UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
CURSO DE BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

INFRAESTRUTURA DE HARDWARE IN THE LOOP PARA TESTES FUNCIONAIS DE SISTEMAS DE MONITORAMENTO REMOTO

TRABALHO DE GRADUAÇÃO

Diogo Pereira de Morais

RECIFE
2017
UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

CURSO DE BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

Diogo Pereira de Morais

Infraestrutura de Hardware in the Loop para testes funcionais de sistemas de monitoramento remoto

Monografia apresentada ao Centro de Informática (CIN) da Universidade Federal de Pernambuco (UFPE), como requisito parcial para conclusão do Curso de Engenharia da Computação, orientada pelo professor Ricardo Martins.

RECIFE

2017

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

CURSO DE BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

Diogo Pereira de Morais

Infraestrutura de Hardware in the Loop para testes funcionais de sistemas de monitoramento remoto

Banca Examinadora:

_____ Orientador

Ricardo Martins

Doutor

_____ Examinador

Silvio Melo

Doutor

Dedico este trabalho à família, amigos e especialmente ao meu pai. Sem tal suporte não teria chegado até aqui.

AGRADECIMENTOS

Agradeço a universidade federal de Pernambuco e os professores do CIn-UFPE pelos conhecimentos compartilhados e todo aprendizado de vida adquirido enquanto estudando de graduação.

Agradeço ao meu pai, à minha namorada e aos amigos pelo apoio durante os momentos que abdiquei da presença deles para me dedicar ao curso. Agradeço em especial os meus pais, pelo acompanhamento e o auxílio em todos os momentos.

Aos meus tutores e professores que me incentivaram ao aprendizado técnico e pessoal, por ensinamentos muito além universo da engenharia e das ciências.

À Tomus, por contribuir no desenvolvimento deste trabalho, e em especial aos engenheiros George Fonseca, Kleber Reis, Leonardo Bueno, Felipe Campos e Alex Fernandes, pelos conhecimentos transmitidos e pela oportunidade de fazer parte de um ambiente desafiador e receptivo.

Enfim, a todas as pessoas que direta ou indiretamente participaram dessa etapa da minha vida acadêmica que agora concluo.

RESUMO

Este relatório é o resultado de um Trabalho de Graduação realizado no Centro de Informática da Universidade Federal de Pernambuco. Teve como objetivo a elaboração, implementação e teste de uma infraestrutura de Hardware in the Loop para testes funcionais de sistemas de monitoramento e controle remoto de equipamento industriais projetadas pela empresa Tomus Soluções. Foi construído um sistema de emulação de hidrômetro e solenoide, assim como sistemas de emulação de servidor aplicação e integração com interface homem-máquina. O sistema inclui um Arduino Uno com interface de comandos para emulação de sinais de entradas e saídas para o sistema testado. Para garantir o reuso da infraestrutura de testes, o software foi construído utilizando uma arquitetura modular.

Verificou-se a eficácia do sistema em testar os diversos cenários de utilização do equipamento de controle de reservatórios reduzindo-se o tempo de desenvolvimento e os bugs reportados após aplicação em campo.

ABSTRACT

This report is the result of a Senior Design Project developed at the Centro de Informática at Universidade Federal de Pernambuco. The main goal was the design, implementation and testing of a Hardware in the Loop infrastructure for functional testing of industrial remote monitoring and control eletronic systems developed by Tomus Soluções. A water valve and hydrometer simulation systems was design, as well as simulation systems for application servers and human-machine interfaces. The systems includes an Arduino Uno as input and output interface. To ensure reuse, the software was built using modular architecture.

The testings validated the effectiness of the infrastructure on testing diverse application scenarios for a water control device, reducing developing time and bugs reported after field evaluation.

Keywords: Hardware in The Loop, Test Driven Design, Simulation

Sumário

Lista de Figuras

LISTA DE TABELAS

TABELA DE SIGLAS

Sigla	Significado	Página
HIL	Hardware-in-the-Loop	
SUT	System Under Test	
IHM	Interface Human-Machine	

1. Introdução

Sistema embarcado é um sistema baseado em microcontrolador ou microprocessador que serve para controlar uma função ou um conjunto de funções e não pode ser reprogramado por um usuário final. Sendo assim, são sistemas que foram projetados para servir apenas a um propósito específico e não se pode adicionar ou alterar suas funcionalidades de maneira prática.

Sistemas embarcados geralmente tem uma missão mais limitada dentro do contexto de uma aplicação. Por isso, é muito comum haver restrições de hardware e necessidade de otimizações para reduzir consumo de energia, custo, dimensões físicas e melhorar a performance.

O hardware não é a única parte do sistema a ter restrições. Em alguns sistemas, o software precisa agir deterministicamente e em tempo real, isto é, reagir com o mesmo comportamento para uma mesma entrada em todas as vezes, e cujas tarefas possuem prazo para ser realizadas pra não ocasionar uma falha. Além disso, alguns sistemas ainda precisam garantir que o software seja tolerante a falhas e consiga se recuperar na presença de erros.

Outras aplicações mais críticas demandam que o software embarcado seja confiável e robusto, isto é, não deve apresentar erros, e caso ocorram, o software não pode gerar instabilidade para o restante do sistema. Um exemplo disso seria um sistema de freios ABS, para aplicações automobilísticas, onde falhas no software embarcado podem ocasionar resultados fatais. Portanto, falhas podem ocasionar degradação de toda a aplicação, que neste caso seria comprometer a frenagem do veículo.

Sendo assim, deve-se garantir que o sistema embarcado seja o mais correto e completo quanto se puder ao final do desenvolvimento, pois geralmente não são atualizáveis de maneira prática e segura. Embora garantir um software de qualidade não seja uma tarefa fácil, existem estratégias para reduzir a presença de *bugs* e faltas através de testes unitários, de integração e principalmente testes de funcionalidades, que validam se o sistema entrega as funções a que se propõe. Além disso, o custo de corrigir problemas e fazer *bug fixes* nas fases finais do projeto, onde o equipamento foi ou está sendo preparado para entrega aos usuários é maior do que em tempo de desenvolvimento. Um exemplo importante disso são os altos custos de Recall por falhas em sistemas críticos de automóveis.

Há diversas maneiras de garantir a qualidade de software embarcado, incluindo aplicação de padrões de projeto, levantamento e validação de requisitos com os principais

interessados, testes unitários, testes de integração e testes de validação de sistema. Testes unitários garantem que partes individuais do sistema se comportam devidamente, isto é, são realizados separando módulos e testando-os individualmente. Testes de integração validam a integração entre hardware e software e sua capacidade de juntos terem potencial para garantir o funcionamento da aplicação. Testes de validação garantem que o sistema entrega uma funcionalidade especificada, isto é, dado um estímulo de entrada verifica-se se o sistema como um todo responde da forma esperada.

Testes de validação são realizados para validar as funcionalidades do sistema, porém, as vezes não é possível realizar testes de validação por questões de segurança, disponibilidade de protótipo e custo. Por exemplo, considere um sistema de controle remoto controlado por um servidor web. Após implementar e testar todos os atuadores envolvidos, deve-se testar se o equipamento responde corretamente aos comandos de um servidor. Porém, o servidor pode não estar implementado, ou indisponível para realizar os testes de validação.

Um dos métodos mais eficientes para validação de sistemas embarcados é a simulação Hardware in the Loop. O método oferece poderosos recursos para tonar os testes eficientes e completos, visto que alguns testes não são possíveis de serem realizados devido ao custo agregado, disponibilidade de tempo e segurança.

Hardware in the Loop consiste em simular subsistemas que não podem ser incluídos nos testes fisicamente, permitindo que seja possível testar todo dispositivo de controle embarcado de forma completa antes de colocar o sistema em ambiente real. Assim, pode-se garantir a confiabilidade do sistema quantos aos requisitos ao mesmo tempo que reduz custos de testes e o tempo de lançamento do produto em mercado.

Figure 1 Interface de usuário do console de simulação da Embraer

1.1. Motivação

Sistemas embarcados integram hardware e software para entregar as funcionalidades, portanto estão suscetíveis às falhas de ambos. Contudo, devem ser confiáveis porque integram aplicações maiores, sendo responsáveis pela aquisição de dados, monitoramento de condições importantes ao sistema e controle de atuadores. Construir um sistema dedicado confiável é um desafio, levando-se em conta que este restringe a possibilidade de ser reprogramado pelo usuário final, e que em algumas situações seria extremamente complicado e custoso fazer correções, a exemplo de satélites e aviões que possuem subsistemas de monitoramento e atuação que são importantes para sua operação.

Sistemas dedicados estão presentes em várias aplicações na nossa sociedade, visando melhorar a vida das pessoas ao oferecer soluções para diversos campos do conhecimento humano. As suas aplicações impulsionam o desenvolvimento tecnológico em áreas como saúde, sistemas automotivos, sistemas aeroespaciais e em aplicações ordinárias, como TVs e semáforos. Logo, percebe-se a importância de garantir a corretude desses subsistemas para o bom funcionamento de quase tudo a nossa volta.

Portanto, é preciso verificar se todas as funcionalidades especificadas são entregues corretamente. Além disso, identificar possíveis cenários hostis para tornar o sistema robusto e tolerante, isto é, sistema tolerante suporta eventuais falhas sem perder a capacidade de recuperar seu funcionamento. Sistema Robusto tem a capacidade de prever possíveis erros e antecipa seu tratamento, evitando falhas.

A maioria das aplicações possuem subsistemas para monitorar condições que alimentam o controle na tomada de decisões, podendo haver também atuadores para modificar o ambiente de alguma forma. No caso de um sistema em tempo real, a possibilidade de haver problemas se torna mais crítica.

A realização de testes para evitar erros e faltas do sistema pode trazer maiores custos ao projeto e um prejuízo ao *Time to Market* e expor um sistema em desenvolvimento a testes em condições reais podem danificar protótipos, gerando custos desnecessários. Além disso, as condições reais de operação em que o sistema será imerso podem não ser facilmente reproduzidas, como um voo teste.

Apesar dessas restrições, a metodologia de Hardware in the Loop tem recursos para validar o sistema de forma eficaz. Com essa metodologia é possível simular as condições de ambiente e operação virtualmente para testar as funcionalidades da aplicação sem prejuízo ao tempo de lançamento e nem gerar custos adicionais. Outra característica importante dessa

metodologia é conseguir simular um ambiente controlado para testes, onde situações podem ser replicadas e repetidas quantas vezes for necessário.

Embora haja muitos benefícios, há um trabalho adicional em construir uma infraestrutura de testes para implementar essa metodologia. Então, torna-se um projeto à parte, mas que pode ser reutilizado e representa um ganho muito significativo para várias empresas que aplicam Hardware in the Loop.

A Embraer realiza testes utilizando simulação Hardware in the Loop para validar o sistema elétrico de controle dos seus aviões. São realizadas simulações antes de colocar o avião em voo teste e apesar do custo da infraestrutura e tempo de desenvolvimento, ainda assim conseguiram reduzir o tempo de desenvolvimento.

1.2. Objetivos

O objetivo deste trabalho é implementar uma infraestrutura de simulação Hardware in the Loop para validação de sistemas de telemetria e controle remoto conceitualmente semelhante aquela construída para a Tomus soluções em eletrônica.

Será demonstrada uma arquitetura modular de uma infraestrutura para simulação de sensores, atuadores e servidor web para controle remoto. Devido a arquitetura ser construída de forma modular permite-se a adição, alteração e retirada de módulos que simulam sensores e atuadores para criar um ambiente controlado de testes em laboratório de desenvolvimento.

Além da arquitetura modular, serão demonstrados os módulos para simulação de servidor web para testar controle remoto e sensores para emular variáveis monitoradas pelo sistema de telemetria sob teste.

Na conclusão, são apresentados os resultados positivos para a Tomus obtidos pela redução do tempo de desenvolvimento, impactando no *Time to Market,* e diminuição significativa de *bugs* em campo que impactaram nos custos de manutenção do sistema.

1.3. Metodologia

O Trabalho foi realizado nas seguintes etapas, as quais foram programadas de acordo com o cronograma apresentado na proposta do projeto:

- Etapa 1: Revisão bibliográfica sobre os conceitos básicos de testes unitários e funcionais, desenvolvimento orientado a testes, ciclo de desenvolvimento de embarcados, simulação Hardware in the Loop e sistemas de telemetria e controle remoto. Também foram analisadas metodologias ágeis de desenvolvimento, métricas de eficiência para desenvolvimento de software embarcado, engenharia de software, arquitetura modular e reutilização de código;
- Etapa 2: Modelagem da arquitetura de software modular para simular ambientes de teste e levantamento dos testes de validação para construção dos cenários.
- Etapa 3: Implementação dos módulos para simular as variáveis de entrada do sistema, incluindo servidor web, sensores e simulação subsistemas da aplicação principal;
- Etapa 4: Execução dos testes funcionais com a infraestrutura no sistema sob teste e análise de resultados e impactos da metodologia;
- Etapa 5: Redação do relatório;
- Etapa 6: Revisão do relatório;

1.4. Estrutura do relatório

Este relatório foi estruturado da seguinte forma:

- Seção 2: será apresentada a fundamentação teórica deste trabalho, de forma a proporcionar ao leitor uma referência sobre os principais assuntos abordados nesse relatório de TG;
- Seção 3: serão apresentados a arquitetura do sistema testador desenvolvido e os artefatos produzidos ao longo do trabalho, os quais incluem: o projeto arquitetural; os módulos de emulação de contexto;
- Seção 4: serão discutidos os critérios a serem adotados na validação dos resultados e os resultados alcançados com os desenvolvimentos realizados nesse TG;
- Seção 5: são apresentadas as considerações finais e sugestões de trabalhos futuros.

2. Fundamentação teórica

Neste capítulo, são introduzidos alguns termos e conceitos utilizados ao longo deste trabalho, incluindo as métricas utilizadas para analisar o impacto do uso de simulação HIL.

Apresentados conceitos e dados que influenciam no direcionamento para este tipo de testes para sistemas embutidos, como modelos de desenvolvimento baseados em testes e arquiteturas modulares para implementação de uma infraestrutura genérica.

2.1. Ciclo de desenvolvimento de software embarcado

O processo de desenvolvimento adotado no projeto é baseado no *Classic V-model*, que é um processo de desenvolvimento derivado do *Waterfall* que relaciona cada etapa de desenvolvimento do software a uma etapa correspondente de validação, adicionando uma abordagem de desenvolvimento orientado a testes. [Scippacercola, Pietrantuomo, Russo e Zentai, 2011]

O processo se inicia com o levantamento de requisitos funcionais e não funcionais do sistema, isto é, definindo as funcionalidades que o software deve entregar e quais as restrições que devem ser obedecidas. Nesta etapa, são levantadas as funções a serem providas pelo sistema, visando que ao final do processo de desenvolvimento todos os requisitos tenham sido implementados da forma como deveriam e possam ser verificados. Portanto, os requisitos são importantes para analisar a completude do sistema, delimitar seu escopo e para servir de base para os testes de validação do sistema completo.

A próxima etapa é *System Design*, que define uma arquitetura alto nível do sistema, planejando como as partes do sistema vão interagir, os protocolos e a modelagem de como serão entregues as funcionalidades. Até aqui, o sistema é visto sem separação entre hardware e software, embora sejam definidos requisitos que o hardware e o software devem atender e obedecer às restrições do projeto. Os testes correspondentes a essa fase são testes de integração, que validam se a interação entre hardware e software estão corretas, harmoniosas e, portanto, validam se o conjunto hardware e software possui potencial para atingir as funcionalidades.

Na etapa de *Component Design*, as responsabilidades de hardware e software são separadas. Neste momento é feita a arquitetura do software embarcado e projetadas as placas

de acordo com as especificações de hardware. Os testes unitários são empregados para validar o funcionamento dos módulos individualmente.

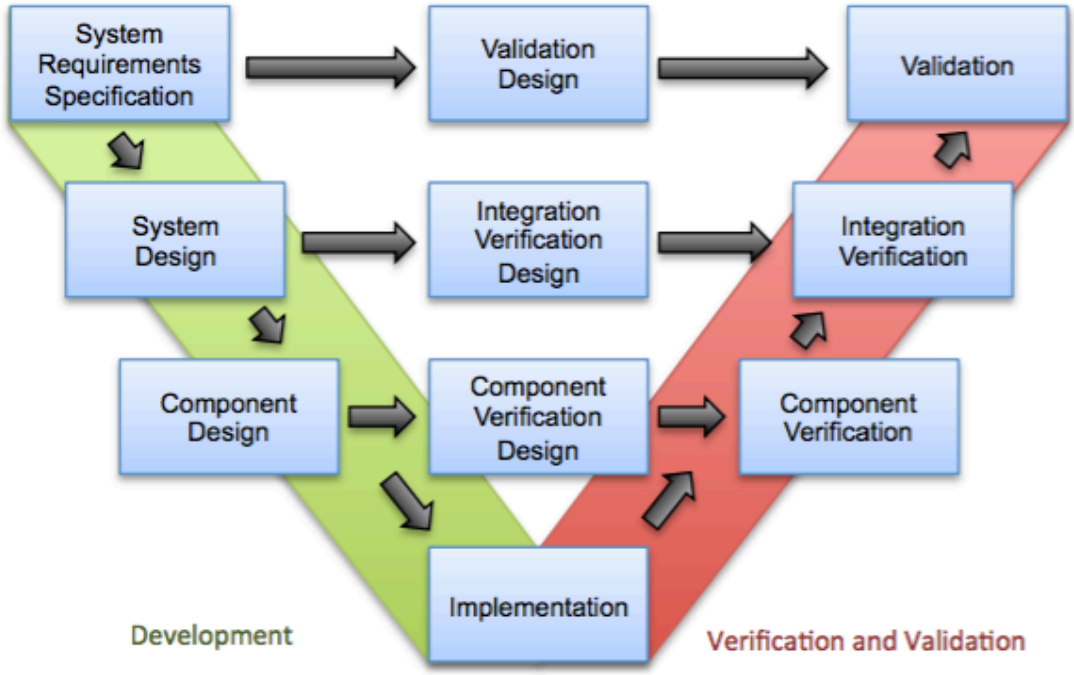

Figure 2 Diagrama V-model

2.2. Conceitos e componentes para simulação Hardware in the Loop

i. Conceitos gerais

Hardware-In-the-Loop (HIL) é uma técnica de simulação em tempo real onde "os sinais de entrada e de saída do simulador mostram os mesmos valores de um processo real" [Sanvido, 2002]

A simulação HIL é um método de simulação em tempo real para teste que oferece poderosos recursos e maior eficiência para testes de sistemas embarcados. Quando se testa um sistema embarcado, nem sempre é possível fazer todos os testes necessários com o sistema completo, principalmente quando se trata de aplicações críticas.

Em aplicações críticas, como aplicações aeroespaciais, automotivas e industriais, geralmente não é possível testar todos os subsistemas fisicamente. Alguns fatores podem

limitar a possibilidade de testes em ambiente real, tais como disponibilidade de cenário, custos de testes e riscos de segurança.

A simulação HIL tem a vantagem de criar um ambiente virtual para testar as funcionalidades do sistema embutido e validar a confiabilidade da aplicação completa. Existem ainda outras vantagens inerentes à simulação interessantes tanto para sistemas críticos quando para sistemas apenas de monitoramento. A capacidade de replicar o teste facilmente é uma grande vantagem de simulações, porque permite que sejam realizadas melhorias incrementais e pode-se testar várias soluções para um mesmo cenário. Outra grande vantagem é a capacidade criar situações de ambiente hostis ao sistema afim de testar sua robustez e capacidade de tolerar tais condições. Para aplicações mais críticas, os maiores ganhos são em relação a redução de custos nos testes, desenvolvendo menos protótipos e eliminação de riscos de testes em ambiente real. Essa última vantagem é especialmente importante para aplicações críticas, como no caso de aviões em que é validado todo sistema elétrico através de HIL e apenas depois dessa etapa que os testes de voo são conduzidos.

A Embraer fabrica aviões incluindo simulações HIL nas suas metodologias de testes. Os aviões passam por testes de simulação antes de iniciarem testes de voos, e validam todo o sistema eletrônico e de controle em ambiente virtual. Redução de time to Market!*

[dados e imagem]

ii. Componentes de sistemas de testes Hardware-in-the-loop

O sistema de testes HIL é composto por elementos de software e hardware dotados de modelos matemáticos que possam descrever fielmente as características do ambiente de testes. É composto por 3 entidades principais: uma unidade de processamento em tempo real, interfaces E/S e uma interface de operação.

O processador de tempo real é o centro do sistema de teste HIL, que é responsável pelos estímulos que caracterizam o ambiente virtual, reproduzindo os modelos matemáticos da simulação e excitando o sistema sob teste (System Under Test - SUT) através dos periféricos. Através desses estímulos, pode-se analisar o comportamento do SUT para quaisquer cenários de testes planejados. Portanto, a unidade de processamento pode estimular diretamente o SUT através da interface E/S ou controlar outros periféricos que o façam.

As interfaces E/S são a ponte entre o sistema de simulação e o SUT, por onde são transmitidos os estímulos na forma de sinais analógicos e digitais. As entradas e saídas são

usadas para enviar estímulos diretamente ao SUT e também enviar comandos a sensores e atuadores para simular os sentidos do software embutido.

A interface com o operador permite entrar com comandos para a unidade de processamento em tempo real para controle dos estímulos e visualização do comportamento do sistema durante os testes.

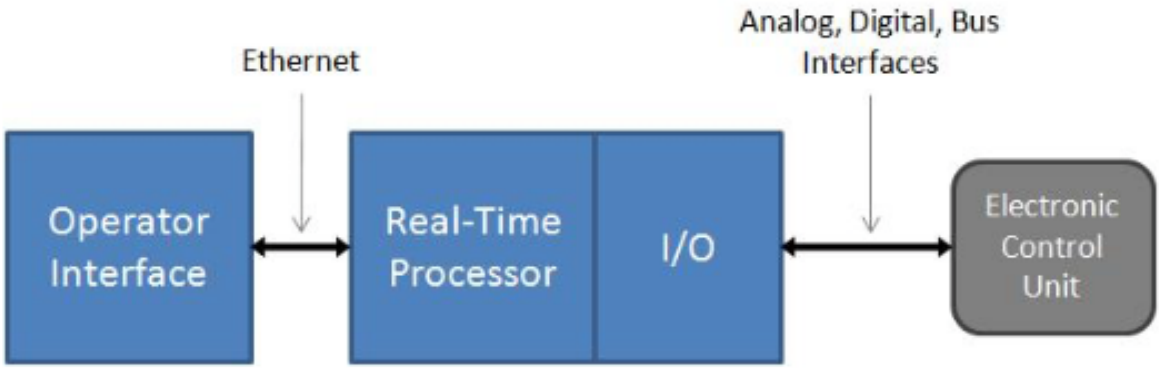

Figure 3 Arquitetura geral Hardware-in-the-Loop

3. Desenvolvimento da infraestrutura HIL

Nesta seção será abordado o processo de desenvolvimento da infraestrutura de HIL para validação de sistemas de telemetria e controle remoto. O projeto da arquitetura da infraestrutura e sua estrutura modular foi pensado para esse tipo de sistemas, considerando um conjunto de sensores, atuadores e um servidor web.

A seção trata de explicar conceitos gerais de sistemas telemétricos e sua arquitetura geral, modelagem das variáveis simuladas para estimular as entradas do SUT (System Under Test) de acordo com os cenários de teste e a implementação da infraestrutura específica para validar o equipamento da Tomus, isto é, para o SUT específico. Após construída a infraestrutura HIL e tendo-se cenários de teste planejados, serão analisados na prática como são realizados os testes e a verificação dos seus resultados.

3.1 Arquitetura geral para sistemas de telemetria e controle remoto

Nesta subseção é elucidado o que são sistemas de telemetria, qual seu escopo e alguns conceitos gerais de sistemas embutidos de telemetria e controle remoto e quais seus elementos principais.

Um sistema de telemetria é um sistema de monitoramento remoto que realiza medições remotamente de variáveis do ambiente onde se encontra inserido, como temperatura, velocidade, localização e outras variáveis que interessem a aplicação. Esses dados são processados pelo equipamento e depois são enviados para um servidor aplicação.

Sistemas de telemetria por definição são meramente monitores, mas pode-se adicionar a capacidade de atuação. Um sistema com a capacidade de atuação que pode ser comandado remotamente, isto é, obedecer a um telecomando, é caracterizado como sistema com controle remoto

Sistemas de telemetria e telecomando são compostos por pelo menos um módulo embarcado que possui sensores acoplados para perceber as variáveis de ambiente e, geralmente, um servidor que faz o registro das medições e pode comandar o equipamento através de comandos remotos.

Os sensores e atuadores variam com a aplicação, e podem ser enviados comandos ao equipamento tanto localmente, através de uma interface local, quanto por telecomando, remotamente pelo servidor web.

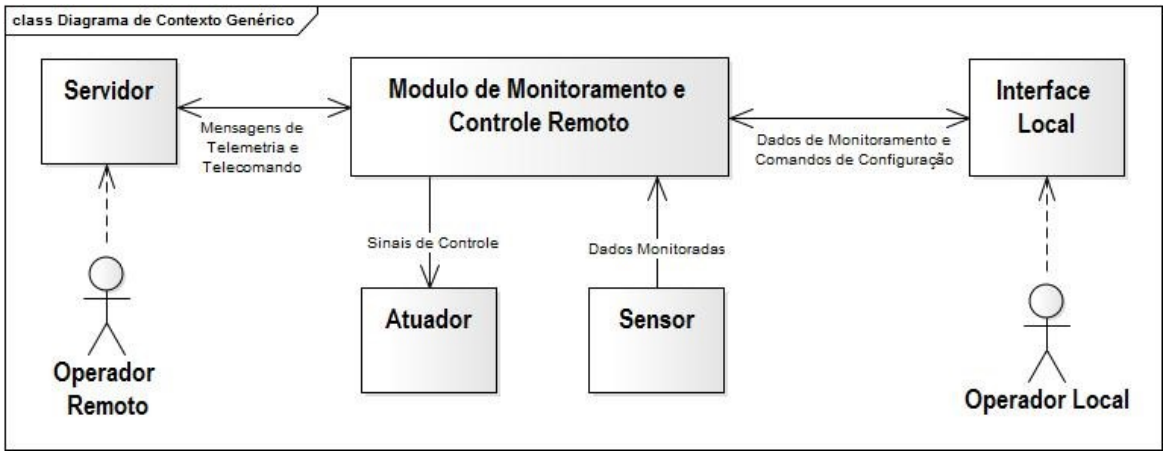

Figure 4 Diagrama de contexto geral para sistema de telemetria

O servidor é responsável por armazenar todas as medições recebida polo módulo de monitoramento e também pode enviar telecomandos para atuação em algum sentido ou para fazer configuração remota.

O módulo de monitoramento é o elemento que controla os sensores e atuadores. Esse elemento faz a aquisição de dados através dos sensores e constrói as mensagens para serem enviadas ao servidor. Esse módulo é o elemento principal da aplicação de monitoramento, porque é quem faz a aquisição de dados através dos sensores, armazena os dados em memória, envia a telemetria ao servidor, realiza filtragens das entradas analógicas e digitais para precisão das amostragens. O envio de dados ao servidor depende de alguma estrutura de comunicação a distância como um modem GPRS.

Pode-se ter diversos tipos de sensores a depender do propósito da aplicação. Sensores de temperatura, GPS (*Global Positioning System*) e sensores de movimento são muito comuns em diversas aplicações. A infraestrutura de simulação HIL substitui ou interage exatamente com estes sensores para simular condições reais de ambiente com o intuito de verificar o comportamento do sistema embarcado diante desse cenário.

Portanto, é possível simular quaisquer cenários virtualmente apenas instalando uma estrutura de simulação HIL, que tem um modelo matemático dos fenômenos do ambiente de teste, para estimular o módulo de monitoramento. Por exemplo, simulação HIL permite que sejam geradas coordenadas de GPS para o sistema embutido, ou simular temperatura para testar alarmes de faixa de temperatura limite facilmente. Isto é, HIL torna possível colocar o sistema de maneira menos custosa e segura, o equipamento em cenários de teste quaisquer, inclusive cenários hostis.

3.2 Arquitetura geral da infraestrutura HIL para sistemas de telemetria

Uma infraestrutura de simulação Hardware-in-the-Loop possui elementos de hardware, que geram sinais analógicos ou digitais para o sistema sob teste e elementos de software que implementam a inteligência dos elementos de hardware e os modelos matemáticos para simular as condições de ambiente.

Portanto, para construir uma simulação HIL que possa validar o SUT há duas etapas:

- Construir modelos que simulem as condições do ambiente de testes, isto é, criar aplicações que simulem um determinado comportamento do ambiente;
- Analisar como essas condições se tornarão estímulos para o SUT do ponto de vista de entrada e saída, isto é, planejar como serão enviados esses estímulos simulados, do ponto de vista de hardware.

Para qualquer metodologia de testes é imperativo que se saiba todos requisitos funcionais e não funcionais do sistema. Pois, os cenários de teste são construídos para submeter o SUT a condições reais e verificar como o equipamento se comporta de acordo com os requisitos. Sendo assim, os modelos devem possuir a capacidade de gerar as condições de todos os cenários de testes.

Cada sensor ou cada variável do sistema que precise ser simulada para construir o cenário de teste precisa ser analisada para que simule fielmente a condição real. Por exemplo, ao simular uma coordenada de GPS, deve-se analisar como estas são criadas por dispositivos de GPS para simulá-las por software.

Com a aplicação da simulação HIL para simular o que é percebido pelo software embarcado, os cenários de teste podem ser hostis como se quiser, para validar a tolerância e robustez do software, sem prejuízo aos protótipos e com total segurança, inclusive sendo possível a realização em laboratório e permitindo a replicação.

A arquitetura da infraestrutura HIL desenvolvida nesse trabalho é suficientemente genérica para esse tipo de sistema de telemetria e controle remoto. A arquitetura modular permite que sejam adicionados módulos simuladores, para substituir sensores e atuadores. Portanto, permite submeter a testes diferentes tipos de sistemas embarcados de telemetria. Cada módulo representa uma variável do sistema que pode ser simulada através da infraestrutura.

Uma simulação interessante realizada neste trabalho que exemplifica a ideia de um módulo simular uma variável monitorada pelo sistema, foi simular um hidrômetro através da infraestrutura. Com um módulo de software para gerar pulsos através de um Arduino Uno, foi possível criar um cenário de teste em que era simulada a passagem de água em um hidrômetro para validar se o sistema permitia a passagem de 250L e depois disso era capaz de atuar e interromper o curso da água. Esse teste é bem caricato, pois várias vantagens da metodologia que foram mencionadas anteriormente estão presentes. Isto é, foi possível fazer o teste de validação com passagem de uma grande quantia de água em laboratório, sem realmente haver nenhuma gota de água. Portanto, não houve necessidade de deslocar o equipamento para um ambiente preparado para fazer testes com água, o experimento pôde ser replicado diversas vezes sem custos adicionais e com um ambiente totalmente controlado.

Do exemplo mencionado é possível imaginar os ganhos produtividade, redução de custos e segurança para sistemas críticos, onde se pode perder protótipos devido as condições dos cenários de teste, como é o caso da indústria de aviação e automobilística.

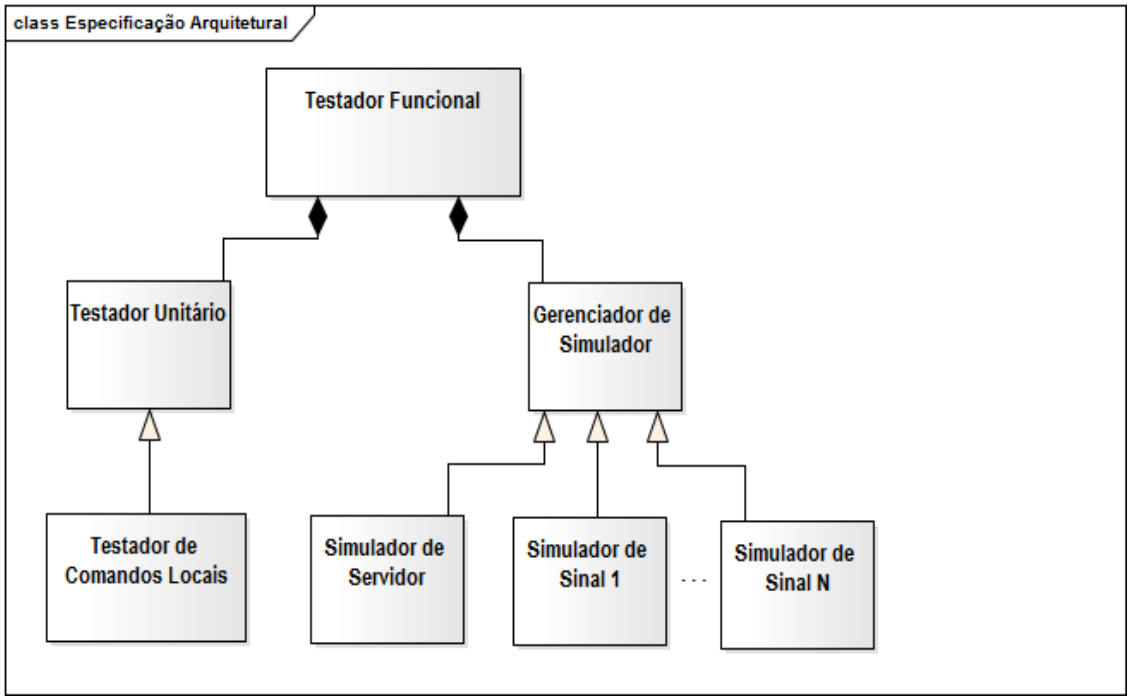

Figure 5 Arquitetura geral da infraestrutura HIL

3.3 Descrição dos módulos componentes da arquitetura

Nesta seção serão descritos os detalhes do funcionamento e implementação dos módulos presentes na arquitetura e como instanciar a arquitetura para que ela seja reutilizada em diversas aplicações de sistemas de telemetria.

A figura 5 mostra como é a arquitetura geral para sistemas de telemetria e controle remoto. Para essa classe específica de sistemas deve-se sempre ter um modulo que emule um servidor para receber as telemetrias e envias telecomandos, portanto é o único módulo de simulação que deve estar sempre presente, obviamente devido ao contexto do sistema.

A arquitetura é dividida em camadas que tem responsabilidades diferentes. A camada inferior corresponde aos módulos simuladores, isto é, o software que controla os dispositivos de hardware para

O lado esquerdo da figura 5 corresponde aos testes unitários, portanto, o que interessa nesse trabalho são os módulos do lado direito e o testador funcional.

i. Módulos simuladores

Os módulos simuladores possuem responsabilidade específica de acordo com os sinais que serão gerados para o SUT, isto é, tais módulos podem ser adicionados e alterados individualmente, afim de serem encaixados na arquitetura.

Cada cenário de testes possui um conjunto de sinais que devem ser simulados para reproduzir virtualmente as condições de campo. Basicamente, os módulos simuladores implementam uma modelagem dos sinais de entrada do sistema sob teste, onde cada módulo módulos é responsável por uma sinal.

Considerando um sistema com várias entradas, cada uma delas deve ser emulada por um módulo diferente de forma individual. Isso garante que sejá simples de incluir ou inibir entradas de acordo com o cenário de testes.

As simulações são, na verdade, comandos enviados a um microcontrolador ou hardware auxiliar, que permite gerar os sinais analógicos e digitais para o sistema sob teste. Portanto, tais módulos são geralmente compostos por:

- Módulo de software alto nível que comanda os sinais de acordo com os parâmetros definidos para cada cenário de teste;
- Módulo de software embarcado para um hardware auxiliar que implementa a modelagem dos sinais de forma parametrizável, para se adequar aos cenários de teste;

- Hardware usado para gerar sinais analógicos e digitais.

Para exemplicar, vamos considerar uma aplicação um equipamento faça a telemetria do fluxo de água que passe por uma saída de água. O sistema consiste então em um sensor de fluxo instalado na saída de água e uma unidade de monitoramento que faz a medição do volume que passa.

O sensor é associado a uma entrada digital do sistema de testes, portanto, é necessário simular uma onda quadrada para simular pulsos que seriam gerados pelo sensor de fluxo. Para gerar essa onda quadrada pode-se usar um microcontrolador auxiliar simples, como um Arduino Uno. No arduino, grava-se o firmware (software embarcado) para gerar a onda quadrada através das saídas PWM (Pulse Width Modulation). O pulso deve ser parametrizável em relação ao período em nível lógico alto e baixo, para que o modelo de onda seja mais flexível.

PWM, do inglês Pulse Width Modulation, é uma técnica utilizada por sistemas digitais para variação do valor médio de uma forma de onda periódica. A técnica consiste em manter a frequência de uma onda quadrada fixa e variar o tempo que o sinal fica em nível lógico alto. Esse tempo é chamado de duty cycle, ou seja, o ciclo ativo da forma de onda. No gráfico abaixo são exibidas algumas modulações PWM:

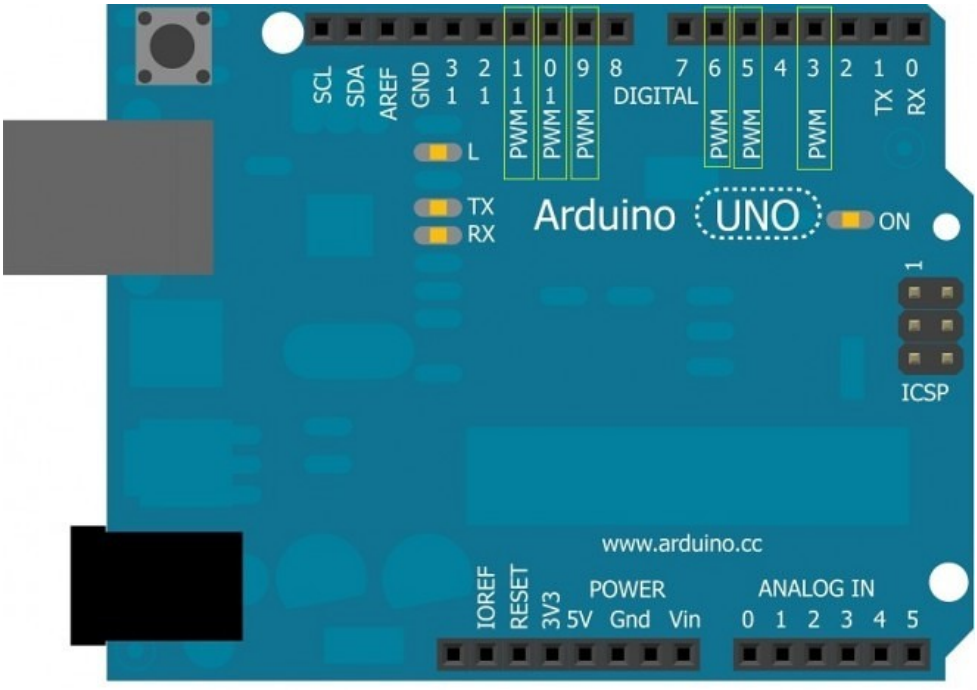

Figure 6 PWM do Arduino Uno

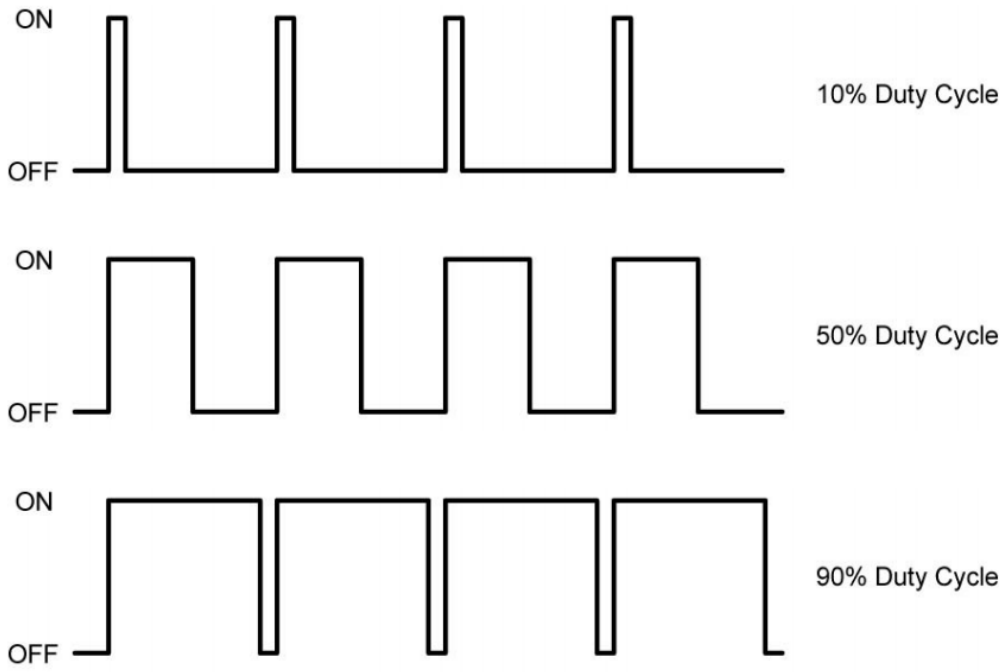

Figure 7 Onda quadrada para simular pulsos do hidrômetro

Agora, tem-se a possibilidade de gerar um pulso de onda quadrada, com tempo em nível alto e tempo em nível baixo parametrizáveis, através do arduino que pode simular um pulso de onda quadrada na entrada digital onde estaria o hidrômetro, no sistema sob teste. É importante notar que os pulsos são parametrizáveis, pois o firmware do Arduino não deve ter nenhuma inteligência sobre o teste que será realizado, sendo assim deve ser o mais flexível possível para que o pulso seja reutilizável para outro propósito.

Por hora, tem-se apenas um pulso de onda e para gerar as simulações é necessário gerar vários pulsos, além de colocar os parâmetros para gerar a onda na forma que se quer.

Portanto, módulo de software mais alto nível, sabendo que condições devem ser emuladas de acordo com as características do ambiente real, pode gerar a onda que melhor simule o comportamento que teria a água passando pelo sensor de fluxo.

O módulo de software que simula a onda através do Arduino é gerada por uma aplicação em Java, que envia o comando para iniciar a simulação e os parâmetros para o microcontrolador auxiliar. Com um cabo USB é possível enviar dados ao Arduino pelo computador (Host). O Arduino, por sua vez, tem sua saida PWM ligada a entrada do hidrômetro.

A classe que implementa esse comportamento é executada como uma Thread de Java, que é um mecanismo da linguagem para execução de códigos em paralelo. Sendo assim, é possível que os módulos sejam executados ao mesmo tempo, simulando as entradas do sistema realisticamente. **O código completo para o módulo do hidrômetro está no Apêndice A**.

Método de geração de pulso, que retorna o número de pulsos gerados:

```
private boolean generatePulses(int pulseAmount){

    String pulseCOmmand = "IPULSO," + pulseAmount + "," +
timeOn + "," + timeOff + "F"; // Comando para o Arduino
iniciar a geração de  pulsos com esses parâmetros
    String answer = "";
    long processingTime = 1000;
    long responseTimeout = (timeOn + timeOff)*pulseAmount +
processingTime;

    // Zera o buffer de recepção antes de enviar o comando
    this.comHidrometro.cleanPort();
    // Envia o comando
    this.comHidrometro.escreverPortaCom(pulseCOmmand);
    // Aguarda o recebimento da resposta
    answer = this.comHidrometro.readAnswer(responseTimeout);

    return answer.contains("Pulses genetated");
}
```

ii. Gerenciador de simulação

Os módulos simuladores podem ser tão numerosos quanto as entrada do sistema a serem emuladas. Porém, as simulações precisam ser configuradas para reproduzirem os diferentes cenários de testes. Isto quer dizer que deve haver um gerenciador reuna os parâmetros para configurar os módulos simuladores para reproduzir um caso de teste específico. Citanto o exemplo anterior, as variáveis que configuram a forma da onda quadrada e a quantidade de pulsos gerados podem ser configuradas pelo gerenciador de simulação (na figura 5, apenas Simulador).

Sendo assim, o gerenciador provê uma interface de configuração dos cenários de simulação, mudando parâmetros, adicionando ou retirando sinais de entrada. Basicamente é um módulo que inicializa as Threads para executarem os simuladores paralelamente.

No exemplo abaixo, p1 e p2 são simuladores para reproduzir algum sinal de entrada do sistema. Todos podem ser executados em paralelo pelo Gerenciador de simulação, além disso, pode-se passar parâmetros para cada simulação como mencionado anteriormente.

Método para execução de simulações pelo gerenciador de simulação:

```
public class Teste {
public static void main(String[] args) {

      Programa p1 = new Programa();
      p1.setId(1);

      Thread t1 = new Thread(p1);
      t1.start();

      Programa p2 = new Programa();
      p2.setId(2);

      Thread t2 = new Thread(p2);
      t2.start();

    }
}
```

Deste modo, as sinais de entrada emulados podem ser adicionados, adicionando-se um módulo de simulação e executando a thread correspondente a este no Gerenciador de simulação.

iii. Simulador de Servidor

Esse módulo é o mais importante da infraestrutura, pois é o simulador que permite emular a interação com o servidor.

Equipamentos de telemetria enviam mensagens a um servidor e algumas vezes possuem capacidade de atuar sobre o sistema quando comandado remotamente. Neste trabalho, foi implementado um servidor de socket para simular a interação com o servidor Web da operação. Essa simulação foi necessário porque até o término do desenvolvimento, o servidor ainda não tinha sido concluído, gerando uma dependência prejudicial entre as equipes.

Socket é um ponto de comunicação entre duas máquinas. Portanto, usando-se a clase de java ServerSocket é possível criar uma aplicação cliente-servidor entre o equipamento e o Host, computador que executa a infraestrutura Hardware-in-the-Loop. Assim, as mensagens enviadas pelo equipamento são direcionadas para o Host, sendo possível receber as telemetrias e enviar telecomandos ao equipamento.

Classe responsável por abrir conexão e tratar as mensagens recebidas:

```
import br.tomus.serversocket.Server;

public class ServerTest
{
    public static void main(String[] args)
    {
        ByteArrayAvailable byte_array_available = new
ByteArrayAvailable();
        Server server = null;

        try
        {
            try
            {
           /* Instancia um servidor na porta 8051, podendo
           abrir 5 conexões e recebe os dados na variável
           byte_array_available */
                server = new Server(8051, 5, byte_array_available);
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }

            if (server != null)
            {
```

```
                while (true)
                {
                    try
                    {
                        Thread.sleep(2000);
                    }
                    catch (InterruptedException e)
                    {
                        e.printStackTrace();
                    }

                    if (server.isRunning())
                    {
                        System.out.println("Server is running...");

                        if (server.connectionIdExist(1))
                        {
                            System.out.println("Sending byte array
to equipment of connection 1...");

                           /* Envia bytes a um terminal com Id de
conexão 1*/
                            server.write(1, new
byte[]{(byte)0xaa,(byte)0x00,(byte)0x00,(byte)0x80});
                        }
                    }
                    else
                    {
                        System.out.println("Server crash!");
                        break;
                    }
                }
            }
            else
            {
                System.out.println("Server is null!");
            }
        }
        finally
        {
            if (server != null)
            {
                server.shutdown();
            }
        }
    }
}
```

Classe responsável receber os dados da conexão aberta previamente, e colocá-los num array de byte:

```
import br.tomus.serversocket.listening.IListeningByteArrayAvailable;

public class ByteArrayAvailable implements
IListeningByteArrayAvailable {

    public void handlerNewByteArray(long id_connection, byte[]
array)
    {
        System.out.println("Mensagem da conexao \"" + id_connection
+ "\":\n" + (new ByteQueue(array)).toString());
    }

     }
```

Todas as mensagens que chegam ao Host são armazenadas em um array e ganham uma identificação da conexão.

As mensagens são de fato tratadas na classe ServerTest, onde se pode manipular os bytes recebidos.

A partir dessa comunicação simples, é possível expandir as classes para adicionar protocolos de mensagens referentes a formato, resposta e telecomandos.

iv. Reutilizando a arquitetura para outras aplicações

Essa subseção mostra como é possível e simples de utilizar a arquitetura para outras aplicações. Considere uma aplicação de rastreamento de veículos, então a telemetria enviada ao servidor consiste basicamente na localização geográfica, velocidade e data/hora.

Então, deve ser possível simular coordenadas de GPS, velocidade e data/hora através da infraestrutura. Um módulo de simulação de GPS deve ser implementado para gerar sentenças NMEA (National Marine Electronics Association) para o sistema sob teste emulando o próprio dispositivo de GPS, data/hora e velocidade.

Para realizar essa simulação, não há necessidade de ter um microcontrolador auxiliar, as mensagens NMEA podem ser geradas pelo módulo de software alto nível por uma aplicação Java e enviadas ao SUT através de comunicação serial. Portanto, para essa simulação é preciso apenas um conversor USB (Universal Serial Bus) para a interface serial do microcontrolador SUT que é TTL (Transistor-Transistor Logic).

Sendo assim, as sentenças são geradas a partir da aplicação em java e enviadas através de uma comunicação USB para o conversor, que por sua vez comunica-se com o microcontrolador através de uma interface serial.

A ligação física se dá conectando os pinos do MCU onde estaria o dispositivo GPS aos pinos do conversor para USB-TTL, isto é, simplesmente enviando as sentenças ao SUT através de uma interface serial. **O código completo para o módulo do GPS está no Apêndice B**.

Método usado para gerar sentenças NMEA, simulando coordenadas de GPS:

```
public String generateNMEASentence(){

        /*Generate sentence with the current location*/
        this.logger.info("Altitude- " + currentLocation.altitude);
        this.logger.info("Longitude- " + currentLocation.longitude);
        this.logger.info("Latitude- " + currentLocation.latitude +
"\n");
        String nmeaSentence =
Route.createNMEASentences(currentLocation,speed,fix,hdop);
        this.logger.info("Sentença NMEA:\n" + nmeaSentence);
        /*Update location*/
        currentLocation =
Route.generateNorthPoint(currentLocation,speed);

        return nmeaSentence;
    }
```

Tendo a implementação de um módulo simulação de coordenadas de GPS, o próximo passo é colocar criar uma Thread no Gerenciador de simulação para executá-lo e configurar os parâmetros de coordenada e velocidade que sejam necessários para desenvolver o cenário de testes.

```
public class Teste {
    public static void main(String[] args) {

        Programa p1 = new Programa();
        p1.setId(1);

        Thread t1 = new Thread(p1);
        //t1.start(); //Do not execute any other simulations

        Programa p2 = new Programa();
        p2.setId(2);

        Thread t2 = new Thread(p2);
       //t2.start(); //Do not execute any other simulations

        GPSSimulator GPS = new GPSSimulator();
        GPS.setId(2);


    /* Thread to simulate GPS coordinates */
        Thread GpsThreadh = new Thread(GPS);
        GpsThreadh.start();
    /* Thread to simulate GPS coordinates */

    }
}
```

Do exemplo anterior percebe-se que é possível adicionar módulos a infraestrutura de forma fácil através da arquitetura. A única mudança se deu ao excluir as Threads que simulavam hidrômetro e adicionar a que simula coordenadas de GPS.

É possível encaixas quaisquer módulos para simulação de sinais sem prejuízo a arquitetura, contanto que o simulador do servidor seja mantido.

4. Sistema específico testado pela infraestrutura HIL

Neste tópico será apresentado o sistema específico conceitualmente similar ao da Tomus em que foi usado o método de simulação HIL. É importante lembrar que HIL pode ser usado tanto para validação de sistemas completos, como teste de funcionalidade, quanto para testes de integração. Testes de integração validam se o hardware e o software têm potencial para entregar as funcionalidades, embora não sejam testadas ainda as funcionalidades.

Foi desenvolvido nesse trabalho uma infraestrutura de simulação de ambiente para testar e validar um sistema de monitoramento e controle remoto de reservatórios hídricos. Este sistema é responsável por despejar um volume específico de água em um reservatório, monitorando em tempo real o volume, enquanto este é despendido, e atuando para interromper ou liberar o fornecimento de água de acordo com as configurações de operação e critérios de funcionamento do equipamento. Essas configurações são parametrizáveis e podem ser alteradas através de um servidor web integrado ao equipamento. O servidor também recebe do equipamento as amostragens dos sensores para monitoramento remoto.

Entre as principais funcionalidades do sistema, estão:

- Amostrar a vazão na saída de água, temperatura;
- Interromper ou liberar o fluxo de água;
- Comunicar-se com um servidor web para enviar telemetria e receber telecomandos de configuração;
- Monitorar o nível de água no reservatório;
- Enviar alarmes ao servidor ao perceber algum cenário indevido, altas temperaturas, ou fluxo de água quando sem permissão;
- Fornecer uma interface de usuário para acionamento local do preenchimento do reservatório.

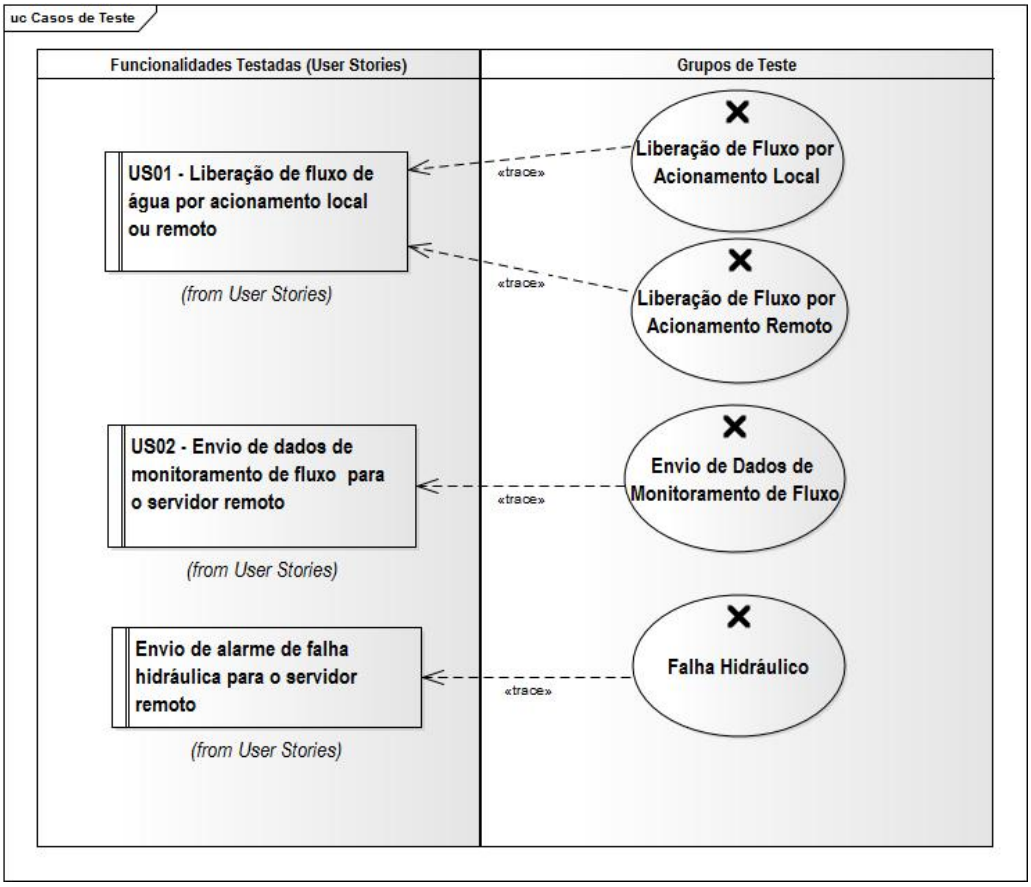

Figure 8 Casos de teste

Mais precisamente, o equipamento é capaz de monitorar a vazão de água despejada no reservatório, através de um hidrômetro colocado na saída de água, ou torneira. Possui também, a capacidade de interromper automaticamente o curso da água quando o volume despendido for suficiente, através de uma válvula solenoide.

O equipamento também monitora o nível de água através de um sensor de pressão colocado no reservatório. Quando o nível da água atinge um limiar específico, o equipamento envia um alarme ao servidor informando que há sobre nível de água.

O servidor configura o equipamento remotamente através de telecomandos. As configurações de operação são: os períodos de amostragem, o critério de parada do enchimento, que pode ser volume ou tempo. O servidor por enviar telecomandos para fechar ou abrir a válvula solenoide.

Há uma interface homem-máquina presente no sistema, que serve para iniciar o enchimento de um reservatório com água localmente.

Portanto, o sistema pode ser dito composto de um módulo de monitoramento e controle, um sensor de vazão, para monitorar a vazão de água que é colocada no reservatório,

um sensor de nível, uma válvula solenoide para atuar permitindo ou interrompendo o fluxo de água e uma interface usuário-máquina para iniciar o enchimento do reservatório. Além de um servidor web para controle remoto de atuadores e reconfiguração.

A arquitetura foi instanciada para ter um servidor web simulando telecomandos para configuração do módulo de controle e monitoramento e simulação da IHM. O sistema foi dividido em duas partes: módulo de monitoramento e IHM.

A IHM é uma interface sem fio que indica os estados do sistema e ao usuário local. O microcrontolador presente no monitor envia os estados do sistema através de rádio para a IHM que é responsável por ligar leds para indicar os estados sistema.

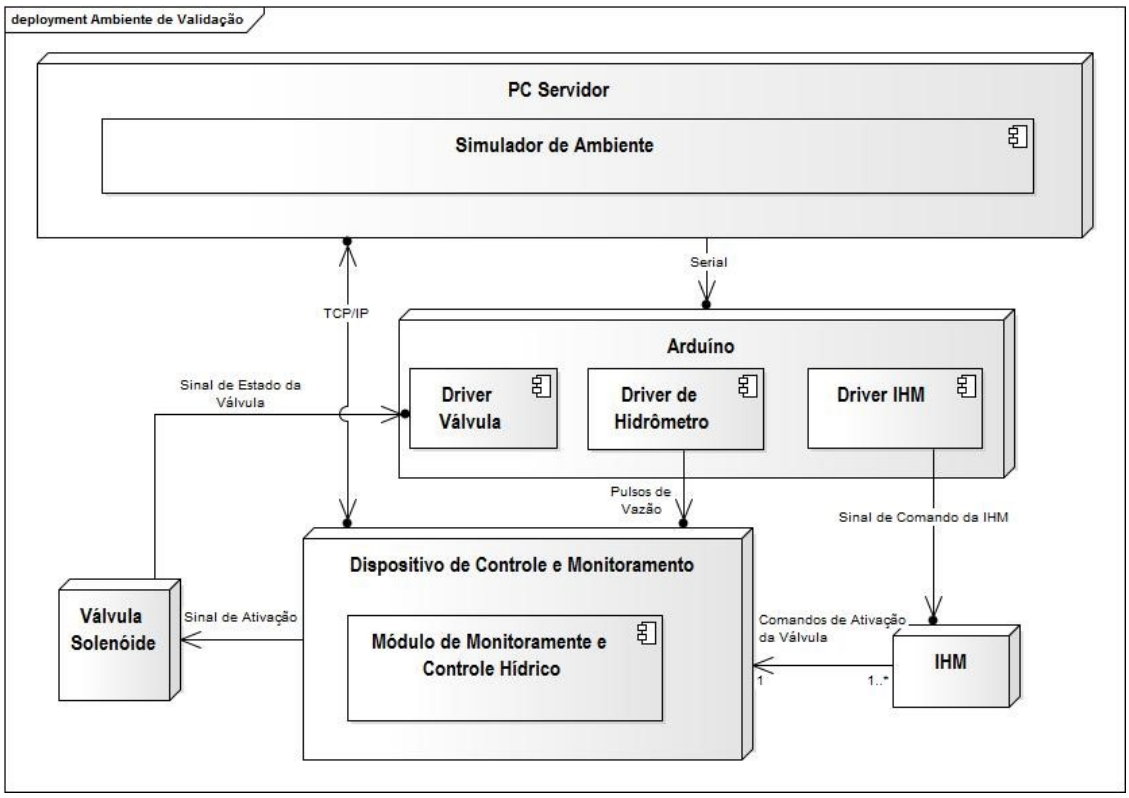

Figure 9 Instância da arquitetura da infraestrutura para o caso de teste

O teste para esse sistema foi dividido em duas etapas: testes de funcionalidade do módulo de monitoramento e os testes de integração da IHM. Essa divisão foi aplicada para isolar o funcionamento e as complicações do sistema entre módulo monitor e IHM, facilitando o desenvolvimento e os testes.

Para testar o módulo monitor, foi possível usar um Arduino Uno como elemento de hardware para enviar as os estímulos virtuais ao SUT. Os modelos para construir os cenários de testes

foram implementados na linguagem Java. Então, uma aplicação em Java implementa o modelo do cenário de teste e envia comandos para um Arduino UNO para este gerar os estímulos para o SUT.
A validação do equipamento inclui testar sua interação com o servidor web, que no momento dos testes não estava pronto. Portanto, foi implementado na infraestrutura uma simulação de servidor de socket (usando a API de socket do Java) para enviar telecomandos e validar o protocolo das mensagens de telemetria.

Descrição dos módulos de software e hardware para simular as variáveis de entrada:

- PC host para executar aplicação de cenário de teste da infraestrutura;
- Comunicação entre a aplicação e o Arduino é através de uma interface serial.
- Aplicação para gerar pulsos, simulando um hidrômetro. Os pulsos gerados podem ser parametrizados em quantidade e duração, portanto é possível gerar 100 pulsos em 10s, por exemplo.;
- A saída do SUT que dispara a válvula solenoide, é conectada a uma entrada digital do Arduino;
- Os sinais do sistema embarcado que indicam seu estado são enviados para entradas digitais do Arduino, assim não é necessário usar a IHM para verificar os estados de alarme do sistema. Isto é feito automaticamente pela infraestrutura.

Para validar a IHM, esta deve ter a capacidade de se comunicar sem fio com o módulo de monitoramento e apresentar as configurações de piscada dos leds corretamente. Para isolar a complexidade do módulo monitor, isolou-se a interface local.

Um módulo para testar a a IHM foi adicionado à infraestrutura, simulando o comportamento do modulo de monitoramento. Todos os estados do módulo de monitoramento foram simulados e enviados como estímulos para a IHM.
Descrição dos módulos de software e hardware para simular as variáveis de entrada:

- PC host para executar aplicação que simula o módulo monitor;
- A infraestrutura envia comandos por uma interface serial a um módulo de rádio, que por sua vez está pareado com o rádio presente na IHM.

Neste caso, os estímulos dos estados simuladores eram enviados a IHM e a verificação era feita visualmente pelo testador, sendo assim, esse teste não foi totalmente automatizado.

5. Realização dos testes

Nesta seção serão mostrados como foram testados alguns requisitos do equipamento de acordo com os cenários de testes para validar as funções.

➢ Liberação do curso de água por telecomando

Caso de uso do equipamento: O servidor comanda o equipamento a liberar o fluxo de água, através da válvula solenoide e monitora o volume de água despendido.
Roteiro de teste:

- O Arduino é ligado ao PC host e sua saída digital é conectada ao módulo de monitoramento, em substituição do hidrômetro;
- Gerar telecomandos através da infraestrutura para liberação de água;
- Infraestrutura comanda o Arduino para gerar pulsos, que correspondam a 255L.

  Para passar no teste, o equipamento deve enviar um sinal a válvula para fechá-la. Então, após a geração dos pulos, a infraestrutura avalia se um sinal digital para a válvula foi percebido no momento em que ainda faltava serem gerados 5 pulsos. Dessa forma, pode-se testar os telecomandos de liberação de fluxo de água e a precisão do monitoramento do volume.

➢ Indicativos visuais da IHM

Caso de uso do equipamento: O equipamento envia os estados em que se encontra para a interface de usuário, para indicar quando há passagem de água, falta de água, alarmes de sobre temperatura ou falha de comunicação com o servidor.
Roteiro de teste:

- O Arduino é ligado ao PC host e sua saída analógica é conectada ao módulo de monitoramento, em substituição do sensor de temperatura;
- Gerar telecomandos através da infraestrutura para liberação de água;
- Não deve ser gerado nenhum pulso para simular falta de água.

Para passar no teste, a IHM deve apresentar alarme de falta de água, pois houve comando da válvula solenoide para liberação do fluxo de água, porém nenhum pulso foi detectado pelo

equipamento. Esses indicativos locais são muito importantes para a operação, portanto devem representar fielmente o estado do sistema.

6. Conclusões e Trabalhos Futuros

No projeto da Tomus Soluções aplicou-se metodologias de testes e metodologias ágeis de gestão que certamente contribuíram para que o tempo de desenvolvimento fosse reduzido. Porém, a empresa tinha problemas mais graves devido a alta incidência de falhas prejudiciais a operação em campo, que geravam altos custos de manutenção.

Os equipamentos passavam por testes unitários durante o desenvolvimento, porém os testes de produto completo eram realizados colocando-se os protótipos em campo, cujos testes de não eram ideais, pois submetiam o sistema às condições reais sem a devida validação. Portanto, era comum que muitos *bugs* fossem descobertos já com equipamentos em campo, fazendo-se necessário trazer o equipamento para realizar correções emergenciais e devolução ao ambiente real sempre que problemas eram detectados. Isso gerava um ciclo de gastos significantes.
Embora a evolução de uma infraestrutura HIL adicione complexidade ao teste e requeira tempo, o retorno em redução de manutenção e diminuição significativa na quantidade de bugs que prejudicam a operação são suficientes para investir nessa metodologia.

De acordo com a tabela de acompanhamento de ciclo de desenvolvimento de projetos da Tomus, confirmam-se os resultados da aplicação da metodologia HIL e da implementação da infraestrutura. Todos os projetos são considerados como tendo o mesmo porte.

- No projeto A, não foram implementados nenhuma metodologia de testes bem definida.
- No projeto B, foram aplicados testes unitários e HIL após o desenvolvimento já ter sido concluído.
- No projeto C, a simulação HIL esteve presente durante todo desenvolvimento, para realizar testes de integração e testes de validação do produto final.

Os testes do projeto A foram realizados em campo, isto é, ao fim do desenvolvimento os equipamentos foram colocados em ambiente real, ainda que em operação apenas para fins de testes. Os equipamentos eram instalados e monitorados ao longo de um período de 5 meses. Caso houvesse falha, o equipamento era recolhido para correção. Vale salientar que os Bugs póstumos na tabela se referem a bugs já na operação, e não as falhas encontradas durante os meses de teste em campo.

No projeto B, houve a preocupação em realizar testes de validação utilizando técnicas de Hardware-in-the-Loop. Apesar de durar 6 meses, isso incluiu o tempo de desenvolvimento da infraestrutura e o testes propriamente ditos.

O tempo investido na infraestrutura foi de suma importância para a redução significante do tempo de testes do projeto C, em que os testes foram finalizados em apenas um mês. Isso se deu devido a generalidade da arquitetura e sua modularidade, fazendo com que fosse possível o reaproveitamento e mitigar o grande *overhead* da sua implantação.

Table 1 Tabela de acompanhamento de projetos

Projeto	Espec	Espec	DEV	DEV	Testes	Testes	Correção	Correção	Dias Corridos	Bugs Póstumo
A	10/1/14	7/31/15	4/1/15	1/26/16	1/26/16	8/12/16	4/21/17	6/16/17	858	75
B	10/21/15	12/22/15	12/23/15	9/30/16	10/3/16	3/17/17	3/20/17	4/20/17	540	2
C	5/30/16	8/30/16	8/31/16	11/25/16	3/20/17	4/14/17	4/17/17	4/20/17	206	0

Pode-se perceber que a redução significativa da quantidade de *bugs* encontrados nos equipamentos, correspondendo uma diminuição de 97%.

Outro fator importante a considerar é que falhas encontradas em ambiente de desenvolvimento são menos onerosas por serem mais fáceis de corrigir e revalidar do que aquelas que são detectadas em campo.

Portanto, a infraestrutura desenvolvida conseguiu resultados positivos nos seguintes critérios:

- Redução no *Time to Market;*
- Redução custos de manutenção;
- Melhora na confiabilidade do sistema, apresentando menos *bugs*;
- Redução dos custos de testes de campo.

7. Trabalhos Futuros

Como possíveis trabalhos futuros pode-se apontar:

- Desenvolvimento de uma infraestrutura que permita a introdução de falhas de hardware A inserção de falhas de hardware está presente em muitos sistemas de teste HIL. Essa função permite a introdução de falhas nos sinais trocados entre a infraestrutura HIL e o restante do sistema, para testar, caracterizar ou validar o comportamento de um dispositivo em condições de falha. Para isso, você pode inserir unidades de inserção de falhas (FIU) entre as interfaces de E/S e a infraestrutura, passando da operação normal dos sinais de interface para uma condição de falha, como "curto com o terra" ou "circuito aberto".
- Desenvolvimento de interface gráfica para descrição do sistema testado, dos critérios de avaliação e análise dos resultados.

Bibliografia

[1] James W. Grenning, Robert C. Marin – “Test Driven Development for Embedded C” – 2011

[2] BUENO, L. D. A.; MORENO, G. B. Z. L.; SWEET, B. Three-Layer Software Architecture Inspired by AUTOSAR Applied in a Telemetry System for a Radio-Controlled Aircraft. 2013 IEEE INTERNATIONAL CONFERENCE on ELECTRO/INFORMATION TECHNOLOGY. Rapid City: [s.n.]. 2013.

[3] Ecker, Wolfgang. “Hardware-dependent Software: Principles and Practice”, section 10, Three-Layer Software Architecture Inspired by AUTOSAR

[4] Lisa Crispin, Janet Gregory. “Agile Testing: A Practical Guide for Testers and Agile Teams”, primeira edição.

[5] Fabio Scippacercola, Roberto Pietrantuomo, Stefano Russo e András Zentai. “Model-Driven Engineering of a Railway Interlocking System”

[6] NMEA Data Sentences, http://www.gpsinformation.org/dale/nmea.htm.

APÊNDICE A – CÓDIGO JAVA MÓDULO DE SIMULAÇÃO DE PULSOS DO HIDRÔMETRO

```
package br.com.tomus.sca.Tester.testDriver;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

import org.apache.log4j.Logger;

import br.com.tomus.sca.comm.SerialDriver;



public class HidrometroSimulator implements Runnable {
    private static String PORTA_HIDROMETRO = "COM36";
    private static int BAUD_HIDROMETRO = 9600;
    private SerialDriver comHidrometro;

    private int timeOn; /* Tempo ON de cada pulso*/
    private int timeOff; /* Tempo OFF de cada pulso */
    private long pulsesGenarated; /* Quantidade de pulos gerados */

    public HidrometroSimulator(SerialDriver comHidrometro) {
     this.comHidrometro = comHidrometro;
     /* Inicializa o simulador */
     initHidrometroSimulator();
    }

    private void initHidrometroSimulator(){
     /*Inicia o logger*/
     //this.logger =
Logger.getLogger("br.com.tomus.sca.Tester.testDriver.HidrometroSimul
ator");
     System.out.println("Iniciando Simulador do HidrÙmetro");

     /*Inicia ConfiguraÁ„o do HidrÙmetro*/
     timeOn = 5;
     timeOff = 5;
//   timeOn = 2; Caso pau
//   timeOff = 2;

     pulsesGenarated = 0;

     /*Inicia a porta de comunicaÁ„o com o hidrÙmetro*/
     this.comHidrometro.restartConnection();
```

```
      /*Aguarda a inicializaÁ„o do gerador de pulsos*/
      while(!this.comHidrometro.readAnswer(100,
500).contains("INIT"));

      System.out.println("Simulador do HidrÙmetro Iniciado");
    }

    private boolean generatePulses(int pulseAmount){
      String pulseCOmmand = "IPULSO," + pulseAmount + "," + timeOn +
"," + timeOff + "F";
      String answer = "";
      long processingTime = 1000;
      long responseTimeout = (timeOn + timeOff)*pulseAmount +
processingTime;

      // Zera o buffer de recepÁ„o antes de enviar o comando
      this.comHidrometro.cleanPort();
      // Envia o comando
      this.comHidrometro.escreverPortaCom(pulseCOmmand);
      // Aguarda o recebimento da resposta
      answer = this.comHidrometro.readAnswer(responseTimeout);

      return answer.contains("Pulses genetated");
    }

    public void run() {
      DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy
HH:mm:ss");
      Calendar calendar = Calendar.getInstance();
      int pulsesAMount = 2800;

      System.out.println("Executando Simulador do HidrÙmetro");
            calendar.setTime(new Date());

            /* Gerar um pulso a cada 2 seg */
                  if(generatePulses(pulsesAMount)){
                        calendar.setTime(new Date());
                        this.pulsesGenarated += pulsesAMount;

                        System.out.println("[" +
dateFormat.format(calendar.getTime()) + "]" + "Pulsos Gerados: " +
pulsesAMount);
                        System.out.println("[" +
dateFormat.format(calendar.getTime()) + "]" + "Acumulado: " +
this.pulsesGenarated + "\n");
                  } else{
                        System.out.println("Falha ao Gerar Pulsos");
                  }
    }

    /**
     * MÈtodo usado para teste isolado do mÛdulo de simulaÁ„o do
hidrÙmetro
```

```
     * @param args
     */
    public static void main(String[] args) {
     /* Cria simulador*/
     HidrometroSimulator hidro = new HidrometroSimulator(new
SerialDriver(PORTA_HIDROMETRO,BAUD_HIDROMETRO));
     /* Cria thread de teste*/
     Thread hidroThread = new Thread(hidro);
     /*Executa Thread*/
     hidroThread.start();
    }

}
```

APÊNDICE B – CÓDIGO JAVA MÓDULO DE SIMULAÇÃO DE COORDENADAS GPS

```
package br.com.tomus.tlm9500.devicesSimulators.gps;


import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

import org.apache.log4j.Logger;
import br.com.tomus.tlm9500.comm.SerialDriver;
```

```
public class GPSSimulator implements Runnable {
    private static String GPS_PORT = "COM10"; // Port configured to
GPS communication
    private static int GPS_BAUDRATE = 4800; // Port configured to
GPS communication
    private SerialDriver commGPS; //Port configured to GPS
communication
    Logger logger; //Log de mensagens
    /*GPS*/
    private DateFormat dateFormat;
    private Date initialDate;
    private Location currentLocation;
    private int speed;
    private int fix;
    private double hdop;


    public GPSSimulator(SerialDriver commGPS){
        this.commGPS = commGPS;
        initGPS();
    }

    private void initGPS(){
        /*Inicia o logger*/
        this.logger =
Logger.getLogger("DevicesSimulator.GPSSimulator");
        this.logger.info("Iniciando Simulador do GPS");

        commGPS.restartConnection();

        /* Save current time with the right format*/
        dateFormat = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(new Date());
        calendar.add(Calendar.HOUR, 3);
        try {
            initialDate =
dateFormat.parse(dateFormat.format(calendar.getTime()));
        } catch (ParseException e) {
            this.logger.error("Não foi possível definir a data
atual!");
        }

        /*GPS*/
        Route.initRoute();
        currentLocation = new Location(initialDate,-8.042721,-
34.946141,27.0);
        speed = 0;
        fix = Route.VALID;
```

```
        hdop = 1.0;

        this.logger.info("Simulador do GPS Iniciado");
    }

    public String generateNMEASentence(){

        /*Generate sentence with the current location*/
        this.logger.info("Altitude- " + currentLocation.altitude);
        this.logger.info("Longitude- " + currentLocation.longitude);
        this.logger.info("Latitude- " + currentLocation.latitude +
"\n");
        String nmeaSentence =
Route.createNMEASentences(currentLocation,speed,fix,hdop);
        this.logger.info("Sentença NMEA:\n" + nmeaSentence);
        /*Update location*/
        currentLocation =
Route.generateNorthPoint(currentLocation,speed);

        return nmeaSentence;
    }

    /**
     * Gera uma setença NMEA com erro.
     * @param errorCode - Código do erro que deve ser introduzido na
sentença:
     *                          0 - Diferença horária de 10 min;
     *                          1 - Erro de checksum
     * @return
     */
    public String generateNMEASentence(int errorCode){
        String wrongCRC = "";
        String nmeaSentence = "";
        int crcIndex = 0;
        Date wrongDate;
        Location wrongLocation;

        /*Generate sentence with the current location*/
        this.logger.info("Altitude- " + currentLocation.altitude);
        this.logger.info("Longitude- " + currentLocation.longitude);
        this.logger.info("Latitude- " + currentLocation.latitude +
"\n");

        if(errorCode == 0){
            /*Cria uma Data Inválida*/
            Calendar calendar = Calendar.getInstance();
            calendar.setTime(new Date(0));
            System.out.println("Hora Definida: " +
dateFormat.format(calendar.getTime()));
            try {
                /*Cria uma sentença NMEA com data/horário errado*/
                wrongDate =
dateFormat.parse(dateFormat.format(calendar.getTime()));
                wrongLocation = new Location(wrongDate,0,0,0);
```

```
                nmeaSentence =
Route.createNMEASentences(wrongLocation,speed,fix,hdop);
            } catch (ParseException e) {
                e.printStackTrace();
            }
        }else if(errorCode == 1){
            /*Cria uma sentença correta e substitui o CRC correto
por um CRC inválido*/
            wrongCRC =
String.format("%02X",Route.calcChecksum("Wrong Sentence"));
            nmeaSentence =
Route.createNMEASentences(currentLocation,speed,fix,hdop);
            crcIndex = nmeaSentence.indexOf("*");
            nmeaSentence = nmeaSentence.substring(0,crcIndex) + "*"
+ wrongCRC + "\r\n";
        }

        this.logger.info("Sentença NMEA:\n" + nmeaSentence);

        return nmeaSentence;
    }

    public void run() {
        this.logger.info("Simulador do GPS Executando");

        /*Gera informações do GPS a cada 1 segundo*/
        while(true){
            commGPS.escreverPortaCom(generateNMEASentence() +
"\r\n");
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

    }

    /**
     * Método usado para teste isolado do módulo simulador de GPS
     * @param args
     */
    public static void main(String[] args) {;
        GPSSimulator gps = new GPSSimulator(new
SerialDriver(GPS_PORT,GPS_BAUDRATE));
        Thread gpsThread = new Thread(gps);

        /*Executar SImulador*/
        gpsThread.start();
    }

     }
```