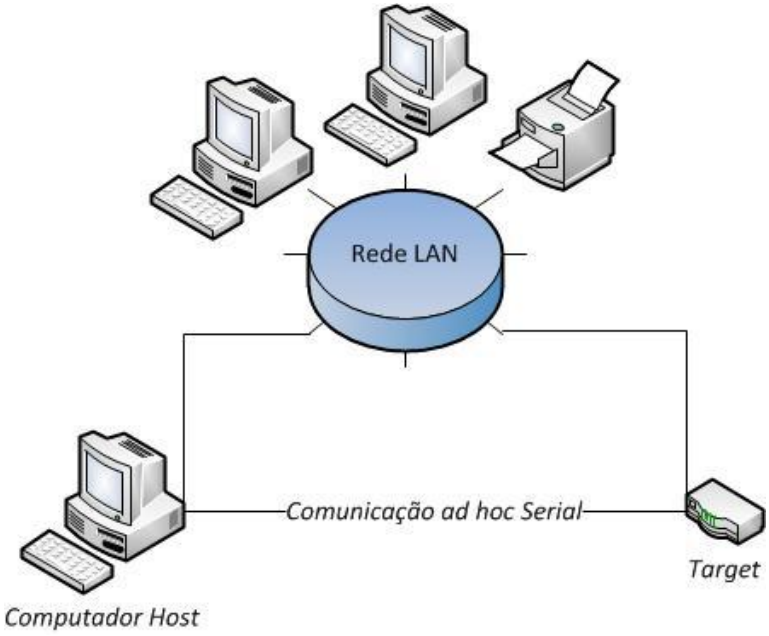

Figura 4.2: Conexão entre o computador *host*, *target* e a rede LAN do LIOc

exemplo, os impactos da latência no acesso à memória *flash* na performance do sistema embarcado em comparação ao relativamente imediato acesso ao disco rígido no computador principal.

A maioria das ferramentas utilizadas no ambiente de desenvolvimento do SisCAT são integradas a um único programa: Eclipse, da Eclipse Foundation. Um ambiente de desenvolvimento integrado(*Integrated Development Environment* - IDE) é uma ferramenta importantíssima para organizar o trabalho que está sendo feito e aumentar a produtividade da equipe. O Eclipse é uma ferramenta *open-source* e uma das mais utilizadas IDE's do mundo, características que justificaram a rápida adoção no LIOc. Originalmente criado para desenvolvimento de aplicações Java, inúmeros plug-ins extendem suas funcionalidades. Em especial, o plug-in C/C++ Development Tools(CDT) permite desenvolvimento em C/C++(figura 4.3). Além de gerenciar o uso dos *toolchains*(tanto o GCC quanto o Sourcery CodeBench) de forma configurável, a ferramenta conta com diversas facilidades como:

- Grafo de chamadas entre funções e métodos.
- Expansão de Macros definidos com a diretiva #DEFINE.

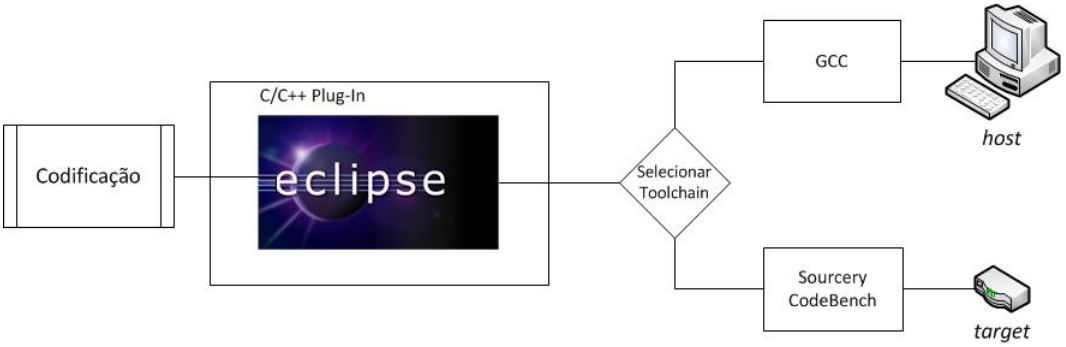

Figura 4.3: Ilustração da IDE Eclipse e o uso de dois *toolchains* distintos

- Navegação pelo código fonte.
- Geração automática de trechos de código

Também de exaltada importância para o ambiente de desenvolvimento, *Shell Scripts* permitem a automatização de manipulação de tarefas relacionadas especificamente ao sistema operacional Linux. Por exemplo, transferir os executáveis gerados pelo *toolchain* para o SBC automaticamente para serem testados ou realizar *back-up* automático dos dados adquiridos nos testes.

4.4 Linguagem de programação: C/C++

A linguagem de programação utilizada para o desenvolvimento do protótipo foi a clássica combinação C/C++. Os fatores que levaram a essa decisão foram a sua proximidade com o hardware, permitindo interagir com periféricos de baixo nível, e as praticidades de uma linguagem alto nível, como suporte ao paradigma de orientação a objetos e a biblioteca padrão STL(*Standard Template Library*). Estas vantagens são determinantes para a finalidade em questão: computação embarcada em uma arquitetura "não convencional", com requisitos de performance, reutilização de componentes e compilação cruzada.

Esta proximidade com o hardware fez com que *toolchains* para diversas arquiteturas de CPU fossem desenvolvidos, tanto comercialmente como pela comunidade

open-source. A seção dedicada à descrição do ambiente de desenvolvimento trata das vantagens e desafios de utilizar o mesmo código do SisCAT no SBC da BMO e na estação de trabalho(computador de mesa regular) onde a maioria do desenvolvimento é feito.

Além das vantagens inerentes à própria linguagem, a popularidade de C/C++ resulta em diversas ferramentas para auxílio ao desenvolvimento, como softwares que geram documentação automaticamente interpretando o código fonte ou, inversamente, softwares que geram código a partir de uma especificação UML.

4.5 Testes

Os testes do SisCAT foram separados em Unitários, Integração e Sistema. Como trata-se de um software embarcado, existem diversas vantagens em testar a integração entre o hardware real e o software desde o início do desenvolvimento. Assim, com a mesma infraestrutura de compartilhamento de código entre o SBC(*target*) e a estação de trabalho(*host*), os testes escritos podem ser executados nos dois ambientes. O contato imediato com o hardware a ser utilizado mostrou-se importantíssimo pois limitações não previstas influenciaram em decisões de modelagem e arquitetura do software(Capítulo 3). O fluxo de desenvolvimento adotado como referência inclui escrever testes unitários sempre que novos trechos de código são adicionados e ter um arquivo com sufixo "Test" para cada arquivo de código. Após uma quantidade mínima de testes unitários entre classes relacionadas, testes de integração garantem a interação entre as mesmas. Por fim, o teste de sistema, ou teste caixa-preta, compara o comportamento do sistema com os requisitos especificados.

Para realização dos testes, tanto unitários quanto de integração, o *framework* de testes automatizados CppUTest[12] foi utilizado. Esta ferramenta permite a criação de testes a partir de macros de C++ pré-definidas que incluem toda a lógica necessária para cada teste e cada pacote de teste. Assim, com base no baixo acoplamento entre as classes, como visto no capítulo 3, é possível cobrir diversos cenários de teste de forma isolada e auto-contida. Para medir a cobertura de código atingida pelos testes, a ferramenta *GNU Code Coverage*(GCov)[13] foi utilizada. Com ela, a

execução dos testes gera um arquivo contendo informações de acesso aos métodos. Este arquivo é, então, interpretado pelo Eclipse com o uso de um plug-in específico para este fim(Figura 4.4 e 4.5).

4.5.1 Testes Unitários

Com o paradigma de orientação a objetos vem a vantagem de cada elemento básico do projeto de software, a Classe, conter os dados e as funcionalidades bem encapsuladas. Assim, utilizar uma Classe como elemento básico para organizar os testes unitários é uma escolha natural. No entanto, no SisCAT, existem relações de dependência entre as classes que devem ser levadas em conta na hora do projeto dos testes. Quando esta dependência não puder ser substituída por *Mock Objects*[14](do inglês, *mock-up* significa "esboçar"), faz-se necessário garantir que o módulo cuja classe a ser testada depende esteja com um nível de qualidade aceitável. Para cada Classe, uma lista inicial de casos de teste foi compilada e a cada iteração de desenvolvimento, a lista sofria o acréscimo de casos de teste conforme tornavam-se necessários. Além de servir como organização dos casos de teste, esta lista ajuda a melhorar o entendimento sobre o comportamento esperado e especificado de cada classe.

Tomando como exemplo a classe *SerialInterfaceHandler*, alguns dos casos de teste, em uma versão simplificada, são:

- Tentar executar uma operação com um *Component* inexistente;
- Tentar executar uma operação com um *Component* em estado de erro;
- Tentar executar uma operação com a interface física serial desconectada;
- Receber mensagem de erro de um *Component* durante verificação de status;

Para realizar estes testes, é necessário que exista uma implementação da interface *Component* para simular o comportamento esperado. Neste caso, o desacoplamento entre a classe *SerialInterfaceHandler* e a implementação da interface *Component* permite a utilização de um *Mock Object*, ficando a relação conforme a figura 4.6.

Figura 4.4: Exemplo de lista com métricas de cobertura de código por testes

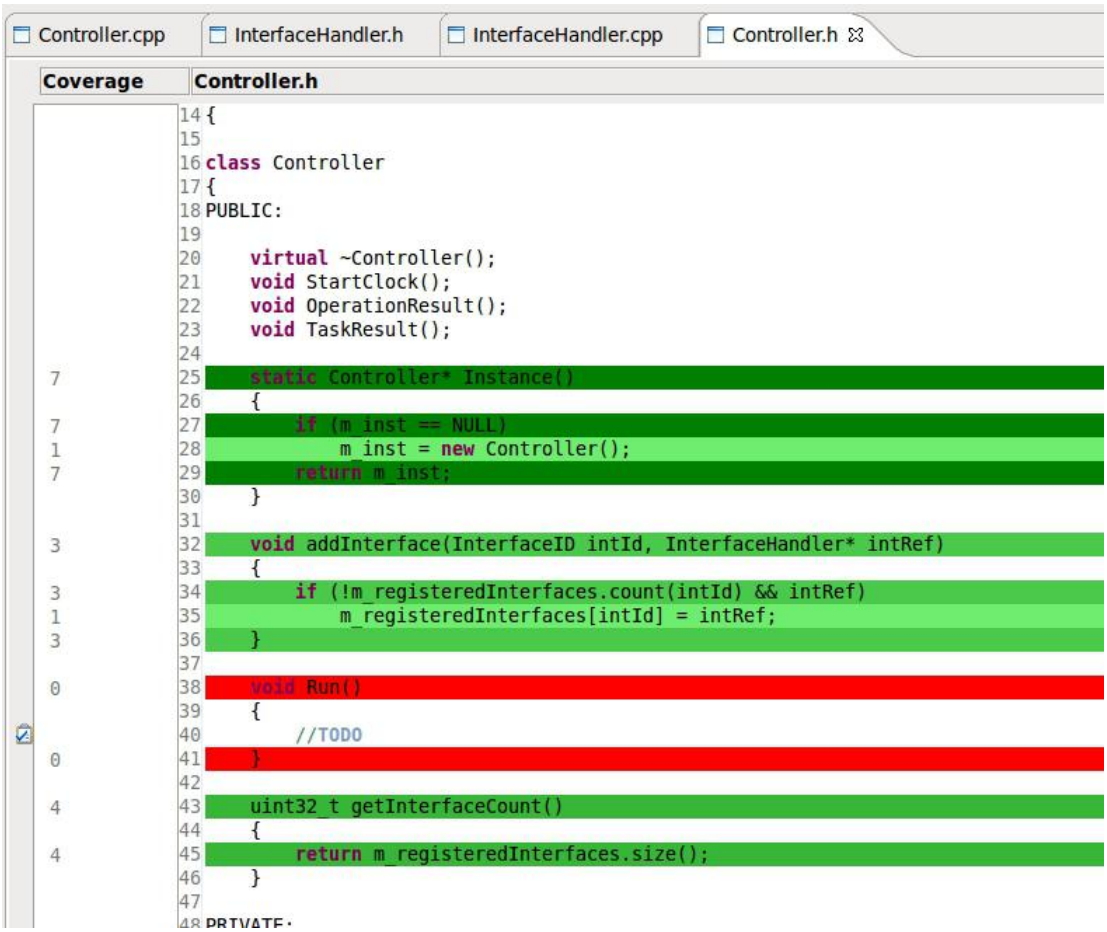

Figura 4.5: Exemplo de identificação visual de cobertura de código. Vermelho: Código não coberto por testes; Verde: Código coberto por testes

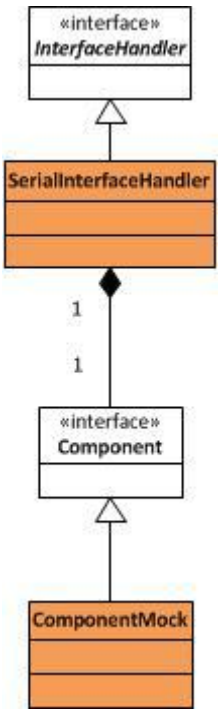

Figura 4.6: Utilização de um *Mock Object* para simular dependências. Classes de cor laranja são concretas

Assim, todo o comportamento necessário por parte da dependência de algum *Component* pode ser simulado pela classe de teste *ComponentMock*. O nome desta técnica é Injeção de Dependência(em inglês *Dependency Injection*) e seu uso permite agilizar o desenvolvimento de componentes independentes.

Por outro lado, quando uma dependência não é isolada através de interfaces, como é o caso das classes *Task*, *Operation* e *Paramaters*(Fig 4.7), o uso de *Mock Objects* é dificultado. Neste caso, a ordem de desenvolvimento leva a classe mais externa, no caso *Parameters*, a ser implementada e testada primeiro. Assim, uma vez que a dependência esteja testada, a classe-pai pode contar com o comportamento real de suas dependências.

4.5.2 Testes de Integração

A partir do momento em que as unidades individuais atingem o comportamento esperado, observado através dos testes unitários, faz-se necessário garantir o funci-

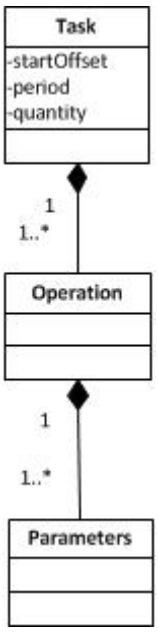

Figura 4.7: Dependência direta entre classes dificulta o uso de *Mock Objects*

onamento de duas ou mais destas unidades em conjunto. O teste de integração é responsável por exercitar a troca de mensagens entre objetos considerados funcionais de forma a detectar anomalias de interface, sincronismo ou problemas sequenciais. No caso do SisCAT, além da integração intrínseca das classes do software, existe também a necessidade de testar a integração entre o software e os periféricos conectados ou pertencentes ao SBC. Assim posto, os cenários de teste de integração têm de cobrir as duas frentes de interação existentes. A figura 4.8 mostra um exemplo de integração entre as classes *Controller*, *SerialInterfaceHandler* e *SBE38*, ocorrido em três iterações distintas. Este exemplo engloba os dois casos citados: A classe *SBE38* interage com o sensor de temperatura Seabird SBE38 e o resto do conjunto interage entre si.

4.5.3 Testes de Sistema

Por fim, os testes de sistema precisam comparar o funcionamento do SisCAT do ponto de vista externo, comparando seu comportamento com o esperado através dos requisitos levantados. Para este fim, o plano de teste de sistema é gerado a partir do documento de especificação de requisitos e deve ser extenso o suficiente

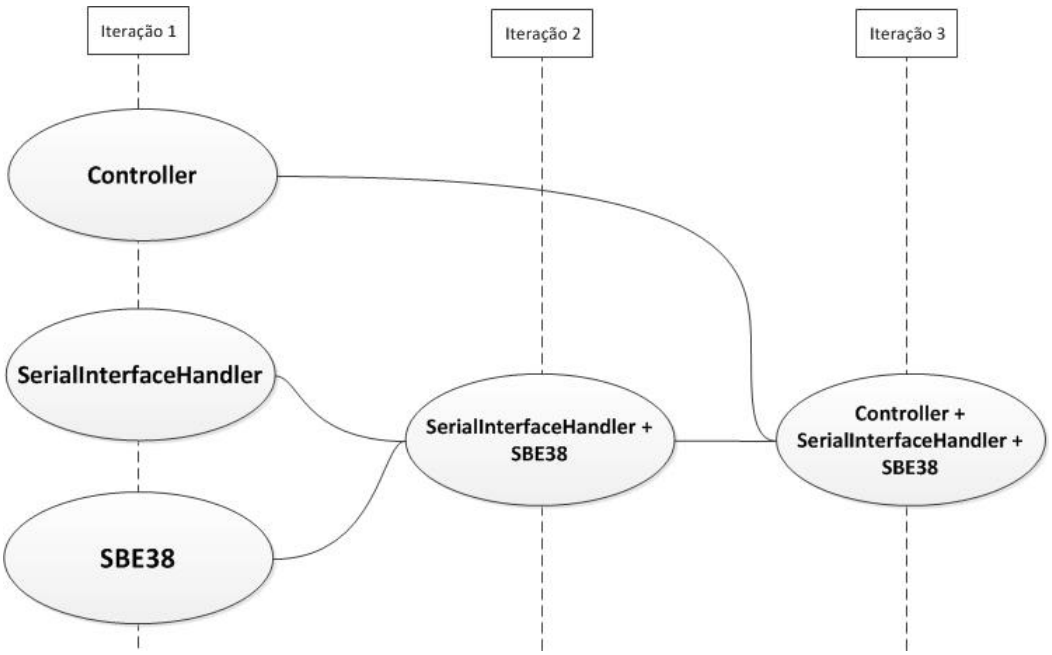

Figura 4.8: Exemplo de integração planejada para as classes *Controller*, *SerialInterfaceHandler* e *SBE38*

para exercitar a maior parte, senão todas, das interações que um usuário tem com o sistema.

4.6 Documentação de Implementação

Para dinamizar a elaboração da Documentação de Implementação, minimizando o trabalho manual, a ferramenta Doxygen[15] foi utilizada. Esta ferramenta lê todos os arquivos de código fonte, interpreta comentários, assinaturas de métodos e relações de orientação a objetos para gerar automaticamente a documentação da base de código implementada. Esta documentação pode ser criada em html, para facil navegação e consulta, ou em PDF, para um documento estático e facilmente versionável. O Doxygen é configurado através de um arquivo de configuração adicionado ao repositório do SisCAT de forma que essa documentação possa ser gerada localmente sempre que necessário. A figura 4.9 exibe um exemplo de documentação gerada pelo Doxygen.

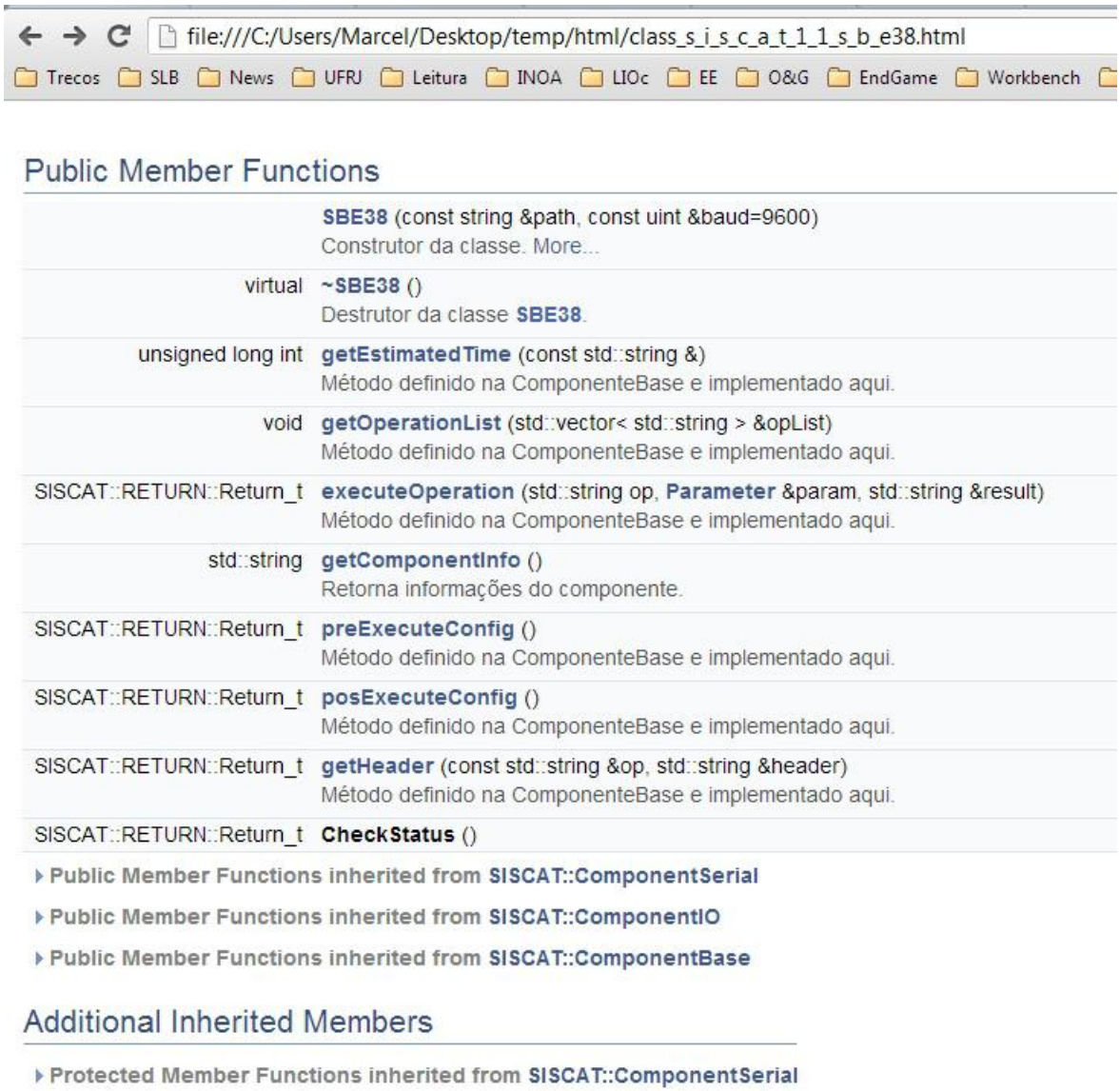

Figura 4.9: Documentação gerada com Doxygen para uma função de exemplo

4.7 Ferramentas de gerenciamento

Durante o desenvolvimento do SisCAT, algumas ferramentas mostraram-se essenciais para o gerenciamento e progresso do projeto. Em primeiro lugar, estabeleceu-se a cultura de utilizar algum sistema de controle de versão de código fonte e de documentos do projeto. Em seguida, por tratar-se de uma equipe com disponibilidade diferenciada, composta por alunos da UFRJ, detectou-se a necessidade[16] de gerenciar a divisão e execução do trabalho atribuído a cada desenvolvedor. Com estas necessidades e com recursos limitados para estabelecer tal infraestrutura de gerenciamento dentro do laboratório com a qualidade necessária, o projeto contou com serviços comerciais hospedados na web.

4.7.1 Repositório e Controle de Versão

Utilizar um sistema de controle de versão para desenvolver software é uma medida responsável e muito bem trabalhada na indústria. Por se tratar de um processo iterativo entre codificação, teste e manutenção, as versões da base de código devem sem muito bem controladas e gerenciadas de forma a permitir que todo o conhecimento embutido no software seja preservado, mesmo em versões consideradas obsoletas. Para este fim, o motor de controle de versão utilizado foi o Mercurial(ou Hg, sigla apropriada da tabela periódica de elementos químicos). Este sistema distribuído de controle de versão(*Distributed Concurrent Versions System*, DCVS) permite cada desenvolvedor colaborar de forma eficiente com o projeto em desenvolvimento, uma vez que réplicas de todo o repositório podem existir localmente(Fig 4.10) em cada estação de trabalho. A hospedagem deste servidor de controle de versão é feita pelo serviço BitBucket[17] em sua infraestrutura de conectividade, armazenamento e segurança.

Além de código fonte, arquivos de configuração e scripts *shell*, o projeto necessita arquivar com segurança documentos com informações confidenciais. Para este tipo de documento, utilizou-se um servidor local com o serviço Alfresco[18]. O Alfresco é um gerenciador de documentos, informações pessoais entre outros conteúdos. Também funciona como um portal(Fig 4.11) exibindo informações de projeto como cronogramas ou marcos importantes. Como a necessidade de acesso remoto pela

equipe de desenvolvimento a esses documentos não é alta, fornecer localmente este serviço mostrou-se suficiente.

4.7.2 Gerenciamento de tarefas

A forma de distribuição de tarefas e gerenciamento do trabalho executado por cada desenvolvedor precisou contemplar a característica distribuída da equipe. Assim, utilizando o serviço Trello[19] foi possível centralizar e administrar o canal de comunicação da equipe sobre tarefas e atividades. O serviço atua como um painel(Fig 4.12) online e interativo, permitindo o monitoramento e uma visão geral do estado das tarefas existentes no projeto. Ao agrupar tarefas em *cards*(cartões, do jargão do Trello) e designar responsáveis para as mesmas, pode-se acompanhar o andamento do projeto pela web. Cada *card* torna-se, então, um grupo de discussão(Fig 4.13) com possibilidades de exibição de documentos, figuras e alertas por e-mail.

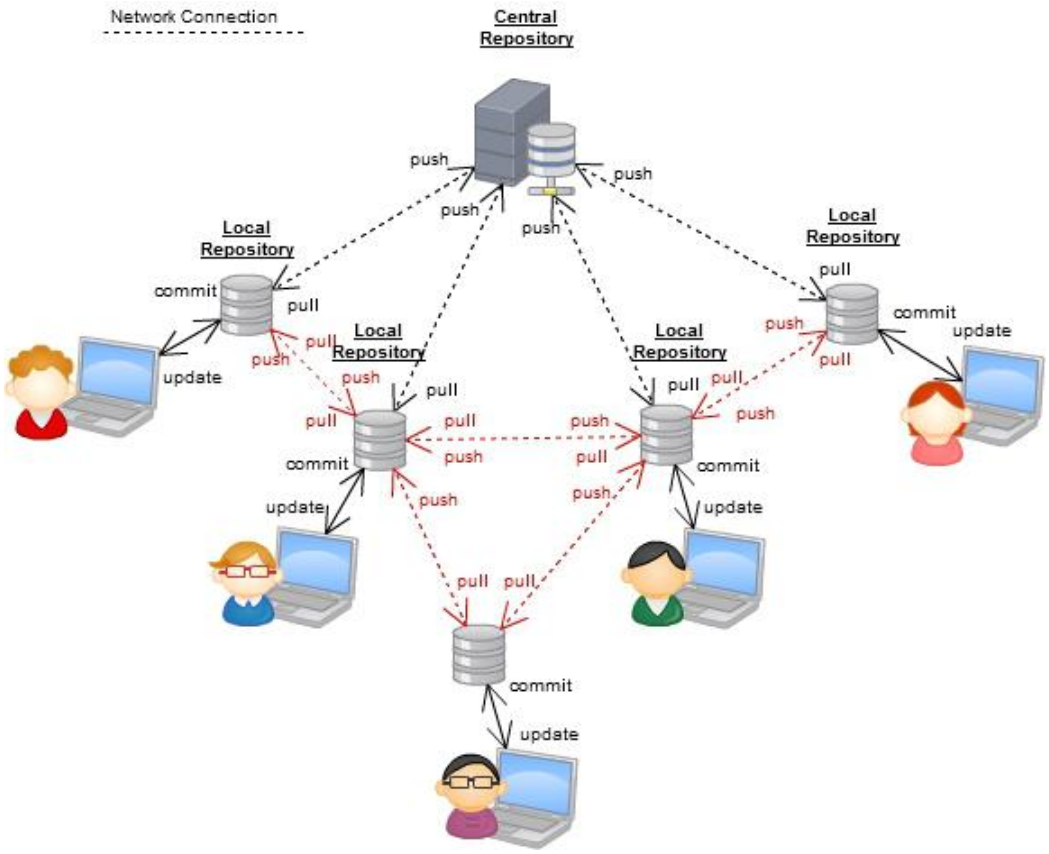

Figura 4.10: Ilustração do funcionamento de um DCVS e suas operações básicas

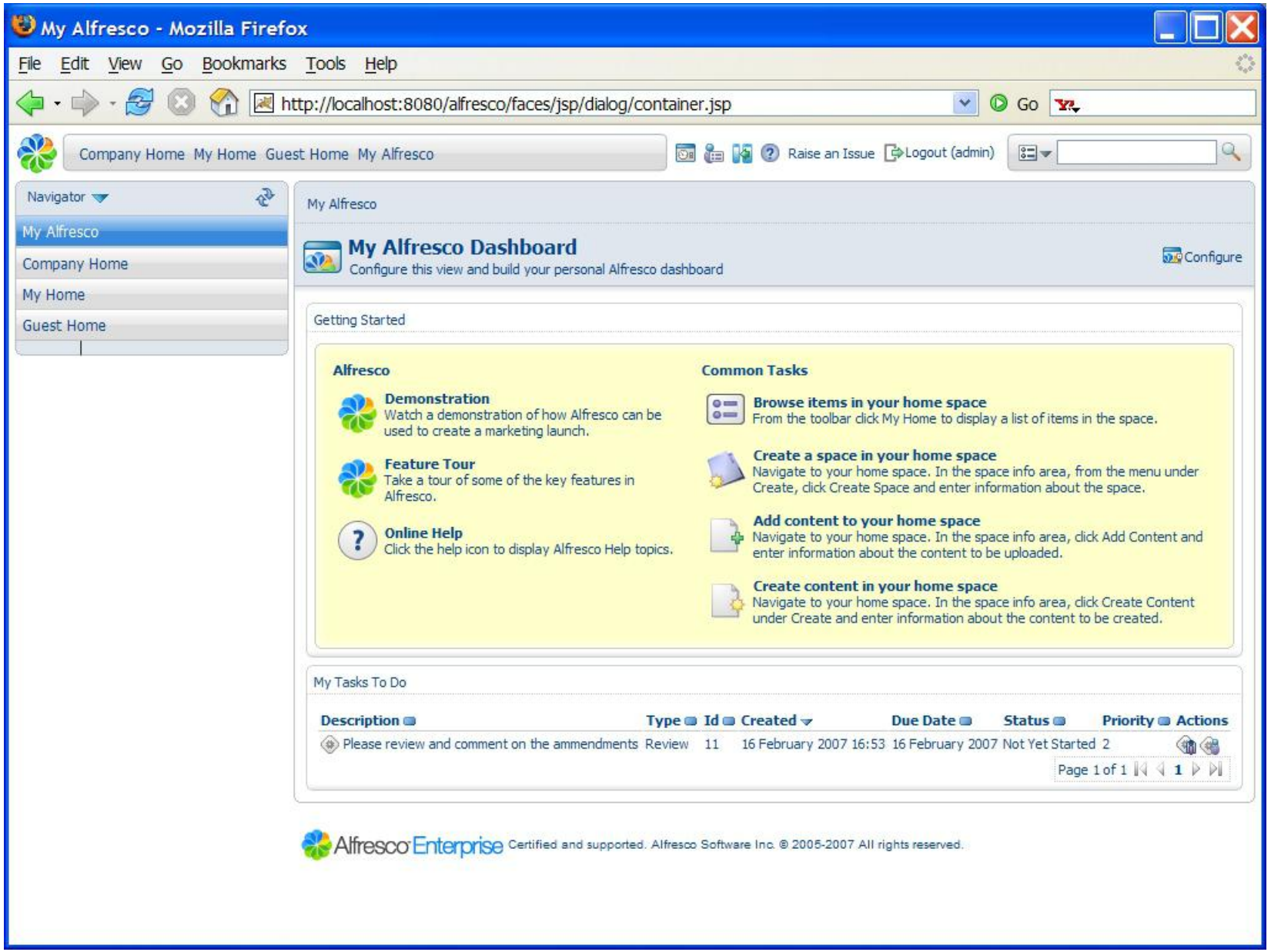

Figura 4.11: Painel de exemplo do Alfresco

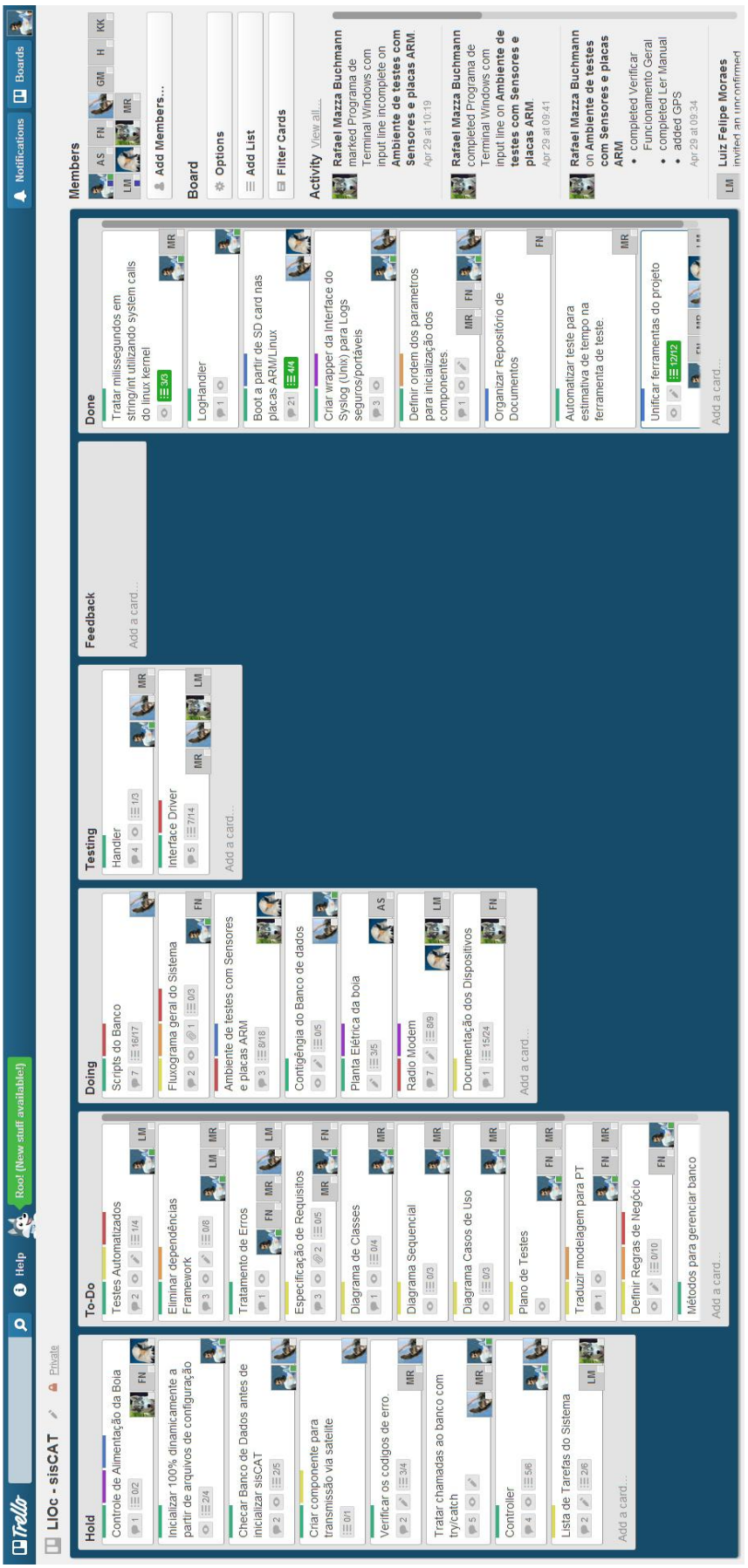

Figura 4.12: Painel de tarefas do Trello

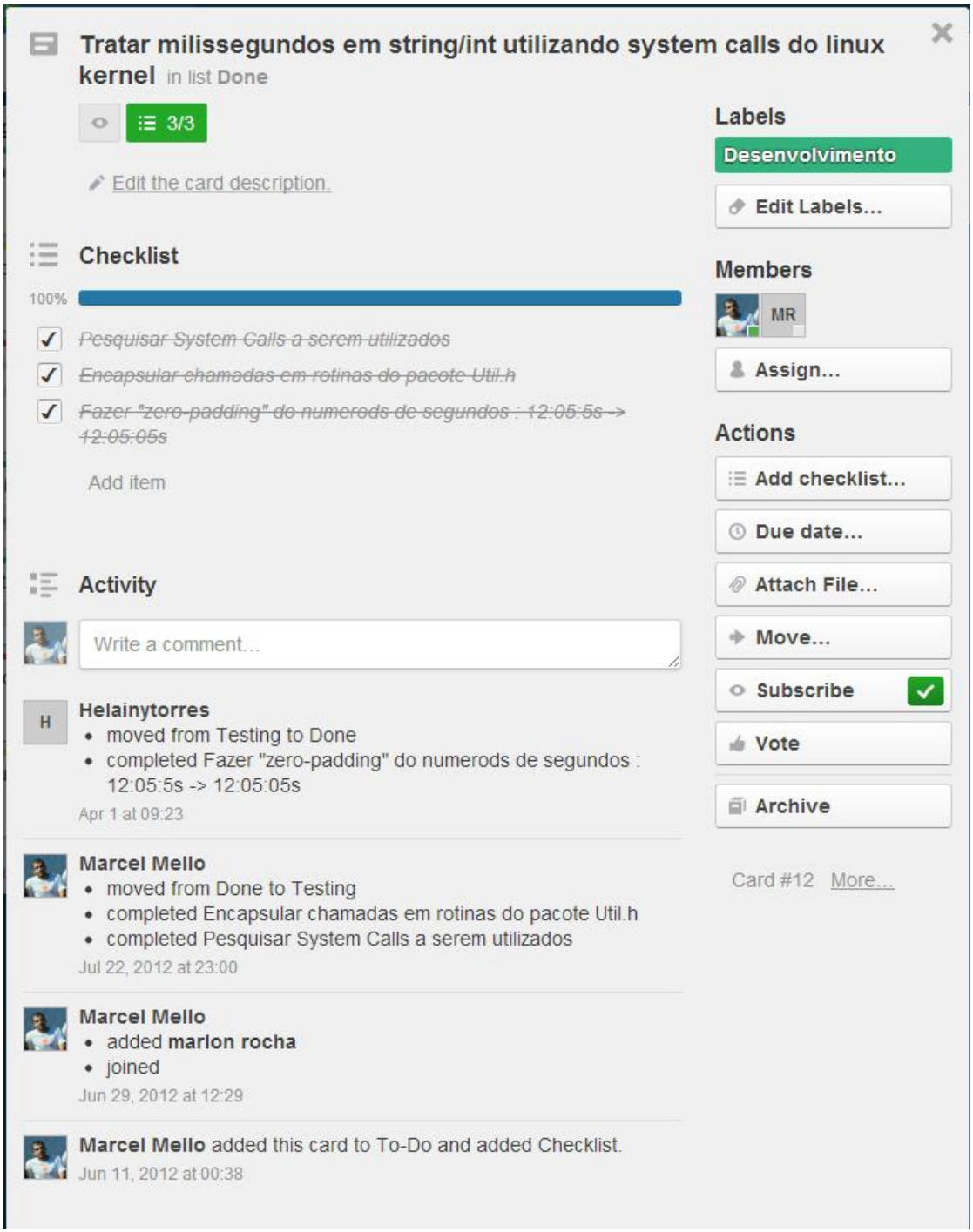

Figura 4.13: Exemplo de *card* expandido no painel

Capítulo 5

Conclusão e Trabalhos Futuros

O SisCAT, software embarcado do projeto REMETEOO, descrito neste trabalho, consiste na iteração atual(datada de abril de 2013) do desenvolvimento. Esta versão é caracterizada como um protótipo funcional servindo de suporte às decisões futuras relacionadas à novas funcionalidades e características. Os conceitos de engenharia de software aplicados na fase de análise e de projeto balizaram o trabalho, tornando a iniciativa o mais profissional possível.

O trabalho de análise de requisitos e a elaboração dos diagramas conceituais foram essenciais para o pleno entendimento do projeto sendo desenvolvido. Os artefatos gerados através destas atividades(Especificação de Requisitos e Diagrama de Classes de Domínio) serviram para capturar a ideia do projeto e para facilitar a comunicação interna da equipe, cujo jargão era exercitado e utilizado constantemente. A arquitetura projetada permitiu experienciar um desenvolvimento paralelizável, pois os *Component's* relacionados aos instrumentos puderam ser desenvolvidos e testados de forma isolada das funcionalidades do núcleo do SisCAT. Esta separação resultou em um desenvolvimento com melhor qualidade e sem contra-tempos comuns de alto acoplamento entre classes. Assim, os requisitos de reusabilidade e extensibilidade foram atendidos.

Outro aspecto interessante do trabalho foi o contato com o sistema operacional Linux embarcado em um computador com baixos recursos e baixíssimo consumo. O *Single Board Computer* com Linux mostrou-se uma plataforma adequada para o projeto, pois as facilidades disponíveis pelo sistema operacional permitiram acelerar

o desenvolvimento e até melhorar as funcionalidades implementadas. O uso de uma linguagem de programação popular e difundida como C++ permitiu acesso à coletânea de conhecimento disponível na literatura e nas comunidades *open-source*. A sinergia da linguagem com o paradigma de orientação a objetos foi essencial para a conversão do Diagrama de Classes de Domínio em código.

Também é válido destacar que a utilização das ferramentas descritas no capítulo 4 para gerenciar o andamento do projeto foi de extrema importância para o aprendizado dos envolvidos. Diversas dificuldades gerenciais foram vencidas com o uso de tais ferramentas e, pela relevância na indústria, a experiência adquirida agregou, de fato, um valor inquestionável à formação dos desenvolvedores. Também evidenciou-se a melhora na produtividade quando frente à um ambiente com processos que, antes manuais e monótonos, puderam ser automatizados. Um dos principais desdobramentos desta abordagem é a diminuição no tempo de treinamento e adequação dos novos membros desenvolvedores.

Para a próxima fase do SisCAT, espera-se refinar a especificação do protocolo utilizado no Arquivo de Configuração e Arquivo de Usuário para, então, considerar os requisitos de programação remota da BMO como plenamente satisfeitos. Também espera-se a conclusão do desenvolvimento de *Component's* relacionados à instrumentos que, por logística ou outros motivos, sofreram atraso para estarem disponíveis à equipe. Por fim, com a realização do subprojeto controlador de alimentação, modelado como *PowerController* no SisCAT, espera-se atingir um fino controle sobre o consumo de energia total da BMO, permitindo ao SisCAT atuar sobre a alimentação individual de instrumentos e periféricos. A presença desta classe no projeto o prepara para tal momento, evitando impactos sobre a arquitetura ou obsolescência precoce.

Referências Bibliográficas

[1] SILVA, L. G. D., "Metodologia de gerenciamento de projetos de inovação em âmbito acadêmico", *Universidade Federal do Rio de Janeiro*, , Fevereiro 2012.

[2] OLDFIELD, P., *Domain Modelling*, Report, Appropriate Process Group, 2002.

[3] PFLEEGER, S. L., *Engenharia de Software: Teoria e Prática*. São Paulo, Prentice Hall, 2004.

[4] LARMAN, C., *Applying Uml and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*. Upper Saddle River, Prentice Hall, 2004.

[5] YUE, K., JIANG, L., YANG, L., *et al.*, "Research of Embedded Database SQLite Application in Intelligent Remote Monitoring System", *2010 International Forum on Information Technology and Applications*, , Shenyang, China, 2010.

[6] HIPP, R., "SQLite3", http://www.sqlite.org/, 2000, (Acesso em 02 Junho 2013).

[7] BUTENHOF, D. R., *Programming with POSIX Threads*. Addison-Wesley, 1997.

[8] KERRISK, M., *The Linux Programming Interface: A Linux and UNIX System Programming Handbook*. No Starch Press, 2010.

[9] ATMEL, *AT91SAM9G20 Reference Manual*, Report, Atmel Corporation, 2008.

[10] BI, C.-Y., LIU, Y.-P., "Research of Key Technologies for Embedded Linux Based on ARM", *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, , Ningbo, China, 2010.

[11] MENTOR GRAPHICS, I., "Sourcery CodeBench Lite", http://www.mentor.com/, 2012, (Acesso em 02 Junho 2013).

[12] VODDE, B., "CppUTest", http://www.cpputest.org/, 2012, (Acesso em 02 Junho 2013).

[13] FREE SOFTWARE FOUNDATION, I., "GNU Code Coverage Tool", http://gcc.gnu.org/onlinedocs/gcc/Gcov.html, 2012, (Acesso em 02 Junho 2013).

[14] BROWN, M. A., TAPOLCSANYI, E., "Mock Object Patterns", http://hillside.net/plop/plop2003/Papers/Brown-mock-objects.pdf, 2003, (Acesso em 02 Junho 2013).

[15] HEESCH, D. V., "Doxygen", http://www.stack.nl/dimitri/doxygen/index.html, 2012, (Acesso em 02 Junho 2013).

[16] NASCIMENTO, F., "A Internet como Ferramenta no Auxílio ao Gerenciamento e Preparação de Experimentos de Campo em Oceanografia", *Anais do XXIV Congresso Latino Americano de Hidráulica*, , Punta Del Este, Uruguai, 2010.

[17] "Atlassian Inc : BitBucket", https://bitbucket.org/, 2009, (Acesso em 02 Junho 2013).

[18] ALFRESCO SOFTWARE, I., "Alfresco Document Management", http://www.alfresco.com/, 2013, (Acesso em 02 Junho 2013).

[19] "Fog Creek Software : Trello", https://trello.com/, 2010, (Acesso em 02 Junho 2013).

Apêndice A

Exemplo de Teste Automatizado

```
/*
 * ControllerTest.cpp
 *
 *  Created on: Feb 2, 2013
 *      Author: marcel
 */

#include "CppUTest/TestHarness.h"
#include "CppUTest/TestRegistry.h"
#include "CppUTest/TestOutput.h"
#include "CppUTest/TestTestingFixture.h"
#include "Mocks/InterfaceMock.h"

#include "Controller.h"

#include "SisCAT.h"

void SISCAT::SisCAT::initialize()
{


}

TEST_GROUP(ControllerTest)
{
   void setup()
   {
      SISCAT::Controller::Instance()->m_registeredInterfaces.clear();
```

```
    }

    void teardown()
    {
    }
};

TEST(ControllerTest, InstanceNotNull)
{
    SISCAT::Controller* cont = SISCAT::Controller::Instance();
    CHECK_TRUE(cont);
}

//Tenta adicionar uma interface com referenecia nula
TEST(ControllerTest, AddNullInterface)
{
    SISCAT::Controller::Instance()->addInterface("Test", NULL);
    CHECK_TRUE(SISCAT::Controller::Instance()->getInterfaceCount()==0);
}

//Tenta adicionar duas interfaces com a mesma identificação, só a
    primeira sucede
TEST(ControllerTest, AddInterfaceSameID)
{
    TEST::InterfaceMock int1("ID");
    TEST::InterfaceMock int2("ID");
    SISCAT::Controller* ctrlInst = SISCAT::Controller::Instance();

    int startIntCount = ctrlInst->getInterfaceCount();

    ctrlInst->addInterface(int1.getId(), &int1);
    CHECK_TRUE(ctrlInst->getInterfaceCount() - startIntCount == 1);

    ctrlInst->addInterface(int2.getId(), &int2);
    CHECK_TRUE(ctrlInst->getInterfaceCount() - startIntCount == 1);
}
```