UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PLANO DE TESTES PARA APLICAÇÃO EM SISTEMAS EMBARCADOS COM ARQUITETURA ORIENTADA A SERVIÇOS

VINÍCIUS FARIAS MARTINS

ORIENTADOR: **PROF. DR. ORIDES MORANDIN JUNIOR**

São Carlos - SP

Fevereiro/2025

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PLANO DE TESTES PARA APLICAÇÃO EM SISTEMAS EMBARCADOS COM ARQUITETURA ORIENTADA A SERVIÇOS

VINÍCIUS FARIAS MARTINS

Projeto de Dissertação ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Sistemas de Automação e Robótica.

Orientador: **Prof. Dr. Orides Morandin Junior**

São Carlos - SP

Fevereiro/2025

Folha de Aprovação

Defesa de dissertação de mestrado do(a) candidato(a) Vinícius Farias Martins, realizada em 25/02/2026

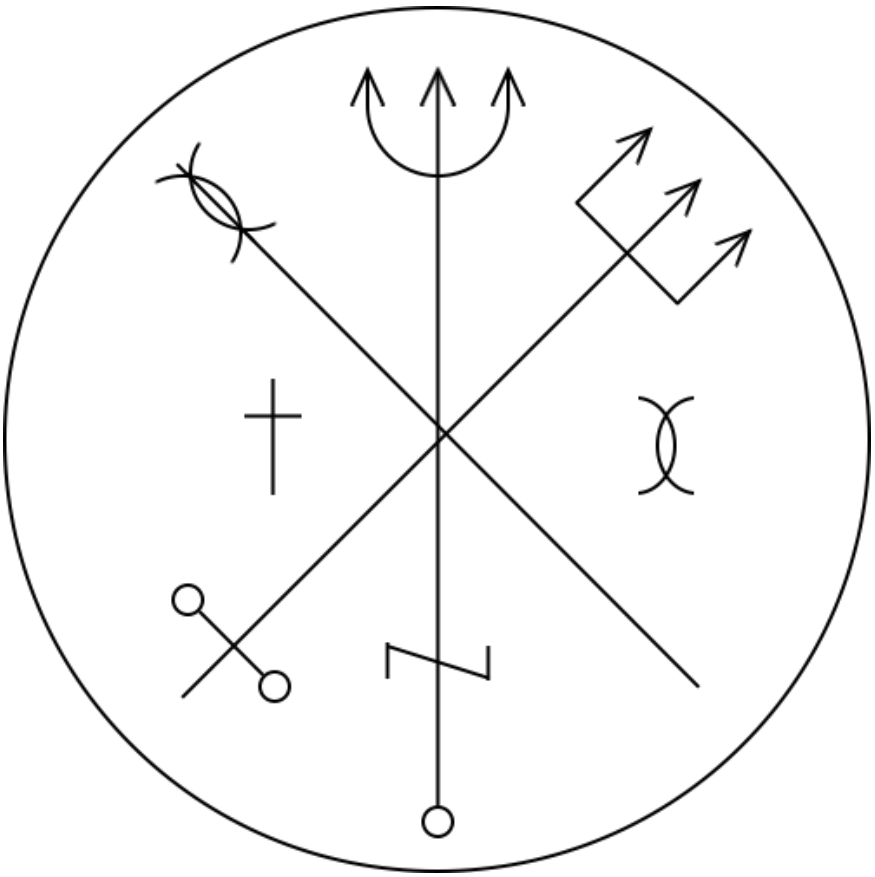
Comissão Julgadora

Prof(a) Dr(a) Orides Morandin Junior (UFSCar)

Prof(a) Dr(a) Wallace Pereira Neves dos Reis (IFRJ)

Prof(a) Dr(a) Alan Demétrius Baria Valejo (UFSCar)

O relatório de defesa assinado pelos membros da comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação



Dedicado a Exú Tiriri.

AGRADECIMENTO

Agradeço a todos os Orixás, principalmente a Oxalá, meu Pai Maior, e a todas as entidades espirituais que andam em terra por terem me dado a força, energia e disposição para completar esta etapa tão importante na minha vida.

Agradeço minha mãe, meu pai, minha irmã, meus amigos e amigas por terem me dado todo o amor, suporte e apoio necessários para alcançar mais uma conquista.

Ao Professor Orides, agradeço a oportunidade de poder me orientar neste trabalho, e por ter partilhado tantos ensinamentos que vou levar para o resto da vida.

Aos meus colegas de trabalho do laboratório Tear e do PET-EnC, por fazerem parte direta e indiretamente deste trabalho, especialmente ao Luan, meu grande companheiro de projeto.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

RESUMO

Um projeto de engenharia de software requer uma boa compreensão do sistema que está sendo criado para que os testes possam ser executados de forma mais eficiente. Com a crescente complexidade dos projetos de sistemas embarcados nas últimas décadas, é importante ter uma fase de testes bem definida para validar os componentes de software e hardware de um sistema embarcado de forma eficaz. Este trabalho propõe um plano de testes organizado, para considerável otimização de tempo de desenvolvimento, e que pode ser replicado para uso em qualquer sistema embarcado construído usando arquitetura orientada a serviços. A abordagem mestra de testes apresentada neste trabalho possui uma ordem definida de documentação e execução para garantir que todos os tipos de testes sejam feitos de maneira organizada. O plano de testes foi executado no AGV AD01, do laboratório Tear da Universidade Federal de São Carlos, de forma a obter resultados do funcionamento do veículo, como do sistema de controle de rodas, a correção automática do erro de percurso, e o comportamento dos parâmetros envolvidos no sistema como um todo. Uma análise dos resultados também foi feita, de forma a compreender os erros encontrados e fazer as correções necessárias.

Palavras-chave: Arquitetura orientada a serviços, automação, engenharia de software, engenharia de testes, sistemas embarcados.

ABSTRACT

A software engineering project requires a good understanding of the system being created so that tests can be executed more efficiently. With the increasing complexity of embedded systems projects in recent decades, it is important to have a well-defined testing phase to effectively validate the software and hardware components of an embedded system. This work proposes an organized test plan, for considerable optimization of development time, which can be replicated for use in any embedded system built using service-oriented architecture. The master testing approach presented in this work has a defined order of documentation and execution to ensure that all types of tests are performed in an organized manner. The test plan was executed on the AGV AD01, from the Tear laboratory of the Federal University of São Carlos, in order to obtain results on the vehicle's operation, such as the wheel control system, the automatic correction of path errors, and the behavior of the parameters involved in the system as a whole. An analysis of the results was also performed to understand the errors found and make the necessary corrections.

Keywords: Automation, embedded systems, service-oriented architecture, software engineering, testing.

Lista de Figuras

1.1 - Modelo espiral simples (adaptado de Pressman e Maxim, 2019) *7*
2.1 - Ilustração de um modelo de arquitetura baseado em SOA usado no laboratório Tear *12*
2.2 - Exemplo de diagrama de casos de uso *16*
2.3 - Exemplo de diagrama de atividades *17*
2.4 - Exemplo de diagrama de sequência *17*
2.5 - Exemplo de diagrama de classes *18*
3.1 - Ilustração do teste ao traduzir o diagrama de um projeto em SOA para UML *23*
3.2 - Diagrama de funcionamento de um teste unitário de serviço *25*
3.3 - Diagrama de funcionamento de um teste de integração de serviço *25*
3.4 - Diagrama de funcionamento de um teste unitário de majoritário *26*
3.5 - Diagrama de funcionamento de um teste de integração de majoritário *27*
4.1 - Diagrama geral da arquitetura de software do AGV AD01 (relatório interno do Tear) *31*
4.2 - Diagrama de caminho de dados dos serviços envolvidos no teste do S1 *37*
4.3 - Diagrama de caminho de dados dos serviços envolvidos no teste de integração do S3 e S4 *47*
4.4 - Diagrama de caminho de dados dos serviços envolvidos no teste do M1 *51*
4.5 - Diagrama de caminho de dados dos majoritários envolvidos no teste do E1 *56*
4.6 e 4.7 - Imagens monocromáticas de um mesmo trecho de linha da pista no chão, com linha deformada na esquerda, e linha corrigida na direita *65*
A.1 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Comunicação com Beckhoff (S2) *72*
A.2 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Compensador PID (S3) *73*
A.3 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Correção de Velocidade (S4) *73*
A.4 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Cálculo de Erros (S5) *74*
A.5 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Cálculo de Erro de Entrada do Compensador (S6) *74*
A.6 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Fusão de Erros de Saída (S7) *75*
A.7 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Entrada Manual de Dados (S8) *75*
A.8 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Geração de Referência (S9) *76*
A.9 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Iniciar Variáveis Globais (S10) *76*
A.10 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Condição Inicial de Hardware (S11) *77*
A.11 - Diagrama de caminho de dados dos serviços envolvidos no teste de integração dos Serviços 5, 6 e 7 *77*
A.12 - Diagrama de caminho de dados dos serviços envolvidos no teste de integração dos Serviços 8, 9 e 10 *78*

A.13 - Diagrama de caminho de dados dos serviços envolvidos no teste do M2 *78*
A.14 - Diagrama de caminho de dados dos serviços envolvidos no teste do M3 *79*
A.15 - Diagrama de caminho de dados dos serviços envolvidos no teste de integração dos majoritários 1 e 3 *80*
Anexo A.1: Documento de requisitos do STU_S1, páginas 1 e 2 *82*
Anexo A.2: Documento de demanda de produção do STU_S1, páginas 1 e 2 *84*
Anexo A.3 - Diagrama de casos de uso do STU_S1 *86*
Anexo A.4 - Diagrama de atividades do STU_S1 *87*
Anexo A.5 - Diagrama de sequência do STU_S1 *88*
Anexo A.6 - Pseudocódigo do STU_S1 *89*
Anexo A.7 - Diagrama de classes do STU_S1 *92*
Anexo A.8 - Documento de requisitos do MTU_S1, páginas 1 e 2 *93*
Anexo A.9 - Documento de demanda de produção do MTU_S1, páginas 1 e 2 *95*
Anexo A.10 - Diagrama de casos de uso do MTU_S1 *97*
Anexo A.11 - Diagrama de atividades do MTU_S1 *98*
Anexo A.12 - Diagrama de sequência do MTU_S1 *99*
Anexo A.13 - Pseudocódigo do MTU_S1 *100*
Anexo A.14 - Diagrama de classe do MTU_S1 *101*
Anexo A.15 - Documento de execução de testes do S1, páginas 1 a 3 *102*

Lista de Abreviaturas e Siglas

AGV	Veículo autoguiado (do inglês: *Automated Guided Vehicle*)
AD01	AGV Diferencial 01
HIL	*Hardware-in-the-loop*
MTI	Majoritário de Teste de Integração
MTU	Majoritário de Teste Unitário
SOA	Arquitetura Orientada a Serviços (do inglês: *Service Oriented Architecture*)
STI	Serviço de Teste de Integração
STU	Serviço de Teste Unitário
UML	Linguagem de Modelagem Unificada (do inglês: *Unified Modelling Language*)

SUMÁRIO

INTRODUÇÃO **1**

1.1 Contextualização *1*

1.2 Motivação *3*

1.3 Justificativa *5*

1.4 Objetivos *7*

1.5 Delimitações e limitações do trabalho *8*

1.6 Método de pesquisa e desenvolvimento *8*

1.7 Organização do trabalho *9*

REVISÃO BIBLIOGRÁFICA **10**

2.1 Considerações iniciais *10*

2.2 Fundamentação teórica *10*

2.2.1 Arquitetura Orientada a Serviços *11*

2.2.2 Testes de software *12*

2.2.3 Protocolo de documentação do Tear *14*

2.3 Trabalhos correlatos *18*

2.3.1 Técnicas de teste *18*

2.3.2 Técnicas de SOA *20*

2.3.3 Técnicas de testes com SOA *20*

2.4 Considerações finais *21*

PROPOSTA DO TRABALHO **23**

3.1 Considerações iniciais *23*

3.2 Proposta *23*

3.3 Considerações finais *29*

VALIDAÇÃO **30**

4.1 Considerações iniciais *30*

4.2 Plano de experimento *31*

4.3 Testes *36*

4.3.1 Testes unitários de serviço *37*

4.3.1.1 Serviço de Intertravamento de Segurança (S1) *38*

4.3.1.2 Serviço de Comunicação com Beckhoff (S2) *40*

4.3.1.3 Serviço de Compensador PID (S3) *40*

4.3.1.4 Serviço de Correção de Velocidade (S4) *41*

4.3.1.5 Serviço de Cálculo de Erros (S5) *42*

4.3.1.6 Serviço de Cálculo do Erro de Entrada do Compensador (S6) *43*

4.3.1.7 Serviço de Fusão de Erros de Saída (S7) *43*

4.3.1.8 Serviço de Entrada Manual de Dados (S8) *44*
4.3.1.9 Serviço de Geração de Referência (S9) *45*
4.3.1.10 Serviço de Iniciar Variáveis Globais (S10) *45*
4.3.1.11 Serviço de Condição Inicial de Hardware (S11) *46*
4.3.2 Testes de integração de serviços *47*
4.3.2.1 Integração dos serviços S3 e S4 *47*
4.3.2.2 Integração dos serviços S5, S6 e S7 *49*
4.3.2.3 Integração dos serviços S8, S9 e S10 *50*
4.3.3 Testes unitários de majoritário *51*
4.3.3.1 Sistema de Intertravamento (M1) *51*
4.3.3.2 Sistema de Controle de Movimento (M2) *53*
4.3.3.3 Sistema de Atribuição de Dados (M3) *54*
4.3.4 Teste de integração dos majoritários M1 e M3 *55*
4.3.5 Teste do Software Escalonador (E1) *56*
4.3.6 Teste de sistema *58*
4.4 Resultados obtidos *60*
4.4.1 Resultados gerais *61*
4.4.2 Resultados específicos *61*
4.4.3 Resultados dos testes de integração de serviços *62*
4.4.4 Resultados dos testes de majoritários *63*
4.4.5 Resultados do teste de integração de majoritários *63*
4.4.6 Resultados do teste de sistema *64*
4.5 Análise dos resultados *65*
4.6 Considerações finais *67*
Conclusão **68**
5.1 Síntese do contexto da proposta e dos objetivos *68*
5.2 Contribuições e delimitações *68*
5.3 Trabalhos futuros *69*
Referências **70**
Diagramas de caminho de dados **73**
Documentação do teste do Serviço de Intertravamento de Segurança **82**

Capítulo 1

INTRODUÇÃO

1.1 Contextualização

A engenharia de testes de software, do ponto de vista geral, é um processo, ou uma série de processos, que são usados para certificar de que um código feito em computador executa a tarefa o qual foi designado a fazer, e que de certa forma, não faça ações inesperadas. Na área de desenvolvimento de software, é comum, por parte da sociedade, dizer que testes são planejados como um meio de certificar que uma função ou ferramenta está funcionando corretamente. Por outro lado, é possível, e mais assertivo, dizer que testes também são feitos com a intenção de encontrar erros. Compreender a verdadeira definição de testes de software podem fazer uma profunda diferença no sucesso dos esforços feitos em um software (MYERS, SANDLER, BADGETT, 2012).

Sabemos que, dentro de um teste de software, temos dois pontos de vista diferentes que contribuem para sua execução: as informações de entrada e de saída. Para que seja possível obter um resultado satisfatório em um teste, é necessário que as partes envolvidas neste processo tenham o conhecimento necessário para que, partindo de uma informação de entrada, seja possível saber como a saída deve se comportar neste cenário (MYERS, SANDLER, BADGETT, 2012). Esta situação mostra que a etapa de desenvolvimento de testes é importante não apenas ao sistema em questão, mas também aos desenvolvedores envolvidos. Isto acontece pois sem o domínio pleno do sistema a ser testado, o projeto corre

sério risco de cometer erros inesperados ou problemas desnecessários, adiando o tempo de desenvolvimento do sistema de forma indesejada para corrigir erros. Estes erros poderiam ser evitados caso os desenvolvedores envolvidos tivessem o conhecimento esperado para manusear o teste de maneira correta.

É importante que, em uma etapa de testes, fundamentos cruciais como o cronograma de execução, as entradas e saídas de teste, o ambiente físico de execução, a estrutura lógica do algoritmo de teste, entre outros fatores, sejam estabelecidos para que o esforço seja contínuo, progressivo e objetivo. Princípios importantes a serem considerados tanto em processo quanto em prática, considerando o parágrafo anterior, incluem: montar uma equipe eficiente, estabelecer mecanismos para boa comunicação e coordenação, foco em transferência de informações, e focar na qualidade em cada etapa (PRESSMAN, MAXIM, 2019). É interessante notar que não é possível montar testes apenas com conhecimentos consolidados, mas também com um fluxo de trabalho viável e efetivo entre as partes envolvidas em um projeto, ou seja, garantir que as partes envolvidas tenham compreensão do teste executado, e que não haja empecilhos, como um ambiente de teste desagradável, ou uma lógica de algoritmo de teste equivocado ou confuso (MYERS, SANDLER, BADGETT, 2012).

Estas definições também podem ser englobadas no desenvolvimento de sistemas embarcados, uma vez que esta área de estudo não somente contém a engenharia de software como um dos seus pilares principais, mas também como estudar e compreender o hardware a ser usado também faz parte da sua concepção (GANSSLE, 2008). Em um sistema embarcado, engenheiros de hardware e software devem trabalhar em conjunto até um ponto em que ambos consigam compreender um problema (BALL, 2002). Não é sempre claro que um problema em um sistema embarcado envolve apenas a parte do software ou a parte do hardware. Então, ambos os lados devem trabalhar juntos para duas ações que funcionam de maneira diferente em um trabalho que envolva somente hardware ou software: isolar um problema e consertá-lo. Com isso, é possível aplicar fundamentos que Pressman (2019) discute como forma de abordar estratégias em sistemas embarcados, e não nos limitarmos a estratégias de teste nos estudos de produção de software isolado.

De forma a alcançar uma forma efetiva de propor, estruturar e executar testes, tipos de arquiteturas de software foram elaborados, e que são frequentemente usados em diferentes projetos, de forma a também abranger projetos de hardware. O design destas arquiteturas são

representadas em um alto nível de abstração, o qual é um nível que pode ser diretamente traçado ao objetivo principal do sistema e a requisitos mais detalhados (PRESSMAN, MAXIM, 2019). Um exemplo deste estilo arquitetural é o modelo SOA (do inglês: *Service Oriented Architecture*), que divide diferentes processos dentro de um projeto em tarefas menores, indicando que cada um trabalha em conjunto de maneira independente, facilitando a busca por erros e a manutenção de um sistema (HAORONGBAM, NAGPAL, SEHGAL, 2022).

1.2 Motivação

Quando tratamos de testes em sistemas embarcados, a afirmação a validar neste trabalho é que alguns fatores, mesmo que poucos, podem ser cruciais para o bom funcionamento de um sistema. É possível citar, como exemplos, ao processo corrente de proibição de uso do veículo Cybertruck na Europa por diversos problemas encontrados envolvendo segurança para pedestres e motoristas, sendo o caso de análise mais recente de acordo com a submissão deste trabalho encontrado na Chéquia em outubro de 2024 (AVENOSO, 2024); e o incidente ocorrido com o lançamento da estação espacial CST-100 Starliner da empresa aérea Boeing, que devido a um mau funcionamento em um dos motores do sistema crítico, os astronautas tripulantes permaneceram fora do planeta Terra de junho de 2024 a março de 2025 (LEINFELDER, 2025).

Existem situações que podem causar problemas em fases de teste para sistemas embarcados. A exemplo, é possível citar a dificuldade para escolher uma arquitetura de desenvolvimento entre as opções já consolidadas no mercado, a falta de clareza por parte dos responsáveis na explicação do padrão ou design de testes, e uma quantia limitada de recursos para o projeto, uma vez que a complexidade de sistemas embarcados cresceu consideravelmente nas últimas décadas, e empresas de tecnologia não estão dispostas a pagar valores tão altos para a produção de seus produtos (SAINI, 2012). O ponto de vista apresentado por Saini (2012), junto aos exemplos dados no parágrafo anterior, podem validar o argumento de que temos problemas a serem enfrentados na área de teste para sistemas embarcados, sejam estes encontrados em empresas tecnológicas de pequeno ou grande porte.

Ao estudarmos livros de engenharia de software ou testes de software, a exemplo dos livros citados neste trabalho, é possível perceber que estes materiais estão focados em apresentar fundamentos de teste, e a importância de manter uma alta coesão e um baixo acoplamento de informações no desenvolvimento de software. Mas ao mesmo tempo, materiais didáticos não apresentam uma forma concreta de executar testes de software, de forma a ser um procedimento para que testes sejam executados. Esta situação nos apresenta a seguinte premissa: não existem boas práticas de teste de hardware disponíveis no mercado, ou seja, uma proposta concisa de um plano de testes que possa ser seguido a risca, para que testes possam ser executados sem a necessidade de um aprofundamento teórico maior da parte dos desenvolvedores de software. Sabendo destas circunstâncias, é possível trazer o caso de estudo deste trabalho, que representa a falta da prática discutida.

O laboratório Tear (Laboratório de Técnicas de Automação e Robótica) é um laboratório encontrado no campus São Carlos da Universidade Federal de São Carlos (UFSCar), o qual trabalha com soluções de automação industrial usando tecnologias e recursos acessíveis, seja para o campus ou aos estudantes de graduação que participam na equipe do laboratório. O Tear, atualmente, possui o mesmo problema anteriormente discutido: não existe um plano de testes concreto para validação eficiente aos seus projetos em desenvolvimento. Por esta ausência, alguns dos projetos, como um sistema de medição externa de precisão de veículos autoguiados (AGV, do inglês: A*utomated Guided Vehicle*), e o próprio AGV produzido pelo laboratório, tiveram seu tempo de entrega de produto constantemente estendido, pelo fato de que as estruturas de software foram terminadas, mas não houve uma etapa de validação desses softwares por meio de testes.

Para garantir uma melhor qualidade nos protótipos e produtos que são desenvolvidos no laboratório, é de extrema importância que os sistemas passem por uma rígida análise por meio de uma fase de testes, com o fim de não apenas encontrar erros em cada código envolvido em um dos projetos e consertá-los, mas também, com o progresso dos testes propostos, obter uma otimização no tempo de desenvolvimento, aumento no desempenho e uma possível perda de recursos que também pode ser evitada (MYERS, SANDLER, BADGETT, 2012).

1.3 Justificativa

Considerando a discussão apresentada na seção anterior, junto ao fato de que o trabalho de Saini foi publicado em 2012, e que a alta gravidade dos incidentes destacados ainda ocorrem em 2025, podemos afirmar que os problemas enfrentados na área de testes para sistemas embarcados persistem até os dias de hoje. Estes assuntos que envolvem engenharia de testes podem ser melhor estudados, com o propósito de alcançar produtos de melhor qualidade, uma maior confiabilidade com usuários/clientes, e que executem tarefas específicas com maior eficiência.

O laboratório Tear encontra o mesmo problema declarado: a ausência de uma etapa de testes clara e concisa está causando problemas como adiamento indefinido de entrega dos produtos, repetidas revisões de produto pelo crescente encontro de problemas em etapas mais próximas à implementação de um sistema completo, e a falta de documentação que contenha informações cruciais, e em alguns casos, triviais, para o bom andamento dos projetos do laboratório. Testes são processos de executar um programa com o intuito de encontrar erros (MYERS, SANDLER, BADGETT, 2012), e seguindo esta lógica, é válido afirmar que um plano de testes é uma solução para os problemas encontrados no Tear.

Como ponto de partida, é importante apontar que o plano de testes desejado ao Tear tem, como inspiração, ideias e conceitos que já foram criados no laboratório, os quais exploraram estas ideias e atividades em um momento anterior, como um projeto que envolvia a implantação da arquitetura SOA para modelagem de software dos projetos do laboratório (SOUZA, 2021), e uma proposta inicial de um plano de testes, o qual definiu estratégias de teste como a divisão entre testes unitários de serviços de software e testes de integração (SOUZA, 2023). Ambos trabalhos foram desenvolvidos com a ideia de aplicação no principal produto encontrado no Tear, o AGV Diferencial 01 (AD01). O AGV AD01, mencionado também na seção anterior, é um AGV seguidor de linha para transporte de materiais, que contém componentes com preço acessível, e apresenta estratégias de engenharia de controle com aplicações de inteligência artificial para seguir o percurso com alta precisão.

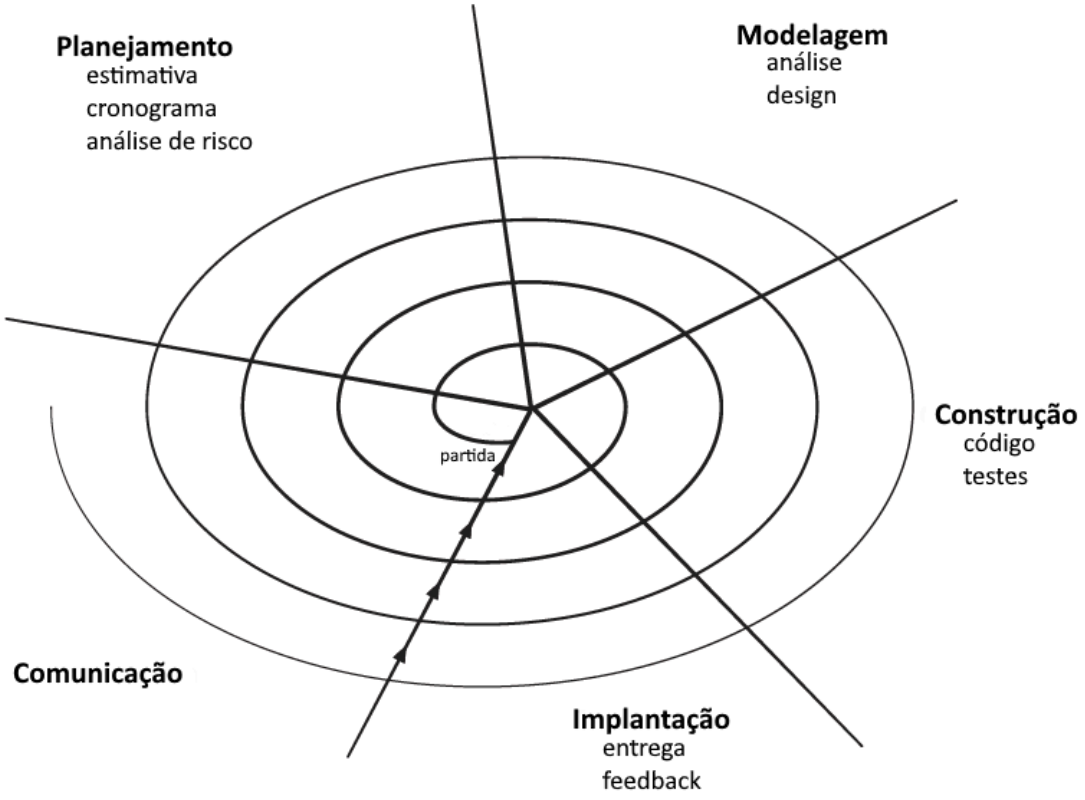
Portanto, este trabalho pretende reunir essas ideias e adaptá-las de forma a atender novas necessidades que acabaram surgindo no Tear com o passar do tempo, sendo estas uma

forma de validar os trabalhos mencionados no parágrafo anterior, e a consolidação de um plano de testes para todos os projetos atualmente em desenvolvimento no laboratório, como forma de também ser aplicado em projetos futuros. É também importante, para validar este trabalho como uma versão definitiva de um plano de testes, usufruir de ideias e planos já estabelecidos na área de engenharia de software. Entre os diversos conceitos do universo de engenharia de software, duas técnicas são abordadas como as principais ferramentas para alcançar a consolidação deste trabalho: o uso do modelo espiral de desenvolvimento, e o uso da técnica UML para modelagem de software.

O modelo espiral é um modelo de processo de desenvolvimento de software que foca em sua evolução (PRESSMAN, MAXIM, 2019), mas que a cada passo, sempre teremos uma revisão da sua documentação e prototipação já existente, para que o alinhamento entre as informações e a coerência entre documentos continue a par do projeto. Como ilustrado na Figura 1.1, a ideia principal é que, em um projeto, todo o seu conteúdo seja frequentemente revisado em diferentes etapas, para que sua funcionalidade se mantenha clara a todo momento.

A linguagem de modelagem unificada (UML, do inglês: *Unified Modelling Language*) é uma linguagem visual feita para modelar softwares, e por ser uma linguagem de modelagem, a UML é capaz de auxiliar a definição das características de um sistema, como seus requisitos, seu comportamento, sua estrutura lógica, a dinâmica de seus processos e também as necessidades físicas em relação ao equipamento sobre o qual o sistema deverá ser implantado (GUEDES, 2018). Usando a UML, é possível ilustrar o funcionamento de um projeto de maneira mais clara, o que o plano de testes apresentado neste trabalho deseja alcançar. Assim, seguindo ideias já discutidas no laboratório Tear, e seguindo as estratégias descritas nesta seção, é possível identificar um caminho promissor para a resolução do problema que o laboratório enfrenta atualmente.

1.1 - Modelo espiral simples (adaptado de Pressman e Maxim, 2019)



1.4 Objetivos

Este trabalho possui, como objetivo principal, a apresentação e detalhamento de um plano de testes para aplicação em sistemas embarcados, como forma de solucionar os problemas descritos na seção anterior. É esperado que este plano de testes possa ser replicado para qualquer sistema embarcado não-crítico encontrado no mercado, e o escopo dos produtos em desenvolvimento pelo Tear será usado como forma de reforçar esta hipótese.

Além disso, o plano de testes será apresentado como uma forma de otimizar o tempo de desenvolvimento de um projeto de sistema embarcado, uma vez que os testes são feitos de forma a tornar ágil o encontro de erros no sistema para correção. É válido apontar que este trabalho deseja fazer contribuições de forma incremental, por meio do plano de testes ser um projeto sistematizado e organizado.

No decorrer da apresentação do plano de testes, este trabalho também explica as etapas de desenvolvimento enfatizando ferramentas essenciais, como a arquitetura de software SOA (do inglês, *Service Oriented Architecture*) para planejamento de projetos, o uso da UML para construção de diagramas para visualização de um projeto, a construção de tipos de testes, como testes unitários e de integração para englobar um ou mais serviços de software; todas as ferramentas escolhidas com o fim de criar um paradigma eficiente e compreensível para o desenvolvimento de testes. É interessante apontar que o trabalho também usa conceitos de *hardware-in-the-loop*, que ou seja, a execução de casos de teste contínuos com cuidados robustos para segurança do sistema, para fazer partes do desenvolvimento de testes, uma vez que as técnicas de teste apresentadas neste trabalho também serão usadas em hardwares em ambiente controlado.

1.5 Delimitações e limitações do trabalho

As delimitações encontradas para este trabalho são listadas abaixo:

- O foco deste trabalho não é apresentar um novo paradigma de desenvolvimento de testes, mas aproveitar conhecimentos já consolidados para adaptação de uma solução para um problema encontrado;
- Códigos fonte, materiais, especificações de produtos, modelos de arquitetura, e estrutura geral de projetos encontrados no laboratório Tear não serão apresentados.

A limitação encontrada neste trabalho é que o plano de testes não foi planejado para funcionamento em sistemas críticos, uma vez que sua estrutura abre margem para múltiplas execuções de um sistema embarcado para encontro de erros e problemas para correção; este fator não encaixa no conceito de sistemas embarcados críticos, uma vez que ocorrem com execução altamente controlada, e erros neste tipo de sistema podem causar perda de materiais de alto custo ou de vidas humanas.

1.6 Método de pesquisa e desenvolvimento

O propósito deste trabalho é de continuar uma pesquisa qualitativa que foi iniciada pelo laboratório Tear em tempos anteriores. As ideias estabelecidas por este trabalho foram discutidas em estudos passados do laboratório, e uma das pendências mais importantes observadas foi a aplicação real dos testes nos projetos em desenvolvimento. Alguns conceitos que serão observados nos capítulos seguintes já foram propostos anteriormente aos membros do Tear, mas este trabalho foca na consolidação destes conceitos, junto à discussão de assuntos que não foram abordados ou criticados de uma maneira mais analítica.

De modo a consolidar conceitos, este trabalho segue um método de pesquisa experimental, com uma análise observacional em segundo plano. Uma vez que planos de ação já foram propostos ao laboratório, este trabalho será responsável por demonstrar que, por meio de experimentos cumpridos usando um plano de testes robusto e organizado, o tempo de desenvolvimento de um projeto do laboratório pode ser otimizado, e problemas podem ser detectados e corrigidos prematuramente, promovendo uma maior qualidade e segurança no produto final.

1.7 Organização do trabalho

Esta dissertação começará com uma discussão da revisão bibliográfica no capítulo seguinte, onde conceitos essenciais para maior compreensão do trabalho serão abordados na seção de fundamentos teóricos; e a seguir, a seção de trabalhos correlatos discutirá sobre trabalhos que apresentam ideias que reforçam as ideias que este trabalho deseja apresentar. O capítulo de proposta, enfim, será onde o plano de testes será detalhado com o maior aprofundamento possível, mostrando as etapas de teste, os requisitos, procedimentos, e sequências de tarefas que devem ser cumpridas para alcançar o desempenho esperado pelo laboratório Tear.

O capítulo de validação introduz o plano de testes aplicado no AGV AD01, e inclui todos os procedimentos apresentados no capítulo da proposta, ou seja, a ordem dos testes,

suas estruturas e a discussão dos resultados alcançados. Em seguida, o capítulo de conclusão busca contextualizar a proposta com os objetivos destacados neste capítulo, apresentando também as contribuições e delimitações encontradas, também destacando os trabalhos que podem ser planejados em um tempo futuro, de forma que o laboratório Tear possa aproveitar os frutos deste trabalho ao máximo.

Capítulo 2

REVISÃO BIBLIOGRÁFICA

2.1 Considerações iniciais

Neste capítulo, vamos definir a fundamentação teórica necessária para melhor compreensão deste trabalho. A seção seguinte está dividida em três partes distintas: a apresentação da arquitetura de software que será aprofundada nos capítulos seguintes, os conceitos básicos sobre engenharia de testes, e o protocolo de documentação que o plano de testes adotará para sua concepção. Em seguida, a seção de trabalhos correlatos destaca uma discussão sobre este trabalho em comparação com outros já existentes. Esta abordagem será feita seguindo três assuntos semelhantes, de forma a observar o propósito de existência deste trabalho de diferentes pontos de vista.

2.2 Fundamentação teórica

Nesta seção, três assuntos que são pertinentes para a melhor compreensão do conteúdo apresentado neste trabalho serão apresentados. A subseção abaixo trata sobre a arquitetura SOA, a base fundamental para o desenvolvimento dos softwares nos projetos em que este trabalho está envolvido. A subseção seguinte detalha os tipos de testes de software que são usados nos projetos do laboratório Tear, e que também serão usados neste trabalho. A última

subseção apresenta o protocolo de documentação do laboratório, ou seja, a ordem hierárquica de documentos a serem criados para o desenvolvimento de um software, e as ferramentas usadas para a criação.

2.2.1 Arquitetura Orientada a Serviços

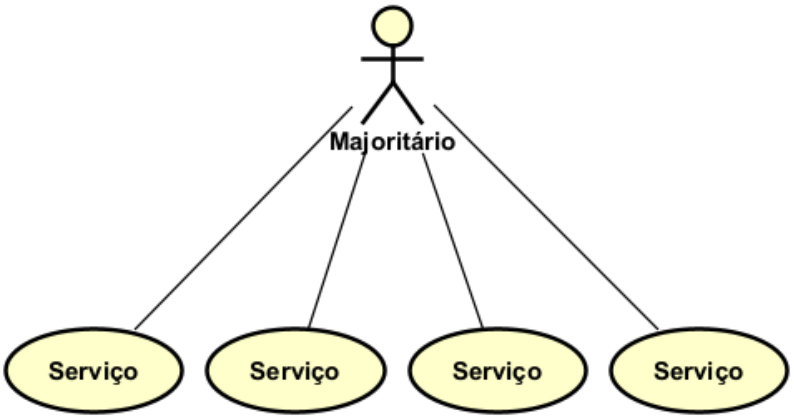
Arquitetura orientada a serviços (do inglês: *service oriented architecture*, ou SOA) é um tópico delicado de ser definido pela sua abordagem extensa. Segundo Josuttis (2007), SOA não é uma arquitetura concreta, mas sim um paradigma que leva a uma tomada de decisões concretas dentro de um projeto. Thomas Erl (2008) reforça essa ideia ao dizer que computação orientada a serviços em geral usa diversas ideias previamente trazidas por conceitos de computação distribuída, e o conceito de SOA é usando de maneira tão abrangente que quase se tornou sinônimo com o próprio conceito de computação orientada a serviços. Neste trabalho, SOA será interpretado como um modelo de arquitetura, uma vez que, como mencionado no capítulo anterior, o objetivo deste trabalho é apresentar um plano de testes para sistemas que foram implementados com conceitos oriundos da arquitetura orientada a serviços.

Por esta interpretação, SOA é um modelo de arquitetura que aumenta a eficiência, agilidade e produtividade de uma empresa ou equipe (ERL, 2008), posicionando diferentes softwares como “serviços”. Estes serviços devem representar uma funcionalidade que corresponde a uma atividade que deve ser executada no mundo real (JOSUTTIS, 2007). Neste trabalho, serviços são softwares provedores, pois oferecem uma tarefa executada para que outro sistema possa chamá-los. Estes sistemas, que chamam um ou mais serviços, são softwares consumidores (JOSUTTIS, 2007), os quais, neste trabalho, são chamados de componentes majoritários. O escalonamento de tarefas nos projetos que envolvem este trabalho, portanto, são feitos na relação entre serviços e majoritários, e também depende da disposição entre qual majoritário chama um determinado serviço, e qual ordem de serviços deve ser feita por um majoritário para que uma rotina de tarefas seja feita da maneira esperada pela equipe ou empresa.

Vantagens para usar SOA incluem (HAORONGBAM, NAGPAL, SEHGAL, 2022):

- Alta eficácia em atender necessidades comerciais emergentes com mais rapidez dado à divisão da arquitetura em serviços que podem ser descartáveis ou reconsiderados;
- Alta escalabilidade de funções, pois serviços podem ser adaptados ou diretamente usados em diferentes aplicações fora do escopo de um projeto específico;
- Alta manutenibilidade em aplicações, dada à independência de serviços facilitar a identificação de um problema de modo mais ativo;
- A individualidade dos serviços envolvidos permite que eles sejam testados de maneira mais efetiva, de forma que sua função poderá ser testada sem a necessidade de acoplamento do sistema completo.

2.1 - Ilustração de um modelo de arquitetura baseado em SOA usado no laboratório Tear



2.2.2 Testes de software

Ao planejar testes, existem conceitos que auxiliam não somente na compreensão das diferentes etapas do teste, mas também em seu contexto. Uma vez que nem sempre a pessoa responsável pelos testes estará familiarizada com a tecnologia por trás da aplicação a ser testada, é importante fazer estas separações com o intuito de otimizar o tempo de desenvolvimento e da busca de resultados, para que uma melhor análise do objeto de teste seja feita mais rapidamente. Tipos de teste podem ser definidos como:

- **Teste estático:** a avaliação da aplicação que não requer o sistema ou software em funcionamento (HOMÈS, 2024). Este tipo de teste é usado para observar o andamento da modelagem da aplicação, ou seja, análises com ou sem o uso de ferramentas. Documentos, diagramas, planos de testes e guias de usuários são exemplos de ferramentas analisadas para este tipo de teste;
- **Teste dinâmico:** a avaliação da aplicação é feita com a implementação real, seja em software ou em hardware (HOMÈS, 2024). Neste tipo de teste, os resultados obtidos são comparados com resultados esperados, e com isso, é feita uma análise mais específica, seja em desempenho, usabilidade, capacidade de manutenção ou confiabilidade. Existem dois tipos específicos de teste dinâmico, sendo eles:
    - **Teste caixa preta:** um tipo de teste dinâmico o qual o conteúdo da implementação não é conhecido na aplicação. Este tipo de teste considera, portanto, apenas as entradas e as saídas do sistema como forma de validação do teste (JORGENSEN, DEVRIES, 2021);
    - **Teste caixa branca:** também chamado de teste caixa transparente, este é um tipo de teste o qual os detalhes internos, ou seja, a estrutura de código do sistema a ser testado, são observados na execução (JORGENSEN, DEVRIES, 2021). Este tipo de teste requer maior conhecimento da estrutura de código do sistema para que os problemas sejam mais facilmente encontrados e resolvidos.

Além dos tipos de teste, existem também os níveis de teste, que são os modos os quais os tipos de teste descritos acima podem ser aplicados em uma determinada situação. Quando pensamos em um projeto com diferentes ferramentas, é necessário testar cada uma para checar seu funcionamento correto, e em seguida, observar o comportamento de duas ou mais ferramentas conjuntas, até que, então, seja possível observar o comportamento do sistema final como um todo. Este tipo de estratégia recebe o nome de *bottom-up*, pois a análise de teste inicia pelos componentes mais atômicos possíveis, em uma perspectiva micro do projeto, e segue por diferentes etapas em diferentes complexidades, subindo o nível de abstração do projeto, até que o sistema final seja alcançado, na perspectiva macro. Para trabalhar nesta estratégia de testes, vamos seguir as ideias apresentadas abaixo:

- **Teste unitário:** uma avaliação de comportamento de componentes individuais do sistema, podendo ser subrotinas, subprogramas, classes ou procedimentos específicos (MYERS, SANDLER, BADGETT, 2012). Este teste é necessário para checar se o objeto testado funciona de acordo com o esperado antes de avançar para próximas etapas nas quais o componente está envolvido. No caso de um projeto de sistema embarcado (a exemplo do AD01), componentes de hardware como motores e sensores, e serviços de software como correção de velocidade das rodas são exemplos de testes unitários cabíveis;
- **Teste de integração:** uma avaliação analisando o comportamento de dois ou mais componentes individuais de um sistema conectados a um software consumidor. Este teste é feito para observar funcionalidades mais complexas que requerem precisão e eficiência no sistema final. Existe mais de uma forma de organizar uma integração entre serviços (PRESSMAN, MAXIM, 2019), mas este trabalho será focado na estratégia *bottom-up*, como mencionado no parágrafo anterior;
- **Teste de sistema:** todos os componentes de um sistema são testados em conjunto para fins de observação de comportamento do sistema mínimo viável (JORGENSEN, DEVRIES, 2021). Análises de interação do sistema com ambiente, usuário ou outros objetos envolvidos são todos considerados para reconhecimento de possíveis erros de performance ou de manutenção.

2.2.3 Protocolo de documentação do Tear

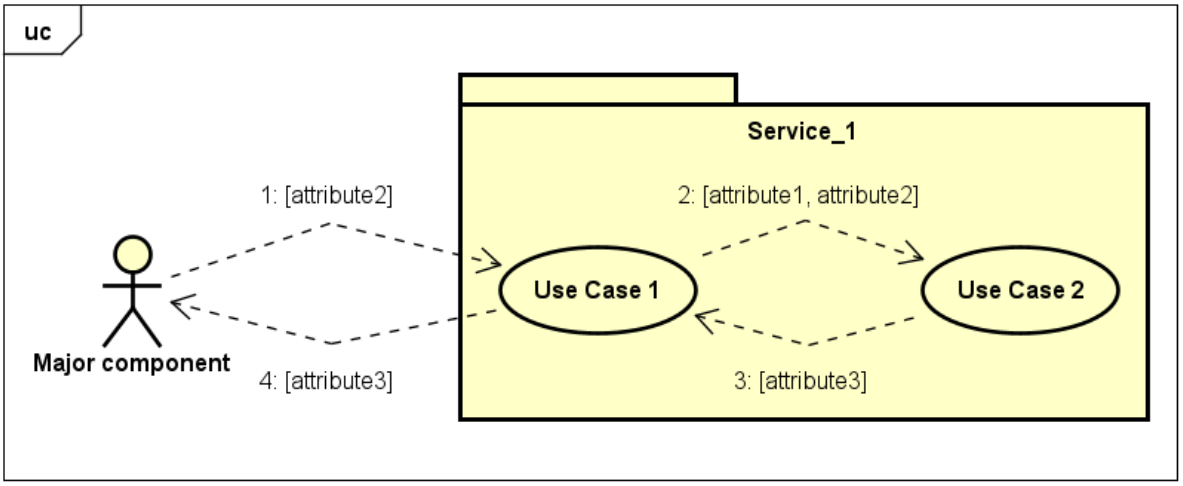
Seguindo a filosofia observada em SOA, existe, no Tear, uma ordem de documentos que devem ser gerados para um melhor detalhamento de cada serviço existente em uma aplicação. Estes documentos seguem uma lógica *top-down*, ou seja, indica sua funcionalidade em uma lógica geral enquanto aprofunda em detalhamentos ao passo de que os documentos seguintes são construídos. É importante lembrar que, nestes documentos, a construção de diagramas usa noções de UML com o objetivo de fornecer múltiplas visões do sistema a ser modelado, a fim de atingir a completude da modelagem, permitindo que cada diagrama complemente os outros (GUEDES, 2018). As Figuras 2.2, 2.3, 2.4 e 2.5 apresentam os diagramas que são usados nos projetos do laboratório Tear e ressaltam a conexão entre eles.

A ordem de documentos discutida no início desta subseção, de forma a apresentar o comportamento de um serviço de uma lógica macro ao micro, é encontrada a seguir:

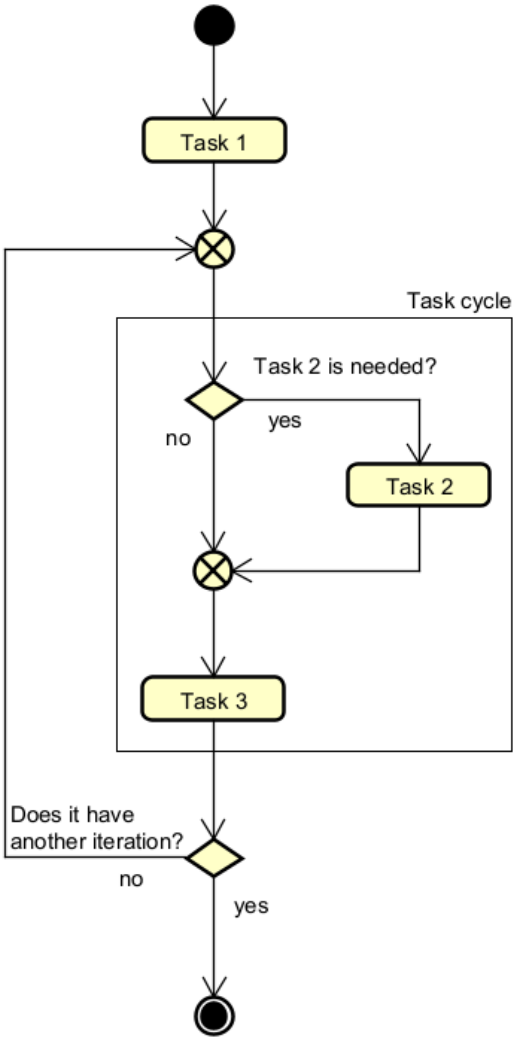
- **Documento de requisitos:** um documento que descreve detalhadamente os requisitos, funcionais e não funcionais, do serviço, também discutindo seu escopo, restrições e dependências, de forma a fornecer uma base clara e compreensível do serviço aos desenvolvedores;
- **Documento de demanda de produção:** um documento com o fim de detalhar o método que as funcionalidades discutidas no documento anterior devem ser implementadas no serviço pelos desenvolvedores, junto com as entradas e saídas esperadas para os casos de teste que se desejam ser executadas;
- **Diagrama de casos de uso:** uma versão adaptada do conceito original o qual descreve os casos de uso do serviço, mostrando as condições em que o serviço deve ser executado junto às interações necessárias para chamada, e ao fluxo de informações atribuído aos parâmetros necessários ao serviço. A Figura 2.2 apresenta um exemplo deste diagrama, com o consumidor representado na figura de um boneco, enquanto o serviço, representado como uma pasta, contém os casos de uso apresentados no documento de demanda de produção, com as setas representando a ordem de ações junto aos parâmetros envolvidos;
- **Diagrama de atividades:** descreve o fluxo de atividades e a lógica de controle de algoritmo, focando na ordem de atividades que devem ser feitas dentro do serviço no momento de interação com o sistema (GUEDES, 2018). A Figura 2.3 apresenta um exemplo de diagrama de atividades, onde os retângulos apresentam a atividade, os losangos indicam uma bifurcação a qual uma atividade será feita dependendo de uma checagem anterior, enquanto o círculo cruzado indica o término de uma bifurcação;
- **Diagrama de sequência:** descreve a sequência temporal de interação entre o serviço e o sistema, detalhando o momento em que o serviço é chamado e o momento em que o serviço envia o resultado da sua execução para o sistema (GUEDES, 2018). A Figura 2.4 apresenta um exemplo de diagrama de sequência, seguindo o exemplo apresentado na Figura 2.3, ou seja, as tarefas apresentadas em 2.3 são as mesmas encontradas em 2.4, mas desta vez, indica a sequência das atividades com os parâmetros envolvidos para a execução delas;

- **Pseudocódigo:** apresenta um esboço do código final, indicando sua estrutura básica para construção na linguagem de programação escolhida para o sistema e, consequentemente, ao serviço;
- **Diagrama de classes:** representa a estrutura estática de uma classe de orientação a objetos, indicando os atributos, funções e métodos necessários para a construção do código final do serviço (GUEDES, 2018). A Figura 2.5 mostra um exemplo de diagrama de classes, que apresenta as classes envolvidas em um determinado serviço, junto à classe de fachada conectada à classe principal, que é uma estrutura usada nos códigos do laboratório Tear para simplificar o uso e chamada da classe principal em maior simplicidade e clareza do módulo, fazendo que o usuário não precise conhecer os detalhes da implementação, e gerando uma maior facilidade na manutenção, em uso para teste, e na segurança de acesso ao código (RODRIGUES, 2025);
- **Documento de execução de testes:** um documento que apresenta os resultados finais adquiridos na execução do teste, contendo as adequações que foram necessárias para a execução do teste, os resultados alcançados para cada parte envolvida do teste, uma análise dos resultados obtidos, e uma conclusão para indicar se o teste foi aprovado ou não.

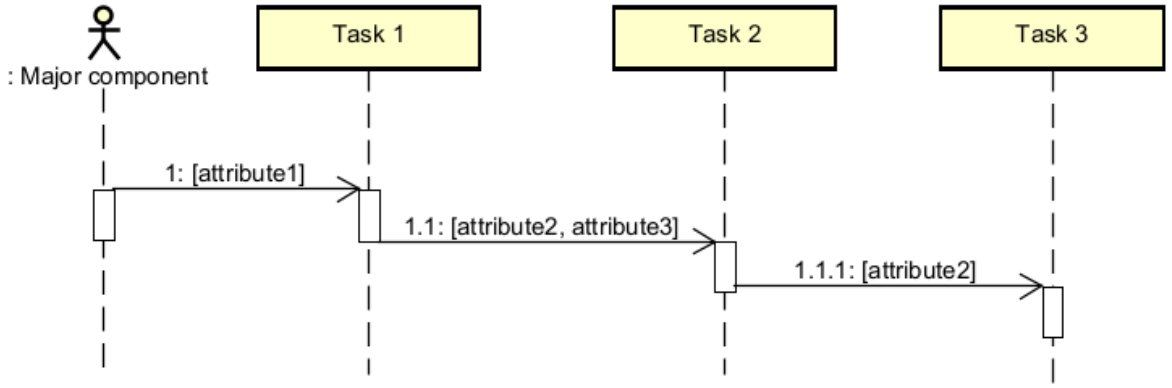
2.2 - Exemplo de diagrama de casos de uso



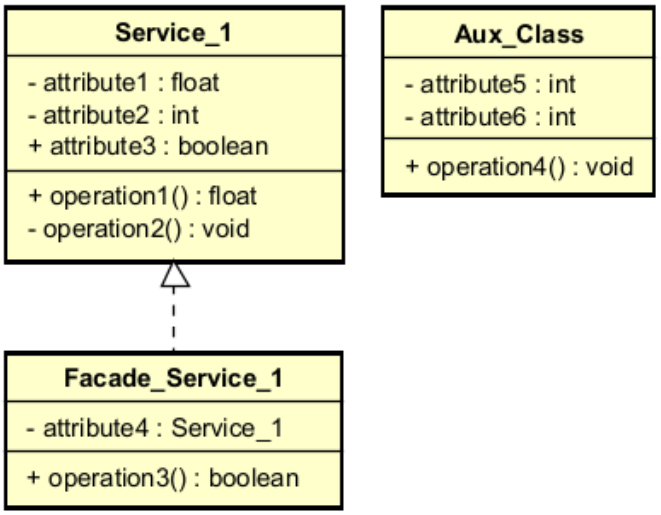
2.3 - Exemplo de diagrama de atividades



2.4 - Exemplo de diagrama de sequência



2.5 - Exemplo de diagrama de classes



2.3 Trabalhos correlatos

Nesta seção, faremos um estudo sobre diferentes trabalhos que possuem relação com a área de estudo deste trabalho. Existem três temas principais que complementam o escopo apresentado, e que podem mostrar um diferente ponto de vista sobre o assunto, sendo estes trabalhos que discutem técnicas de teste, trabalhos que usam técnicas de SOA para alcançar os resultados esperados, e trabalhos que aproveitam de técnicas de SOA como estratégia de testes. Os temas estão divididos em diferentes seções, acompanhando uma breve discussão sobre os trabalhos envolvidos.

2.3.1 Técnicas de teste

O trabalho de Ignarra et al. (2025) apresenta um fluxo de trabalho estruturado para teste de subsistemas embarcados de segurança crítica usando um ambiente de *hardware-in-the-loop* (HIL). Este fluxo de trabalho é definido por três fases principais: especificação de teste, que consiste em definir todo o conteúdo mais detalhado dos testes a serem executados, como o comportamento da funcionalidade a ser testada, variações de sinal, cenários de simulação e scripts de teste; execução de teste, que é a execução adequada de cada sequência automatizada no alvo real a ser testado por meio dos cenários de teste de

hardware-in-the-loop; e avaliação de resultados, que consiste na coleta de todas as informações necessárias para avaliar os resultados obtidos. Neste trabalho, o método HIL foi escolhido porque, segundo Ignarra et al., ao utilizar um ambiente de *hardware-in-the-loop*, sistemas embarcados de segurança crítica podem ser testados adequadamente no início do processo de validação para atender a rigorosos padrões de segurança funcional, graças à natureza do ambiente HIL, que promove a execução contínua de casos de teste, o tratamento robusto de erros e o monitoramento consistente do comportamento do sistema.

O trabalho de Krishnan, Manju e Jayalekshmy (2023) apresenta técnicas de teste em um computador de bordo de um veículo de lançamento, o qual possui um software chamado *Real Time Executive*. Este software é responsável pelo gerenciamento de threads, entradas e saídas, sincronização e comunicação de processos internos, e agendamento de tarefas. É interessante observar que a lógica usada no trabalho de Krishnan, Manju e Jayalekshmy é direcionada a um veículo de lançamento (considerado um sistema crítico), então é importante que todos os testes sejam feitos com muita cautela para evitar possíveis falhas e, consequentemente, uma perda de recursos de altíssimo valor. Diferente do trabalho discutido, este trabalho tem o foco em sistemas que não são considerados críticos, então é importante ressaltar que o plano de testes apresentado ainda possui uma maleabilidade maior na busca de erros para correção.

Tomimori, Oyama e Azumi (2023) discutem sobre um *framework* de método de teste de integração contínua (CI) para utilização em componentes embarcados. Para testar as execuções incluídas neste *framework*, o sistema TECS (da abreviação em inglês: *TOPPERS Embedded Component Systems*) foi usado para montar o seguinte projeto: um sistema de divisão de componentes para construção de sistemas embarcados. Para isso, duas ferramentas do sistema TECS foram usadas: TECS CDL, que é uma ferramenta para trabalho em Linguagem de Descrição de Componentes (do inglês: *Component Description Language*, ou CDL); e TECS Unit, que é um framework para teste unitário de células criadas em TECS. Com estas duas ferramentas, houve a montagem de diferentes estruturas de *framework* para teste em diferentes componentes embarcados. Este trabalho mostra uma ferramenta diferente para execução de testes, deixando alguns métodos convencionais de lado para aprofundar em métodos utilizando softwares já consolidados no mercado.

2.3.2 Técnicas de SOA

Iseni e Halili (2022) propõem um projeto baseado em SOA para desenvolvimento de um sistema de comunicação para um veículo espacial que contém componentes de serviços em rede para comunicação. Este projeto usa o protocolo HTTP e um ESB (do inglês: Enterprise Service Bus) como sistema de comunicação entre o veículo e os serviços de rede. O projeto possui três níveis de comunicação, sendo eles o cliente, que é o próprio veículo com um computador de bordo para se comunicar com o ESB e executar funções; o intermediário, que é o ESB responsável por escalonar as informações entre o veículo e os serviços de rede; e o servidor, acoplado em estações espaciais que trabalham com dados para comunicação com o ESB. É possível observar que o modelo SOA implementada neste estudo como forma de tratar o veículo e as informações dos serviços em rede como serviços independentes, enquanto o ESB é tratado como o componente majoritário. Portanto, pode-se dizer que o estudo respeita o modelo SOA da maneira correta.

Zhang e An (2021) apresentam um modelo SOA adaptado para controle de robôs. A estrutura proposta pelo trabalho é dividida em um framework composto de componentes de software, que são divididos em componentes individuais ou compostos; e um middleware que funciona como o componente escalonador das tarefas que devem ser executadas pelos softwares. A estrutura dos componentes oculta seus detalhes internos de implementação e fornece interfaces uniformes, então a comunicação entre os componentes pode ser realizada pelo middleware. É interessante apontar que, assim como o trabalho de Iseni e Halili (2022), Zhang e An alcançam um padrão de SOA que respeita os padrões da arquitetura, sendo o middleware visto como o componente consumidor da estrutura, enquanto os softwares se comportam como os serviços que devem ser executados por qualquer robô que receba este modelo de arquitetura.

2.3.3 Técnicas de testes com SOA

Aziz, Ullah e Rashid (2021) propõem um modelo chamado SOPES (do inglês, *Service-Oriented Process model for developing Embedded Software systems*), que é um guia descritivo para desenvolvimento de sistemas de software embarcados que utilizam softwares como serviços. O SOPES se divide em duas etapas: a etapa de design é responsável por

decidir qual o grupo de serviços que deve ser construído, os dispositivos embarcados envolvidos e as suas interações com o sistema final; e a etapa de projeto é responsável por desenvolver a arquitetura de software do sistema, usando os serviços identificados pela etapa de design. Em comparação com a utilização de conceitos orientados a serviços de maneira isolada neste domínio, a utilização do modelo SOPES produziria sistemas de software de alta qualidade, se aproveitando de fundamentos encontrados em SOA como base para seu funcionamento.

O trabalho de Zhang *et al.* (2021) apresenta a construção de um algoritmo genético cuja tarefa é gerar sequências de teste de integração baseado em SOA para serviços de software. Este algoritmo possui duas etapas principais para a geração da sequência de testes. A primeira etapa analisa os requisitos e especificações de cada nível do sistema, junto aos outros documentos de software relacionados, com o propósito de calcular o valor de importância inicial do serviço analisado. A segunda etapa é feita após a análise dos serviços incluídos, e após coletar as informações de importância inicial de cada um, o algoritmo gera uma sequência final de teste de integração com base na prioridade de teste dos serviços. O esquema de testes é apresentado pelo algoritmo em uma representação semelhante ao que este trabalho deseja alcançar, ou seja, aproveitando técnicas de montagem de diagramas de SOA para montar um diagrama final de testes de integração.

2.4 Considerações finais

Este capítulo apresentou os fundamentos teóricos necessários para uma maior compreensão deste trabalho, junto à discussão de trabalhos que procuram objetivos semelhantes a este. É importante lembrar que as referências discutidas complementam informações pertinentes nos próximos capítulos, assim como estaremos retomando diferentes conceitos mostrados nos fundamentos teóricos pelo decorrer da discussão da proposta do trabalho, no capítulo seguinte. Outro ponto importante que deve ser levantado é que todos os trabalhos correlatos apresentam técnicas próprias para desenvolvimento de software e testes, reforçando a problemática deste trabalho de que não houve o uso de um plano concreto de

testes, e que apenas fundamentos básicos foram usados como inspiração para as teses apresentadas nesses trabalhos.

Capítulo 3

PROPOSTA DO TRABALHO

3.1 Considerações iniciais

Este capítulo contém uma seção única, com foco na apresentação da proposta do trabalho. Na seção seguinte, vamos detalhar como os conceitos de SOA serão aproveitados, junto com a abordagem mestra dos testes que forem montados, e como ela se relaciona com os tópicos de motivação, justificativa e objetivos, apresentados no Capítulo 1 deste trabalho.

3.2 Proposta

Este trabalho é dedicado a apresentar um plano de testes de software de forma que as partes envolvidas no plano consigam compreender seu conteúdo. O plano de testes apresentado tem o propósito de ser usado não somente em projetos de software mas como também em projetos de hardware, devido ao seu escopo envolver estratégias de teste que se encaixam em ambos os espectros da Engenharia da Computação. É esperado que o plano de testes tenha uma contribuição incremental, de sistematização, organização e fácil aplicação, de forma a otimizar o tempo de execução dos testes de forma considerável, como apresentado na seção de objetivos deste trabalho.

Uma característica mínima para uso do plano de testes apresentado neste trabalho é que o sistema que deseja ser testado deve ter um software embarcado, uma vez que o plano usa SOA, uma arquitetura primordialmente usada para criação de softwares.

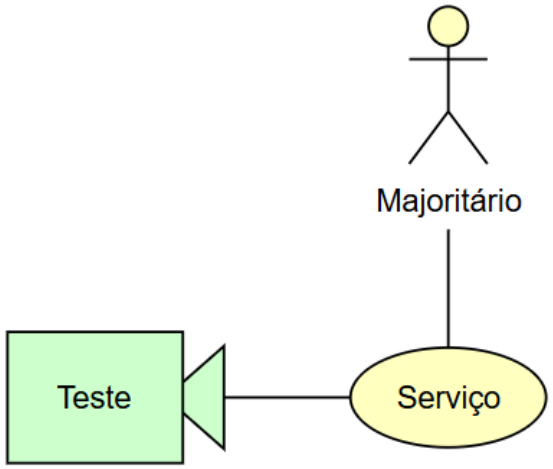
Levando em conta a estrutura deste plano, é possível afirmar que ele possui um caráter semi automático, pelo fato de que a forma de verificação de dados e construção são feitas de forma manual, enquanto a forma de validação de entradas e saídas de cada teste feito passa por um processo automatizado, uma vez que cabe à pessoa desenvolvedora envolvida criar a lógica de desenvolvimento do teste, e cabe ao software criado indicar os valores e índices esperados do teste. É interessante apontar que existem outras formas de execução de testes, como o método test-driven, mas este trabalho foca em propor uma ideia própria de execução de testes.

O plano de testes apresentado neste trabalho possui uma estrutura baseada em SOA, pelo motivo de que ele é voltado a projetos desenvolvidos com a mesma arquitetura como base. Porém, é importante destacar que esta atribuição possui um limiar que impede o plano ser completamente identificado como projeto de SOA, e este limiar é a inclusão dos testes em uma arquitetura que representa apenas duas entidades, seguindo a lógica dada por Josuttis (2007): provedor (serviço) e consumidor (majoritário). Ao incluir testes na arquitetura, o fundamento principal dela acaba se rompendo, e uma abordagem diferente se mostra necessária. Por isso, podemos afirmar que o plano de testes é, de fato, um projeto que não se apresenta como desenvolvido via SOA, mas possui regras e fundamentos que são respeitados como elementos desse modelo. Outra estrutura que este trabalho possui é a técnica de *hardware-in-the-loop* (HIL), uma vez que estamos contando com um modelo de testes que envolve não apenas cuidados mais estudados, mas também uma estrutura contínua de testes, e o monitoramento constante do hardware na execução, como apresenta o trabalho de Ignarra (2025).

Em outras palavras, este trabalho propõe um plano de testes inspirado em SOA, em que a diferença principal é a inserção dos testes na arquitetura usando alguns conceitos de HIL. Para ilustrar, é possível usar a nomenclatura comum em UML. Temos os “atores” que chamam ou comandam ações (consumidores ou majoritários), e as tarefas, tempos de execução ou casos de uso (GUEDES, 2018). Com a ideia de atores e tarefas, é interessante, e de certa forma, um modo descontraído de rotular, considerar os testes como as “câmeras” do

projeto em questão, que relatam se os serviços e softwares gerados se comportam como esperado. A Figura 3.1 apresenta este conceito de forma ilustrada.

3.1 - Ilustração do teste ao traduzir o diagrama de um projeto em SOA para UML



É por conta do limiar mencionado no parágrafo anterior que é possível aplicar o plano proposto em sistemas embarcados no geral: ao encararmos os serviços e entidades majoritárias como partes independentes, uma estratégia oriunda de SOA, e inserirmos serviços de teste (testadores) em sua estrutura, é possível que o plano de testes seja aplicado em diferentes projetos, não se limitando apenas à engenharia de software, mas também englobando engenharia de sistemas de hardware. Isso se dá por esta arquitetura ser capaz mostrar um caráter escalável, interoperável, reutilizável, e de alta manutenção, assim como poder se adaptar a eventos não previstos e de lidar com certos impedimentos que podem aparecer durante a operação principal (ISENI, HALILI, 2022).

Para que a aplicação de SOA em testes gerais seja melhor compreendida, é necessário analisarmos alguns dos conceitos discutidos na Seção 2.2.2 do capítulo anterior, o qual os tipos de teste foram contextualizados. Deve-se esclarecer que esses procedimentos devem seguir um protocolo, com o fim de garantir um fluxo de trabalho eficaz. Sabendo disso, a estrutura da abordagem de testes será apresentada, indicando como cada tipo de teste deve ser tratado nesta estratégia proposta (SOUZA, 2023):

- **Testes unitários:** elaborados pelos desenvolvedores envolvidos em determinado serviço, com aprovação do supervisor de testes. Cada um dos testes unitários devem ser montados individualmente, reforçando o paralelismo de cada, com o

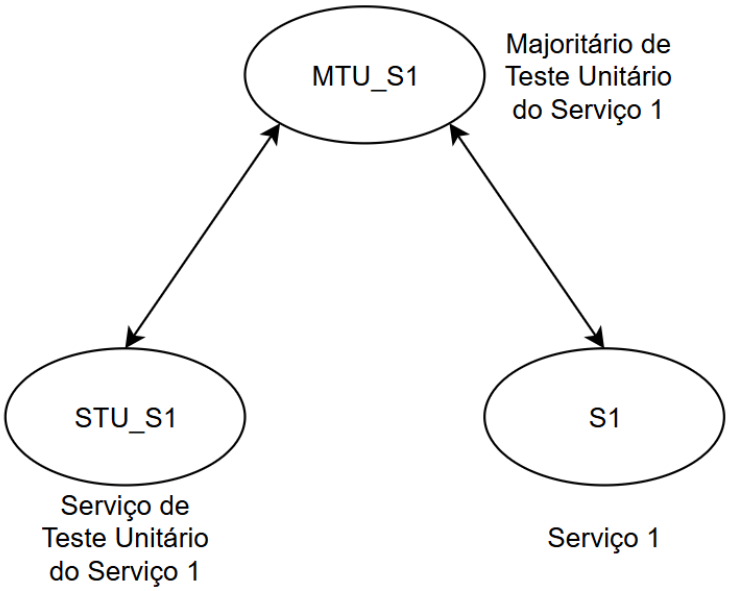
propósito de seguir a independência de tarefas e serviços promovida pela arquitetura SOA;

- **Testes de integração:** elaborados pelo gerente de testes junto aos desenvolvedores envolvidos nos serviços a serem testados. Estes devem ser criados somente após os devidos testes unitários serem executados e aprovados;
- **Teste de sistema:** elaborado pelo gerente de testes junto aos desenvolvedores envolvidos nos testes unitários e de integração. Este teste deve ser criado somente após todos os testes de integração propostos serem executados e aprovados. É importante ressaltar que o teste de sistema requer atenção especial nos seus requisitos, por envolver uma análise rígida de comportamento do sistema.

Com os conceitos de teste melhor ressaltados, agora podemos apresentar o método de aplicação de uma forma que respeite SOA, e que também garanta uma melhor rigidez no momento da execução dos diferentes testes que envolvem o sistema como um todo:

- **Teste unitário de serviço:** feito para testar um serviço, o qual é o componente mais atômico da estrutura SOA. É composto do serviço a ser testado e seu testador conectados a um software majoritário responsável pela comunicação entre eles. Seu diagrama pode ser observado na Figura 3.2;

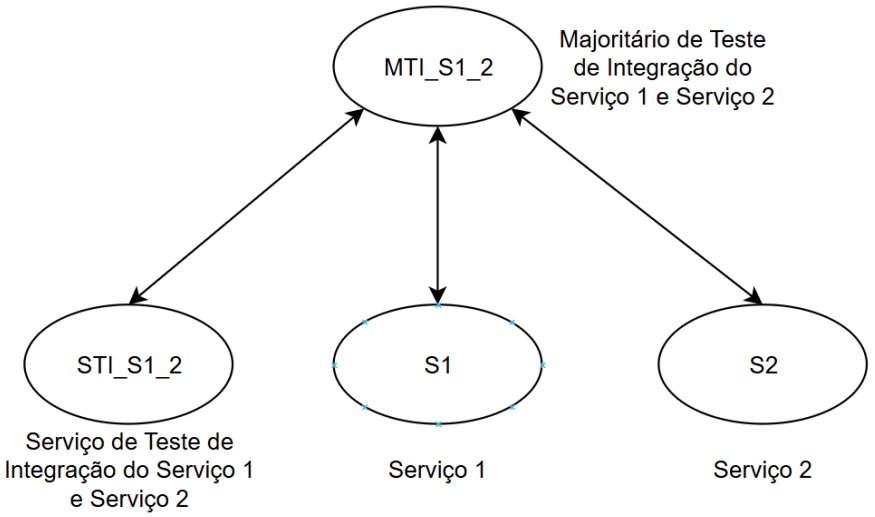
3.2 - Diagrama de funcionamento de um teste unitário de serviço



- **Teste de integração de serviço:** feito para testar dois ou mais serviços executados de maneira conjunta. É composto pelos serviços envolvidos e o testador de ambos

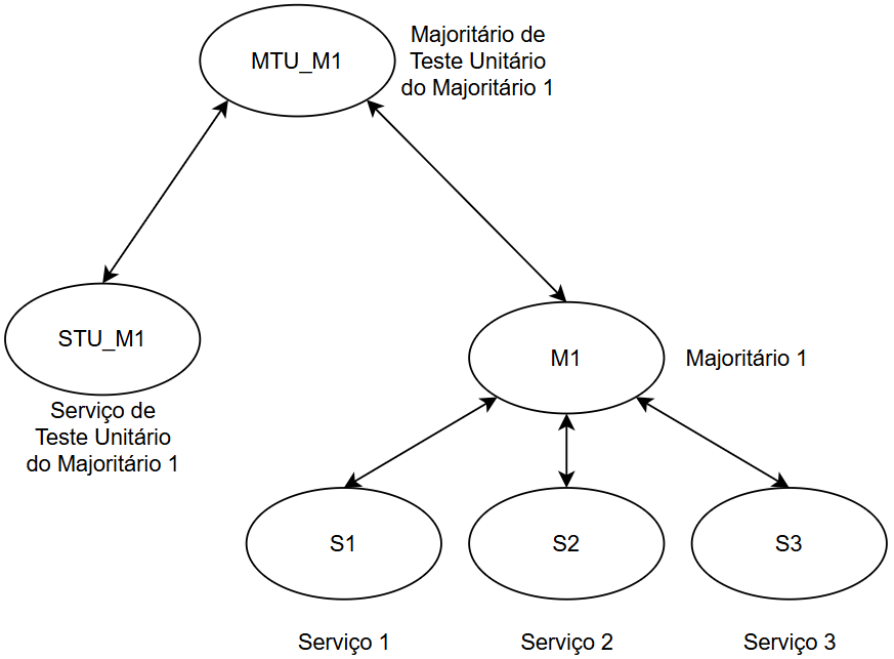
os serviços integrados, conectados a um software majoritário responsável pela comunicação. Seu diagrama pode ser observado na Figura 3.3;

3.3 - Diagrama de funcionamento de um teste de integração de serviço



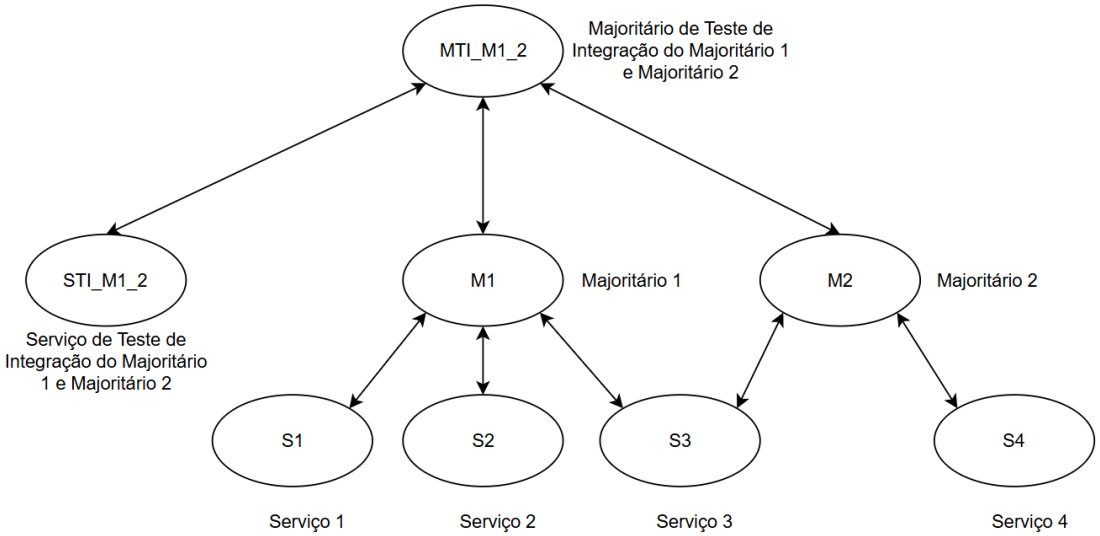
- **Teste unitário de majoritário:** feito para testar o software majoritário de um teste unitário ou de integração. É composto pelo majoritário que se comunica com o(s) serviço(s) testado(s), o testador do majoritário, e um outro software majoritário que se comunica com o majoritário a ser testado e seu testador. Seu diagrama pode ser observado na Figura 3.4. É importante ressaltar que o software majoritário a ser testado deve estar executando o seu papel no teste unitário/de integração, junto ao testador e os serviços envolvidos, para que a análise do teste seja feita da maneira correta;

3.4 - Diagrama de funcionamento de um teste unitário de majoritário



- **Teste de integração de majoritário:** feito para testar a integração entre mais de um software majoritário. É composto pelos softwares majoritários a serem testados, o testador de integração, e um outro software majoritário que se comunica com os softwares majoritários e o testador. Seu diagrama pode ser observado na Figura 3.5.

3.5 - Diagrama de funcionamento de um teste de integração de majoritário



É importante lembrar que todos os testadores envolvidos no plano de testes, assim como alguns dos softwares majoritários, não são uma parte do produto final, assim como sugere a arquitetura do projeto. Por isso, os testadores devem ser usados somente nas etapas

de testes, e dependendo do software analisado, adaptados no software escalonador final, ou removidos completamente da arquitetura final após o uso.

Com os conceitos de testes melhor definidos, é possível, enfim, apresentar a abordagem mestra para as rotinas de teste. Esta abordagem inclui um procedimento o qual todos os testes devem seguir rigorosamente, com o fim de apresentar resultados concretos. Existem quatro etapas principais nesta abordagem, as quais devem ser seguidas em sequência:

- **Aplicação:** o passo inicial da abordagem mestra. Este passo requer muita atenção dos desenvolvedores envolvidos, pois o comportamento esperado do teste depende do conhecimento da ferramenta a ser testada. Aqui, deve-se obter:
  - Os documentos de requisitos e de demanda de produção, para indicar os requisitos necessários para o bom funcionamento do testador desejado, os requisitos de código bem definidos para produção do código final, e as entradas e saídas desejadas no teste em questão;
  - Diagramas criados em UML para análise do comportamento do testador, estes sendo, como apresentados na Seção 2.2.1 do capítulo anterior, os diagramas de casos de uso, atividades e sequência e diagrama de classes;
  - Construção do pseudocódigo, respeitando a ordem estabelecida na Seção 2.2.1, ou seja, anterior ao diagrama de classes.
- **Análise:** contém o estudo de testes estáticos com os diagramas montados na etapa anterior. Os resultados encontrados serão usados para analisar a corretude do comportamento do teste, e correções de possíveis erros encontrados;
- **Implementação:** início da construção do código do testador, seguido de sua execução para colheita de resultados;
- **Documentação:** montagem do relatório final, contendo os resultados obtidos da implementação do código por meio do documento de execução de testes. O documento deve ser enviado para o supervisor de testes para aprovação.

Com a abordagem mestra de testes, é esperado que seja possível melhorar a qualidade de vida de um projeto de maneiras como auxílio na busca e correção de bugs, otimização no tempo de desenvolvimento e a própria validação do software ou hardware envolvido, assim como mencionado no Capítulo 1.

3.3 Considerações finais

Este capítulo apresentou a proposta que este trabalho deseja aplicar em projetos de sistemas embarcados dentro do laboratório Tear. Foi apontado o modo que SOA será aplicado no plano de testes, o direcionamento adequado para suas etapas de desenvolvimento, e o conteúdo de cada procedimento escolhido para o plano de testes discutido neste trabalho, por meio da apresentação da abordagem mestra de testes. O próximo capítulo se aprofunda no procedimento experimental usado no AGV AD01, para que o plano de testes seja validado.

Capítulo 4

VALIDAÇÃO

4.1 Considerações iniciais

Este capítulo contém a apresentação das etapas necessárias para a validação da proposta deste trabalho, na forma de implantação do plano de testes em um dos projetos em desenvolvimento no laboratório Tear. Na Seção 4.2, teremos uma rápida síntese do AGV que foi testado, sua arquitetura de software apresentada em tópicos para cada componente, e a estrutura dos testes montados para a arquitetura. A Seção 4.3 apresenta a execução de todos os testes propostos na Seção 4.2, detalhando a justificativa de existência de cada teste, a maneira de como alguns testes se comunicam com os outros por meio dos dados obtidos, e os procedimentos importantes como a lógica em escala macro do teste em questão, os dados de entrada planejados, e os dados de saída desejados em cada situação.

Os resultados obtidos nos testes podem ser encontrados na Seção 4.4, enquanto a Seção 4.5 apresenta uma análise e uma discussão dos resultados, de forma a compreender alguns pontos que devem ser explicitados em testes mais delicados ou complexos. É esperado que esta análise contribua para o argumento de que testes são importantes para a situação de encontrar erros e corrigir problemas que seriam de grande impacto no momento em que um produto é lançado ao mercado para acesso a diferentes clientes (PRESSMAN, MAXIM, 2019).

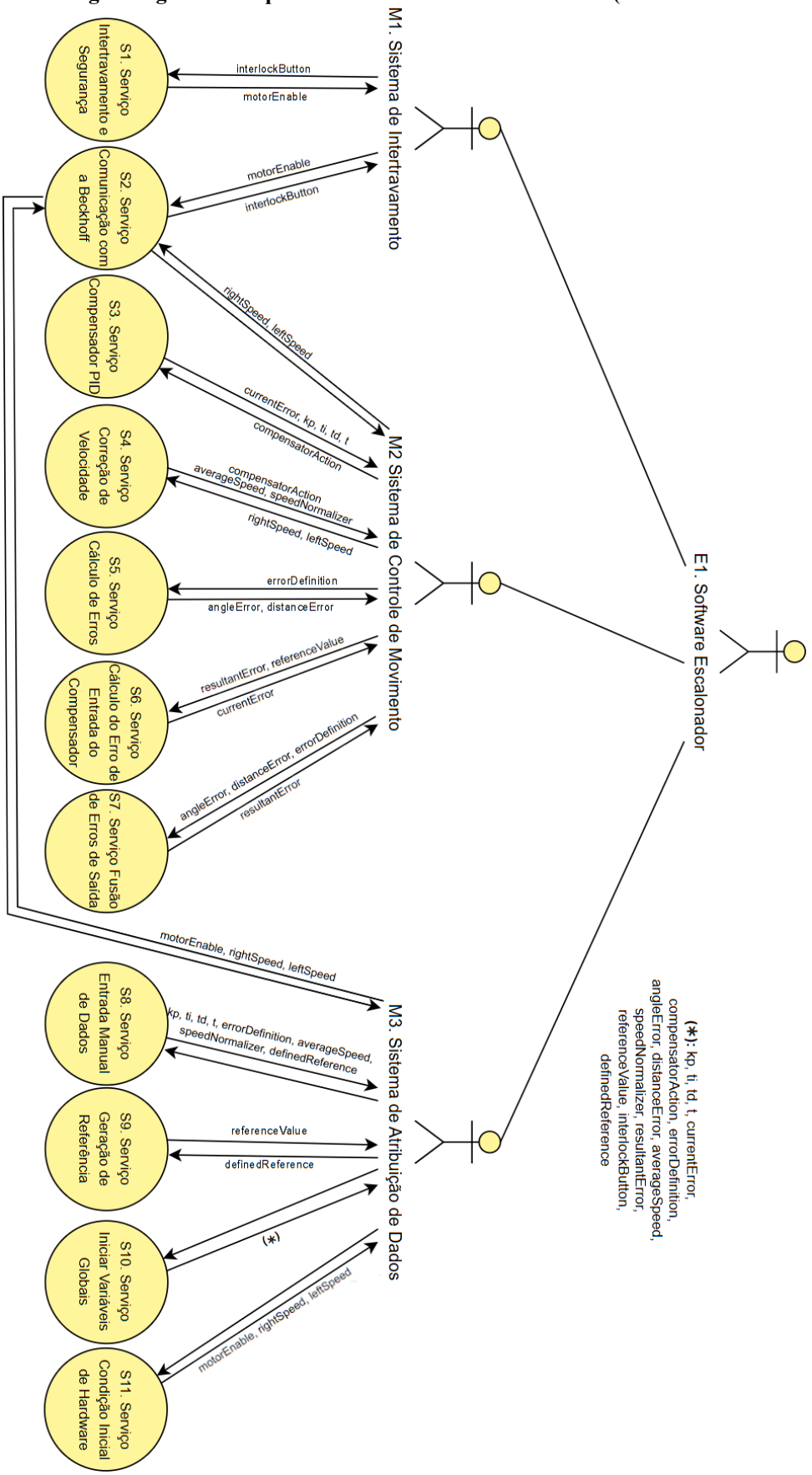
4.2 Plano de experimento

Esta seção contém o plano de experimento usado para validação do projeto apresentado neste trabalho. O plano em si foi completamente montado usando todos os conceitos apresentados no capítulo de proposta do trabalho, usando um dos projetos em construção no laboratório Tear: o AGV AD01. Para que os testes e os resultados façam sentido ao leitor, vamos retomar alguns pontos apresentados nos capítulos anteriores.

Como estabelecido no Capítulo 1, o AD01 é um AGV seguidor de linha montado pelo Tear. Este veículo possui uma arquitetura de hardware em preço mais acessível para o mercado, e uma arquitetura de software baseada em SOA, a qual inclui uma rotina de controle em malha fechada e um compensador PID para correção automática de erro de percurso. Para que seja possível compreender o processo de execução de todos os testes seguindo o plano mostrado neste trabalho, é necessário observar a estrutura do software do AGV. Na Figura 4.1, o diagrama geral da arquitetura de software do AD01 pode ser observado.

O diagrama geral de arquitetura de um software é uma das ferramentas que o laboratório Tear usa para que haja uma melhor compreensão do software. É perceptível que o diagrama é baseado em SOA, assim como a arquitetura de software em questão, por seguir o padrão de relacionamento serviço-majoritário, mas não podemos dizer que o diagrama em questão está respeitando o padrão original estabelecido em SOA por dois motivos: (1) temos o caso de que um componente majoritário está escalonando outros componentes majoritários em uma camada de abstração abaixo; e (2) estamos explicitando os parâmetros de entrada e saída em cada um dos serviços. Estas violações da arquitetura original servem não apenas para que o software seja de maior compreensão e menor complexidade ao observá-lo na escala macro, mas para que as partes envolvidas no desenvolvimento do software possam ter uma melhor visão das entradas e saídas dos serviços, o que auxilia no momento de projetar uma solução viável ao problema que um determinado serviço deve resolver.

4.1 - Diagrama geral da arquitetura de software do AGV AD01 (relatório interno do Tear)



A seguir, os serviços e os majoritários serão apresentados de forma sucinta. Tomando a Figura 4.1 como referência, na camada inferior, da esquerda para a direita, temos os seguintes serviços:

- **Serviço de Intertravamento de Segurança (S1):** serviço encarregado pela indicação de parada crítica das rodas motorizadas do AGV, caso detecte o acionamento de algum dos botões de emergência do veículo.
- **Serviço de Comunicação Com Beckhoff (S2):** serviço que executa as funções de software no sistema Beckhoff do veículo, o qual é encarregado de executar os drivers necessários no hardware para a movimentação do AGV.
- **Serviço de Compensador PID (S3):** serviço que funciona como o compensador PID da malha de controle encontrada no software do AGV.
- **Serviço de Correção de Velocidade (S4):** serviço responsável por corrigir as velocidades dos motores do AGV, a fim de manter o veículo andando na linha encontrada na pista.
- **Serviço de Cálculo de Erros (S5):** serviço que executa o cálculo de erros encontrados após o processamento de uma imagem da linha na pista, capturada pela câmera encontrada no veículo.
- **Serviço de Cálculo de Erro de Entrada do Compensador (S6):** serviço que calcula o valor de erro que deve ser inserido no compensador PID do AGV, para que este faça os cálculos necessários para manter o veículo na linha da pista.
- **Serviço de Fusão de Erros de Saída (S7):** serviço encarregado de gerar um sinal único de saída caso o usuário físico opte pelo cálculo do erro de movimentação do AGV de acordo com a distância da linha ao centro do veículo, erro de acordo com o ângulo da linha em relação à captura de imagem, ou a união dos dois tipos de erro.
- **Serviço de Entrada Manual de Dados (S8):** serviço que recebe os valores de parâmetros globais, que devem ser inseridos pelo usuário físico para atribuição no sistema completo.
- **Serviço de Geração de Referência (S9):** serviço que estabelece o valor global de referência para a malha de controle do AGV, após atribuído pelo usuário físico.

- **Serviço de Iniciar Variáveis Globais (S10):** serviço responsável por ativar e atribuir valores iniciais para todos os parâmetros globais relacionados ao software do veículo, para que todos os serviços possam usá-los sem comprometer o funcionamento do sistema.
- **Serviço de Condição Inicial de Hardware (S11):** serviço que deve ativar e atribuir os valores iniciais dos parâmetros globais que se comunicam diretamente com o hardware do AGV e o sistema Beckhoff, atribuído ao S2.

Na camada central, da esquerda para a direita, temos os seguintes componentes majoritários, que escalonam as chamadas dos serviços anteriormente mencionados:

- **Sistema de Intertravamento (M1):** majoritário responsável por fazer a parada de emergência do AGV, obtendo os valores de intertravamento e indicar a parada do veículo ao sistema Beckhoff.
- **Sistema de Controle de Movimento (M2):** majoritário que, de certo modo, executa os serviços atrelados a si para atuar como uma malha de controle do AGV, a qual trata do movimento do veículo na pista.
- **Sistema de Atribuição de Dados (M3):** majoritário encarregado de fazer o tratamento inicial dos parâmetros envolvidos no sistema, seja a atribuição de valores pelo usuário físico, e a ativação dos parâmetros globais de software e de hardware.

Por fim, a camada superior contém o Software Escalonador (E1), que faz a chamada dos componentes majoritários anteriormente citados em uma determinada ordem, a fim de manter o funcionamento do sistema de maneira organizada.

É importante explicitar que, ao observar o diagrama geral de arquitetura na Figura 4.1, e ter uma melhor compreensão dos componentes envolvidos no sistema do AGV AD01, é possível perceber, de uma certa maneira, que existe uma ordem de chamadas de serviços e componentes majoritários, e esta percepção é possível graças ao modelo de desenvolvimento do diagrama adotado pelo laboratório Tear. Isto se dá porque temos os caminhos de dados entre serviço e majoritário explicitados pelas entradas e saídas de cada um. Por exemplo: é possível perceber que entre os serviços atrelados ao Sistema de Atribuição de Dados (M3), o Serviço de Iniciar Variáveis Globais (S10) deve ser o primeiro serviço a ser executado,

seguido do Serviço de Condição Inicial de Hardware (S11) para iniciar o hardware do AGV com valores padrão, o Serviço de Entrada Manual de Dados (S8) para atribuição dos parâmetros pelo usuário físico, e finalmente, o Serviço de Geração de Referência (S9), que gera um valor fixo de referência para o sistema completo do AGV, após receber o valor do usuário físico por meio do S8; por isso, é também possível compreender que o M3 será o primeiro componente majoritário a ser chamado pelo Software Escalonador (E1), no início da execução do software final do AGV.

Após o sistema completo de software do AD01 ser apresentado, podemos, enfim, prosseguir com o plano de experimento planejado para aplicação do plano de testes neste software em questão. O primeiro passo foi o desenvolvimento de uma estrutura analítica de projeto (EAP), que é um diagrama que auxilia na organização de todas as entregas do projeto em níveis, subdividindo tarefas específicas em partes menores. Após a criação da EAP incluindo todos os testes envolvidos para o projeto, o plano de testes foi seguido de acordo com a sequência apresentada na proposta deste trabalho (Capítulo 3):

- Testes unitários para cada serviço apresentado na Figura 4.1;
- Testes de integração entre os serviços já testados, cujas tarefas em conjunto foram consideradas fundamentais para o bom funcionamento do sistema;
- Testes unitários para cada componente majoritário também apresentado na Figura 4.1;
- Testes de integração entre os componentes majoritários; e
- Teste de sistema para analisar o comportamento do sistema em seu hardware, o qual também pode ser visto como o teste unitário do Software Escalonador (E1) pela estrutura do software e sua semelhança com um sistema de software final.

Cada um dos testes teve seu protocolo único, como mencionado no Capítulo 3, ou seja, foram aprovados pela corretude dos seguintes passos:

- Aplicação da abordagem mestra de testes;
- Análise de documentação e algoritmo;
- Construção de código e implementação do teste para colheita de resultados;
- Documentação final contendo detalhes de execução e análise de resultados.

Com esta estrutura para o plano de experimento, foi possível iniciar a execução de todos os testes necessários para garantir o funcionamento esperado do software que será executado no AGV AD01. É muito importante ressaltar que a ordem de execução dos testes, no caso do AGV, não será feita de forma aleatória, uma vez que alguns serviços possuem dependências de alguns serviços específicos a serem resolvidas, para que o teste deste serviço em específico seja executado corretamente.

Na próxima seção, todos os testes que foram realizados serão detalhados, de modo que seja possível, para cada teste, compreender sua existência, sua operação, e os procedimentos necessários para sua execução.

4.3 Testes

Esta seção apresenta todos os testes criados, aprovados e executados no AGV AD01. Para que esta seção seja melhor compreendida, é importante que três observações sejam feitas, como forma de relembrar o procedimento que os testes foram feitos, e também de auxiliar na compreensão da aplicação do plano de testes proposto no AD01. Estas observações se encontram nos três parágrafos seguintes. Assim como discutido em seções anteriores deste trabalho, o protocolo de documentação do Tear (sua estrutura completa podendo ser encontrada no Capítulo 3) foi usado para estruturar o conceito geral de cada teste, desde o algoritmo padrão à execução do código final para a documentação dos resultados obtidos no teste.

Além disso, um roteiro de compilação cruzada usando a ferramenta CMake também foi criado após o código final. Para executar o arquivo gerado pelo código no AGV, a compilação cruzada é necessária pois a arquitetura do computador principal, o qual as compilações foram montadas, não é a mesma encontrada no veículo. A arquitetura do computador principal é a arquitetura x86, enquanto a do AGV é a arquitetura ARM. A compilação cruzada faz com que o arquivo gerado em uma arquitetura específica seja executável em uma arquitetura diferente da máquina a qual o código foi compilado, e a ferramenta CMake foi usada para que esta compilação fosse alcançada.

Por último, após as compilações cruzadas, todos os arquivos executáveis foram executados no sistema operacional encontrado no AD01, a versão 6.8.1 do sistema Toradex Embedded Linux Reference Multimedia Image (RODRIGUES, 2025). Este sistema operacional é um software usado em sistemas embarcados, controlando as operações necessárias para o bom funcionamento do sistema. A versão encontrada no AGV é de um sistema simples, suficiente para executar o software final do AGV em um tempo de 150 milissegundos por rotina de controle.

A seguir, esta seção estará organizada em diferentes subseções; cada uma terá a discussão de cada teste executado para que o modelo final do software de execução do AD01 seja alcançado, desde o pensamento crítico de planejar entradas de informação para saídas esperadas em cada teste (MYERS, SANDLER, BADGETT, 2012), à manutenção da linha do tempo de execução para contribuição de um teste específico para testes seguintes. A ordem de subseções respeita a ordem de testes apresentada no plano de experimento, na seção anterior, e é importante que esta ordem seja respeitada para que a coerência do processo de obtenção do software final seja mantida.

4.3.1 Testes unitários de serviço

Os primeiros testes montados são os que envolvem os serviços individuais do software final (ver Figura 4.1). Segundo o plano de testes proposto neste trabalho, estes testes são os primeiros a serem executados para que outros mais complexos possam ser considerados no futuro, ou seja, estamos respeitando a estratégia *bottom-up* mencionada na Seção 2.2.2 do Capítulo 2 por fins de coesão do projeto. Em todos os testes, existe um diagrama de caminho de dados, introduzido no Capítulo 3, e que será apresentado diretamente apenas na subseção seguinte, para uma maior compreensão da estrutura. Todos os diagramas de caminho de dados dos outros testes podem ser lidos no Apêndice A deste trabalho.

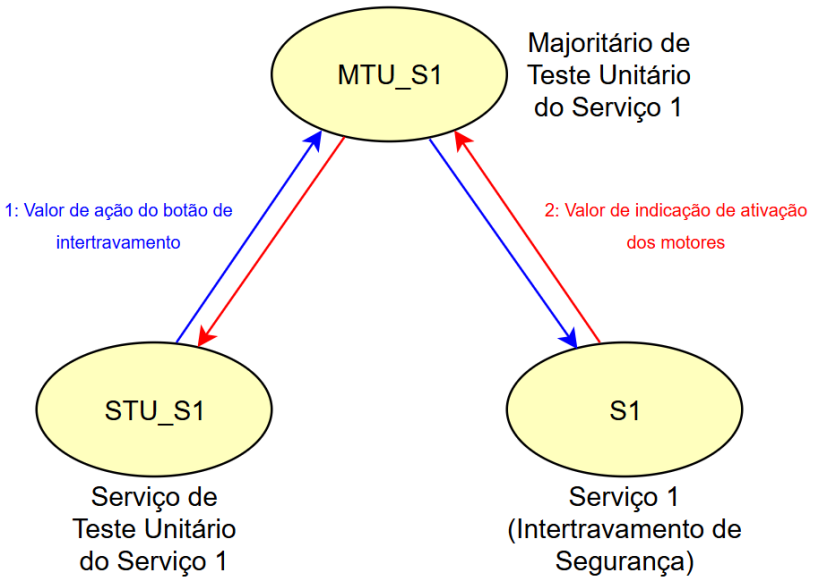
É importante lembrar que, em todos os teste unitários, ambos Serviço de Teste Unitário (STU) e Majoritário de Teste Unitário (MTU) foram desenvolvidos, de forma a respeitar o modelo de software de teste proposto na Seção 3.2, e pode ser melhor observado na Figura 3.2. Para fins de ilustração e compreensão do desenvolvimento de cada documento para um teste, a próxima subseção apresenta um dos testes unitários criados de maneira mais

detalhada do que as demais subseções. Os documentos envolvidos no teste se encontram no Anexo A deste trabalho.

4.3.1.1 Serviço de Intertravamento de Segurança (S1)

O Serviço de Intertravamento de Segurança executa a parada dos drivers de motores do AGV quando um sinal de parada é acionado por meio da botoeira externa. O serviço de teste unitário (STU), para checar o funcionamento correto do intertravamento, deve enviar valores aleatórios do sinal de parada para uso no serviço, por meio do software majoritário de teste unitário (MTU).

4.2 - Diagrama de caminho de dados dos serviços envolvidos no teste do S1



O Serviço de Teste Unitário do S1 (STU_S1) tem três funções principais:

- **Envio de valor:** o testador atualiza o valor do parâmetro do botão de intertravamento por meio do Majoritário de Teste Unitário do S1 (MTU_S1), para que o Serviço de Intertravamento de Segurança possa executar sua função;
- **Recebimento de valor atualizado:** o STU_S1 recebe o valor do parâmetro de habilitação dos motores e checa, em um caso de teste, se o valor recebido era o esperado;

- **Apresentação dos resultados:** após calcular todos os casos de teste, o STU_S1 deve apresentar os valores calculados pelo S1 junto aos valores esperados para cada.

A função principal do Majoritário de Teste Unitário do S1 (MTU_S1) é de escalonar o envio e recebimento de dados entre o S1 e o STU_S1. Este escalonamento funciona por meio de duas funções principais: indicar os parâmetros iniciais na chamada do S1, atualizados pelo STU_S1; e indicar o valor de ação dos motores ao STU_S1, atualizados pelo S1. O MTU_S1 deve fazer esta troca de dados entre os serviços em todas as iterações de cada caso de teste que o STU_S1 possuir.

Este teste tem como objetivo a observação de comportamento do parâmetro ***motorEnable***, ou seja, verificar se o serviço está enviando o sinal correto de saída pelo valor indicado no parâmetro ***interlockButton***. O valor de saída ***motorEnable*** deve estar diretamente relacionado ao valor de entrada ***interlockButton***, e o teste foi planejado para observar se este comportamento se mantém nas duas ocasiões específicas: botão apertado ou não, ***interlockButton*** com valor 1 ou 0, respectivamente. Ressaltamos que, neste teste, o comportamento dos componentes de hardware não são estritamente necessários para sua execução, uma vez que apenas o comportamento dos parâmetros envolvidos estão sendo testados, e não o hardware em si.

Com os requisitos e métodos que devem ser incluídos no teste definidos de forma concreta para garantir o funcionamento do teste, o diagrama de casos de uso do STU_S1 foi criado, seguido do diagrama de atividades e diagrama de sequência. O diagrama de casos de uso apresenta os casos em que o STU_S1 será usado ao ser chamado pelo MTU_S1, enquanto o diagrama de atividades apresenta o fluxo de atividades que o STU_S1 deve executar, e o diagrama de sequência aprofunda o esquema apresentado anteriormente em um formato temporal. Em seguida, o pseudocódigo para apresentar a lógica computacional que deve existir no código final do STU_S1 foi criado, seguido, por fim, do diagrama de classes. Após o término da documentação do Serviço de Teste, a documentação do MTU_S1 foi criada em seguida, com a mesma sequência de geração de documentos que o STU_S1 seguiu, ou seja, pelo modo que a abordagem mestra de testes, apresentada na Seção 3.2, define como a etapa de aplicação.

Como mencionado na subseção anterior, é possível observar a ordem de documentos criados, a fim de exemplificar o processo de aplicação da abordagem mestra deste teste, e explicar a lógica por trás da sua execução, no Anexo A deste trabalho. É esperado que o leitor compreenda que este exemplo seja único, ou seja, os documentos criados para todos os outros testes apresentados neste trabalho não serão mostrados por forma de evitar redundâncias e respeitar a confidencialidade do trabalho do Laboratório Tear.

4.3.1.2 Serviço de Comunicação com Beckhoff (S2)

O Serviço de Comunicação com Beckhoff tem a função de fornecer uma comunicação da CPU e as entradas e saídas do sistema Beckhoff. O Serviço de Teste Unitário do S2 (STU_S2) tem duas funções principais:

- **Chamada de funções:** o testador chama as funções definidas para o S2: leitura digital, escrita digital e escrita analógica;
- **Apresentação dos resultados:** a cada chamada de serviço, o STU_S2 deve apresentar os valores obtidos pelo S2, indicando a corretude ao usuário.

A função principal do Majoritário de Teste Unitário do S2 (MTU_S2) é de escalonar o envio e recebimento de dados entre o S2 e o STU_S2, por meio da indicação dos métodos definidos do S2 ao STU_S2. Este teste não tem a observação de comportamento de algum parâmetro específico, e sim a movimentação dos motores do AD01, e se o movimento do veículo é interrompido ao pressionar algum dos botões de emergência.

4.3.1.3 Serviço de Compensador PID (S3)

O Serviço de Compensador PID realiza o cálculo do compensador PID usando os valores de ganho proporcional, termos integrativo e derivativo, e de erro para encontrar um valor de correção para um sinal desejado. O STU_S3 tem quatro funções principais:

- **Envio de valores:** enviar diferentes valores de ambos os parâmetros de ganho proporcional (***kp***), termo integrativo (***ti***), termo derivativo (***td***) o período (***t***), e o valor de erro corrente (***currentError***), de acordo com cada caso de teste específico;

- **Recebimento de valores:** o testador deve receber os valores de ação de compensador do S3 em cada caso de teste encontrado;
- **Cálculo de erro corrente:** o STU_S3 deve regular o valor de erro corrente para as próximas entradas de cálculo de ação de compensador;
- **Apresentação dos resultados:** o STU_S3 deve apresentar o valor obtido junto ao valor esperado pelo serviço, indicando a corretude ao usuário físico.

A função principal do MTU_S3 é de escalonar o envio e recebimento de dados entre o S3 e o STU_S3, indicando os parâmetros iniciais de um caso de teste ao S3, recebidos pelo STU_S3; e indicando o valor de ação de compensador ao STU_S3, recebido pelo S3. O MTU_S3 deve fazer esta troca de dados entre os serviços toda vez que o STU_S3 for chamado.

O objetivo deste teste é a observação do comportamento do erro atual (***currentError***) com as correções feitas pelo valor de ação de compensador (***compensatorAction***). O teste tratou a entrada como um sistema de controle linear de degrau unitário, ou seja, uma entrada de sinal que vale 0 o qual, no início do teste, recebe valor 1. A saída deve ser o sinal corrigido pelo compensador (S3), enquanto seu comportamento é observado junto ao valor de ação do compensador.

4.3.1.4 Serviço de Correção de Velocidade (S4)

O S4 calcula a velocidade dos motores esquerdo e direito do AGV de acordo com o valor de saída do compensador PID (S3). O cálculo das velocidades dos motores envolve a ação do compensador (***compensatorAction***), o parâmetro de normalização (***speedNormalizer***), e a velocidade desejada do veículo (***averageSpeed***). O STU_S4 tem função única, que é de enviar diferentes valores de ambos parâmetros de ação de compensador, velocidade desejada de um AGV e o parâmetro de normalização, para que o Serviço de Correção de Velocidade (S4) faça os cálculos necessários para, assim, corrigir a velocidade das rodas localizadas no lado direito e esquerdo do veículo em uma trilha. O escalonamento feito pelo MTU_S4 funciona indicando os parâmetros iniciais na chamada do S4, atualizados pelo STU_S4; e os valores de velocidade das rodas do AGV corrigidas ao STU_S4, atualizados pelo S4. O MTU_S4 deve fazer esta troca de dados entre os serviços em todas as iterações de cada caso de teste que o STU_S4 possuir.

Este teste tem um objetivo simples de checar se as operações de modificação de velocidade dos motores do AGV estão ocorrendo da maneira correta, comparando valores desejados de velocidade, de acordo com o caso de teste em questão, com os valores obtidos. Esta checagem se dá pela observação, pelo próprio serviço de testes (STU_S4), dos valores dos parâmetros de velocidade do motor do lado direito do AD01 (***rightSpeed***), e do motor esquerdo (***leftSpeed***), os quais são os valores de saída do serviço original.

4.3.1.5 Serviço de Cálculo de Erros (S5)

O S5 tem a função de calcular o erro em distância e ângulo do AGV em relação à linha guia no chão, por meio de capturas e processamento de imagens. O STU_S5 tem três funções principais:

- **Envio de valores:** o testador deve enviar o caminho de imagem e um valor de definição de erro para o S5, de acordo com cada caso de teste específico;
- **Recebimento de valores:** o STU_S5 deve receber o valor de cálculo de erro do S5 em cada uma das imagens enviadas ao serviço por meio externo;
- **Apresentação dos resultados:** após receber os resultados calculados pelo S5, o STU_S5 deve apresentar os valores obtidos pelo serviço ao usuário físico para análise.

O escalonamento feito pelo MTU_S5 funciona pela indicação dos parâmetros iniciais ao S5, recebidos pelo STU_S5; e dos valores de erro de ângulo e erro de distância ao STU_S5, recebido pelo S5.

O objetivo principal deste teste é a observação dos valores de erro em distância e ângulo em diferentes situações, ou seja, em imagens da linha no chão em modos diferentes, como linha reta, diagonal, curva para a esquerda e curva para a direita. É importante que diferentes cenários sejam observados para que, na questão de processamento de imagem e de identificação de trajetos por meio das técnicas de visão computacional, o veículo possa seguir a pista dedicada de modo seguro, e de forma que não haja vícios no algoritmo original, ou seja, garantir que o AD01 se movimente da forma correta em apenas um cenário.

4.3.1.6 Serviço de Cálculo do Erro de Entrada do Compensador (S6)

O S6 executa o cálculo do erro de entrada do Serviço de Compensador PID (S3) usando os valores de erro resultante e valor de referência de uma malha de controle para encontrar o erro resultante para uso no PID. O STU_S6 tem quatro funções principais:

- **Envio de valores:** o testador deve enviar valores de erro resultante e valor de referência, de acordo com cada caso de teste específico;
- **Recebimento de valores:** o testador deve receber o valor de erro corrente do S6 em cada caso de teste encontrado;
- **Comparação de resultados obtidos:** o STU_S6, após receber o valor de erro corrente, deve compará-lo ao valor esperado para dado caso de teste;
- **Apresentação dos resultados:** após a comparação feita em um caso de teste, o STU_S6 deve apresentar o valor obtido junto ao valor esperado pelo serviço, indicando a corretude ao usuário físico.

O escalonamento feito pelo MTU_S6, assim como nos serviços de teste anteriores, funciona por meio de duas funções principais: indicar os parâmetros iniciais ao S6, recebidos pelo STU_S6; e indicar o valor de erro corrente ao STU_S6, recebido pelo S6.

O propósito deste teste é verificar se o valor de referência (***referenceValue***) está sendo somado ao valor de erro ***currentError*** da forma correta para ação no compensador PID (S3) do AD01. Logo, ao obter um valor de erro final, o teste deve verificar se o valor obtido é o mesmo que o valor desejado em cada uma das entradas de teste.

4.3.1.7 Serviço de Fusão de Erros de Saída (S7)

O S7 executa o cálculo do erro de saída resultante usando os valores de erro em ângulo e erro em distância do AGV, usando também o valor de definição de erro para fazer o cálculo correto para um instante específico do movimento do AGV. O STU_S7 tem quatro funções principais:

- **Envio de valores:** o testador deve enviar valores de erro de ângulo de inclinação do AGV, erro de distância do AGV à linha, e o valor que indica o

tipo de erro a ser usado na malha de controle, de acordo com cada caso de teste específico;

- **Recebimento de valores:** o testador deve receber o valor de erro resultante do Serviço de Fusão de Erros de Saída (S7) em cada caso de teste encontrado;
- **Comparação de resultados obtidos:** o STU_S7, após receber o valor de erro resultante, deve compará-lo ao valor esperado para dado caso de teste;
- **Apresentação dos resultados:** após a comparação feita em um caso de teste, o STU_S7 deve apresentar o valor obtido junto ao valor esperado pelo serviço, indicando a corretude ao usuário físico.

O escalonamento feito pelo MTU_S7 também funciona por meio de duas funções principais: indicar os parâmetros iniciais ao S7, recebidos pelo STU_S7; e indicar o valor de erro corrente ao STU_S7, recebido pelo S7.

Este teste, assim como testes anteriores, foi feito para analisar o comportamento dos parâmetros que possuem os valores de erro de ângulo e erro de distância do AD01 à linha (***angleError*** e ***distanceError***, respectivamente), e se os valores são corretamente atribuídos no parâmetro de erro resultante (***resultantError***), segundo o valor de tipo de erro desejado para uso (***errorDefinition***). Para isso, o teste também funciona com a comparação entre valores obtidos e desejados.

4.3.1.8 Serviço de Entrada Manual de Dados (S8)

O S8 tem a função de atribuir valores, determinados pelo usuário físico, nas variáveis globais encontradas no sistema de software do AGV. Neste caso, enquanto o MTU_S8 apenas indica os valores atribuídos aos parâmetros ao STU_S8, este tem três funções principais:

- **Recebimento de valores:** o STU_S8 deve receber os valores dos parâmetros inseridos pelo usuário físico por meio do Serviço de Entrada Manual de Dados (S8), em cada caso de teste encontrado;
- **Checagem de resultados obtidos:** o STU_S8, após receber os valores de cada parâmetro, deve observá-los para checar se os critérios de atribuição foram respeitados, para dado caso de teste;

- **Apresentação dos resultados:** após a comparação feita em um caso de teste, o STU_S8 deve apresentar os valores das variáveis, indicando a corretude geral ao usuário.

Este teste é mais simples, comparado aos outros testes unitários de serviço, e isso acontece pois seu único objetivo é a verificação de que todos os parâmetros tiveram seus valores atribuídos, em vez de uma comparação direta com valores desejados.

4.3.1.9 Serviço de Geração de Referência (S9)

O S9 tem a função única de gerar o sinal de referência para a malha de controle de um sistema. Enquanto o MTU_S9 faz o escalonamento de tarefas entre testador e serviço como nos serviços de teste anteriores, O STU_S9 tem quatro funções principais:

- **Envio de valores:** o testador deve enviar um valor único de referência definida, de acordo com cada caso de teste específico;
- **Recebimento de valores:** o testador deve receber o valor de referência do S9 em cada caso de teste encontrado;
- **Comparação de resultados obtidos:** o STU_S9, após receber o valor de referência, deve compará-lo ao valor esperado para dado caso de teste;
- **Apresentação dos resultados:** após a comparação feita em um caso de teste, o STU_S9 deve apresentar o valor obtido junto ao valor esperado pelo serviço, indicando a corretude ao usuário físico.

Assim como nos outros testes que requerem comparações, este teste é ainda mais simples, pois busca garantir apenas que o valor de referência dado pelo usuário físico (***definedReference***) está sendo passado para o valor de referência a ser usado pelo sistema (***referenceValue***).

4.3.1.10 Serviço de Iniciar Variáveis Globais (S10)

O Serviço de Iniciar Variáveis Globais tem a função de iniciar as variáveis globais do software geral. Enquanto o MTU_S10 faz o escalonamento de chamadas de comparação entre serviço e testador, o STU_S10 tem três funções principais:

- **Recebimento de valores:** o STU_S10 deve receber os valores das variáveis globais do S10;
- **Comparação de resultados obtidos:** o testador, após receber os parâmetros com valores atualizados, deve checar se todos eles possuem um valor diferente do valor esperado.
- **Apresentação dos resultados:** após a comparação feita em um caso de teste, o STU_S10 deve apresentar o valor obtido, indicando a corretude ao usuário físico.

Diferente dos outros testes, este requer a verificação contrária, ou seja, se os valores de cada parâmetro são diferentes do esperado. De todos os parâmetros do sistema geral, é desejado que eles não possuam valores nulos, absurdos (exemplo: um parâmetro que armazena apenas números obter a letra "A" como valor), ou incompatíveis (exemplo: valores que não são um número ou *Not a Number*, conhecido como "NaN"). Por isso, a verificação contrária se prova mais eficiente neste teste.

4.3.1.11 Serviço de Condição Inicial de Hardware (S11)

O Serviço de Condição Inicial de Hardware tem a função de iniciar parâmetros de hardware do sistema. Enquanto o MTU_S11 também escalona as tarefas entre serviço e testador, o STU_S11 tem três funções principais:

- **Recebimento de valores:** o testador deve receber os valores das variáveis do S11;
- **Comparação de resultados obtidos:** o STU_S11, após receber os parâmetros com valores atualizados, deve checar se todas elas possuem um valor diferente do valor esperado;
- **Apresentação dos resultados:** após a comparação feita em um caso de teste, o STU_S11 deve apresentar o valor obtido, indicando a corretude ao usuário.

Este teste é bastante similar ao teste feito para o serviço da subseção anterior, uma vez que o S11 funciona de forma bastante semelhante ao S10, com a diferença sendo os parâmetros a serem testados, que dependem da ação do hardware do sistema, e não de uma ação direta do testador ou do usuário físico.

4.3.2 Testes de integração de serviços

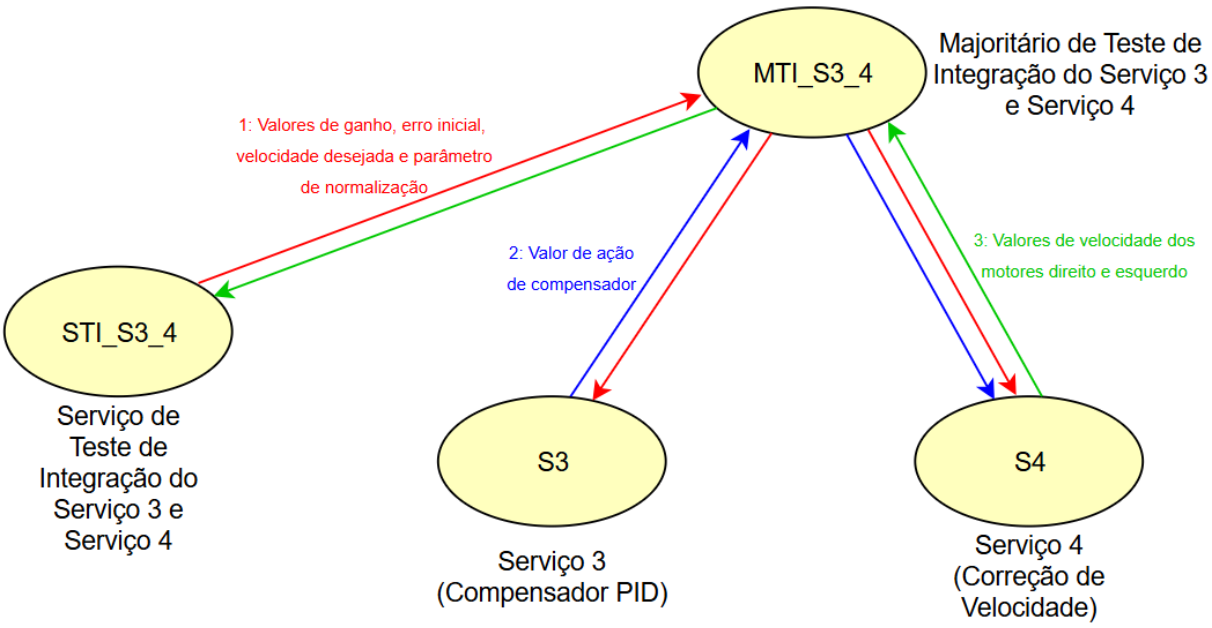
Após a validação dos testes unitários e a aprovação de todos, os testes seguintes a serem implementados foram os testes de integração de serviços, que contam com a união da execução de dois ou mais serviços do software final do AD01 para executar uma tarefa específica do teste em questão. Todos os testes de integração são formas de observar se alguns serviços já testados trabalham em conjunto antes de serem conectados em seu respectivo software majoritário para escalonamento, e não exatamente a checagem de dados dentro de cada um, uma vez que o desempenho individual de cada serviço já é conhecido graças aos testes unitários. Esta lógica de "testar antes de conectar ao respectivo majoritário" também foi usada para selecionar quais testes de integração fariam sentido em existir, de forma que a escolha de integração não seja aleatória.

Como comentado na Seção 4.3.1, todos os testes possuem um diagrama de caminho de dados para melhor interpretação das suas estruturas. Na próxima seção, um dos testes de integração será apresentado com o seu diagrama de caminho de dados, enquanto os testes restantes terão seus diagramas inseridos no Apêndice A deste trabalho.

4.3.2.1 Integração dos serviços S3 e S4

O Serviço de Teste de Integração dos Serviços 3 e 4 (STI_S3_4) tem a função de gerenciar o funcionamento conjunto do Serviço de Compensador PID (S3) e o Serviço de Correção de Velocidade (S4). O Serviço de Teste de Integração deve indicar os valores de ação de compensador, junto aos valores de correção de velocidade aos serviços de software envolvidos. Todos estes serviços funcionam por meio de um software Majoritário de Teste de Integração (MTI_S3_4). A Figura 4.2 indica o caminho de dados que foi usado no teste a ser discutido a seguir.

4.3 - Diagrama de caminho de dados dos serviços envolvidos no teste de integração do S3 e S4



O STI_S3_4 tem cinco funções principais:

- **Envio de valores:** o serviço de teste deve enviar diferentes valores de ambas as variáveis de ganho proporcional (***kp***), termo integrativo (***ti***), termo derivativo (***td***), o período (***t***), o valor de erro atual, a velocidade desejada pelo veículo, e o parâmetro de normalização de velocidade, de acordo com cada caso de teste específico;
- **Recebimento de valor de erro e ação de compensador:** o STI_S3_4 deve receber os valores de erro e da ação de compensador atualizados pelo S3;
- **Recebimento de velocidade dos motores:** o testador deve, em seguida, receber os valores de velocidade esquerda e direita dos motores (***leftSpeed*** e ***rightSpeed***, respectivamente), calculados pelo S4;
- **Apresentação dos resultados:** após a comparação feita, o STI_S3_4 deve apresentar os valores obtidos de ação de compensador, erro atual e velocidade dos motores para análise do usuário físico;
- **Cálculo de erro atual:** o STI_S3_4 deve regular o valor de erro atual para as próximas entradas de cálculo de ação de compensador.

A função principal do MTI_S3_4 é de escalonar o envio e recebimento de dados entre o S3, o S4, e o STI_S3_4. Este escalonamento funciona por meio das seguintes funções: indicar os parâmetros iniciais de um caso de teste ao S3 e S4, recebidos pelo STI_S3_4; fazer

a chamada do S3 para cálculo de ação do compensador; fazer a chamada do S4 para cálculo de velocidade dos motores; e fazer a chamada do STI_S3_4 para apresentar os resultados do caso de teste em questão. O MTI_S3_4 deve fazer esta troca de dados entre os serviços toda vez que o STI_S3_4 for chamado.

4.3.2.2 Integração dos serviços S5, S6 e S7

O Serviço de Teste de Integração dos Serviços 5, 6 e 7 (STI_S5_6_7) gerencia o funcionamento conjunto do Serviço de Cálculo de Erros (S5), Serviço de Cálculo de Erro de Entrada do Compensador (S6), e do Serviço de Fusão de Erros de Saída (S7). Para checar o funcionamento correto da manipulação de erros, o STI_S5_6_7 deve verificar se o valor obtido de erro de entrada do compensador no S6 está coerente com o valor esperado, de acordo com a imagem enviada para cálculo de erro no S5, seguido do valor de erro resultante no S7. Todos estes serviços funcionam via escalonamento feito pelo software Majoritário de Teste de Integração (MTI_S5_6_7). O STI_S5_6_7 tem quatro funções principais:

- **Envio de valores:** o serviço de teste deve enviar diferentes valores aos parâmetros de definição de erro a ser calculado, valor de referência, e caminho de imagem para filtragem, de acordo com cada caso de teste específico;
- **Recebimento de valores parciais de erro:** o STI_S5_6_7 deve receber os valores de erro em ângulo e distância, atualizados pelo S5;
- **Recebimento de erro resultante:** o testador deve, em seguida, receber o valor de erro resultante, calculado pelo S7;
- **Recebimento de erro atual:** o STI_S5_6_7 deve receber o valor de erro atual, calculado pelo S6, para comparação final de resultado;
- **Apresentação dos resultados:** após a comparação feita, o STI_S5_6_7 deve apresentar o valor obtido de erro atual, junto ao valor esperado, para análise do usuário físico.

Enquanto o STI_S5_6_7 faz o gerenciamento dos dados envolvidos nos testes, o escalonamento do MTI_S5_6_7 funciona por meio das seguintes funções: indicar o valor das variáveis globais do STI_S5_6_7 aos outros serviços; fazer a chamada do S5 para cálculo de erros de ângulo e distância; fazer a chamada do S6 para cálculo de erro resultante; fazer a

chamada do S7 para cálculo de erro corrente; e fazer a chamada do STI_S5_6_7 para apresentar o resultado final do caso de teste em questão.

4.3.2.3 Integração dos serviços S8, S9 e S10

O Serviço de Teste de Integração dos Serviços 8, 9 e 10 (STI_S8_9_10) gerencia o funcionamento conjunto do Serviço de Entrada Manual de Dados (S8), Serviço de Geração de Referência (S9), e do Serviço de Iniciar Variáveis Globais (S10). O STI_S8_9_10 deve checar se a inicialização foi feita corretamente pelo S10, seguido da checagem da entrada manual de dados, e se a geração de referência foi feita de acordo com a entrada manual deste valor. Todos estes serviços funcionam por meio do escalonamento feito pelo software Majoritário de Teste de Integração (MTI_S8_9_10). O STI_S8_9_10 tem cinco funções principais:

- **Checagem de variáveis globais:** o testador deve receber os valores das variáveis globais do S10 e checar se foram iniciadas corretamente;
- **Recebimento de valores do usuário físico:** o STI_S8_9_10 deve receber os valores dos parâmetros inseridos pelo usuário físico por meio do S8;
- **Recebimento de valor de referência:** o testador deve, em seguida, receber um novo valor de referência, calculado pelo S9, após atualização do parâmetro ***definedReference*** por meio do S8;
- **Comparação de resultados obtidos:** o STI_S8_9_10, após receber o valor de referência, deve compará-lo, junto às outras variáveis declaradas pelo usuário via S8, ao valor esperado para dado caso de teste;
- **Apresentação dos resultados:** o STI_S8_9_10 deve apresentar o valor obtido junto ao valor esperado pelo serviço, indicando a comparação ao usuário físico.

Assim como nos outros testes de integração, o MTI_S8_9_10 tem função principal de escalonar o envio e recebimento de dados entre os serviços envolvidos, estes sendo o S8, S9, S10 e STI_S8_9_10. Este escalonamento funciona por meio das seguintes funções: indicar o valor das variáveis globais ao STI_S8_9_10, recebidos pelo S10; escalonar a chamada dos serviços S8 e S9; fazer a chamada do S8 para entrada manual de dados pelo usuário físico; indicar o valor de definição de referência ao S9; e fazer a chamada do STI_S8_9_10 para apresentar o resultado final do caso de teste em questão.

4.3.3 Testes unitários de majoritário

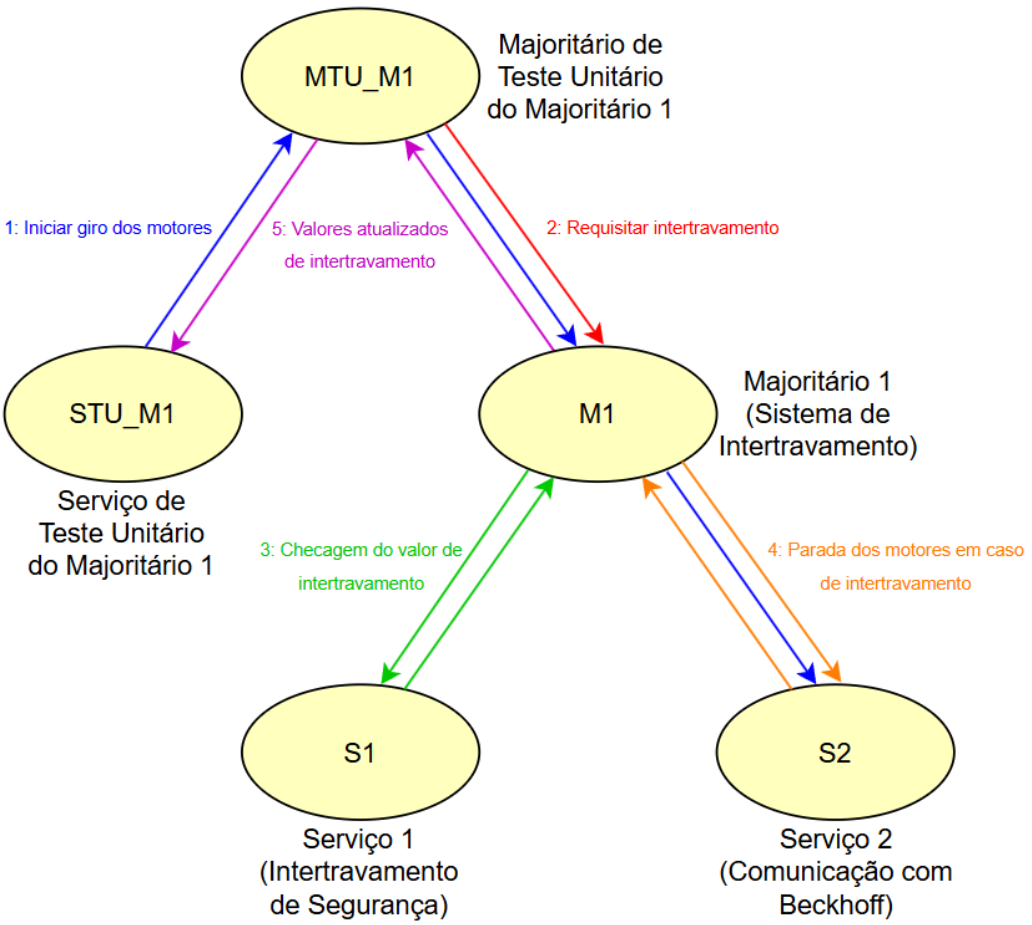
Respeitando a estratégia *bottom-up* de implementação do plano de testes deste trabalho, os próximos softwares a serem testados foram os componentes majoritários do projeto. Estes testes puderam ser iniciados a partir do momento em que todos os serviços foram testados, mas de forma que os testes de integração propostos também foram validados, uma vez que um ponto interessante pode ser discutido neste caso. Alguns conceitos para validação dos componentes majoritários foram iniciados antes mesmo do teste destes softwares em si, por meio dos testes de integração entre serviços. Isso acontece pois os componentes majoritários trabalham com a integração de mais de um serviço, como pode ser observado na Figura 4.1. É possível perceber que os testes de integração propostos nas subseções anteriores encaixam perfeitamente na arquitetura que compõe os componentes majoritários do projeto, de forma a testar parte da sequência de tarefas de cada majoritário.

As subseções seguintes contém os testes de cada componente majoritário, levando o funcionamento dos serviços atrelados a cada um em conta. Como ocorrido nas subseções anteriores, a subseção seguinte apresenta a estrutura de um diagrama de caminho de dados em um teste unitário de majoritário para fins de ilustração, enquanto os diagramas dos testes restantes podem ser lidos no Apêndice A deste trabalho.

4.3.3.1 Sistema de Intertravamento (M1)

O Majoritário 1 (M1) tem a função de interromper o movimento do AGV. O Serviço de Teste Unitário (STU_M1), para checar o funcionamento correto da transmissão de dados, deve testar o método único do M1 para checar se ambos Serviço de Comunicação com Beckhoff (S2) e o Serviço de Intertravamento de Segurança (S1) estão se comunicando corretamente com o M1, por meio do escalonamento de tarefas por um software Majoritário de Teste Unitário (MTU_M1). A Figura 4.4 apresenta o diagrama de caminho de dados do teste discutido.

4.4 - Diagrama de caminho de dados dos serviços envolvidos no teste do M1



O STU_M1 tem duas funções principais:

- **Iniciar giro dos motores:** o testador chama uma função de escrita analógica ao S2 para que os motores do AGV AD01 comecem a girar;
- **Apresentação de valores e resultados:** após o MTU_M1 requisitar o M1, o STU_M1 deve apresentar os valores dos parâmetros envolvidos no teste, para indicar a corretude ao usuário físico.

A função principal do MTU_M1, assim como outros majoritários de teste discutidos anteriormente, é o escalonamento de tarefas por meio da manipulação dos dados que passam pelo STU_M1 e M1. Este escalonamento funciona a partir de duas funções principais: fazer a chamada do STU_M1 para iniciar e gerenciar o teste; e fazer a chamada do M1 para executar a integração entre S1 e S2.

4.3.3.2 Sistema de Controle de Movimento (M2)

O Majoritário 2 (M2) tem a função de fazer a ordem de execução da malha de controle no AGV. O Serviço de Teste Unitário (STU_M2), para checar o funcionamento correto desta execução, deve testar se o método único do M2 funciona corretamente, este sendo o método que faz a chamada dos serviços de Comunicação com Beckhoff (S2), Compensador PID (S3), Correção de Velocidade (S4), Cálculo de Erros (S5), Cálculo de Erro de Entrada do Compensador (S6) e Fusão de Erros de Saída (S7). Todas estas chamadas são feitas por meio do software Majoritário de Teste Unitário (MTU_M2). O STU_M2 tem três funções principais:

- **Envio de valores:** o serviço de teste deve enviar diferentes valores de ambas as variáveis de ganho proporcional (***kp***), termo integrativo (***ti***), termo derivativo (***td***), período (***t***), a velocidade desejada pelo AGV, o parâmetro de normalização de velocidade, a definição de erro a ser usado, e valor de referência da malha de controle, de acordo com cada teste específico;
- **Recebimento de valores:** o STU_M2 deve receber diferentes valores atualizados pelo M2 nos serviços atrelados a ele:
    - Erros em ângulo e distância do AGV, erro resultante, e valor de erro atual (vindos do S5, S7 e S6, respectivamente);
    - Ação de compensador (S3);
    - Velocidade dos motores (S4);
    - Ações do AGV pela comunicação com a Beckhoff (S2).
- **Apresentação dos resultados:** após receber todos os valores, o STU_M2 deve apresentar todos os valores obtidos ao usuário físico.

Enquanto o escalonamento entre serviços ocorre em M2, o MTU_M2 faz o escalonamento entre componente majoritário e testador. Este escalonamento funciona por meio das seguintes funções: indicar os parâmetros iniciais de um caso de teste ao M2, recebidos pelo STU_M2; fazer a chamada do M2 para os cálculos dos serviços envolvidos; e fazer a chamada do STU_M2 para apresentar os resultados do caso de teste em questão.

4.3.3.3 Sistema de Atribuição de Dados (M3)

O Majoritário 3 (M3) tem a função de iniciar todas as condições iniciais do software e hardware. O Serviço de Teste Unitário (STU_M3), para checar o funcionamento correto da inicialização, deve testar o método único do M3 para checar se ambos Serviço de Comunicação com Beckhoff (S2), Serviço de Entrada Manual de Dados (S8), Serviço de Geração de Referência (S9), Serviço de Iniciar Variáveis Globais (S10) e Serviço de Condição Inicial de Hardware (S11) estão se comunicando corretamente com o M3. O STU_M3 tem seis funções principais:

- **Checagem de variáveis globais:** o testador deve receber os valores das variáveis globais do S10 e checar se foram iniciadas corretamente;
- **Checagem de giro dos motores:** o testador checa os valores das variáveis envolvidas no giro dos motores, ou seja, o parâmetro de habilitação dos motores, e a leitura digital do giro dos motores pelo Serviço de Comunicação com Beckhoff (S2);
- **Recebimento de valores do usuário físico:** o STU_M3 deve receber os valores dos parâmetros inseridos pelo usuário físico por meio do S8;
- **Recebimento de valor de referência:** o testador deve, em seguida, receber um novo valor de referência, calculado pelo S9, após o valor ser indicado pelo usuário via S8;
- **Comparação de resultados obtidos:** o STU_M3, após receber o valor de referência, deve compará-lo, junto às outras variáveis declaradas pelo usuário via S8, ao valor esperado para dado caso de teste;
- **Apresentação dos resultados:** após a comparação feita, o STU_M3 deve apresentar o valor obtido junto ao valor esperado pelo serviço ao usuário físico.

O MTU_M3 funcionará da mesma forma que os MTUs dos testes de majoritário apresentados nas subseções anteriores, ou seja, possui a função de escalonar o envio e recebimento de dados entre M3 e STU_M3, por meio das seguintes funções: indicar o valor das variáveis globais ao STU_M3, recebidos pelo M3; fazer a chamada do M3 para executar

os serviços atrelados a ele; e fazer a chamada do STU_M3 para apresentar o resultado final do teste.

4.3.4 Teste de integração dos majoritários M1 e M3

Esta subseção contém o único teste de integração de majoritários cogitado na fase de testes de software do AGV AD01. Testes de integração de componentes majoritários envolvendo o M2 se mostraram desnecessários pois qualquer teste de integração entre majoritários precisaria de movimentação do AGV de alguma forma, seja por controle do M2 ou por um comando de movimento simples por meio da ativação do Serviço de Comunicação com Beckhoff (S2). Logo, para manter o escopo do teste mais simples, de forma a focar no funcionamento de dois majoritários funcionando de maneira simultânea, apenas o teste de integração entre M1 e M3 foi elaborado.

O Serviço de Teste de Integração dos Majoritários 1 e 3 (STI_M1_3) tem a função de gerenciar o funcionamento conjunto do Sistema de Intertravamento (M1) e o Sistema de Atribuição de Dados (M3). Como discutido anteriormente, o M1 tem a função de interromper o movimento do AGV, e o M3 tem a função de iniciar todas as condições iniciais do software e hardware. O STI_M1_3 deve indicar os valores atribuídos pelo M3, seguido dos valores do M1 caso o intertravamento seja ativado pelo usuário físico via botoeiras do AGV. Todos estes serviços funcionam por meio de um software Majoritário de Teste de Integração (MTI_M1_3). O diagrama de caminho de dados deste teste pode ser encontrado no Apêndice A deste documento.

Por conter mais elementos de software envolvidos no teste, o STI_M1_3 possui sete funções principais:

- **Checagem de variáveis globais:** o testador deve receber os valores das variáveis globais do Serviço de Iniciar Variáveis Globais (S10) e checar se foram iniciadas corretamente;
- **Checagem de giro dos motores:** o testador checa os valores das variáveis envolvidas no giro dos motores, ou seja, o parâmetro de habilitação dos motores, e a leitura digital do giro dos motores pelo Serviço de Comunicação com Beckhoff (S2);

- **Recebimento de valores do usuário físico:** o STI_M1_3 deve receber os valores dos parâmetros inseridos pelo usuário físico por meio do Serviço de Entrada Manual de Dados (S8);
- **Recebimento de valor de referência:** o testador deve, em seguida, receber um novo valor de referência de malha de controle, calculado pelo Serviço de Geração de Referência (S9), após atualização do parâmetro de referência definida por usuário físico por meio do S8;
- **Comparação de resultados obtidos:** o STI_M1_3, após receber o valor de referência, deve compará-lo, junto às outras variáveis declaradas pelo usuário físico via S8, ao valor esperado para dado caso de teste;
- **Iniciar giro dos motores:** o testador chama uma função de escrita analógica ao Serviço de Comunicação com Beckhoff (S2) para que os motores do AGV AD01 comecem a girar;
- **Apresentação de valores e resultados:** após o MTI_M1_3 requisitar o Sistema de Intertravamento (M1), o STI_M1_3 deve apresentar os valores das variáveis envolvidas no teste ao usuário físico.

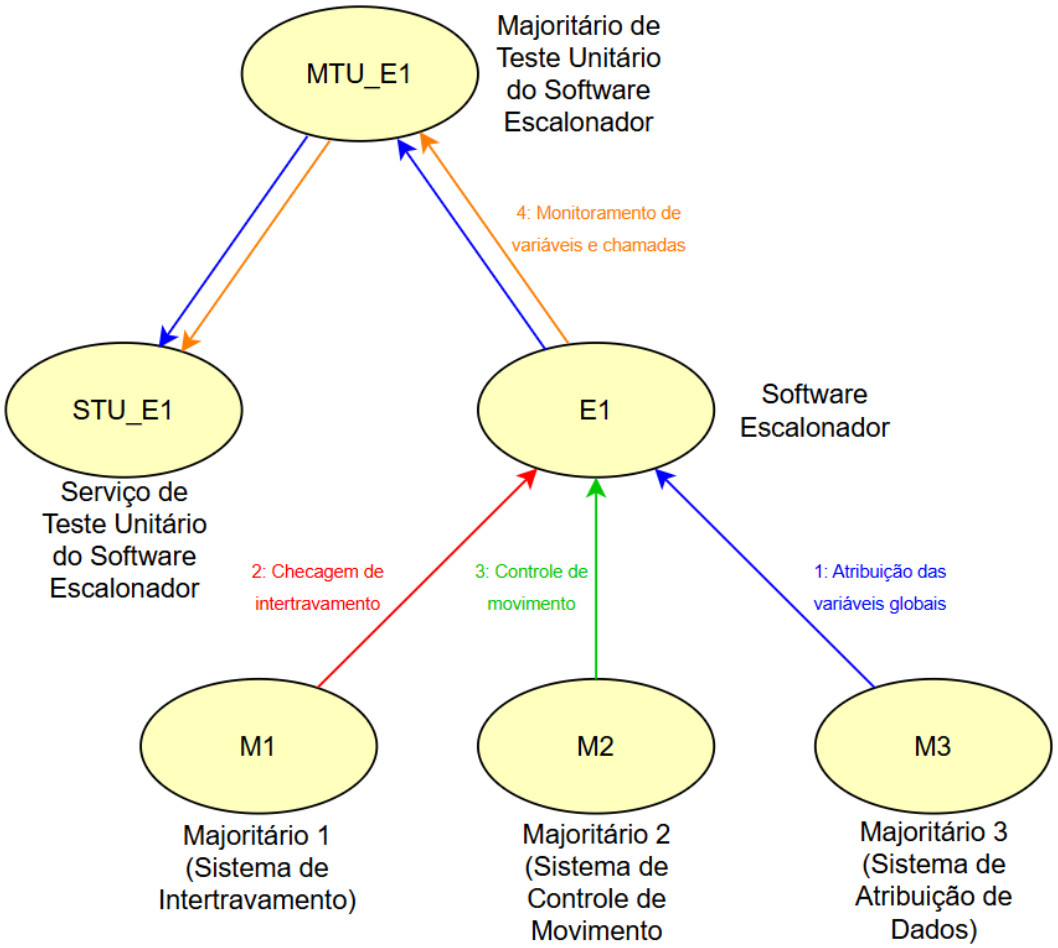
Como esperado, o MTI_M1_3 faz o serviço de escalonamento de dados entre tarefas executadas pelos majoritários envolvidos e o STI_M1_3. Este escalonamento funciona pelas seguintes funções: fazer a chamada do STI_M1_3 para iniciar e gerenciar o teste; indicar o valor das variáveis globais ao STI_M1_3, recebidos pelo M3; fazer a chamada do M3 para executar os serviços ligados a ele; fazer a chamada do STI_M1_3 para apresentar o resultado final da atribuição das variáveis globais; e fazer a chamada do M1 para executar a integração entre os seus serviços.

4.3.5 Teste do Software Escalonador (E1)

Esta subseção apresenta a estrutura do teste unitário do Software Escalonador (E1) do AGV AD01. De certa forma, este teste pode ser considerado como uma análise da integração entre os três componentes majoritários que existem no sistema final. Entretanto, não podemos afirmar isso com convicção, uma vez que este teste foi executado com a validação do funcionamento do E1 em mente, em vez da execução dos três majoritários de forma integrada.

O E1 tem a função de realizar a ordem de execução dos sistemas de Intertravamento (M1), Controle de Movimento (M2) e Atribuição de Dados (M3), em um determinado período de tempo. O Serviço de Teste Unitário (STU_E1) deve apresentar os dados de todos os parâmetros envolvidos no sistema por meio da chamada feita pelo software Majoritário de Teste Unitário (MTU_E1). A Figura 4.5 abaixo apresenta o diagrama de caminho de dados deste teste.

4.5 - Diagrama de caminho de dados dos majoritários envolvidos no teste do E1



O STU_E1 tem quatro funções principais:

- **Recebimento de valores:** o STU_E1 deve receber todos os valores obtidos pelo E1 ao iniciar o Sistema de Atribuição de Dados, ou seja, os valores das variáveis globais.
- **Monitoramento de valores:** após receber todos os valores, o STU_E1 deve fazer uma checagem se alguns dos valores obtidos estão de acordo com o esperado em cada ciclo de execução do E1.

- **Apresentação de valores:** o STU_E1 deve apresentar os valores de todas as variáveis globais envolvidas no sistema ao usuário físico de maneira clara e consistente.

O MTU_E1 deve, similarmente a todos os outros casos, escalonar o envio e recebimento de dados entre o E1 e o STU_E1. Este escalonamento funciona por meio das seguintes funções: indicar o valor das variáveis globais ao STU_E1, recebidos pelo E1; fazer a chamada do E1 para executar os serviços atrelados a ele; e fazer a chamada do STU_E1 para apresentar os valores dos parâmetros de cada iteração.

4.3.6 Teste de sistema

Por fim, esta subseção apresenta as informações necessárias para a execução do teste de sistema. Este teste foi executado após a aprovação do uso direto do E1, ou seja, após todos os testes apresentados neste capítulo serem executados e aprovados. O teste de sistema do AD01 tem como objetivo a análise do sistema como uma entidade única, contendo todos os componentes de hardware e software para que seja possível observar o comportamento do sistema completo. Com isso, o AD01 completo foi testado para observar diferentes elementos e comportamentos. Os seguintes métodos foram levantados para as execuções do teste de sistema:

- **Checagem da câmera:** verificar o funcionamento da câmera e dos LEDs que iluminam o trajeto do AGV, para que possíveis erros possam ser encontrados, como reflexos de iluminação da linha guia, comprometendo o processamento de imagem; e o alinhamento incorreto do veículo com a linha;
- **Checagem das baterias:** pelo veículo possuir baterias muito velhas, a atuação delas nas etapas de uso e carregamento foi necessária, para analisar se a troca das baterias usadas por modelos mais novos deveria ser feita;
- **Fixação da fonte de alimentação externa:** caso testes sejam feitos com alimentação por fonte externa, foi preciso certificar que ela esteja fixada em um local em que o cabo de alimentação, por todo o tempo do teste, não estique e cause danos no veículo ou na própria fonte;

- **Execução do teste unitário do Software Escalonador:** para fazer o teste de sistema, o teste unitário do E1 foi usado para que os dados de entrada sejam inseridos antes do início do movimento do AGV;
- **Sintonização do controlador PID do AGV:** o veículo foi testado com diferentes valores de ganho proporcional (***kp***), ganho integrativo (***ki***), e ganho derivativo (***kd***) para que fosse possível encontrar uma rotina de controle de movimento do veículo em que não haja uma quantidade indesejada de oscilações do veículo na linha ao se mover. As pessoas responsáveis pelo teste manipularam o controlador PID pelo E1, alterando valores das variáveis globais de ***kp***, o termo integrativo (***ti***) e o termo derivativo (***td***) para encontrar um valor de compensação adequado para um índice menor de erro.
- **Uso do Sistema de Intertravamento (M1) em caso de falhas ou término:** Foi compromisso do usuário físico certificar estar próximo ao veículo por todo o procedimento de um caso de teste, pois é possível que o AGV saia da linha no meio do teste ou no término da linha. Neste caso, seria necessário que o usuário físico apertasse um dos botões de parada do AGV para que o intertravamento do veículo fosse feito.

Para validar o funcionamento do AD01, foi necessário tratar diferentes ambientes de execução. É importante apontar que as definições de cada ambiente foram documentadas pelo usuário físico em outros documentos, como checklists ou planilhas. Três tipos principais de ambiente foram cogitados para uso, de forma a avaliar o funcionamento do AGV:

- **Linha reta:** um trajeto composto por apenas uma linha reta, podendo variar em diferentes comprimentos;
- **Linha reta com curva:** um trajeto que contém duas linhas retas conectadas a uma curva. O raio de curvatura pode variar para observar o comportamento do AGV em diferentes acentuações de curva;
- **Circuito em 8:** um circuito em formato do número 8 para avaliar o comportamento do AGV em curvas de diferentes sentidos e no encontro de dois trajetos perpendiculares.

A exemplo de uso e serviço, foi criada, no documento de demanda de produção do teste de sistema, a lista abaixo ilustrando o procedimento completo. O usuário físico, em todo caso de teste:

1. Move o AGV para colocá-lo no chão;
2. Checa a carga das baterias ou a posição da fonte externa;
3. Liga o AGV em modo de alimentação por baterias ou fonte externa;
4. Posiciona o AGV na linha o mais reto possível no começo do trajeto;
5. Executa o teste unitário do E1 para inserção de parâmetros;
6. Insere os parâmetros desejados para dado caso de teste;
7. Analisa o movimento do AGV próximo a ele em caso de parada;
8. Aperta um dos botões de emergência em caso de parada;
9. Interrompe a execução do software usado no teste;
10. Faz as anotações necessárias em um documento de execução;
11. Reinicia o procedimento a partir do passo 4, exceto todos os casos de teste desejados sejam executados, a carga das baterias acabe ou um novo erro seja detectado para correção.

4.4 Resultados obtidos

Nesta seção, os resultados encontrados dos testes executados serão discutidos. Antes da discussão em si, é importante apontar que alguns dos testes feitos compartilham a mesma estrutura de resultados, e isso se dá pela estrutura de teste ser bem semelhante entre alguns casos. Com isso em mente, e para evitar um número elevado de redundâncias, esta seção está dividida em quatro subseções principais. A subseção de resultados gerais apresenta os resultados dos testes unitários com estruturas semelhantes entre si. A subseção de resultados específicos apresenta os testes unitários que tiveram resultados mais pertinentes para uma discussão mais detalhada. As subseções seguintes apresentam os resultados dos testes de integração gerais (incluindo o teste de escalonador), e de sistema, respectivamente.

Outro ponto importante a ser lembrado é que após todos os resultados alcançados para cada teste, a abordagem mestra de testes apresentada no Capítulo 3 continuou a ser seguida

para o procedimento de documentação em todos os casos. Logo, todos os testes tiveram seus documentos de execução escritos e passados pelo processo de aprovação.

4.4.1 Resultados gerais

Nesta subseção, os testes unitários dos serviços S1, S4, S6, S7, S8, S9, S10 e S11 serão discutidos. Como apontado no parágrafo anterior, é possível resumir os resultados alcançados dos testes de todos estes serviços devido à estrutura que ambos compartilham, sendo esta composta pelo envio de entradas ao serviço por meio do Serviço de Teste Unitário (STU) e a observação da saída obtida para checar a compatibilidade com a saída desejada para um determinado caso de teste.

É possível confirmar que os testes executados para estes serviços seguiram todos os protocolos incluídos na abordagem mestra de testes apresentada no Capítulo 3, e todas as execuções obtiveram sucesso. É importante lembrar que, quando a palavra “sucesso” é usada neste aspecto, significa que se um determinado serviço possuía erros de composição (MYERS, SANDLER, BADGETT, 2012), como escrita de código ou erros de lógica computacional, tais erros foram encontrados e corrigidos, de forma a fazer com que todos os casos de teste de cada serviço fossem executados e validados após as correções.

4.4.2 Resultados específicos

Nesta subseção, será feita a discussão dos resultados obtidos nos testes unitários dos serviços S2, S3 e S5. Estes serviços são os mais importantes para o bom funcionamento do AGV AD01, e por isso, apresentaram resultados igualmente importantes. Nos parágrafos seguintes, os resultados de cada um dos três serviços serão detalhados.

Os resultados do Serviço de Comunicação com Beckhoff (S2) não apresentaram valores explícitos, pois neste caso, a lógica de comunicação com a Beckhoff foi feita de forma que o teste foi realizado em método de caixa preta. Ou seja, uma vez que as declarações são feitas no serviço em si, e não por meio do testador, a única observação possível foi a movimentação das rodas dada a ordem de execução do movimento.

Os resultados do Serviço de Compensador PID (S3) apresentaram diversos valores sequenciais para cada caso de teste, e de forma a interpretar todos os valores de uma forma mais coerente, eles foram usados para plotagem de gráficos de comportamento do sinal. Cada caso de teste teve seu próprio gráfico criado, de forma a compreender o comportamento do compensador PID do AGV AD01 de uma maneira mais clara e objetiva, e também com o fim de interpretar os resultados com conhecimentos de Controle para chegar a uma conclusão.

Os resultados encontrados no Serviço de Cálculo de Erros (S5) foram interessantes em uma primeira execução, pois o teste havia sido realizado de forma a observar se os cálculos de erro estavam sendo feitos corretamente. Por mais que o cálculo de erro em questão tenha apresentado um sucesso, o processamento de imagem ainda estava encontrando falhas, ou seja, o cálculo ocorria da maneira certa com arquivos de imagem indesejados. Uma melhor discussão para a solução do problema encontrado pode ser lida na seção de análise dos resultados (Seção 4.5), pois esta solução está bastante atrelada à execução do teste de sistema, usado para analisar os resultados de todos os testes de forma conjunta.

4.4.3 Resultados dos testes de integração de serviços

Cada teste de integração de serviços obteve seu resultado de maneira composta, ou seja, dependia dos resultados obtidos de cada serviço envolvido de maneira separada para que os resultados da integração fossem válidos. Por conta disso, nesta subseção, discutiremos os resultados dos três testes de integração de maneira geral.

Todos os testes de integração feitos neste projeto obtiveram resultados considerados desejados, uma vez que os serviços de software individuais já haviam sido testados e aprovados. Os erros encontrados na integração, neste caso, se devia a erros de lógica e de programação nos seus respectivos Serviços e Majoritários de Teste. Em testes que possuíam serviços com resultados específicos (S3 e S5), uma análise dos resultados mais profunda foi necessária para averiguar se o teste poderia ser considerado um sucesso ou não. Todas as informações pertinentes apontadas de cada teste de integração, assim como os testes de serviço, foram relatadas em seu respectivo documento de execução de testes.

4.4.4 Resultados dos testes de majoritários

Os testes de majoritários obtiveram resultados semelhantes aos testes de integração de serviços discutidos na subseção anterior. Isso se dá ao fato de que um componente majoritário se comporta de maneira semelhante a um Majoritário de Teste de integração, ou seja, de certa forma, a capacidade de escalonar as chamadas dos serviços conectados ao majoritário da maneira correta é o verdadeiro elemento a ser testado nestes casos. Após cada teste de integração, todos os detalhes de teste para seu majoritário específico foram relatados no documento de execução de teste, assim como em todos os outros testes anteriormente concluídos.

Dos três componentes majoritários testados neste projeto, é válido ressaltar os resultados encontrados no teste do Sistema de Controle de Movimento (M2). Foi importante observar o comportamento do M2 pois foi desejado que este majoritário funcionasse como uma malha de controle para o AGV AD01, justamente por compor todos os elementos de uma malha por meio dos serviços conectados a ele. Os resultados alcançados para este majoritário também foram satisfatórios, por todos os elementos conseguirem trabalhar em conjunto de maneira independente, mas também usando as saídas compartilhadas entre si.

4.4.5 Resultados do teste de integração de majoritários

Os resultados encontrados nos testes de integração de majoritários também seguiram o mesmo raciocínio dos testes de integração de serviços, justamente pelo fato de que a estrutura de ambos tipos de teste são bastante semelhantes, com a única diferença sendo o componente o qual a integração está sendo testada.

Assim como nos testes de integração de serviços, os resultados encontrados na integração entre Sistema de Intertravamento (M1) e Sistema de Atribuição de Dados (M3) estavam relacionados à forma de como o M1 e M3 faziam o escalonamento entre si, com os serviços conectados a cada um deles. No caso do Software Escalonador (E1), os resultados encontrados podem ser encarados similarmente como um teste de integração de serviços, pois a estrutura dos resultados é compartilhada com o teste de integração direta de majoritários, sendo que o E1 faz a integração de todos os componentes majoritários envolvidos no sistema completo. Por fim, os resultados encontrados em ambos os casos foram satisfatórios. Foi

possível observar o comportamento de cada componente majoritário trabalhando com as saídas compartilhadas entre si de forma organizada.

4.4.6 Resultados do teste de sistema

O teste de sistema apresentou uma estrutura mais robusta nos seus resultados. Para que resultados reais fossem interpretados, e para que fosse possível encontrar uma sintonia de controle do veículo correta, diversos casos de teste foram elaborados, executados e anotados, de forma que fosse possível interpretar o funcionamento do sistema completo da melhor forma possível.

Diferente dos outros testes, o teste de sistema não obteve resultados para todos os casos propostos, pois não foi possível desenhar todos os tipos de pista para teste do AD01, descritos na Seção 4.3.6. O circuito em 8 não foi desenhado no espaço dedicado aos testes por falta de espaço suficiente no laboratório Tear, local o qual o teste de sistema foi executado. Portanto, apenas os testes em pista de linha reta e linha reta com curva foram executados para coleta de dados.

O teste de sistema também necessitou de uma interpretação mais profunda de cada resultado individual, para que fosse possível planejar um próximo caso de teste e encontrar a sintonia correta para que o AD01 percorrer o trajeto completo sem sair da linha no chão em algum momento. Os testes foram documentados em uma planilha contendo os valores de sintonia do compensador PID (S3), o tipo de erro a ser usado (erro em distância, erro em ângulo ou ambos), a média de velocidade dos motores, o período de cada execução da malha de controle, o tempo de execução total, o tempo em que o AGV permanece na linha caso saia dela, e a quantidade de oscilação do veículo na linha. Por fim, quatro casos de teste diferentes foram executados: pista reta com erro em distância, pista reta com ambos erros, pista reta e curva com erro em distância, e pista reta e curva com ambos erros.

Os resultados encontrados foram satisfatórios, pois apresentavam o movimento do AGV sendo feito de forma correta. O diferencial usado para encontrar o melhor resultado possível foi o uso de diversas iterações de teste para encontrar a sintonia a qual o AD01 pudesse percorrer as duas pistas diferentes com os mesmos valores embutidos na malha de controle (M2). Para que a malha de controle fosse melhor observada, foi também

implementado, junto à compilação do arquivo executável, um roteiro de execução no sistema operacional do veículo o qual guardava os valores de erro atual e ação de compensador, para que gráficos fossem gerados de forma a auxiliar na interpretação do resultado, e também em quais valores poderiam ser usados na iteração seguinte para alcançar um valor mais ótimo de execução da rotina de controle do AD01.

4.5 Análise dos resultados

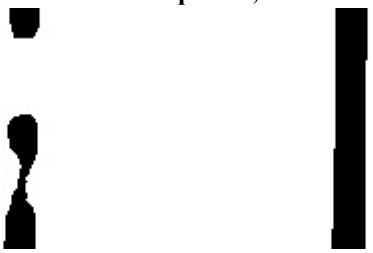
Esta subseção apresenta uma discussão mais detalhada dos resultados obtidos, mas de uma forma que se encaixe em uma discussão mais direcionada ao teste de sistema. Com a apresentação dos resultados e com a conclusão de cada etapa de teste, faz sentido argumentar que todos os testes executados convergem para um único resultado, encontrado no teste de sistema. Portanto, esta análise aborda o processo de obtenção do teste de sistema final, considerando quaisquer obstáculos encontrados neste processo.

Primeiramente, é importante discutir o material usado para a linha em que o AGV deve seguir. A linha da pista foi montada com fita isolante comum, mas o seu uso apresentou problemas que comprometeram o processo de obtenção de resultados. Um deles foi sua alta capacidade de reflexão de luz, que foi um fator contribuinte para a revisão do algoritmo de processamento de imagem no Serviço de Correção de Erros (S5). Pela alta taxa de reflexão, as luzes que iluminam o percurso para que a câmera do AGV capture imagens eram refletidas na fita isolante, causando problemas no algoritmo de processamento de imagem.

Com este problema em mente, é importante retomar os resultados obtidos no S5. Como discutido na seção de resultados específicos, a etapa de processamento de imagem, encontrada dentro do próprio S5, não apresentava uma eficiência satisfatória para o cálculo de erros em si, por mais que o cálculo em questão funcionasse da forma desejada para o sistema. O algoritmo de processamento de imagem não resultava em uma imagem de uma linha completa, como é apresentada na Figura 4.6, de forma que o sistema pudesse trabalhar com o movimento do AD01 de forma correta em um período constante, e este erro também é dado pela alta reflexão de luz causada pelo material encontrado em fitas isolantes comuns. Este erro foi encontrado e discutido quando erros de movimentação do AGV foram encontrados no

teste de sistema, e esta discussão gerou uma reformulação no algoritmo de processamento de imagem encontrado no S5, de forma a apresentar imagens mais limpas para o cálculo de erro de movimento do AD01, como apresentado na Figura 4.7.

4.6 e 4.7 - Imagens monocromáticas de um mesmo trecho de linha da pista no chão, com linha deformada na esquerda, e linha corrigida na direita



Outro processo importante para obter os resultados finais do teste de sistema foi o encontro da sintonia do compensador PID (S3). Por se tratar de um sistema que necessita de atualizações constantes de posição e movimento, o compensador PID deve fazer esta atualização em todas as iterações de movimento do AGV dado um período (***t***) via usuário físico. Logo, para que as atualizações de movimento sejam feitas da melhor forma pelo PID, é importante que suas entradas, que são constantes, tenham valores capazes de executar a rotina de controle do AD01 de forma eficiente, ou seja, manter o veículo na linha da pista com baixa oscilação em todo o percurso (RODRIGUES, 2025). Os valores obtidos nos resultados finais foram alcançados após sessenta e cinco execuções documentadas no total, sendo que cada iteração do teste envolvia um valor diferente para as entradas do S3, fixando o maior número possível de parâmetros que devem ser atribuídos ao veículo pelo Serviço de Atribuição de Dados (S8).

Também é importante discutir que as preparações feitas antes da execução de cada caso de teste de sistema também se mostraram cruciais para o bom desempenho do veículo nos testes. Foi observado que as rodas de apoio do veículo, as quais não possuem motores conectados para oferecer velocidade, precisam estar alinhadas de forma que auxiliem o AD01 a andar para frente ou para trás. Qualquer alteração brusca de eixo nestas rodas comprometeram o caso de teste em que este impasse ocorreu, forçando o reinício do teste.

Outra preparação importante foi a certificação de que a pista em si tivesse atrito suficiente para que as rodas pudessem ser empurradas pelos motores, gerando o movimento do AGV. No momento em que o chão da pista começava a apresentar desgastes, resultando

em uma superfície mais lisa, tal seção da pista era considerada problemática para o teste, e por isso, iniciar os testes em determinada seção foi desconsiderada em testes seguintes. A troca completa do material da pista não foi uma opção, pois o chão do ambiente de testes, o laboratório Tear em si, é uma superfície de piso, sendo trabalhoso remover para substituir por outro material.

4.6 Considerações finais

Este capítulo apresentou o plano de experimento usado para o desenvolvimento do plano de testes para o AGV AD01, veículo autoguiado criado pelo laboratório Tear usando SOA como arquitetura de software. Cada um dos testes realizados foram discutidos de forma a apresentar suas estruturas, os resultados esperados, e os resultados obtidos. Também foi apresentada uma discussão mais detalhada sobre o teste de sistema, passo final para validação do sistema de software completo, e obstáculos encontrados no processo de obtenção do software final para esta versão do AGV. É válido apontar que os resultados obtidos foram satisfatórios, e também alcançaram os objetivos propostos por este trabalho, pelo fato de ter seguido todos os procedimentos e ferramentas apresentadas no Capítulo 3 de forma sequencial e rigorosa.

Capítulo 5

CONCLUSÃO

5.1 Síntese do contexto da proposta e dos objetivos

Este trabalho apresentou uma proposta de um plano de testes voltado a sistemas embarcados desenvolvidos em SOA, com técnicas de HIL para complementar a natureza da execução dos testes. Pela ausência de um plano de testes que possa garantir uma qualidade de vida a diversos projetos encontrados no mercado e no laboratório Tear, é esperado que o plano apresentado neste trabalho possa atender às necessidades identificadas, e também conscientizar a importância de uma etapa de testagem por meio de um plano de testes.

5.2 Contribuições e delimitações

Como apresentada na discussão sobre a situação do laboratório Tear no Capítulo 1, e o detalhamento da proposta no Capítulo 3, a contribuição esperada para este trabalho era a de aplicar o plano de testes no laboratório em seus projetos, enfatizando que, dado o escopo do modelo apresentado, é possível afirmar que o conteúdo apresentado é replicável para sistemas embarcados em geral. Com sua aplicação, foi possível obter a otimização no tempo desejada de desenvolvimento de um projeto que usa SOA como arquitetura de software. Isso se deu por

conta do plano de testes se mostrar como uma ferramenta feita para corrigir os erros, na mesma velocidade em que foram encontrados.

Também é válido ressaltar o discurso mostrado no Capítulo 1, Seção 1.5. Pelo decorrer da aplicação do plano de testes em um dos projetos do laboratório como forma de validação, uma delimitação encontrada é que neste trabalho é que não procuramos fazer uma análise crítica do modelo SOA ou da técnica HIL, e sim respeitar técnicas de desenvolvimento de software e hardware já consolidadas. Não deve-se esperar a criação de uma nova arquitetura para trazer uma proposta diferente, mas sim usar ferramentas já conhecidas como uma forma de adaptação para um modelo que atenda às necessidades encontradas no laboratório, e por consequência, projetos desenvolvidos em SOA.

5.3 Trabalhos futuros

Com a validação do plano de testes apresentado neste trabalho, o próximo passo é que o plano seja replicado para outros projetos. Um trabalho futuro a ser considerado é o uso do plano de testes em uma equipe separada das partes envolvidas na obtenção deste trabalho, para que seja possível observar a otimização do desenvolvimento de um projeto de forma menos enviesada. É esperado que este plano de testes seja usado no laboratório Tear em breve, em dois produtos diferentes, de forma a otimizar o tempo de desenvolvimento de ambos projetos.

Outro trabalho futuro a ser considerado é o estudo de implementação do plano de testes de forma que seja cabível a sistemas críticos, de forma a colocar maior ênfase em ambientes controlados ou simulados. A mesma consideração de estudo pode ser feita para uso do plano em trabalhos de simulação, podendo se aproximar da técnica *hardware-in-the-loop*, observada no Capítulo 2, mas de forma a abordar simulações com SOA envolvido, e não apenas a simulação integrada.

REFERÊNCIAS

AVENOSO, Antonio, *et al*. **CARTA aberta sobre os Cybertrucks na União Europeia.** *Transport & Environment*, [*et al*]. Endereçado ao Ministro dos Transportes da Chéquia. 7 de outubro de 2024. Disponível em: <https://www.transportenvironment.org/uploads/files/2024-10-07-Cybertrucks-IVA-letter-to-Minister-Kupka-1.pdf>. Último acesso em: 12 ago. 2025.

AZIZ, M. W.; ULLAH, N.; RASHID, M. A process model for service-oriented development of embedded software systems. **IT Professional**, v. 23, n. 5, p. 44-49, set./out. 2021. DOI: 10.1109/MITP.2021.3057164.

BALL, Stuart R. **Embedded Microprocessor Systems: Real World Design.** 3. ed. Boston, Massachussets: Newnes, 2002. ISBN 0-7506-75349.

ERL, Thomas. **SOA: Principles of Service Design**. 1. ed. New Jersey: Prentice Hall, Pearson Education, Inc., 2008. ISBN 0-13-234482-3.

GANSSLE, Jack. **The art of designing embedded systems**. 2. ed. Amsterdam: Newnes, 2008. ISBN 978-0-7506-8644-0.

GUEDES, Gilleanes T. A. **UML 2: uma abordagem prática**. 3. ed. São Paulo: Novatec Editora, 2018. ISBN 978-85-7522-646-9.

HAORONGBAM, L.; NAGPAL, R.; SEHGAL, R. Service Oriented Architecture (SOA): A literature review on the maintainability, approaches and design process. **2022 12th International Conference on Cloud Computing, Data Science & Engineering (Confluence)**, Noida, India, 2022. p. 647-652. DOI: 10.1109/Confluence52989.2022.9734153.

HOMÈS, Bernard. **Fundamentals of Software Testing**. 2. ed. London: ISTE Ltd, John Wiley & Sons, Inc., 2024. ISBN 978-1-78630-982-2

IEEE. IEEE Standard for Software and System Test Documentation. **IEEE Std 829-2008**, 18 July 2008. p. 1-150. DOI: 10.1109/IEEESTD.2008.4578383.

IGNARRA, P, *et al*., A workflow for automated testing of a safety critical embedded subsystem, **Proceedings of the IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)**. Naples, Italy, 2025, p. 149–156. DOI: 10.1109/ICSTW64639.2025.10962490.

ISENI, P.; HALILI, F. Reliable service-oriented architecture for NASA's Mars exploration rover mission. **2022 11th Mediterranean Conference on Embedded Computing (MECO)**, Budva, Montenegro, 2022. p. 1-5. DOI: 10.1109/MECO55406.2022.9797097.

JORGENSEN, Paul C.; DEVRIES, Byron. **Software Testing: A Craftsman's Approach**. 5 ed. Boca Raton: CRC Press, Taylor & Francis Group, LLC, 2021. ISBN 978-0-367-35849-5.

JOSUTTIS, Nicolai M. **SOA in Practice**. 1 ed. Sebastopol, Florida: O'Reilly Media, Inc, 2007. ISBN 0-596-52955-4.

KRISHNAN, R.; MANJU, C. R.; JAYALEKSHMY, L. Testing of real-time embedded onboard software: a launch vehicle case study. **International Conference on Computing Communication and Networking Technologies (ICCCNT)**, 14., 2023, Delhi, India. Delhi: IEEE, 2023. p. 1-7. DOI: 10.1109/ICCCNT56998.2023.10307847.

LEINFELDER, Andrea. NASA astronauts splash down off Florida coast, ending 9-month saga. *Houston Chronicle*, 18 mar. 2025. Disponível em: https://www.houstonchronicle.com/news/houston-texas/space/article/astronauts-returning-to-earth-nasa-spacex-starline-20227751.php. Acesso em: 29 ago. 2025.

MYERS, Glenford J.; SANDLER, Corey; BADGETT, Tom. **The art of software testing**. 3rd ed. Hoboken: Wiley, 2012.

PRESSMAN, Roger S.; MAXIM, Bruce R. **Software engineering: a practitioner's approach**. 9. ed. New York: McGraw-Hill Higher Education, 2019. ISBN 978-1-259-87297-6.

RODRIGUES, Luan D. **Plataforma de desenvolvimento de sistemas de controle: um estudo de caso aplicado em um AGV**. 2025. 61 f. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de São Carlos, São Carlos, 2025.

SAINI, Dinesh K. Software testing for embedded systems. **International Journal of Computer Applications**, v. 43, p. 1-6, abr. 2012. DOI: 10.5120/6192-8700.

SOUZA, Heitor P. **Plano de testes para Arquitetura Orientada a Serviços em Veículos Autoguiados**. 79 f. Trabalho de Conclusão de Curso (Graduação em Engenharia da Computação) – Universidade Federal de São Carlos, São Carlos, 2023.

SOUZA, Luís Fernando F. **Arquitetura de Software Orientada a Serviços para AGV**. 80 f. Trabalho de Conclusão de Curso (Graduação em Engenharia da Computação) – Universidade Federal de São Carlos, São Carlos, 2021.

TOMIMORI, H.; OYAMA, H.; AZUMI, T. Automated testing framework for embedded component systems. **IEEE 26th International Symposium On Real-Time Distributed**

Computing (ISORC), 2023, Nashville, TN, USA. Nashville: IEEE, 2023. p. 176-183. DOI: 10.1109/ISORC58943.2023.00032.

ULLRICH, G. **Automated Guided Vehicle Systems**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. ISBN 978-3-662-44813-7 978-3-662-44814-4. Disponível em: <http://link.springer.com/10.1007/978-3-662-44814-4>.

ZHANG, B.; YI, G.; WANG, Y.; FEI, Q. Research on generation algorithm of SOA-oriented integration test order. **IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)**, 2021, Hainan, China. Hainan: IEEE, 2021. p. 107-116. DOI: 10.1109/QRS-C55045.2021.00025.

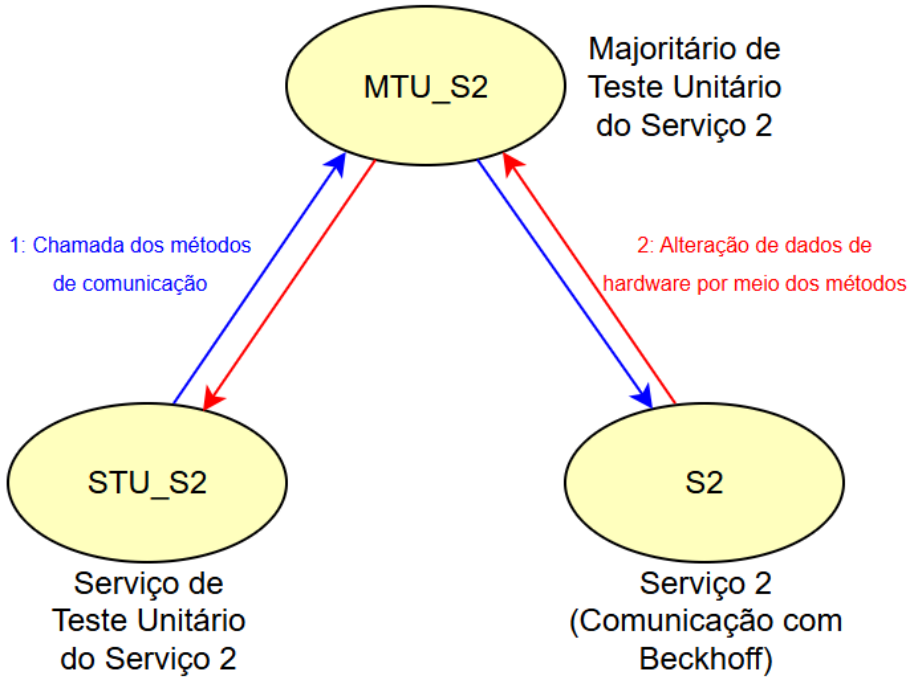
ZHANG, Y.-N.; AN, M.-Q. A compact and reliable software architecture for robotics control. **IEEE International Conference on Information Communication and Software Engineering (ICICSE)**, 2021, Chengdu, China. Chengdu: IEEE, 2021. p. 41-46. DOI: 10.1109/ICICSE52190.2021.9404085.

Apêndice A

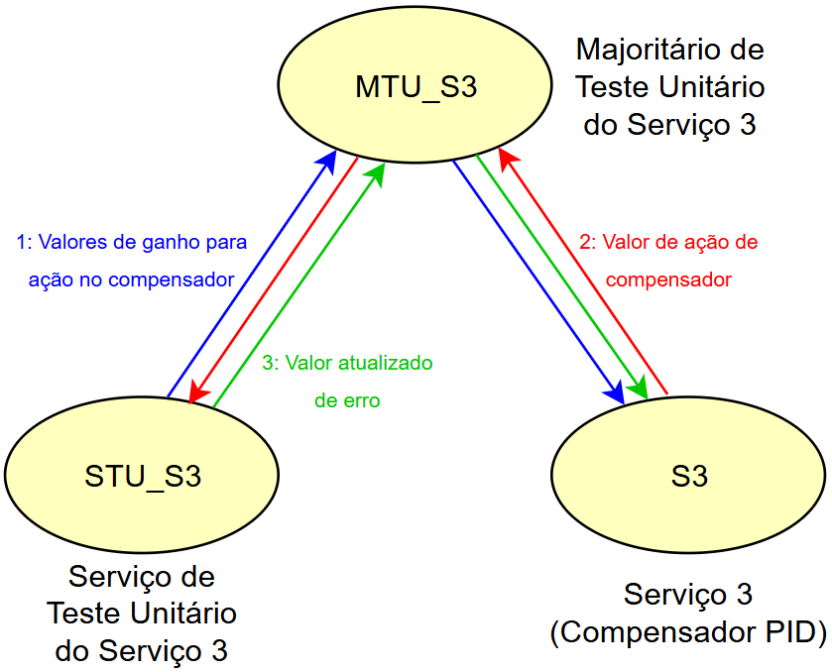
DIAGRAMAS DE CAMINHO DE DADOS

Este apêndice contém todos os diagramas de caminho de dados dos testes introduzidos na Seção 4.3 do Capítulo 4, apresentando as figuras dos diagramas que não foram diretamente referenciados neste trabalho. Os diagramas de caminho de dados dos testes unitários são apresentados das Figuras A.1 a A.10. Os diagramas de caminho dos testes de integração de serviços são apresentados nas Figuras A.11 e A.12. Finalmente, os diagramas de caminho dos testes de serviço majoritário são apresentados nas Figuras A.13 e A.14.

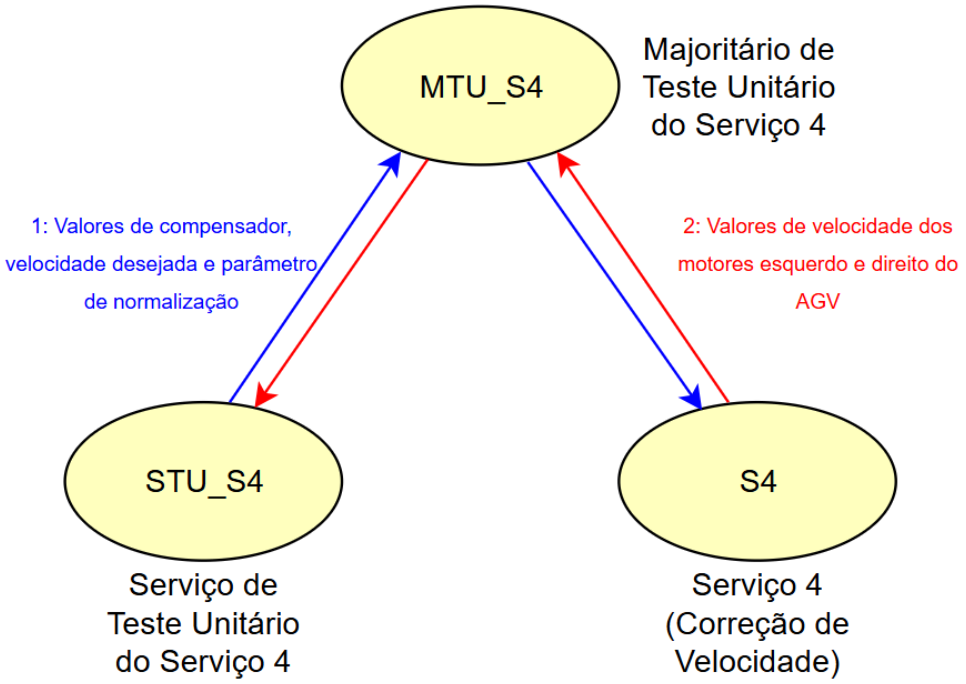
A.1 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Comunicação com Beckhoff (S2)



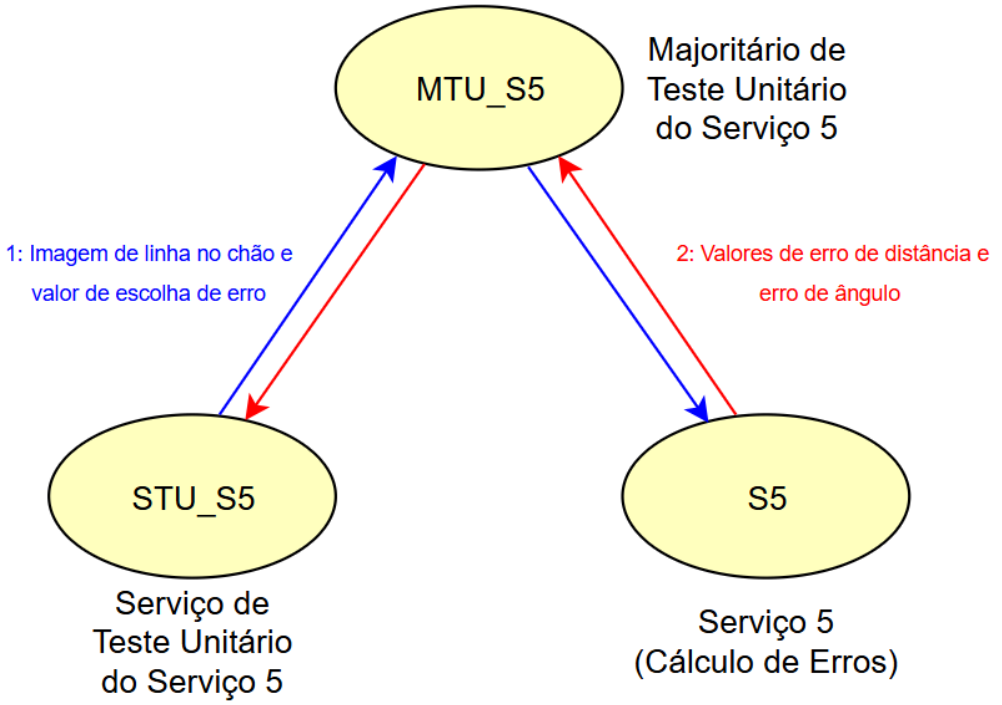
A.2 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Compensador PID (S3)



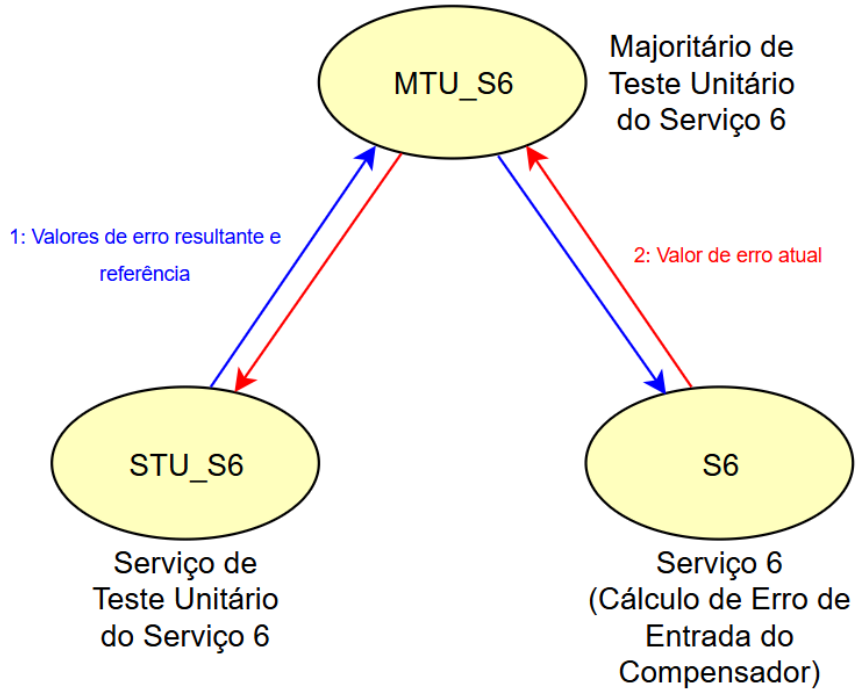
A.3 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Correção de Velocidade (S4)



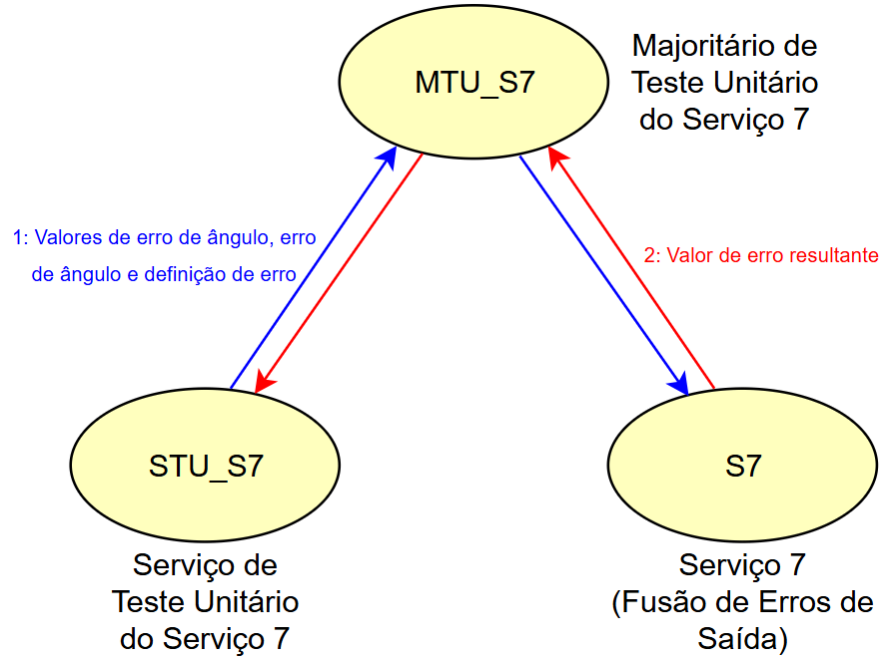
A.4 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Cálculo de Erros (S5)



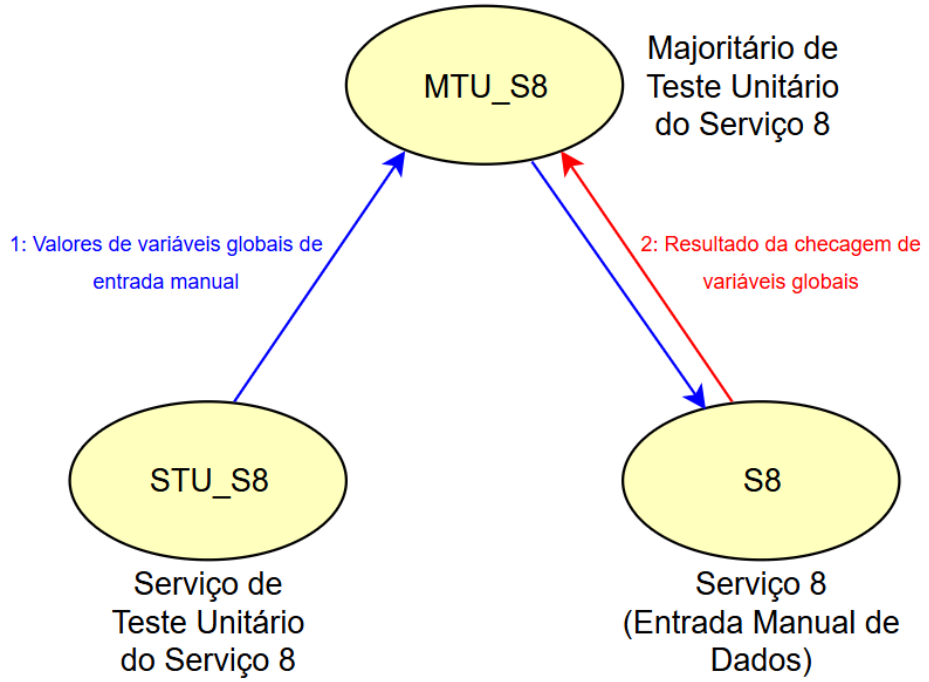
A.5 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Cálculo de Erro de Entrada do Compensador (S6)



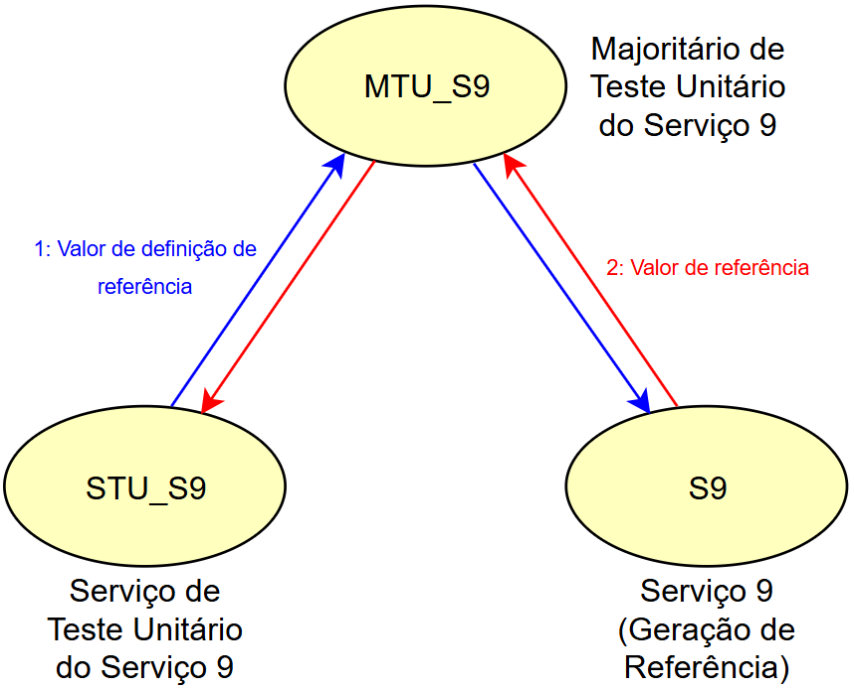
A.6 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Fusão de Erros de Saída (S7)



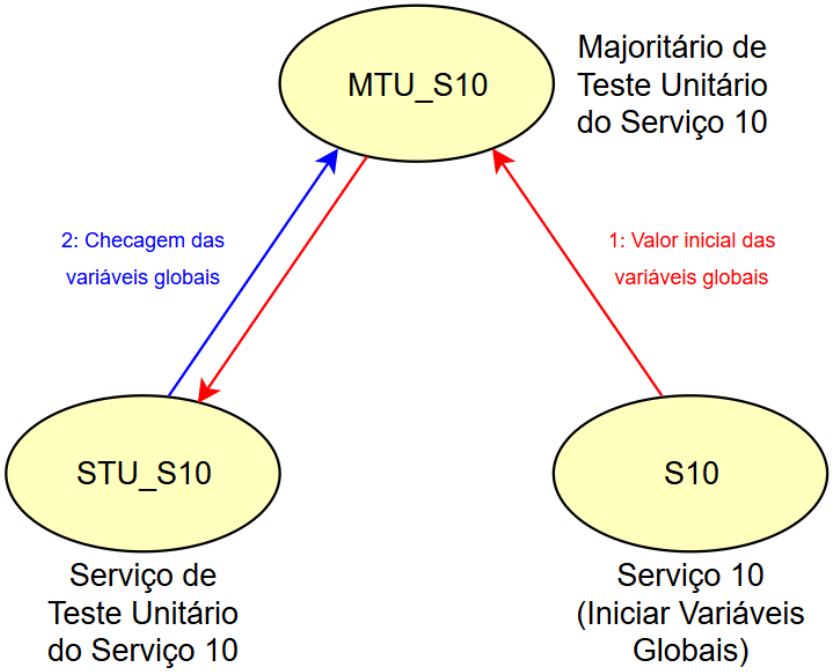
A.7 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Entrada Manual de Dados (S8)



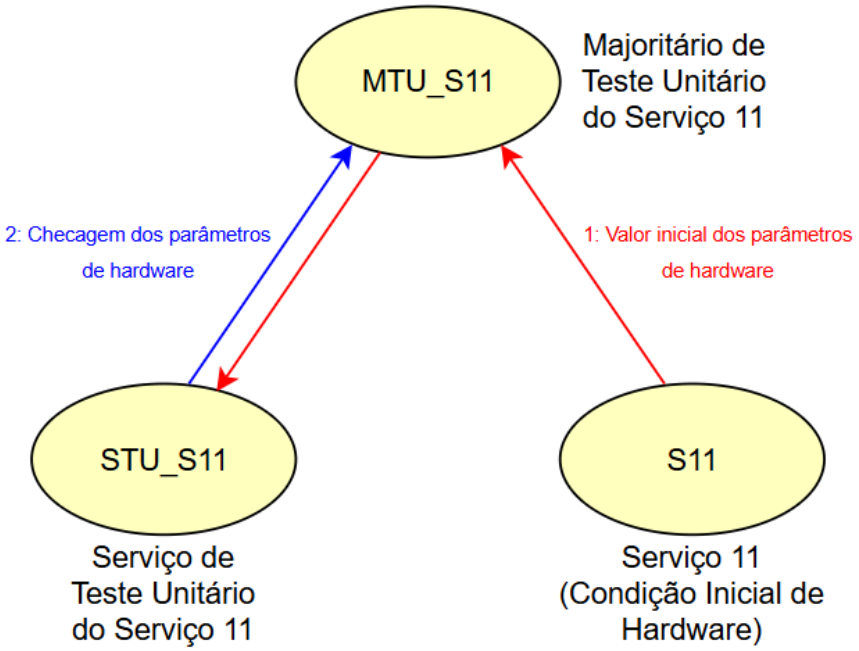
A.8 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Geração de Referência (S9)



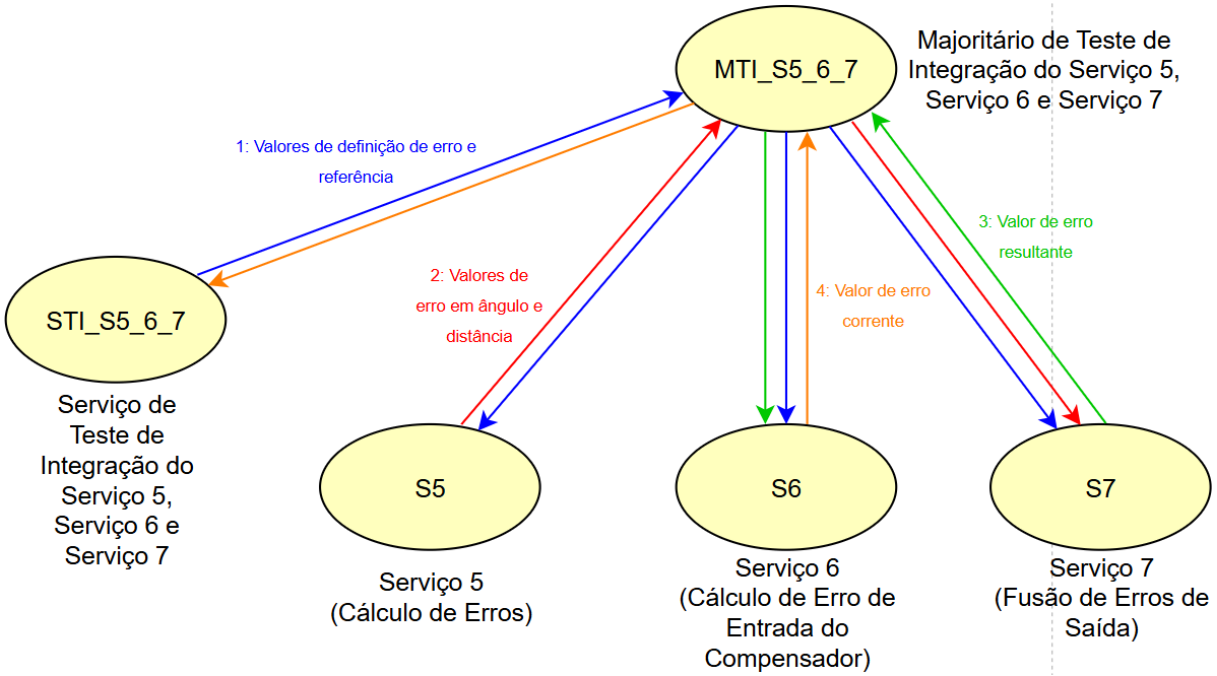
A.9 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Iniciar Variáveis Globais (S10)



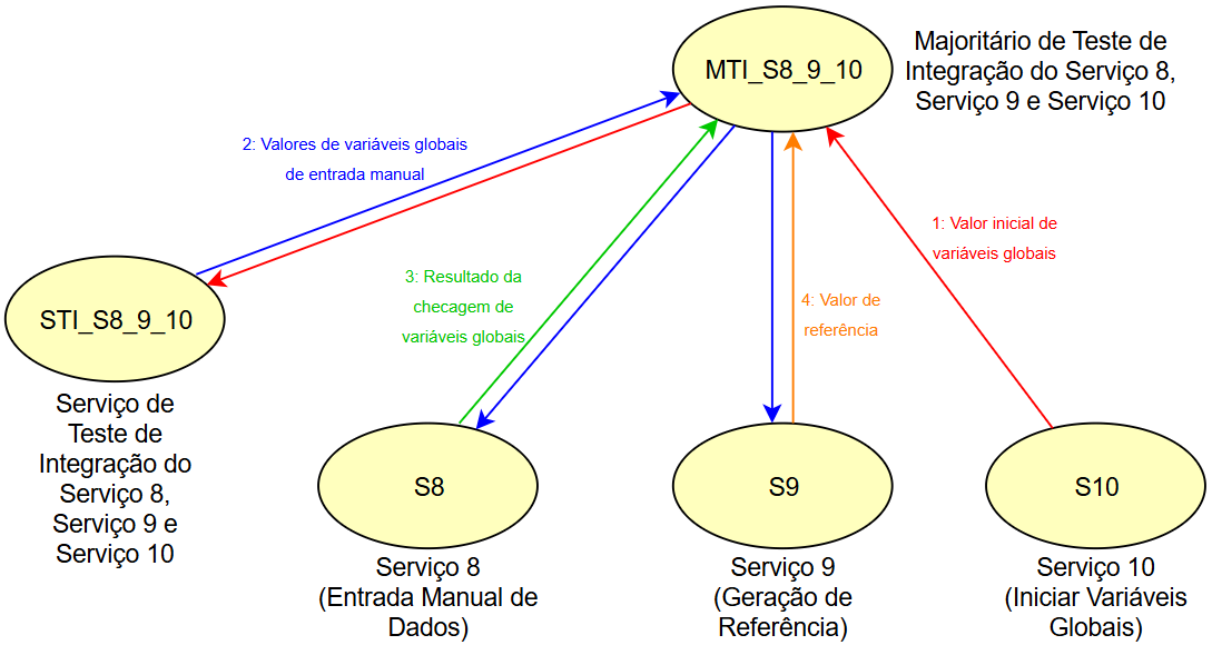
A.10 - Diagrama de caminho de dados dos serviços envolvidos no teste do Serviço de Condição Inicial de Hardware (S11)



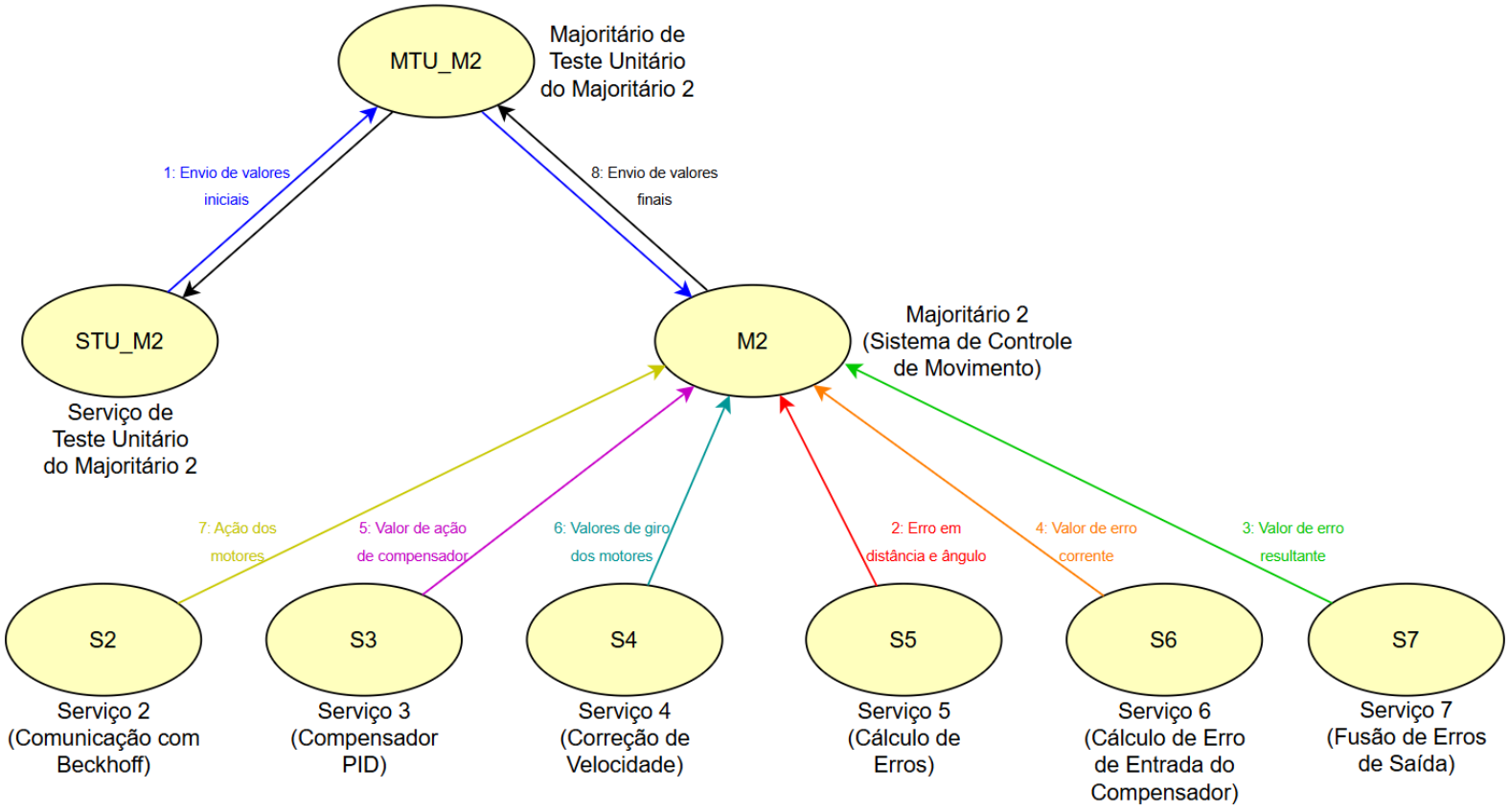
A.11 - Diagrama de caminho de dados dos serviços envolvidos no teste de integração dos Serviços 5, 6 e 7



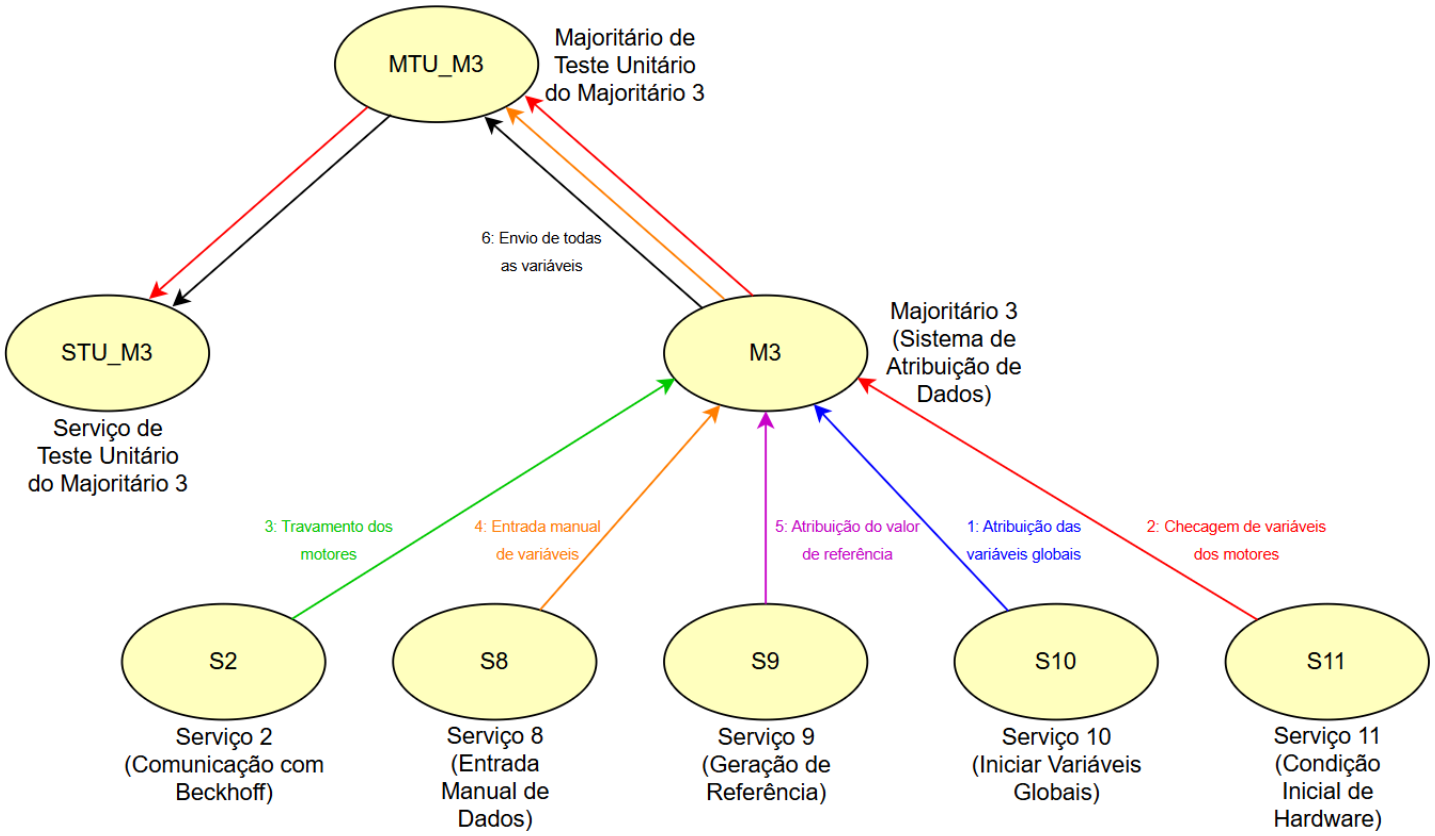
A.12 - Diagrama de caminho de dados dos serviços envolvidos no teste de integração dos Serviços 8, 9 e 10



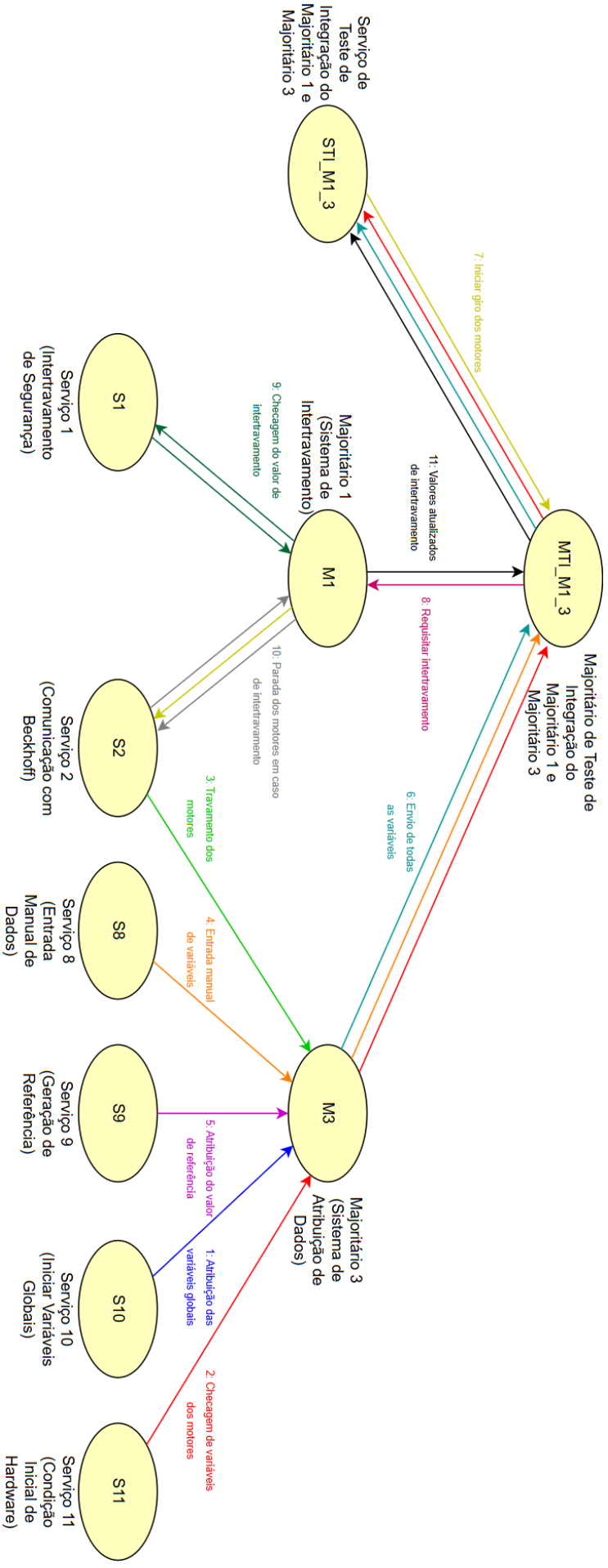
A.13 - Diagrama de caminho de dados dos serviços envolvidos no teste do M2



A.14 - Diagrama de caminho de dados dos serviços envolvidos no teste do M3



A.15 - Diagrama de caminho de dados dos serviços envolvidos no teste de integração dos majoritários 1 e 3



Anexo A

Documentação do teste do Serviço de Intertravamento de Segurança

Este anexo apresenta todos os documentos feitos para documentação e implementação do teste para o Serviço de Intertravamento de Segurança (S1). Os documentos foram inseridos neste anexo de forma a exemplificar os conceitos discutidos na Seção 4.3 do Capítulo 4 deste trabalho. Este anexo contém a documentação de ambos Serviço de Teste (STU_S1) e Majoritário de Teste (MTU_S1), seguindo a lógica encontrada no diagrama de caminho de dados do teste em questão (Figura 4.2).

Anexo A.1: Documento de requisitos do STU_S1, páginas 1 e 2

Documento de Requisitos

Serviço de Teste Unitário de Intertravamento de Segurança (STU_S1)

Descrição Geral

Funções do Produto

O STU_S1 terá três funções principais:

- **Envio de valor:** o testador atualiza o valor do parâmetro global **interlockButton** por meio do Majoritário de Teste Unitário do Serviço de Intertravamento de Segurança **(MTU_S1)**, para que o Serviço de Intertravamento de Segurança **(S1)** possa executar sua função;
- **Recebimento de valor atualizado:** o STU_S1 recebe o valor do parâmetro **motorEnable** e checa, em um caso de teste, se o valor recebido era o esperado.
- **Apresentação dos resultados:** após calcular todos os casos de teste, o STU_S1 deve apresentar os valores calculados pelo S1 junto aos valores esperados para cada.

Características do Usuário

- Este serviço é consumido pelo MTU_S1, o qual se comunica com este Serviço de Teste e o S1.
- O MTU_S1 deve manter as operações do STU_S1 em coordenação com as operações do S1 da maneira correta.

Restrições Gerais

- O STU_S1 foi criado apenas para uso em testes internos do laboratório Tear, e não pode ser usado em produtos finais que possuam o S1.

Suposições e Dependências

- O S1 deve, obrigatoriamente, estar conectado ao MTU_S1, para que a transmissão de dados e a manipulação de valores possa ocorrer no STU_S1 da forma correta.
- Os valores que o STU_S1 inserir aos devidos parâmetros devem condizer com o caso de teste em questão, e não podem ser alterados após sua concepção, para que os cálculos do STU_S1 e do S1 não sejam feitos para casos que não condizem com os valores a ser calculados.
- Após a execução, o S1 deve atualizar o valor do parâmetro **motorEnable**, enquanto o STU_S1 faz a aplicação do parâmetro no caso de teste em questão. A Figura 1 ilustra o caminho de dados que serão realizados na execução de um caso de teste.

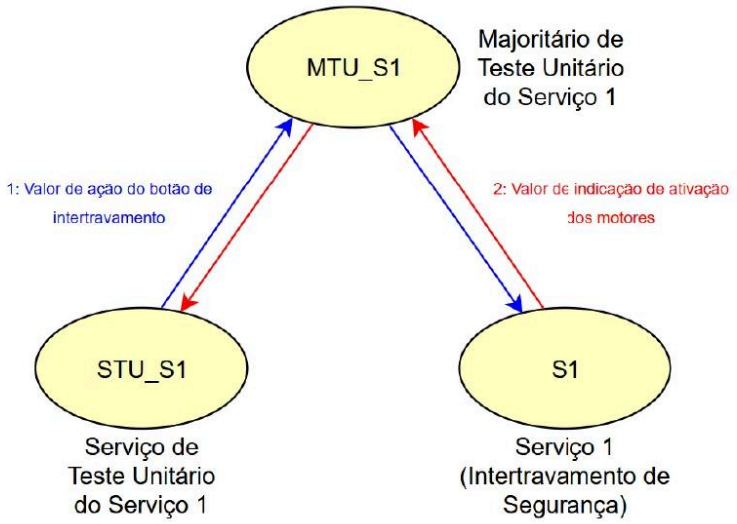


Figura 1: Diagrama de caminho de dados dos serviços envolvidos no teste do S1.

Requisitos Específicos

Requisitos Funcionais

RF1 - O STU_S1 deve conter todos os casos de teste que serão usados no teste geral.

RF2 - Os casos de teste serão valores de ativação ou desativação do botão de intertravamento.

O STU_S1 também deve:

RF3 - Enviar dados correspondentes ao caso de teste corrente.

RF4 - Receber o valor de ação de motores, previamente calculado pelo S1.

RF5 - Avançar para o próximo caso de teste após terminar as operações de um caso de teste anterior.

RF6 - Armazenar cada valor de saída obtido para comparação com os valores esperados.

Requisitos Não Funcionais

- **Desempenho**
  - RNF1 - Executar o STU_S1 em um computador que comporte as operações e possua uma taxa de processamento satisfatória;
- **Escalabilidade**
  - RNF2 - O STU_S1 deve incluir pelo menos 100 casos de teste totais, com o âmbito de "estressar" o S1.
- **Confiabilidade**
  - RNF3 - Apresentar os valores de saída obtidos lado a lado com os valores esperados para que o usuário possa observá-los e comparar visualmente.
- **Manutenibilidade**
  - Não identificado.

Anexo A.2: Documento de demanda de produção do STU_S1, páginas 1 e 2

Documento de Demanda de Produção

Serviço de Teste Unitário de Intertravamento de Segurança (STU_S1)

Descrição de métodos

Tratamento de casos de teste

Múltiplos casos de teste serão inseridos no STU_S1, e cada um será uma estrutura separada do STU_S1, contendo dois valores diferentes: o valor de ação do botão de intertravamento (**interlockButton_Test**), e o valor esperado de ação dos motores (**motorEnable_expected**) para um valor dado de **interlockButton_Test**. O valor de resultado do teste (**testResult**) é localizado no STU_S1. O parâmetro **interlockButton_Test** será usado para atualizar o valor da variável global **interlockButton**, e **motorEnable_expected** será usado internamente para comparação de resultados.

Dados de entrada

A variável global **interlockButton** será atualizada na execução do Majoritário de Teste Unitário **(MTU_S1)**, com o valor dado de **interlockButton_Test** em um dos casos de teste do STU_S1.

Recebimento de saída obtida

O valor de ação dos motores (**motorEnable**), determinado pelo S1, deve ser recebido pelo MTU_S1, o qual recebe o valor do próprio S1.

Comparação de resultados

Após o envio e recebimento de dados entre os serviços, o STU_S1 deve comparar o resultado obtido de **motorEnable** com o resultado esperado (**motorEnable_expected**). Se houver discrepâncias entre os valores, o STU_S1 deve indicar, em **testResult,** que o caso de teste falhou na execução. Caso contrário, o STU_S1 deve indicar, em **testResult**, que o caso de teste obteve sucesso.

Apresentação de resultados

No fim da execução do caso de teste, o STU_S1 deve mostrar o valor obtido de ação dos motores lado a lado com o valor esperado. Ambas as apresentações tem o propósito de permitir ao usuário observar e analisar os resultados do caso de teste por conta própria.

Exemplo de uso e serviço

- Em um caso de teste, temos os valores para cada variável:
  - **interlockButton_Test** = 1;
  - **motorEnable_expected** = 0;

- O STU_S1, ao iniciar este caso de teste, deve atualizar o valor de **interlockButton** com **interlockButton_Test**. O S1 deve receber este parâmetro por meio do MTU_S1 e observar o valor obtido.
- Após o S1 terminar o cálculo, este deve atualizar o valor de **motorEnable**. O STU_S1 deve, então, compará-lo com o valor de **motorEnable_expected**.
- Se o valor esperado de ação dos motores for igual ao valor obtido pelo cálculo feito pelo S1, o caso teste (**testResult**) é dado como sucesso. Caso o valor não seja condizente com o esperado, o caso será dado como falho.

Casos de teste

Apenas dois casos serão necessários neste teste:

- Quando **interlockButton** = 0, checar se **motorEnable** = 1;
- Quando **interlockButton** = 1, checar se **motorEnable** = 0.

Referências

RODRIGUES, Luan D. Documento de demanda de produção - Serviço de Intertravamento de Segurança. Último acesso em 05/02/25.

Histórico de alterações

05/02/25:

- Criação do documento.

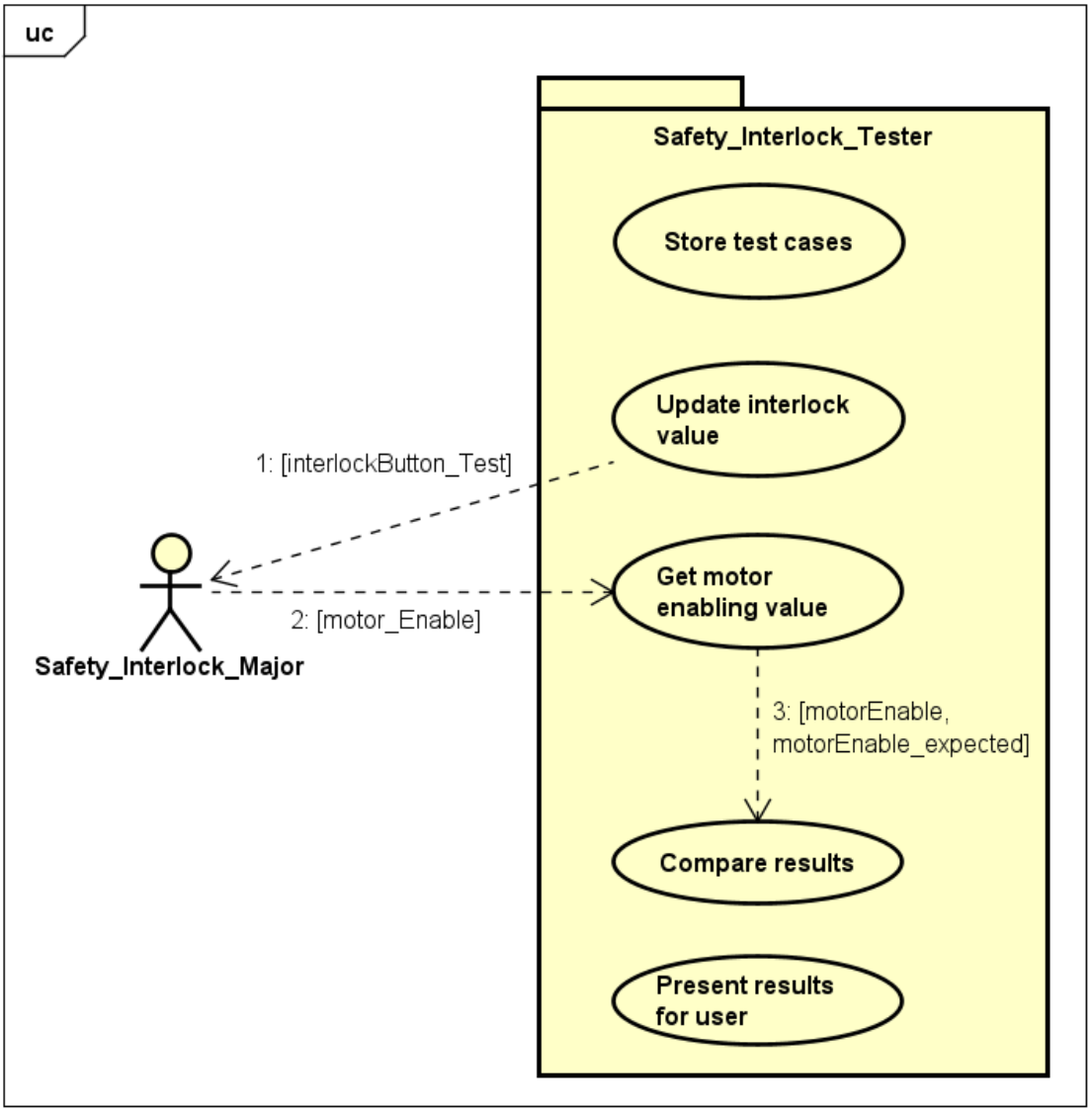
06/02/25:

- Foi detalhado o formato dos casos de teste e seu conteúdo.

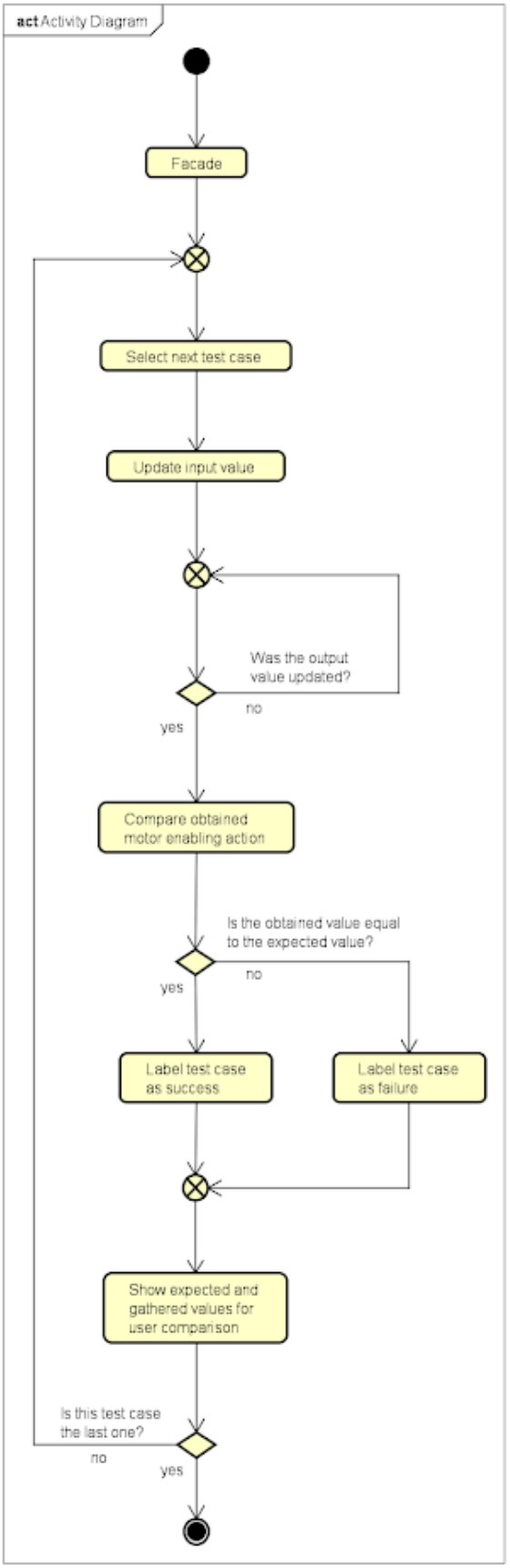
10/02/25:

- Variáveis globais desvinculadas do majoritário.

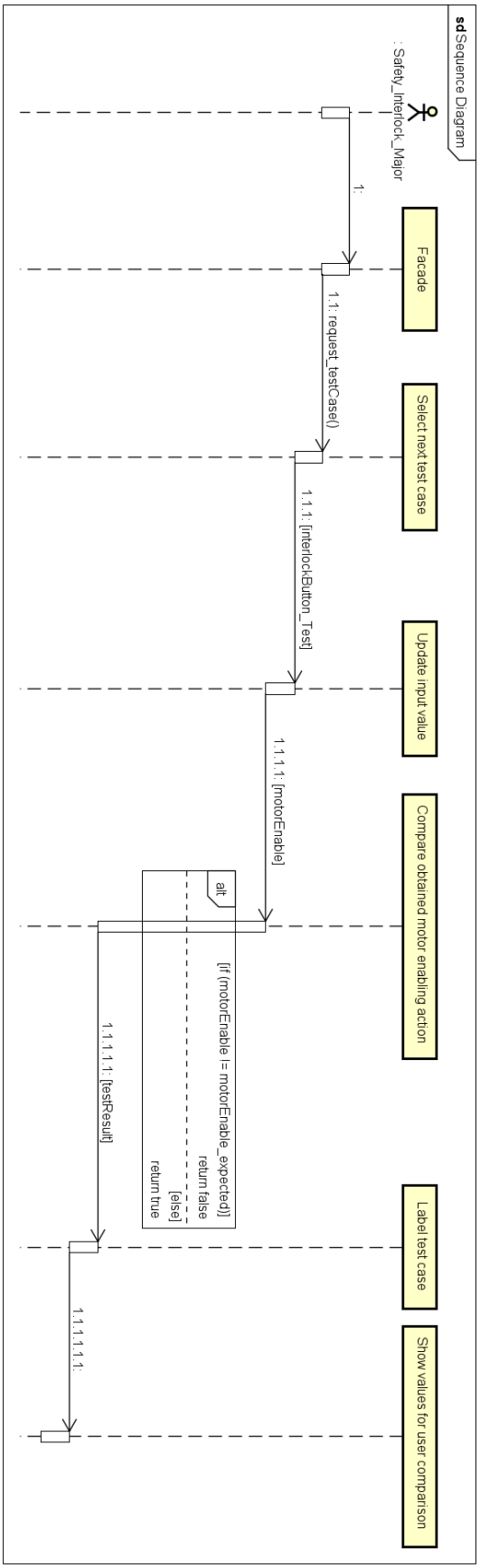
Anexo A.3 - Diagrama de casos de uso do STU_S1



Anexo A.4 - Diagrama de atividades do STU_S1



Anexo A.5 - Diagrama de sequência do STU_S1



Anexo A.6 - Pseudocódigo do STU_S1

Safety_Interlock_Tester

include: iostream
globals.h
define: N/A
variables: Test[ ] // testCase
motorEnable_expected // int
testResult // bool

// First of all, every parameter that lies in "testCase" has to be assigned, so in order to do that, the
// tester must execute all "set" operations necessary for all test cases to be correctly attributed.
method manipulate_cases()
Execute all "set" methods for every parameter found in every "testCase"
end method

method request_testCase(int n)
// Select next test case
interlockButton ← Test[n].get_interlockButton_Test()
motorEnable_expected ← Test[n].get_motorEnable_expected()
end method

method update_parameters(int n)
// Update input value
request_testCase(n)
end method

method compare_values(int n)
// Compare obtained motor enabling action
if (motorEnable != motorEnable_expected)
return false
else
return true
end method

method finalResult()
// Label test case
testResult ← compare_values()
end method

```
method printResult()
        // Show values for user comparison
        print ("Obtained 'motorEnable': ", motorEnable, " | Expected 'motorEnable': ",
        motorEnable_expected, " | ")

        if (testResult == true)
                print ("SUCCESS \n")
        else
                print ("FAILURE \n")
end method
```

testCase

```
include: NA
define: NA
variables:  interlockButton_Test // int
            motorEnable_Test // int

method get_interlockButton_Test()
        return interlockButton_Test
end method


method get_motorEnable_Test()
        return motorEnable_Test
end method


method set_interlockButton_Test(n)
        interlockButton_Test ← n
end method


method set_motorEnable_Test(n)
        motorEnable_Test ← n
end method
```

Facade_Safety_Interlock_Tester

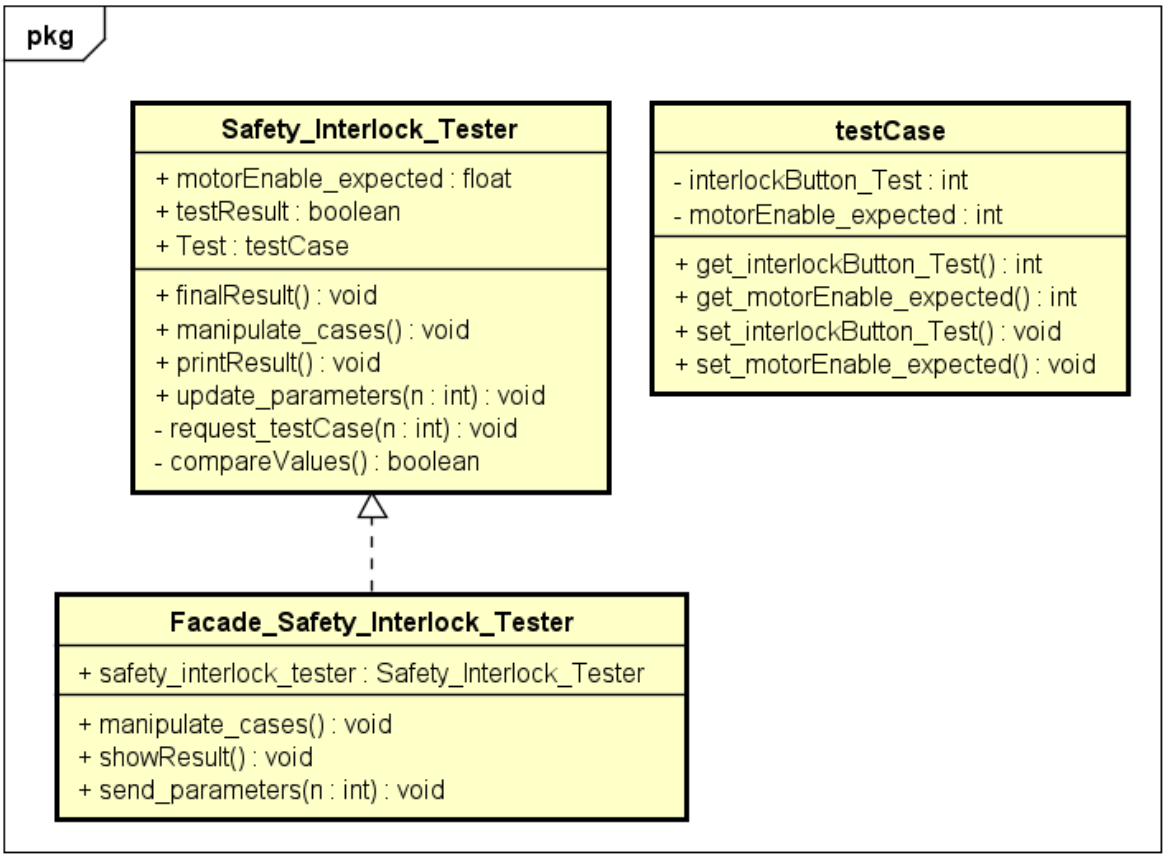
include: NA
define: NA
variable: safety_interlock_tester // Safety_Interlock_Tester instantiation from the
// Safety_Interlock_Tester class

method manipulate_cases()
    safety_interlock_tester.manipulate_cases()
end method

method update_parameters(n)
    safety_interlock_tester.update_parameters(n)
end method

method show_result()
    safety_interlock_tester.finalResult()
    safety_interlock_tester.printResult()
end method

Anexo A.7 - Diagrama de classes do STU_S1



Anexo A.8 - Documento de requisitos do MTU_S1, páginas 1 e 2

Documento de Requisitos

Software Majoritário de Teste Unitário do Serviço de Intertravamento de Segurança (MTU_S1)

Descrição Geral

Funções do Produto

- A função principal do MTU_S1 é de escalonar o envio e recebimento de dados entre o Serviço de Correção de Velocidade **(S1)** e o Serviço de Teste Unitário de Intertravamento de Segurança **(STU_S1)**. Este escalonamento funciona por meio de duas funções principais:
  - Indicar os parâmetros iniciais na chamada do S1, atualizados pelo STU_S1;
  - Indicar o valor de ação dos motores ao STU_S1, atualizados pelo S1.
- O MTU_S1 deve fazer esta troca de dados entre os serviços em todas as iterações de cada caso de teste que o STU_S1 possuir.

Características do Usuário

Não identificado.

Restrições Gerais

- Assim como o STU_S1, este software não deve ser usado no produto final em que o S1 for implementado.

Suposições e Dependências

- Este serviço funciona como um consumidor de ambos S1 e STU_S1, e por isso, é importante que os serviços estejam apropriadamente conectados ao MTU_S1 para que o teste seja executado corretamente. É importante garantir que os serviços mencionados estejam funcionando da maneira esperada antes de sua implementação e uso.
- Os valores que o STU_S1 possui para ação no S1 e vice versa precisam ser coordenados por meio do MTU_S1. Após a execução, o S1 deve atualizar o valor de ação dos motores pelo MTU_S1, o qual o STU_S1 usa para comparação no caso de teste em questão. A Figura 1 ilustra o caminho de dados que serão realizados na execução de um caso de teste.

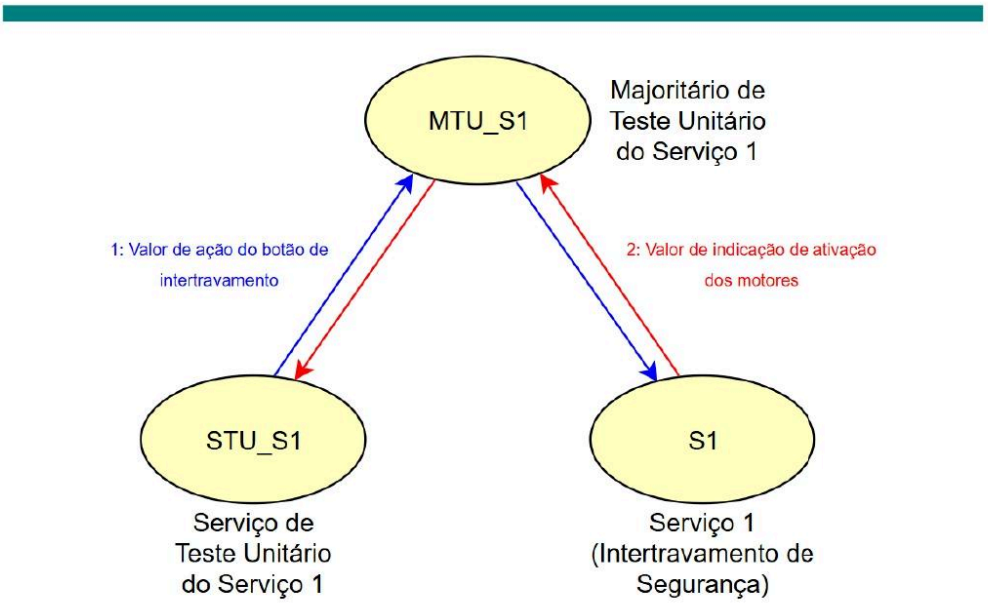


Figura 1: Diagrama de caminho de dados dos serviços envolvidos no teste do S1.

Requisitos Específicos

Requisitos Funcionais

RF1 - O MTU_S1 deve requisitar casos de teste ao STU_S1;

RF2 - O MTU_S1 deve indicar os parâmetros iniciais de correção de velocidade de motores, oriundos do STU_S1, diretamente ao S1;

RF3 - O MTU_S1 deve indicar o valor de ação dos motores (**motorEnable**), calculado pelo S1, diretamente ao STU_S1.

RF4 - Garantir que o escalonamento de tarefas seja feito no número de vezes necessário para cada caso de teste.

Requisitos Não Funcionais

- **Desempenho**
  - Não identificado.
- **Escalabilidade**
  - RNF1 - Executar o MTU_S1 em um computador que comporte as operações e possua uma taxa de processamento satisfatória;
  - RNF2 - Garantir que o STU_S1 e o S1 estejam operacionais antes da execução do STU_S1.
- **Confiabilidade**
  - RNF3 - Indicar que um caso de teste está sendo executado.
- **Manutenibilidade**
  - RNF4 - Certificar que o MTU_S1 esteja enviando e recebendo dados na ordem correta aos dois serviços, sem riscos de deadlocking ou envio de dados fora de ordem.

Anexo A.9 - Documento de demanda de produção do MTU_S1, páginas 1 e 2

Documento de Demanda de Produção

Software Majoritário de Teste Unitário do Serviço de Intertravamento de Segurança (MTU_S1)

Descrição de métodos

Solicitação de dados do Serviço de Teste Unitário de Intertravamento de Segurança (STU_S1)

Em cada caso de teste, o software majoritário deve solicitar, ao STU_S1, o valor de ação do botão de intertravamento (**interlockButton**). Este valor deve ser atualizado para que o Serviço de Intertravamento de Segurança **(S1)** obtenha o valor de ação dos motores, armazenado na variável global **motorEnable**.

Sugestão: ler o Documento de Demanda de Produção do S1 e do STU_S1 para melhor compreensão do uso das variáveis.

Chamada do Serviço de Intertravamento de Segurança (S1)

Ao receber os dados do STU_S1, o MTU_S1 deve fazer a chamada de S1 com estes novos valores, para que os cálculos necessários no serviço sejam feitos, a fim de obter o valor de ação dos motores.

Sugestão: ler o documento de demanda de produção do STU_S1 para aprofundar no funcionamento do testador.

Chamada do Serviço de Teste Unitário de Intertravamento de Segurança

Após o S1 atualizar o valor de **motorEnable**, o MTU_S1 deve indicá-los ao STU_S1, para que este valor seja checado no caso de teste corrente.

Tratamento de casos de teste

Com a execução completa de um dos casos de teste, o MTU_S1 deve iniciar a troca de dados entre serviços novamente caso haja um próximo caso de teste em STU_S1, ou interromper a execução de ambos serviços caso todos os casos de testes tenham sido concluídos.

Exemplo de uso e serviço

1. O MTU_S1 inicia um laço de repetição pelo número de casos de teste existentes no STU_S1, para definir o número de vezes em que a comunicação entre os dois serviços será executada.

1. Na primeira instância do caso de teste, o MTU_S1 coleta o valor da variável **interlockButton** do STU_S1, e faz a chamada do S1 com este valor para calcular **motorEnable**.

2. O MTU_S1 recebe o valor atualizado de **motorEnable** do S1 e faz a chamada do STU_S1.

3. Após a comparação no serviço de testes, o MTU_S1 solicita a impressão do resultado final do caso de teste.

4. Caso o caso de teste aplicado seja o último, o MTU_S1 interrompe ambos os serviços e conclui os testes. Se houver um próximo caso de teste, o MTU_S1 recomeça o ciclo a partir do passo 2, até que todos os casos de teste sejam executados.

Referências

MARTINS, Vinícius F. Documento de demanda de produção - Serviço de Teste Unitário do Serviço de Intertravamento de Segurança. Último acesso em 05/02/25.

RODRIGUES, Luan D. Documento de demanda de produção - Serviço de Intertravamento de Segurança. Último acesso em 05/02/25.

Histórico de alterações

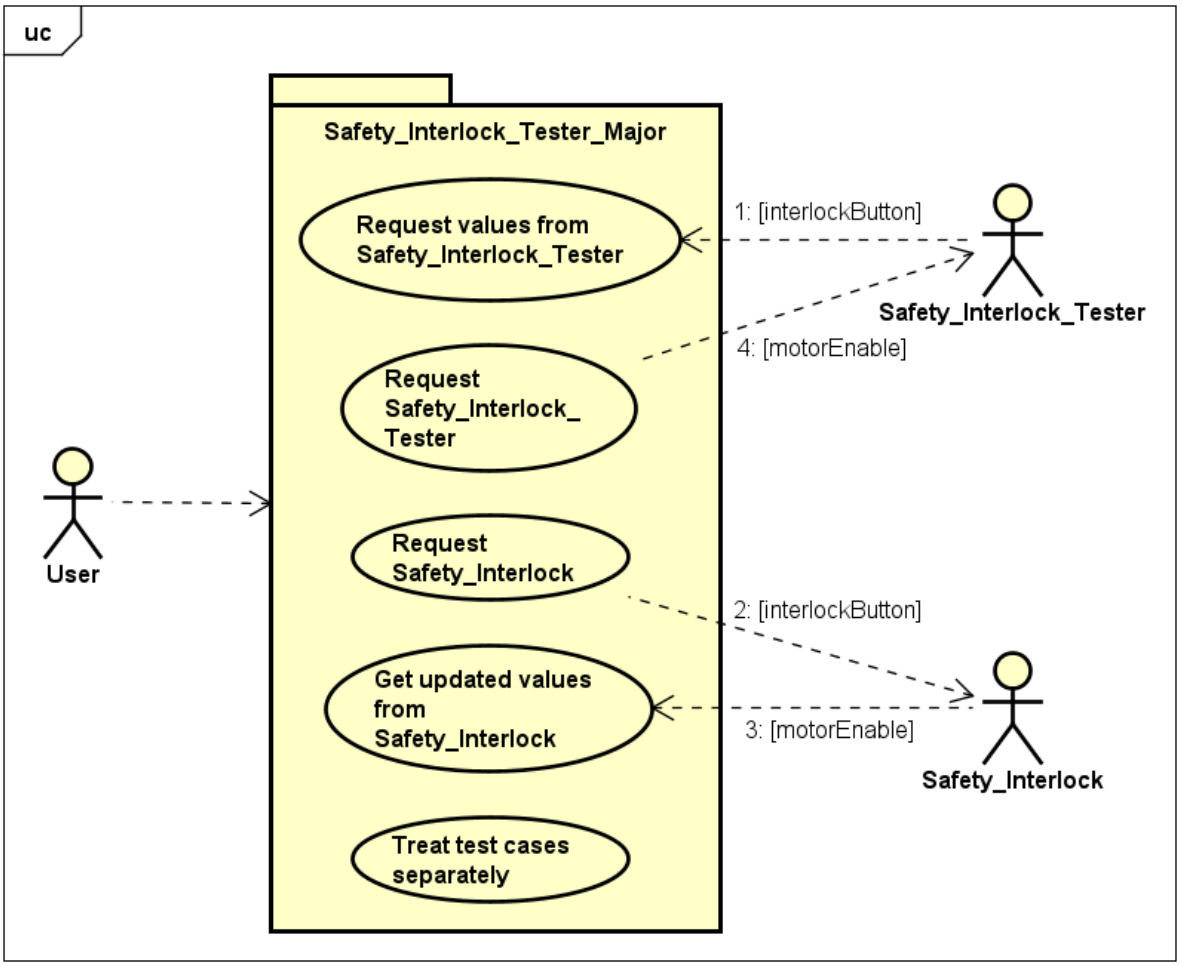
05/02/25:

- Criação do documento.

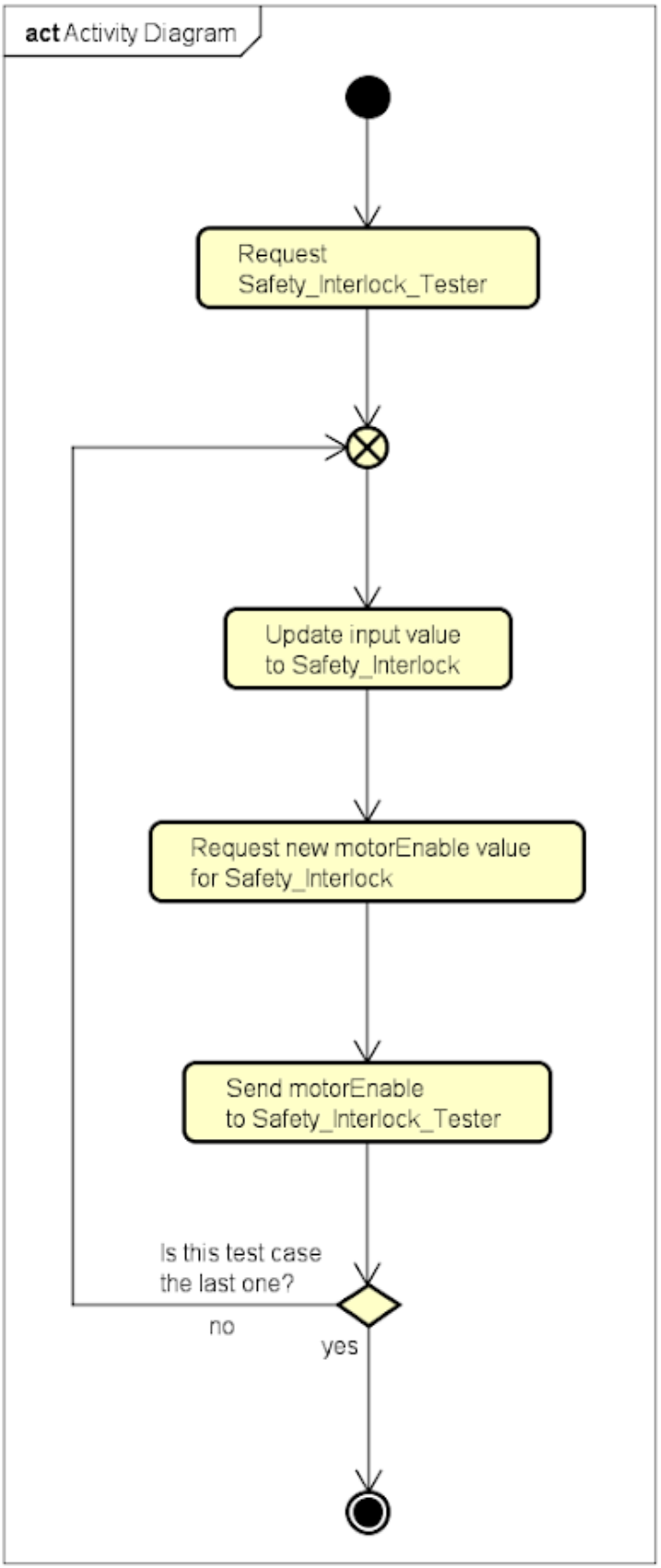
10/02/25:

- Lógica de funcionamento do majoritário revista.

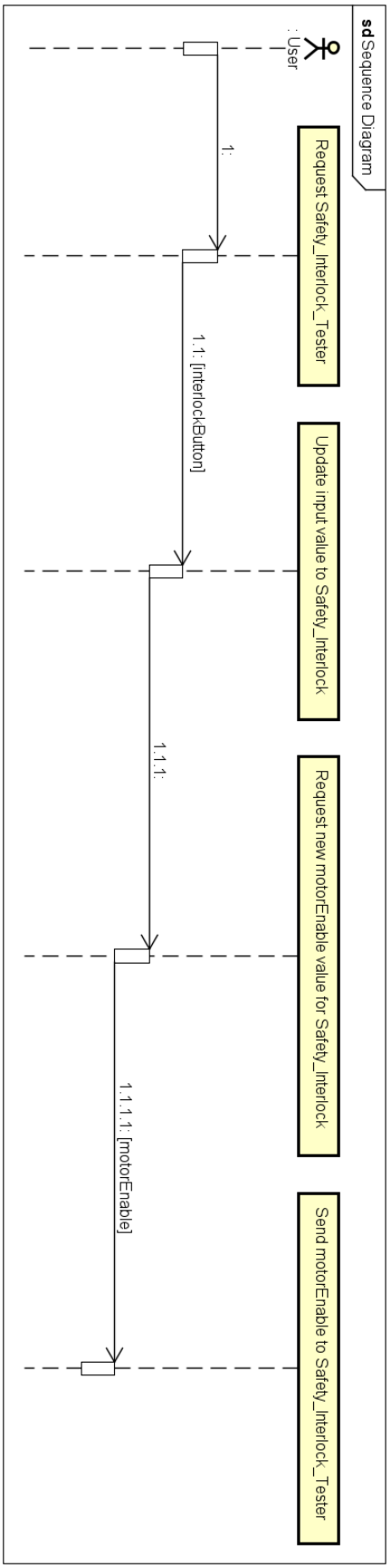
Anexo A.10 - Diagrama de casos de uso do MTU_S1



Anexo A.11 - Diagrama de atividades do MTU_S1



Anexo A.12 - Diagrama de sequência do MTU_S1



Anexo A.13 - Pseudocódigo do MTU_S1

Safety_Interlock_Tester_Major

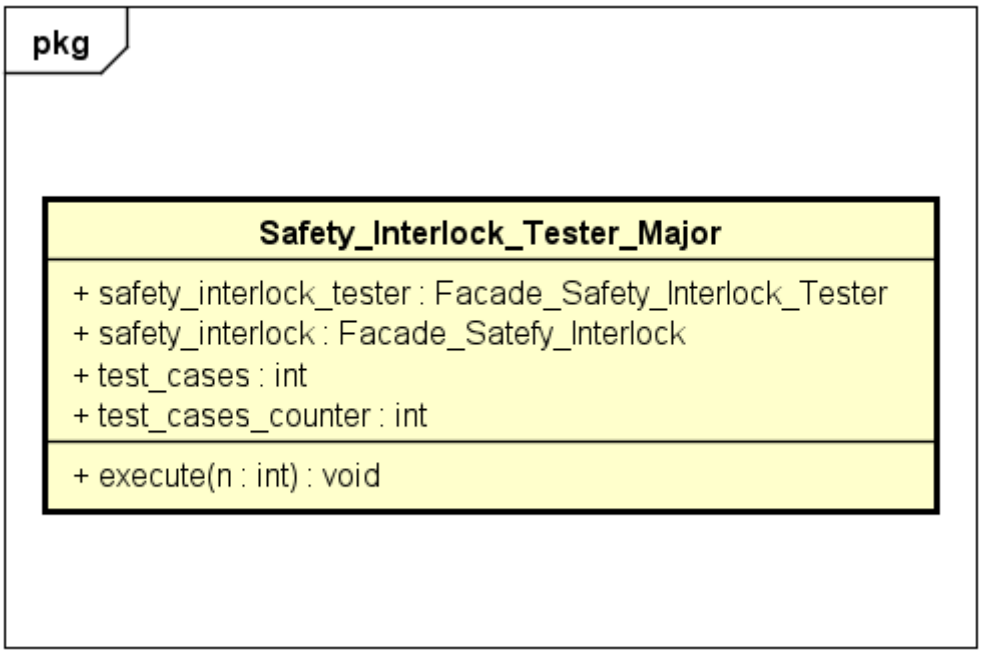
```
include: iostream
         Facade_Safety_Interlock
         Facade_Safety_Interlock_Tester
define: NA
variables: safety_interlock_tester // Facade_Safety_Interlock_Tester
           safety_interlock // Facade_Safety_Interlock
           test_cases // int
           test_cases_counter // int

method execute(int n)
      test_cases = n
      for test_cases_counter = 0 to test_cases
            // Request Safety_Interlock_Tester
            // Update input value to Safety_Interlock
            safety_interlock_tester.send_parameters(test_cases_counter)

            // Request new motorEnable value for Safety_Interlock
            safety_interlock.interlock()

            // Send motorEnable to Safety_Interlock_Tester
            safety_interlock_tester.show_result()
end method
```

Anexo A.14 - Diagrama de classe do MTU_S1



Anexo A.15 - Documento de execução de testes do S1, páginas 1 a 3

Documento de Execução de Testes

Serviço de Intertravamento de Segurança (S1)

Resumo

Síntese

O Serviço de Intertravamento de Segurança executa a parada dos drivers de motores do AGV quando um sinal de parada é acionado por meio da botoeira externa. O Serviço de Teste Unitário, para checar o funcionamento correto do intertravamento, deve enviar valores aleatórios do sinal de parada para uso no serviço, por meio de um software Majoritário de Teste Unitário. A Figura 1 indica o caminho de dados que foi usado no teste a ser discutido a seguir.

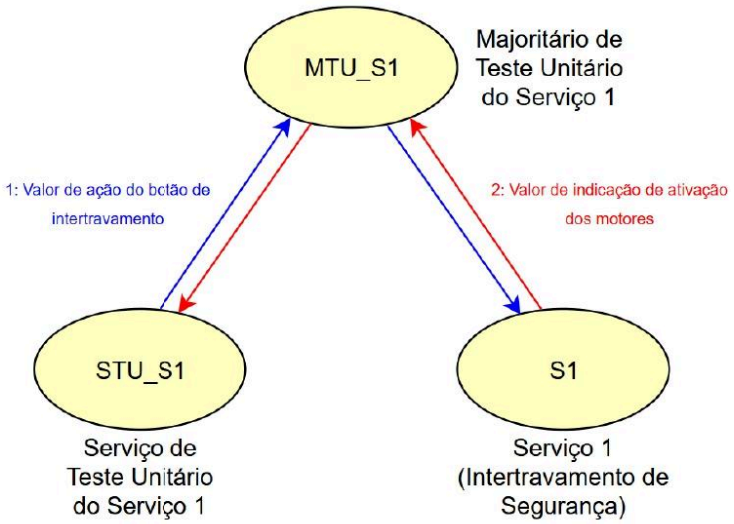


Figura 1: Diagrama de caminho de dados dos serviços envolvidos no teste do S1.

Ambos serviço, testador e majoritário foram documentados e programados. Ao final da montagem do código, os softwares foram conectados entre si e executados por meio de um programa "main" para verificação do funcionamento.

Adequações necessárias

Serviço de Intertravamento de Segurança (S1)

- Não foram encontradas adequações no código do S1.

Serviço de Teste Unitário do Serviço de Intertravamento de Segurança (STU_S1)

- Não foram encontradas adequações no código do STU_S1.

Majoritário de Teste Unitário do Serviço de Intertravamento de Segurança (MTU_S1)

- Não foram encontradas adequações no código do MTU_S1.

Resultados alcançados

Serviço de Intertravamento de Segurança (S1)

- **Desativar o driver:** A função de mudar o valor de ativação dos motores por meio do parâmetro **motorEnable** em caso de intertravamento foi feita em todos os casos de teste da maneira correta.

Serviço de Testes do Serviço de Intertravamento de Segurança (STU_S1)

- **Tratamento de casos de teste:** Os casos de teste foram atribuídos com seus respectivos valores condizentes antes da execução completa do STU_S1. Observar a corretude desta etapa foi importante, pois é fundamental que cada valor seja atribuído em seu parâmetro determinado para que o teste funcione corretamente.
- **Dados de entrada:** O envio dos valores de entrada para cada caso de teste e uso do S1 foi feito da forma correta. Por se tratar de alterar o valor de uma variável global diretamente, era esperado que não houvesse erros diretos na execução deste método.
- **Recebimento de saída obtido:** Os valores de saída foram recebidos pelo STU_S1 sem impasses. Novamente, a saída do S1 depende da alteração de uma variável global de maneira direta, ou seja, erros nesta execução depende do funcionamento do S1, e não exatamente do STU_S1 em questão.
- **Comparação de resultados:** As comparações de cada caso de teste foram feitas corretamente. Todos os casos de teste foram indicados como sucesso, e foram atribuídos como tal.
- **Apresentação de resultados:** Cada caso de teste teve seus resultados mostrados na tela do ambiente de execução para comparação via usuário físico.
- **Informações adicionais:** Nenhuma informação adicional a tratar.

Majoritário de Teste Unitário do Serviço de Intertravamento de Segurança (MTU_S1)

- **Solicitação de dados do STU_S1:** Este método envolve se o método de envio de entradas do STU_S1 está operando como esperado. Pela corretude do método já esclarecida, é também possível dizer que os dados foram solicitados pelo MTU_S1 e recebidos pelo STU_S1, por meio do software de fachada do testador, da maneira correta.
- **Chamada do S1:** O MTU_S1 executa a chamada do S1 corretamente, por meio do método declarado no software de fachada do S1.

- **Chamada do STU_S1:** O MTU_S1 executou a chamada do STU_S1 por meio do software de fachada de forma correta, sendo possível executar as comparações e apresentação de resultados de cada teste na ordem esperada.
- **Tratamento de casos de teste:** A troca de dados entre cada caso de teste foi feita em ordem, ou seja, não houve repetição de casos de teste, e o MTU_S1 parou a execução do S1 e do STU_S1 após todos os casos de teste serem observados.
- **Informações adicionais:** É importante que o majoritário esteja indicando o número correto de casos de teste, ou é possível que os softwares continuem sendo executados com valores incoerentes e apresentando resultados isentos de significado.

Análise dos resultados

Interação entre interlockButton e motorEnable

É notável que os parâmetros **interlockButton** e **motorEnable** são proporcionais. Logo, não é possível obter um caso de teste em que ambos não possuam o mesmo valor e que o caso indique sucesso. Para funcionar corretamente, o S1 deve ter valor 1 em **interlockButton** e 1 em **motorEnable**, e vice versa.

Dependência de endereços de entrada e saída da Beckhoff

Por ser um serviço que depende de variáveis de endereçamento de hardware, é importante garantir que o S1 esteja condizente com o protocolo de endereçamento da Beckhoff usada no sistema. Por consequência, é igualmente importante que o Serviço de Condição Inicial de Hardware **(S11)** esteja montado corretamente para que o S1 não seja comprometido em sua execução.

Conclusão

Este documento apresentou o processo de execução dos testes envolvendo o Serviço de Intertravamento de Segurança. Ao observar os resultados e a estrutura de cada caso de teste, é possível dizer que o teste do serviço foi considerado um sucesso, e o funcionamento do S1 foi validado por meio desta execução.