Universidade Federal da Paraíba
Centro de Informática
Programa de Pós-Graduação em Informática

Verificação de Projetos de Sistemas Embarcados através de Cossimulação Hardware/Software

José Cláudio Vieira e Silva Junior

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Informática da Universidade Federal da Paraíba como parte dos requisitos necessários para obtenção do grau de Mestre em Informática.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Sinais, Sistemas Digitais e Gráficos

Alisson Vasconcelos de Brito (Orientador)
Tiago Pereira do Nascimento (Coorientador)

João Pessoa, Paraíba, Brasil

S586v Silva Junior, José Cláudio Vieira e.
Verificação de projetos de sistemas embarcados através de cossimulação hardware/software / José Cláudio Vieira e Silva Junior.- João Pessoa, 2015.
69f. : il.
Orientador: Alisson Vasconcelos de Brito
Coorientador: Tiago Pereira do Nascimento
Dissertação (Mestrado) - UFPB/CI
1. Informática. 2. Sistemas embarcados. 3. Verificação funcional. 4. Cossimulação. 5. High Level Architecture. 6. Hardware-in-the-loop.

UFPB/BC CDU: 004(043)

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de **JOSÉ CLÁUDIO VIEIRA E SILVA JUNIOR**, candidato ao título de Mestre em Informática na Área de Sistemas de Computação, realizada em 17 de agosto de 2015.

Ao décimo sétimo dia do mês de agosto do ano de dois mil e quinze, às dez horas, no Centro de Informática - Universidade Federal da Paraíba (unidade Mangabeira), reuniram-se os membros da Banca Examinadora constituída para julgar o Trabalho Final do **Sr. José Cláudio Vieira e Silva**, vinculado a esta Universidade sob a matrícula 2013103662, candidato ao grau de Mestre em Informática, na área de "*Sistemas de Computação*", na linha de pesquisa *"Sinais, Sistemas Digitais e Gráficos"*, do Programa de Pós-Graduação em Informática, da Universidade Federal da Paraíba. A comissão examinadora foi composta pelos professores doutores: **Alisson Vasconcelos de Brito (PPGI-UFPB)**, Orientador e Presidente da Banca, **Tiago Pereira do Nascimento (PPGI-UFPB)**, Co-orientador e Examinador Interno, **Vivek Nigam (PPGI – UFPB),** Examinador Interno e **Elmar Uwe Kurt Melcher (UFCG)**, Examinador Externo à Instituição. Dando início aos trabalhos, o professor Alisson Vasconcelos de Brito, cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse, oralmente, a exposição do trabalho de dissertação de mestrado intitulado "Verificação de Projetos de Sistemas Embarcados Através de Cossimulação Hardware/Software". Concluída a exposição, o candidato foi arguido pela Banca Examinadora que emitiu o seguinte parecer: "*aprovado*". Assim sendo, eu, Alisson Vasconcelos de Brito, Coordenador interino do Programa de Pós Graduação em Informática - PPGI, lavrei a presente ata que vai assinada por mim e pelos membros da Banca Examinadora. João Pessoa, 17 de agosto de 2015.

Alisson Vasconcelos de Brito

Profº Dr. Alisson Vasconcelos de Brito (PPGI-UFPB)
Orientador e presidente da Banca

Profº Dr. Tiago Pereira do Nascimento (PPGI-UFPB)
Co-orientador e Examinador Interno

Profº Dr. Vivek Nigam (PPGI – UFPB)
Examinador Interno

Profº Dr. Elmar Uwe Kurt Melcher (UFCG)
Examinador Externo à Instituição

Resumo

Este trabalho propõe um ambiente para verificação de sistemas embarcados heterogêneos através da cossimulação distribuída. A verificação ocorre de maneira síncrona entre o software do sistema e o sistema embarcado usando a High Level Architecture (HLA) como middeware. A novidade desta abordagem não é apenas fornecer suporte para simulações, mas também permitir a integração sincronizada com todos os dispositivos de hardware físico. Neste trabalho foi utilizado o Ptolemy como uma plataforma de simulação. A integração do HLA com Ptolemy e os modelos de hardware abre um vasto conjunto de aplicações, como o de teste de vários dispositivos ao mesmo tempo, executando os mesmos, ou diferentes aplicativos ou módulos, a execução de multiplos dispositivos embarcados para a melhoria de performance. Além disso a abordagem de utilização do HLA, permite que sejam interligados ao ambiente, qualquer tipo de robô, assim como qualquer outro simulador diferente do Ptolemy. Estudo de casos são apresentado para provar o conceito, mostrando a integração bem sucedida entre o Ptolemy e o HLA e a verificação de sistemas utilizando Hardware-in-the-loop e Robot-in-the-loop.

Palavras-chave: Sistemas embarcados, Verificação funcional, Cossimulação, High Level Architecture, Hardware-in-the-loop, Ptolemy.

Abstract

This work proposes an environment for verification of heterogeneous embedded systems through distributed co-simulation. The verification occurs in real-time co-simulating the system software and hardware platform using the High Level Architecture (HLA) as a middleware. The novelty of this approach is not only providing support for simulations, but also allowing the synchronous integration with any physical hardware devices. In this work we use the Ptolemy framework as a simulation platform. The integration of HLA with Ptolemy and the hardware models open a vast set of applications, like the test of many devices at the same time, running the same, or different applications or modules, the usage of Ptolemy for real-time control of embedded systems and the distributed execution of different embedded devices for performance improvement. Furthermore the use of HLA approach allows them to be connected to the environment, any type of robot, as well as any other Ptolemy simulations. Case studies are presented to prove the concept, showing the successful integration between Ptolemy and the HLA and verification systems using Hardware-in-the-loop and Robot-in-the-loop.

Keywords: Embedded systems, Functional hardware verification, Co-simulation, High level Architecture, Hardware-in-the-loop, Ptolemy.

Agradecimentos

Gostaria de agradecer aos meus familiares, em especial aos meus pais, por sempre me apoiar e contribuir para que eu pudesse alcançar meus objetivos.

Ao meu orientador Professor Alisson Brito pelo incentivo, dedicação e disponibilidade ao longo de toda esta caminhada. Ao meu coorientador Professor Tiago Nascimento pelo auxílio e contribuição durante a pesquisa.

Aos professores do PPGI José Antônio, Alexandre Duarte, Vivek Nigam por terem compartilhado seus conhecimentos nas disciplinas que ministraram.

Aos amigos que conheci durante o curso, Ramon Leonn, Rharon Maia, Frank Cesar, em especial o Luís Feliphe pela parceria e apoio.

E, por fim, a todos aqueles que, direta ou indiretamente contribuíram para que essa jornada pudesse ser concluída.

”Sonhos determinam o que você quer. Ação determina o que você conquista.”

— Aldo Novak

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

A necessidade por sistemas embarcados de alto desempenho, disponibilidade e confiabilidade tem tornado os sistemas computacionais cada vez mais complexos. Diante de tal complexidade, realizar um projeto com apenas uma linguagem ou até mesmo com um único nível de abstração tem se mostrado insuficiente [Mello e Wagner 2002], já que atualmente os sistemas computacionais quase sempre são compostos por módulos desenvolvidos em software, hardware, e podendo ainda conter módulos analógicos e partes mecânicas [Marrec et al. 1998].

Esses sistemas que combinam componentes de diferentes domínios que precisam cooperar são chamados de sistemas heterogêneos. Para verificar esse tipo de sistema é comum validar cada módulo que foi desenvolvido separadamente. Diante dessa heterogeneidade nos projetos de sistemas embarcados, verificar suas funcionalidades torna-se uma tarefa cada vez mais difícil. De acordo com [Bergeron 2003] e [Piziali 2004] a etapa de verificação funcional é um dos maiores gargalos dentro de um projeto de desenvolvimento de hardware. Estima-se que cerca de 70% dos recursos empregados em um projeto de hardware são utilizados na etapa de verificação funcional [Bergeron 2003] [Bergeron 2006].

É comum utilizar simulações no ambito do desenvolvimento de projetos de hardware, uma vez que alguns tipos de simulações como o Hardware-in-the-loop (HiL) e Robot-in-the-loop (RiL), possibilitam prever o comportamento do projeto em seu ambiente real. A cossimulação é uma técnica onde se utiliza uma simulação composta por sistemas heterogêneos, podendo conter simuladores ou dispositivos físicos, todos os integrates da simulação se comunicam entre si. Com base nisso, simulações HIL e RIL são realizadas através de

uma cossimulação, pois a utilização de dispositivos de hardware reais podem aumentar a realidade dos resultados obtidos nessas simulações.

Cossimular sistemas embarcados em hardware com modelos de simulação em software pode possibilitar a verificação de sistemas em hardware, quando há disponível um modelo de referência em software. Também é possível verificar um modelo de simulação quando há disponível um modelo de referência em hardware. Tais situações podem acelerar o projeto de sistemas embarcados. Estas são causas que motivaram o desenvolvimento desse trabalho. Porém alguns pontos necessitam ser superados, tendo em vista que a grande dificuldade para o desenvolvimento está em integrar simuladores em software com implementações também heterogêneas em hardware.

Nesse contexto, este trabalho vem contribuir apresentando uma solução para cossimulação de modelos de hardware de forma síncrona com outros modelos visando a verificação funcional de sistemas embarcados, além de possibilitar que tais simulações sejam capazes de simular modelos heterogêneos de forma distribuída. Para isso, foi utilizado o High Level Architecture [IEEE 2000] para a integração e sincronização da ferramenta, o Ptolemy [Lee e John 1999], para simular o projeto em software e três plataformas de hardware o Arduino Mini [Banzi 2009] e o Freescale Freedom KL25Z e a Raspberry Pi para execução de modelos em Hardware. A utilização da HLA possibilitou realizar uma comunicação síncrona e consistente com o Ptolemy e com os sistemas embarcados. Com essa integração foi possível implementar modelos computacionais no Ptolemy e verificar a implementação com modelos de hardware. Para isso foram realizados vários experimentos, demonstrando o uso de HIL e RIL, através destes experimentos ficou demonstrado que o ambiente desenvolvido possibilita a verificação ou teste de sistemas robóticos, podendo ser verificados tanto o sistema embarcado quanto a parte física do robô.

1.1 Objetivos

O objetivo geral deste trabalho é desenvolver um ambiente para verificação de projetos de sistemas embarcados através de cossimulação hardware/software.

Os objetivos específicos são:

- Especificação de um ambiente genérico e simplificado de verificação funcional que

possibilite uma integração com o HLA de forma síncrona;

- Possibilitar a integração de dispositivos de hardware (HIL) e robôs (RIL) com o modelo de testbench proposto;
- Validar o ambiente desenvolvido com aplicações reais;

1.2 Metodologia

O primeiro passo para viabilizar o desenvolvimento de um ambiente de verificação de sistemas heterogeneos será o estudo de ferramentas que possibilitem a implementação de modelos heterogêneos, em especial o framework Ptolemy. Após isso, a especificação HLA será estudada em detalhes, uma vez que ela possibilita a realização de simulações distribuídas. Após o conhecimento adquirido sobre o Ptolemy e o HLA, será necessário especificar e desenvolver um modelo de verificação, em especial o testbench, que seja capaz de interagir com estes dois mundos com a finalidade de permitir a verificação de modelos heterogeneos de forma distribuída. Para isso serão estudados os métodos e tipos de verificação que são utilizados para verificar projetos de hardware.

Para poder realizar cossimulações e validar o ambiente que foi desenvolvido, algumas etapas tiveram que ser realizadas:

- Desenvolvimento de algoritmos de software e hardware para que sejam utilizados como modelo de referencia;
- Desenvolvimento de um robô móvel para possibilitar uma simulação RIL;
- Planejamento e Modelagem de simulações HIL e RIL;
- Validação do ambiente desenvolvido com aplicações reais;

1.3 Trabalhos relacionados

A cossimulação possibilita a execução paralela dos simuladores necessários para realizar a validação de um sistema. Tais simuladores podem ser caracterizados por um sistema de computador, recursos de hardware ou uma parte real de um sistema. Estes ambientes simulados

podem ser homogêneos ou heterogêneos, as simulações homogêneas permitem apenas a utilização de um modelo computacional, já as simulações heterogêneas possibilitam a interação de simuladores que trabalhem com diversos tipos de modelos computacionais. Dadas essas duas possibilidades em que o simulador é caracterizada, há alguns trabalhos na literatura que utilizam todas essas características, a fim de executar a verificação funcional de sistemas embarcados.

O autor em [Deprá et al. 2008] apresenta uma metodologia para verificação funcional de hardware através de cossimulação paralela. O trabalho utiliza funcionalidades de "threads e "handshakes"fornecida pela linguagem Verilog PLI (Programming Language Interface) para garantir que a cossimulação seja feita corretamente de forma síncrona. São colocadas linhas de execução paralelas através de threads “autônomas” em relação ao fluxo de execução do simulador de hardware, para que seja possível enviar os dados aos simuladores simultaneamente. Já o handshake é o mecanismo de sincronização utilizado entre as partes de software e hardware através das interfaces fornecidas pela API (Application Program Interface) da PLI combinadas às funções específicas de sincronização.

Em [Shah e Irfan 2005] é apresentado um ambiente para verificação e validação utilizando a técnica de Hardware-in-the-loop (HIL). O trabalho se concentra em uma integração entre o Matlab Simulink com o Microsoft Visual C. Uma proposta para a sincronização de tempo entre os modelos de simulação é apresentada, ela possibilita estabelecer uma comunicação em tempo real entre os modelos envolvidos. Para isso, o computador que está executando o Simulink fica responsável pela lógica de controle para sincronizar o tempo. Foi utilizado um microcontrolador PIC como dispositivo físico, integrado a um Programmable Interval Timer (PIT), para obter informações sobre o tempo.

Os autores em [Souza et al. 2003] apresenta um ambiente que visa integrar os componentes virtuais em modelos de cossimulação distribuída. O modelo de cossimulação descrito se baseia em uma versão modificada do HLA chamada Distributed Backbone co-simulation (DCB). Ele impõe normas próprias para a troca de dados e diferente do HLA, exige que componentes façam chamadas explicitas para funções do Run-time infrastructure (RTI). De maneira geral o funcionamento é igual ao do HLA, onde um modelo de cossimulação é uma federação, composto por federados autônomos e distribuídos.

Há também trabalhos que utilizam a técnica de Hardware-in-the-loop com a finalidade

de verificar sistemas embarcados, mas eles estão restritos a um conjunto limitado de modelos e dispositivos [Shah e Irfan 2005], no entanto estes trabalhos foram desenvolvidos para verificar um conjunto restrito de modelos, não sendo possível simular e verificar qualquer tipo de dispositivo ou aplicação, ficando restrito apenas a modelos de simulação homogênea.

Ambientes que executam apenas simulações de modelos homogêneos muitas vezes são incapazes de implementar a heterogeneidade devido ao grande esforço que é empregado para adaptação a diferentes plataformas de hardware e comunicação e protocolos de sincronização. Tais características como heterogeneidade e distribuição de uma plataforma de verificação e simulação é fornecida neste trabalho através da integração do simulador Ptolemy com o HLA.

1.4 Estrutura do documento

Este trabalho está dividido da seguinte maneira: O capitulo 1 apresentou o contexto geral deste trabalho bem como os trabalhos relacionados. O capitulo 2 aborda os contéudos principais necessários para o entendimento deste trabalho. O capitulo 3 trata sobre o desenvolvimento do ambiente proposto. O capitulo 4 apresenta o estudo de caso e os resultados obtidos. O capitulo 5 apresenta a conclusão e os trabalhos futuros.

Capítulo 2

Estado da Arte

Este capítulo apresenta os conceitos básicos necessários para compreensão do trabalho desenvolvido. Em particular abordamos os aspetos das simulações, o padrão referente a arquitetura de alto nível, metodologias de verificação de projetos de hardware e também o framework Ptolemy.

2.1 Princípios de simulação distribuída

Na área da simulação distribuída, existem duas grandes linhas de pesquisa que atuam em linhas de pesquisas diferentes: a *Parallel and Distribued Simulation* (PADS) e a *Distributed Interactive Simulation* (DIS) [Fujimoto 1999], a PADS está voltada para assuntos que envolvem a execução distribuída de simulações discretas baseadas em eventos [Fujimoto 2001], e a DIS está voltada para atender as simulações de tempo real. Apesar dessas duas linhas de pesquisa possuírem objetivos diferentes, elas adotam alguns conceitos e metodologias comuns. A simulação distribuída possibilita que um único modelo de simulação tenha suas partes (também chamada de Processos Lógicos-PL [Ferscha e Tripathi 1998]) executadas em ambientes computacionais distribuídos [Fujimoto 1990]. A simulação distribuída é usada principalmente por oferecer alguns benefícios como a redução do tempo de execução, a simulação geográficamente distribuída e a possibilidade de integração de diferentes tipos simuladores.

Os mecanismos de sincronização são primordiais para a simulação distribuída, pois eles possibilita que modelos simulados em diferentes ambientes sejam sincronizados. Esses me-

canismos entendem que as simulações são compostas por vários processos lógicos (PLs) que se comunicam por meio de mensagens ou eventos, sempre havendo um timestamp associado. A sincronização garante que eventos realizados pelos PLs ocorram no seu respectivo tempo (timestamps) de modo ordenado [Fujimoto 1990]. Os algoritmos de sincronização podem ser divididos entre conservativos e otimistas, conforme detalhado a seguir.

2.1.1 Sincronização conservativa

No modelo de sincronização conservativa, o objetivo principal é evitar possíveis erros, ou seja, antes de executar algum evento com o timestamp "t" o processo lógico (PL) deve confirmar se ele é um evento seguro ou não. Esse procedimento sendo respeitado por todos os PLs, e garantido que todos as informações estarão corretas sobre o ponto de vista da sincronização.

De maneira geral, este mecanismo conservativo garante que o simulador somente receba eventos com timestamps maiores ou iguais que o seu próprio tempo virtual [Fujimoto 1990] [Misra 1986]. Este método de sincronização é amplamente utilizado para a sincronização de simuladores [Donath et al. 1995].

2.1.2 Sincronização otimista

Diferente da sincronização conservativa, nesta abordagem os processos lógicos (PLs) podem processar eventos no mesmo momento em que recebem, ou seja, independente da ordem. Caso os PLs recebam eventos com timestamp menores, o PL realizará um processo de rollback para um estado interno anterior que é considerado correto e depois propagar o rollback para as outras LPs afetadas [Jefferson 1985], [Quaglia e Santoro 2003].

2.2 Ambiente de cossimulação

O princípio fundamental da cossimulação é o suporte para execução cooperativa de diferentes simuladores [Souza et al. 2003]. O processo de cossimulação necessita de simuladores capazes de simular conjuntamente as partes de software e hardware garantindo a consistencia na interaçao entre as partes. Cada componente da cossimulação fica responsável de executar

individualmente alguma parte do sistema, que é descrita em alguma linguagem e utiliza algum modelo computacional e nível de abstração específico [HESSEL 2001]. Ou seja, a cossimulação permite que os modelos possam ser especificados em diferentes níveis de abstração [Wolf 1994], executando tarefas em conjunto com a utilização de um formalismo comum.

A cossimulação utiliza mecanismos que são aplicados em simulações distribuídas, já que em uma cossimulação se faz necessário interligar diferentes tipos de simuladores ou dispositivos de hardware que possuem diferentes tipos de dominios. Essa interligação pode ser de forma descentralizada, ou seja, simuladores geograficamente distribuídos, sendo assim, o ambiente de cossimulação deve possuir estruturas especificas para o controle da comunicação, sincronização além de uma padronização referente aos dados trocados entre os simuladores [Bishop e Loucks 1997]. A característica principal que o ambiente de cossimulação utiliza de simulações distribuidas é em relação a sincronização, uma vez que, os diversos tipos de simuladores que integram o sistema possuem velocidades de execução diferentes. Com isso a sincronização irá garantir que os eventos disparados pelos simuladores possam ocorrer em seu devido tempo durante a simulação.

A técnica de cossimulação é bastante utilizada, e pode auxiliar em diversas áreas. Uma delas é a realização de análise e teste quando existe a necessidade para escolha de componentes [Sung, Oh e Ha]. Outro ponto é possibilitar que decisões sejam tomadas quando se faz necessário realizar algum tipo de particionamento de sistemas [Coumeri e Thomas 1995] [Kim et al.]. Outra possibilidade é a de verificação de implementações de sistemas [Brito, Vieira e Nascimento 2015] [Kim et al.], essa abordagem possibilita verificar se o sistema ou simulador está em conformidade com alguma especificação.

Pelo simples fato que de uma cossimulação permita que vários simuladores possam ser combinados em um único ambiente [Schmerler, Tanurhan e Muller-Glaser 1995] e também permita que sejam inseridos dispositivos de hardware nesta mesma simulação, surge um problema crucial que é o desenvolver uma interface de comunicação entre eles que defina a forma como eles irão se comunicar.

2.2.1 Modelos de cossimulação

Um ambiente de cossimulação pode ser desenvolvido utilizando duas abordagens. A primeira é quando se usa apenas uma máquina de simulação, ou seja uma simulação homogênea. A outra abordagem é quando se utiliza duas ou mais máquinas de simulação diferentes, tornando uma simulação heretogênea.

Ambiente de cossimulação homogêneo geralmente não oferece problemas relacionados a sincronização, pois por possuir apenas uma máquina de simulação, todos as informações internas da simulação se encontra sempre acessíves para os subsistemas. Assim, os envolvidos podem utilizar estas informações e coordenar as interações entre todos os subsistemas da simulação.

Em ambientes de cossimulação heterogêneos, é possível utilizar vários mecanismos de simulação, ou seja, podem ser interligados vários simuladores ou plataformas de hardware. Diante dessa heterogeneidade, é primordial ter controle sobre a sincronização, comunicação e tipos de mensagens trocadas entre os simuladores e modelos de hardware [Bishop e Loucks 1997] [Hübert 1998]. Para contornar o problema de comunicação entre os diversos simuladores, se faz necessário que o ambiente de cossimulação possua uma semântica que seja compreendida por todos os envolvidos [Adams e Thomas 1996]. Já para realizar esta comunicação, é necessário que os envolvidos tenham interfaces adequadas que possibilite a transferência de dados [Bishop e Loucks 1997] de forma consistente e unificada [Kim et al.].

Levando em consideração o mecanismo de sincronização, uma cossimulação pode ser não-temporizada ou temporizada. Em uma cossimulação Não-temporizada, o fator tempo não é levado em conta quando os eventos são executados, ou seja, se utiliza do modelo de sincronização conservador. Já em uma cossimulação Temporizada, o tempo de execução dos eventos é considerado e cada integrante deve ter seu relógio local, neste caso se faz necessário a utilização de um modelo de sincronização.

2.3 Princípios de verificação de sistemas embarcados

Quando se desenvolve projetos de sistemas embarcados sempre se encontra algum tipo de erro lógico. Estes erros podem ser causados por algum tipo de erro na especificação ou por divergencias encontradas entre o comportamento de um modelo implementado e um

modelo que está de acordo com uma especificação. Os erros que contém divergencia na implementação da especificação pode ser detectado e poseteriormente corrigido se for usado uma verificação. [Kirkwood].

A verificação é um processo utilizado para demonstrar que as funcionalidades do projeto a ser desenvolvido estão em conformidade com a sua especificação [Bergeron 2006].

Os tipos de verificação mais conhecidos são: verificação formal, verificação funcional e verificação híbrida.

2.3.1 Verificação formal ou estática

De acordo com as definições de [Bergeron et al. 2005], a verificação formal, também denominada verificação estática realiza comparações por métodos formais ou matemáticos de uma implementação contra uma especificação a fim de detectar se a implementação viola a especificação. Este tipo de verificação se divide em verificação de modelos, de equivalencia e prova de teoremas. Através deste método é possível provar a inexistencia de erros no projeto, no entanto esse procedimento não é trivial, e não pode ser aplicado a todos os problemas práticos devido a sua complexidade computacional.

2.3.2 Verificação funcional ou dinâmica

A verificação funcional, também conhecida como verificação dinâmica, define algumas etapas afim de demonstrar que o objetivo do projeto é preservado em sua implementação [Bergeron 2006]. Este modelo tem o objetivo principal detectar erros lógicos na implementação. A verificação funcional é realizada através de simulações, sendo assim, não existe limitações quanto ao tamanho do modelo que será verificado. No entanto com este tipo de verificação, não se pode provar a ausência de erros, se limitanto apenas a detecção dos mesmos.

O fato deste modelo ser realizado através de simulações, o torna o método mais utilizado para realizar verificação de modelos de hardware, já que as técnicas formais são complexas e não podem ser aplicado em todos os tipos de problemas. [Edwards et al. 1997].

2.3.3 Verificação híbrida ou semi-formal

A verificação híbrida, também chamada de semi-formal, tenta unir as técnicas de verificação formal e funcional. Esta união se da pelo fato de tentar superar a limitações referente a complexidade na utilização de modelos formais e também superar limitações referente a simulações que são realizadas em verificações funcionais.

2.3.4 Conceitos sobre verificação funcional

Conforme [Bergeron 2003] verificação funcional é um método utilizado para comparar o Design under Test (DUT) com sua especificação. Para que seja realizada essa comparação se faz necessário a utilização de algumas técnicas, dentre elas a utilzação de testbench e gerador de estímulos.

Testbench

Como já dito anteriormente, a verificação funcional é realizada através de uma simulação. O ambiente de verificação deve possibilitar que na simulação possa ser integrado um DUV (Design under Verification) ou um DUT (Design under Test), insiram estímulos para o DUV e consequentemente compare as respostas com algum modelo que condiz com a especificação, ou seja um modelo ideal. O ambiente que dispõe de todas estas características é chamado de testbench, este é usado para verificar o DUV.

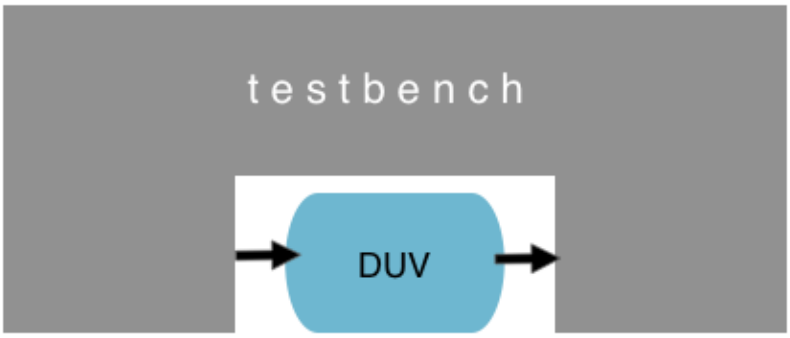

Figura 2.1: Esquema de um testbench genérico.

Na Figura 2.1 pode ser observado a estrutura genérica de um testbench. O testbench é representado pelo U invertido, como pode ser visto na figura, é a região que envolve o DUV, que também pode ser utilizado um DUT. O DUV é utilizado quando é realizada uma

verificação de uma implementação. Já o DUT é realizado um teste de um hardware, ou seja o DUV não verifica dispositivos concluídos, verifica apenas quando estes estão sendo implementados. Nos ambientes de verificação que utilizam esta estrutura, o DUV ou DUT devem receber estímulos ao mesmo tempo em que estes são enviados para um modelo de referência (também chamado de Golden Model). Posteriormente os resultados devem ser enviados para um modelo que realizará essa coleta e fará uma comparação dos resultados, normalmente este modelo é chamado de Scoreboard com comparador. O Scoreboard por sua vez poderá validar ou não o DUV, consequentemente confirmando ou não a conformidade da implementação com a especificação, ou no caso do DUT, testar se o DUT (modelo de hardware) se comporta como o esperado.

Geração de estímulos

A geração de estímulos tem o objetivo de enviar algum tipo de dado para o DUT e modelo de referencia. Normalmente estes estímulos são citados nas especificações, podendo também estimular de forma randômica ou dados reais de algum modelo. Essa geração de estímulos geralmente é realizada através de alguma ferramenta, também podendo ser feita de forma manual. A geração manual é pouco eficiente, pois diversas possibilidades de dados devem ser enviados para conseguir alcançar o objetivo. Já a geração com a utilização de ferramentas possibilita maior chance para detectar possíveis erros, uma vez que é mais fácil enviar uma quantidade maior de estímulos.

2.4 Padrão IEEE 1516, a Arquitetura de Alto Nível

Para que a comunicação entre os simuladores no ambiente de cossimulação seja eficaz, é necessário que este ambiente entenda a semântica de todos os modelos envolvidos e como ações em um domínio podem afetar o estado do outro [Adams e Thomas 1996], para possibilitar essa comunicação, se faz necessário utilizar interfaces adequadas para cada um dos envolvidos. Também há a necessidade que o ambiente possua mecanismos para traduzir as informações e executar a transferencia de dados [Bishop e Loucks 1997] de forma unificada e consistente [Kim et al.].

Existem algumas tecnologias que oferecem os recursos necessários que possibilitam a

comunicação entre componentes em uma simulação distribuídas. Dentre as mais utilizadas estão: Remote Method Invocation (RMI), Common Object Request Broker Architecture (CORBA) e a High Level Architecture (HLA). Dentre essas três arquiteturas a única que é específica para a simulação distribuída é a HLA. O CORBA e o RMI são arquiteturas genéricas para aplicações distribuídas.

A arquitetura HLA foi definida sob liderança do Defence Modelling Simulation Office (DMSO) para suportar o reuso e a interpolaridade atavés de um vasto número de diferentes tipos de simuladores mantidos pelo Departamento de Defesa Americano (DoD). Esta arquitetura é definida por tês padrões da IEEE: o primeiro trada do famework de forma geral e suas principais regras [IEEE 2000], o segundo diz respeiro a especificação da interface entre os simuladores [IEEE 2010] e o terceiro trata do modelo para especificação dos dados (OMT) transferidos entre os simuladores [IEEE 2000].

O HLA tem como idéia principal, oferecer uma estrutura de prosito geral que tem a finalidade de interligar qualquer tipo de simulador disponibilizando sincronização, gerenciamento de tempo além de manter uma semantica unificada para troca de mensagens entre os simuladores. Os simuladores que desejam se integrar a uma simulação que utiliza o HLA, devem utilizar uma Runtime Infraestructure (RTI) para que seja possível realizar a comunicação com os outros simuladores.

A RTI é a infra-estrutura que realiza o controle das trocas de mensagens, sendo esta, responsável pelo gerenciamento do tempo total de simulação, alé de forcecer o protocolo de comunicação. Através dele, todos os simuladores terão acesso a estrutura global do HLA.

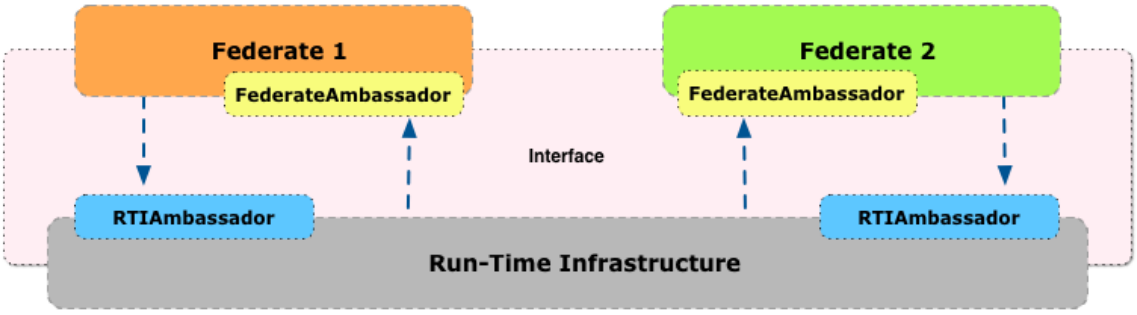

Figura 2.2: Federação genérica com dois federados.

Como pode ser visto na Figura 2.2 os simuladores que são conectado a uma RTI é chamado de Federado. É denominada federação os federados que atuam em conjunto em uma simulação visanto alcançar um objetivo comum. A federação pode conter um ou vários fede-

rados, os federados são os componentes ou modelos de simulação, podendo ser simuladores, dispositivos de hardware, modelos em softwares ou algum modelo de propósito específico. A interterface, denominada Runtime Interface Specification padroniza a forma que os federados interagem com o RTI, seja para responder requisições recebidas através do RTI ou realizar a requisicão de serviços.

A especificação da HLA define três principais componentes, a interface (HLA Interface Specification - IS), o modelo de objeto (Object Model Template - OMT) e as regras HLA (HLA Rules).

- **Regras do HLA:** Especifica como devem ser realizadas as interações entre os federados em uma federação além de definir as responsabilidades das federações e federados.
- **Objetc Model Template (OMT):** Basicamente o OMT define a estrutura necessária para cada Federation Object Model (FOM). o FOM prove uma especificação para a troca de todos os dados através de um formato comum e padronizado.
- **Interface Specification:** Especifica a interface entre os federados e o RTI.

2.4.1 Regras do HLA

As regras definem os objetivos e restrições das federações e federados.

No desenvolvimento das federações, estas devem ser definidas no "HLA Federation Object Model" (FOM). Os objetos associados na simulação devem obrigatoriamente estar associado aos federados e não na RTI. Toda a comunicação de dados entre federados devem ocorrer através do RTI. Quando uma federação entrar em execução, todos os federados devem se comunicar com o RTI respeitanto a "HLA Interface Specification".

Basicamente os federados devem estar de acordo com o OMT e aptos para atualizar, refletir, transmitir e receber os atributos do objeto. Os federados também devem gerenciar o tempo local para que seja possível a realização da troca de dados entre outros intrgrantes da federação.

2.4.2 Grupos de Serviços do HLA

A especificação de interface HLA descreve os serviços que o federado pode usar para comunicar-se com outros federados via Runtime Infraestructure (RTI).

- **Federation management:** oferece serviços necessários para realizar as funções básicas de criar e operar uma federação;
- **Declaration management:** oferece suporte para a realização de troca de dados;
- **Object management:** oferece serviços que permitem especificar objetos de dados que serão enviados por meio de publicação ou subscrição.
- **Ownership management:** oferece serviços para que um federado possa prover valores de um atributo de instância de objeto;
- **Time management:** oferece suporte a sincronização entre federados.
- **Data distribution management:** oferece suporte a distribuição eficiente de dados entre os federados durante a execução da federação.

  A interface do HLA está encapsulada dentro do ambassador. Sendo assim os federados se comunicam com o RTI através de seu ambassador RTI. Já o RTI se comunica com o federado por meio do ambassador federado.

2.4.3 Modelo de Objetos (OMT)

Esta especificação do HLA descreve os modelos de objetos que integram o HLA. O Simulated Object Model (SOM) especifica e trata dos tipos de informação de aplicação federada individual pode receber ou oferecer das federações. O Federation Object Model (FOM) especifica o compartilhamento de informações em tempo de execução, informações estas que podem incluir classes, atributos de objetos de classes, conteúdo MOM, entre outros. E por fim o Management Object Model (MOM) que especifica os construtores que proporcionam o suporte para monitorar e controlar a execução das federações.

2.4.4 Comunicação com o HLA

Diversas ferramentas vêm sendo desenvolvidas para facilitar a implementação das simulações distribuídas. O padrão IEEE 1516 que especifica a High Level Architecture (HLA) é amplamente utilizado. Como pode ser visto em: Georgia Tech FDK [FDK 2014], MAK RTI [MAK 2014], Pitch RTI [Pitch 2014], CERTI Free HLA [Certi 2013], OpenSkies Cybernet [OpenSkies 2014].

Com base em experiencias obtidas através de trabalhos anteriores [Brito et al. 2013] [Negreiros e Brito 2012] [Negreiros e Brito] [Brito, Vieira e Nascimento 2015], neste trabalho foi utilizado uma implementação chamada CERTI RTI [Certi 2013]. Essa implementação foi desenvolvida por um grupo francês de pesquisas aeroespaciais chamada ONERA. Após a escolha da implementação é necessário que sejam também escolhidas as bibliotecas necessárias para a integração no ambiente, uma vez que existem bibliotecas em diferentes linguagens de programação. Os trabalhos [Brito et al. 2013] [Negreiros e Brito 2012] [Negreiros e Brito] [Brito, Vieira e Nascimento 2015] utilizaram a biblioteca JCERTI (escrita em JAVA) nos componentes do Ptolemy, permitindo assim que os modelos desenvolvidos no Ptolemy se conectassem ao HLA através do RTI.

Visando demonstrar uma heterogeneidade entre federados, uma implementação em Python da biblioteca foi utilizada neste trabalho. A PyHLA [Nongnu 2014], possibilitou que federados fossem escritos na linguagem Python. Ela foi usada especificamente no desenvolvimento do Robô Móvel. Diante dessa plataforma CERTI e as bibliotecas JCERTI e PyHLA foi possível unir dois mundos diferentes.

2.5 Ptolemy II Framework

O projeto Ptolemy suporta a modelagem, projeto e simulação de qualquer tipo de sistemas baseados em atores, principalmente sistemas embarcados que mesclam tecnologias, além de sistemas com complexidade nos vários níveis de abstração [Eker et al. 2003]. Os Atores são modelos visuais que são implementados na linguagem Java e realizam o comportamento que foi especificado. Este projeto foi criado e é mantido na Universidade de Berkley. O Ptolemy é um framework desenvolvido na linguagem Java. Ele possui uma interface gráfica chama vergil, que permite o desenvolvimento de hardware de forma gráfica. Um grande diferencial

desta ferramenta é o suporte ao desenvolvimento de modelos heterogêneos, permitindo que vários tipos de atores realizem interações em um ambiente único [Lee e John 1999].

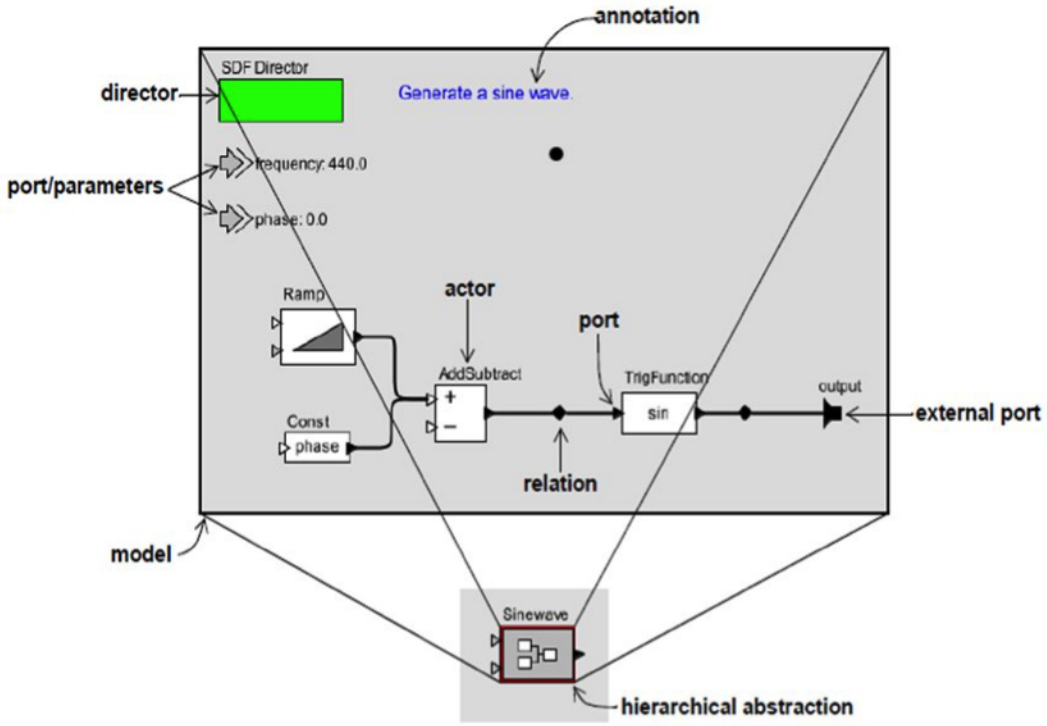

Figura 2.3: Modelo genérico de um ator composto e seus componentes.

Este framework apresenta algumas características importantes, tais como possibilitar o uso de hierarquia entre modelos. Modelos podem ser desenvolvidos utilizando diversos tipos de domínios e modelos computacionais (MoC) tais como, Discret Events (DE), Synchronous Data Flow(SDF), Continuous Time (CT), entre outros. Permite também a execução de modelo baseados em XML. Outro ponto importante é que seu código é aberto e gratuito, fazendo com que pesquisadores sejam encorajados a desenvolver seus próprios modelos além de expandir a infraestrutura que é fornecida.

A Figura 2.3 é apresentada no trabalho [Brooks et al. 2005]. Ela apresenta um modelo genérico de um ator composto. Os atores compostos podem conter vários outros atores dentro dele, ou seja, é possível ter atores com MoCs diferentes em um mesmo ambiente. O Diretor é responsável por especificar o modelo de computação que será usado no projeto. As portas podem ser de entrada ou saída, e é utilizada para manter os relacionamentos entre os atores e permitir a troca de dados entre eles.

Capítulo 3

Verificação de projetos de sistemas embarcados através de cossimulação hardware/software

3.1 Aquitetura do ambiente de cossimulação

Para desenvolver um ambiente de cossimulação é necessário definir como os diversos simuladores irão se comunicar, além de oferecer um mecanismo de sincronização e também especificar como as mensagens serão trocadas entre os integrantes da cossimulação. No ambiente desenvolvido, foi utilizado o HLA já que ele fornece todos os elementos necessários para possibilitar a integração/sincronização de simuladores heterogêneos, como visto no capítulo 2.

Quando se trabalha com HLA, os simuladores são chamados de federados e estes devem utilizar uma implementação da Runtime Infraestructure (RTI) que seja comum a todos, como por exemplo o CERTI RTI. Para possibilitar o acesso ao RTI, cada federado deve possuir uma interface que possibilite gerenciar as entradas e saídas da federação, troca de dados e sincronização, como é o caso do *RTI Ambassador*. Através do RTI Ambassador, os federados podem invocar os serviços do RTI, um serviço bastante utilizado é o da requisição de atualização dos valores dos atributos dos objetos.

Como pode ser observado na Figura 3.1, o federado (*PtolemyFederate*) também possui uma interface chamada *FederateAmbassador* que possibilita a comunicação entre os fede-

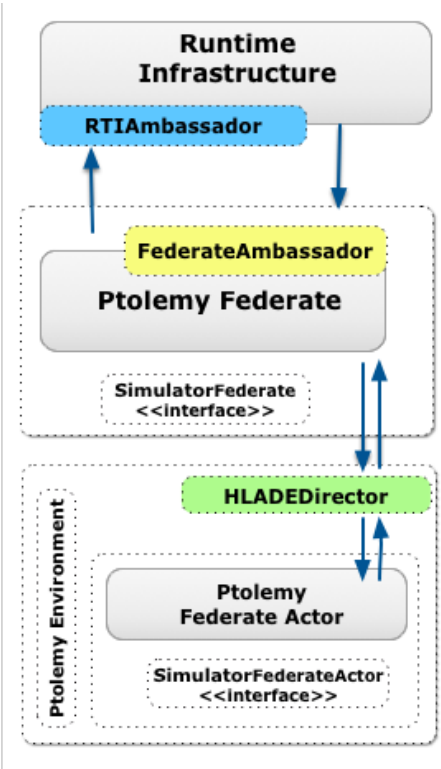

Figura 3.1: Arquitetura do Federado Ptolemy.

rados. Ou seja, através de trocas de mensagens com o RTI, um federado pode atualizar informações nos outros federados da simulação, como por exemplo, o repasse de um novo valor de algum atributo. De maneira geral o RTIAmbassador define serviços mais gerais do HLA, enquanto o FederateAmbassador define serviços mais especificos da federação.

A Figura 3.1, mostra a arquitetura do federado Ptolemy. É nele onde o ambiente do Ptolemy (Ptolemy Environment) faz toda a comunicação com o HLA. Para isso foi necessário desenvolver o módulo *HLADEDirector*, que é um diretor baseado em Discret Event. Este diretor é responsável por coordenar a ordem e a execução dos atores dentro do domínio, verificando quando deve enviar um dado do RTI para o ator e também do ator para o RTI. Esta integração entre o modelo em software (Ptolemy) com o HLA pode ser visto com detalhes em [Brito et al. 2013].

Conforme a Figura 3.2, os federados *PtolemyFederate* e *HardwareFederate* podem ser conectadas por meio de uma conexão LAN/WAN. Ambos federados utilizam a API de Java jCerti, sendo assim, tanto o RTIAmbassador quanto o FederateAmbassador são interfaces Java.

O federado PtolemyFederate é a responsável pela simulação e verificação de modelos

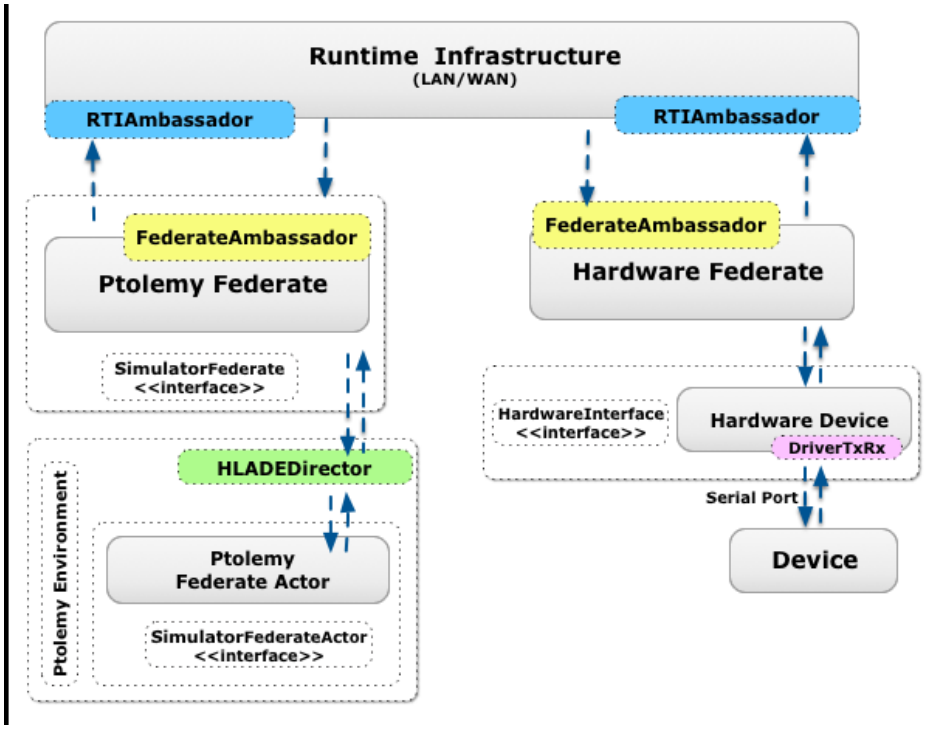

Figura 3.2: Arquitetura do PtolemyFederate conectado ao HardwareFederate.

heterogêneos (vide Cap. 3.2), nele está contido toda a lógica que proporciona o ambiente de verificação. Teoricamente ele pode ser considerado um projeto mestre, pois ele é o responsável por iniciar a simulação e fica aguardando que os módulos de hardware(escravo) iniciem também a simulação para que seja iniciada a verificação.

Já o federado *HardwareFederate* é responsável pela integração entre o hardware físico com o modelo de software, esta abordagem possibilita a interligação de quaisquer dispositivos de hardware, consequentemente possibilitando uma simulacão através de Hardware-in-the-loop (HIL).

Para poder realizar a comunicação de qualquer dispositivo de hardware com o modelo de software, foi necessário desenvolver uma interface de comunicação e a padronização de um protocolo de dados. O *Hardware Device* é um módulo que deve ser implementado de acordo com cada dispositivo. Na sua implementação deve conter uma instância da classe *DriverTx/Rx* que é a interface responsável pela aquisição dos dados vindos do dispositivo de hardware. Ela detecta automaticamente a porta e o id do hardware que está conectado na porta USB, no entanto há a necessidade de inserir previamente as informações de cada dispositivo. Como a maioria dos dispositivos se utilizam de uma comunicação serial via USB, esta interface será totalmente compatível com qualquer dispositivo, bastando apenas a

implementação do protocolo.

A idéia principal do protocolo (Fig. 3.3) é que a mensagem fique entre dois bytes previamente definidos como início e fim, possibilitando assim, não fixar um tamanho para a mensagem. Com base na idéia apresentada, foi necessário desenvolver o protocolo de forma simplificada para viabilizar o funcionamento dos experimentos do estudo de caso, porém a interface permite que sejam desenvolvidos outros protocolos mais complexos.

Figura 3.3: Protocolo de dados.

Dentre os dispositivos testados, o Arduino Mini, Freescale Freedom KL25Z, BeagleBone Black e Raspberry Pi, todos funcionaram perfeitamente, não foi necessário nenhuma mudança na interface, necessitou apenas que o protocolo que foi especificado fosse implementado na arquitetura alvo.

Um terceiro federado foi desenvolvido visando possibilitar a simulação com robôs reais através de Robot-in-the-loop (RIL).

Diferente do PtolemyFederate e HardwareFederate, este federado não foi implementado em Java. Foi usada a implementação do RTI PyHLA, que é implementado em python, demonstrando assim uma heterogeneidade de federados. Como pode ser visto na figura 3.4

Para possibilitar a padronização das mensagens foi necessário criar um arquivo FED. O arquivo FED especifica um tipo de objeto que descreve os dados que serão compartilhados entre os federados. Isso significa que eles podem ser lidos e/ou escritos. Vale resaltar que todos federados envolvidos na simulação terá que fornecer a mesma arquitetura com o arquivo FED compatível. O FED utiliza uma descrição relativamente simples, para mais detalhes sobre as suas regras de sintaxe podem ser obtidas em [IEEE 2010].

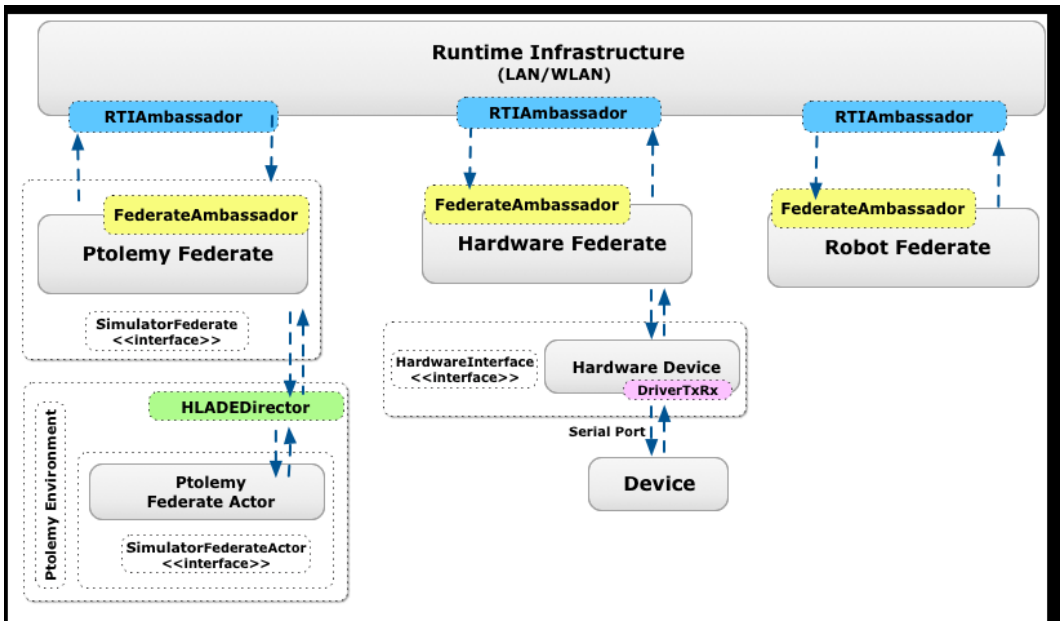

Figura 3.4: Arquitetura do PtolemyFederate conectado ao HardwareFederate.

Código Fonte 3.1: Arquivo de configuração do objeto .FED

```
1  (FED
2    (Federation TestFed)
3    (FEDversion v1.3)
4    (spaces
5    )
6    (objects
7      (class ObjectRoot
8        (attribute privilegeToDelete reliable timestamp)
9        (class RTIprivate)
10       (class robot
11         (attribute battery reliable timestamp)
12         (attribute temperature reliable timestamp)
13         (attribute sensor1 reliable timestamp)
14         (attribute sensor2 reliable timestamp)
15         (attribute sensor3 reliable timestamp)
16         (attribute gps reliable timestamp)
17         (attribute compass reliable timestamp)
18         (attribute goto reliable timestamp)
19         (attribute rotate reliable timestamp)
20         (attribute activate reliable timestamp)
21       )
22     )
23   )
24   (interactions
25   )
26 )
```

- *Battery:* Variável responsável por compartilhar porcentagem da bateria do robô, esta informação pode permitir ao simulador saber quando uma simulação deve terminar por existir um nível baixo de bateria do robô ou algum dispositivo de hardware;

- *Temperature:* Usado para monitorar a temperatura do robô. Esta informação é usada porque alguns componentes são sensíveis à certos níveis de temperatura. Algumas vezes quando têm uma alta temperatura que necessitam desligar para evitar a perda de informação. Outra aplicação é quando o hardware ou o robô está monitorando a temperatura ambiente;

- *Sensor1, Sensor2 and Sensor3:* São sensores genéricos, que permite ao programador escolher qual tipo de sensor será utilizado;

- *GPS:* Compartilha a posição do robô;

- *Compass:* Compartilha as informações da bússola;

- *Goto:* Este dado é responsável para enviar comandos para robô. Esses comandos são usados para alterar a posição do robô;

- *Rotate:* É usado para acionar os motores do robô. Com base no ângulo e sentido informado o robô se movimenta;

- *Activate:* Responsável para ativar ou desativar os atuadores de cada robô. É útil para ativar as luzes, motores ou outros dispositivos.

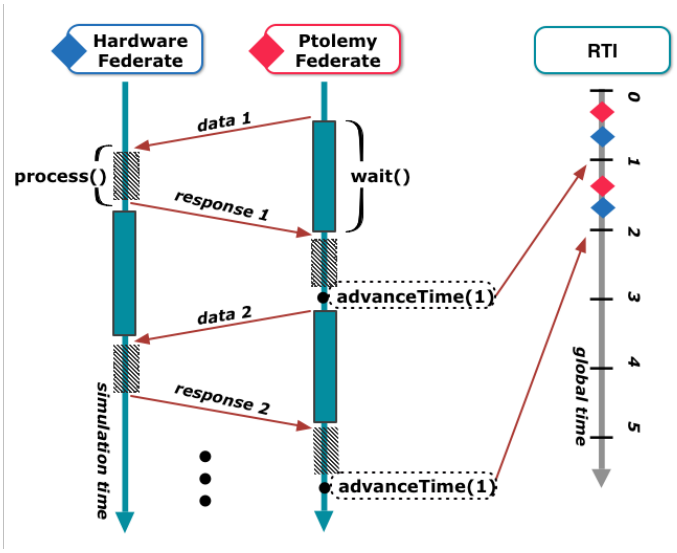

Figura 3.5: Modelo de Sincronização utilizado.

3.1.1 Modelo de sincronização

A comunicação entre os federados é gerido pelo RunTime Infrastructure (RTI) através de uma conexão TCP/IP, permitindo assim, que a simulação possa ser executada através da Internet, seja em redes locais ou através de redes geograficamente distribuídas. Naturalmente, a distribuição ao longo de grandes redes impõe atrasos mais longos e, portanto, limita a dinâmica do sistema embarcado em teste que a cossimulação pode lidar.

O ambiente de verificação foi desenvolvido inicialmente para realizar uma simulação funcional, a abordagem utilizada no desenvolvimento foi a conservadora, no entanto nada impede que seja realizada uma simulação otimista. No HLA, por meio do Time Management que é gerenciado pelo RTI, é disponibilizado uma forma de avanço de tempo chamada “Time stepped Simulation”, seu funcionamento faz com que o tempo não adiante para o próximo passo de tempo até que todas as atividades de simulação com o tempo atual global sejam concluídas.

Na Figura 3.5 mostra que o PtolemyFederate faz inicialmente o envio de um dado (evento) para o HardwareFederete, e logo fica a espera de uma resposta para que possa processar a informação. O HardwareFederate inicialmente fica a espera de um dado, que ao receber realiza o devido processamento fazendo em seguida o envio da informação para o PtolemyFederate, e logo após ficando a espera de um novo evento.

Após cada ciclo de eventos (send, wait, process) o PtolemyFederate faz a chamada da função advanceTime(), que avisa ao RTI o momento para que o tempo global da simulação

seja avançado, garantindo assim que todos os eventos sejam ordenados de acordo com o timestamp de envio em relação ao tempo global.

Sincronização entre os federados

Embora o HLA seja originalmente baseado em simulações conservadoras, existem soluções para abordagens otimistas [Wang et al. 2004]. Na sincronização otimista a RTI sempre tenta manter o tempo de simulação global sincronizado com os tempos locais, mas não impede que cada tempo local avançe seu tempo mais rapidamente. Com roll-back, no caso de um evento tardio chegar na fila global, os componentes mais rápidos retornam aos seus estados anteriores e só então, o evento anterior é processado. Em alguns casos, os eventos tardios são ignorados, como acontece com simulações distribuídas interativas (DIS). Isso faz com que a sincronização otimistas seja mais adequada quando a consistência dos dados é menos importante que o avanço da simulação no que diz respeito ao tempo de relógio [Fujimoto 2000].

Já a sincronização conservadora é mais apropriadas para simulações analíticas, quando os resultados utilizando várias máquinas devem ser os mesmos que os obtidos a partir de uma única máquina [Fujimoto 2000]. Neste caso, os relógios locais deve sempre exigir a permissão RTI para avançar a hora local. Essa autorização só é concedida quando é seguro para avançar o tempo local de todos os federados, garantindo um tempo global consistente.

Em nossa abordagem, temos ambos os tipos de sincronização. O robô atua de um modo interativo, enquanto é monitorado e verificado por meio de uma simulação analítica. No entanto, como o alvo principal é o teste de algoritmos que funcionam no robô, foi usado a abordagem conservadora. Assim, o tempo que decorre na simulação pode não corresponder ao tempo do robô no cenário real. Sendo assim, o tempo de simulação e o desempenho do sistema são ditadas pela federação mais lenta, que por sua vez determina o tempo real da cossimulação. Por esta razão cossimulando sistemas embarcados, tais como robôs, no nosso caso, pode exigir o uso de uma velocidade mais lenta do robô se o computador de simulação incluindo comunicações é mais lento do que o controlador do robô. Com base nisso, o uso da abordagem conservadora é o suficiente, pois para o caso de verificação, não importa a precisão do tempo, apenas a consistencia dos dados.

Isto não representa um problema, no nosso caso, uma vez que é apenas uma prova de

conceito da abordagem cossimulação proposta para o teste de um planejamento de trajetoria. No entanto, no caso geral, é indesejável ou impossível ter os componentes de hardware abrandado pelo modelo de simulação e, assim, uma estratégia de sincronização optimistas sem retrocesso deve ser considerada uma plataforma ou outra simulação adequada deve ser utilizado, que permite satisfazer as limitações de tempo imposta pelo hardware.

3.2 Arquitetura do modelo de verificação proposto

A etapa da verificação funcional é considerada a fase de maior importância no desenvolvimento de projetos de sistemas embarcados. Nesta fase é possível detectar erros lógicos do sistema, isso faz com que erros sejam detectados nas etapas iniciais do projeto, fazendo com que e eles não sejam repassados para etapas futuras. As metodologias tradicionais de verificação funcional consistem basicamente em injetar um conjunto de estímulos de entrada em um hardware, para que os resultados esperados que já são conhecidos com antecedência, sejam comparados ao resultado do modelo de referência [Zatt et al. 2006].

No entanto quando se trata de sistemas heterogêneos, que contém tanto modelos em software quanto em hardware, este processo de obtenção das entradas e coleta das saídas não é algo simples, devido sua dependência em relação a tecnologia. Por isso é proposto um ambiente genérico e simplificado de verificação funcional. Utilizamos alguns conceitos da *Universal Verification Methodology* (UVM) [Accellera 2011], em particular sobre o modelo testbench.

Como se sabe, a maioria dos testbench possuem um gerador de estímulo, um modelo de referência, um DUT e um checker. Com base nisso foi desenvolvido o modelo conceitual, e que pode ser visto na Figura 3.6. O que se diferencia dos modelos tradicionais é o DUT, pois este possui outro ambiente ferramental que permite a ligação de qualquer tipo de hardware, seja por HIL ou RIL, além de modelos tradicionais.

Como pode ser observado na Figura 3.6, todo o ambiente de verificação está inserido no ambiente do Ptolemy, possibilitando assim que diversos tipos de sistemas possam ser verificados em um ambiente único de cossimulação. O Ptolemy possibilita um ambiente de simulação heterogênea, onde é possível ser modelado vários sistemas em diferentes níveis de abstração, que podem ser usados como Modelo de Referência. A heterogeneidade entre

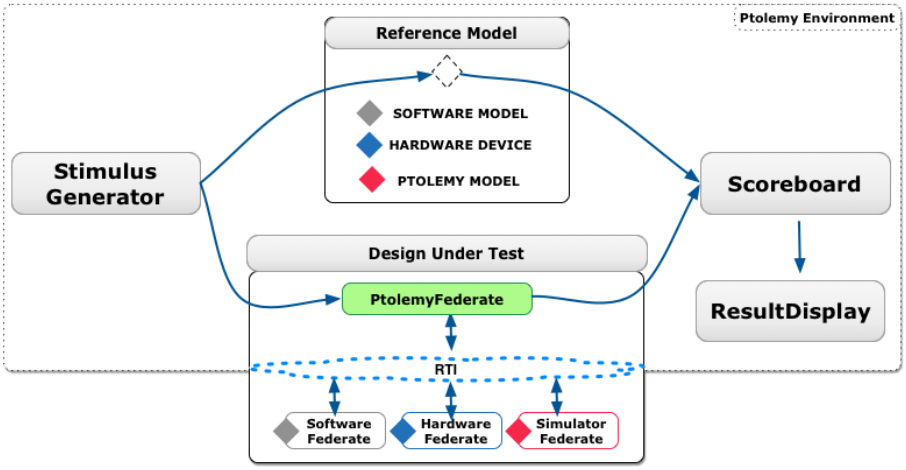

Figura 3.6: Arquitetura do ambiente de cossimulação proposto.

os simuladores (externo ao Ptolemy) é garantida através da integração entre o Ptolemy e o HLA.

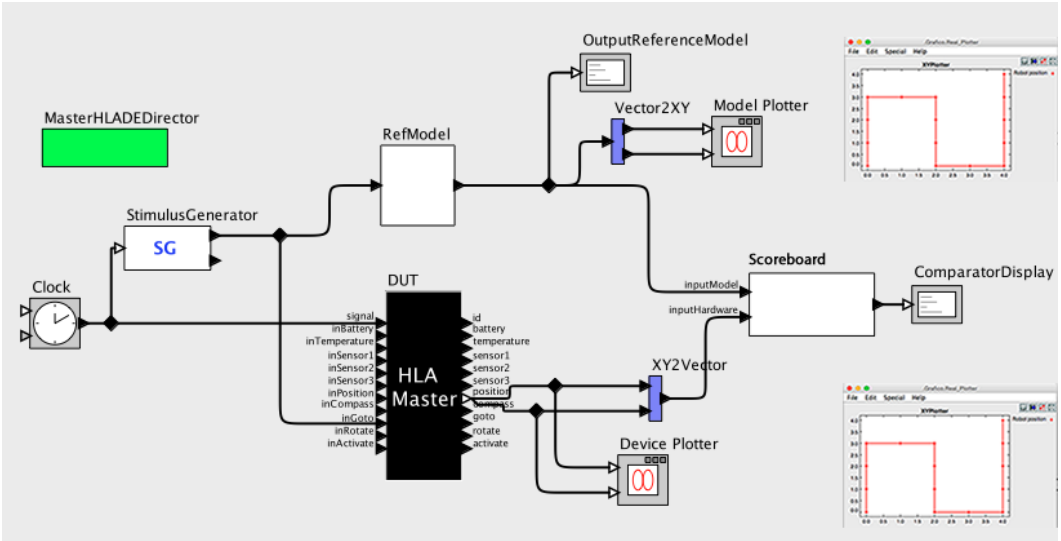

Figura 3.7: Modelo do ambiente de cossimulação proposto.

No ambiente do Ptolemy (ver Figura 3.6) há o MasterHLADEDirector, que administra a simulação do Ptolemy respeitando políticas baseadas em eventos discretos, coordenando a ordem de execução dos atores no domínio, verificando quando enviar os dados dos atores para a RTI e também do ator para a RTI. O DUT (HLA Master) é o ator que representa a comunicação com a RTI dentro do modelo do Ptolemy. Todos os dados enviados para ele antes passa pelo RTI, que faz o envio para a Federação apropriada.

Cada sistema que for adicionado ao ambiente, deve ser desenvolvido baseado no modelo de atores. Os atores são módulos que processam os dados presentes nas suas portas de entrada ou criam e enviam dados para outras entidades por meido se suas portas de saída. O Ptolemy possibilita modelar e simular sistemas embarcados utilizando diferentes modelos computacionais, enquanto o HLA disponibiliza meios para integrar tecnologias heterogêneas em tempo real, de forma síncrona.

No ambiente proposto, o DUT *(Design Under Test)* é um ator que realiza uma comunicação direta com o hardware. Ele recebe estímulos e gera saídas, que são comparadas com os resultados esperados recebidos a partir do modelo de referência. É no DUT que a implementação do hardware está sob verificação integrado ao modelo de simulação por meio do HLA.

Como pode-se observar na Figura 3.6, alguns elementos do testbenh foram implementados no Ptolemy, onde cada ator representa uma etapa do fluxo do testbench proposto, são elas:

- *Stimulus Generator:* É o ator responsável por enviar os dados pré-definidos armazenados em um arquivo que servirá como entrada para o os modelos do sistema. Estes dados podem vir do próprio ambiente Ptolemy ou de um hardware físico;

- *Reference Model:* Modelo da aplicação que pode ser um software, hardware ou modelo escrito no Ptolemy, e que está em conformidade com a especificação;

  Como pode ser visto na Figura 3.8 a ferramenta permite que sejam implementados inúmeros algoritmos para que sejam selecionados quando forem necessários. O Ptolemy possibilita essa facilidade, pois em apenas 2 clicks o modelo de referencia pode ser mudado.

- *Scoreboard:* Responsável pela coleta e análise dos dados enviados pelo Reference-Model e pelo DUT. Após a recepção dos dados é feita uma comparação, levando em consideração o tempo de atraso e a igualdade dos dados;

Código Fonte 3.2: Algoritmo de comparação (Scoreboard)

```
1   public static <K extends Comparable<? super K>, V> Map<K, Boolean>
        compareEntries(
```

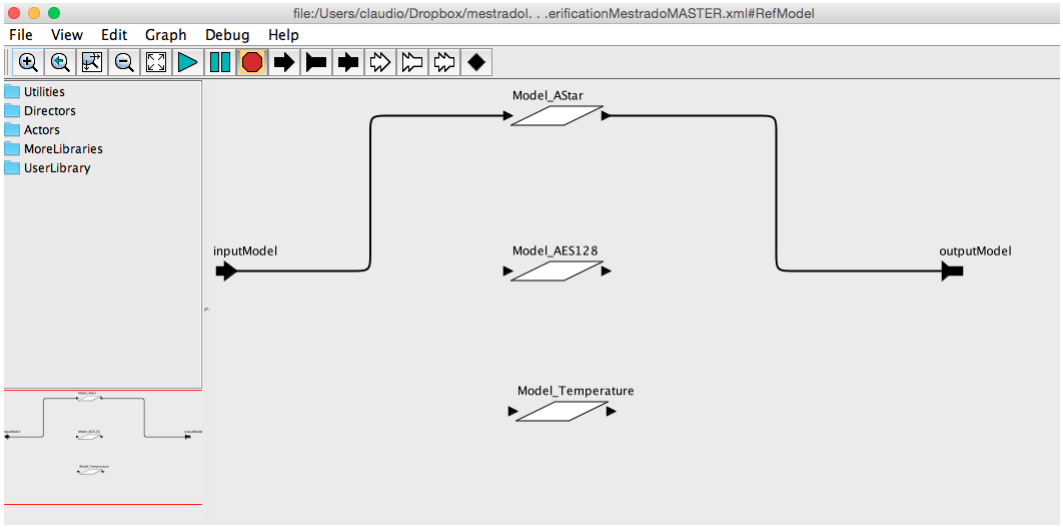

Figura 3.8: Selecionando algoritmo para no modelo de referencia.

```
2                   final Map<K, V> map1, final Map<K, V> map2) {
3                   final Collection<K> allKeys = new HashSet<K>();
4                   allKeys.addAll(map1.keySet());
5                   allKeys.addAll(map2.keySet());
6                   final Map<K, Boolean> result = new TreeMap<K, Boolean>();
7                   for (final K key : allKeys) {
8                           result.put(key, map1.containsKey(key) == map2.containsKey(key
                                )
9                                           && Boolean.valueOf(equal(map1.get(key), map2.
                                                get(key))));
10                  }
11                  return result;
12          }
```

Neste algoritmo, são recebidos como chaves o tempo do recebimento do dado, e também o valor recebido, ou seja, se o modelo A envia no tempo 1 o dado 30 e o modelo B no tempo 1 envia o dado 35, os índices que serão comparados é o do tempo, como, modeloA[1]=30 igual a modeloB[1]=35, neste caso para o tempo 1 os dados não foram equivalentes.

- *Comparator Display:* Responsável pela exibição dos resultados que foram gerados pelo ScoreBoard. É possível observar se o modelo foi validado ou não, mostrando em quais períodos de tempo os resultados se divergem;

- *Design Under Test:* O DUT normalmente é uma descrição do dispositivo a ser desenvolvido, codificado em alguma linguagem de descrição de hardware HDL *(Hardware Description Language)* ou simulado em alguma ferramenta. Porém o que propomos é que no DUT esteja sob verificação um dispositivo de hardware real. Aqui é onde o simulador Ptolemy se comunica com os outros simuladores ou hardware através do HLA.

Capítulo 4

Avaliação Experimental

Este capítulo trata dos resultados obtidos a partir dos experimentos realizados após o desenvolvimento do ambiente proposto. Inicialmente será apresentado como foi feito o desenvolvimento de três experimentos. O primeiro trata de um simples exemplo com a finalidade de validar o ambiente. O segundo experimento é um exemplo mais elaborado, pois lida com um algoritmo de criptografia e um grande número de dados. O terceiro experimento é mais contextualizado e também mais sofisticado, pois trata da verificação de um robô móvel real.

4.1 Experimento 1: Validação do modelo proposto

Neste primeiro experimento foi utilizado uma implementação em software para conversão de temperatura. A aplicação recebe um valor referente a temperatura na escala Fahrenheit e retorna o valor convertido na escala Celsius. Este cenário demonstra uma troca de dados entre federados de forma síncrona, garantindo assim o funcionamento da integração entre o Ptolemy e o HLA. Nesta simulação foram utilizados dois federados, um rodando em uma máquina X e outro rodando em uma máquina Y. A máquina X representa o ambiente de cossimulação e o modelo de referência, já a maquina Y representa o modelo em hardware.

Neste experimento os federados podiam trocar apenas mensagens de um tipo (String), através de apenas um canal, conforme Figura 4.1. Esta limitação, não chegou a ser um problema, já que somente era necessário enviar um número, e receber outro número.

É apresentado na Figura 4.2 os resultados obtidos a partir do ambiente de cossimulação desenvolvido. Nela é mostrado o resultado da verificação funcional a partir da cossimulação

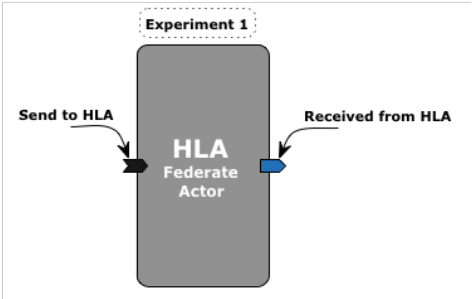

Figura 4.1: Ator Ptolemy HLA com 1 entrada e 1 saída.

de um modelo escrito na plataforma de hardware, na qual espera-se que esteja de acordo com a especificação com um modelo de software que está em conformidade com a especificação. Então neste caso, espera-se que o ambiente retorne como resultado uma saída válida para que de fato seja comprovado a equivalência entre o modelo escrito em software *(modelo referência)* e o modelo em hardware.

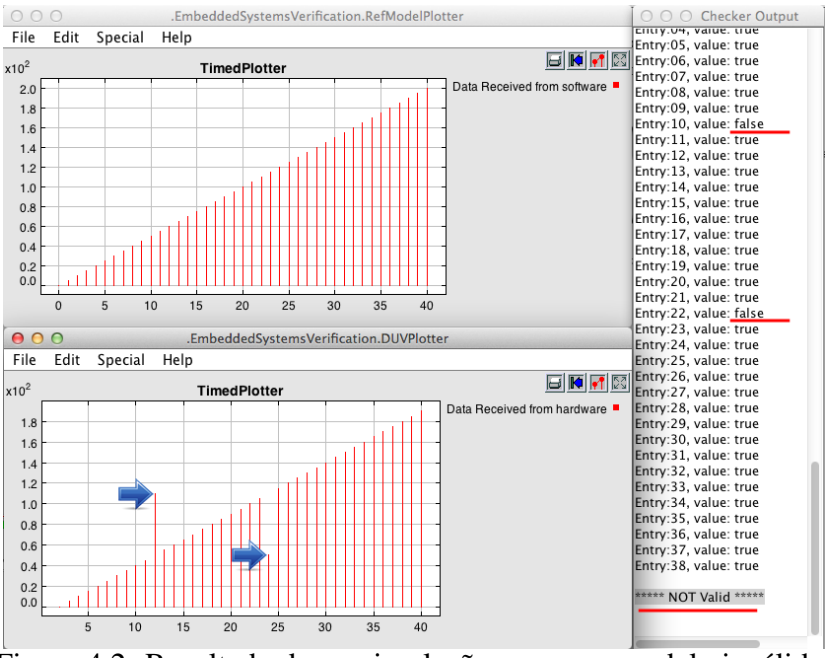

Figura 4.2: Resultado da cossimulação, com um modelo inválido.

No entanto, visualmente no gráfico, percebe-se que ambos não são idênticos. O *"Scoreboard Output"* exibe na janela o resultado completo da simulação, levando em conta o tempo e o delay, ambos os modelos precisam estar sincronizados, caso contrário, os dados irão divergir, gerando assim uma mensagem que modelo não foi validado.

O cenário da Figura 4.2, onde o modelo não é válido, foi criado justamente para testar se a ferramenta conseguia detectar quando os modelos não fossem equivalentes. Na janela

do gráfico *"Data received from hardware"* que representa os dados recebidos pelo modelo de hardware, as duas setas indicam os resultados que foram enviados intencionalmente de forma errada, diferentemente da janela do gráfico *"Data received from software"* que mostra os dados de forma correta.

Na Figura 4.3 é apresentado o resultado da simulação sem nenhuma indução ao erro, e com isto, pode ser observado que os dados recebidos dos modelos são equivalentes.

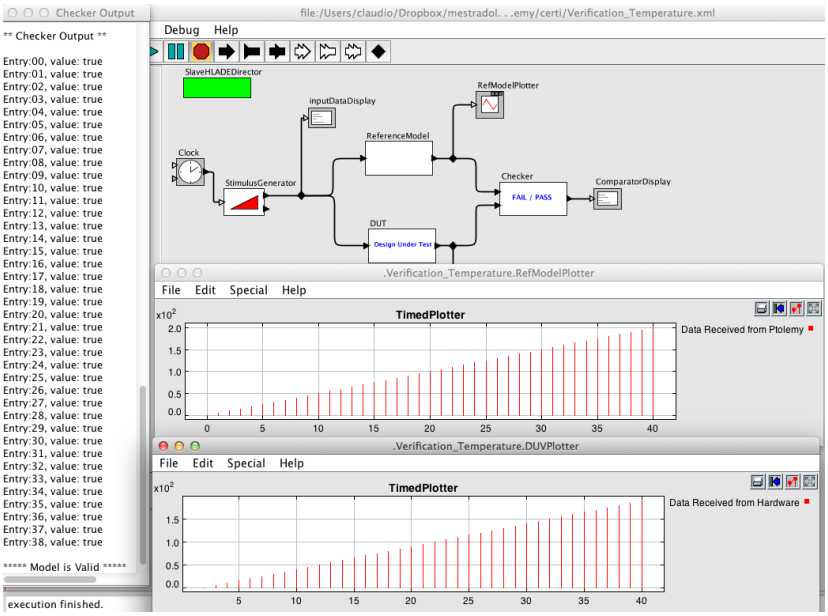

Figura 4.3: Resultado da cossimulação, com um modelo válido.

Como pode ser observado na Figura 4.1 o federado possibilita apenas o recebimento e o envio de apenas um dado por vez, já que o mesmo possui apenas um canal para o recebimento de mensagens e outro para o envio. Isto pode ser um problema, pois quase nunca se verifica apenas uma variável/dado ou comportamento de um hardware. No entanto o ambiente se mostrou bastante eficaz em todos os testes que foram realizados. Mesmo quando foram inseridos um grande número de dados o ambiente se comportou de forma síncrona. O mais importante neste experimento não é sua importância para os contextos do mundo real, mas sim a constatação da sincronização entre os dois modelos, sendo este um ponto essencial quando se realiza uma cossimulação.

4.2 Experimento 2: Verificação através de cossimulação de um sistema embarcado

Diferentemente do primeiro experimento onde se utilizava apenas computadores para os modelos de hardware e de software, este segundo experimento faz a utilização das placas de hardware Arduino Mini e a Freedom KL25Z para serem o modelo de hardware.

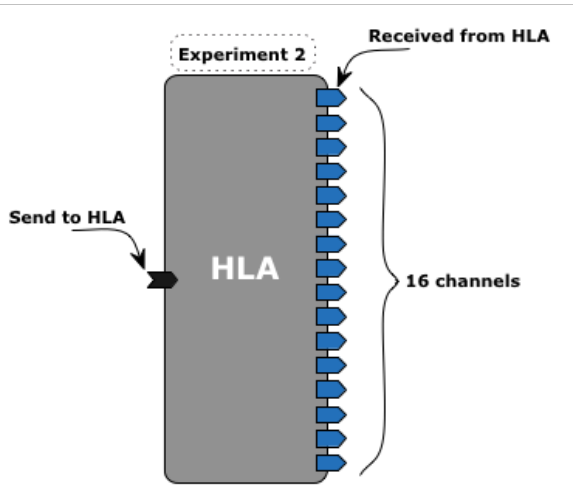

Figura 4.4: Ator Ptolemy HLA com 1 entrada e 16 saídas.

Se for observado a Figura 4.1 em comparação com a Figura 4.5, nota-se que esta segunda possui 16 canais para comunicação de dados, isso possibilita verificar o comportamento apenas de parte do hardware ou diversas partes dependendo da modelagem do sistema.

Após os avanços no desenvolvimento do ambiente, foram realizados novos testes utilizando como aplicação o algoritmo de criptografia AES128. O AES *(Advanced Encryption Standard)* também conhecido como Rijndael, é uma cifra de bloco adotada como padrão de criptografia pelo governo dos Estados Unidos. É um modelo bastante utilizado tanto em sistemas de softwares quando em sistemas de hardware, seja para proteger dados pessoais, trafego na rede ou informações sigilosas. Atualmente está sendo muito utilizado em sistemas embarcados críticos [Pigatto], existindo assim, a necessidade de ser implementado em diversos tipos de hardware.

Foi definido uma chave padrão de 16 bytes 1122334455667788 e as mensagens a serem criptografadas eram pré-definidas e enviadas aleatoriamente.

Na Figura 4.5 pode ser observado o resultado após a criotografia dos dados, no lado esquedo da figura (Stimulus Data) são os dados que foram enviados pelo gerados de estimulos,

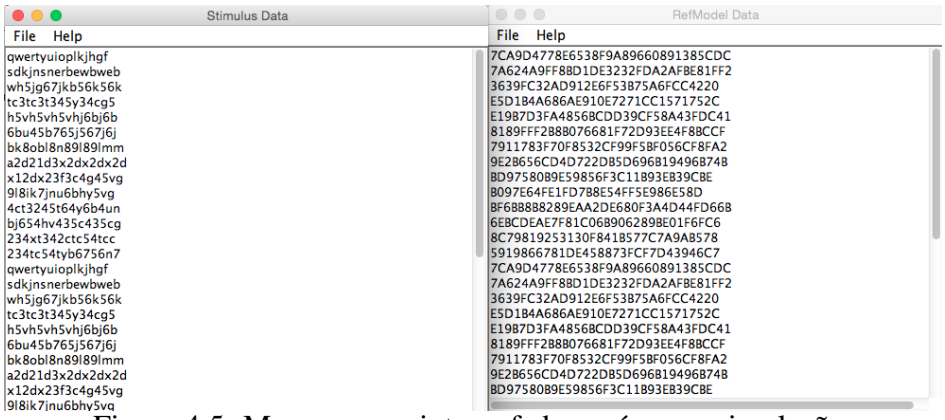

Figura 4.5: Mensagens criptografadas após a cossimulação.

e no lado direito (RefModel Data) é o resultado da criptografia. É importante frisar que para manter uma padronização nos dados, os resultados da criptografia são em Hexadecimal.

Como já foi dito, neste segundo experimento foram utilizadas as plataformas Arduino Mini e o Freedom KL25Z como modelo de hardware e as aplicações escritas na linguagem Java como modelo de software. Conforme a tabela 1, os modelos de hardware e software foram utilizados tanto para servir como modelo de referência quanto para ficar sob verificação. Todas as implementações foram desenvolvidas por diferentes autores, usando diferentes linguagens. Para realizar a implementação das aplicações no Arduino foi utilizado a linguagem Wiring, já no Freedom foi utilizado a linguagem C.

Tabela 4.1: Modelos de hardware e software utilizados.

Reference Model	**Design Under Test**		
	Arduino	Freedom	Java
Arduino	-	X	X
Freedom	X	-	X
Java	X	X	X

Posteriormente foi colocado como modelo de referencia o ator Rijndael que já vem implementado no ptolemy (ver Figura 4.6). No entando foi necessário realizar adaptações para que os dados se saída fossem equivalentes ao ambiente de cossimulação.

Por não apresentar dados apenas numéricos, não foi possível plotar os resultados graficamente, nesta simulação o resultado final foi apenas retorno do Scoreboard, que informa se os dados foram equivalentes ou não.

Esta segunda prova de conceito foi um exemplo mais contextualizado em relação ao primeiro, tendo em vista que utiliza um algoritmo de criptografia. E por permitir ligação do

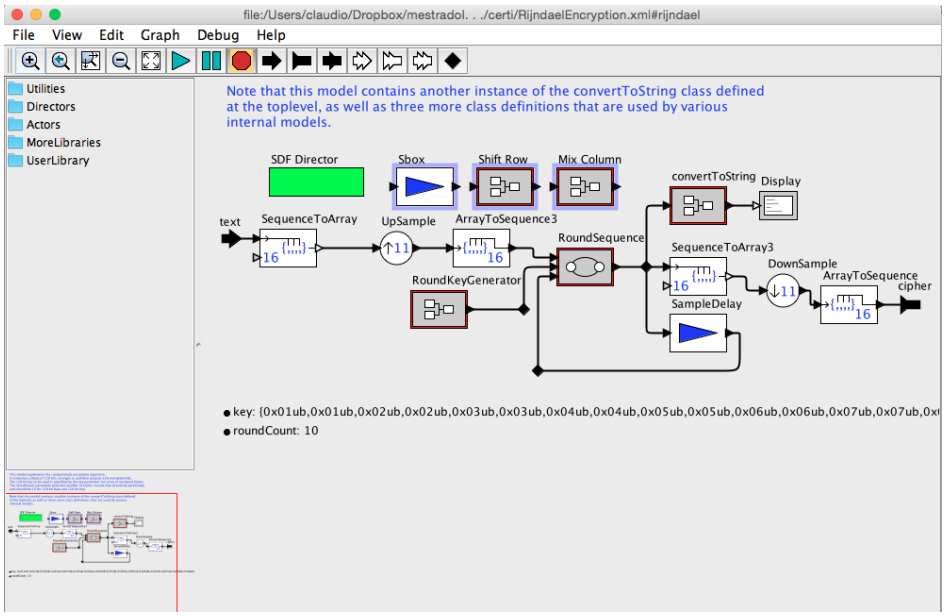

Figura 4.6: Modelo Rijndael do Ptolemy adaptado para AES128.

Arduino e o Freedom KL25Z no ambiente de cossimulação, fica claro o suporte a modelos de Hardware-in-the-loop. No entanto o modelo se limita a apenas um dado de entrada, podendo não ser muito útil para verificar modelos reais.

4.3 Experimento 3: Teste de um robô utilizando Hardware-in-the-loop

Como pode ser observado na Figura 4.7 este modelo permite a inclusão de N portas de entrada de dados e N portas para saída. Este é um modelo próximo do ideal, já que possibilita a verificação de inúmeras portas, possibilitando assim, a verificação de sistemas reais.

Para este terceiro experimento, o robô consiste apenas em uma placa Arduino, que será responsável por gerar os estímulos. Onde a partir de sua interação com o ambiente ele envia os dados necessários para que seja feita uma verificação. Neste exemplo o robô se resume a um Arduino integrado a um módulo sensor ultrassônico para captar obstáculos e sua trajetória está sendo gerada apartir de um software.

Na Figura 4.8 é apresentado a tela principal do ambiente de verificação. A comunição entre os módulos é realizada través do ator "Sender", que é um ator composto, dendro dele, está o módulo que é responsável pela captura dos dados do RTI.

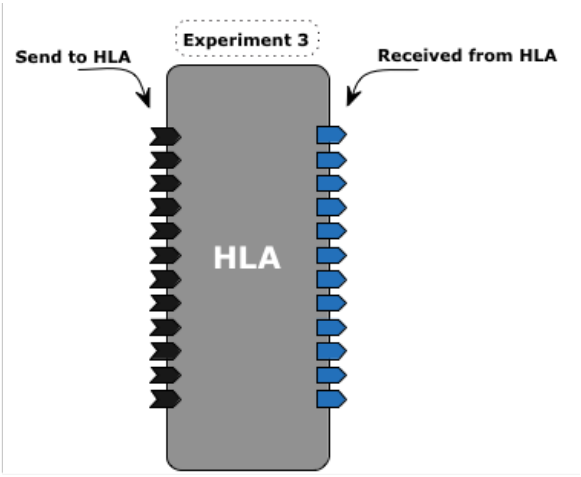

Figura 4.7: Ator Ptolemy HLA com N entradas e N saídas.

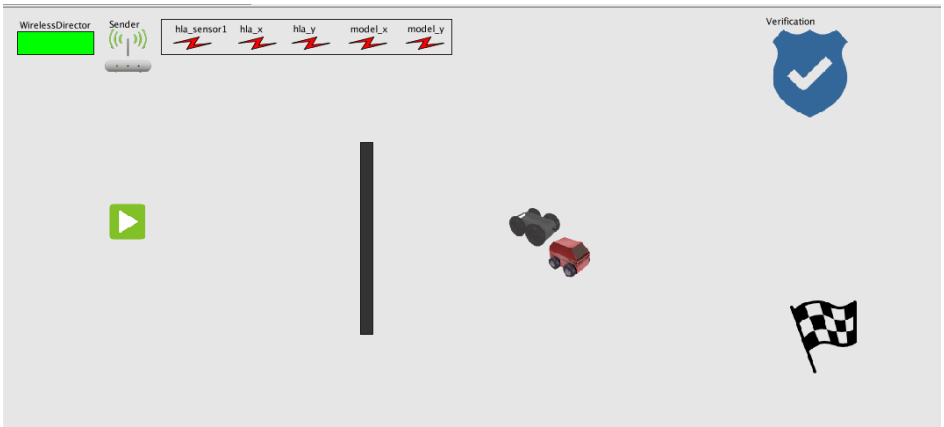

Figura 4.8: Tela Principal do Ambiente de Verificação.

Diferentemente dos experimentos anteriores esse novo modelo é capaz de receber vários tipos de dados em diferentes portas, isso possibilita a verificação de uma parte isolada ou completa do sistema. No caso deste exemplo foram usadas duas portas, a sensor1 que estava conectada do um sensor ultrassônico e a outra chamada gps que contém as coordenadas referente a posição do robô.

Inicialmente o robô fica transmitindo sua posição e o valor lido do sensor ultrassônico. Ao detectar um obstáculo, tanto o robô quanto o modelo de referencia devem desviar do obstaculo. Todo o percurso realizado por ambos modelos podem ser verificados na tela (ver Figura 4.8). O algoritmo de posicionamento que está sendo emulado no Arduino poder ser observado a seguir.

```
1 //initialization of variables:
2 //gotoX = true;
3 //gotoY = true;
4 public void controlLoop(double sensor1) {
```

```
5     if (hasObstacle(sensor1) == true) {
6        if (gotoX) {
7           actualPosY += step;
8           gotoX = false;
9        } else {
10          actualPosX += step;
11          gotoX = true;
12       }
13    }
14    else if (gotoX) {
15       if (actualPosX < targetPosX) {
16          actualPosX += step;
17       }
18       if (actualPosX > targetPosX) {
19          actualPosX -= step;
20       }
21    }
22    else{
23       if (actualPosY > targetPosY) {
24          actualPosY -= step;
25       }
26       if (actualPosY < targetPosY) {
27          actualPosY += step;
28       }
29    }
30    if (actualPosX == targetPosX) {
31      gotoX = false;
32    }
33    if (actualPosY == targetPosY) {
34      gotoX = true;
35    }
36    if ((actualPosX == targetPosX) && (actualPosY == targetPosY)) {
37         // Target reached!
38    }
39    //send actual position to RTI
40    outputPosX.send(0, new DoubleToken(actualPosX));
41    outputPosY.send(0, new DoubleToken(actualPosY));
42
43    //advance global time, when authorized by RTI
44    ambassador.advanceTime(1.0);
45 }
```

Nesse modelo o robô envia os dados através do HLA para o Ptolemy que repassa os dados recebidos para o ator Scoreboard que é responsável pela validação dos resultados. Posteriormente os resultados são exibidos gráficamente para uma possível comparação visual, como

poder ser visto na Figura 4.9 os valoes recebidos tanto do Robô quanto do outro modelo foram idênticos, confirmando assim que ambos estão conforme a especificação. Devido a especificação do robô deste experimento e por não possuir motores não é possível detectar erros de trajetória causados por fatores externos, como atrito por exemplo. Neste exemplo apenas é possível detectar erros de implementação da especificação.

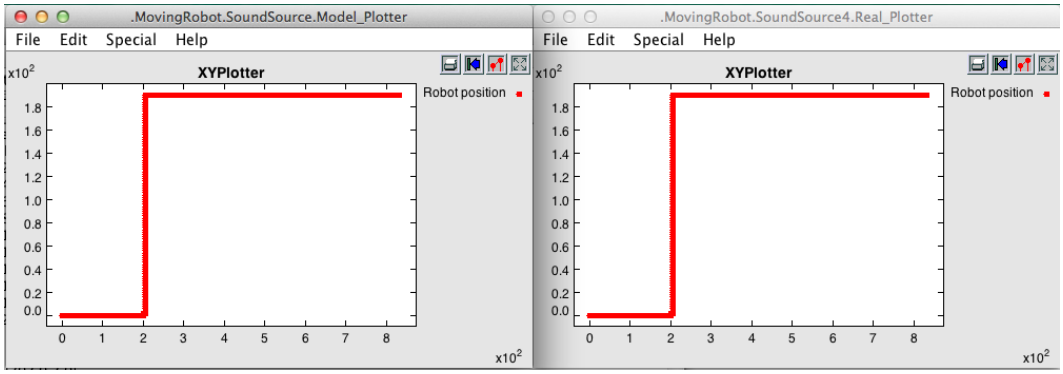

Figura 4.9: Resultado da cossimulação, com um modelo válido.

Neste

4.4 Experimento 4: Teste em tempo real de um robô móvel utilizando Robot-in-the-loop

4.4.1 Desenvolvimento do Robô Móvel

Para possibilitar uma simulação com um robô real afim de realizar Robot-in-the-loop, foi necessário desenvolver um robô móvel. Para isso, foi utilizado um chassi, uma Raspberry Pi e um driver L298N que é necessário para o acionamento dos motores DC. A ligação dos componentes pode ser visualizada na Figura 4.10. Essas ligações foram as mais complicadas em relação ao desenvolvimento da parte física do robô.

A parte operacional do robô consiste em: Embarcar o certi HLA mais especificamente o pyHLA na plataforma; Desenvolver a federação, e o Federate Ambassador em python, que seja equivalente aos do que foram desenvolvidos em Java(Experimentos anteriores). A Equivalencia entre os federados garantem a sincronização e acesso ao RTI.

Conforme a Figura 4.11 o driver L298N foi ligado nos seguintes pinos: pinENA = 33, pinIN1 = 35, pinIN2 = 37, pinIN3 = 36, pinIN4 = 38, pinENB = 40, 5V = 4 e GND = 39.

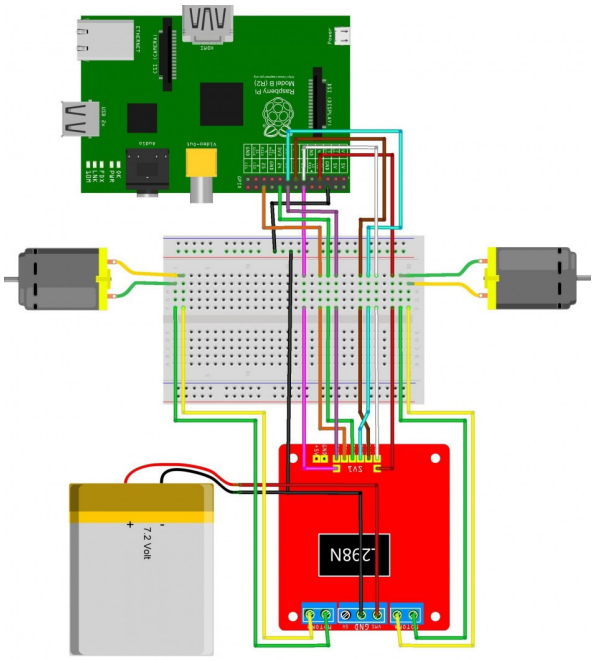

Figura 4.10: Ligações entre a Raspberry, Driver e motores DC.

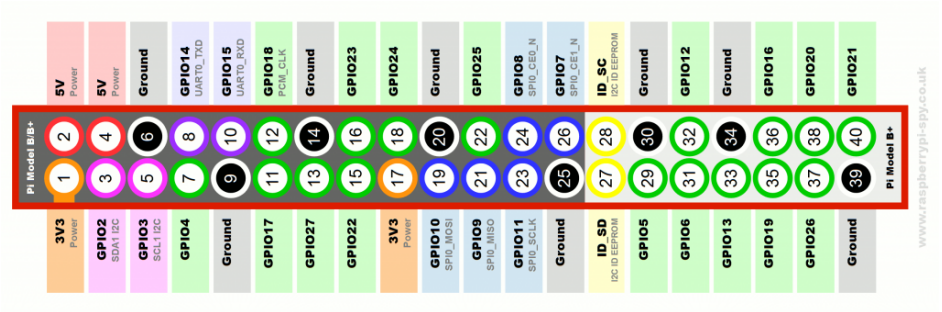

Figura 4.11: GPIO da Raspberry Pi.

Já o sensor ultrassonico foi ligado aos pinos: TRIG = 11 e ECHO = 12. E o Led RGB aos pinos: pinLEDSingle = 7, pinLEDR = 15, pinLEDG = 16, pinLEDB = 18.

Após toda configuração da parte física o resultado final do robô ficou conforme a Figura 4.12. Como pode ser observado, ele possui um sensor ultrassônico mas não possui nenhum tipo de sensor para odômetria, com isso, se faz necessário realizar um controle via software de posicionamento além de realizar uma calibração dos motores.

4.4.2 Definição do Experimento

Foi utilizado um algoritmo de planejamento de trajetória para que fosse validado pelo sistema, já que eles são amplamente utilizados na robótica móvel. A idéia principal é permitir que um robô possa encontrar um caminho ótimo ou não, entre um ponto de partida e o seu

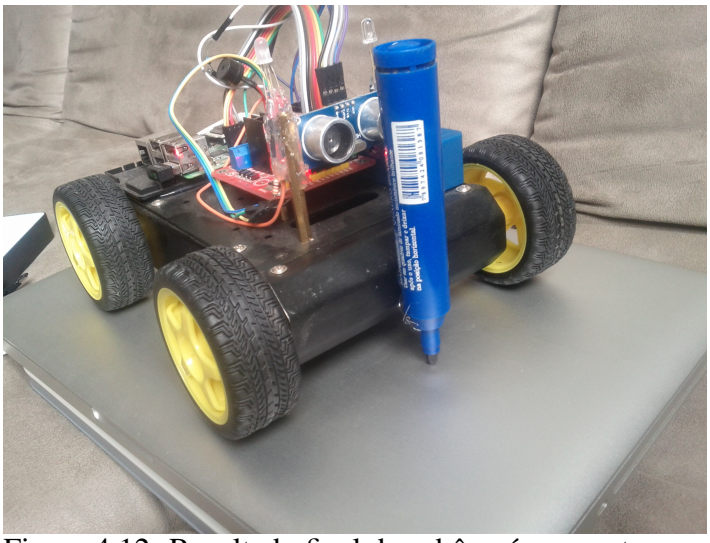

Figura 4.12: Resultado final do robô após a montagem.

objetivo. O algoritmo usado nesse experimento foi o A*, o qual é uma derivação do Dijkstra, um clássico na área de planejamento de trajetória.

O A* é um algoritmo de planejamento de trajetória e percorrimento de grafos. O mesmo foi criado por Peter Hart, Nils Nilsson e Bertram Raphael do Instituto de Pesquisa de Stanford em 1968 [Hart, Nilsson e Raphael 1968]. O grande diferencial do A* dentre os outros algoritmos, é o uso de heurísticas, o que reduz o tempo computacional de forma significativa, sem perder a acurácia. O A* usa best-first search e encontra o caminho de menor custo de um nó inicial até um nó objetivo. À medida que o A* percorre o grafo, ele constrói uma árvore de caminhos parciais. As folhas dessa árvore são chamadas de open list e são guardadas em uma fila de prioridade a qual ordena as folhas através de uma função de custo. A função de custo 4.1 combina uma estimação heurística do custo pra chegar até o objetivo e da distância percorrida do nó inicial até o atual. Onde, para um dado nó "x", $f(x)$ é o custo, $g(x)$ é a distância percorrida até o momento e $h(x)$ é a heurística (Distância de Manhattan, Distância Euclidiana, etc).

$$f(x) = g(x) + h(x) \tag{4.1}$$

Enquanto o algoritmo percorre o grafo, os nós "x" com menor valor para $f(x)$ terão maior prioridade para serem visitados. Com isso, ao término da execução, o A* será capaz de determinar o caminho com menor custo entre dois pontos.

Neste experimento tanto o robô quanto o modelo de referência, deve conhecer o ponto de

partida, o ponto alvo e também todos os obstaculos, ou seja, ele deve conhecer previamente o mapa, para a partir daí o robô calcular uma trajetória. A cada passo o robô deve enviar via RTI seu posicionamento e o resultado da leitura do sensor ultrassônico. Possibilitanto assim, que o sistema faça uma intervenção ou não no seu posicionamento. Cada passo ou movimento do robô foi definido como uma unidade de tempo, ou seja, um movimento em linha reta é representado por 1s de acionamentos dos motores. Já movimentos com ângulos de 90 graus, representa 2s de acionamento dos motores.

Foram realizados vários testes utilizando 2 mapas diferentes (ver Figura 4.13), o primeiro com tamanho 5x5, o segundo 6x6, no entanto este último só foi realizado uma verificação funcional, onde se desejava apenas validar os posicionamentos recebidos pelo robô e pelo modelo de referência. Além de validar o algoritmo de controle de posicionamento e acionamento dos motores do robô.

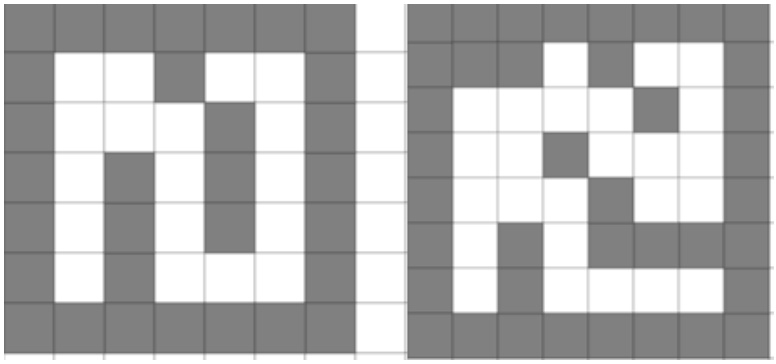

Figura 4.13: Mapas 12x12, 6x6 e 5x5 que foram utilizados nas simulações.

Nesta abordagem de teste pretende-se testar os modelos em tempo real que está em execução no hardware contra seus respectivos modelos de referência desenvolvidos no Ptolemy. Todos os dados obtidos pelos sensores e as respectivas reações são transferidas em tempo real para o Ptolemy, que realiza o análise dos dados recebidos contra um modelo de referência.

4.4.3 Análise do Tempo

A fim de verificar se a cossimulação satisfaz as limitações de tempo de um determinado sistema de tempo real (RTS), particularmente quando se utiliza uma abordagem de sincronização conservadora, é importante realizar uma analise dos atrasos envolvidos.

Considerando um evento que ocorre em um determinado instante de tempo de clock t_{action}, que é capturado com um certo atraso d_{RT}. Isto implica que o instante em que o

evento é observada é dado por t_{observ} como se segue:

$$t_{obs} = t_{action} + d_{RT} \tag{4.2}$$

Em geral, um limite superior para d_{RT} pode ser calculado usando a seguinte equação:

$$d_{RT} = max(t^{emb}_{cycle}, t^{PT}_{cycle}) + C^{emb}_{RTI} + t^{RTI}_{cycle} + C^{RTI}_{PT} \tag{4.3}$$

onde,

$$\begin{aligned} &t^{emb}_{cycle}\text{: ciclo de controle no sistema embarcado} \\ &C^{emb}_{RTI}\text{: tempo de comunicação do dispositivo para o RTI} \\ &t^{RTI}_{cycle}\text{: ciclo de simulação do RTI} \\ &C^{RTI}_{PT}\text{: tempo de comunicação do RTI com o Ptolemy} \\ &t^{PT}_{cycle}\text{: ciclo de simulação do Ptolemy} \end{aligned} \tag{4.4}$$

O valor máximo de d_{RT} determina a aplicabilidade da abordagem de cossimulação para o teste de sistemas em tempo real. Basicamente, o máximo de d_{RT} deve estar dentro das limitações de tempo do sistema físico associado. Por sua vez, o tempo de comunicação entre os federados (C^{emb}_{RTI} e C^{RTI}_{PT}) depende da tecnologia de rede utilizada e os ciclos de simulação dependem principalmente do número de eventos de ser processado em cada ciclo. Estes eventos estão relacionados com o número de componentes em ação em cada ciclo. Em particular, t^{RTI}_{cycle} é dependente do número de federados na simulação, enquanto t^{PT}_{cycle} baseia-se no número de agentes no modelo e t^{emb}_{cycle} depende das ações executadas pelo sistema embarcado em cada ciclo de controle, tais como a aquisição de dados de sensores, a execução do algoritmo de controle e atuação. O trabalho de [Brito et al. 2013] apresenta a análise de desempenho de um ambiente de simulação distribuída usando o Ptolemy, variando de 1 a 16 máquinas. Os resultados demonstraram a dependência de atraso na federação mais lenta e sobre o número de agentes em cada modelo de simulação.

4.4.4 Teste do robô em tempo real utilizando o mapa 5x5

Neste primeiro experimento foi utilizado o mapa 5x5. A Figura 4.14 representa o mapa e a trajetoria que deve ser realizada, onde o quadrado verde representa a posição inicial

do robô *startPosition = (0,0)* e o quadrado vermelho *endPosition = (4,4)* a posição final. Os obstaculos são representados pelos blocos cinza (*walls = ((1,0), (1,1), (1,2), (2,4), (3,1), (3,2),(3,3))*). Sendo assim, tanto o robô quanto o modelo de referencia devem ser alimentados com esses dados.

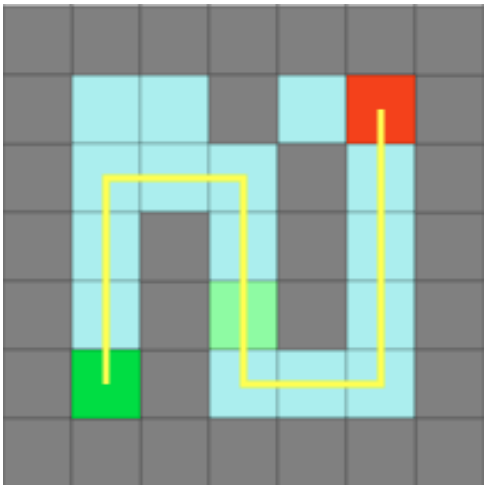

Figura 4.14: Mapa 5x5 exibindo a trajetoria.

Como resultado o robô deverá percorrer o caminho representados pelos pontos result=((0,0), (0, 1), (0, 2), (0, 3), (1, 3), (2, 3), (2, 2), (2, 1), (2, 0), (3, 0), (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)).

Um aspecto positivo da simulação RIL é que os distúrbios decorrentes ocasionador por interferências externas podem ser capturados na cossimulação. Neste caso, essa abordagem é capaz de capturar os erros de estimativa de posição feita pelo robô causada por atrito, imprecisão de sensores, motores descalibrados, etc. (ver Figura 4.16). Esses distúrbios podem mudar a percepção do robô sobre sua própria posição e não seria facilmente detectado com uma abordagem de verificação funcional.

O modelo de referência do robô é apresentado na Figura 4.15. Ambos os robôs, o real e o simulado recebem os dados iniciais para configuração, a partir do ator StimulisGenerator. Sempre que o robô móvel decidir mudar para uma nova posição e passar para o passo seguinte, ele envia para o RTI sua posição atual e uma rotina é chamada para o avanço de tempo global. Esta é uma chamada de bloqueio que retorna somente após a RTI tem assegurado a sincronização sintonia com o modelo de referência de execução, do Ptolemy. Em seguida, o robô avança sua hora local por uma unidade de tempo e repete o ciclo de controle.

Como pode ser observado na Figura 4.16, o robô apresentou um resultado que não era

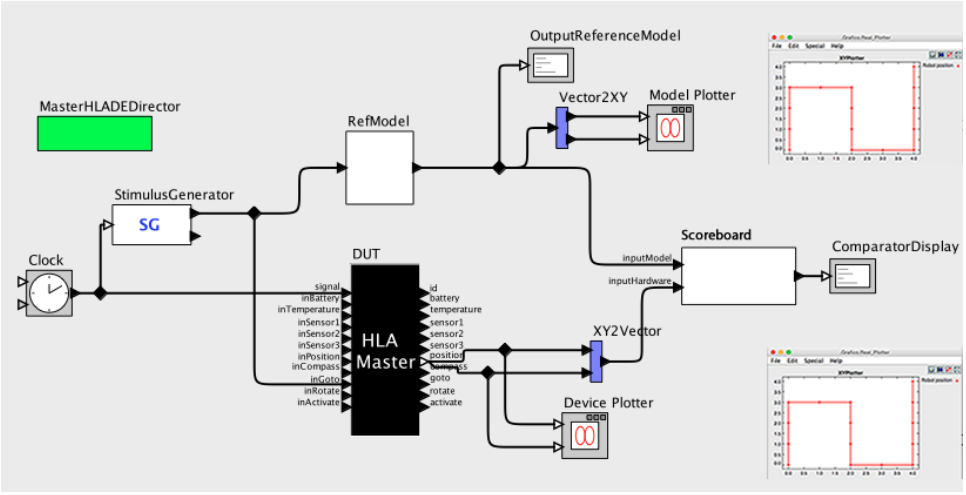

Figura 4.15: Ambiente de cossimulação no Ptolemy.

esperado, ou seja, ele e a simulação agiram de forma diferente. Os caminhos percorrido pelo robô real sofreu algum tipo de deslocamento indesejado, ou possívelmente os motores não estavam calibrados corretamente. Em um modelo de verificação tradicional, esse descolamento do robô real não é detectado, tendo em vista que a finalidade da verificação funcional é apenas verificar os dados. No entando quando se utiliza um ambiente de cossimulação estas divergencias entre o robô real e seu modelo de referência podem ser detectadas e classificadas como um erro.

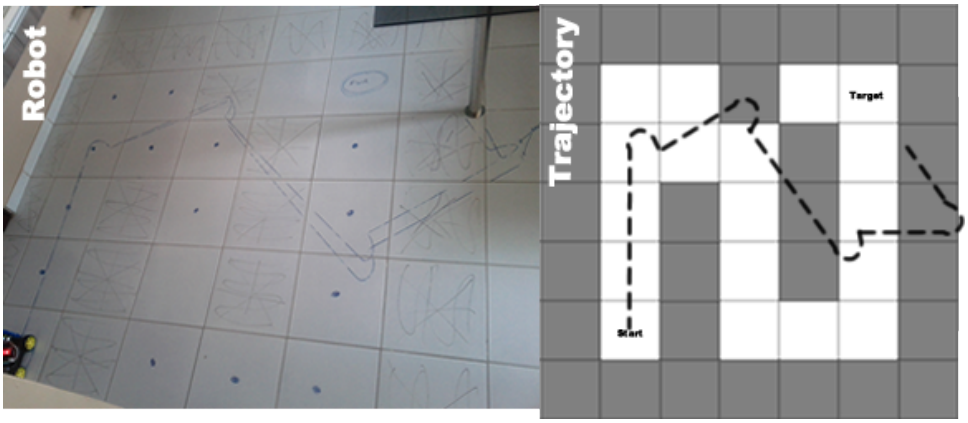

Figura 4.16: Real and simulated robots trajectories.

A Figura 4.16 mostra os resultados da simulação em que a verdadeira trajetória não se encaixa com o modelo de referência. Diante deste fato, o sistema de cossimulação proposto neste trabalho possibilita ver que o comportamento do robô é diferente do especificado no modelo de referência e possivelmente detectar e prever em falhas, possibilitando uma interação para algum tipo de ajuste.

Resultado da Simulação

Na Figura 4.17 pode ser observado que a execução da trajetoria do robô real e simulado seguiram trajetórias semelhantes.

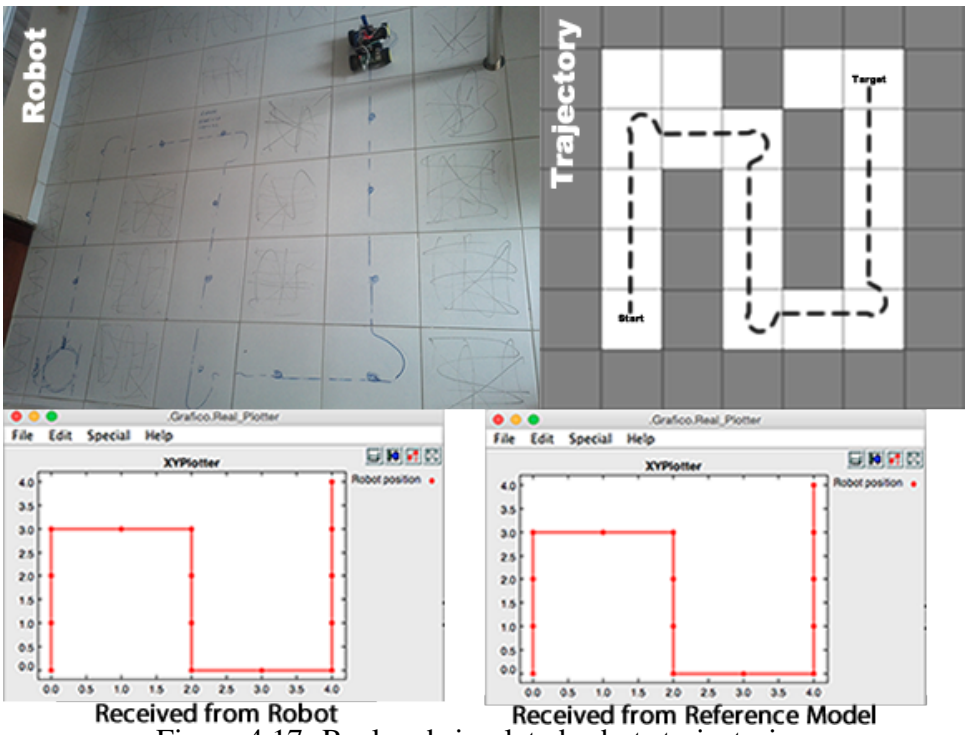

Figura 4.17: Real and simulated robots trajectories.

Como pode ser visto na Figura 4.17, ambos modelos apresentaram a mesma trajetória, representada pelos pontos *((0,0), (0, 1), (0, 2), (0, 3), (1, 3), (2, 3), (2, 2), (2, 1), (2, 0), (3, 0), (4, 0), (4, 1), (4, 2), (4, 3), (4, 4))*. A fim de alcançar este resultado, o robô teve de ser calibrado. Considerando que cada quadrado (cerâmica) possui 33 centímetros, os comandos Gofront() devem ficar ativos por 2 segundos. Já para movimentar o robô para o lado direito (rotacionar 90 graus) ou esquerdo , os motores devem ser ativados durante 1 segundo. A calibração deve ser realizada até que seja definido a distancia percorrida por segundo, ou seja, como cada grid possui 33cm, cada passo, o robô deverá andar 33cm. Esta abordagem demonstrou ser muito útil calibrar os robôs reais e simulados visando obter modelos mais precisos. Na verdade, utilizando um robô real como modelo de referência, é possível afinar o modelo de simulação e então, por sua vez, utilizar o modelo de simulação para calibrar novos robôs reais.

Análise dos Resultados

Com relação ao tempo, medimos os componentes do atraso descritos nas Equações 4.3 e

4.4. Mil ciclos foram executados e cada variável foi recolhida. A Tabela 4.2 mostra a média, máximo, mínimo e o desvio padrão de todas as execuções.

Tabela 4.2: Timing results (ms)

	Average	Maximum	Minimum	Std. Dev.
t_{cycle}^{emb}	107.3	137	100	5.47
t_{cycle}^{PT}	160.18	238	103	50.08
C_{RTI}^{emb}	6.5	8.6	5.3	0.3
t_{cycle}^{RTI}	6.4	7.2	5.8	0.2
C_{PT}^{RTI}	6.3	8.3	5.1	0.4
d_{RT}	179.38	262.1	119.2	50.98

Como esperado, o tempo de ciclo de simulação do Ptolemy é sempre maior do que o ciclo do sistema embarcado. A média do ciclo do Ptolemy (t_{cycle}^{PT}) é $160.18ms$ significando que em média, os eventos que acontecem no robô são recebidos pelo simulador após esse tempo. Em comparação, o ciclo de sistema embarcado (t_{cycle}^{emb}) em média foi de $107.3ms$. Isso significa que o atraso do ciclo completo da simulação d_{RT} para esse cenário será determinada pelo tempo do ciclo do Ptolemy, com média de $179.38ms$. Isso significa que robôs com ciclo de controle não superior a este valor poderão ser adicionados a este cenário sem que tenha uma diminuição significativa tempo de simulação.

Também é importante observar o desvio padrão de (t_{cycle}^{PT}) e (t_{cycle}^{emb}). O desvio do sistema embarcado é baixo, porque é um robô simples com uma tarefa em cada ciclo. Por outro lado, o tempo de ciclo do Ptolemy varia muito mais devido ao número de tarefas contínuas do sistema, ou seja, os atores em execução no modelo de simulação, os processos do sistema.

4.4.5 Verificação funcional do robô em tempo real

Visando validar o algoritmo de controle e posicionamento do robô foram realizado testes em diversos tipos de mapas. O Algoritmo de controle se baseia no posicionamento do plano cartesiano, onde se utiliza a posição X e Y. Para diferenciar a direção quando robô está posicionado no eixo X, foi utilizado a direção, ou seja, se está indo para o Norte ou para o Sul. Assim como no eixo Y, se está indo para o Leste ou Oeste. O algoritmo sempre é chamado quando o robô decide para qual coordenada deseja ir, sendo assim, o controle recebe a nova coordenada e indica quais motores o robô deverá acionar para poder chegar ao objetivo, que é o próximo passo.

Resultado Mapa 6x6

Como pode ser observado na Figura 4.18 e Figura 4.19 a posição inicial e final do robô são diferentes. E o robô realizou toda a trajetória de forma satisfatória.

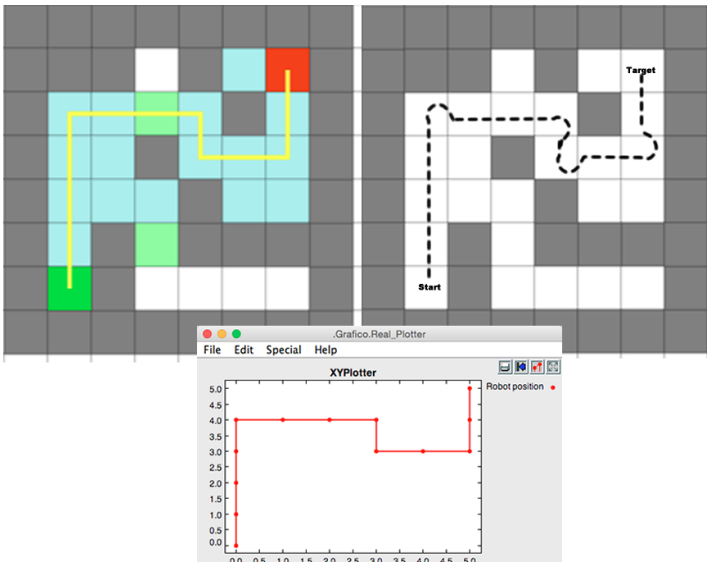

Figura 4.18: Resultado da simulação do cenário 6x6 modelo 1.

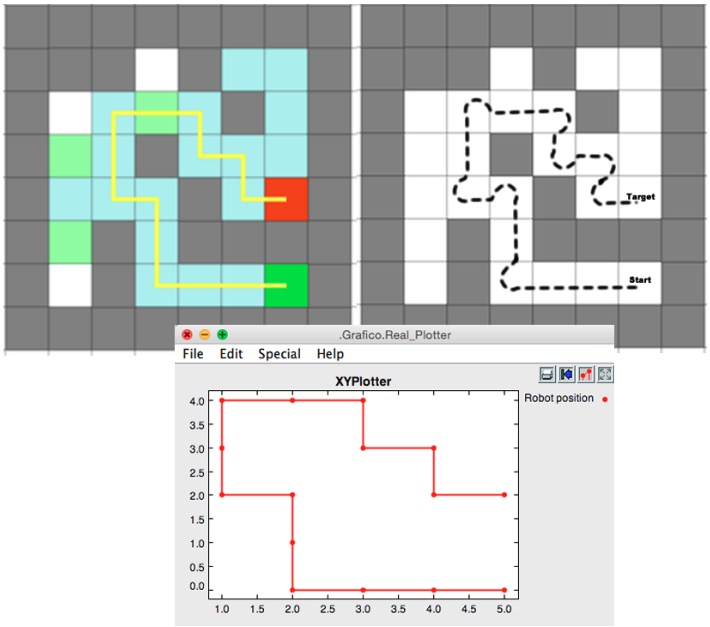

Figura 4.19: Resultado da simulação do cenário 6x6 modelo 2.

Capítulo 5

Conclusões e Trabalhos Futuros

Este trabalho propos uma metodologia de verificação e teste distribuída para sistemas embarcados através de cossimulação utilizando hardware-in-the-loop e robot-in-the-loop, o que permitiu o teste de dispositivos físicos que são afetados por fatores externos e que influenciam diretamente no seu comportamento. A utilização de ambientes distribuídos para verifição de sistemas embarcados pode oferecer uma maior flexibilidade aos projetistas, possibilitando um desenvolvimento descentralizado. O ambiente desenvolvido possibilita a verificação ou teste de sistemas robóticos, podendo ser verificados tanto o sistema embarcado quanto a parte física do robô. Concentrando-se em sistemas de tempo real, a abordagem verifica não apenas os dados gerados de saída mas também a sincronização, isto é, os instantes em que os dados foram enviados para a saída. Além disso, a abordagem também capta a reação do sistema de fatores ambientais externos, que apenas a cossimulação HIL e RIL permite.

O ambiente desenvolvido permite o teste de qualquer dispositivo de hardware que esteja em conformidade com a especificação da interface do HLA. Além disso, o HLA fornece uma comunicação síncrona e consistente com qualquer simulador através de TCP/IP. Ou seja, com esta abordagem de utilização do HLA, é possível a integração com outras estruturas, como, Matlab, VHDL, SystemC, etc. também é possível, além disso, qualquer tipo de robô pode ser utilizado, assim como qualquer outro simulador diferente do Ptolemy. No entanto, Ptolemy permite a integração de diferentes modelos de computação na mesma simulação e utilizando a mesma linguagem gráfica.

Para o melhor de nosso conhecimento, este é o primeiro trabalho a utilizar a integração

de HLA com Ptolemy para simulação e testes de hardware-in-the-loop e também robot-in-the-loop, considerando os dispositivos em tempo real. Isso abre um vasto conjunto de novas possibilidades, a partir do teste de muitos dispositivos que executam simultaneamente os mesmos ou diferentes aplicativos ou módulos, até usando Ptolemy para controle em tempo real de sistemas embarcados. Além disso, seria possível a cossimulação distribuída de múltiplos dispositivos embarcados para a melhoria do desempenho, ou apenas para a execução colaborativa.

Como trabalhos futuros pretendemos analisar mais detalhadamente o impacto do uso de diferentes abordagens de sincronização, a fim de reduzir o atraso da cossimulação e aproximar os comportamentos de dispositivos simulados e reais. Para melhorar ainda mais o ambiente de cossimulação, pode-se realizar a integração de motores de física na cossimulação, particularmente simuladores de robôs, para melhorar a precisão dos seus modelos de referência.

Bibliografia

[Accellera 2011]ACCELLERA, U. V. M. *Universal Verification Methodology I. 0 User's Guide*. 2011.

[Adams e Thomas 1996]ADAMS, J.; THOMAS, D. The design of mixed hardware/software systems. In: *Design Automation Conference Proceedings 1996, 33rd*. [S.l.: s.n.], 1996. p. 515–520. ISSN 0738-100X.

[Banzi 2009]BANZI, M. *Getting Started with arduino*. [S.l.]: O'Reilly Media, Inc., 2009.

[Bergeron 2003]BERGERON, J. *Writing Testbenches: Functional Verification of HDL Models, Second Edition*. Norwell, MA, USA: Kluwer Academic Publishers, 2003. ISBN 1402074018.

[Bergeron 2006]BERGERON, J. *Writing testbenches using system Verilog*. [S.l.]: Springer Heidelberg, 2006.

[Bergeron et al. 2005]BERGERON, J. et al. *Verification Methodology Manual for SystemVerilog*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005. ISBN 0387255389.

[Bishop e Loucks 1997]BISHOP, W.; LOUCKS, W. A heterogeneous environment for hardware/software cosimulation. In: *Simulation Symposium, 1997. Proceedings., 30th Annual*. [S.l.: s.n.], 1997. p. 14–22. ISSN 1080-241X.

[Brito et al. 2013]BRITO, A. V. et al. Development and evaluation of distributed simulation of embedded systems using ptolemy and hla. In: *17th IEEE / ACM International Symposium on Distributed Simulation and Real Time Applications*. [S.l.: s.n.], 2013.

[Brito, Vieira e Nascimento 2015]BRITO, A. V.; VIEIRA, J. C.; NASCIMENTO, T. Verification of devices for ambient assisted living through hardware-software co-simulation.

In: LACKOVIć, I.; VASIC, D. (Ed.). *6th European Conference of the International Federation for Medical and Biological Engineering*. [S.l.]: Springer International Publishing, 2015, (IFMBE Proceedings, v. 45). p. 906–909. ISBN 978-3-319-11127-8.

[Brooks et al. 2005]BROOKS, C. et al. Ptolemy ii-heterogeneous concurrent modeling and design in java. Citeseer, 2005.

[Certi 2013]CERTI. *CERTI HLA RTI*. nov. 2013. Disponível em: <http://savannah.nongnu.org/projects/certi>.

[Coumeri e Thomas 1995]COUMERI, S. L.; THOMAS, D. E. A simulation environment for hardware-software codesign. In: IEEE. *Computer Design: VLSI in Computers and Processors, 1995. ICCD'95. Proceedings., 1995 IEEE International Conference on*. [S.l.], 1995. p. 58–63.

[Deprá et al. 2008]DEPRá, D. et al. Metodologia para verificação funcional de hardware através de co-simulação paralela dentro de sistemas de software complexos usando pli: Decodificador h.264/avc como estudo de caso. *HÍFEN*, v. 31, n. 59, 2008. ISSN 1983-6511. Disponível em: <http://revistaseletronicas.pucrs.br/teo/ojs/index.php/hifen/article/view/3898/2965>.

[Donath et al. 1995]DONATH, U. et al. Parallel multi-level simulation with a conservative approach. *Syst. Anal. Model. Simul.*, Gordon and Breach Science Publishers, Inc., Newark, NJ, USA, v. 21, n. 2-3, p. 187–201, dez. 1995. ISSN 0232-9298. Disponível em: <http://dl.acm.org/citation.cfm?id=217120.217127>.

[Edwards et al. 1997]EDWARDS, S. et al. Design of embedded systems: formal models, validation, and synthesis. *Proceedings of the IEEE*, v. 85, n. 3, p. 366–390, Mar 1997. ISSN 0018-9219.

[Eker et al. 2003]EKER, J. et al. Taming heterogeneity - the ptolemy approach. *Proceedings of the IEEE*, v. 91, n. 1, p. 127–144, 2003. Disponível em: <http://chess.eecs.berkeley.edu/pubs/488.html>.

[FDK 2014]FDK. *Federated Simulations Development Kit*. jan. 2014. Disponível em: <http://www.cc.gatech.edu/computing/pads/fdk/>.

[Ferscha e Tripathi 1998]FERSCHA, A.; TRIPATHI, S. K. Parallel and distributed simulation of discrete event systems. 1998.

[Fujimoto 1990]FUJIMOTO, R. M. Parallel discrete event simulation. *Commun. ACM*, ACM, New York, NY, USA, v. 33, n. 10, p. 30–53, out. 1990. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/84537.84545>.

[Fujimoto 1999]FUJIMOTO, R. M. Parallel and distributed simulation. In: ACM. *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future-Volume 1*. [S.l.], 1999. p. 122–131.

[Fujimoto 2000]FUJIMOTO, R. M. *Parallel and distributed simulation systems*. [S.l.]: Wiley New York, 2000.

[Fujimoto 2001]FUJIMOTO, R. M. Parallel simulation: parallel and distributed simulation systems. In: IEEE COMPUTER SOCIETY. *Proceedings of the 33nd conference on Winter simulation*. [S.l.], 2001. p. 147–157.

[Hart, Nilsson e Raphael 1968]HART, P.; NILSSON, N.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, v. 4, n. 2, p. 100–107, July 1968. ISSN 0536-1567.

[HESSEL 2001]HESSEL, F. Concepção de sistemas hererogêneos multi-linguagens. In: SBC. *Jornada de Atualização em Informática – JAI. XXI Congresso da Sociedade Brasileira de Computação.* [S.l.], 2001.

[Hübert 1998]HüBERT, H. *A Survey of HW/SW Cosimulation Techniques and Tools*. 1998.

[IEEE 2000]IEEE. Ieee standard for modeling and simulation (m&s) high level architecture (hla)– framework and rules. *IEEE Std 1516.1-2010 (Revision of IEEE Std 1516.1-2000)*, 2000.

[IEEE 2000]IEEE. Ieee standard for modeling and simulation (m&s) high level architecture (hla)– object model template (omt) specification. *IEEE Std 1516.1-2010 (Revision of IEEE Std 1516.1-2000)*, 2000.

[IEEE 2010]IEEE. Ieee standard for modeling and simulation (m&s) high level architecture (hla)– federate interface specification. *IEEE Std 1516.1-2010 (Revision of IEEE Std 1516.1-2000)*, p. 1–378, 2010.

[Jefferson 1985]JEFFERSON, D. R. Virtual time. *ACM Trans. Program. Lang. Syst.*, ACM, New York, NY, USA, v. 7, n. 3, p. 404–425, jul. 1985. ISSN 0164-0925. Disponível em: <http://doi.acm.org/10.1145/3916.3988>.

[Kim et al.]KIM, Y. et al. An integrated hardware-software cosimulation environment for heterogeneous systems prototyping. In: IEEE. *Design Automation Conference, 1995. Proceedings of the ASP-DAC'95/CHDL'95/VLSI'95., IFIP International Conference on Hardware Description Languages. IFIP International Conference on Very Large Scal*. [S.l.]. p. 101–106.

[Kirkwood]KIRKWOOD, C. E. *Verification of LOTOS Specifications using Term Rewriting Techniques*. Tese (Doutorado) — Citeseer.

[Lee e John 1999]LEE, E. A.; JOHN, I. *Overview of the ptolemy project*. [S.l.]: Electronics Research Laboratory, College of Engineering, University of California, 1999.

[MAK 2014]MAK. *MAK Technologies*. jan. 2014. Disponível em: <http://www.mak.com/products/rti.php>.

[Marrec et al. 1998]MARREC, P. L. et al. Hardware, software and mechanical cosimulation for automotive applications. In: *Rapid System Prototyping, 1998. Proceedings. 1998 Ninth International Workshop on*. [S.l.: s.n.], 1998. p. 202–206. ISSN 1074-6005.

[Mello e Wagner 2002]MELLO, B. A. D.; WAGNER, F. R. A standardized co-simulation backbone. In: *SoC Design Methodologies*. [S.l.]: Springer, 2002. p. 181–192.

[Misra 1986]MISRA, J. Distributed discrete-event simulation. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 18, n. 1, p. 39–65, mar. 1986. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/6462.6485>.

[Negreiros e Brito 2012]NEGREIROS, A. L. V. d.; BRITO, A. V. The development of a methodology with a tool support to the distributed simulation of heterogeneous and com-

plexes embedded systems. In: IEEE. *Computing System Engineering (SBESC), 2012 Brazilian Symposium on*. [S.l.], 2012. p. 37–42.

[Negreiros e Brito]NEGREIROS, Â. L. V. de; BRITO, A. V. Análise da aplicação de simulação distribuída no projeto de sistemas embarcados.

[Nongnu 2014]NONGNU. *Pyhla — python bindings for hla*. nov. 2014. Disponível em: <http://www.nongnu.org/certi/PyHLA>.

[OpenSkies 2014]OPENSKIES. *Cybernet*. jan. 2014. Disponível em: <http://www.openskies.net/features/features.html>.

[Pigatto]PIGATTO, D. F. *Segurança em sistemas embarcados críticos-utilização de criptografia para comunicação segura*. Tese (Doutorado) — Universidade de São Paulo.

[Pitch 2014]PITCH. *Pitch Technologies*. jan. 2014. Disponível em: <http://www.pitch.se/products/prti>.

[Piziali 2004]PIZIALI, A. *Functional Verification Coverage Measurement and Analysis, First Edition*. Massachusetts, USA: Kluwer Academic Publishers, 2004.

[Quaglia e Santoro 2003]QUAGLIA, F.; SANTORO, A. Nonblocking checkpointing for optimistic parallel simulation: Description and an implementation. *Parallel and Distributed Systems, IEEE Transactions on*, IEEE, v. 14, n. 6, p. 593–610, 2003.

[Schmerler, Tanurhan e Muller-Glaser 1995]SCHMERLER, S.; TANURHAN, Y.; MULLER-GLASER, K. A backplane approach for cosimulation in high-level system specification environments. In: *Design Automation Conference, 1995, with EURO-VHDL, Proceedings EURO-DAC '95., European*. [S.l.: s.n.], 1995. p. 262–267.

[Shah e Irfan 2005]SHAH, S. M.; IRFAN, M. Embedded hardware/software verification and validation using hardware-in-the-loop simulation. In: IEEE. *Emerging Technologies, 2005. Proceedings of the IEEE Symposium on*. [S.l.], 2005. p. 494–498.

[Souza et al. 2003]SOUZA, U. R. F. et al. Tangram-virtual integration of heterogeneous ip components in a distributed co-simulation environment. In: IEEE. *Integrated Circuits and*

Systems Design, 2003. SBCCI 2003. Proceedings. 16th Symposium on. [S.l.], 2003. p. 125–130.

[Sung, Oh e Ha]SUNG, W.; OH, M.; HA, S. Interface design of vhdl simulation for hardware-software cosimulation.

[Wang et al. 2004]WANG, X. et al. Optimistic synchronization in hla based distributed simulation. In: *Parallel and Distributed Simulation, 2004. PADS 2004. 18th Workshop on*. [S.l.: s.n.], 2004. p. 123–130. ISSN 1087-4097.

[Wolf 1994]WOLF, W. H. Hardware-software co-design of embedded systems. In: *PROCEEDINGS OF THE IEEE*. [S.l.: s.n.], 1994. p. 967–989.

[Zatt et al. 2006]ZATT, B. et al. Validação de uma arquitetura para compensação de movimento segundo o padrão h. 264/avc. In: *XII IBERCHIP WORKSHOP, Costa Rica*. [S.l.: s.n.], 2006.