UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Desenvolvimento e Avaliação de Simulação Distribuída para Projeto de Sistemas Embarcados com Ptolemy

Ângelo Lemos Vidal de Negreiros

João Pessoa - PB
Janeiro de 2014

ÂNGELO LEMOS VIDAL DE NEGREIROS

Desenvolvimento e Avaliação de Simulação Distribuída para Projeto de Sistemas Embarcados com Ptolemy

Dissertação de Mestrado apresentada ao Programa de Pós-graduação de Informática do Centro de Informática da Universidade Federal da Paraíba, como requisito para obtenção do título de mestre em informática.

Orientador: Prof. Dr. Alisson Vasconcelos de Brito

João Pessoa - PB
Janeiro de 2014

N385d *Negreiros, Ângelo Lemos Vidal de.*
Desenvolvimento e avaliação de simulação distribuída para projeto de sistemas embarcados com Ptolemy / Ângelo Lemos Vidal de Negreiros.-- João Pessoa, 2014.
131f.
Orientador: Alisson Vasconcelos de Brito
Dissertação (Mestrado) – UFPB/CI
1. Informática. 2. Simulação heterogênea. 3. Simulação distribuída. 4. Sistemas embarcados. 5. High Level Architecture *(HLA). 6. Ptolemy.*

ÂNGELO LEMOS VIDAL DE NEGREIROS

Desenvolvimento e Avaliação de Simulação Distribuída para Projeto de Sistemas Embarcados com Ptolemy

Dissertação de Mestrado apresentada ao Programa de Pós-graduação de Informática do Centro de Informática da Universidade Federal da Paraíba, como requisito para obtenção do título de mestre em informática.

Aprovado no dia 29 de janeiro de 2014

BANCA EXAMINADORA:

Prof. Dr. Alisson Vasconcelos de Brito
Universidade Federal da Paraíba – UFPB

Profª. Drª. Monica Magalhães Pereira
Universidade Federal do Rio Grande do Norte

Prof. Dr. Tiago Pereira do Nascimento
Universidade Federal da Paraíba - UFPB

Dedico este trabalho especialmente à minha família, à meus amigos, e à minha noiva os quais me deram todo o apoio necessário para conseguir concluir esta etapa tão importante da minha vida.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por me dar forças nessa minha caminhada em alcançar os meus objetivos da vida profissional, pois sem Ele eu não teria a sabedoria de conseguir trilhar pelos caminhos certos nos momentos mais difíceis dessa fase da vida.

Agradeço aos meus familiares, especialmente aos meus pais, por sempre me apoiarem, de toda a forma possível, e me proporcionarem o melhor ambiente para que eu pudesse alcançar todos os meus objetivos.

Agradeço também aos meus professores da Universidade Federal da Paraíba (UFPB) por dividirem comigo seus conhecimentos, especialmente ao professor Dr. Alisson Vasconcelos de Brito, por ter me auxiliado da melhor forma possível durante a pesquisa e na redação deste documento.

Agradeço também aos amigos de um modo geral, especialmente ao grupo do Movimentos dos Focolares por ter me dado os ensinamentos necessários para saber superar as dificuldades que a vida apresenta.

Por fim, agradeço à minha noiva por ter tido paciência comigo nos momentos mais difíceis, no qual apresentei alto grau de estresse e ainda assim ela me apoiou para seguir sempre em frente nessa caminhada.

“O sucesso nasce do querer, da determinação e persistência em se chegar a um objetivo. Mesmo não atingindo o alvo, quem busca e vence obstáculos, no mínimo fará coisas admiráveis”.

José de Alencar

RESUMO

Atualmente, sistemas embarcados têm apresentado grande poder computacional e consequentemente, alta complexidade. É comum encontrar diferentes aplicações sendo executadas em sistemas embarcados. O projeto de sistemas embarcados demanda métodos e ferramentas que possibilitem a simulação e a verificação de um modo eficiente e prático. Este trabalho propõe o desenvolvimento e a avaliação de uma solução para a modelagem e simulação de sistemas embarcados heterogêneos de forma distribuída, através da integração do Ptolemy II com o *High Level Architecture* (HLA), em que o último é um *middleware* para simulação de eventos discretos distribuídos. O intuito dessa solução é criar um ambiente com alto desempenho que possibilite a execução em larga escala de modelos heterogêneos. Os resultados dos experimentos demonstraram que o uso da simulação não distribuída para algumas situações assim como o uso da simulação distribuída utilizando poucas máquinas, como, uma, duas ou três podem ser inviável. Demonstrou-se também a viabilidade da integração das duas tecnologias, além de vantagens no seu uso em diversos cenários de simulação, através da realização de diversos experimentos que capturavam dados como: tempo de execução, dados trocados na rede e uso da CPU. Em um dos experimentos realizados consegue-se obter o *speedup* de fator quatro quando o modelo com quatro mil atores foi distribuído em oito diferentes computadores, em um experimento que utilizava até 16 máquinas distintas. Além disso, os experimentos também demonstraram que o uso do HLA apresenta grandes vantagens, de fato, porém com certas limitações.

Palavras-chave: Simulação Heterogênea, Simulação Distribuída, Sistemas Embarcados, Alto desempenho, *High Level Architecture* (HLA), Ptolemy.

ABSTRACT

Nowadays, embedded systems have a huge amount of computational power and consequently, high complexity. It is quite usual to find different applications being executed in embedded systems. Embedded system design demands for method and tools that allow the simulation and verification in an efficient and practical way. This paper proposes the development and evaluation of a solution for embedded modeling and simulation of heterogeneous Models of Computation in a distributed way by the integration of Ptolemy II and the *High Level Architecture* (HLA), a middleware for distributed discrete event simulation, in order to create an environment with high-performance execution of large-scale heterogeneous models. Experimental results demonstrated that the use of a non distributed simulation for some situations as well as the use of distributed simulation with few machines, like one, two or three computers can be infeasible. It was also demonstrated the feasibility of the integration of both technologies and so the advantages in its usage in many different scenarios. This conclusion was possible because the experiments captured some data during the simulation: execution time, exchanged data and CPU usage. One of the experiments demonstrated that a speedup of factor 4 was acquired when a model with 4,000 thousands actors were distributed in 8 different machines inside an experiment that used up to 16 machines. Furthermore, experiments have also shown that the use of HLA presents great advantages in fact, although with certain limitations.

Key-words: Heterogeneous Simulation, Distributed Simulation, Embedded Systems, High Perfomance, *High Level Architecture* (HLA), Ptolemy

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ÍNDICE DE GRÁFICOS

ÍNDICE DE ABREVIATURAS

AMT	Automatic Teller Machine
API	Application Programming Interface
CT	Continuous Time
CSP	Communicating Sequential Processes
CPS	Cyber-Physical Systems
DES	Discrete Event Simulation
DE	Discrete Event
DF	Data Flow
DMSO	Defence Modelling and Simulation Office
DoD	Departamento de Defesa Americano
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
FIFO	First In First Out
FOM	Federation Object Model
FPGA	Field Programmable gate arrays
FSM	Finite State Machine
GALT	Greatest Available Logical Time
GUI	Graphical User Interface
HLA	*High Level Architecture*
IS	Interface Specification
JVM	Java Virtual Machine
LBTS	Lower Bound on TimeStamp
MoC	Model of Computation
MOM	Management Object Model
NMA	NULL Message Algorithm

OMT	Object Model Template
OMG	Object Management Group
PADS	Parallel and Distributed Simulation
PC	Personal Computer
PDA	Personal Digital Assistant
PDES	Parallel Discrete Event Simulation
PL	Processo Lógico
PN	Process Network
RFID	Radio Frequency Identification
RSSF	Redes de Sensores Sem Fio
RTI	RunTime Infrastructure
RTIA	RTI Ambassador
RTIG	RTI Gateway
SDF	Synchronous Data Flow
SoC	System on Chip
SOM	Simulation Object Model
SR	Synchronous Reactive
VHDL	VHSIC *Hardware* Description Language
VHSIC	Very High Speed Integrated Circuits

SUMÁRIO

Capítulo

1

Introdução

Atualmente, existe um número crescente de dispositivos presentes no dia a dia das pessoas. A maioria desses dispositivos contém diversos circuitos e *software* embarcados. Alguns exemplos podem ser citados como: *Personal Digital Assistants* (PDA), *Automatic Teller Machines* (AMT), relógios digitais, micro-ondas, aparelhos de som, videogames, carros, *smartphones*, entre outros [1]. Observa-se que tais dispositivos embarcados apresentam maior poder computacional a um custo cada vez menor, confirmando de forma superficial a *Lei de Moore*. Esse crescimento possibilitou que sistemas cada vez mais complexos fossem desenvolvidos, necessitando dessa forma de ambientes de projeto e simulação que acompanhassem essa evolução. Como exemplo, podem ser citados principalmente as Redes de Sensores sem Fio (RSSF), os sistemas baseados em *smartphones*, os sistemas de automação e controle, os *Cyber-Physical Systems* (CPS) [2], *Smart Grids* e os sistemas de apoio à saúde.

A área de sistemas embarcados apresentou maior interesse nos últimos anos. São sistemas que envolvem desde eletrônica simples até sistemas distribuídos complexos os quais combinam *hardware* com *software*. Existem diversos sistemas complexos de computação que podem ser considerados de uso geral, que existem normalmente para solucionar diversos problemas de cunho específico, porém cada um desses sistemas executa de forma eficiente suas especificidades e por isso conseguem solucionar uma grande variedade de problemas. Alguns, por exemplo, são limitados por exigências como restrição de desempenho em tempo real por motivos de segurança; outros, podem apresentar requisitos de baixo ou nenhum desempenho, muitas vezes utilizados com a intenção de otimizar ou mesmo reduzir os custos [1].

Sistemas embarcados são considerados uma área multidisciplinar, pois combina teorias e técnicas de diferentes campos de estudo, como por exemplo, eletrônica, projeto de *hardware*, projeto de *software*, ciência da computação e telecomunicação.

Como já citado, uma característica comum de tais sistemas complexos embarcados é a heterogeneidade dos seus Modelos de Computação (MoC). Tais sistemas executam diversas aplicações heterogêneas tanto em *hardware* quanto em *software* (processamento de sinais, multimídia, processamento analógico, fluxo de controle, sistema operacional, etc.). Pesquisas sobre MoC formais iniciaram nas décadas de 30 e 40 com pesquisadores renomados como Alan Turing, Alonzo Church e outros. Um modelo de computação define as operações que podem ser realizadas durante uma computação e seus respectivos custos. Há vários modelos de computação e variedades de máquinas de computação que realizam tais modelos. Como principais modelos podem ser destacados o *Communicating Sequential Processes* (CSP) [3], *Continuous Time* (CT) [4], *Discret Event* (DE), *Process Network* (PN) [5], *Synchronous-reactive* (SR) [6] e *Synchronous Dataflow* (SDF) [7]. Como exemplo de máquinas que executam tais modelos temos Circuitos Lógicos, Autômatos de Pilha, Máquinas de Acesso Aleatório, Hierarquia de Memória, Máquinas de Turing, chips VLSI, além de outra variedade de máquinas paralelas [8]. Tais modelos de computação serão detalhados no Capítulo 3.

1.1. Problema

Uma abordagem bastante comum em projetos de sistemas embarcados é a prática de simulações [9]. A simulação por computador é uma computação que modela o comportamento do sistema sobre o tempo. A simulação de um modo geral se baseia em modelos que por sua vez são uma simplificação abstrata do sistema. Existem alguns diferentes modelos de simulação, como por exemplo, o que representa propriedades estáticas do sistema, como funcionalidades ou dimensões físicas, e outro que é chamado de construtivo, em que se define um procedimento computacional que imita o comportamento do sistema. Os modelos construtivos são executáveis e normalmente descrevem o comportamento do sistema em resposta a um estímulo externo. Em sistemas embarcados algumas vezes a análise detalhada não é possível de ser realizada de forma simples e viável, então a simulação é utilizada como forma de suprir tal deficiência.

A computação baseada em simulação dinâmica normalmente é executada para sistemas embarcados, em que pode ser definido como sendo “uma imitação, através do uso do computador, de um sistema no decorrer do tempo” [10]. Atualmente, simulações

de sistemas estão sendo amplamente utilizadas para analisar e prever aspectos do comportamento do sistema simulado sem necessariamente precisar construir o sistema propriamente dito de fato. Essa técnica normalmente é utilizada na fase de verificação e validação de *software*, o que faz com que o custo final de desenvolvimento de um sistema embarcado seja minimizado, pois um erro em uma situação não prevista pode demandar bastante tempo no projeto de um novo sistema embarcado. A simulação é considerada uma ferramenta essencial para o estudo de redes de sensores sem fio (RSSF) devido à inviabilidade das análises de forma mais profunda de tais redes, além das dificuldades de configurar experimentos reais. Além disso, simulações também tentam amenizar situações em que existe um grande conjunto de estados impossíveis de serem todos alcançáveis, gerando dessa forma uma amostra significativa de cenários que requer que a simulação seja executada várias vezes. Essas características juntamente com o aumento de tamanho e especialmente da complexidade dos modelos resultam em uma alta demanda de computação e consequentemente em um maior tempo de simulação [1].

Normalmente, processadores únicos não podem suportar essa demanda. A própria Lei de *Moore* diz que o poder computacional aumenta de forma exponencial, em termos de frequência de operação dos microprocessadores. Mesmo com esse poder computacional, algumas vezes não é possível lidar com a quantidade de trabalho (instruções e complexidade das instruções). Além disso, os processadores em termos de frequência já estão chegando ao limite, pois o aumento na frequência faz também aumentar a temperatura, o que não é interessante. Logo, o foco no desenvolvimento de *hardware* não é mais na frequência e sim aumentar a quantidade de core dos processadores.

Para lidar com a questão de que processadores únicos não conseguem suportar alta demanda de processamento em algumas situações, então a abordagem da distribuição de tal processamento em vários processadores distintos passa a ser utilizada no intuito de viabilizar tal execução. É interessante notar também que enquanto um sistema com *n* processadores paralelos pode ser menos eficiente que um processador *n* vezes mais rápido, o sistema paralelo é normalmente mais barato de se construir, e por isso muitas vezes os sistemas embarcados optam pelo menos custoso [1].

Sistemas embarcados complexos, com MoCs heterogêneos, apresentam normalmente nível de dificuldade grande para realizar testes, com várias possibilidades, e por isso se faz necessário o uso de técnicas de simulações para saber se o sistema ou

protótipo desenvolvido está correto ou não, e se vai suportar ou não determinada demanda de processamento. Algumas vezes, o sistema pode ser complexo ao ponto em que uma simples simulação que trabalha com MoCs heterogêneos em um único processador seja muito dispendiosa, ou seja, alta demanda de processamento, tornando a simulação até mesmo inviável de ser finalizada. Dessa forma, pode ser necessário o uso de simulações distribuídas, que usem vários processadores, sem perder a característica de trabalhar com MoCs heterogêneos, para acelerar ou até mesmo viabilizar o processo da simulação por completo.

Contextualizando, recentemente, ambientes com vários grupos de pessoas tem sido objeto de discussão de pesquisas. Lugares onde existe movimento intenso de pessoas têm sido estudados com os mais variados propósitos possíveis, como: emissão de alertas contra acidentes envolvendo pedestres e veículos no cruzamento de rodovias, como descrito em Sun *et al* [11], no qual propõe métodos que podem estimar o caminho das pessoas, acoplando nestas diversos sensores que enviam sinais para outros sensores espalhados por todas as ruas da cidade. Tais aspectos mostram que existe necessidade de controlar um ambiente que contenha um fluxo crescente de pessoas, além de estimar que deslocamentos são esses, para evitar assim que situações indesejáveis ocorram. Um exemplo a ser citado, que seria interessante o uso desse tipo de controle, foi o caso da Boate Kiss, no Rio Grande do Sul, onde centenas de pessoas morreram e outras centenas ficaram feridas por não saberem onde ficavam as saídas de emergências, entre outros fatores.

Outra abordagem que está sendo utilizada é o uso do RFID (*Radio Frequency Identification*) que parece trazer grandes benefícios para as pessoas. Novas cidades estão surgindo, utilizando *tags* RFID. De acordo com Lee *et al* [12], essas *tags* são utilizadas para identificar qualquer tipo de objeto, como cartões de identificações, controle de acesso, monitoramento de carros, monitoramento de pessoas, etc. As *tags* RFID são também utilizadas como base em aplicações de redes de sensores sem fio (RSSF), como monitoramento de alvos em movimentos e monitoramento de meio ambiente.

Diante do exposto, o uso da computação distribuída, ou ainda, da simulação distribuída, apresenta uma boa alternativa de custo X benefício para simular situações e sistemas como os citados anteriormente. Simulação esta que já está sendo utilizada em aplicações com ênfase em alta demanda de computação de acordo com Cuadrado [1].

Pode-se então dizer que o principal problema a ser estudado por esta dissertação é como possibilitar a simulação distribuída em larga escala de modelos complexos heterogêneos. A solução de Cuadrado, por exemplo, enfatiza apenas em alta demanda de computação, ou seja, em simulação distribuída, o que não é o único escopo desta dissertação, na qual tem o objetivo principal de apresentar uma solução de simulação distribuída de sistemas heterogêneos em larga escala, o que não é encontrado facilmente na literatura existente.

.

1.2. Motivação

Computação distribuída possibilita a execução da simulação de programas em um sistema computacional contendo múltiplos processadores [1]. Normalmente é amplamente utilizada para aprimorar e acelerar a simulação, possibilitando a análise de sistemas complexos através da exploração simultânea de dois pontos principais, o poder computacional dos processadores e a capacidade de memória melhorada provenientes do uso de múltiplos computadores espalhados em uma rede de computadores. Na prática, desenvolver simulação distribuída é mais difícil do que desenvolver uma simulação sequencial, pois é necessário lidar com todas as questões que a computação distribuída apresenta, como por exemplo, *interfaces*, mapeamento de atividades para os processadores, além de problemas de comunicação e sincronização dos dados.

O projeto de simulação distribuída requer a utilização de técnicas que não estão presentes necessariamente em uma simulação sequencial. Diante de tal especificidade das simulações distribuídas, grande parte das ferramentas de simulação que lidam com sistemas heterogêneos normalmente não possibilitam simulação distribuída de tais modelos.

Como tais sistemas complexos utilizam o processador e a memória de forma intensa, então se faz necessário que este tipo de simulação passe a ser distribuída para acelerá-la. Uma ferramenta que proporcione a simulação de MoCs heterogêneos, característica comum de sistemas embarcados, e de forma distribuída se faz necessário. Como esta ferramenta única no contexto em questão não foi encontrada, então este trabalho utiliza ferramentas existentes com características distintas e as integra.

1.3. Objetivos

O objetivo deste trabalho é apresentar uma proposta para modelar e simular sistemas embarcados heterogêneos (em especial, projetos de redes de sensores sem fio) em larga escala de forma distribuída, para possibilitar, por exemplo, que sistemas que lidam com situações reais como as citadas no início deste capítulo sejam analisadas.

1.3.1. Objetivos específicos

- Estudar ferramentas de simulação de sistemas embarcados e simulação distribuída, em especial o Ptolemy e o HLA;
- Integrar o Ptolemy que auxilia na modelagem e simulação de sistemas heterogêneos, com o HLA que permite a simulação dos sistemas em larga escala de forma distribuída;
- Desenvolver um ambiente de simulação para facilitar a integração entre o Ptolemy e o HLA.
- Validar a integração do HLA com o Ptolemy, por meio de experimentos, verificando assim sua viabilidade.
- Avaliar o desempenho (uso da CPU, tempo de execução, dados trocados, *speedup*) dos experimentos citados anteriormente.

1.4. Trabalhos Relacionados

Sistemas embarcados são intrinsecamente formados de computação paralela. Abordagens tradicionais de modelagem e implementação de sistemas paralelos, como MPI e *Threads* podem gerar soluções não determinísticas, difíceis de depurar e passíveis de *deadlock*, como demonstrado em [13].

Outra limitação é a grande lacuna de abstração entre a modelagem e o desenvolvimento de sistemas embarcados. Metodologias, como UML, pecam pelo seu alto grau de abstração e pela sua dificuldade de representação de computação paralela. Já outras metodologias suprem essas faltas, mas são geralmente restritas a domínios específicos, como Linguagens de Descrição de *Hardware*, ou HDL (como VHDL, *Verilog*, *SystemC*, *SystemVerilog*), ferramentas EDA (para projeto de sistemas eletrônicos), *OMNET* e *NS-2* (para simulação de comunicação e rede de computadores).

Já projetos como *Matlab*, *Simulink* e Ptolemy oferecem soluções com suporte a multi domínios, apesar de não suportarem a simulação distribuída de seus modelos.

Na literatura não é incomum encontrar trabalhos que se dedicam a simulação heterogênea de sistemas diversos. Em Wetter *et al* [14] é apresentada a modelagem e simulação de um sistema heterogêneo para controle de edifícios utilizando o Ptolemy II. A contribuição maior desse trabalho se deve a integração dos diversos simuladores através do Ptolemy, como é o caso do *Simulink* e sistemas próprios de automação predial. Entretanto, esse trabalho não possibilita a simulação distribuída desses sistemas.

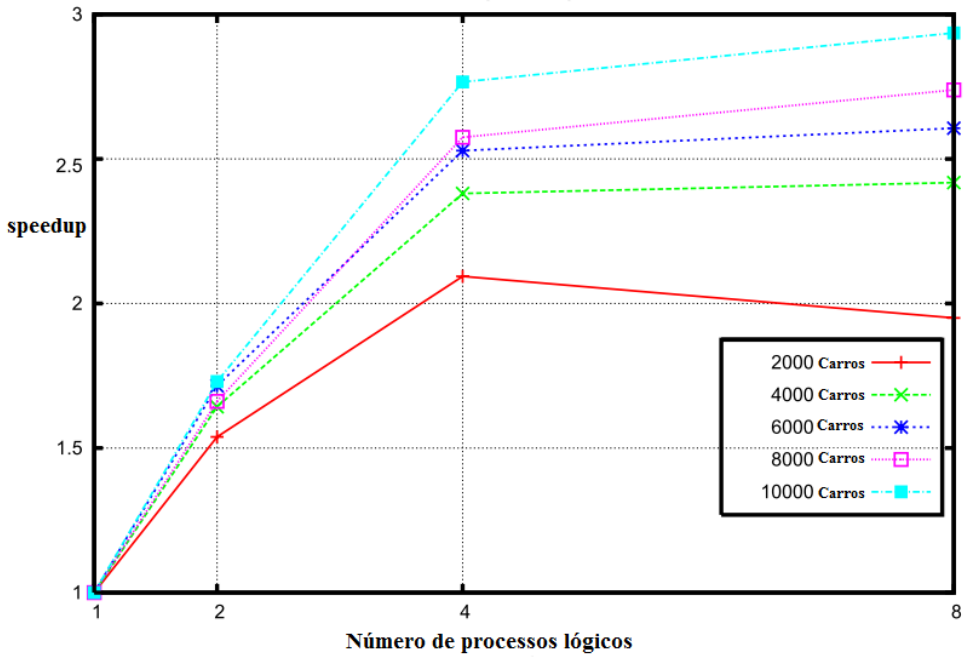
Há também esforços para a integração de sistemas heterogêneos e sua simulação distribuída. Em Sung *et al* [15] é apresentada a interoperabilidade de dois modelos de computação (*Discret Event* e *Continuous Time*) para simulação de um sistema de controle de nível de reservatório de água, utilizando HLA como plataforma para simulação distribuída e o *Matlab* para simular o sistema de controle. Outra proposta de Sung *et al* [16] integra o *Matlab* e o *DEVSim++* num ambiente único e distribuído de simulação através do HLA, resultado de um esforço para integrar dois modelos de computação através de simuladores híbridos. O que é importante ressaltar nas duas propostas de Sung é a sua baixa escalabilidade no que diz respeito a possibilidade de expandir a simulação para outros modelos de computação, o que não ocorreria caso o Ptolemy fosse utilizado, pois é uma estratégia mais abrangente na qual é utilizada neste trabalho.

O trabalho de Bononi *et al* [17] utiliza o middleware ARTIS que é baseado no HLA para viabilizar simulações de sistemas dinâmicos, de RSSF em larga escala. A abordagem do artigo citado lida com uma rede de veículos que circulam e transmitem sinais um para o outro dentro de uma mesma cidade. O cenário apresenta algumas entidades, como: ruas, cruzamentos, veículos, etc. Bononi utiliza diversas máquinas, e cada uma delas pode possuir um ou mais processos lógicos para gerenciar um subconjunto específico de entidades. Ele ainda divide o mapa da área simulada em diversas porções, em que cada uma dessas contém estradas, cruzamentos e veículos se locomovendo e interagindo entre si. Cada porção ficará dentro de um processo lógico específico para computar os dados necessários. A sincronização utilizada por este trabalho é a time-step.

Bononi utiliza um *Quad* Intel Xeon CPU 1.5GHz 1GB RAM e um *Dual* Intel Xeon CPU 2.6 GHz 2GB RAM em uma rede ethernet *Gigabit* e divide os experimentos da seguinte forma: 2000, 4000, 6000, 8000 e 10000 veículos; 2, 4 e 8 processos lógicos.

De um modo geral, quanto mais processos lógicos (PL) envolvidos, e quanto mais máquinas são usadas nos experimentos, melhor será o *speedup*. Porém, Bononi mostra um dado interessante, em que quando se usa um caso mais leve, ou seja, poucas entidades envolvidas, ou pouca computação envolvida, para os casos em que se usa 4 PL e 8 PL, o *speedup* cai um pouco quando comparado ao caso que se usa apenas 2 PL. Isso, de acordo com o autor, acontece devido ao impacto do *overhead* de comunicação que é necessário durante a sincronização entre os PLs na simulação distribuída. Porém, se usar um contexto com mais entidades, a concorrência passa a superar o limite do *overhead* da comunicação, melhorando o *speedup* até certo limite. A Figura 1, retirada do artigo de Bononi, apresenta alguns pontos interessantes a serem analisados, em que o uso de pouca computação, ou seja, poucos veículos sendo envolvidos faz com que o *speedup* seja minimizado. Observa-se também que o *speedup* aumenta bastante quando se passa de 2 para 4 PLs para todos os casos em que se varia a quantidade de veículos, porém no caso em que se usa 2000 carros, por exemplo, o *speedup* cai quando se passa de 4 para 8 PLs.

Figura 1. Valor do speedup com números de carros e PLs envolvidos variados [17]



Em outra oportunidade em seu trabalho, o autor mostra também que em determinadas configurações, a simulação sequencial (*speedup* igual a 1) tem um desempenho maior do que a simulação paralela.

O trabalho de Bononi apresenta várias semelhanças com a proposta desta dissertação (Gráfico 1), pois tanto Bononi quanto esta dissertação utilizam o HLA para dividir o cenário de estudo de RSSF em diversas porções, em que cada uma delas ficará,

ou em uma PL específica, ou em uma máquina a parte. Além disso, observa-se também que ambos os trabalhos apresentaram um *speedup* crescente à medida que se aumentava o número de PLs ou o número de máquinas até um determinado limite. Depois desse limite, ou o *speedup* decrescia, ou aumentava de forma bastante amena, tendendo a apresentar variações mínimas. Tal semelhança nos resultados demonstra que esta dissertação apresenta resultados coerentes com os demais artigos publicados da área.

Outro trabalho interessante para se analisar é o de Guha *et al* [18] que avalia o desempenho de duas tecnologias de computação distribuída, o HLA e o TSPACE (*Tuple Space*) em um cluster de computadores. Ele utiliza as tecnologias para resolver o problema: "Sistema de Detecção de Intrusão (*Intrusion Detection System* – IDS)" em redes de computadores, utilizando para isso redes neurais com retro-propagação (*Back Propagation Neural Network* - BPNN) para o processo de mineração de dados em uma abordagem de computação paralela. A ideia é detectar o mais rápido possível invasões que acontecem na comunicação entre pacotes em uma rede, através da análise dos dados, verificando padrões incomuns de dados quando estes acontecem na rede. Para isso, foi utilizado o algoritmo de treinamento específico do BPNN para IDS. Assim, foi fornecido um grande conjunto inicial de dados em um arquivo, em que o detector de intrusão aprende a tarefa de predizer o modelo, ou seja, classificá-lo, distinguindo entre conexões "ruins", chamadas de ataques (intrusões) de conexões "boas" (normais). O processo de treinamento aconteceu em um arquivo de 2 milhões de registros particionados igualmente em N processadores, em que cada processador é um computador auto-contido que faz parte do cluster, chamado de Nó. Um dos nós é designado de Mestre (*Master*), responsável por alocar as tarefas e unificar os resultados; os demais, são chamados de Trabalhadores (*Workers*), responsáveis pelo algoritmo de BPNN. O propósito deste trabalho foi utilizar as duas tecnologias HLA e TSPACE para realizar o algoritmo de treinamento em um grande arquivo de dados, verificando assim quais dos dois algoritmos foi mais eficiente trabalhando de forma distribuída, em termos de tempo de execução. Além disso, Guha utilizou esses resultados como uma forma de dar mais confiabilidade para usar o HLA em outros projetos, como por exemplo, na implementação de sistemas de simulação distribuída para projetos de RSSF. Guha pôde afirmar a partir de seus experimentos que o HLA foi melhor do que o TSPACE para problemas que necessitam trabalhar em grandes escalas. Dessa forma, esta dissertação também utilizou o HLA para esse mesmo fim.

O autor durante seus experimentos utiliza duas formas de configuração para o uso do HLA, onde a primeira é chamada de *Scerola Cluster* que utiliza computadores de um único core, e a segunda *Ariel Cluster* que utiliza computadores *dual* core. O uso do TSPACE é mais simples de modelar em termos de engenharia de software quando comparado ao HLA, porém o primeiro apresenta um desempenho pior em termos de tempo de execução da simulação quando comparado ao segundo, pois os resultados mostram que o uso do HLA decresce de forma mais significativa o tempo de execução, já que a comunicação entre os nós é mais rápida. O desempenho do HLA não é bom quando se tem somente um trabalhador envolvido, que pode ser atribuído ao overhead que o HLA apresenta como uma solução mais pesada e também robusta para problemas de sistemas distribuídos. A Tabela 1 foi retirada do artigo de Guha que mostra o tempo de execução da simulação utilizando cada uma das tecnologias citadas.

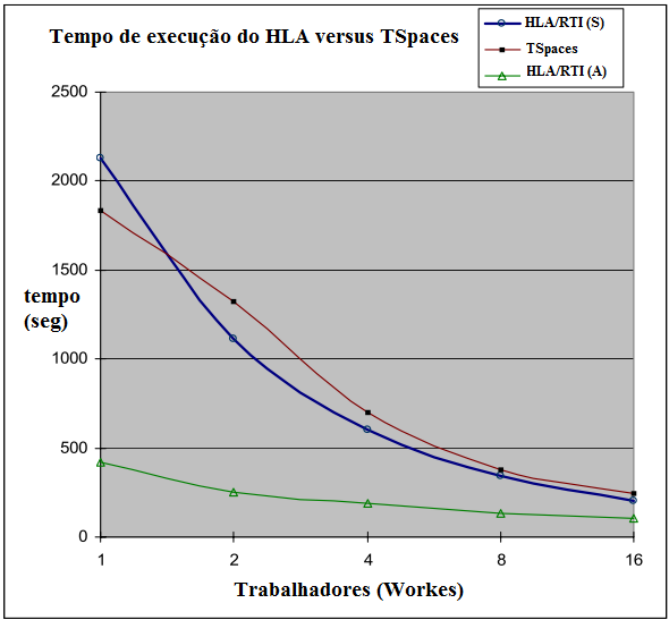
Tabela 1. Tempo de execução do programa BPNN paralelo [18]

Nós / Tempo de Execução (seg)	HLA/RTI (*Scerola Cluster*)	TSpaces (*Scerola Cluster*)	HLA/RTI (*Ariel Cluster*)
1	2129.37	1837.13	417.83
2	1115.98	1323.58	251.16
4	601.24	697.64	188.95
8	345.02	380.83	134.93
16	204.23	243.49	105.32

Outra análise interessante deste trabalho é que segundo o autor, baseado em seus testes, se usar o algoritmo com 32 nós no *Cluster Ariel*, a execução pode ocasionalmente entrar em *deadlock*. Isso acontece devido aos pequenos tempos de processamento nos processadores clientes, resultando em uma utilização crescente do canal de comunicação. Quanto mais processadores em um sistema paralelo, mais são os problemas de implementação os quais precisam ser planejados e avaliados cuidadosamente para evitar problemas de *deadlock*, pois a comunicação na rede se torna o gargalo.

O gráfico dos resultados de Guha (Figura 2) exposto a seguir apresenta grande semelhança ao gráfico desta dissertação (Gráfico 2), pois o tempo de execução decresce à medida que se tem mais computadores envolvidos na simulação, porém em um determinado momento a curva é suavizada, sofrendo poucas variações. Isso demonstra que o uso do HLA nessas situações apresenta comportamentos semelhantes.

Figura 2. Tempo de execução do programa BPNN paralelo [18]



Dessa forma, somente aumentando a quantidade de máquinas pode não ser a solução ideal quando se trabalha com HLA. Nesses casos, é interessante utilizar técnicas de engenharia de software para melhorar a implementação propriamente dita.

Outro trabalho interessante que se assemelha bastante em termos de implementação com o desta dissertação, é o de Lasnier et al [19], que em parceria com a empresa Onera [20] (responsável por desenvolver o CERTI, que é uma implementação do HLA) e a Universidade de Berkeley [21] (responsável pelo desenvolvimento do Ptolemy), desenvolveu uma extensão para o Ptolemy que pudesse interagir com o HLA, no intuito de possibilitar a co-simulação distribuída de sistemas físicos cibernéticos (Cyber-Physical Systems – CPS). A ideia do autor é combinar os dois *frameworks* para elaborar experimentos com: heterogeneidade, provida pelo Ptolemy e a interoperabilidade provida pelo HLA. Percebe-se então que a integração desses dois *frameworks* é viável e de interessante interesse tanto da comunidade do CERTI, quanto do Ptolemy. Para tal, Lasnier desenvolveu um ambiente de co-simulação em que diversas simulações do Ptolemy podiam ser executadas como federados na federação do HLA. Desta forma, os federados do Ptolemy podiam trocar dados com os outros federados que podem também ser simulações do Ptolemy ou até mesmo federados C++ ou Java. No intuito de facilitar a interação entre o HLA e o Ptolemy, o autor criou uma extensão do Ptolemy com componentes dedicados que possibilitassem a conexão com o HLA e também a troca de dados. Um novo atributo foi introduzido no modelo do Ptolemy, intitulado de *HlaManager* que gerencia de forma centralizada o avanço do tempo no Ptolemy e no HLA. Além disso, dois atores foram criados, chamados

HlaPublisher e *HlaSubscriber*, que funcionam como uma interface entre os ambientes Ptolemy e HLA/CERTI, sendo responsáveis pela comunicação dos dados em si.

O trabalho de Lasnier tem a finalidade de integrar o Ptolemy com o HLA, assim como a proposta desta dissertação. O primeiro tem como vantagem em relação ao segundo, as parcerias realizadas com os desenvolvedores dos dois *frameworks* integrados, e foi publicado somente no ano de 2013 [19] no mesmo evento em que um artigo desta dissertação foi publicado [22]. O foco de Lasnier está em interoperar diferentes simulações utilizando esses dois *frameworks*. Já o objetivo principal desta dissertação, está em simular sistemas heterogêneos no Ptolemy, distribuí-los com HLA para que seja possível a simulação distribuída em larga escala do cenário em questão, e integrar os dois *frameworks* para possibilitar tal distribuição no Ptolemy, avaliando também o desempenho da integração realizada [23].

Pode-se observar através dos trabalhos anteriores, que de um modo geral não há uma abordagem que una completamente o melhor dos dois mundos, simulando sistemas heterogêneos de forma abrangente e distribuída. O único trabalho que aborda tal união é o trabalho de Lasnier, que propõe exatamente a mesma integração desta dissertação, do HLA com o Ptolemy. Porém, Lasnier tem o foco maior apenas na integração desses dois *frameworks*, tornando algo fácil de configurar e utilizar, já esta dissertação, além da integração, tem como foco principal avaliar o desempenho de projetos reais (ex. deslocamento de pessoas em um determinado ambiente) que utilizam tal integração. Para isso, realizaram-se diversos experimentos com variáveis distintas para avaliar o desempenho da simulação que integra os dois *frameworks*.

Os demais trabalhos abordavam basicamente uma arquitetura que suportava simulação distribuída de alguns modelos, ou forneciam apenas o suporte a multi-domínios, sem se preocupar com a questão de simulações em larga escala e distribuída. Weter, por exemplo, se preocupa apenas em realizar simulação heterogênea; Sung faz o uso da ligação entre simulação heterogênea com HLA, utilizando para isso o Matlab; esta dissertação, por outro lado, se preocupa em elaborar projetos de modelos heterogêneos com o Ptolemy que é um framework mais abrangente para o interesse deste trabalho comparado ao Matlab. Além disso, o trabalho proposto por esta dissertação tem o interesse de possibilitar que projetos heterogêneos do Ptolemy sejam executados em larga escala de forma distribuída através do uso do HLA. Dessa forma, o uso do Ptolemy integrado ao HLA já é um diferencial em relação aos trabalhos de Weter e Sung; e a análise de desempenho e viabilidade dessa integração para modelos

reais contextualizados é outro ponto adicional quando comparado ao trabalho de Lasnier.

Outros trabalhos interessantes citados são os de Bononi e Guha. Entretanto, esta dissertação apresenta como vantagem em relação a eles a possibilidade de criar modelos variados, já que utiliza o Ptolemy com HLA, o que possibilita um número maior de experimentos a serem elaborados, apresentando assim resultados e análises de desempenho de diversos casos que podem mostrar de forma mais clara a viabilidade do uso do HLA com o Ptolemy. Bononi se preocupa mais em paralelizar e distribuir um projeto de redes sem fio de veículos para melhorar o desempenho, através do uso do HLA. Tal trabalho apresenta duas principais semelhanças com o desta dissertação: o projeto de redes de sensores sem fio, os quais utilizam o HLA no intuito de aprimorar o desempenho; e os resultados, que são bastantes semelhantes um com o outro. Porém, o primeiro não faz o uso de um Framework como o Ptolemy, e tem o foco apenas nas redes de veículos, não possibilitando dessa forma, a criação de diversos projetos heterogêneos. Estes podem ser elaborados com o segundo trabalho, já que este faz o uso do Ptolemy, possibilitando assim, projetos mais generalistas. Guha, por outro lado, inicialmente tem a nítida intenção de realizar um teste utilizando tecnologias distintas com o mesmo propósito para chegar a uma conclusão de qual poderia ser a melhor opção para o estudo dele. Desse modo, o autor não tem o objetivo inicial de desenvolver projetos práticos heterogêneos para serem distribuídos, como esta dissertação o tem. Como um projeto prático, pode-se citar o exemplo do deslocamento de várias pessoas em uma boate específica ou qualquer ambiente escolhido para esse fim. O resultado do trabalho de Guha pode ser útil, na verdade, para a decisão da escolha da tecnologia a ser utilizada para a distribuição de projetos que fazem o uso do Ptolemy, no caso, a tecnologia HLA.

A Tabela 2 mostra um breve resumo das principais características dos trabalhos relacionados e o que esta dissertação apresenta como adicional em relação a eles.

Tabela 2. Comparação entre os trabalhos relacionados

Trabalho	Simulação Heterogênea	Simulação Distribuída	Uso do Ptolemy	Uso do HLA	Modelos práticos (reais)	Análise de desempenho
Weter	SIM	NÃO	SIM	NÃO	SIM	NÃO
Sung	SIM	SIM	NÃO	SIM	SIM	NÃO
Bononi	NAO	SIM	NAO	SIM	SIM	SIM
Guha	NAO	SIM	NAO	SIM	NAO	SIM
Lasnier	SIM	SIM	SIM	SIM	NAO	NAO
Dissertação	SIM	SIM	SIM	SIM	SIM	SIM

Esta dissertação propõe então uma solução para a modelagem e simulação distribuída de sistemas embarcados heterogêneos, através da integração do Ptolemy II com o HLA, criando assim um ambiente que permita a fácil configuração de todo o ambiente e principalmente a execução de modelos de larga escala com alto desempenho, avaliando cada caso específico em termos do desempenho da simulação.

1.5. Metodologia e Hipótese

O primeiro passo para viabilizar a simulação heterogênea em larga escala será o estudo de ferramentas que lidam com a simulação de MoCs distintos, aprofundando especialmente o *framework* Ptolemy. Após isso, será estudado de forma detalhada uma especificação que possibilita simulação distribuída, o HLA, entrando em detalhes em sua implementação. Tais estudos são apresentados no estado da arte apresentados nos Capítulos 2, 3 e 4. Tendo o conhecimento tanto do Ptolemy quanto do HLA, será necessário integrar estes dois mundos para possibilitar maior eficiência nas simulações, integração esta que até meados de 2013 não era conhecida na literatura estudada.

Para poder medir a eficiência do ambiente de simulação que integra o Ptolemy com o HLA, e o ambiente do Ptolemy de forma isolada, alguns experimentos específicos, utilizando o mesmo cenário, serão realizados:

1. Experimentos utilizando exclusivamente o Ptolemy.
2. Experimentos utilizando um ambiente que integra o Ptolemy com o HLA.

As principais perguntas de pesquisa que este trabalho busca resolver são: "**É possível integrar o HLA ao Ptolemy, e ainda essa integração viabiliza a simulação de grandes sistemas heterogêneos em larga escala?**", ou ainda, "**O uso do HLA integrado ao Ptolemy apresenta vantagens, em termos de desempenho da aplicação, comparado ao uso do Ptolemy de forma isolada?**".

A hipótese apresentada neste contexto é que os experimentos que fazem o uso da simulação normal (não distribuída), a que utiliza de forma exclusiva o Ptolemy, são inviáveis em algumas situações ou apresentam certos limites em outras, os quais poderão ser amenizados com o uso da simulação distribuída (HLA) de forma integrada com o Ptolemy. Com isso, diversas aplicações interessantes como as citadas nas seções anteriores (deslocamento de pessoas em um ambiente fechado, desastres naturais, entre outras) podem ser simuladas e assim validadas.

Além disso, este trabalho apresenta, de forma mais detalhada, a abordagem de realização dos experimentos nos dois cenários distintos, verificando se os resultados obtidos por ambos os casos são consistentes, e analisando o tempo de execução, o tráfego na rede, e o uso da CPU em cada um dos cenários e em cada máquina envolvida nos experimentos.

Por fim, um ambiente de fácil configuração através do uso da interface gráfica foi desenvolvido, para facilitar a integração entre o Ptolemy e o HLA.

1.6. Estrutura da dissertação

Esta dissertação está organizada em seis Capítulos. O Capítulo 2 lida com o conceito de Simulação Heterogênea que abordará a principal ferramenta utilizada neste trabalho, o Ptolemy. Já o Capítulo 3 trata do consolidado MoC (Modelos de Computação). O Capítulo 4 apresenta a segunda base deste trabalho, a simulação distribuída, que abordará o *framework* HLA e a implementação do CERTI. A aplicação desenvolvida e os resultados obtidos são apresentados, respectivamente, nos Capítulos 5 e 6. Por fim, o Capítulo 7 cita as considerações finais, com alguns trabalhos futuros.

Capítulo

2

Simulação Heterogênea

Este Capítulo apresenta algumas ferramentas que lidam com Modelos de Computação (MoC) heterogêneos, em especial a ferramenta Ptolemy, em sua segunda versão. Será apresentada a utilidade de cada ferramenta e o motivo pela escolha do Ptolemy para este trabalho.

2.1. Ferramentas gerais de simulação

Existem várias ferramentas que analisam de um modo geral o desempenho de redes de sensores que podem ser classificadas em ferramentas analíticas, ferramentas de simulação, emuladores, e *deployment* em tempo real. As ferramentas de simulação, ou os chamados simuladores, são as utilizadas neste trabalho, por razões palpáveis, como: baixo custo, escaláveis, tempo e facilidade de implementação. Além disso, são consideradas ferramentas básicas e comuns de serem utilizadas para modelar ambientes reais com precisão razoável.

Ferramentas de simulação de propósitos gerais podem ser vistas em: *NS-2*, *OMNET++*, *J-SIM, GloMoSim, QualNet, OPNET, Matlab, SSF, JiST/SWAN* e a utilizada neste trabalho, o Ptolemy. Algumas dessas ferramentas podem trabalhar de modo mais específico com Redes de Sensores Sem Fio (RSSF), e outras não, porém, existem ferramentas específicas para as RSSF, como por exemplo, *SensorSim,TOSSIM NSrlSensorSim, Castalia, VisualSense, Viptos, Sidh, Prowler/JProwler*, *SENS, SENSIM, EYES, Mobility Framework, Mixim*. O Ptolemy, no caso, faz parte dos dois conjuntos de simuladores, os de propósito geral, e os específicos que trabalham com RSSF.

Vale salientar que os simuladores gerais não apresentam todas as características que projetos de sistemas embarcados podem ter, e por isso, eles possibilitam que funcionalidades, bibliotecas ou pacotes adicionais sejam desenvolvidos e integrados aos

propósitos gerais dos simuladores. Dessa forma, ou se tem simuladores de propósitos gerais com pacotes adicionais, ou se tem simuladores específicos de RSSF que em sua essência já apresentam características mais especializadas.

Existem desvantagens em usar *frameworks* específicos para RSSF, em que a principal é que o usuário está preso a uma única plataforma tanto em nível de *software* quanto em *hardware*, e também a uma única linguagem de programação. Essa característica faz com que os *frameworks* de propósitos gerais com suas devidas extensões também sejam úteis. Mesmo assim, sabe-se que os *frameworks* existentes para lidar com RSSF de forma específica são bastante eficientes para o propósito em questão.

2.2. O Ptolemy

A simulação é uma ferramenta essencial para o estudo de redes de sensores sem fio (RSSF) devido à inviabilidade das análises de forma profunda de tais redes, além das dificuldades de configurar experimentos reais.

Grande parte das ferramentas citadas anteriormente permitem pontos de extensão, nos quais os projetistas podem definir novas funcionalidades através da adição de códigos na ferramenta. Algumas são de código aberto e praticamente todas apresentam o Modelo de Computação (MoC) *Discret Event* para simulação, porém nenhuma possibilita a integração de diversos MoCs como o Ptolemy II, modelos como: C*ontinuous-Time, Syncrhonous/Reactive, Finite State Machine (FSM),* etc [24]. Tal capacidade de lidar com MoCs heterogêneos, possibilita o uso de diferentes tipos de aplicações, como por exemplo: dinâmica física de mobilidade dos nós dos sensores, circuitos digitais, consumo e produção de energia, processamento de sinal ou comportamento de *software* em tempo real, entre outras possibilidades.

Tanto o *J-SIM* (elencado anteriormente), quanto o Ptolemy II possibilitam compor modelos através do uso de componentes básicos ou atores básicos, respectivamente, proporcionando dessa forma maior flexibilidade dos modelos.

Em se tratando de desempenho, a simulação paralela pode executar e escalar melhor do que a simulação sequencial. O *trade-off* nesse caso é o aumento da complexidade da programação. A simulação paralela, como pode ser tratada em

GloMoSim, tem como objetivo principal o desempenho ao invés de escalabilidade, e pode simular até 10000 (dez mil) nós de sensores sem fio. A simulação distribuída de forma mais detalhada pode ser vista no Capítulo 4.

Quase todas as ferramentas e pacotes citados anteriormente apresentam o suporte a *interface* gráfica, que é o caso do *OMNET++* e de forma otimizada o *J-SIM* e o Ptolemy, os quais utilizam o poder da GUI para animação e depuração. Além disso, o Ptolemy e *OMNET++* são melhores que os demais no que diz respeito a algumas características, como: inspeção, modificação de parâmetros em tempo de execução, entre outras.

Adicionalmente, o Ptolemy II apresenta um editor gráfico com boa usabilidade. Os editores dos demais pacotes não apresentam tanta usabilidade, logo algumas vezes, é preferível utilizar os modelos orientados a *scripts* para criar novos modelos do que o editor gráfico, o que acaba retardando, de certa forma, o projeto dos modelos de simulação na maioria das vezes.

Ptolemy II é uma infraestrutura de *software* que atualmente está sob o controle e desenvolvimento do departamento da Universidade da Califórnia em *Berkeley*. O Ptolemy II pode ser visto como uma linguagem de programação visual que suporta hierarquias agrupando modelos de computação heterogêneos [25]. Ele aparece como sendo o único entre os vários *frameworks* existentes citados anteriormente que modela ambientes com modelos de máquinas de estado finito juntamente com diversos outros modelos quaisquer, ou seja, ele não está restrito para trabalhar somente com um modelo de computação único [26]. Ele também é o único que permite um sistema moderno de tipo no nível de abstração como atores, diferentemente das demais ferramentas que normalmente permitem somente abstração em nível de código [27]. Além disso, o Ptolemy II possibilita checar o tipo de dados dos pacotes que estão em comunicação em tempo de execução, através do chamado *Record Types*, o que ajuda na implementação e abstração dos modelos construídos. O seu objetivo principal está no projeto e simulação de sistemas heterogêneos complexos, permitindo diversos MoCs serem executados no mesmo modelo e de forma hierárquica, como processamento de sinais, controle de *feedback*, visualização 3D e *interfaces* de usuários [24] , [28].

Diante de todas as características citadas anteriormente, como: lidar com MoCs heterogêneos e assim com aplicações complexas embarcadas, facilidade do uso da interface gráfica, facilidade da depuração do modelo construído, projeto em alto nível de

abstração com uso de atores, checagem dinâmica dos dados, com o uso do *Record Types*, entre outras, foram determinantes para a escolha da ferramenta de simulação Ptolemy como estudo de caso deste trabalho.

O Ptolemy é *framework* baseado no Modelo de Atores, que são entidades que processam os dados presentes nas suas portas de entrada ou que criam e enviam dados para outras entidades por meio de suas portas de saída. Ele possibilita a modelagem de sistemas embarcados, particularmente aqueles que mesclam tecnologias, como dispositivos eletrônicos analógicos e digitais, dispositivos eletromecânicos e *hardware*s dedicados, além de sistemas com complexidade nos vários níveis de abstração [29]. Tais sistemas complexos normalmente necessitam de uma simulação prévia antes mesmo de sua concretização.

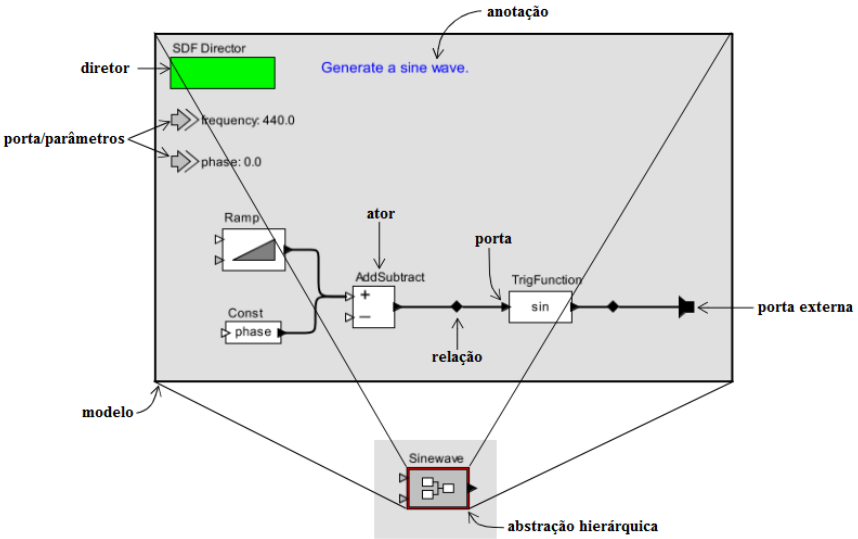
Em sua primeira versão, o Ptolemy I foi implementado todo na linguagem C. Porém, visando maior utilização em aplicações e pesquisas do mundo inteiro, e usufruindo os diversos benefícios da orientação a objetos, a segunda versão (Ptolemy II), foi escrita em linguagem Java e possui os seguintes pontos:

- Provê um rico conjunto de mecanismos de interação que estão embutidos nos domínios do Ptolemy II. Os domínios forçam os desenvolvedores de componentes a pensarem no padrão como os outros componentes já pertencentes ao domínio estão interagindo;
- Seus componentes são polimorfos no campo do domínio, isso significa que eles podem interagir com outros componentes mesmo esses sendo de domínios distintos;
- Possui um *framework* formal e abstrato que descreve modelos de computação (MoC) possibilitando maior facilidade de extensão desses modelos;
- Possui ferramentas para a simulação dos modelos, permitindo assim a execução de tais modelos como aplicações locais ou *web*, podendo ser utilizadas para apresentações à distância. Possui uma linguagem própria de marcação (*Mark-up Language*), baseada em XML (*MoML*), além de possuir uma ferramenta visual (*Vergil*), que facilita a realização e o entendimento de modelos mais complexos;
- Seu código é aberto e gratuito, além de possuir milhares de usuários em todo o mundo.

O Ptolemy interpreta a execução e a interação semântica de componentes através do que se chama Modelo de Computação, implementados através de Domínios. Dependendo do domínio, os componentes representam sub-rotinas, *threads*, processos, endereço IP, ou até mesmo partes mecânicas. Cada domínio segue um Modelo de Computação (vide Capítulo 3) bem definido que especifica a semântica formal do comportamento do componente. Mais especificamente, ele provê uma abstração de um conjunto de regras na qual cada ator se comporta. Essas regras podem incluir: a ordem em que os atores são executados, a progressão do tempo, a quantidade de *buffer* de memória que existe nas portas de cada ator, entre outros [28].

Os componentes do Ptolemy II são chamados de atores, que são conectados diretamente numa topologia de grafo através de *links*, chamados relações. A Figura 3 apresenta os principais componentes de um modelo no Ptolemy II. Cada ator possui portas de entrada e de saída, meio pelo qual a comunicação com outros atores é realizada. Os atores se comunicam entre si, através da troca de mensagens chamadas de *tokens*. Os atores produzem e consomem *tokens* durante a execução do ator. No intuito de facilitar a execução dos atores, cada domínio tem sua própria classe diretor e receptor. O diretor coordena a ordem e a execução dos atores dentro de um domínio. E o receptor contém o *buffer* necessário para o fluxo dos *tokens*. Ainda, em alguns domínios, o diretor também mantém o caminho do tempo e concorrência dos atores. A Figura 3 [28] apresenta alguns atores existentes nesse ambiente de simulação.

Figura 3. Ambiente do Ptolemy II



Capítulo

3

Modelos de Computação

Este Capítulo apresenta uma noção geral do conceito de Modelos de Computação, conhecido em inglês como *Models of Computation* (MoC). Serão expostos os principais modelos utilizados nas aplicações existentes e como tais modelos podem trabalhar de forma heterogênea.

3.1. Conceitos Gerais

Um sistema em chip, em inglês, *A system on a chip* (SoC) pode integrar microcontroladores, processadores de sinais digitais, memórias, *hardwares* customizáveis e reconfiguráveis, tudo isso na forma de FPGA (*Field Programmable Gate Arrays*) juntamente com uma estrutura de comunicação e conversores em um único chip digital-analógico (D/A) e analógico-digital (A/D). No total devem existir dezenas ou centenas de componentes desse tipo em um único SoC. Tais arquiteturas oferecem um potencial enorme. Entretanto, apresentam também enorme complexidade e heterogeneidade. Isto não se aplica apenas a *hardware*, mas também a *software*. Além disso, a complexidade de tais sistemas cresce muito mais rápido do que o tamanho do sistema em si, devido à interação entre os componentes ser intensa. Na verdade, a comunicação dentro do sistema está se tornando o principal fator na hora de projetar, validar e analisar o desempenho de tais sistemas. Consequentemente, problemas de comunicação, sincronização e paralelismo aparecem como um papel importante em todos os projetos do tipo [30].

Os sistemas embarcados são objetos complexos que contém um percentual significativo de dispositivos eletrônicos que interagem com o mundo real através de dispositivos sensitivos e acionáveis. Um sistema é heterogêneo quando existe a

coexistência de um grande número de componentes de vários tipos e funções, interagindo entre si.

Antigamente, os esforços para o projeto de tais sistemas era focado somente no *hardware*, deixando o projeto do *software* para ser feito como implementação final. Porém, atualmente mais de 70% do custo de desenvolvimento dos sistemas complexos é atribuído ao desenvolvimento do *software*.

No intuito de gerenciar a complexidade e a heterogeneidade de aplicações SoC, Edward *et at.* [31] acredita que a abordagem do projeto deve ser baseada no uso de um ou mais métodos formais para descrever o comportamento do sistema em um nível alto de abstração, os chamados *Model of Computation* (MoC), antes mesmo da decisão de decompor esse sistema em *hardware* e *software*. A implementação final de tais sistemas deve ser feita utilizando síntese automática desse alto nível de abstração para confirmar que a implementação está "correta por construção" [30].

Validação através de simulação ou verificação deve ser feita nos níveis mais altos de abstração. Desse modo, o presente trabalho lida com a simulação como forma de validação de tais sistemas que podem ser descritos em alto nível de abstração, apresentando complexidade e heterogeneidade na forma de comunicação, sincronização e paralelismo.

O conceito de MoC foi expandido e atualizado [29], como sendo a representação do estado do sistema, a comunicação dentro do sistema e a forma como a computação acontece entre uma estrutura de componentes computacionais, dando efetivamente uma semântica a tal estrutura.

O termo MoC é usado para focar especificamente assuntos relacionados a concorrência e o tempo. Consequentemente, embora ele tenha que ser definido de forma diferente por vários autores distintos, usa-se esse termo para definir a representação do tempo e a semântica da comunicação e sincronização entre processos em uma rede de processos. Logo, MoC define como a computação acontece em uma estrutura de processos concorrentes, por isso fornece a semântica para tal estrutura [29] [32]. Essa semântica pode ser utilizada para elaborar uma máquina abstrata que possibilita executar um modelo. É interessante compreender que linguagens de programação não são modelos computacionais na sua essência, mas possuem como base tais modelos. Por exemplo, linguagens como VHDL, *Verilog* e *SystemC* compartilham o mesmo MoC, no caso, *discrete time* ou *event-driven.* Por outro lado, as linguagens também podem ser utilizadas para possibilitar o uso de mais de um modelo computacional. Em *ForSyDe*

[33], a linguagem funcional *Haskell* [34] é usada para expressar diversos modelos de computação, em que bibliotecas foram criadas para MoCs que lidam com sincronização, com o fluxo dos dados e com eventos discretos. *SystemC* também possibilita o uso de SDF (*Synchronous Data Flow*) e CSP (*Communicating Sequential Process*) como uma extensão ao seu modelo de computação nativo, no caso, o *Discrete Event*.

Em outras palavras, de acordo com Lavagno *et al* [35], MoC é composta de um mecanismo de descrição (sintaxe) e regras para a computação do comportamento do sistema (semântica). Ele é escolhido de acordo com cada situação, por exemplo, um MoC pode ser bom para descrever funções de transferência de dados complicadas e pode não ser nada bom para controles complexos, enquanto que outros MoCs podem ser ótimos para projetar controles complexos.

Os modelos de computação englobam [35]:

- Os tipos de relação que são possíveis em uma semântica denotativa.
- O comportamento de uma máquina abstrata em uma semântica operacional.
- O comportamento individual especificado e composto.
- A forma como a hierarquia abstrai tal composição.
- O estilo da comunicação.

O projeto, em todos os níveis de abstração na hierarquia, desde a especificação funcional até a implementação final, é geralmente representado como um conjunto de componentes, em que cada um pode ser considerado blocos monolíticos isolados, os quais interagem um com o outro e com um ambiente que não faz parte do projeto. O modelo de computação define o comportamento e a interação entre esses blocos. Tal interação segue um comportamento específico, que de um modo geral lida com as seguintes características: concorrência, heterogeneidade e hierarquia. Essas características são exemplificadas a seguir.

3.1.1. Concorrência

Atualmente, a complexidade de sistemas embarcados implica na evolução dos comportamentos dos projetos. Uma das funcionalidades complexas introduzidas nesse novo contexto é a concorrência. O sistema deve permitir a execução independente de diversos processos ao mesmo tempo. Um sistema de comunicação é introduzido entre os módulos de processos distintos. Porém, alguns MoCs não permitem a concorrência entre os processos, ou seja, não possibilitam que a execução destes seja feita de forma

paralela. Por outro lado, existem MoCs específicos que permitem que a execução dos processos seja realizada concorrentemente, que é o caso do SDF.

O uso de execução paralela é bastante importante em algumas situações, entretanto, sabe-se que o controle de tais execuções apresenta maior nível de dificuldade em comparação à execução serial, no que diz respeito ao projeto do comportamento. O MoC mais comum para trabalhar como controlador nesse caso é o FSM. Este é baseado em estados alcançáveis. Logo, a natureza finita do FSM (número finito de estados) provê segurança no controle do modelo, pois a principal função do controlador é prevenir que estados perigosos sejam alcançáveis. A união entre esses dois MoCs pode ser vista na Seção 3.2.

Essa abordagem permite que se combine o modelo de concorrência do SDF com o uso do FSM, como papel de controlador.

3.1.2. Hierarquia

A maior fraqueza do MoC FSM é o seu grande número de estados e transições, o que torna a complexidade do problema maior, no sentido em que ela pode se tornar difícil para representar, analisar e entender o sistema. Nesse caso, é interessante o uso de hierarquias.

A hierarquia é um conjunto de sistemas dentro de outro sistema. Como pode ser visto posteriormente (Seção 3.2), os MoCs são representados pelos estados e pelas arestas. A hierarquia permite a representação do comportamento de cada estado ou de cada aresta por outro MoC qualquer, o que significa que quando se está dentro de um estado ou quando uma aresta específica é chamada, o sub-MoC interno será ativado, e caso este sub-Moc contenha outro sub-Moc interno, este por sua vez, também será ativado e assim continua até o fim da hierarquia.

3.1.3. Heterogeneidade

Heterogeneidade significa o uso de elementos distintos no mesmo sistema global [36]. Precisamente, no escopo deste trabalho, por razões da diversidade do comportamento de sistemas embarcados, a heterogeneidade usa diferentes comportamentos de MoCs cujas propriedades são boas para representar os subsistemas em questão. No caso, SDF é adequada para vários tipos de computação, mas não para controle lógico. O *Discrte Event* é o mais adaptado para descrição do *hardware*, mas a

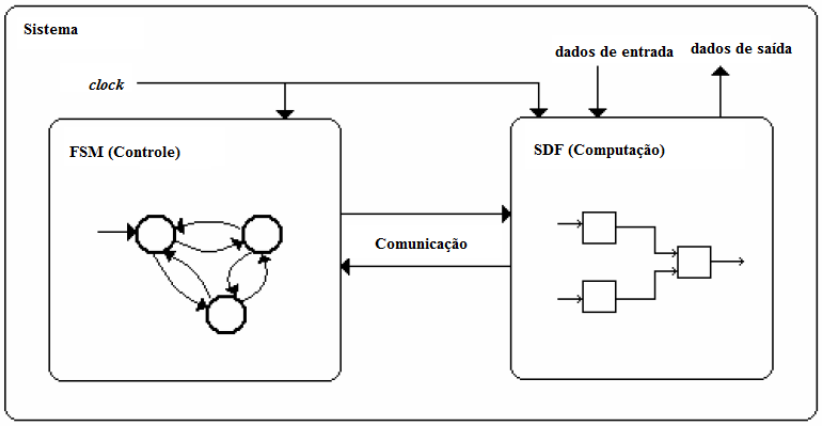
sua dependência de tempo não o permite que alcance computação mais abstrata. FSM é um MoC que representa um tipo de modelo complementar, pois normalmente é utilizada como controlador, mas não é apropriado para computação numérica, por exemplo. Então, como existem diversas funções que são modeladas, vários MoCs distintos podem ser usados no mesmo sistema para tentar suprir todas essas demandas. Todos os MoCs citados serão detalhados na Seção 3.2.

3.2. Unindo dois MoCs

Como mencionado anteriormente, a diversidade de comportamento dos subsistemas implica no uso de vários MoCs distintos dentro de um mesmo sistema. Foi citado também que o modelo FSM pode auxiliar o SDF adicionando o controle da computação que nele não existe. Essa abordagem lida com concorrência heterogênea. Todo o sistema age como um *wrapper* que provê ciclos de *clocks*, consequentemente todos os dados gerais para os dois modelos concorrentes interagem com o uso de um protocolo de comunicação baseado em eventos.

A Figura 4 mostra a simulação de um modelo heterogêneo que apresenta uma comunicação entre o MoC FSM e o MoC SDF [36].

Figura 4. Modelo Heterogêneo FSM com SDF [36]



Para melhorar o desempenho do comportamento da computação, o desenvolvimento de um comportamento heterogêneo hierárquico é bastante útil. Na hierarquia observou-se que um MoC pode ser inserido dentro de um estado de um super MoC de um mesmo tipo. Quando o super estado é alcançado significa que o sub MoC já está sendo executado. Dessa forma, a seguir será apresentada a heterogeneidade juntamente com a hierarquia, dos dois modelos já citados, o FSM e o SDF, de duas

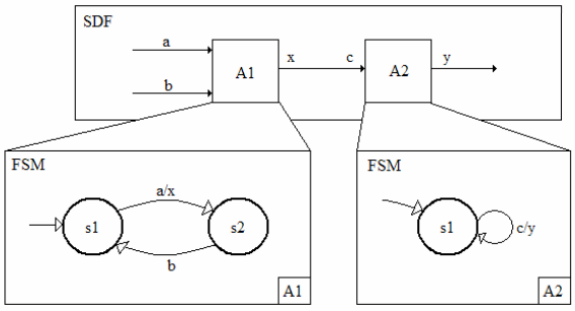
formas distintas, em que a primeira apresenta o FSM como um sub MoC do SDF, e a segunda, o SDF como sub Moc do FSM.

3.2.1. FSM dentro do SDF

FSM dentro de um ator SDF deve ser feito com semântica externa que é originada do SDF. De acordo com o modelo do SDF, o FSM deve consumir e produzir um número específico de *tokens*. A tradução entre os *tokens* de entradas no ator do SDF para os dados de entrada do FSM deve ser feita. Nesse caso, os *tokens* consumidos pelo SDF são traduzidos para uma entrada cujo alfabeto é conhecido pelo FSM, que pode reagir a mesma.

A Figura 5 apresenta como acontece tal conversão. No caso, o ator A1 tem duas portas de entradas (entrada "a" e entrada "b"). Isso significa que o ator A1 consome um *token* da entrada "a" e um da entrada "b". A computação do ator irá diferir dependendo da informação que foi recebida na entrada. Os *tokens* de entrada do ator referido são traduzidos para o alfabeto correspondente do FSM. Nesse exemplo, a presença ou ausência de *tokens* de entrada na entrada "a" e/ou "b" do ator A1 irá ser traduzida para o FSM A1 através da presença ou ausência do evento "a" e/ou "b". O maior problema nesse caso seria o formalismo dos *tokens* de saída. A ausência de qualquer evento no sub-FSM irá produzir um *token* de saída. Esse *token* irá traduzir a ausência do evento que pode ser interpretado pelo SDF [36].

Figura 5. FSM dentro do SDF [36]



Para entender melhor como este modelo realmente funciona é interessante mostrar um exemplo, fornecendo alguns *tokens* de entrada. Como não há nenhum *token* inicial no arco entre A1 e A2, então A1 deve ser executado primeiro. Supõe-se que um *token* é enviado para a entrada "a" e nenhum *token* para a entrada "b". Executando o ator A1, o *token* presente em "a" será consumido. A informação fornecida pelos *tokens* de entrada é "presença do evento 'a' e ausência do evento 'b'". A computação interna

do ator A1 é observada. O estado corrente do FSM A1 é s1 e o evento “a” está presente. O estado do A1 muda para s2 e a saída é “x”. A presença da saída na visão do FSM produz também um *token* na saída “x” do ator A do SDF. Como c = x está presente na entrada do ator A2 do SFD, então A2 é executado. No A2 do FSM, o estado corrente é s1. Para o evento “c”, ele fica no s1 produzindo a saída “y”. Essa saída no FSM cria uma *token* de saída na saída “y” do ator A2 do SDF.

Um exemplo simples elaborado pelo autor deste trabalho no ambiente do Ptolemy II é mostrado nas Figura 6 e Figura 7.

Figura 6. FSM dentro do SDF no ambiente Ptolemy II [elaborado pelo autor]

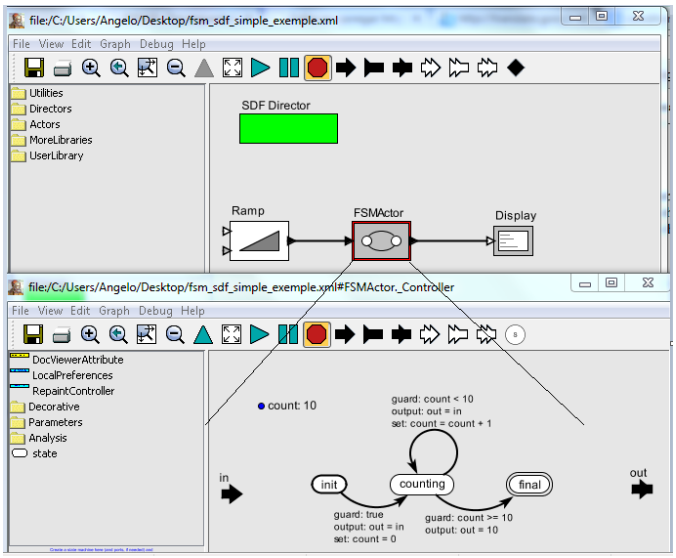
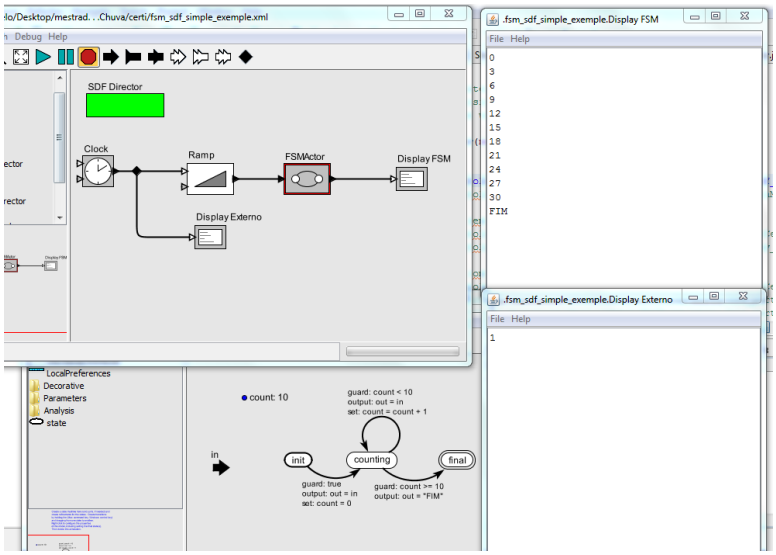


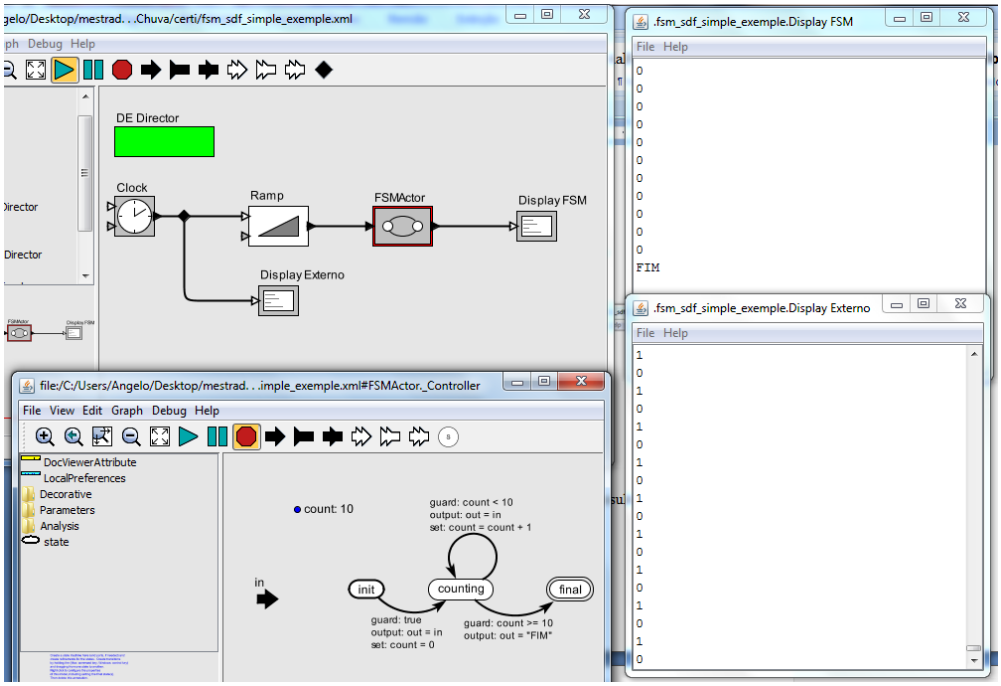
Figura 7. Execução do modelo heterogêneo - FSM dentro do SDF no Ptolemy II [elaborado pelo autor]



Se, ao invés de usar o SDF como o MoC externo, usar o DE, o resultado será diferente, pois o uso do DE considera o tempo, ou seja, a cada evento, a cada novo pulso no *clock*, o processo entrará novamente no FSMActor. Como isso será feito a

cada *clock*, e o *Ramp* inicia-se com 0, então sempre a saída continuará sendo 0. No caso do SDF, como ele considera somente um evento, com a quantidade de entrada sendo a mesma da saída, então a porta de entrada do *Ramp* será exercitada apenas uma única vez, e ele sempre será somado de 3 em 3. No caso do DE isso não ocorre, pois o evento será iniciado a cada *clock*, e por isso o *Ramp* será reiniciado sempre. A Figura 8 mostra a execução deste caso.

Figura 8. Execução dentro do modelo heterogêneo - FSM dentro do DE no Ptolemy II [elaborado pelo autor]



3.2.2. SDF dentro do FSM

Quando alguns estados do FSM são refinados dentro do SDF, o mesmo processo sempre é aplicado: se o estado refinado do FSM é alcançado, então exatamente uma iteração do SDF interno é feita, seguido da reação do FSM devido ao novo dado gerado pela iteração.

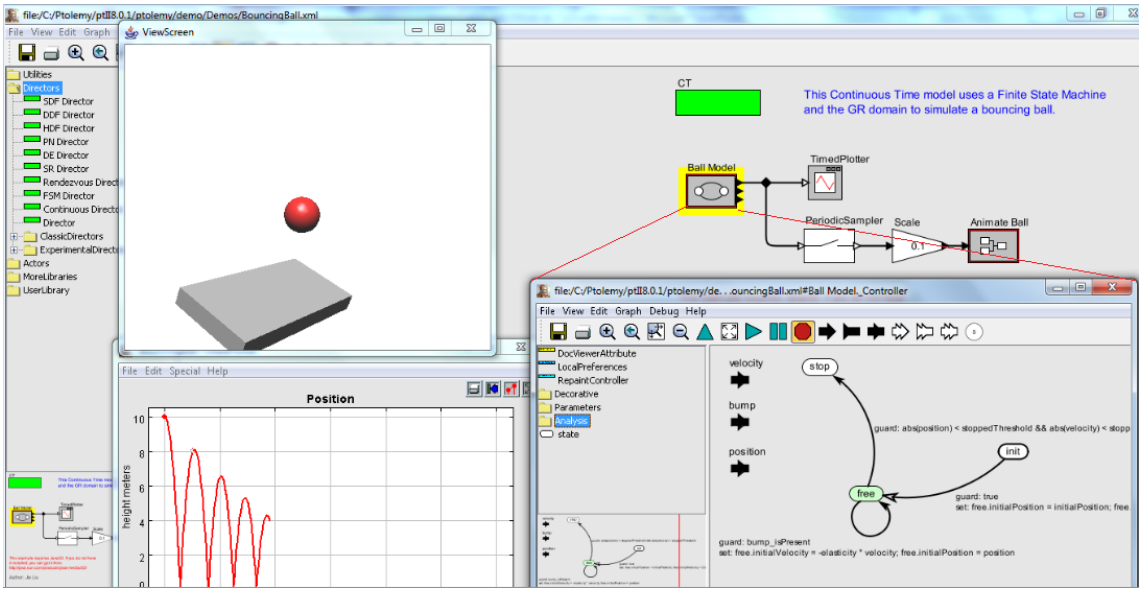
No caso do SDF homogêneo, a computação se torna mais simples, pois a entrada no FSM fornece um símbolo que é traduzido para um *token* no SDF. Mesmo com ausência de eventos sendo traduzida, o SDF recebe um *token* que é consumido no intuito de executar. Depois da iteração, um novo *token* é produzido que é traduzido em uma nova entrada para o nível do FSM.

3.3. Principais Modelos de Computação

O *framework* de modelagem conhecido que lida com modelagem heterogênea é o Ptolemy. Ele permite a integração de um amplo alcance de diferentes MoCs através da definição de regras de interação de diferentes domínios de MoCs.

Um exemplo prático de uma aplicação que vem embutida no Ptolemy, chamada *Bouncing Ball* (Figura 9), utiliza o modelo CT como controle externo global do tempo (ex. como o gráfico será plotado com o passar do tempo), e para fazer o controle inteiro da lógica (ex. aumentar ou diminuir a velocidade da bola), utiliza o FSM.

Figura 9. Aplicação *demo Bouncing Ball* do Ptolemy II [*printscreen* da aplicação *demo* do Ptolemy]



O benefício do Ptolemy II é possibilitar o uso de diferentes tipos de modelos computacionais juntos, fazendo com que o ambiente computacional se torne mais eficiente e completo. Assim, é possível fazer várias tarefas no mesmo ambiente, característica essencial na modelagem de sistemas embarcados.

Sistemas embarcados geralmente são formados por vários componentes realizando computação através de MoCs heterogêneos, como citados anteriormente. O Ptolemy II provê uma variedade enorme de modelos computacionais que lidam com a questão da concorrência e a questão de enviar e consumir dados ao longo do tempo de vários modos diferentes. Tais MoCs normalmente são divididos de acordo com a abstração do tempo. Estes são distinguidos entre modelos de tempos discretos e modelos síncronos, nos quais o ciclo que denota uma noção abstrata de tempo, e por fim, os modelos que não lidam com noção de tempo [32], [37]. A seguir serão apresentados tais modelos de computação com suas propriedades chaves.

3.3.1. Modelos de tempo contínuo (*Continuous time model*)

Quando o tempo é representado por um conjunto contínuo, usualmente números reais, está se tratando de *Continuous Time* MoC (Tempo Contínuo). Alguns exemplos de simuladores de um modo geral podem ser citados, como: *Simulink* [38], [14], VHDL-MAS e Modelica [39]. O comportamento é tipicamente expresso como uma equação em números reais. Simuladores para esse MoC são baseados em solucionadores de equações diferenciais que computam o comportamento do modelo.

Devido a necessidade de solucionar equações diferenciais, simulações com modelo continuo são muito lentas. Então, somente uma pequena parte do sistema é normalmente modelada com esse MoC, como componentes analógicos e sinais mistos. Para ser possível modelar e analisar um sistema completo que contém componentes analógicos, simuladores como VHDL-MAS precisam ser desenvolvidos. Eles possibilitam a modelagem de partes digitais puras no MoC *Discrete Time* (tempo discreto) e as partes analógicas no MoC *Continuous Time* (tempo contínuo). Isso possibilita que sistemas completos de simulação sejam construídos com o desempenho aceitável. Além de ser um exemplo típico em que modelos heterogêneos baseado em múltiplos MoC apresentam benefícios claros.

O CT é utilizado na modelagem de dinâmicas físicas, principalmente em *Cyber-Physical Systems*. Este modelo lida com mudanças de valores no tempo.

3.3.2. Modelos de tempo discreto

Modelos em que os eventos são associados com tempos instantâneos e esse tempo é representando por um conjunto discreto, como de números inteiros ou naturais. São chamados de modelos de tempos discretos. Algumas vezes esse mesmo grupo de MoC é chamado de modelos de eventos discretos (DE). De forma mais rígida “eventos discretos” e “tempos discretos” são independentes, são conceitos ortogonais. Por exemplo, no caso de modelos de eventos discretos, os valores dos eventos é que fazem parte de um conjunto discreto, e não é o tempo que rege o modelo. Dessa forma, na prática, algumas combinações podem ocorrer: tempo contínuo/evento contínuo, tempo contínuo/evento discreto, tempo discreto/evento contínuo e tempo discreto/evento discreto.

O DE é utilizado normalmente em simulações de *hardware* e na modelagem de comunicação em rede e sistemas de tempo real. Tanto o VHDL como o Verilog usam o

modelo de tempo discreto para a semântica de suas simulações. Este modelo reage a eventos que ocorrem em tempo de execução e produzem outros eventos, ou no mesmo tempo específico, ou em algum momento no futuro. A execução é de forma cronológica, com estados de transições de eventos [40].

Um simulador que lida com MoCs tempo discreto é normalmente implementado com uma fila de eventos globais que automaticamente ordenam a ocorrência dos eventos.

3.3.3. Modelos síncronos

Em modelos síncronos, o tempo também é representado por um conjunto discreto, como inteiros, mas a unidade elementar do tempo não é uma unidade física, mas mais abstrata devido a dois mecanismos de abstração:

1. O tempo das atividades e eventos não é precisamente definido, mas é restrito pelo início e o fim de uma janela elementar de tempo.
2. O tempo de eventos intermediários que não são visíveis no final da janela elementar do tempo é irrelevante e pode ser ignorado.

Em cada unidade de tempo, todos os processos avaliam uma vez e todos os eventos que ocorrem dentro do processo são considerados para ocorrer de forma simultânea.

A abordagem síncrona considera sistemas reativos ideais para produzir as saídas de forma síncrona com as entradas. E as reações não levam o tempo em consideração. Isso implica que a computação de um evento de saída é instantâneo. Essa abordagem síncrona leva a uma clara separação entre a computação e a comunicação. Um relógio global dispara a computação que é conceitualmente simultânea e instantânea. Essa suposição libera o projetista de modelar mecanismos de comunicação complexos e provê uma base sólida para métodos formais.

A técnica do projeto síncrono vem sendo usada para o projeto de *hardware* para circuitos síncronos baseados em tempo. O comportamento do circuito pode ser descrito deterministicamente independente de detalhes de tempo das portas, através da separação de blocos combinacionais com cada um dos registros de tempos. A implementação irá ter o mesmo comportamento do circuito abstrato de acordo com a suposição que os blocos combinacionais são rápidos o suficiente.

Benveniste e Berry [1] definem o sistema SR (*synchronous reactive*) como um sistema no qual, tanto a computação em um componente, quanto a comunicação entre eles, acontecem instantaneamente. Outra análise interessante é que o SR possui o controle preciso sobre o tempo dos eventos.

Este modelo possibilita um diálogo instantâneo, em que um componente pode fazer uma consulta a outro, receber uma resposta, executar alguma computação, e produzir uma saída, tudo isso com o *delay* de tempo zero [42]. O sistema é considerado reativo porque eles respondem a entradas do ambiente.

Normalmente este tipo de modelo é utilizado quando se trata da reação de objetos em movimento, ou a própria percepção do ser humano em sua volta, pois tais fatos são instantâneos. Este tipo de modelo se encontra também na modelagem e projeto de sistemas de *software* concorrentes críticos, já que necessita de *delay* zero, ou o mínimo possível.

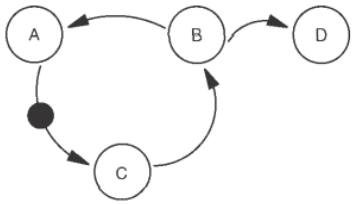
3.3.4. Modelos sem tempo

O modelo de computação que está presente neste grupo de modelos é o DF (*Data flow*) [43], ou chamado *Data Flow Process Networks*, que é uma variação das redes dos processos de *Kahn*. No processo de *Kahn*, a comunicação dos processos via rede um para outro se dá via canais infinitos de filas (FIFO). A escrita nesses canais é considerado algo "não blocado", pois eles sempre serão bem sucedidos e nunca irão parar o processo, enquanto que a leitura a partir desses canais é "blocado", ou seja, o processo que lê de um canal vazio irá parar e apenas poderá continuar quando o canal contiver dados (*tokens*) suficientes.

Os processos na rede de processos de *Kahn* são monotônicos, o que significa que eles só precisam de informações parciais do *stream* de entrada para produzir informações parciais para o *stream* de saída. A monotonicidade permite o paralelismo, pois o processo não precisa de todo o sinal de entrada para começar a computação dos eventos de saída. Vale salientar que os processos não são permitidos realizar testes nos canais de entrada se existir *tokens* sem serem consumidos. Além disso, existe uma ordem total dos eventos dentro do sinal. Entretanto, não existe ordem de relação entre os eventos em diferentes sinais. Diante do exposto, os processos de *Kahn* são considerados "parcialmente ordenados", o que os classificam como modelo sem tempo [37].

Um programa que lida com *data flow* é um gráfico direcionado que consiste em nós (atores) que representam comunicação e os arcos que representam sequências (*streams*) ordenadas de eventos (*tokens*) como pode ser visto na Figura 10 em seguida. Os círculos vazios são representados pelos nós, as setas representam os *streams* e o círculo preenchido representa o *token*. O fluxo dos dados na rede pode ser hierárquico, pois os nós podem representar um fluxo de dados no gráfico [37].

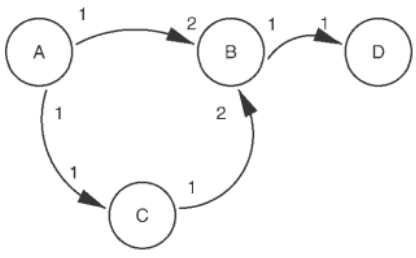
Figura 10. Redes de processo Data Flow [37]



A execução do processo do fluxo de dados é uma sequência de execuções ou avaliações. Para cada execução, *tokens* são consumidos e *tokens* são produzidos. O número de *tokens* consumidos e produzidos pode variar a cada diferente execução e assim é definido nas regras da execução de cada ator do fluxo de dados. O processo do fluxo de dados na rede parece ser bastante útil em aplicações de processamento digitais de sinais. Quando se implementa o fluxo de dados na rede em um único processador, a sequência de execuções, também conhecida como planejamento de execuções, precisa ser encontrada. De forma geral os modelos de fluxo de dados (*data flow*) não podem decidir se tal planejamento existe, pois essa resposta depende dos dados de entrada.

Uma variação do fluxo de dados simples é o *Synchronous Data Flow* (SDF) [44] que acrescenta mais restrições no modelo de fluxo de dados, pois ele requer que o processo consuma e produza um número limitado e fixo de *tokens* para cada execução. Com tal restrição é possível testar a eficiência, se um planejamento finito estático existe. Se existir, ele pode ser efetivamente computado. A Figura 11 mostra o processo SDF. O número de arcos mostra quantos *tokens* são produzidos e consumidos durante cada execução. Um planejamento possível para a rede SDF exibida pode ser {A,A,C,C,B,D} [37].

Figura 11. Redes de processo *Synchronous Data Flow* [37]



Neste grupo de modelos de computação, pode-se citar, por exemplo, dois: PN (*Process Network*) e o já citado SDF (*Syncrhonous Data Flow*). Para a modelagem de *hardware* e aplicações embarcadas, é extremamente necessário um modelo de concorrência, e o PN é um mecanismo para prover tal concorrência. Este modelo permite operações assíncronas para componentes de processos em uma rede e vem sendo estendido para possibilitar mutações em redes para algumas formas não determinísticas. Os receptores neste modelo implementam uma fila (FIFO – *First In First Out*), nos quais se utilizam da estratégia de bloquear para a leitura e liberar para a escrita. O PN é considerado um modelo que não trabalha com noção de tempo, sendo utilizado principalmente para modelar sistemas distribuídos assíncronos e processamento baseado em *streams* (vídeo e áudio), que são comumente encontrados em aplicações eletrônicas [21] e [45].

Já o SDF é muito usado para modelar processamento de sinais e aplicações multimídia. Além disso, algumas linguagens baseadas em SDF, como *StreamIt* [46], têm sido aplicadas na programação de *data flows* e multicores [47]. Este modelo consiste em uma rede de componentes conectados por arcos por onde passam os dados. Este modelo trabalha com um número fixo de *tokens* para cada entrada e saída, de acordo com a execução. Quando um ator é executado neste modelo, ele consome um numero fixo de *tokens* de cada porta de entrada e produz um número fixo de *tokens* para cada porta de saída. O fluxo de dados que passará nos vários nós existentes tanto como entrada, como saída é planejado estaticamente, ou seja, em tempo de compilação [44].

Capítulo

4

Simulação Distribuída

Simulação de sistemas computacionais é uma computação que modela o comportamento de sistemas reais e imaginários no decorrer do tempo [48]. Na prática, ela consiste em um conjunto de técnicas que são fundamentais para avaliação do desempenho de sistemas existentes, para o estudo de novas soluções e para a criação de um mundo virtual (ex. *games online*, ambientes digitais virtuais) [49].

Existem várias razões por trás do uso de técnicas de simulação. Por exemplo, um sistema que precisa ser avaliado, mas que não pode ser construído, por razões de altos custos; um sistema que precisa ser testado, mas que o teste em determinadas aplicações pode ser bastante perigoso, colocando em risco a vida de pessoas, além de que em alguns casos, testes de stress são de certa forma impossíveis de serem realizados; e a exploração de diversos tipos de solução para um sistema, em que a melhor precisa ser descoberta e assim escolhida. A demanda por sistemas cada vez mais complexos levou a uma ampla difusão de técnicas de simulação, ao aumento da pesquisa nesse campo e a disponibilização de diferentes tipos de ferramentas de *software* [49].

No decorrer dos anos, diversos paradigmas de simulação têm sido propostos, cada um apresentando vantagens e desvantagens. Entre esses, pode-se citar o conhecido MoC *Discrete Event Simulation* (DES) [50] que é bastante poderoso em termos de expressividade e é de fácil entendimento pelos desenvolvedores de modelos de simulação. Tal modelo já foi citado previamente no Capítulo 3. Nesse caso, todas as atividades são executadas por uma única unidade de processamento, e isso significa que as atividades estão sendo executadas de forma sequencial (monolítica). A unidade de execução única é responsável por modelar todo o sistema e por fazer o gerenciamento de toda a evolução do sistema de forma sequencial. A grande vantagem dessa abordagem é a sua simplicidade, mas tal característica introduz algumas severas limitações: as fontes de memórias de uma unidade de execução única podem ser

insuficientes para a modelagem de sistemas mais complexos. Além disso, a quantidade de tempo necessário para completar a simulação pode ser também bastante excessiva [51] [49].

Uma alternativa para esse problema é o chamado *Parallel Discrete Event Simulation* (PDES) [52], que apresenta unidades de execução múltiplas interconectadas. Nesse caso, cada unidade de execução gerencia somente a sua parte da simulação modelada. Então, é possível representar modelos extensos e complexos usando fontes agregadas de várias unidades de execução. Diferentemente do DES, no PDES cada unidade de execução tem que gerenciar sua lista de eventos locais e os eventos gerados localmente precisam ser enviados a unidades de execução remotas. Além disso, o seu processamento precisa ser sincronizado com o resto do simulador, o que torna a gerência da simulação mais complicada. Logo, o grande benefício do uso de computação e memória de forma agregada é que ele permite que a simulação de grandes sistemas complexos seja possível, e também, em vários casos, a simulação distribuída ou paralela de eventos concorrentes [53] podem trazer um *speedup* significativo para a execução das simulações [49]. *Speedup* pode ser definido como o tempo de execução da computação sequencial dividido pelo tempo de execução da computação paralela [54], ou seja, o quanto a computação distribuída pode ser mais rápido que a serial.

Algumas simulações complexas envolvem diferentes tipos de simulações individuais, combinadas com outros aspectos em um ambiente global que precisa ser simulado. Normalmente, as simulações de alguns desses tipos de componentes já existem, tendo sido desenvolvidas com um propósito bem específico. Porém, pode-se observar também a necessidade de fazer diversas modificações quando é necessário adaptar um componente de simulação de forma que este seja integrado em uma nova simulação combinada. Em alguns momentos, sabe-se que é mais fácil implementar completamente um novo simulador de um componente de sistema do que modificar um já existente.

Então, ao invés de se ter um único programa sendo executado em um único computador, se faz necessário, algumas vezes, ter vários programas sendo executados em diferentes tipos de computadores distribuídos e interagindo um com o outro em tempo de execução.

4.1. Simulação Paralela e Distribuída

Uma das maneiras mais simples e genérica de definir Simulação Distribuída e Paralela, do inglês *Parallel and Distributed Simulation* (PADS) é: qualquer simulação, na qual exista mais de uma unidade de processamento envolvida [55]. Existem várias razões para investir nas PADS, como: obtenção de resultados de forma mais rápida, simulação de cenários maiores, integração de simuladores geograficamente distribuídos, composição de diferentes modelos de simulação em um único simulador [48].

As PADS são obtidas através da interconexão de um conjunto de componentes, chamados de Processos Lógicos (PLs) [56]. Então, cada PL é responsável por gerenciar a evolução de sua parte do sistema e a interação com outras PLs no que diz respeito à sincronização e distribuição de dados. Em termos práticos, cada PL é executada por um processador (ou um core em arquiteturas multicores). O tipo de rede que interconecta os processadores tem sua importância, na qual afeta diretamente o desempenho e as características do simulador.

É interessante também entender a diferença existente entre simulação paralela e distribuída, pois este trabalho lida de forma mais enfática com a simulação distribuída. Normalmente, a simulação paralela é utilizada em duas situações, ou quando os processadores tem acesso a uma única memória de forma compartilhada ou na presença de uma rede altamente acoplada. De forma contrária, em se tratando de simulação distribuída, esta é comumente utilizada em casos em que a arquitetura de rede é fracamente acoplada, ou seja, apresenta várias memórias e/ou processadores de forma distribuída. Obviamente, as plataformas do mundo real apresentam uma mistura desses dois tipos de simulação.

As PADS apresentam de um modo geral, as seguintes características:

- O modelo que representa o sistema simulado tem que ser dividido em componentes (os chamados PLs) [57]. Em alguns casos, esse particionamento é guiado de acordo com a estrutura e a semântica do sistema simulado, por exemplo, se o sistema é composto por partes, cada uma deve ter seu próprio comportamento e sua própria estrutura, porém interagindo com as demais. Em outros casos, a tarefa de divisão pode ser mais complexa, por exemplo, um sistema monolítico dificilmente pode ser dividido em partes. Em todos os casos, quando há particionamento, vários aspectos distintos devem ser levados em consideração. Por exemplo, tanto o uso da rede de forma mais amena pelo

simulador quanto o balanceamento de carga da execução da arquitetura apresentam impactos profundos no desempenho do simulador.

- Os resultados da simulação paralela e/ou distribuída somente podem ser considerados corretos se os resultados são idênticos aos obtidos pelo uso da simulação sequencial. Isso seria impossível se os PADS não implementassem algum tipo de sincronização entre as diferentes partes que compõe o simulador. Algoritmos específicos são necessários para a sincronização das PLs envolvidas durante os processos de execução.
- Cada componente do simulador irá produzir atualizações de estado que são relevantes para os demais componentes. A distribuição dessas atualizações durante a execução é chamada "distribuição de dados" e devido ao *overhead* ela não pode ser implementada utilizando *broadcast*. A abordagem correta é comparar os dados produzidos e consumidos baseados em critérios de interesses, ou seja, somente os dados necessários devem ser entregues aos componentes interessados [58].

4.2. Sincronização

Implementar PDES em arquiteturas PADS requer que todos os eventos gerados sejam marcados com tempo (*timestamped*) e entregues seguindo uma abordagem baseada em mensagens. Dois eventos são ditos que estão em ordem casual se um deles pode ter consequências no outro [53]. Essa característica é bem fácil de ser satisfeita em simulações sequenciais, nas quais todos os eventos têm de ser considerados em ordem não decrescente respeitando o seu *timestamp*. Em arquiteturas de simulação paralela ou distribuída, os componentes podem ser processados em velocidades diferentes, a rede pode introduzir um *delay* imprevisível e ainda poderá haver perdas nas mensagens entregues. Para garantir que as PADS não violem o princípio da causalidade, todas as PLs envolvidas na execução da simulação têm de ser coordenadas utilizando algum tipo de algoritmo de sincronização.

O planejamento dos eventos da simulação entre diferentes Pls acontecem na troca de mensagens de notificações de eventos, nas quais carregam o conteúdo e o momento em que aconteceu o evento (*timestamp*). No intuito de assegurar resultados de simulação corretos, mecanismos de sincronização são usados para manter a execução do *timestamp* dos eventos de cada PL ordenada durante a simulação.

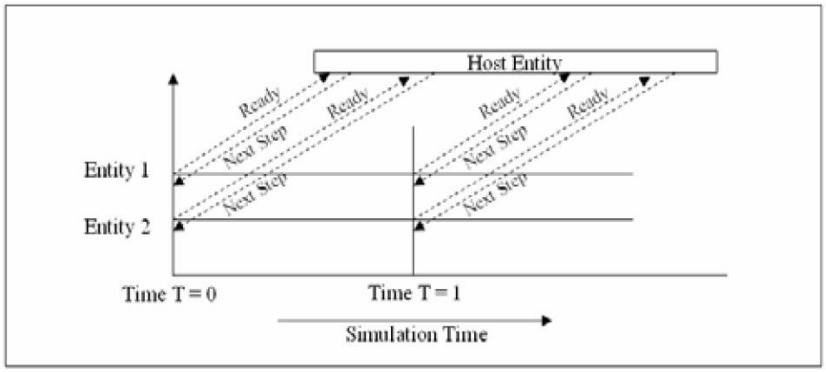
Nas últimas décadas, diferentes abordagens e variações foram investigadas, mas tratando de forma simplificada, podem-se citar três métodos principais: *time-stepped*, conservativo, otimista. Tais abordagens, nos últimos anos, estão sendo bastante investigadas e estudadas, e por isso diversas variações também estão sendo propostas. Pode-se analisar que o desempenho de algoritmos de simulação depende fortemente de vários fatores como: o modelo de simulação, o ambiente de execução e o cenário específico. Predizer o desempenho das PADS é bastante difícil, pois esta depende de vários fatores, alguns sendo estáticos e conhecidos no decorrer do tempo, enquanto outros são desconhecidos ou dependem de condições em tempo de execução [56].

4.2.1. Time-stepped

O tempo de simulação é dividido em *timesteps* (passos de tempo) de tamanhos fixos e cada PL só pode proceder a execução do próximo *timestep* somente quando todas as outras PLs tiverem completadas a elaboração do seu *timestep* corrente [59]. Sobre o ponto de vista de implementação, essa abordagem é simples, mas a divisão em timesteps pode ser de certa forma desafiadora para alguns modelos de simulação.

Nesse modelo, todas as entidades participantes na simulação estão no mesmo *timestep* em qualquer posição no decorrer da execução. Normalmente, a simulação é dividida em *timesteps* de tamanhos iguais e a simulação avança de um *timestep* para o próximo. No i-ésimo step, o algoritmo simula todos os eventos que estão dentro do mesmo intervalo $[(i-1)\Delta, i\Delta]$, sabendo que o Δ é um parâmetro de projeto [60]. Se o Δ é muito pequeno, a eficiência do método pode degenerar, pois a barreira de sincronização é desperdiçada em intervalos que podem conter nenhum evento. Por isso, o Δ deve ser grande o suficiente para que o processamento de um elemento típico tenha diversos eventos sendo processados em qualquer intervalo dado $[(i-1)\ \Delta, i\Delta]$. Por outro lado, caso o Δ seja grande demais, a simulação se torna com alto acoplamento, pois todos os eventos dentro do intervalo são simulados como se ocorressem de forma simultânea. A Figura 12 seguinte ilustra o avanço de uma simulação *timestep* convencional.

Figura 12. Progresso da simulação em simulações *time-stepped* tradicional [60]



4.2.2. Conservativo

O objetivo dessa abordagem é prevenir erros casuais, ou seja, antes de processar um evento com o *timestamp* "t", o PL tem que decidir se esse evento é "seguro" ou não. Ele é considerado "seguro" se, no futuro não existir eventos com *timestamp* menor do que "t". Se essa regra for seguida por todas as PLs, então a PADS irá obter resultados que são corretos sobre o ponto de vista de sincronização.

Para explicar de forma mais detalhada o mecanismo conservativo, assume-se que um simulador A muda o sinal no nó a no tempo *t_stamp*. Essa mudança irá levar a um novo valor de sinal conectado no nó b no simulador B. Nesse caso, o simulador A envia uma "mensagem de evento" para o simulador B. Ele informa ao simulador B que o sinal no nó b precisa ser atualizado no tempo *t_stamp*. Tal mensagem leva a adição de um evento (*t_stamp*, nó b, sinal a) na fila de eventos de B. *t_stamp* é chamado de *timestamp* do evento.

O tempo local virtual de um simulador é igual ao *timestamp* de um evento que acabou de ser executado. Se o simulador não processa eventos, o tempo virtual local pode ser atualizado para o tempo em que a próxima execução do evento é esperada.
O mecanismo conservativo de simulação garante que o simulador pode somente receber eventos com *timestamps* maiores ou iguais que o seu próprio tempo virtual [52][61]. Tal mecanismo é utilizado para a sincronização dos simuladores [62].

4.2.3. Otimista

Nesta abordagem, os PLs são livres para violar as restrições de causalidade (correção) e, por exemplo, podem processar os eventos na ordem em que recebem e estão disponíveis. Não existem tentativas *a priori* para predizer a chegada de novos

eventos com *timestamp* menores que podem causar violação de causalidade. Se, por um acaso, isto acontecer, o PL terá que fazer um processo de *rollback* (voltar atrás) para um estado interno anterior que é considerado correto e depois propagar o *rollback* para as outras LPs afetadas [63], [64]. Isso é feito através da exploração de mecanismos de gravação de informação de estados, chamados *checkpoints* (*state-saving*), obtidos durante a execução paralela ou distribuída.

Os *checkpoints* gravam os valores dos estados das variáveis anterior aos eventos de uma computação e faz o *restore* através da referência de tais valores salvos durante o mecanismo de *rollback*.

A abordagem otimista apresenta algumas características, como:

- A cópia dos estados salvos cria uma cópia inteira dos estados dos processos
- Incrementar os estados salvos a outros mantém um *log* de mudanças para cada variável de estado individual
- Estados salvos não frequentes salva periodicamente a cópia do estado inteiro;
- Computação reversa é uma nova técnica que realiza *rollbacks* através da execução de operações individuais inversas executadas em um evento de computação. Esta técnica está sendo explorada em simulação paralela de pequenas e grandes escalas.

4.3. Ferramentas de software

Diversas ferramentas vêm sendo desenvolvidas para facilitar a implementação das PADS. A maioria está de acordo com o padrão IEEE 1516 – *High Level Architecture* (HLA) detalhado posteriormente. Alguns exemplos podem ser citados: RTI NG Pro [65], *Georgia Tech FDK* [66], *MAK RTI* [67], *Pitch RTI* [68], ***CERTI Free HLA* [20]**, *OpenSkies Cybernet* [69], *Chronos* [70] e o projeto *Portico* [71].

4.4. Padrão IEEE 1516 – *High Level Architecture* (HLA)

O HLA é uma arquitetura de propósito geral definido sob liderança do *Defence Modelling and Simulation Office* (DMSO) para suportar o reuso e a interoperabilidade através de um vasto número de diferentes tipos de simuladores mantidos pelo Departamento de Defesa Americano, o DoD. É definida não pelo *software* em si, mas pelo conjunto de documentos que estão relacionados a essa arquitetura. Documentos

definidos por três padrões da IEEE: o primeiro trata do *framework* de forma geral e suas principais regras [72], o segundo diz respeito a especificação da *interface* entre os simuladores, o HLA [73], e o terceiro trata do modelo para especificação dos dados (OMT) transferidos entre os simuladores [74]. O HLA tem um serviço de gerenciamento de tempo para a sincronização entre modelos heterogêneos. Um dos objetivos do HLA é possibilitar a interoperação de modelos distintos, e poder reutilizá-los quando necessário, além de prover um ambiente de simulação distribuída para sistemas que necessitam de simulações em larga escala.

A ideia principal do HLA é separar as funcionalidades específicas de cada simulador através de uma infraestrutura de propósito geral. Cada simulador deve utilizar uma *Runtime Infraestructure* (RTI) [75] para se comunicar com o HLA e os demais simuladores. O RTI é responsável pela interface das estruturas específicas de cada simulador com a estrutura global do HLA. Cada simulador conectado a uma RTI forma uma chamada Federação.

O HLA é a cola que permite a combinação de simulações de computadores em grandes ambientes de simulação. O HLA permite que se crie uma fábrica básica de simulação a partir das peças. Por exemplo, podem-se combinar simuladores de controle de tráfego aéreo em diferentes regiões de um país com simuladores individuais de aviões. Essa arquitetura auxilia a combinação dessas simulações em uma simulação única e compreensível. Além disso, ela ajuda a estender qualquer tipo de simulação no futuro, através da adição de, por exemplo, outra máquina ou aeroporto adicional.

Como já citado, a arquitetura é dividida em três partes [76]:

- **Regras do HLA:** São princípios e convenções que devem ser seguidos para que as interações entre federados ocorra de forma apropriada durante a execução da cada federação. Tais regras são os princípios de projetos para a especificação da interface e do OMT. Elas também descrevem as responsabilidades dos federados e dos projetistas das federações.
- ***Objetc Model Template* (OMT):** Cada federação tem uma FOM (Federation Object Model). O OMT prescreve a estrutura permitida de cada FOM. O OMT é um meta-modelo de todas as FOMs.
- **Especificação da *Interface*:** Especifica a *interface* entre os federados e o RTI. O RTI é um *software* que permite que a federação seja executada de forma consistente. A interface entre o RTI e os federados é padronizada. Porém a implementação do RTI pode ser feita de diversas formas.

4.4.1. Visão geral

O HLA define uma arquitetura de *software* e não uma implementação, cujo principal objetivo é permitir a criação de aplicações de simulação através da combinação de outras simulações. Dessa forma, o HLA possibilita o desenvolvimento de simulação baseada em componentes, em que os componentes são os federados (não as entidades que serão simuladas).

Como evolução do HLA, existem dois esforços paralelos para aprimorar sua especificação. O primeiro é feito através da OMG (*Object Management Group*), que tem como propósito padronizar objetos de computação distribuídos. O segundo esforço é feito pelo *Institute of Electrical and Electronics Engineers* (IEEE). Nesse caso, o intuito está em definir melhor as regras, a especificação da interface e o OMT, ou seja, os três pilares do HLA [76].

O HLA é baseado em algumas premissas e suposições para justificar a sua importância [76]:

- Nenhuma simulação única e monolítica pode satisfazer as necessidades de todos os usuários. Os usuários se diferem um do outro devido aos seus interesses e requisitos.
- Os desenvolvedores de simulação apresentam conhecimentos distintos dos domínios que serão simulados. Além disso, nenhum grupo simples de desenvolvedores é especialista em todos os detalhes, inclusive em um único domínio.
- Ninguém pode antecipar toda a utilidade da simulação e nem todas as formas que a simulação pode ser utilmente combinada.

Tais observações fazem com que os projetistas do HLA tenham os seguintes objetivos [76]:

- Deve ser possível decompor um grande problema de simulação em várias partes menores. As partes menores de um problema apresentam maior facilidade de definição, criação da forma correta e verificação.
- Deve ser possível combinar os resultados de simulações menores para obter um único resultado global em um grande sistema de simulação.
- Deve ser possível combinar pequenas simulações para formar um grande sistema de simulação.
- A *interface* entre as simulações e a infraestrutura genérica deve isolar as

simulações de mudanças em termos de tecnologia que é utilizada para implementar a infraestrutura, além de isolar também a infraestrutura de mudanças ocorridas nas tecnologias das simulações.

Shaw e Garlan [77] definem que de forma abstrata, uma arquitetura de *software* envolve a descrição de elementos que irão compor os sistemas construídos, a interação entre tais elementos, os padrões que guiam sua composição e as restrições sobre tais padrões.

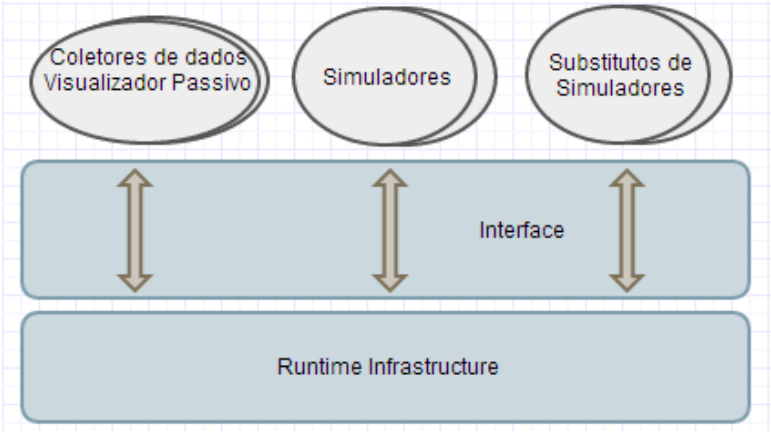
As três partes do HLA mapeiam os elementos, as interações e os padrões da seguinte forma [76]:

- **Elementos**: As regras e a especificação da *interface* definem os elementos da federação HLA que podem ser: os federados, o RTI, e modelo de objeto comum.
- **Interações**: As regras e a especificação de *interface* definem as interações entre os federados e o RTI, e entre federados com os federados (sempre mediados pelo RTI). Dada uma federação, os modelos de objetos das federações definem o tipo de dados transportados pelas interações entre os federados e o RTI. O OMT é um meta-modelo para todos os FOMs (*Federation Object Models*), ou seja, ele define a estrutura de cada FOM válido. Essa estrutura é assumida na especificação da *interface*.
- **Padrões**: Os padrões permitidos na composição do HLA são restringidos pelas regras e definidos na especificação da interface.

Os federados e o RTI são considerados *software*s. O FOM, por sua vez, é a descrição e o relacionamento entre os dados que os federados irão trocar na execução da federação. Ele expressa uma espécie de acordo entre os dados dos federados.

O relacionamento dos componentes de *software* pode ser visto na Figura 13. Os federados são exibidos na figura como: simuladores, substitutos para jogadores ao vivo, ou ferramentas para simulação distribuída. Esta última característica foi a utilizada neste trabalho. Um federado é definido como tendo um único ponto de ligação para o RTI. Ele pode consistir de diversos processos, talvez sendo executados em vários computadores distintos. Além disso, um federado também pode modelar uma entidade única, como um veículo, ou pode modelar várias entidades, como todos os veículos de uma cidade. Na perspectiva do HLA, um federado é definido pelo seu único ponto de ligação para com o RTI.

Figura 13. Componentes do HLA [elaborado pelo autor]



Como pode ser visto na figura anterior, um federado pode modelar várias entidades, ou pode ter um propósito diferente. Ele pode ser um coletor e/ou visualizador de dados, recebendo passivamente dados e gerando nenhum dado. Ele pode atuar como um substituto para participantes humanos em uma simulação. Nesse papel, o federado deve refletir um estado da grande simulação para o participante através de alguma interface e deve transmitir os controles de entrada ou decisões de um participante para o resto da federação.

O RTI usado pela federação pode ser implementado como vários processos ou como apenas um processo. Ele pode requerer que vários computadores sejam executados, mas conceitualmente existe apenas um único RTI. O RTI possibilita a execução de várias federações por vez.

A seguir serão detalhadas as três partes constituintes do HLA: *framework* de forma geral e suas principais regras [72], a especificação da *interface* entre os simuladores, o HLA [73], e o modelo para especificação dos dados (OMT) transferidos entre os simuladores.

4.4.2. *Framework* e Regras

Para entrar em mais detalhes na arquitetura do HLA, é interessante conceituar de forma mais detalhada alguns termos [76]:

Federation (Federação): É o conjunto das interfaces entre os federados, modelos de objetos comuns à federação e a infraestrutura (RTI) necessária para que todo o conjunto atenda a um objetivo específico. É um sistema de simulação combinado criado a partir de simuladores constituintes.

A federação contém os seguintes elementos:

- Contém o *software* principal chamado de *Runtime Infrastructure* (RTI)
- Um modelo de objeto comum para a troca de dados entre os federados em uma federação, é conhecido como *Federation Object Model* (FOM)
- Vários federados.

Federate (Federado): É cada membro de uma Federação. Ou seja, é toda aplicação que participa de uma Federação HLA, sejam elas aplicações de gerenciamento, coleta de dados, simulações ativas ou simplesmente simulações passivas. É um membro de uma federação, ou seja, é um ponto de ligação com o RTI, que pode ser desde uma plataforma, como um simulador da cabine de um piloto, ou pode representar uma agregação de simulação, como a simulação de um fluxo de tráfego aéreo inteiro de um país.

O HLA é fundamentalmente uma arquitetura que trabalha com simulações baseadas em componentes, em que os componentes são simulações individuais. A arquitetura também permite a criação de simulações que são distribuídas através de múltiplos computadores, porém nada na arquitetura assume ou requer que a implementação seja distribuída. O HLA abstrai grande parte da implementação distribuída.

Quando se diz que o HLA é uma arquitetura de componentes, o "componente" é o que chamamos de federados (*federate*). O federado é a simulação ou ferramenta que pode ser usada em outras federações (*federation*), além da própria federação na qual ele foi originalmente projetado. Como já citado, federações são constituídas por federados, logo o federado é uma unidade de construção, que pode ser um *software* reusável. Os federados normalmente são maiores do que componentes de *software* comuns. Eles são programas completos em execução, e não rotinas ou objetos em uma biblioteca.

O HLA permite a interação dos componentes da simulação (federados) em tempo de execução da federação, além de permitir também a extensão de alguns componentes no próprio projeto da federação. O projeto de um ambiente para a construção de um federado requer a adoção de políticas em vários casos, especialmente com relação ao HLA.

As regras do HLA é uma das três partes que constitui o padrão HLA. As regras expressam os objetivos de projetos e restrições das federações HLA. O benefício para o desenvolvedor em considerar as regras do HLA é que ele simplifica a forma com que o HLA deve ser utilizado. As primeiras cinco regras lidam com as federações e últimas

cinco com os federados.

4.4.2.1. Regras para as federações

1) Federações devem estar definidas no "HLA *Federation Object Model*" (FOM), seguindo o HLA OMT.

2) Em uma federação, todas as instâncias de objetos associados a uma simulação devem estar nos federados e não na "*Run Time Infrastructure*" (RTI). Isso é feito para garantir uma separação entre funcionalidades específicas da simulação e de propósito geral.

3) Todas as conexões de dados entre federados (FOM) ocorrem via RTI.

4) Durante a execução de uma federação, os federados interagem com o RTI seguindo a "HLA *Interface Specification*" (IS).

5) Durante a execução de uma federação, todos os atributos de instância devem ter posse definida por pelo menos um federado ao longo do tempo de execução.

4.4.2.2. Regras para os federados

1) Todos os federados devem estar definidos no "HLA *Simulation Object Model*" (SOM), documentados de acordo com o HLA OMT;

2) Todos os federados devem estar aptos a variar condições (atualizar, refletir, transmitir e receber) sobre os quais são atualizados os atributos de objetos;

3) Todos os Federados devem estar aptos a transferir ou aceitar a posse de atributos dinamicamente durante uma execução de federação, especificadas pelos seus SOMs;

4) Todos os federados devem estar aptos a variar as condições sobre os quais são providas as atualizações de atributos, especificadas por seus SOMs;

5) Todos os federados devem estar aptos a gerenciar o Tempo Local de maneira a permitir a troca de dados coordenados com outros membros da federação.

4.4.3. *Framework* e Regras para a Especificação da *Interface* das Federações

Apresenta a especificação das interfaces funcionais entre federados e a "*Run Time Infrastructure*" (RTI). A RTI provê serviços para federados de maneira análoga que um sistema operacional distribuído provê serviços para suas aplicações.

Essa estrutura de *software* (RTI) faz parte da aplicação final e serve para prover os serviços comuns de sistemas de simulação. Sua função mais importante é prover um canal comum de comunicação aos grupos lógicos de entidades federadas [76].

Em particular, o RTI é definido pela sua interface. Diferentes implementações do RTI podem satisfazer a interface como especificada. A especificação da interface define não somente a interface que o RTI apresenta para os federados, mas também a interface que os federados apresentam para o RTI (Figura 14).

A especificação da interface é abstrata. Os serviços, em ambas as direções, são definidos como chamadas de procedimentos que recebem e retornam parâmetros, através de pré e pós-condições nas chamadas, além das exceções. A definição dos serviços não faz referência alguma a qualquer linguagem de programação. Somente depois que os serviços são definidos, a especificação da interface define as APIs (*Application Programming Interfaces*) para várias linguagens de programação.

Um dos grandes objetivos do HLA é evitar definir o HLA em termos de tecnologias transitórias. Essa arquitetura mantém o ritmo com as novas tecnologias através da definição de APIs para novas linguagens e através da incorporação de novas tecnologias de rede na implementação do RTI.

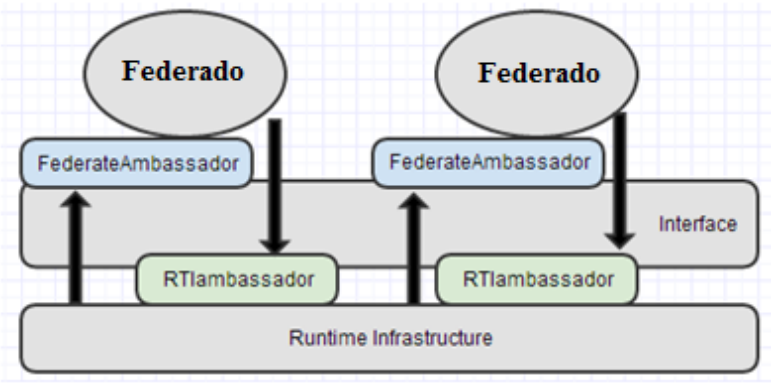
A ideia é fazer com que todo o comportamento específico de um dado modelo ou simulação esteja contido no próprio federado que o implementa, e a infraestrutura, no caso, contenha funções genéricas para a interoperabilidade da simulação. Então, a mesma implementação do RTI pode ser utilizada para suportar diferentes aplicações de simulação. No caso dos desenvolvedores dos federados, estes ficam livres de qualquer preocupação relacionada aos principais problemas de infraestrutura [76].

Como pode ser visto na Figura 14, existe sempre uma interface entre o RTI e os vários tipos de federados. É importante notar também que os federados não se comunicam diretamente com outros federados. Cada um é conectado ao RTI, e assim eles se comunicam entre si utilizando apenas os serviços que o RTI provê. Cada federado usa os serviços apropriados do RTI para cada propósito. O RTI, por sua vez, não distingue os federados. Finalmente, cada federado apresenta um único ponto de contato com o RTI. Esse ponto de contato é a definição do federado sobre a perspectiva do RTI. O federado pode conter múltiplos processadores, talvez até rodando em diversos computadores, mas ele mantém uma única conexão com o RTI, e o RTI é o responsável por se comunicar com os demais federados.

A Figura 14 ainda mostra que o RTI oferece uma *interface* chamada *RTIAmbassador* para cada federado. O federado invoca operações nessa *interface* para requisitar serviços ao RTI, por exemplo, requisição de atualização do valor de um atributo de um objeto. Isso recebe o nome de serviços de inicialização dos federados. Cada federado também apresenta uma *interface* chamada *FederateAmbassador* para o RTI. O RTI invoca operações nessa *interface* quando ele precisa chamar o federado, por exemplo, para repassa-lo o novo valor de um atributo. Isso é chamado de serviços de inicialização do RTI. Então, alguns serviços do RTI são definidos como parte da *interface* do *RTIAmbassador*, enquanto outros são definidos como parte da *interface* do *FederateAmbassador*.

Como parte da implementação do RTI, o desenvolvedor do federado recebe uma implementação do *RTIAmbassador* que é ligado aos processos do federado. O desenvolvedor também recebe uma classe abstrata do *FederateAmbassador*, e é responsável por implementar a classe concreta derivada dessa classe abstrata. No caso da API de Java, utilizada neste trabalho, tanto o *RTIAmbassador* quanto o *FederateAmbassador* são *interfaces* Java.

Figura 14. Arquitetura de comunicação HLA [elaborado pelo autor]



4.4.4. "HLA *Object Model Template*" (OMT)

Especifica o "HLA *Object Model Template*" (OMT) que contém a sintaxe e o formato em que as informações devem ser armazenadas em modelos de objetos HLA.

O OMT prescreve a estrutura de todas as FOMs. Cada federação tem uma FOM que apresenta o vocabulário específico desta federação. Em cada execução da federação, a FOM desta federação é utilizada. A FOM define os nomes dos dados e ocorrências sobre os quais os federados falam um com outro. Ela não descreve assuntos internos relacionados a um único federado, trata apenas de assuntos que são

compartilhados entre todos os federados. De certa forma, é um vocabulário de trocas de dados através do RTI para a execução de uma federação [76].
Os desenvolvedores de cada federação são livres para dotar os seus próprios modelos de objetos sem mudanças para o RTI ou mesmo para o padrão HLA.

Para entender melhor, é interessante conceituar de forma mais detalhadas alguns conceitos relacionados, como: OMT, FOM, SOM e MOM.

4.4.4.1. OMT (*Object Model Template*)

O HLA OMT provê um "*template*" (modelo) para a documentação relevante de HLA sobre objetos federados (classes de simulação), seus atributos e as interações que podem ocorrer entre os objetos de uma federação. Um "*framework*" padronizado ou "*template*" é essencial para a especificação de modelos de objetos HLA, para:

- Prover um mecanismo "entendido por todos" para especificar a troca de conteúdo público e gerenciar a coordenação entre os membros de uma federação;
- Representar o formato do contrato entre membros de uma federação (Federados) na forma de objetos e interações de modo a suportar a interoperabilidade entre múltiplas simulações;
- Prover um mecanismo comum e padronizado para descrever as capacidades dos membros da federação. O que fornece a base para a comparação entre diferentes simulações e federações;
- Facilitar o projeto e aplicação de um *kit* de ferramentas comuns para se desenvolver modelos de objeto HLA;
- Permitir que o modelo de objetos HLA possa ser usado para descrever um membro individual da federação (criado no "HLA *Simulation Object Model*", SOM), ou descrever um nome de um conjunto de múltiplas interações entre federados (criado no "*Federation Object Model*", FOM);

4.4.4.2. FOM (*Federation Object Model*)

O objetivo primordial do HLA FOM é prover uma especificação para a troca de todos os dados através de um formato comum e padronizado. Assim, os componentes de uma HLA FOM estabelecem um "modelo de contrato de informação" que é necessário

para garantir a interoperabilidade entre federados. O conteúdo de uma federação de objetos descreve:

- A enumeração de todas as classes que representam o mundo real para a planejada simulação.
- A descrição de todas as interações entre classes que representam as associações do mundo real.
- A especificação de atributos e parâmetros entre as classes;
- O nível de detalhamento em que essas classes descrevem o mundo real.
- Cada objeto encontrado em uma execução de uma federação é uma instância de uma classe de objeto predefinida no FOM. O FOM permite descrever as interações entre classes (ação explícita obtida por um objeto que pode opcionalmente ser direcionada a outro objeto) para cada modelo de objeto. Descrevem também os tipos de interações possíveis entre objetos, os atributos afetados e a especificação de interação entre parâmetros. Tudo isso é feito seguindo-se a notação HLA OMT.

4.4.4.3. SOM (*Simulation Object Models*)

O principal passo na formação de uma federação é o processo de determinação da composição de sistemas de simulação individuais de maneira a alcançar os objetivos.

Um HLA SOM é uma especificação (HLA OMT) das capacidades intrínsecas de uma simulação individual que podem ser oferecidas a outras potenciais federações HLA.

Diferentemente dos FOMs, os SOMs são caracterizados em termos de seus objetos, atributos e interações.

4.4.4.4. MOM (*Management Object Model*)

As federações HLA são tipicamente sistemas distribuídos. Os federados normalmente rodam em várias máquinas distintas. Logo, as federações são sujeitas aos problemas comuns dos sistemas distribuídos e por isso devem ser gerenciadas. O RTI, por ser um sistema sofisticado para a manutenção de uma visão compartilhada de um conjunto de entidades, este também pode ser utilizado para manter e gerenciar uma visão compartilhada da federação como um sistema distribuído.

Por exemplo, como cada federado se junta a uma federação, então o RTI cria uma instância da classe *Manager.Federate* e provê valores para os atributos da instância. O federado quer saber quantos outros federados se juntaram e podem subscrever para a classe *Manager.Federate* utilizando os mesmos mecanismos de qualquer outro dado FOM. Então, o sistema de gerenciamento pode ser realizado através do uso de federados projetados para esse fim.

Os principais componentes do OMT citados nos conceitos anteriores são: classes de objetos e classes de interação. Tais conceitos são detalhados a seguir.

- **Interação: Uma coleção de dados através do RTI**

Uma interação é uma coleção de dados enviados em um momento específico através do RTI para outros federados. Uma interação pode representar uma ocorrência ou um evento no modelo da simulação que pode ser interessante para mais de um federado.

O federado envia uma interação. Outro federado recebe uma interação. A interação deixa de existir depois que ela é recebida. Cada interação carrega com ela um conjunto de dados chamados de parâmetros.

A FOM define classes de interação. Quando um federado envia uma interação, essa é uma interação específica de uma classe. Interação entre as classes forma uma hierarquia simples de herança. Essa hierarquia tem uma raiz única chamada *InteractionRoot*. Cada classe de interação define os parâmetros que podem ser enviados com ela. Cada classe herda os parâmetros definidos por todas as suas superclasses.

- **Objetos: Entidades simuladas que tem estado persistente**

Objetos no RTI se referem a entidades simuladas que são interesse de mais de um federado e então são asseguradas pelo RTI; persiste ou sobrevive por algum intervalo de tempo da simulação.

O OMT define classes de objetos, nas quais cada uma tem seu nome. Cada uma define um conjunto de dados chamados atributos. Os federados criam instâncias de cada classe, cada uma possuindo uma identidade distinta na federação, além de possuir instâncias distintas de seus atributos. Os federados envolvem o estado de uma instância

de um objeto em tempo de simulação através do fornecimento de novos valores para os seus atributos.

Os federados conversam com o RTI em termos de interações e objetos. Como os federados se comunicam com cada um através do RTI, então eles conversam em termos de interações e objetos. Cada federado deve fazer alguma tradução de sua noção interna de entidades simuladas para os objetos HLA especificados de acordo com a FOM. Se o federado foi escrito com a intenção de tendência HLA, então a tradução pode ser direta; já no caso em que o federado é adaptado ao HLA, a tradução pode ser mais complicada. O FOM representa a parte comum e aceita dos vocabulários entre os membros e a federação.

Esse modelo também possibilita hierarquia, e a herança de todos os atributos das superclasses.

Existem dois tipos de atributos, os de classe de objeto, considerado classe, e o outro é um atributo específico de uma instância de um objeto, chamados de atributos de instância.

4.4.5. Características do HLA

A seguir serão apresentadas algumas características do padrão HLA [76].

4.4.5.1. O HLA é uma arquitetura de camadas

De acordo com Shaw e Garlan [77], o HLA é uma arquitetura de camadas. Um sistema de camadas é organizado hierarquicamente, em que cada camada provê serviços para a camada inferior e serve como cliente para a camada posterior. Em alguns sistemas de camada, as camadas internas são escondidas de todas as demais camadas, exceto a camada adjacente mais externa, exceto para funcionalidades específicas cuidadosamente selecionadas para se expor.

Na perspectiva dos federados, o RTI aparece como uma camada posterior a todas as demais camadas que encapsula completamente as funções do RTI. No caso, em federações distribuídas, o RTI contem funções de rede que são necessárias para possibilitar a distribuição. As funções de distribuição são escondidas dos federados por trás da interface do RTI.

Essa separação das funções do RTI com relação aos federados traz dois importantes pontos. O primeiro, ela remove dos federados o que é genérico na

interoperabilidade da simulação. O código dos federados não precisa duplicar os serviços necessários para a interoperabilidade. O segundo ponto, a camada isola os federados de mudanças em tecnologias que podem refletir no RTI. Se o RTI precisa ser modificado para acomodar um novo tipo de rede, os federados não são afetados.

4.4.5.2. O HLA é uma arquitetura de dados abstratos

Shaw e Garlan [77] descreve uma arquitetura de dados abstratos, como: "No estilo baseado em abstração de dados e utilizando o paradigma orientado a objetos, a representação dos dados e a associação de suas operações primitivas são encapsuladas em um tipo de dado abstrato ou objeto. Os componentes desse estilo são os objetos, ou se preferir, instâncias de tipos de dados abstratos."

O princípio das camadas no HLA, na verdade, trabalha em duas direções: do federado em direção ao RTI, e do RTI em direção ao federado. Isso constitui a abstração dos dados. O RTI apresenta uma interface para o federado, em que por trás disso todos os estados estão escondidos. Cada federado também apresenta uma interface para o RTI, que por trás seus estados estão escondidos. A partir da perspectiva do RTI, existem múltiplos federados, cada um com a mesma interface, mas apresentando identidades distintas. Essa é a essência da abstração de dados e da organização orientada a objetos. O grande benefício de trabalhar dessa forma é que a mesma implementação do RTI é utilizada para vários federados. Se os federados mudarem o RTI não é afetado.

Porém como desvantagem de sistemas como esse é que os objetos devem saber sobre cada outro objeto, no intuito de invocar operações em cada objeto explicitamente. Isso é verdade para o HLA, já que é orientado a objetos de forma geral; o federado deve encontrar sua implementação do RTI para se juntar a federação, e o RTI deve manter sempre a referência de algo relacionado aos federados. Isso de certa forma é uma desvantagem, mas não uma grande limitação.

Pode-se apontar aqui que de acordo com as regras do HLA, um federado nunca interage diretamente com outro, mas sempre através da federação do RTI. Os federados não precisam manter nenhum tipo de referência para outros federados, além de não ser nem necessário saber da existência de outros federados.

4.4.5.3. O HLA é uma arquitetura baseada em eventos (invocação implícita, integração reativa, *broadcast* seletivo)

A ideia por detrás da invocação implícita é que ao invés de invocar um procedimento diretamente, o componente pode anunciar (via *broadcast*) um ou mais eventos. Outros componentes no mesmo sistema podem realizar um registro com o interesse em algum evento específico através da associação de um procedimento a esse evento. Quando o evento é anunciado, o sistema por si só invoca todos os procedimentos que foram registrados para o evento em questão. Então, o anúncio do evento de forma implícita causa a invocação de procedimentos em outros módulos.

Por padrão, um federado invoca um serviço ao RTI que irá fazer com que o RTI invoque serviços em outros federados. É papel do RTI decidir que federado será chamado.

De acordo com Shaw e Garlan [77], o principal benefício desse estilo de arquitetura é possibilitar o reuso e a facilidade com que cada sistema pode ser envolvido. Isso é precisamente o propósito desse estilo do HLA: o federado escrito para esperar um FOM específico pode ser combinado com outros federados com o mesmo FOM. Como desvantagem, pode-se dizer que na invocação implícita, os componentes renunciam o controle sobre a computação do sistema. Em particular, eles não podem fazer suposições sobre a ordem em que os eventos acontecem. Quem se responsabiliza disso são os serviços do RTI.

4.4.6. Serviços do HLA

Para finalizar este Capítulo que trata de simulação distribuída, especialmente o padrão HLA, é interessante apresentar algumas funções da *interface* do HLA, ou seja, os serviços que o RTI oferece para os federados e vice-versa. Os serviços do HLA estão divididos em seis grupos que são definidos por similaridade de interesses [76].

O objetivo do projeto dos serviços do HLA é fazer com que o desenvolvedor dos federados não necessite utilizar os serviços de todos os grupos, ou seja, ele pode escolher de acordo com a sua necessidade. Os serviços em um grupo podem ser usados sem referência alguma a outro serviço, no caso: gerenciamento de tempo, de propriedade, e distribuição de dados. Os únicos serviços que sempre são requeridos para a transferência dos dados são: gerenciamento de declaração, de objetos e da federação.

4.4.6.1. Gerenciamento da Federação

O serviço de gerenciamento da federação gerencia a federação de duas formas:

- Através da definição da federação a ser executada em termos da existência de membros e de como fazer parte de uma federação.
- Através do fornecimento de várias operações relacionada a federação.

Para definir uma federação, existem alguns serviços para iniciar a execução da mesma e para permitir que um federado se junte a uma federação em execução ou mesmo saia dela. Todos os federados devem se juntar a uma federação em execução, logo nenhum federado pode ignorar completamente esse grupo de serviço.

As operações de uma federação incluem a coordenação do salvamento da federação (*checkpoints*) e possibilidade de realizar *restores*. Existem também serviços que permitem que a federação defina e encontre o ponto de sincronização da federação.

4.4.6.2. Gerenciamento de declaração

O HLA é caracterizado pelo estilo invocação implícita de transferência de dados. Os federados não enviam dados diretamente para outros federados, eles o deixam disponíveis para a federação, e o RTI assegura que tais dados serão entregues somente para as partes interessadas. Esse serviço é a forma como os federados declaram o seu interesse por produzir (*publish*) e por consumir (*subscribe to*) dados. O RTI também utiliza tais declarações para roteamento de dados, transformação de dados e gerenciamento de interesses.

A respeito do roteamento, o RTI utiliza a assinatura para decidir qual federado deve ser informado sobre criação ou atualização das entidades.

Declarações são utilizadas para transformar os dados recebidos pelos federados. Os dados recebidos sofrem uma seleção e avaliação de acordo com a assinatura dos federados antes de serem entregues. Isso é um dos vários mecanismos para proteger os federados de extensões no FOM e encorajar o reuso dos federados.

Finalmente, o RTI utiliza declarações para indicar interesse para publicar os federados. O RTI pode dizer ao federado se algum outro federado assinou os dados que ele pretende produzir. Dessa forma é possível cessar a produção quando nenhum outro federado precisa de tal informação.

4.4.6.3. Gerenciamento de Objetos

É um serviço que lida com os dados correntes que estão sendo enviados e recebidos. O federado utiliza os serviços desse grupo para enviar e receber interações.

Esses serviços também são utilizados para registrar novas instâncias de classes de objetos e atualizar seus atributos. Outros federados irão ter serviços invocados desse grupo para receber as interações, descobrir novas instâncias, e receber atualizações de instâncias de atributos.

Outros serviços desse grupo são utilizados para controlar como os dados são transportados, para solicitar atualização de valores dos atributos e para informar ao federado se ele deve ficar esperando dados ou não.

4.4.6.4. Gestão de propriedade

Em termos de HLA, simular uma entidade significa fornecer valores para os atributos de instâncias da mesma. O serviço de gerenciamento de propriedade no RTI implementa a noção de responsabilidade do HLA para simular entidades. Esses serviços também permitem que a responsabilidade seja dividida ou transferida entre os federados.

Como citado anteriormente, as regras 5 e 8 do HLA requerem que o federado seja dono do atributo de instância antes que ele possa atualizar o seu valor. O RTI assegura que no máximo um federado por tempo seja o dono para um dado atributo de instância. O federado responsável é responsável pela atualização dos valores. A responsabilidade por simular uma entidade pode ser compartilhada entre os federados de duas formas.

A primeira, a modelagem completa de uma entidade deve ser dividida entre os federados. Se a entidade é representada por uma instância com vários atributos, então federados distintos podem ser donos de vários atributos dessa instância e então ser responsável pela atualização dos atributos no qual ele ficou responsável. Os serviços de gerência de propriedade possibilita a aquisição da propriedade para permitir todo esse processo.

A segunda forma acontece quando a modelagem de entidade passa de um federado para outro durante o curso de uma federação em execução. A propriedade de um atributo de instância pode ser transferida de um federado para outro. A transferência da propriedade pode ser iniciada pelo dono atual ou o dono futuro.

Esse serviço pode ser ignorado se a federação não necessitá-lo.

4.4.6.5. Gerenciamento do tempo

Com os federados executando em suas próprias *threads* de controle, a ordenação correta de eventos entre os federados é um problema significativo que precisa ser resolvido. No HLA, a ordenação de eventos é expressa em "tempo lógico". Tempo lógico é uma noção abstrata. Não é necessariamente ligado a qualquer representação ou unidade de tempo. O serviço de gerenciamento de tempo do RTI faz duas tarefas:

- Permite que cada federado avance no seu tempo lógico em coordenação com outros federados.
- Controla a entrega de eventos de "*timestamp*" para que o federado não precise receber nunca eventos de outros federados no tempo passado, ou seja, eventos com tempos lógicos menores que o seu tempo lógico.

O RTI permite que o federado escolha o grau de sua participação no gerenciamento de tempo. O federado pode ser *time-constrained* (tempo-restrito), no qual o avanço do tempo lógico é restringido por outros federados. Também pode ser *time-regulating* (tempo-regulado), em que o avanço do seu tempo lógico regula outros federados. E ainda pode ser *time-constrained* e *time-regulating*, ou um de cada vez, ou nenhum. Os federados farão escolhas diferentes dependendo de seus propósitos e dos requisitos da federação.

4.4.6.6. Gerenciamento de distribuição dos dados

Os serviços desse grupo controlam a relação produtor-consumidor entre os federados. Enquanto o grupo de serviços de gerenciamento de declaração gerencia as relações em termos de interações e classes de objetos, o grupo de serviços de gerenciamento de distribuição de dados gerencia em termos de instâncias de objetos e espaços de roteamento abstrato.

Este grupo de serviço provê ferramentas poderosas para refinar as relações entre produtor e consumidor.

4.4.7. Implementações do RTI

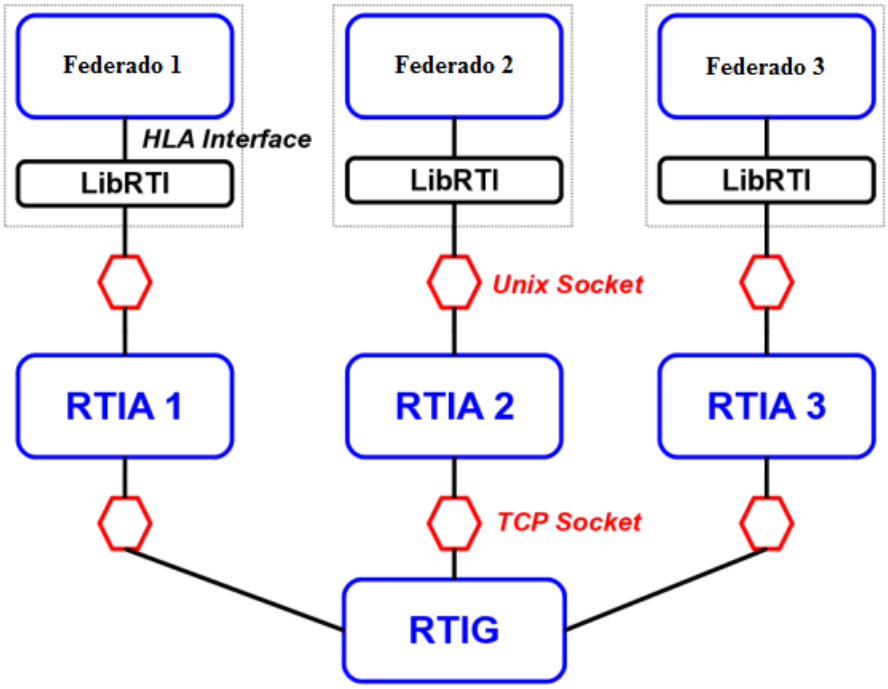
O RTI, como já explicado anteriormente, é um *middleware* que é necessário quando se implementa o HLA, sendo o componente fundamental desta arquitetura. Ele provê um conjunto de serviços de *software* que são necessários para dar suporte aos federados, serviços como: coordenação das operações e transferências de dados durante a execução da federação. Pode-se dizer que é a implementação da especificação da interface do HLA, mas não é parte da especificação propriamente dita.

Algumas implementações podem ser citadas, dentre as quais uma foi escolhida para ser utilizada neste trabalho, o CERTI.

- *MAK High Perfomance RTI*
- *SimWare RTI*
- *Open HLA*
- *Openskies RTI*
- *Portico Project*
- *Certi RTI*

O Certi é um RTI HLA *opensource* (GPL) [20], que implementa a especificação HLA 1.3 (C++ e Java) e desenvolvido pela sua comunidade *opensource* e mantido pela ONERA. É um RTI reconhecido pela sua arquitetura original de comunicação de processos. Em sua arquitetura está incluso um componente local RTI, chamado RTIA para cada federado e um único componente global, chamado RTIG, assim como uma biblioteca libRTI ligado a cada federado (Figura 15). Cada processo do federado interage localmente com o processo *RTIAmbassador* (RTIA) através de um *socket* Unix ou TCP *socket* para *windows*. O processo RTIA troca mensagens na rede através do processo RTIG, utilizando o *socket* TCP e também o UDP, no intuito de executar os diversos algoritmos distribuídos associados com os serviços do RTI. O RTIG é um gateway central responsável por entregar e realizar o *broadcast* de mensagens para todos os RTIA [78].

Figura 15. Arquitetura do Certi [78]



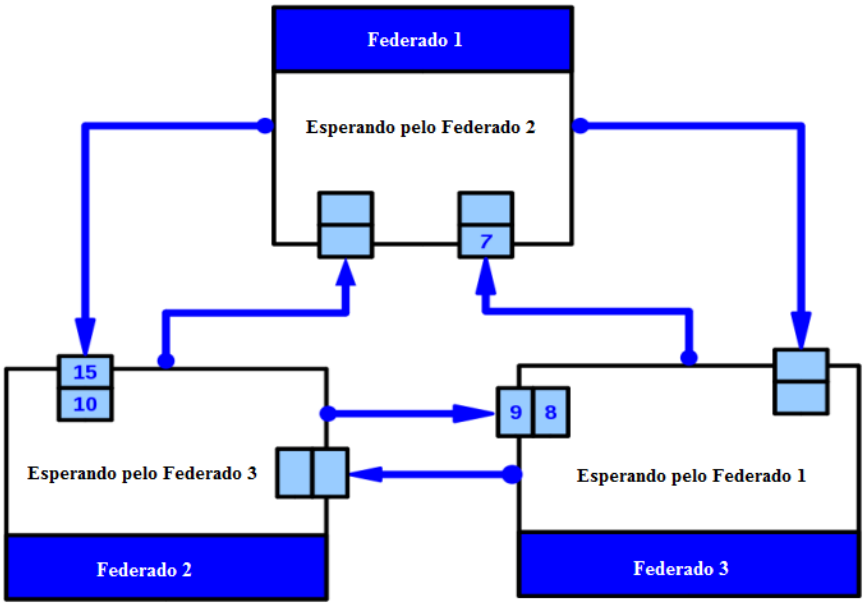
O grande benefício de entender bem a implementação desse RTI é que é possível aprimorar e modificar o seu código fonte no intuito de melhorar o desempenho das simulações.

4.5. Lookahead, LBTS, GALT

Os mecanismos de gerenciamento, herdados do padrão ALSP [79] e provido pelo *middleware* HLA, são um dos maiores benefícios do padrão da simulação [80]. Esses serviços permitem um tempo global consistente durante toda a simulação, utilizando diferentes métodos e algoritmos. Especificamente, cada mensagem da simulação é assinada por um *timestamp*, e o RTI assegura que a mensagem seja entregue para cada federado na ordem do *timestamp*, e que nenhuma mensagem seja entregue ao federado no passado. A principal operação requerida para implementar os serviços de gerenciamento é a determinação do GALT (*Greatest Available Logical Time*), também chamado de LBTS (*Lower Bound on TimeStamp*) para o padrão HLA 1.3 [81], de cada federado. O padrão HLA não provê ou alerta uma implementação específica para o serviço de gerenciamento de tempo oferecido pelo RTI. Existem duas abordagens já citadas anteriormente neste Capítulo, que assegura a restrição de causalidade, a abordagem otimista e a conservativa.

A abordagem conservativa de sincronização foi utilizada neste trabalho para tentar obter o melhor controle de consistência dos dados e tempo. Essa abordagem apresenta certas limitações para aplicações em tempo real de alto desempenho. Esse serviço de gerenciamento de tempo é baseado no conhecido "*NULL Message Algorithm*" (NMA) de Chandy e Misra [82]. Esse é o principal algoritmo implementado pelo CERTI, implementação utilizada neste trabalho. Ele é usado para evitar *deadlock* em federações conservativas como pode ser ilustrado na Figura 16, extraída do trabalho de Richard Fujimoto [78].

Figura 16. Exemplo de *deadlock* [78]



Essa abordagem é baseada no contrato de cada federado chamado de *lookahead*. Cada federado tem uma regra em que não pode enviar mensagens de simulação com o *timestamp* menor que o seu próprio time local mais o *lookahead*. Se o contrato for respeitado, então possibilita que mensagens adicionais sejam trocadas, as chamadas *NULL messages* (mensagens contendo apenas o *timestamp*), indicando que o LBTS (*Lower Bound on the Time Stamp*), algo equivalente ao GALT, de mensagens futuras possam ser enviadas.

A principal deficiência dessa abordagem para simulações em tempo real e de alto desempenho é o *overhead* que a comunicação de várias mensagens nulas entre os simuladores implica. Além disso, se o parâmetro de *lookahead* não é bem escolhido, a simulação está sujeita ao problema conhecido por "*lookeadhead time creep*", ou em português "rastejar o tempo de *lookeahead*", ou seja, o número de mensagens nulas (*NULL messages*) enviadas pode ser inaceitável e isso pode limitar o desempenho da

simulação. No Capítulo 8 serão apresentados diversos casos, nos quais existem variações do *lookahead*, e assim será analisado o possível *lookahead* ideal para o cenário proposto [78].

4.6. Lei de Amdahl

A famosa lei de Amdahl é bastante conhecida pelos cientistas da computação. Entende-se por *speedup* como sendo o tempo de execução original dividido pelo tempo de execução aumentado. A versão moderna da lei de Amdahl cita que se alguém consegue aumentar a fração f de uma computação através do *speedup S*, então o *speedup* geral aumentado é dado pela equação (1) [54]:

$$Speedup_{aumentado}\ (f, S) = \frac{1}{(1-f)+\frac{f}{S}} \quad (1)$$

A lei de Amdahl se aplica amplamente e tem alguns corolários importantes, como:

- Ataque ao caso comum: Se f é pequeno, então as otimizações realizadas terão poucos efeitos.
- Mas os aspectos ignorados também podem limitar o *speedup*: Como *S* tende ao infinito, o *speedup* tende a 1 / (1 – f).

Algumas décadas atrás, originalmente Amdahl definia sua lei para casos especiais de uso de *n* processadores (cores atualmente) em computação paralela, quando ele argumentava para a "Validação da abordagem de processador único para atingir capacidades de computação em larga escala [83]". Ele assumia simplesmente que a fração *f* de um tempo de execução de um programa era paralelizada indefinidamente sem nenhum *overhead* sequer, enquanto que a fração restante, 1 - *f*, era totalmente sequencial. Ele notou que o *speedup* nos n processadores é governado pela seguinte equação do *speedup* paralelo (2) [54]:

$$Speedup_{paralelo}\ (f, n) = \frac{1}{(1-f)+\frac{f}{n}} \quad (2)$$

Finalmente, ele argumentou que os valores típicos de 1-f são largos o suficiente para os processadores únicos [83].

A equação do *Speedup* paralelo, foi a utilizada para obter os resultados deste trabalho com relação ao *speedup* de Amdahl. Para obter o *speedup* da abordagem deste trabalho, utilizou-se a conceito geral de *speedup*, em que o tempo de execução original é

dividido pelo tempo de execução aumentado, ou seja, o pior tempo é dividido pelos respectivos melhores tempos.

Os resultados do uso da Lei de Amdahl são apresentados no Capítulo 6.

Capítulo

5

Desenvolvimento

Modelar sistemas embarcados que apresentam MoCs heterogêneos, numa semântica única e clara em um mesmo ambiente é um desafio, que o uso do Ptolemy, devido a suas características inerentes, ajuda a solucionar. Desafio ainda maior, buscado nesse projeto, é a modelagem e a simulação dessa natureza de sistema em larga escala, composta por dezenas de milhares de dispositivos, como é o caso de Redes de Sensores sem Fio e sistemas embarcados para TV Digital Interativa. Para tal, foram necessários uma alta flexibilidade na composição de diferentes MoCs, um alto nível de paralelismo e grande poder computacional. Dessa forma, foram integrados o Ptolemy II e o *High Level Architecture* (HLA) para simulação distribuída de grandes modelos em diversas máquinas em rede, no intuito de apresentar uma prova de conceito e aprimorar o desempenho de grandes cenários de simulação.

As principais perguntas de pesquisa que este trabalho busca resolver são: 1. “É possível integrar o HLA ao Ptolemy, e ainda essa integração viabiliza a simulação de grandes sistemas heterogêneos em larga escala?”, ou ainda, 2. “O uso do HLA integrado ao Ptolemy apresenta vantagens, em termos de desempenho da aplicação, comparado ao uso do Ptolemy de forma isolada?”.

Este capítulo busca primeiramente mostrar a viabilidade da integração entre o Ptolemy II com o HLA, criando assim um ambiente que permita a fácil configuração de todo o ambiente e a execução de modelos heterogêneos de computação em larga escala com alto desempenho, respondendo assim, a primeira pergunta de pesquisa.

Além da integração, este trabalho visa avaliar o desempenho de tal integração em cenários contextualizados elaborados para este fim. Foram realizados experimentos em cenários específicos que em sua implementação utilizam apenas o Ptolemy, e em outros cenários que contém o Ptolemy integrado ao HLA. Dessa forma, foi feito um comparativo apresentando as duas soluções, especialmente a solução que existe a

integração com HLA, solução esta que foi detalhada em termos de alterações em variáveis que afetam diretamente o desempenho da simulação, variáveis como: *lookahead*, quantidade de sensores e quantidade de máquinas. Assim, a segunda pergunta de pesquisa também será respondida.

O capítulo 6, apresentado posteriormente, também ajuda a responder as duas principais perguntas de pesquisa expostas anteriormente.

5.1. Princípio da integração Ptolemy com HLA

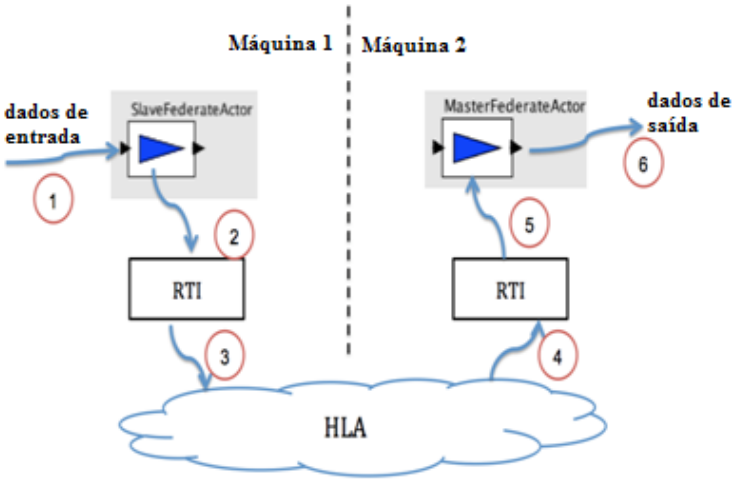
Experimentos iniciais realizados nos laboratórios demonstraram com sucesso a integração do Ptolemy II com o HLA [23]. A metodologia para a realização de tais experimentos será explicada em detalhes nesta mesma seção, e os resultados serão explicados na seção 6.2 no próximo Capítulo. Depois de chegar à conclusão que existe viabilidade técnica para realizar tal integração, é possível aprimorar os experimentos e aplicá-los a uma simulação em larga escala, composta por vários dispositivos, como por exemplo, em redes de sensores sem fio.

A Figura 17 apresenta o princípio dessa integração. Vale salientar que na literatura até este momento, foi encontrado apenas um trabalho publicado e somente em meados do ano de 2013 que aborda a integração entre o Ptolemy II e o HLA. Para tal, esta dissertação projetou a criação de dois atores, além de um diretor especial para coordenar a execução do tempo de simulação do Ptolemy, com o tempo global do HLA. Os atores foram nomeados de *SlaveFederateActor* (escravo) e *MasterFederateActor* (mestre). O mestre é responsável por iniciar a simulação, aguardar até que todos os escravos iniciem e só assim, a simulação é liberada.

Essa mesma figura ilustra também os passos necessários para um dado ser enviado do escravo, situado em uma máquina da rede, para o mestre, situado em outra máquina. Esses dois atores funcionam como uma espécie de interface entre o Ptolemy e o RTI. O primeiro passo é o escravo receber um dado de entrada, que pode ser de qualquer tipo. Como se sabe, de acordo com as regras do HLA (seção 4.4.2), não existe comunicação direta entre dois federados. Logo, esse dado é repassado pelo escravo para o RTI do lado do escravo. Nesse momento ocorre a sincronização conservativa tanto dos dados a serem enviados quanto do tempo para que não exista perda de dados. Dessa forma, verifica-se a necessidade de avançar o tempo global de simulação (tempo global do HLA). Em seguida, o dado é passado para HLA, que o repassa para o RTI destino

(ligado ao mestre), através da infraestrutura de rede. Ao chegar ao RTI destino, o dado é repassado para o ator mestre que o replica para sua porta de saída. O processo semelhante é realizado caso o ator mestre necessite enviar algum dado para um ator escravo.

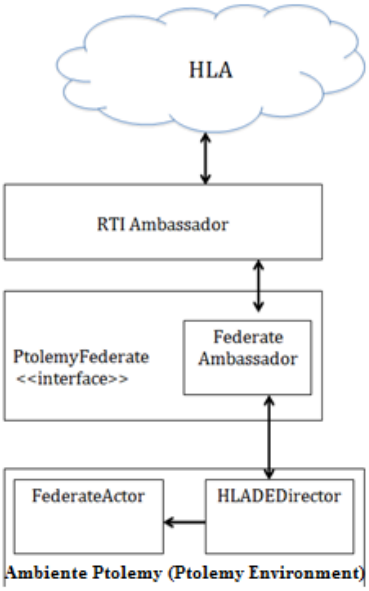
Figura 17. Princípio geral para integração do Ptolemy II ao HLA [elaborado pelo autor]



Para tal, foi utilizado o protocolo padrão do HLA, o *Publish and Subscribe*, em que o ator mestre se inscreve para receber um determinado dado, e o escravo é configurado para publicar esse mesmo dado. Assim, o escravo só envia o que deseja, e o mestre só recebe os dados para o qual foi inscrito.

Para entrar mais em detalhes na implementação da simulação, pode-se observar inicialmente a Figura 18, que mostra a arquitetura da aplicação em alto nível. Quando se trabalha com HLA, é necessário que cada simulador, representado pelo federado, utilize uma implementação da *Runtime Infraestructure* (RTI) em comum, como o CERTI RTI, e para que o acesso a essa infraestrutura seja possível, deve existir para cada federado uma interface para esse fim, no caso, o *RTI Ambassador*, que provê os serviços do RTI e possibilita a comunicação com o mesmo, que por sua vez está de acordo com a especificação HLA (nuvem). Cada federado invoca os serviços apropriados do RTI para cada propósito através do *RTI Ambassador*. Um serviço que pode ser citado é o de requisição de atualização do valor de um atributo de um objeto. O RTI, por sua vez, não distingue os federados existentes na simulação. A gerência do tempo global da simulação é feita de um modo geral pelo RTI (no caso da Figura 17, pelo HLA), e é ele também o responsável por se comunicar com os demais simuladores (vide Figura 13). O RTI (*RTI Ambassador*) é o responsável pela interface das estruturas específicas de cada simulador para a estrutura global do HLA.

Figura 18. Arquitetura da Implementação [elaborado pelo autor]



Além do *RTI Ambassador*, a Figura 18 mostra também o *Federate Ambassador*. Cada federado também apresenta uma interface chamada *Federate Ambassador* para o RTI acessar. O RTI invoca operações nessa interface quando ele precisa chamar o federado, por exemplo, para repassá-lo o novo valor de um atributo. Em resumo, alguns serviços do RTI são definidos como parte da interface do *RTI Ambassador* (serviços mais gerais do HLA), enquanto outros são definidos como parte da interface do *Federate Ambassador* (serviços mais específicos de cada federação).

Neste trabalho, como foi utilizada a API de Java, tanto o *RTIAmbassador* quanto o *FederateAmbassador* são *interfaces* Java. Outro conceito importante quando se lida com HLA é o de Federação, que é o conjunto de todos os federados conectados a uma RTI.

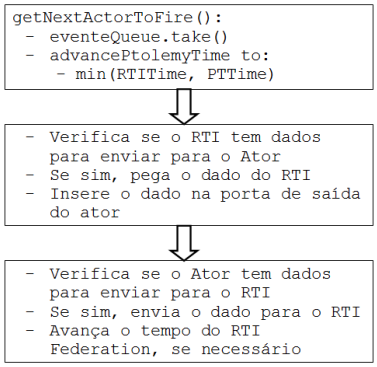
Ainda analisando a Figura 18, no ambiente do Ptolemy propriamente dito (Ptolemy *Environment*), há o *HLADEDirector*, diretor criado baseado em *Discret Event*. Tal diretor é responsável por coordenar a ordem e a execução dos atores dentro do domínio, verificando quando deve enviar um dado do RTI para o ator e também do ator para o RTI. A Figura 19 exemplifica esta coordenação de envio e recebimento de dados.

No primeiro momento retira-se da fila o próximo ator a ser executado (*getNextActorToFire()*). Nota-se que o avanço do tempo do Ptolemy é sempre o mínimo entre o tempo do RTI, e o tempo do próprio Ptolemy, garantindo assim que o tempo do Ptolemy de uma máquina não avance mais que o outro no ambiente de simulação, garantindo o sincronismo entre todos.

No segundo momento é verificado se algum dado chegou através do RTI para algum ator dentro do modelo Ptolemy. Em caso afirmativo, o dado é inserido na porta de saída do *FederateActor* em questão, que o replica para os atores conectados a ele.

Como último passo, é verificado se algum ator do Ptolemy possui dado para ser enviado para outra máquina através do RTI. Nesse caso, o dado é enviado para o RTI e o tempo do RTI é avançado e comunicado a todos os federados presentes na Federação.

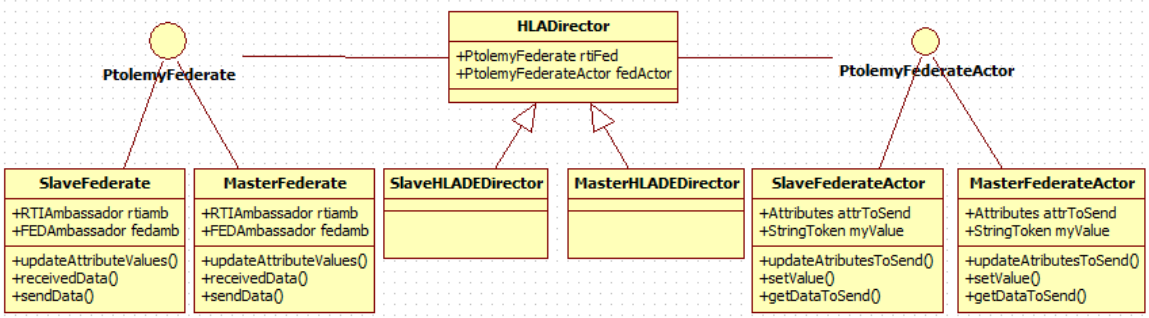
Figura 19. Funcionamento [elaborado pelo autor]



5.1.1. Arquitetura em detalhes

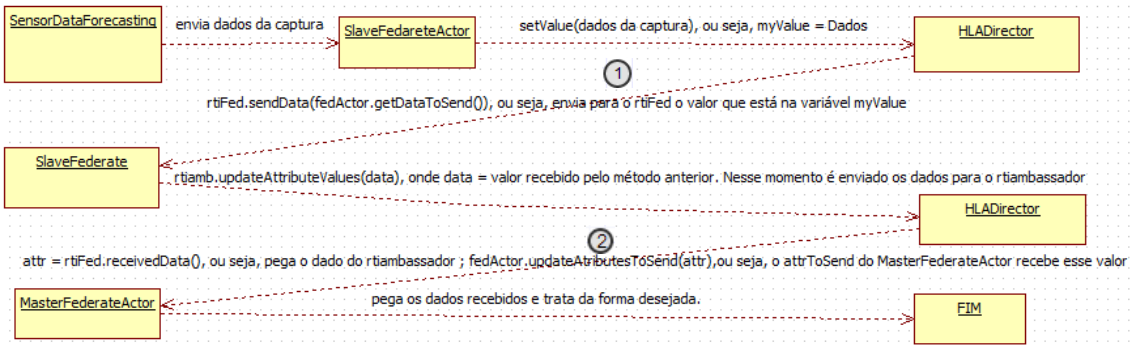
Além dos atores criados no próprio ambiente do Ptolemy (*SlaveFederateActor* e *MasterFederateActor*) para possibilitar a integração com HLA, foi necessário criar diversas outras classes para auxiliar e viabilizar o acesso às interfaces HLA (*FederateAmbassador* e *RTIAmbassador*) e permitir assim a transferência dos dados de forma sincronizada. As principais classes criadas, mas que não são atores Ptolemy, são as classes concretas: *SlaveFederate*, *MasterFederate*. E as interfaces: *PtolemyFederate* e *PtolemyFederateActor*. O diagrama de classe a seguir mostra um pouco de como essas classes estão relacionadas (Figura 20). Percebe-se que a principal classe é o *HLADirector* que implementa duas principais interfaces (*PtolemyFederate* e *PtolemyFederateActor*), e é nessa classe que existe o envio dos dados para o RTI, e o recebimento dos dados através do RTI, além da sincronização do avanço do tempo global da simulação.

Figura 20. Diagrama das principais classes envolvidas na comunicação entre os atores [elaborado pelo autor]



Os atores de fato que fazem parte do modelo gráfico do Ptolemy implementam a interface *PtolemyFederateActor*, e as classes auxiliares implementam a interface *PtolemyFederate*. Essas últimas classes são as que acessam de fato o *RTIAmbassador* e o *FederateAmbassador*, que são as classes responsáveis por fornecer os principais serviços da implementação CERTI do HLA. A figura a seguir apresenta o fluxo interno de como essas classes se relacionam, no momento em que o Escravo envia dados para o Mestre, e o Mestre recebe tais dados (Figura 21). Percebe-se também através desse diagrama que o *HLADirector* é acionado duas vezes durante esse fluxo de troca de mensagens, pois é essa classe a responsável por fazer a intermediação do envio dos dados para o RTI provenientes do *SlaveFederateActor* e também por capturar os dados do RTI e enviar para o *MasterFederateActor*. E como foi citado anteriormente o *SlaveFederate* é o responsável por se comunicar de fato com *RTIAmbassador* ou *FederateAmbassador* para usar os serviços do RTI.

Figura 21. Fluxo da troca de mensagens do escravo para o mestre [elaborado pelo autor]



Como a classe *HLADirector* é a responsável por fazer a intermediação do envio e recebimento dos dados, então é importante explicar com mais detalhes a implementação dessa classe. As figuras a seguir (Figura 22 e Figura 23) mostram o mesmo fluxo citado da figura anterior com relação ao *HLADirector*, porém com trechos de códigos. A Figura 22, por exemplo, mostra como o HLA verifica se o ator tem algum dado a ser enviado para o RTI (fluxo 1 na Figura 21). No algoritmo da Figura 22, essa parte é exibida com uma marcação de uma seta na linha em questão. Isso significa que, se o federado (*masterFederateActor* ou *slaveFederateActor*) possuir dados para enviar, esses serão capturados pelo método *getDataToSend()* e enviados através do método *sendData()*. O restante do algoritmo mostra basicamente como é feita a lógica de avanço do tempo global do HLA utilizando o algoritmo conservativo de sincronização (*nextHLAEventTime()* e *getRTINextTme()*). Nesse caso, o menor tempo sempre é enviado para o próximo ator da fila ser executado (*fireAt(fedActor, time)*).

Figura 22. Trecho de código que verifica se o ator tem dados para enviar para o RTI

```
/*******
HLA checa os dados que o ator quer enviar para o RTI
 *********/
PtolemyFederateActor fedActor = (PtolemyFederateActor)actorToFire;
fedActor.fire();
try {

    if(timeToSendDataToRTI < timeWindow){
        timeToSendDataToRTI++;
    }else{
        if(fedActor.hasDataToSend()){
            //pedi o dado para o fedActor em questao e envia para o RTIFederator
            this.rtiFederation.sendData(fedActor.getDataToSend().stringValue());

        }
        double nextTime = nextHLAEventTime();
        double certiTime = rtiFederation.getRTINextTime();
        this.rtiFederation.advanceTimeTo(certiTime);

        timeToSendDataToRTI = 0;
        if(certiTime < nextTime)
            this.fireAt((Actor)fedActor, new Time(this, certiTime));
    }

} catch (RTIexception e) {
    System.out.println("Erro");
    e.printStackTrace();
}
```

Já o algoritmo do fluxo 2 da Figura 21 é representado na Figura 23. O importante é notar que essa parte do algoritmo verifica se existe algum dado a ser capturado do RTI (*receiveData()*) e enviado para o federado em questão (*updateAtributesToSend()*). Nesta parte do algoritmo também é exibido o avanço do tempo global do HLA (*nextHLAEventTime()*) .

Figura 23. Trecho de código que verifica se tem algum dado para ser capturado do RTI

```
/*******
HLA se tem algum dado a ser capturado do RTI
 *********/
Attributes attrs = rtiFederation.receivedData(nextHLAEventTime());
if(attrs!=null){
    if(actorToFire instanceof PtolemyFederateActor){
        PtolemyFederateActor fedActor = (PtolemyFederateActor)actorToFire;

        if(timeToGetDataFromRTI < timeWindow){
            timeToGetDataFromRTI++;
        }else{
            timeToGetDataFromRTI = 0;
            fedActor.updateAtributesToSend(attrs);
        }
        double nextTime = nextHLAEventTime();
        rtiFederation.consumeReceivedData(nextTime);
    }
}
```

5.2. Distribuindo um modelo do Ptolemy com HLA

Para possibilitar que um modelo comum que utiliza como MoC principal DE (*Discret Event*) possa ser distribuído utilizando o HLA, foram criados os diretores *SlaveHLADEDirector* e *MasterHLADEDirector* na biblioteca do Ptolemy, em que ambos são subclasses da classe principal, *HLADEDirector*, já citada anteriormente. Os

diretores criados foram adaptados para lidar com o HLA e apenas devem substituir os diretores comuns (DE *Director*). O HLADEDirector é quem possui toda a lógica relacionada ao uso dos serviços do RTI através das classes auxiliares, e as suas subclasses *SlaveHLADEDirector* e *MasterHLADEDirector* apenas instanciam as respectivas classes concretas SlaveFederate e MasterFederate para serem usadas pela classe mãe.

Além dos diretores, alguns atores envolvidos na simulação pura com Ptolemy devem também ser substituídos ou adicionados. Cada ator *SlaveFederateActor* ficará em máquinas distintas, onde os dados provenientes das portas de entrada desse ator serão enviados para o RTI. A quantidade de portas de entradas desse ator pode ser configurada na própria simulação. Tais dados poderão ser transformados em qualquer tipo de informação desejada de acordo com o interesse da aplicação. Nesse momento, é necessário a implementação por parte do desenvolvedor para lidar com os dados recebidos da forma desejada. Já o ator *MasterFederateActor* deverá ser adicionado também em outra máquina, ator esse que ficará recebendo em suas portas de saída as informações que já foram tratadas pelo RTI e assim tomará ações de acordo com a necessidade do projeto da simulação. Se for de interesse, também os dados recebidos na porta de entrada desse ator, poderão ser enviados para o RTI e tratados de acordo com as necessidades de projeto. Nesse caso, o *SlaveFederateActor* receberá em sua porta de saída os dados enviados pelo *master* e tratados pelo RTI.

É interessante notar que o ator *MasterFederateActor* deve atuar de acordo com o *MasterHLADEDirector*, ou seja, no mesmo “.xml” (projeto) da simulação. Sendo assim, esses dois atores devem estar juntos. Assim como o *SlaveFederateActor*, que deve estar junto com o *SlaveHLADEDirector*.

No Capítulo 6, será apresentado em detalhe como foi feita essa mudança de um projeto (deslocamento de pessoas) que utilizava o Ptolemy puro, para um projeto que possibilitava a simulação distribuída, ou seja, o Ptolemy integrado ao HLA

Capítulo

6

Resultados e Discussão

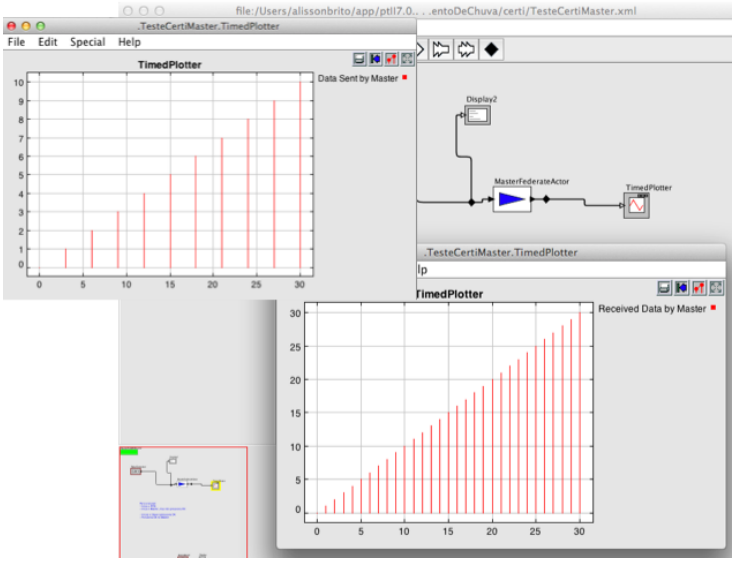
Este Capítulo trata dos resultados obtidos a partir dos experimentos realizados no Campus IV da UFPB. Inicialmente será apresentado como foi feito o desenvolvimento de duas provas de conceito. A primeira trata de um simples exemplo de troca de dados e *timestamp* para provar primeiramente que a integração entre o Ptolemy e o HLA é viável, e após isso mostrar também que os resultados são consistentes da simulação elaborada. A segunda prova de conceito é um exemplo mais contextualizado e também mais sofisticado, que lida com redes de sensores sem fio. Nesse caso, o exemplo é o deslocamento de pessoas em ambientes de médio para grande porte. Esta segunda prova de conceito foi elaborada em conjunto com outro aluno do grupo de pesquisa, em que o seu trabalho se limitou apenas ao uso do Ptolemy de forma pura, o que apresentou algumas limitações. Já o trabalho que está sendo apresentado nesta dissertação apresenta uma evolução do cenário anterior além de integrar o HLA ao modelo para possibilitar a simulação de grandes cenários e assim obter resultados mais realistas.

Percebe-se então que este capítulo tentará responder as duas perguntas de pesquisa, em que a primeira mostra que a integração do Ptolemy com o HLA é viável e também que o uso dessa integração possibilita simular grandes sistemas em larga escala. A segunda está relacionada a análise do desempenho de forma mais detalhada da integração proposta e com isso mostrar que tal integração é vantajosa quando comparado ao modelo que não há integração. No decorrer de todo este capítulo, estas perguntas serão respondidas de forma bastante clara. Desse modo, é possível projetar e simular sistemas reais, como o de deslocamento de pessoas em um ambiente qualquer, o que pode auxiliar, por exemplo, na segurança das pessoas.

6.1. Prova de conceito 1 - Estudo de caso

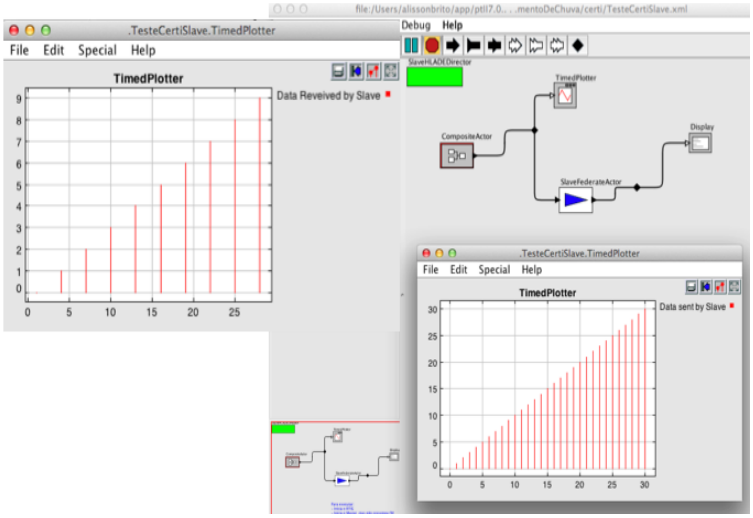
Nas Figura 24 e Figura 25 são apresentados os primeiros resultados da simulação dos atores desenvolvidos para interface com o RTI e HLA. Na Figura 24 é apresentado um modelo no qual o ator mestre é utilizado. Um gerador de números crescentes (de uma em uma unidade) envia dados a um período de 5 (cinco) unidades de tempo para a porta de entrada do ator mestre (*MasterFederateActor*), que por sua vez utiliza os serviços da interface *RTIAmbassador* para enviar os dados para o HLA. Observando a Figura 25 é possível constatar que esses mesmos dados (número de 1 a 10 – eixo Y) nos tempos exatos (eixo X) foram recebidos pelo ator escravo, executado em outro computador. A Figura 24, tem uma imagem que ao lado tem a seguinte informação, "*Data Sent by Master*", que são os mesmos valores recebidos pelo *Slave*, vide Figura 25, em que se tem "*Data Recieved by Slave*".

Figura 24. Exemplo de execução do ator mestre [*printscreen* da aplicação – elaborado pelo autor]



Ao mesmo tempo, o modelo apresentado na Figura 25 utiliza o ator escravo para enviar dados de volta ao ator mestre através do HLA. O mesmo gerador de dados é utilizado nesse modelo, mas dessa vez, num período mais curto, com 1 unidade de tempo. Também é possível constatar na Figura 24 que os dados foram recebidos corretamente pelo ator mestre.

Figura 25. Exemplo de execução do ator escravo [*printscreen* da aplicação – elaborado pelo autor]



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. Os dados foram transferidos corretamente e no tempo exato. Mesmo sendo executado num período de tempo mais longo, esta implementação pôde controlar o tempo do modelo utilizado pelo ator mestre, para que o mesmo só avançasse quando o dado vindo do ator escravo fosse recebido. Tal controle foi feito pelo HLA, que apresenta como um de seus grupos de serviços, o gerenciamento de tempo. Isso garantiu a consistência da simulação.

A principal utilidade desta prova de conceito é demonstrar que existe viabilidade técnica para a integração do Ptolemy II com o HLA, e com isso outros exemplos mais aprofundados podem ser elaborados (Seção 6.2).

6.2. Prova de conceito com cenário mais elaborado

O cenário projetado tem o objetivo de simular um ambiente no qual existem várias pessoas circulando em direção a um alvo específico e emitindo sinais para diversos sensores distribuídos neste mesmo ambiente. Os sensores detectam a presença de todas as pessoas que passam em sua região. Dessa forma, alguma ação pode ser tomada de acordo com os dados capturados.

Essa situação de movimento de pessoas foi projetada inicialmente em [84] e aprimorada por este trabalho. No anexo pode ser visto o cenário simulando uma rede de sensores sem fio em um campo de futebol. Contudo, para efeito didático, será apresentado um cenário diferente que é mais fácil de compreender. O projeto deste

cenário é interessante por apresentar um efeito prático e importante para a sociedade. Por exemplo, em um evento organizado e planejado em que acontece um incêndio em um ambiente fechado com milhares de pessoas, todas as pessoas deveriam sair do local, através das portas de emergência, sem danos sérios.

Esse tipo de situação pode ser implementada utilizando uma rede de sensores sem fio, na qual existem vários sensores base *RFID* espalhados nesse ambiente e as pessoas que estão caminhando neste local possuem também sensores *RFID* (ou *Smart Card*). Esse cenário foi designado previamente utilizando o Ptolemy de forma isolada. Os experimentos mostraram que quando este ambiente é simulado com 4 mil pessoas, a máquina que está executando a simulação sem o HLA não é capaz de suportá-la, ou seja, um erro no *heap* de memória da JVM (*Java Virtual Machine*) ocorre. Logo, uma das soluções para esta situação foi integrá-la a um modelo distribuído, ou seja, realizar a integração do HLA com o Ptolemy, de acordo com a explicação citada no Capítulo anterior. A forma como foi feita essa integração pode ser vista na Seção seguinte.

6.2.1. Cenário apenas com Ptolemy

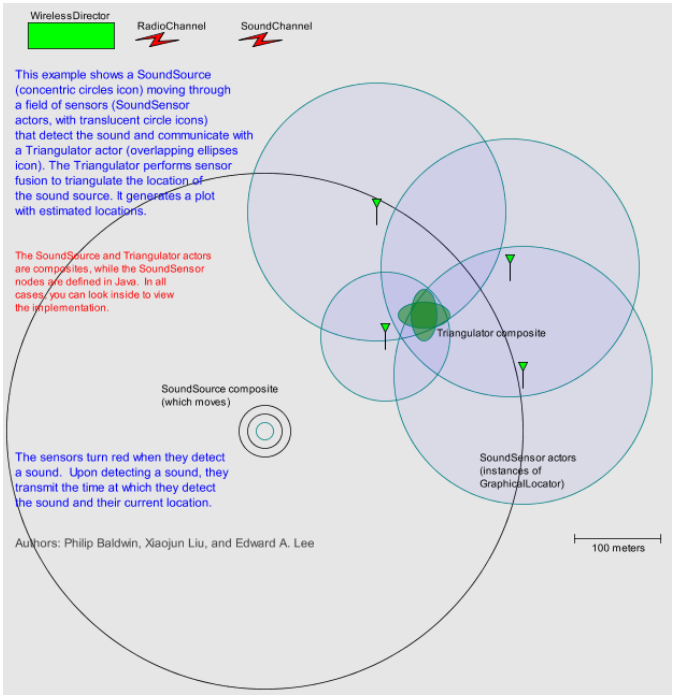
O cenário criado foi baseado em uma aplicação demo existente no próprio Ptolemy, chamada "*WirelessSoundDetection*.xml" (Figura 26).

Esse exemplo mostra o ator *SoundSource* (círculo concêntrico maior) se movendo através de um conjunto de sensores (ator *SoundSensor*, com círculo translúcido). Tais sensores detectam o *SoundSource* e se comunicam com o ator *Triangulator* (duas elipses sobrepostas em direções perpendiculares). O *Triangulator* atua como um sensor de fusão para dizer a localização do *SoundSource*.

Tanto o *SoundSource* quanto o *Triangulator* são atores compostos, ou seja, dentro deles existem outros atores de forma hierárquica. Já o *SoundSensor* está definido diretamente em JAVA e assim não apresenta hierarquia de atores dentro do mesmo.

Quando um som é detectado pelo *SoundSensor*, este transmite o momento da detecção do som e sua posição atual para o *Triangulator*.

Figura 26. Aplicação *demo SoundDetection* do Ptolemy **II [printsreen da aplicação demo]**

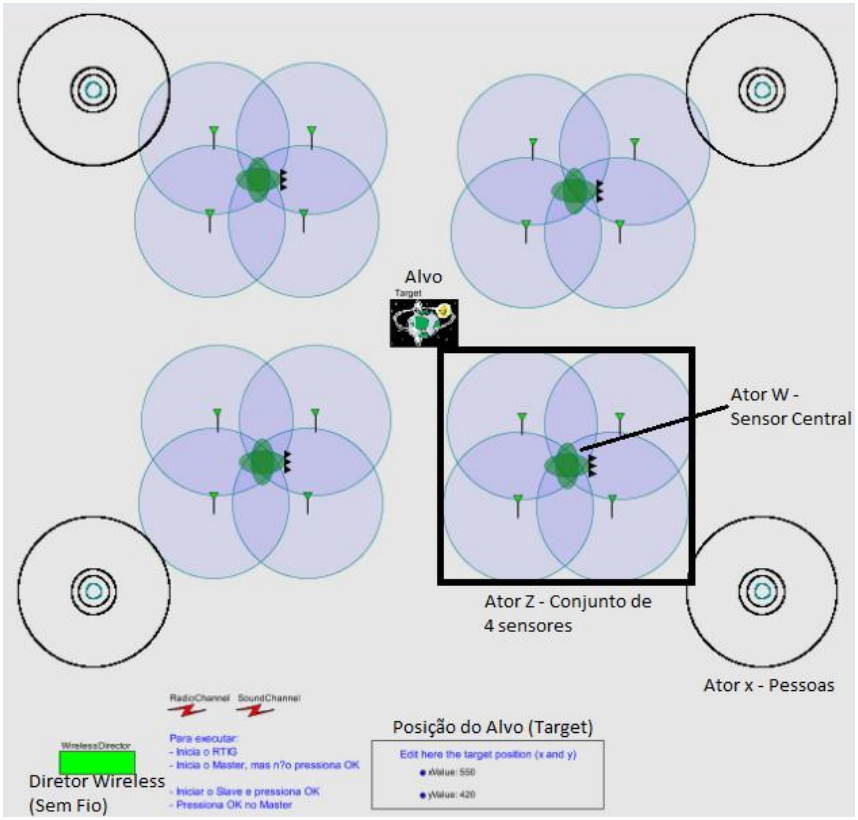


Um cenário específico foi criado a partir do exemplo demo citado anteriormente. O intuito de criar esse novo cenário é então possibilitar e avaliar de forma mais precisa o modelo de simulação distribuída proposto por este trabalho. A Figura 27 apresenta o cenário projetado que servirá como base para os experimentos expostos a seguir.

O cenário projetado tem o objetivo de simular um ambiente no qual existem várias pessoas (ator X) circulando em direção a um alvo específico (*target* – no centro) e emitindo sinais para diversos sensores distribuídos no mesmo ambiente. Os sensores detectam a presença de todas as pessoas que passam em sua região. A cada quatro sensores (ator Z), existe mais um sensor central (ator W) que captura os dados dos outros quatro sensores de sua região e envia pela rede, formando um conjunto de sensores.

Ainda na Figura 27, se pode observar a presença do *WirelessDirector* (Diretor Sem Fio) que é o responsável por fazer o gerenciamento do tempo e envio dos dados do cenário global em questão, já que este cenário se trata de uma rede de sensores sem fio. Nota-se também que a posição do *target* (alvo) pode ser alterada antes de iniciar a simulação, podendo o mesmo ser fixado em qualquer lugar do ambiente de acordo com o interesse do usuário.

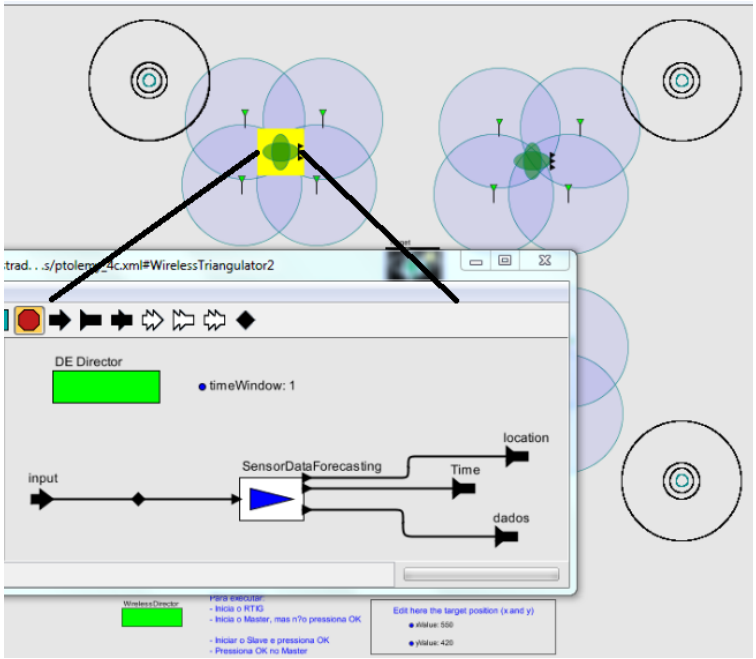
Figura 27. Cenário no ambiente com Ptolemy puro [*printscreen* da aplicação – elaborado pelo autor]



De forma mais detalhada, pode-se analisar os principais atores desse cenário, que serão substituídos ou adicionados a eles novos atores quando houver a integração com a arquitetura de alto desempenho (HLA).

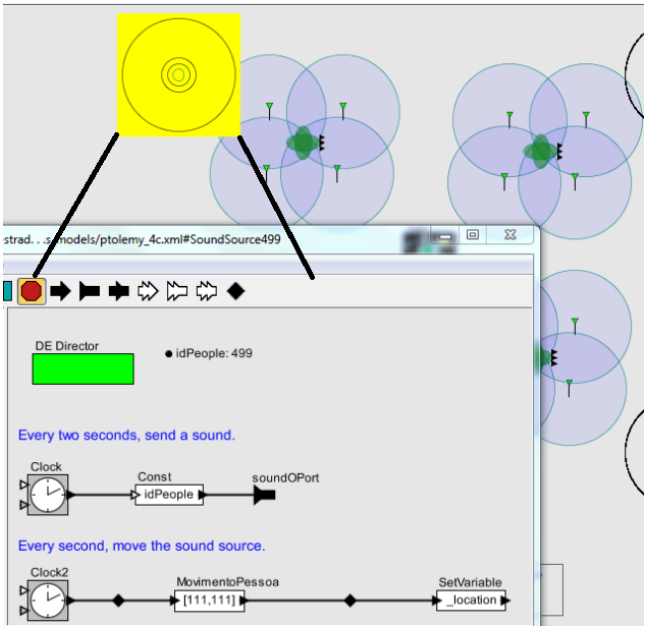
Por exemplo, na Figura 28 pode ser visto o chamado sensor central em detalhes. Este é um ator composto que contém como principais atores o *SensorDataForecasting*, que é uma classe criada especialmente para receber os dados capturados, através de sua porta de entrada, dos 4 sensores ao redor do sensor central e assim tratá-los da forma desejada. Esse ator que receberá as alterações para possibilitar a integração com HLA. As portas de saída do *SensorDataForecasting* apresentam os dados recebidos de forma mais limpa e tratada, como: a localização da pessoa capturada, o tempo (momento) em que essa pessoa foi capturada, e dados referente a essa captura de forma mais abrangente e detalhada (sensor que capturou a pessoa, a pessoa capturada, o sensor central que está lidando com a informação, e o momento da captura). Além desse ator, é interessante notar a presença do Domínio chamado DE, que lida com o MoC *Discret Event*. A variável *timeWindow* que existe nesse cenário é responsável por agrupar os resultados durante o período *timeWindow*, e só após esse período, liberar todos os dados agrupados para serem enviados para outros atores.

Figura 28. Sensor Central em detalhes do ambiente Ptolemy puro [*printscreen* da aplicação – elaborado pelo autor]



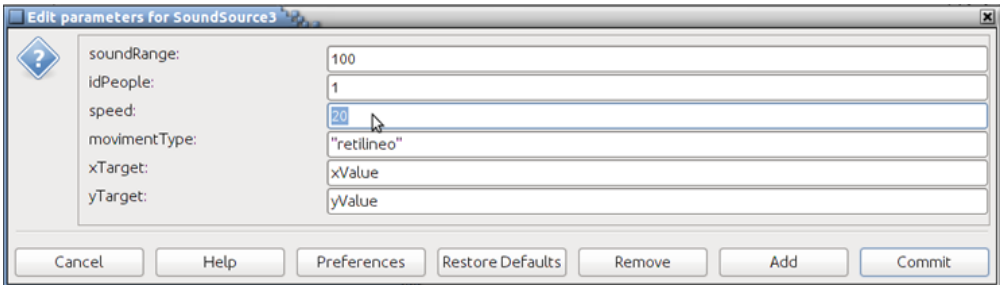
Outro ator composto é o *SoundSource*, no caso desse cenário em questão, a pessoa. A Figura 29 apresenta esse ator em detalhes internamente. Os atores principais que compõe o *SoundSource* são: MovimentoPessoa criado baseado em um ator do ambiente Ptolemy, chamado *Expression*, o qual apresenta como característica a lógica do deslocamento da pessoa para a posição X, Y dentro do ambiente a cada período de tempo; *setVariable* é outro ator responsável por fazer o valor que aparece em MovimentoPessoa deslocar a pessoa de fato para posição X e Y do ambiente; *idPeople* identificador único de cada pessoa. O diretor que gerencia o tempo e o envio dos dados das pessoas é *Discret Event* (DE *Director*).

Figura 29. SoundSource (Pessoa) em detalhes [*printscreen* da aplicação – elaborado pelo autor]



Ainda relacionado ao ator pessoa, este possibilita que alguns parâmetros sejam editados antes da simulação iniciar, ou até mesmo em tempo de execução da simulação. Tais parâmetros podem ser vistos na Figura 30. Pode-se observar os principais parâmetros, como: *idPessoa* (identificador único de pessoa), *speed* (velocidade com que a pessoa se desloca no ambiente), *movimentType* (tipo do movimento que deslocará a pessoa no ambiente, que pode ser, retilíneo, hiperbólico e parabólico), e por fim o *xTarget* e *yTarget* que não devem ser alterados, pois estes parâmetros contém a posição X e Y do alvo que será seguido pela pessoa.

Figura 30. Parâmetros do ator Pessoa [*printscreen* da aplicação – elaborado pelo autor]

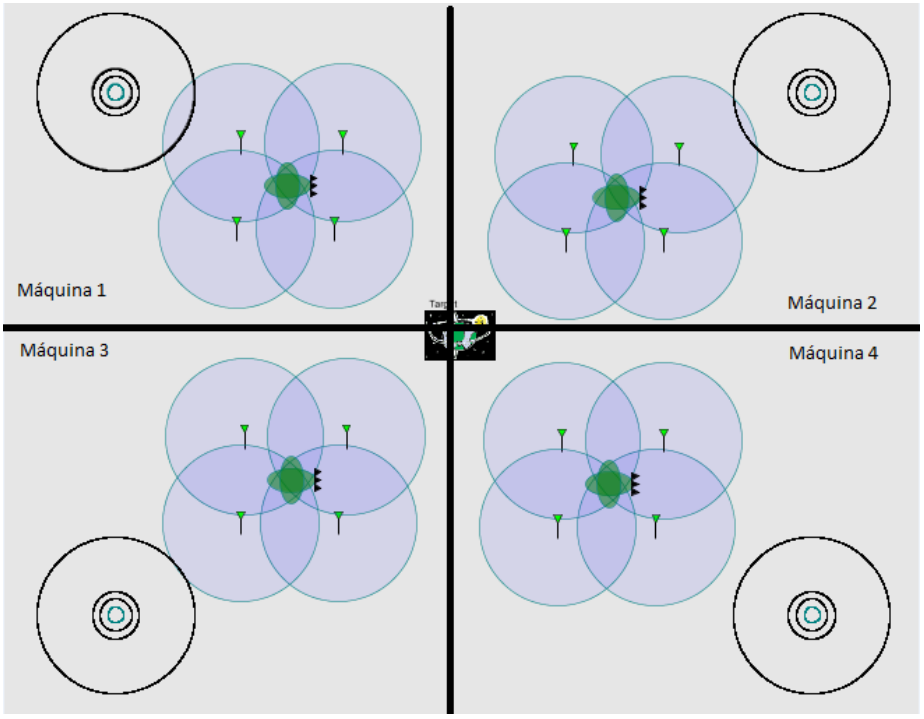


É interessante notar que o modelo em questão utiliza dois MoCs distintos, o primeiro é gerenciado pelo Diretor DE (*Discret Event*), que lida com o ator pessoa e o sensor central, e o segundo é o Diretor Wireless, que gerencia de forma global os sensores sem fio em se tratando do tempo e o envio dos dados da simulação.

6.2.2. Cenário integrado ao HLA

Para integrar o modelo proposto ao HLA é necessário realizar algumas mudanças em alguns atores do cenário e também na forma espacial como está dividido o cenário. Dessa forma, o modelo foi dividido em diversas partes, nas quais as mesmas pessoas envolvidas na simulação inicial continuarão andando ao redor do mesmo ambiente projetado. Tais pessoas estão dispersas de várias formas entre as máquinas escravas envolvidas na simulação. A Figura 31 mostra como pode ser feita tal divisão.

Figura 31. Divisão espacial do cenário do Ptolemy puro [*printscreen* da aplicação – elaborado pelo autor]



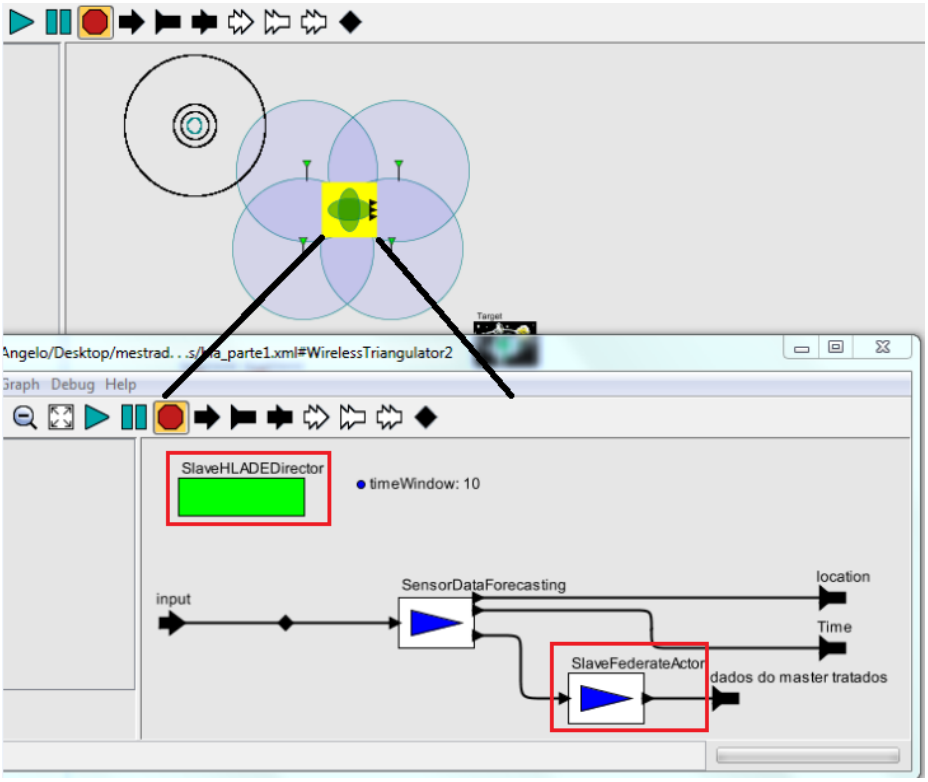
A Figura 32 apresenta como pode ficar tal divisão do modelo em diferentes máquinas. Como se pode notar, cada arquivo que está em uma máquina distinta irá apresentar parte das pessoas envolvidas na simulação, um conjunto de sensores, o alvo posicionado no mesmo lugar do modelo original, e os diretores responsáveis por fazer o gerenciamento de toda a simulação.

Figura 32. Divisão efetuada em máquinas escravas distintas [*printscreen* da aplicação – elaborado pelo autor]

O sensor central é um ator composto que sofreu as alterações necessárias para viabilizar a integração com o HLA. Como pode ser visto na Figura 33, este sensor contém tanto o *SensorDataForecasting* como também um novo ator, chamado *SlaveFederateActor* (moldado com um quadrado na Figura 33), que é o ator responsável por receber os dados em suas portas de entrada referente a captura das pessoas através dos sensores e enviá-los para o RTI. Os dados que irão para porta de entrada do *SlaveFederateActor*, que, por sua vez, será enviado ao RTI, são os dados do *SensorDataForecasting* que são referentes a essa captura de forma mais detalhada (sensor que capturou a pessoa, a pessoa capturada, o sensor central que está lidando com a informação, e o momento (time) da captura). O papel do *SlaveFederateActor* neste cenário é poder enviar tais dados ao RTI. Além desse ator, o diretor responsável por gerenciar as atividades que ocorrem nesse ator composto, também foi alterado. Criou-se um novo Diretor, chamado *SlaveHLADEDirector* (moldado com um retângulo na Figura 33), que é baseado no DE *Director*, adaptado para as necessidades do HLA.

Como pode ser notado na Figura 32 anterior, cada grupo de sensores que contém o *SlaveFederateActor* pode ser alocado em máquinas *slaves* distintas. Além de que, as pessoas envolvidas na simulação também podem estar espalhadas por várias máquinas *slaves*.

Figura 33. Sensor central em detalhes do escravo do cenário com HLA [*printscreen* da aplicação – elaborado pelo autor]



Além das máquinas escravas envolvidas na simulação, existe também para este caso específico, uma única máquina mestre isolada em uma máquina a parte da rede, que será a responsável por agrupar todos os dados resultantes das capturas dos sensores que, por sua vez, foram enviados ao RTI. É bom lembrar que o mestre só tem acesso aos dados capturados através do RTI (Regra 3 do HLA). Logo, a principal finalidade do mestre é ser uma espécie de gateway que receberá todos os dados envolvidos na simulação, de todas as máquinas escravas, através do RTI. Com os dados referentes a todos os escravos (federados) envolvidos na simulação, é possível realizar uma tomada de decisão, de acordo com a necessidade, que não faz parte do escopo desta dissertação.

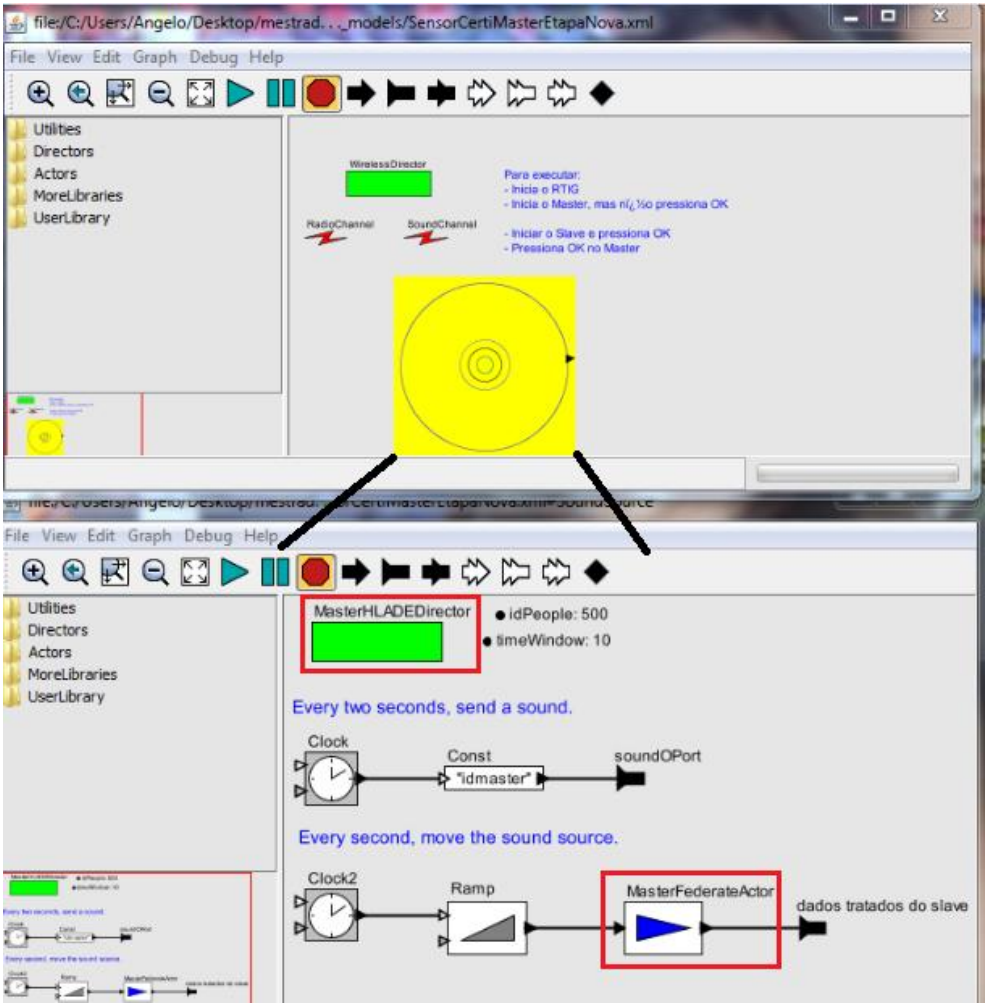
A Figura 34 a seguir mostra o projeto da máquina mestre em detalhes, projeto esse que não faz presente no cenário que contém o Ptolemy de forma isolada, pois nesse caso não existia a necessidade de um mestre (ou um ponto central), porque todos os atores (pessoas, sensores que capturam as pessoas, sensores centrais) estavam inseridos no mesmo modelo (ou projeto) de simulação e assim a captura dos dados não precisava ser enviada para nenhum outro ator de fora do modelo em questão.

Assim como nas máquinas escravas, todas eram gerenciadas pelo *WirelessDirector*, a máquina mestre também é gerenciada por esse mesmo diretor. O *soundSource* (esferas concêntricas) em questão é um ator composto que contém os

atores necessários para integrar o modelo por completo ao HLA. Nota-se então que esse ator contém o diretor *MasterHLADEDirector*, baseado no DE *Director*, responsável por gerenciar os dados que serão trocados entre os atores. O outro ator necessário para a integração com o HLA é o *MasterFederateActor*, que irá receber através do RTI todos os dados provenientes de todos os escravos, e assim irá enviar para a sua porta de saída (*dados tratados do slave*). Com tais dados na porta de saída, pode-se exibir na própria simulação tais dados, através de gráficos, animação, entre outras possibilidades. Os dados da porta de entrada desse ator são meramente ilustrativos, pois no exemplo em questão, o escravo não irá receber nenhum dado do mestre através do RTI. Os dados da porta de entrada do *MasterFederateActor* são enviados para o RTI também de tempo em tempo, que por sua vez, caso os escravos precisem, podem requisitar ao RTI tais dados, que serão copiados para sua porta de saída (*SlaveFederateActor – dados do máster tratados* – Figura 33). Os demais parâmetros desse ator não se fazem necessário uma explicação mais detalhada, já que não apresentam importância para este caso.

Para entender melhor, basta analisar a Figura 17 (Seção 5.1) que explica em detalhes a troca desses dados. Em resumo, o escravo envia dados para o RTI através do *SlaveFederateActor*, os dados que chegarão do *slave* para o RTI são os dados da porta de entrada do *SlaveFederateActor*. O mestre por sua vez, recebe os dados enviados pelos escravos, através do *MasterFederateActor*, também através da requisição via RTI. Os dados recebidos são copiados para a porta de saída do *MasterFederateActor*. O sentido contrário também pode ser realizado.

Figura 34. Mestre em detalhes do cenário com HLA [*printscreen* da aplicação – elaborado pelo autor]



Uma observação interessante é que tanto o *MasterHLADEDirector* quanto o *SlaveHLADEDirector* são subclasses do *HLADirector*, e é a classe principal que faz praticamente toda a lógica necessária para lidar com o RTI. Essa interação através das classes envolvidas pode ser vista na Seção 5.1.1.

Outra importante característica desse cenário criado é que o mesmo continuou preservando a característica de heterogeneidade quando foi integrado ao HLA. Os dois modelos, o com e o sem HLA, trabalham com dois diretores distintos, sendo o *WirelessDirector* o responsável pela gerencia global, e o DE *Director* responsável por gerências locais.

6.2.3. Resultados dos experimentos

Para efeito de comparação de modo abrangente, dois tipos de simulação foram executados nas mesmas condições, utilizando o seguinte cenário:

- 4000 pessoas + 4 conjuntos de sensores (simulação sem HLA);
- 4000 pessoas + X conjuntos de sensores, em que X é o número de máquinas escravas envolvidas na simulação.

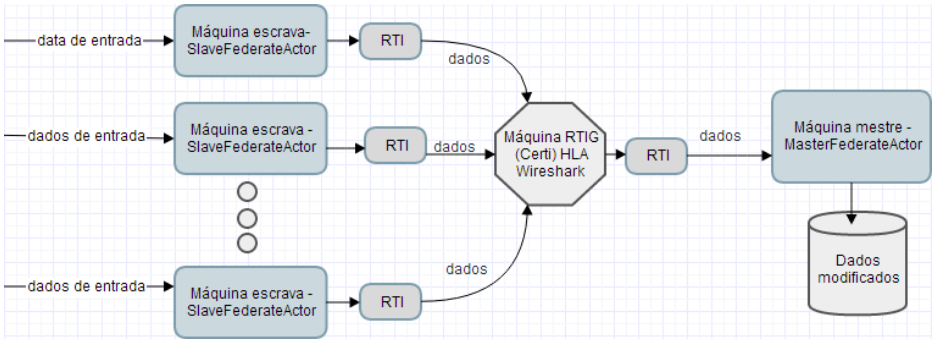
A intenção de conduzir esses experimentos é poder avaliar cada caso da simulação, e assim chegar a uma conclusão sobre o uso da simulação distribuída utilizando o HLA com o Ptolemy em sistemas de redes de sensores sem fio.

Para desenvolver os experimentos, utilizou-se dos seguintes recursos: 18 máquinas HP com as mesmas configurações – AMD *Core* 2 *quad* 2.1 GHz, 2 GB de memória RAM, 260GB de disco rígido, *Windows 7 Ultimate*, IDE de desenvolvimento Eclipse, Ambiente de Simulação Ptolemy II (uso da linguagem JAVA), Ambiente de Simulação Distribuída CERTI (HLA), API Sigar para monitorar o uso da CPU do PC durante a execução das simulações.

Os experimentos foram realizados da seguinte forma:

- Executou-se o experimento com o Ptolemy de forma isolada com apenas uma única máquina, apresentando 4 mil pessoas e 4 conjunto de sensores (Figura 27). Esse será chamado de modelo original. E foi realizada a captura dos dados necessários, como: pessoas capturadas pelos sensores, tempo de execução, *bytes* enviados e recebidos pela rede.
- Executou-se o experimento que apresenta a integração entre Ptolemy e HLA, em que esse mesmo experimento foi dividido em 16 casos. Nos casos a seguir, a quantidade de pessoas envolvidas no modelo inicial original, no caso 4 mil, foi dividida igualmente pelo número de máquinas existentes na simulação. Além disso, cada máquina tem um conjunto de sensor para capturar as pessoas que por nele passam. Os dados capturados são: pessoas capturadas pelos sensores, tempo de execução, *bytes* enviados e recebidos pela rede e uso da CPUs de cada máquina envolvida na simulação. Para esses casos foram utilizadas X máquinas escravas de acordo com os casos a seguir, uma máquina mestre a parte, e uma máquina que contém o RTIG do CERTI, implementação da RTI do HLA, que é por onde passam obrigatoriamente todos os dados da simulação. E essa mesma máquina RTIG tem o *software Wireshark* [85] que é o responsável por capturar os dados que estão sendo trocados na rede. A Figura 35 mostra tal arquitetura citada por onde passam os dados na rede.

Figura 35. Arquitetura geral da simulação [elaborado pelo autor]

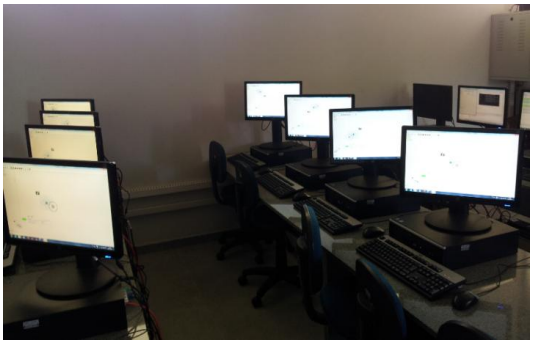


- Caso 1: Apresenta uma única máquina. Nesse caso essa única máquina possui 4 mil pessoas.
- Caso 2: Apresenta duas máquinas. Logo, 2 mil pessoas ficaram em uma máquina escravo e as outras 2 mil ficaram em outra máquina escravo.
- Caso 3: Apresenta três máquinas. Logo, 1333 pessoas ficaram cada máquina escravo envolvida.
- Caso 4: Apresenta quatro máquinas. Logo, 1000 pessoas ficaram cada máquina escravo envolvida.
- Caso 5: Apresenta cinco máquinas. Logo, 800 pessoas ficaram cada máquina escravo envolvida.
- Caso 6: Apresenta seis máquinas. Logo, 666 pessoas ficaram cada máquina escravo envolvida.
- Caso 7: Apresenta sete máquinas. Logo, 571 pessoas ficaram cada máquina escravo envolvida.
- Caso 8: Apresenta oito máquinas. Logo, 500 pessoas ficaram cada máquina escravo envolvida.
- Caso 9: Apresenta nove máquinas. Logo, 444 pessoas ficaram cada máquina escravo envolvida.
- Caso 10: Apresenta dez máquinas. Logo, 400 pessoas ficaram cada máquina escravo envolvida.
- Caso 11: Apresenta onze máquinas. Logo, 363 pessoas ficaram cada máquina escravo envolvida.
- Caso 12: Apresenta doze máquinas. Logo, 333 pessoas ficaram cada máquina escravo envolvida.
- Caso 13: Apresenta treze máquinas. Logo, 307 pessoas ficaram cada máquina escravo envolvida.

- Caso 14: Apresenta quatorze máquinas. Logo, 285 pessoas ficaram cada máquina escravo envolvida.
- Caso 15: Apresenta quinze máquinas. Logo, 266 pessoas ficaram cada máquina escravo envolvida.
- Caso 16: Apresenta dezesseis máquinas. Logo, 250 pessoas ficaram cada máquina escravo envolvida.

A Figura 36 apresenta o caso 8 sendo executado no laboratório do Campus IV da Universidade Federal da Paraíba, localizado na cidade de Rio Tinto.

Figura 36. Simulação do caso 8 sendo executado no ambiente real - Campus IV da UFPB



A quantidade de dados transferidos durante a simulação, como a média da banda da rede utilizada e o tempo total da simulação é apresentada na Tabela 5. As máquinas utilizadas nesses experimentos não suportaram o modelo com 4 mil atores, nos casos em que se utiliza apenas uma máquina no modelo Ptolemy puro, e nos casos em que existe a integração entre o Ptolemy e o HLA nos casos em que se utiliza 1, 2, 3 e 4 máquinas distintas. Logo, o resultado para essas situações, apresentadas na Tabela 5, foram estimados baseados em resultados previamente obtidos em experimentos com 1000 atores. A estimativa foi feita utilizando uma simples regra de três que pode ser visto nas tabelas seguintes (Tabela 3 e
Tabela 4).

Tabela 3. Estimativa de tempo de execução no modelo Ptolemy **puro**

Nº de atores	Time (s)
1000	450
4000	**x = 1800**

Tabela 4. Estimativa de tempo de execução no modelo com HLA para os casos de 1 a 4

Caso 1	
Atores	Tempo (s)
1000	6750
4000	**x = 27000**

Caso 2	
Atores	Tempo (s)
1000	5400
4000	**x = 21600**

Caso 3	
Atores	Tempo (s)
1000	4275
4000	**x = 17100**

Caso 4	
Atores	Tempo (s)
1000	3150
4000	**x = 12600**

Na Tabela 5 também é apresentada uma coluna com referência ao *speedup* da simulação em cada caso. Nesse caso, o *speedup* mostra como é o desempenho da simulação em comparação com o cenário estimado que utiliza apenas uma única máquina com o modelo Ptolemy puro, sem o HLA. Para obter tal *speedup* da abordagem deste trabalho, utilizou-se o conceito geral de *speedup*, em que o tempo de execução original é dividido pelo tempo de execução aumentado [54]. O cálculo para preencher essa tabela (coluna *speedup*) e o Gráfico 1 (linha "experimento") foi feito da seguinte forma.

- Tempo de execução original = tempo de execução do ambiente rodando o Ptolemy sozinho sem HLA = 30 minutos = 1800 segundos.
- Caso 1 - 1 máquina – S = 1800 / 27000 = 0.067
- Caso 2 – 2 máquinas – S = 1800 / 21600 = 0.083;
- E assim continua até o Caso 16.

A Tabela 5 mostra além dos dados relacionados ao *speedup*, também os demais dados para cada caso, como: tempo de execução (tempo total para a simulação ser completada), e bytes trocados (somatório dos bytes trocados durante a simulação entre todos os escravos e o mestre).

Tabela 5. Casos da simulação

Número de máquinas	Dados trocados (MBytes)	Bytes/seg	Tempo de execução (sec.)	Speedup
1 (est.)	501.301	18,57	27000,00	0,066667
2 (est.)	461.300	21,36	21600,00	0,083333
3 (est.)	411.303	24,05	17100,00	0,105263
4 (est.)	381.346	30,27	12600,00	0,142857
5	331.451	38,92	8516,00	0,211367
6	301.301	67,56	4214,89	0,427057
7	255.372	458,48	535,94	3,358591
8	189.450	465,48	395,01	4,55687
9	177.336	89,56	1882,50	0,956174
10	160.747	178,61	959,95	1,875102
11	134.228	93,80	1442,86	1,247521
12	111.882	91,03	1265,61	1,422235
13	116.750	123,15	949,79	1,895152
14	100.582	310,44	326,87	5,506827

15	104.414	139,22	751,19	2,396204
16	105.314	186,73	567,65	3,170946

No Gráfico 1, o *speedup* utilizando o HLA com o Ptolemy é apresentado com relação ao *speedup* teórico proposto pela lei de Amdahl [83], que estima o *speedup* esperado quando se usa processadores paralelos *versus* a utilização de um único processador. Esse *speedup* depende da porção de instruções paralelas sendo utilizadas com relação as instruções sequenciais. No Gráfico 1, a curva de Amdahl foi plotada com porções de 95%, 90%, 75% e 50% de paralelismo.

Os cálculos utilizados para plotar o gráfico com as porções citadas de paralelismo de Amdahl foram feitos utilizando como base a Lei de Amdahl, apresentada no Capítulo 5 (Simulação Distribuída). A equação utilizada foi a do *Speedup* Paralelo, a fim de obter os resultados deste trabalho com relação ao *speedup* de Amdahl. Por exemplo, para o cálculo do *speedup* com relação a 95% de paralelismo a equação fica da seguinte forma.

$S(f,n) = 1 / (1 - f) + f / n$, em que f é a fração paralelizada (95% no caso), e n é o número de processadores (ou cores envolvidos), que vai variar de 4 a 64 (Gráfico 1). Os resultados ficam os seguintes:

- 1 máquina = 4 núcleos = S = 1 / (1 – 0.95) + (0.95/4) = 3,478261
- 2 máquinas = 8 núcleos = S = 1 / (1 – 0.95) + (0.95/8) = 5,925926
- 3 máquinas = 12 núcleos = S = 1 / (1 – 0.95) + (0.95/12) = 7,741935
- 4 máquinas = 16 núcleos = S = 1 / (1 – 0.95) + (0.95/16) = 9,142857
- E assim segue a mesma lógica até chegar ao último caso.
- 16 máquinas = 64 núcleos = S = 1 / (1 – 0.95) + (0.95/64) = 15,4216

A Tabela 6 mostra os demais resultados utilizados para plotar o gráfico apresentado, para o caso de utilizar a fração 0.95 e também para as demais frações, como 0.9 , 0.75 e 0.5.

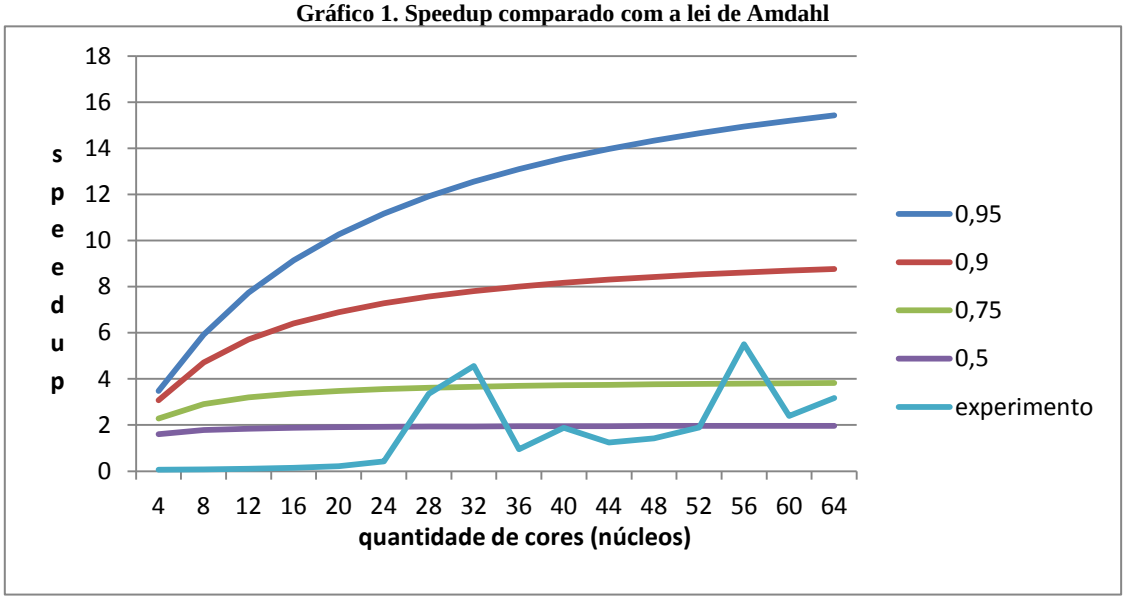
Tabela 6. Lei de Amdahl

Máquinas	**Núcleos**	**0,95**	**0,9**	**0,75**	**0,5**
1	**4**	3,47826087	3,076923077	2,285714286	1,6
2	**8**	5,925925926	4,705882353	2,909090909	1,777777778
3	**12**	7,741935484	5,714285714	3,2	1,846153846
4	**16**	9,142857143	6,4	3,368421053	1,882352941
5	**20**	10,25641026	6,896551724	3,47826087	1,904761905
6	**24**	11,1627907	7,272727273	3,555555556	1,92

7	**28**	11,91489362	7,567567568	3,612903226	1,931034483
8	**32**	12,54901961	7,804878049	3,657142857	1,939393939
9	**36**	13,09090909	8	3,692307692	1,945945946
10	**40**	13,55932203	8,163265306	3,720930233	1,951219512
11	**44**	13,96825397	8,301886792	3,744680851	1,955555556
12	**48**	14,32835821	8,421052632	3,764705882	1,959183673
13	**52**	14,64788732	8,524590164	3,781818182	1,962264151
14	**56**	14,93333333	8,615384615	3,796610169	1,964912281
15	**60**	15,18987342	8,695652174	3,80952381	1,967213115
16	**64**	15,42168675	8,767123288	3,820895522	1,969230769

O eixo X do Gráfico 1 a ser apresentado a seguir representa o número de unidades de processamento envolvidas (cores).

Gráfico 1. Speedup comparado com a lei de Amdahl



Através do gráfico é possível ver que na abordagem proposta por este trabalho o *speedup* foi menor que 1 até quando se utilizou 24 cores (6 máquinas), não havendo muito benefício. Já no caso de 28 e 32 cores (7 e 8 máquinas, respectivamente) o *speedup* foi em média 75% de paralelismo e piorou para em torno de 50% quando se utilizou de 36 a 52 cores (9 a 13 máquinas, respectivamente). E novamente o *speedup* aumenta quando se usa 56 cores (14 máquinas) e diminui em 15 e 16 máquinas. Observa-se então que a tendência de *speedup* é se manter, mesmo que com poucas variações, na faixa entre 50% e 75% de paralelismo, podendo algumas vezes ter um pico um pouco maior que 75%. Logo, o melhor custo benefício seria utilizar 8 máquinas, já que até essa quantidade de computadores o comportamento é mais previsível, e o custo é menor usando 8 do que 14 máquinas, já que o tempo de execução e o *speedup* de ambos os casos é bem próximo.

O comportamento irregular da curva de *speedup* após o caso 8 e dos demais gráficos a serem apresentados posteriormente pode ser justificado por algumas hipóteses a serem constatadas, como por exemplo, a política de gerenciamento de tempo (*lookeahed, NMA algorithm)* e o esforço computacional limitado pelo número de máquinas para cada situação específica. Essas duas hipóteses foram estudadas e os resultados são apresentados a seguir.

1. A política de gerenciamento de tempo

O algoritmo de sincronização conservativa do CERTI, que é uma implementação do HLA, foi o utilizado durante a simulação. Tal algoritmo é baseado no conhecido conceito "*NULL Message Algorithm*" (NMA) de Chandy e Misra [82]. Essa abordagem é baseada no contrato de cada federado chamado de *lookahead*. Cada federado tem uma regra em que não pode enviar mensagens de simulação com o *timestamp* menor que o seu próprio tempo local mais o *lookahead*. Dessa forma, o valor do *lookahead* pode influenciar no desempenho da simulação. Para que esse algoritmo possibilite a sincronização de forma correta, mensagens adicionais são trocadas, as chamadas *NULL messages* (mensagens contendo apenas o *timestamp*), no intuito de preencher o tempo necessário até poder enviar o dado em si. Logo, tais mensagens ajudam a indicar quando que mensagens futuras com dados possam ser enviadas, obedecendo assim o *timestamp* em questão.

A principal deficiência dessa abordagem para simulações em tempo real e de alto desempenho é o *overhead* que a comunicação de várias mensagens nulas entre os simuladores implica. Ainda, se o parâmetro de *lookahead* não é bem escolhido, a simulação está sujeita ao problema conhecido por "*lookahead time creep*", ou seja, o número de mensagens nulas (*NULL messages*) enviadas pode ser inaceitável e isso pode limitar o desempenho da simulação [78]. No resultado apresentado, utilizou-se o tempo de *lookahead* igual a 10, o que pode ter causado certa instabilidade no experimento em alguns casos, como explicado anteriormente. Sabe-se que cada configuração da simulação apresenta o tempo de *lookahead* ideal para a maioria dos casos apresentados para poder obter o *speedup* máximo. No caso dos resultados apresentados por esse experimento demonstrou que o *speedup* ideal utilizando o *lookeahead* com valor igual a 10 acontece quando se usa 8 ou 14 máquinas e o uso de mais de 8 máquinas pode piorar o desempenho, o qual pode ser justificado pelo *overhead* que os serviços de

gerenciamento apresentam, em especial, o envio de mensagens nulas para possibilitar a sincronização correta, que é baseado no valor de *lookahead* escolhido (1ª hipótese apresentada). É interessante então notar que do caso 1 ao caso 8 existe uma tendência de melhora de *speedup*, e após o caso 8, o experimento passa a ficar mais instável, apresentando poucas variações na maioria das situações.

Para testar essa hipótese, um experimento adicional simples foi realizado em que utilizou-se um número fixo de 9 máquinas, que é o momento em que o gráfico começa a ficar instável, para simular um ambiente com 4000 pessoas, e variou-se o *lookahead* da simulação. Assim, o resultado obtido para este caso específico de 4000 pessoas, com 9 máquinas e variando o *lookahead* entre 10, 50 e 100, foi bem semelhante um ao outro. Dessa forma, essa hipótese não foi estudada de forma mais detalhada nesta dissertação, mas também não pode ser descartada completamente, pois pode-se realizar estudos mais profundos, variando a quantidade de máquinas, *lookahead*, entre outras variáveis da simulação. Para esta dissertação, a hipótese a seguir, no caso, foi a principal para se analisar durante o restante deste capítulo.

2. Esforço computacional limitado pelo número de máquinas

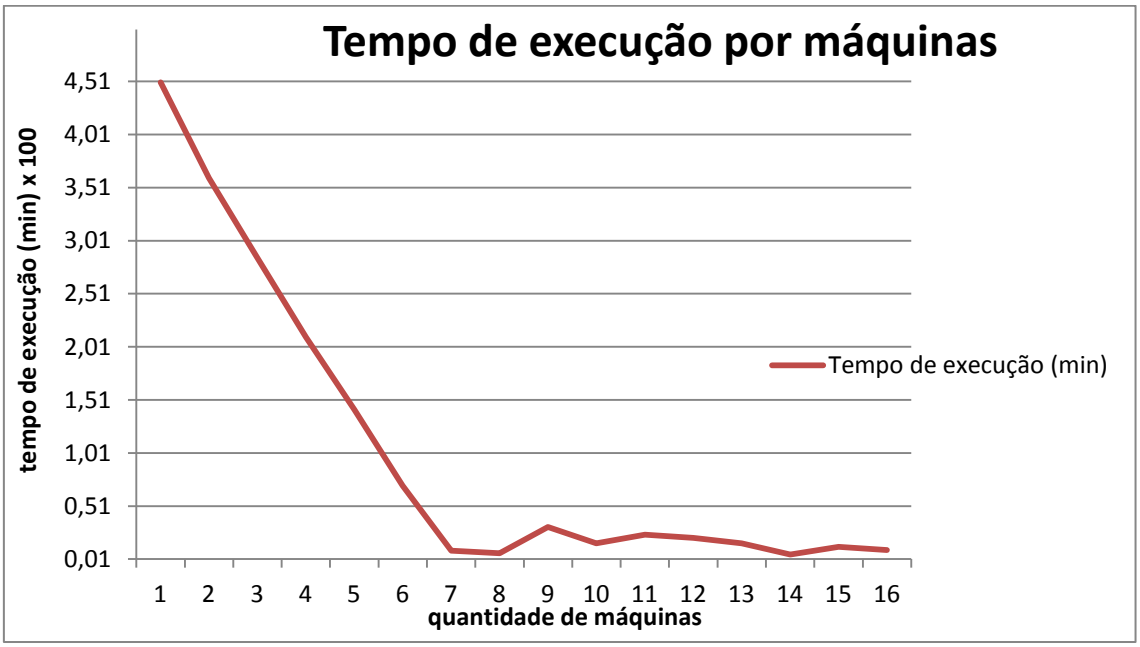
Uma segunda hipótese foi levantada e estudada de forma mais detalhada por este trabalho para tentar entender o comportamento irregular das curvas em todos os gráficos apresentados nesta dissertação. A hipótese se baseia no esforço computacional em que após o uso de determinada quantidade de máquinas, em que o máximo de desempenho já é atingido, tal esforço aumenta mesmo não havendo necessidade alguma, pois adicionando novas máquinas na simulação sem necessidade, estas tendem a ficar ociosas, e o HLA, neste caso, trabalha apenas para manter a sincronização correta. Nesta hipótese, não se considerou a variação do valor do *lookahead.* Logo, o objetivo está em descobrir para cada caso o número ideal de máquinas a serem utilizadas na simulação, já que após tal número, as máquinas passam a ficar mais ociosas. O estudo de ociosidade das CPUs é feito no decorrer deste capítulo.

O trabalho de Bononi [17], por exemplo, apresentado na Figura 1 exibe o *speedup* de uma simulação que utiliza o HLA, e em uma determinada configuração, quando se usou até 4 unidades de processamento houve aumento de *speedup*, mas quando utilizou-se 8 unidades de processamento, o *speedup* decresceu, comportamento semelhante ao que acontece nos experimentos dessa dissertação. Porém, o trabalho de

Bononi limitou seu experimento em apenas 8 unidades de processamento, o que não possibilita se fazer uma comparação mais detalhada com esta dissertação. Ainda, este mesmo trabalho mostra que para os casos mais leves (menos veículos, pessoas) o uso de muitas máquinas pode ter um *speedup* menor quando comparado aos casos mais pesados (mais veículos, pessoas), utilizando a mesma quantidade de máquinas. Isso pode ocorrer pelo motivo que esta segunda hipótese considera.

Além do *speedup* outros fatores podem ser analisados com mais detalhes, como: CPU, tempo de execução, dados enviados e recebidos. Os gráficos (Gráfico 2 e Gráfico 5) apresentam comportamentos semelhantes, em que o primeiro mostra o tempo de execução dos experimentos, e o segundo exibe o esforço computacional da simulação, que significa dizer o percentual de uso médio da CPU de cada máquina envolvida na simulação, multiplicado pelo tempo de execução da simulação. Ambos os gráficos apresentam os resultados dos experimentos do caso 1 ao 16. Observa-se do caso 1 ao caso 8 a diminuição significativa dos valores em questão, já do caso 9 ao caso 16, outro comportamento é observado: os valores não diminuem e nem aumentam tanto, apresentando assim poucas variações.

Gráfico 2. Experimento 4000 pessoas – análise do tempo de execução



Observa-se no gráfico de tempo de execução, que este cai drasticamente quando se aumenta a quantidade de máquinas na simulação, porém depois de 8 máquinas o tempo de execução tende a variar pouco, com picos acontecendo em alguns momentos, o que mostra que após o caso 8, assim como na curva do *speedup*, o comportamento mesmo tendendo a ser constante apresenta instabilidades. Esse mesmo comportamento

pode ser observado no trabalho de Guha, em que o tempo decresce rapidamente até determinado ponto, e após esse marco, o tempo tende a não apresentar variações significativas. Esse comportamento pode ser explicado pelo aumento da ociosidade da CPU após o uso de mais de 8 máquinas (Gráfico 6).

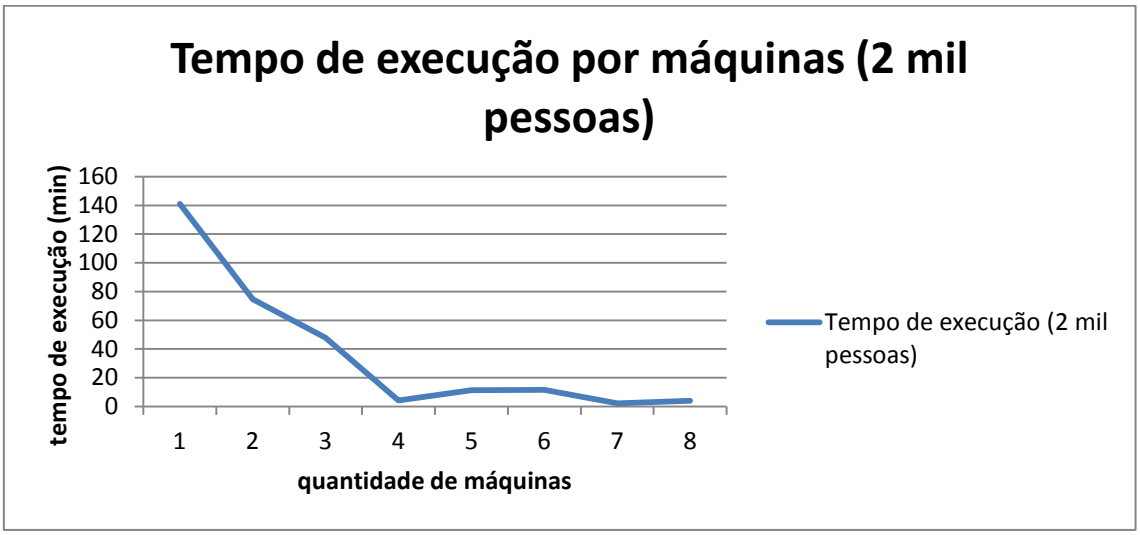
Sabe-se ainda que, inicialmente o tempo de execução é estimado em 30 minutos para as configurações estabelecidas para o caso em que não se utiliza o HLA. O tempo, no caso, é estimado, o que quer dizer que na verdade, o ambiente não suportou esse modelo utilizando apenas uma única máquina sem HLA. Por outro lado, utilizando pelo menos 5 máquinas com HLA, o ambiente já suporta a simulação mesmo que o tempo de execução seja em torno de 2 horas e 30 minutos. No caso do uso de 7 máquinas, o tempo é em média 9 minutos, ou seja, já passa a ser melhor que o próprio tempo estimado em 30 minutos. Com isso, é possível também chegar a conclusão de que o uso da simulação com HLA além de viabilizar a simulação (possibilitando que seja executada até o fim), também apresente um desempenho melhor em várias situações, por exemplo, quando se utiliza 7 máquinas, em comparação ao modelo que não utiliza HLA.

Para realizar a medição do tempo de execução, utilizou-se a própria API do Ptolemy. Quando a simulação iniciava-se, os milissegundos daquele horário eram capturados, e quando terminava também capturava-se os milissegundos daquele horário, e por fim, realizava-se a subtração entre as duas capturas e o resultado gerava o tempo total de execução da simulação.

Para confirmar o comportamento instável da curva no momento em que se aumenta até um número específico de máquinas, então realizou-se um novo experimento, porém dessa vez, com apenas 2 mil pessoas envolvidas e variando de 1 até 8 máquinas. Percebeu-se então um comportamento semelhante ao experimento com 4 mil pessoas, em que no caso de 2 mil pessoas quando se tem de 1 até 4 máquinas envolvidas na simulação a curva decresce significativamente e após 4 máquinas, a curva passa a ficar um pouco instável. No caso de 2 mil pessoas, o melhor custo benefício pode estar no uso de 4 máquinas apenas, já que é nesse ponto que se obtém um dos menores tempo, com a menor quantidade de máquinas envolvidas. Nesse caso, pode ser melhor usar 4 máquinas para se obter um tempo de 4 minutos, do que usar 7 máquinas e ter um tempo de 3 minutos. Já no caso explicado anteriormente com 4 mil pessoas era necessário ter 8 máquinas para se ter um bom custo benefício. Em ambos os experimentos (4 mil e 2 mil pessoas) o gráfico apresentou um comportamento

semelhante após o uso de uma quantidade específica de máquinas, que pode ser explicado pelo aumento da ociosidade da CPU como será detalhado posteriormente.

Gráfico 3. Experimento 2000 pessoas – análise do tempo de execução



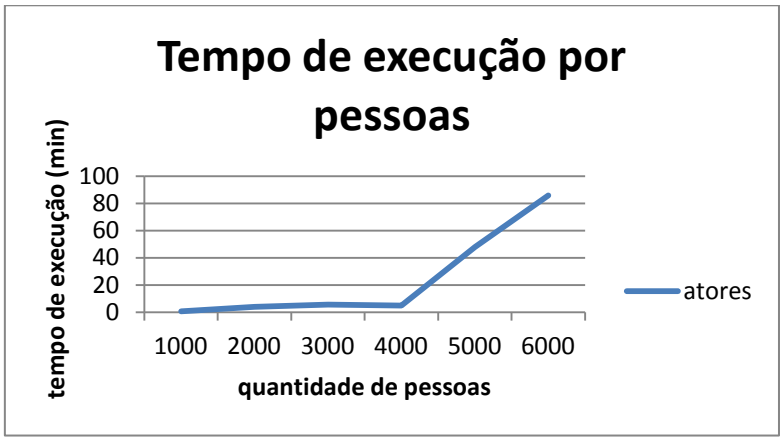
Devido a esse comportamento semelhante aos dois casos (4000 e 2000 pessoas), outro experimento foi realizado no intuito de verificar o comportamento do tempo de execução na situação em que é fixado o número de 8 máquinas, e se varia a quantidade de atores envolvidos, entre 1000 até 6000 atores. Analisando a Tabela 7 e o Gráfico 4 percebe-se que com 8 máquinas é possível ter um bom tempo de execução até com 4000 atores, porém acima de 4000 atores, o uso de 8 máquinas pode não ser o suficiente, já que o tempo de execução sai de 4,8 minutos do caso com 4000 pessoas para 47 minutos do caso com 5000 pessoas. Nesse caso, para encontrar o número ideal de máquinas para a configuração dada, por exemplo, no caso de 5000, se fazem necessário novos experimentos.

Tabela 7. Tempo de execução X Quantidade de atores

Quantidade de atores	* Tempo médio (minutos)
1000	0,529997917
2000	3,904975
3000	5,60434375
4000	4,8481
5000	47,87252292
6000	85,75711875

* Tempo médio: somatório do tempo de execução de cada uma das 8 máquinas envolvidas na simulação dividido por 8.

Gráfico 4. Gráfico do Tempo de execução x quantidade de atores



É interessante também notar que de 2000 a 4000 pessoas, usando essas mesmas 8 máquinas, o tempo de execução apresentado é aproximado e fica em média 4,7 minutos. Então para 8 máquinas, pode-se utilizar de 2 mil a 4 mil pessoas que o desempenho não será muito diferente. Por outro lado, pode-se também fazer um estudo mais detalhado para o caso de 3000 pessoas, pois observa-se que o melhor custo X benefício pode não ser necessariamente 8 máquinas, já que no experimento em questão com menos atores, no caso 3 mil, o tempo de execução foi maior do que o tempo obtido com 4 mil atores para a mesma quantidade de máquinas envolvidas na simulação. Isso pode indicar que o uso de 8 máquinas nesse caso seja excessivo, apresentando assim CPUs mais ociosas, não gerando benefício algum (2ª hipótese). Nesse caso específico, o desempenho máximo pode ser atingido utilizando menos até que 8 máquinas. Assim, levando em consideração a 2ª hipótese, pode ser que com menos máquinas se encontre o melhor custo benefício do que com 8 máquinas no caso de 3 mil atores envolvidos na simulação. Para ter essa certeza é necessário que se façam novos experimentos para encontrar o número ideal de máquinas na configuração em questão da simulação. Logo, percebe-se que para cada configuração tem-se o número ideal de máquinas para se usar.

Voltando a explicação do experimento inicial com 4 mil pessoas, além do tempo de execução pode-se medir também o esforço computacional através do percentual de uso da CPU em cada máquina envolvida na simulação. Para isso, utilizou-se a API Sigar para realizar essa medição em cada máquina da simulação. Um código específico foi utilizado em uma *Thread* a parte e quando a simulação era iniciada, essa *Thread* também o era. A figura a seguir mostra o trecho de código que usa a API Sigar [86] e grava os valores capturados em um arquivo.

Figura 37. Trecho de código do uso da API Sigar

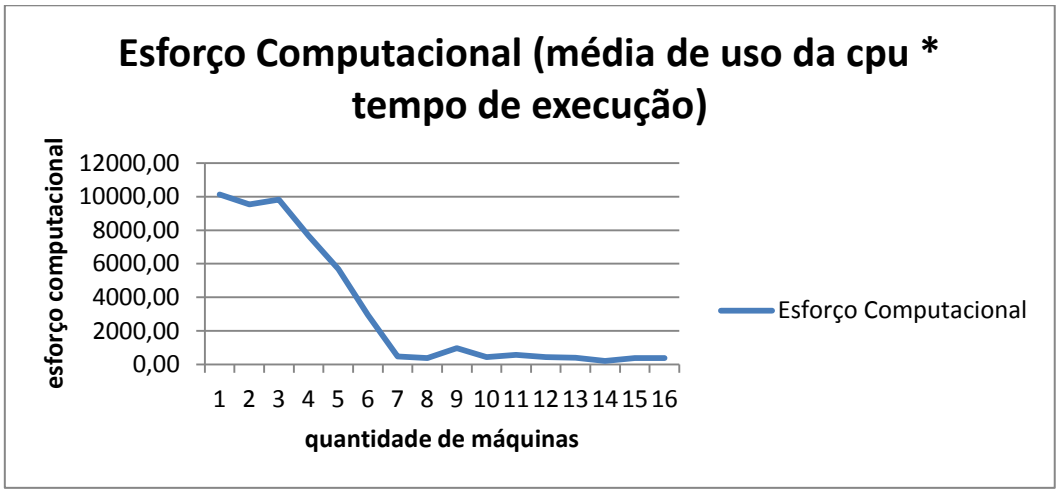
```
CpuPerc[] cpus = null;
try {
    cpus = this.sigar.getCpuPercList();
} catch (SigarException e1) {
    e1.printStackTrace();
}
while(true){
    try {
        this.writer = new BufferedWriter(new FileWriter("cores.csv", true));

        cpus = this.sigar.getCpuPercList();
         this.writer.write(System.currentTimeMillis() + ";");
            for (int i=0; i<cpus.length; i++) {
                this.writer.write(CpuPerc.format(cpus[i].getUser() + cpus[i].getSys()) + ";");
                Thread.sleep(300);
            }
            this.writer.write("\n");
            this.writer.close();
    } catch (SigarException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

Finalizada a simulação, cada arquivo gerado era lido, e fazia-se uma média dos percentuais de uso da CPU que estavam contidos em cada arquivo, que por sua vez, foram gerados por cada máquina em cada caso específico, do 1 ao 16. Com essa média calculada, multiplicava-se esse valor pelo tempo de execução daquele caso específico, e assim obtinha-se o valor do esforço computacional. A tabela a seguir mostra os percentuais médio de uso de cada CPU em cada caso, e o respectivo valor do esforço computacional. A coluna do esforço computacional foi a utilizada para gerar o gráfico a seguir.

Tabela 8. Experimento 4000 pessoas – análise do esforço computacional

Casos	Média de uso CPU (%)	Tempo (minutos)	Esforço computacional
1	22,50	450,00	10125,00
2	26,50	360,00	9540,00
3	34,50	285,00	9832,50
4	36,50	210,00	7665,00
5	40,00	141,93	5677,33
6	42,10	70,25	2957,37
7	52,78	8,93	471,44
8	58,04	6,58	382,11
9	30,90	31,38	969,54
10	26,82	16,00	429,16
11	23,81	24,05	572,67
12	20,43	21,09	430,93
13	25,15	15,83	398,12
14	36,86	5,45	200,82
15	30,42	12,52	380,85
16	40,53	9,46	383,40

Gráfico 5. Experimento 4000 pessoas – análise do esforço computacional



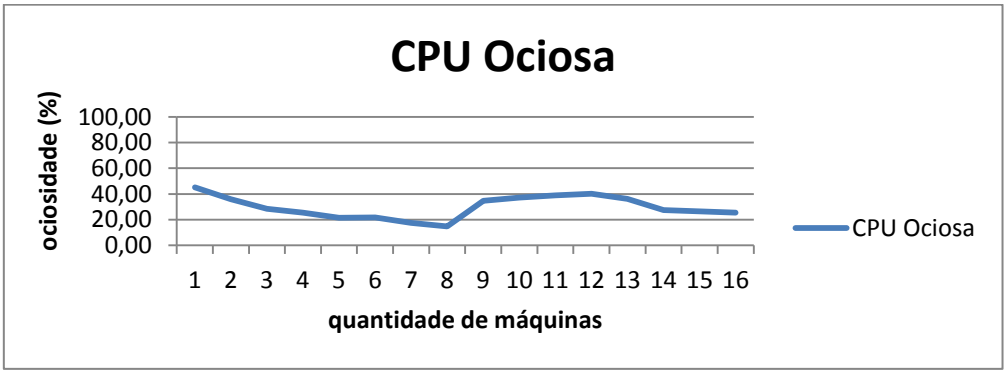
Como já observado anteriormente, o comportamento do tempo de execução é bem semelhante ao do esforço computacional, em que até o caso 7 ou 8, o esforço computacional decresce substancialmente a medida em que se aumenta o número de máquinas envolvidas na simulação. Após isso, aumentar o número de máquinas pode não surtir muito efeito, tendendo também a apresentar variações mínimas nos demais casos. A hipótese então levantada é que o uso de mais de 8 máquinas no caso de 4000 pessoas é desnecessário já que o esforço computacional apresenta poucas variações e pode existir apenas para manter a sincronização do HLA, já que várias máquinas tendem a ficar ociosas por mais tempo quando se aumenta o número de máquinas, fazendo com que o trabalho de detectar as pessoas de fato ocorra de forma menos enfática. Isso significa dizer que o melhor custo benefício nas configurações realizadas pode estar em 8 máquinas, pois pode-se até aumentar a quantidade de máquinas na simulação, porém o ganho não é significativo em relação ao uso de 8 máquinas, então é melhor usar 8 máquinas, do que 16, já que o esforço computacional e o próprio tempo de execução é bem semelhante em ambos os casos.

Os experimentos anteriores, o primeiro com 2 mil pessoas, o segundo variando entre 1 mil até 6 mil pessoas com o número fixo de máquinas, e os demais com 4 mil pessoas foram uteis para mostrar que existe um comportamento semelhante da simulação para todos esses casos, e que se faz necessário tentar testar alguma hipótese do porque deste comportamento da simulação.

O Gráfico 6 tenta validar a hipótese levantada, mostrando assim o quão ociosa a CPU pode ficar em cada caso específico. Assim como os demais gráficos, é possível perceber uma tendência até o caso 8, e após tal caso, ocorre instabilidade no decorrer do gráfico. A CPU é considerada ociosa para os efeitos deste trabalho quando o seu uso

está abaixo de 10%. Assim, é possível notar no gráfico três comportamentos, em que do caso 1 ao 8, a ociosidade diminui com o tempo (quanto maior o valor, mais ociosa é a CPU), do caso 9 ao 12 a ociosidade aumenta, e do caso 13 ao 16 a CPU volta a ficar menos ociosa, porém nesta última parte, tendendo a apresentar poucas variações. Assim, percebe-se também que no caso 8 é o momento em que a CPU fica menos ociosa (o que pode representar o melhor custo benefício), e após isso a tendência é ficar mais ociosa, permanecendo na ociosidade com o aumento da quantidade de máquinas. Logo, após o caso 8, como afirma a hipótese em questão é que já existem máquinas suficientes para lidar com o problema específico, e que a adição de novas máquinas na simulação de nada trará novos benefícios. Esta explicação pode ser a razão da instabilidade dos gráficos dos experimentos anteriores após o caso 8.

Gráfico 6. Experimento 4000 pessoas – análise da ociosidade da CPU



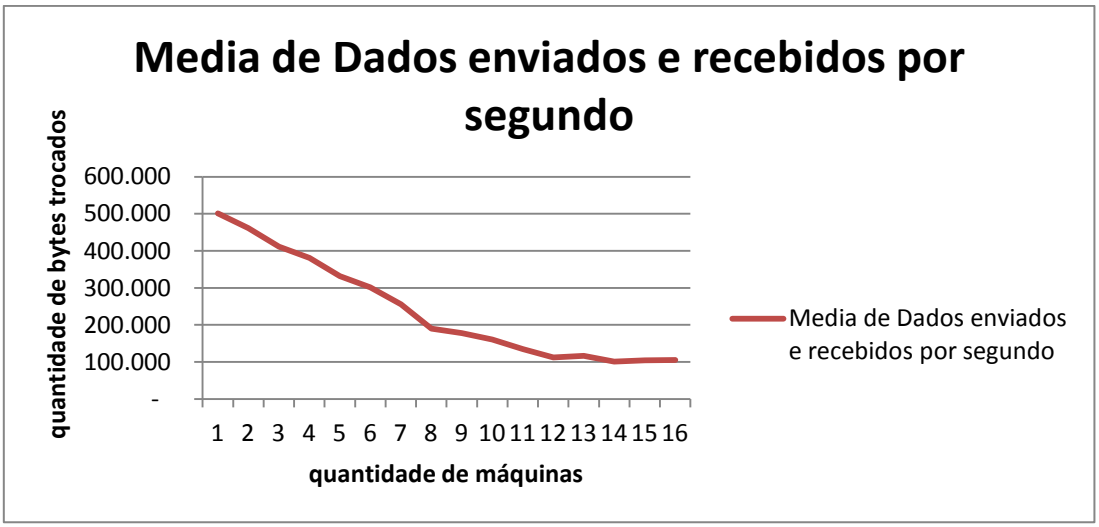
Assim como o gráfico do esforço computacional utilizou a API sigar para capturar os valores da CPU, este último gráfico também a utiliza.

Os experimentos realizados mostraram que pode existir para qualquer tipo de simulação que utilize HLA, neste caso integrado ao Ptolemy, um número limite de máquinas a serem utilizadas no projeto de simulação, dependendo da quantidade de atores envolvidos para cada caso. Como os experimentos apresentaram semelhança de comportamento, então tentou-se mostrar o porque de tal comportamento. Uma das hipóteses que foram testadas para tentar entendê-lo é a ociosidade das CPUs após adicionar mais do que a quantidade necessária de máquinas. O Gráfico 6 é uma tentativa de validar a hipótese levantada com relação a instabilidade dos experimentos (gráfico 1 ao 5) após o uso de determinada quantidade de máquinas. Por outro lado, descobrir a quantidade de máquinas a serem utilizadas de forma prática e automática para cada situação pode ser um desafio que não está no escopo desta dissertação.

Uma última análise que ainda pode ser feita é a quantidade de bytes enviados e recebidos durante a simulação por segundo, que no caso, essa quantidade diminui

proporcionalmente com o aumento do número de máquinas. Porém, nesse caso, a curva tende a variar menos somente a partir do caso 14 e não do caso 8 como os demais gráficos. Essa curva mostra que quanto menos pessoas têm em cada máquina (ex. se utilizar 4 máquinas, então terá 1000 pessoas para cada máquina. Por outro lado, se usar 8 máquinas, então existirá 500 pessoas para cada máquina. A fórmula utilizada é 4000/X, em que X é a quantidade de máquinas envolvidas), ou seja, quanto mais máquinas envolvidas na simulação, menor é a quantidade de pessoas detectadas em cada máquina, e assim menor é a quantidade de bytes enviados e recebidos na rede. O que pode ser que aumente na rede é a quantidade de *NULL messages*, de acordo com Chandy e Misra [82], ou seja, o que pode aumentar é a quantidade de pacotes vazios. Esse pode ser um dos motivos da diminuição da quantidade de bytes enviados e recebidos.

Gráfico 7.Experimento 4000 pessoas - análise dos dados transmitidos



6.2.4. Consistência dos resultados

É interessante notar que, se o resultado da simulação entre os dois modelos distintos, um com HLA e outro sem, forem diferentes, então não faz sentido algum realizar a análise do desempenho das simulações. Só é válido a comparação entre os dois modelos distintos quando, mesmo que se separe o modelo em máquinas distintas, apresentam os resultados iguais, ou pelo menos bastante semelhantes.

Para isso, foi realizado alguns experimentos utilizando os mesmos cenários expostos anteriormente, porém com menos quantidade de atores envolvidos, e utilizando o caso 4 como referência, ou seja, dividindo o modelo em 4 partes iguais em 4 máquinas distintas, onde cada máquina possui um conjunto de sensor. Nas tabelas a

seguir, sensor 1, significa o primeiro conjunto de sensor que contém 4 sensores mais o sensor central. Tais sensores capturam as pessoas que passam ao seu redor, sendo identificadas pelo id da pessoa, que são os números que são exibidos nas tabelas.

O primeiro experimento teve as seguintes configurações: *Lookahead* = 10; Tempo de simulação = 100; Quantidade de pessoas = 40.

Nessas configurações os resultados foram exatamente iguais nos dois modelos distintos (com e sem HLA), como pode ser visto nas tabelas a seguir.

Tabela 9. Verificando a consistência do modelo com 40 pessoas - sensor 1 e 2

Sensor 1 - Sem HLA	Sensor 1 - Com HLA	Sensor 2 - Sem HLA	Sensor 2 - Com HLA
1	1	11	11
2	2	12	12
3	3	13	13
4	4	14	14
5	5	15	15
6	6	16	16
7	7	17	17
8	8	18	18
9	9	19	19
10	10	20	20

Tabela 10. Verificando a consistência do modelo com 40 pessoas - sensor 3 e 4

Sensor 3 - Sem HLA	Sensor 3 - Com HLA	Sensor 4 - Sem HLA	Sensor 4 - Com HLA
21	21	31	31
22	22	32	32
23	23	33	33
24	24	34	34
25	25	35	35
26	26	36	36
27	27	37	37
28	28	38	38
29	29	39	39
30	30	40	40

O segundo experimento teve as seguintes configurações:*Lookeahead* = 10; Tempo de simulação = 100; Quantidade de pessoas = 200. Nessa configuração, os resultados foram quase iguais em todos os sensores. Porém quando há diferença, esta é mínima. O sensor dois, como pode ser visto na Tabela 11 apresentou resultados diferentes. Com o HLA, os sensores conseguiram detectar duas pessoas a mais do que o modelo sem HLA, no caso a pessoa com o id 68 e 92. Isso pode se dá devido ao modelo HLA possuir menos “colisão” no envio das mensagens, já que no modelo com HLA, os escravos estão em máquinas distintas, utilizando processadores distintos, o que pode dar

mais precisão. O fato é que, os resultados de uma forma geral estão bem consistentes, e para o propósito desta aplicação, as não consistências não geram problema algum.

Tabela 11. Verificando a consistência do modelo com 200 pessoas - sensor 1 e 2

Sensor 1 - Sem HLA	Sensor 1 - Com HLA		Sensor 2 - Sem HLA	Sensor 2 - Com HLA
1	1		51	51
2	2		52	52
3	3		53	53
4	4		54	54
5	5		55	55
6	6		56	56
7	7		57	57
8	8		58	58
9	9		59	59
10	10		60	60
11	11		61	61
12	12		62	62
13	13		63	63
14	14		64	64
15	15		65	65
16	16		66	66
17	17		67	67
18	18		69	*68*
19	19		70	69
20	20		71	70
21	21		72	71
22	22		73	72
23	23		74	73
24	24		75	74
25	25		76	75
26	26		77	76
27	27		78	77
28	28		79	78
29	29		80	79
30	30		81	80
31	31		82	81
32	32		83	82
33	33		84	83
34	34		85	84
35	35		86	85
36	36		87	86
37	37		88	87
38	38		89	88
39	39		90	89
40	40		91	90
41	41		93	91
42	42		94	*92*
43	43		95	93
44	44		96	94
45	45		97	95
46	46		98	96
47	47		99	97
48	48		100	98
49	49			99
50	50			100

Tabela 12. Verificando a consistência do modelo com 200 pessoas - sensor 3 e 4

Sensor 3 - Sem HLA	Sensor 3 - Com HLA		Sensor 4 - Sem HLA	Sensor 4 - Com HLA
101	101		151	151
102	102		152	152
103	103		153	153
104	104		154	154
105	105		155	155
106	106		156	156
107	107		157	157
108	108		158	158
109	109		159	159
110	110		160	160
111	111		161	161
112	112		162	162
113	113		163	163
114	114		164	164
115	115		165	165
116	116		166	166
117	117		167	167
118	118		168	168
119	119		169	169
120	120		170	170
121	121		171	171
122	122		172	172
123	123		173	173
124	124		174	174
125	125		175	175
126	126		176	176
127	127		177	177
128	128		178	178
129	129		179	179
130	130		180	180
131	131		181	181
132	132		182	182
133	133		183	183
134	134		184	184
135	135		185	185
136	136		186	186
137	137		187	187
138	138		188	188
139	139		189	189
140	140		190	190
141	141		191	191
142	142		192	192
143	143		193	193
144	144		194	194
145	145		195	195
146	146		196	196
147	147		197	197
148	148		198	198
149	149		199	199
150	150		200	200

6.3. Publicações aceitas e submetidas

Como resultado deste trabalho, alguns trabalhos foram escritos e submetidos para congressos e anais de eventos. Segue a lista:

- Negreiros, A. L. V., Brito, A. V. "The Development of a Methodology with a Tool Support to the Distributed Simulation of Heterogeneous and Complexes Embedded Systems". *In: SBESC 2012 - Simpósio Brasileiro de Engenharia de Sistemas Computacionais. III Workshop de Sistemas Embarcados (WSE)*. 2012. Aceito e publicado.
- Negreiros, A. L. V., Brito, A. V. "Development and Evaluation of Distributed Simulation of Embedded Systems using Ptolemy and HLA". *In: IEEE Computer Society Symposium on VLSI 2013*. 2013. Artigo enviado e não aceito.
- Negreiros, A. L. V., Brito, A. V. "Análise da Aplicação de Simulação Distribuída no Projeto de Sistemas Embarcados". *In: SBSI – 2013. Simpósio Brasileiro de Sistemas de Informação. VI Workshop de Teses e Dissertações em Sistemas de Informação*. 2013. Aceito e publicado.
- Negreiros, A. L. V., Brito, A. V., Christoph R., Oliver S. "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*. 2013. Aceito e publicado.
- Negreiros, A. L. V., Brito, A. V., Christoph R., Oliver S. "Large-Scale Systems evaluation with Ptolemy through Distributed Simulation". *In: SBESC 2013 - Simpósio Brasileiro de Engenharia de Sistemas Computacionais. IV Workshop de Sistemas Embarcados (WSE)*. 2013. Aceito e publicado

Capítulo

7

Considerações Finais

Um dos objetivos deste trabalho de dissertação é realizar a integração de duas ferramentas amplamente utilizadas, o Ptolemy e o HLA, para possibilitar que simulações heterogêneas, de ambientes complexos sejam executadas em larga escala. Além disso, o principal objetivo é encontrar alguma indicação que o uso de simulação distribuída com o Ptolemy para simular redes de sensores sem fio (ou outros projetos quaisquer) traga vantagens em termos de desempenho. Como primeiro resultado, foi demonstrada a viabilidade de integrar o HLA com o Ptolemy e depois que é possível ter benefícios utilizando tal integração. Foi exposto, por exemplo, que o *speedup* de fator 4 foi adquirido quando o modelo com 4 mil atores foram distribuídos em 8 diferentes máquinas (em um experimento que utilizou até 16 máquinas), além de diversos outros resultados. Mostrou-se também que o uso do HLA apresenta limitações, em que utilizando mais máquinas do que o necessário pode acarretar em perda de desempenho. Duas hipóteses foram levantadas para justificar tal comportamento, em que a primeira hipótese considera o uso das configurações do *lookahead* como fator fundamental no algoritmo conservativo do HLA para aprimorar ou deteriorar o desempenho da simulação. Esta primeira hipótese não foi bastante aprofundada neste trabalho, já que alguns experimentos foram realizados e resultados semelhantes foram obtidos mesmo com a mudança das configurações do *lookahead* da simulação. Já a segunda hipótese, foi a mais detalhada neste trabalho, em que esta considera que a simulação apresenta uma quantidade limite de máquinas a serem utilizadas para cada caso específico, e que o uso de mais máquinas do que o necessário, as deixa ociosa, havendo a necessidade do HLA lidar com a sincronização de mais máquina sem ser necessário.

Esses resultados abrem uma nova alternativa de modelar e simular sistemas embarcados em larga escala, utilizando o Ptolemy, para lidar com sistemas complexos e

heterogêneos, isto é, grandes redes de sensores sem fio, e o HLA para auxiliar e viabilizar diversos experimentos.

7.1. Trabalhos Futuros

Como trabalho futuro pretende-se realizar mais variações entre as variáveis que estão envolvidas na simulação, como por exemplo, o *lookeahead*, o *timeWindow* e número de atores envolvidos. Nesse caso, pretende-se investigar principalmente o impacto de diferentes tempos de *lookahead* no desempenho da simulação para vários casos específicos. E ainda, pode-se analisar outro fator como resultado, além do uso da rede e tempo de execução, como por exemplo, o uso da memória. Os modelos já estão configurados e adaptados para possibilitar a análise desses fatores adicionais. Para isso, utiliza-se a API Sigar no intuito de fazer a captura do uso de memória das máquinas durante a simulação. Outra análise interessante a se fazer é avaliar internamente a implementação da integração do HLA com o Ptolemy e verificar se fazendo melhorias no código com técnicas de engenharia de software (algoritmo de sincronização, de integração), melhora também o desempenho da simulação. Além disso, pode-se também tentar descobrir de forma automática para cada cenário e para cada situação qual seria o número ideal de máquinas para lidar com aquele problema em específico.

Para embasar melhor o trabalho, pode-se também utilizar técnicas estatísticas, se for o caso, como por exemplo, Projeto Fatorial 2^2. Com isso pode-se dizer com precisão se o número de experimentos e amostras coletadas está suficiente, especialmente no caso de coleta dos dados referente ao uso de CPU e memória.

Anexos

Anexo 1

Ambiente de simulação de pessoas caminhando em direção a um estádio de futebol.

Figura 38. Objetos pessoa prontos para iniciar a simulação

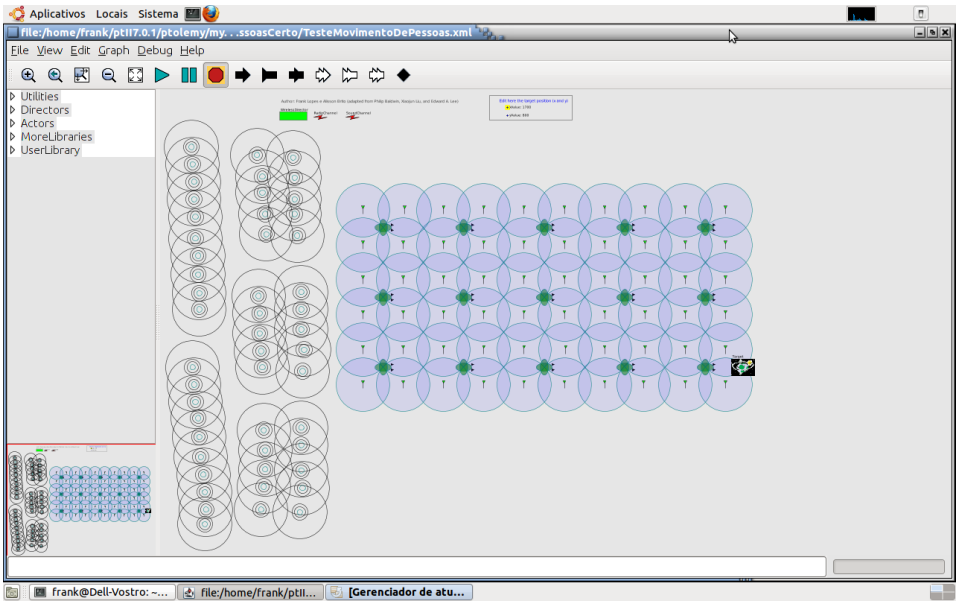


Figura 39. Ambiente de simulação das pessoas chegando ao estádio

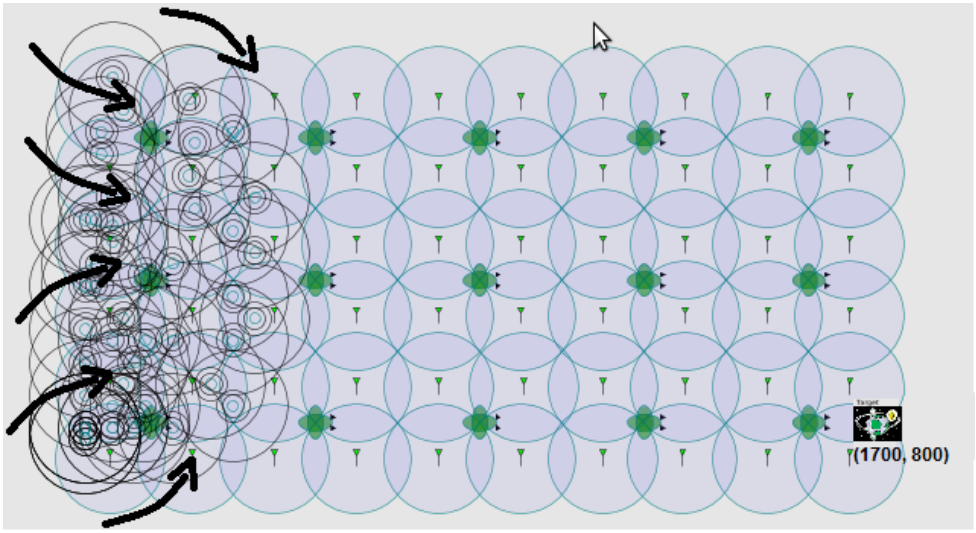
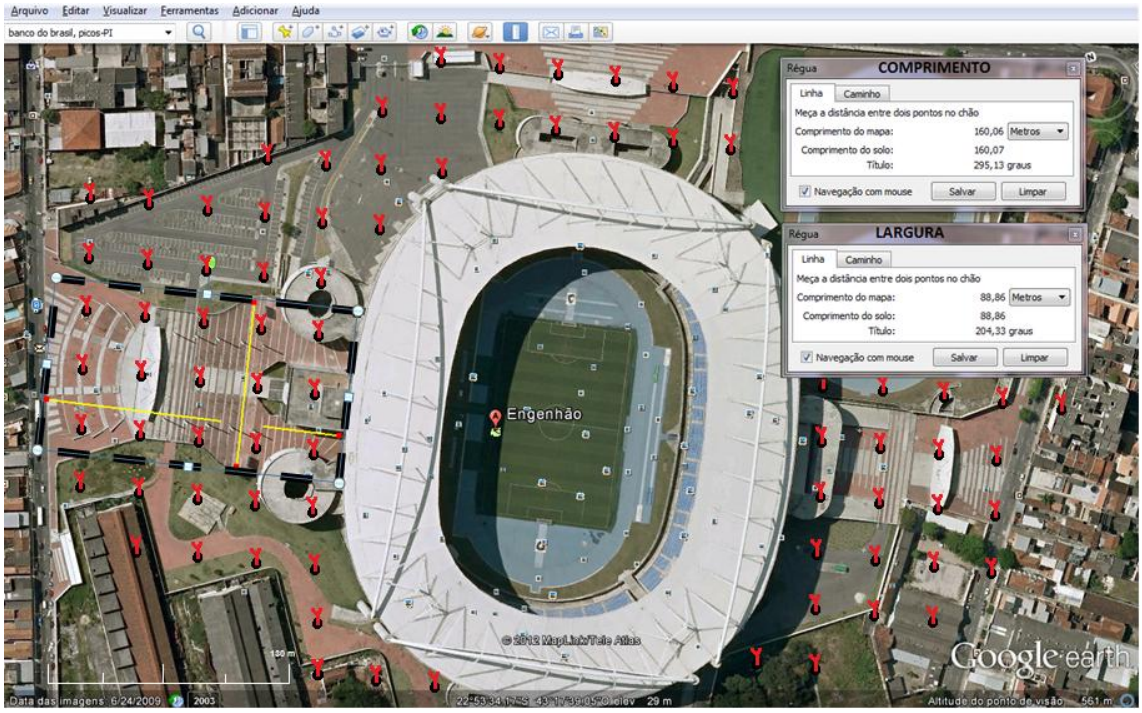


Figura 40. Área de localização da RSSF



Referências

[1] Cuadrado, D. L. “Automated Distributed Simulation in Ptolemy II”. *In: Parallel and Distributed Computing and Networks*. 2007

[2] Lee, E. A., S. A. Seshia, “Introduction to Embedded Systems, A Cyber-Physical Systems Approach”. ISBN 978-0-557-70857-4, 2011.

[3] Hoare, C. A. R. “A theory of CSP”. *In: Communication ACM, vol. 21, no. 8*. Aug. 1978.

[4] Liu, J. “Continuous time and mixed-signal simulation in Ptolemy II,” Univ. California Elec. Eng. Comput. Sci., Berkeley, CA, Memo M98/74, UCB/ERL, 1998.

[5] Kahn, G., MacQueen, D. B. “Coroutines and networks of parallel processes”, *In: Proc. IFIP Cong. 77, 1977, pp. 993–998*.

[6] Edwards, S. A., Lee, E. A.. “The semantics and execution of a synchronous block-diagram language”. *In: Sci. Comput. Programm. [Online]*. 48(1), pp. 21–42. Available: http://dx.doi.org/10.1016/ S0167-6423(02)00096-5. 2003

[7] Lee, E. A., D. G. Messerschmitt, “Synchronous data flow,” Proc. IEEE, vol. 75, pp. 1235–1245, Sept. 1987.

[8] Savage, J. E. “Models of Computation: Exploring the Power of Computing” (1st ed.). *In: Addison-Wesley Longman Publishing Co., Inc*., Boston, MA, USA. 1997.

[9] Artist Embedded Systems Design. The ARTIST Roadmap for Research and Development Lecture Notes in Computer Science, Vol 3436. Disponível em: http://www.artist-embedded.org. Acesso em: 25 de junho de 2013.

[10] Robinson, S. “Simulation - The practice of model development and use”. Wiley 2004.

[11] Sun, W. *et al*. “Range-based Localization for Estimating Pedestrian Trajectory in Intersection with Roadside Anchors”. *In: IEEE Vehicular Networking Conference*. 2009.

[12] Lee, D., Kim, S.; Kim, H.; Park, N. “Mobile Platform for Networked RFID Applications”. *In: IEEE Seventh International Conference on Information Technology (ITNG)*, 2010

[13] Lee, E. A., “The Problem with Threads”, Computer, vol. 39, no. 5, pp. 33-42 , doi:10.1109/MC.2006.180. May 2006

[14] Wetter, M., Haves, P. “A Modular Building Controls Virtual Test Bed for the Integration of Heterogeneous Systems”. *In: Third National Conference of IBSPA-USA (SimBuild 2008)*. Berkeley, California, July 30th to August 1st, 2008.

[15] Sung, C., Kim, T. G. “Framework for Simulation of Hybrid Systems: Interoperation of Discrete Event and Continuous Simulators Using HLA/RTI”. *In: Principles of Advanced and Distributed Simulation (PADS)*. 2011.

[16] Sung, C. H., Hong, J. H., Kim, T. G. “Interoperation of DEVS models and differential equation models using HLA/RTI: hybrid simulation of engineering and engagement level models”. *In: Proceedings of the 2009 Spring Simulation Multiconference (SpringSim '09). Society for Computer Simulation International*, San Diego, CA, USA, Article 150 , 6 pages. 2009

[17] Bononi, L., Felice, M. D., Bertini, M., Croci, E. "Parallel and Distributed Simulation of Wireless Vehicular Ad Hoc Networks". In: MSWiM'06. Spain. 2006

[18] Guha, R., Lee, J., Kachirski, O. “Evaluating Perfomance Of Distributed Computing Technologies - HLA and TSPACE on a Cluster Computer”. *In: 19th European Conference on Modelling and Simulation*. 2005

[19] Lasnier, G., Cardoso, J., Siron, P., Pagetti, C., Derler, P. “Distributed Simulation Of Heterogeneous and Real-time Systems”. *In: 17th IEEE/ACM Internacional Symposium on Distributed Simulation and Real Time Applications.* 2013.

[20] CERTI RTI. Disponível em: http://savannah.nongnu.org/projects/certi, 2011. Acesso em: 26 de junho de 2013.

[21] Goel, M. “Process Networks in Ptolemy II”, University of California at Berkeley, 2001

[22] Brito, A. V., Negreiros, A. L. V., Roth, C., Sander, O. “Development and Evaluation of Distributed Simulation of Embedded Systems using Ptolemy and HLA”. *In: 17th IEEE/ACM Internacional Symposium on Distributed Simulation and Real Time Applications.* 2013.

[23] Negreiros, A. L. V., Brito, A. V. “The Development of a Methodology with a Tool Support to the Distributed Simulation of Heterogeneous and Complexes Embedded Systems”. *In: SBESC 2012 - Simpósio Brasileiro de Engenharia de Sistemas Computacionais. III Workshop de Sistemas Embarcados (WSE)*. 2012.

[24] Lee, E. A. “Overview of the Ptolemy Project”. *In: Technical Memorandum UCB/ERL M01/11, University of California, Berkeley, EUA*. Março de 2001. Disponível em http://Ptolemy.eecs.berkeley.edu. Acesso em: 02 jul. 2012.

[25] Eker, J., Janneck, J. W., Lee, E. A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y., “Taming Heterogeneity-the Ptolemy Approach,” Proceedings of the IEEE, 91(2), January, 2003.

[26] Girault, A., Lee, B. and Lee, E. A., “Hierarchical Finite State Machines with Multiple Concurrency Models,” *In: IEEE Transactions On Computer-aided Design Of Integrated Circuits And Systems*, 18(6), June 1999.

[27] Xiong, Y. “An Extensible Type System for Component-Based Design” *In: Technical Memorandum UCB/ERL M02/13*, University of California, Berkeley, CA 94720, May 1, 2002.

[28] “System Design, Modeling and Simulation”. Editor: Claudius Ptolemaeus. UC Berkley. 2011

[29] Eker, J., Janneck, J. W., Lee, E. A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y., “Taming Heterogeneity-The Ptolemy Approach”. *In: Proceedings Of The IEEE, V.91, No. 2*, January 2003.

[30] Jantsch A., Sander, I. “Models of computation and languages for embedded system design”. *In: IEE Proceedings online no. 20045098*. 2005

[31] Edwards, S., Lavagno, L., Lee, E.A., and Sangiovanni-Vincentelli, A. “Design of embedded systems: formal models, validation, and synthesis”. *In: Proc. IEEE, 1997, 85*, (3), pp. 366–390. 1997.

[32] Jantsch, A. “Modeling embedded systems and SoCs - concurrency and time in models of computation: Systems on silicon”. *In: Morgan Kaufmann Publishers*, June 2003.

[33] Sander, I., Jantsch, A.: “System modeling and transformational design refinement in ForSyDe”. *In: IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2004, 23, (1), pp. 17–32

[34] Jones, S. P. “Haskell 98 language and libraries”. *In: Cambridge University Press*, 2003

[35] Lavagno, L., Vincentelli, A. S., Sentovich, E. “Models of Computation for Embedded System Design”. 1998

[36] Bruni, T. “Heterogeneous Use of Models of Computation”. 2006

[37] Jantsch, A. “Models of embedded computation”. *In: Embedded systems (CRC Press, 2004)*. 2004

[38] Dabney, J., Harman, T. L. “*Master*ing SIMULINK 2”. *In: Prentice Hall*, 1998.

[39] Elmqvist, H., Mattsson, S. E., Otter, M. “Modelica - the new object-oriented modeling language”. *In: Proceedings of the 12th European Simulation Multiconference*, June 1998.

[40] Lee, E. A. “Heterogeneous Simulation - Mixing Discrete - Event Models with Dataflow”. *In: Journal of VLSI Signal Processing 15, Pg. 127-144*. 1997.

[41] Benveniste, G. Berry, “The Synchronous Approach to Reactive and Real-Time Systems,” *In: Proceedings of the IEEE, Vol. 79, No. 9, 1270-1282*. September, 1991.

[42] Whitaker, P. “The Simulation of Synchronous Reactive Systems In Ptolemy II”, University of California at Berkeley, 2001

[43] Lee, E.A., Parks, T.M. “Dataflow process networks”. *In: Proc. IEEE,1995, 83, (5)*, pp. 773–799

[44] Murthy, P. K., Lee, E. A., “Multidimensional Synchronous Dataflow”. *In: IEEE Transactions On Signal Processing*, Vol. 50, N. 7, July, 2002

[45] Dijk, H. W. V., Sips, H. J. “Context - Aware Process Networks”, *In: International Journal Of Embedded System*, February, 2005

[46] Thies, W., Karczmarek, M., Amarasinghe, S. “StreamIt: A language for streaming applications”. *In: 11th Intl. Conf. on Compiler Construction*, CC’02, volume LNCS 2304. Springer, 2002.

[47] Tripakis, S., Bui, D., Geilen, M., Rodiers, B., Lee, E. A., “Compositionality in Synchronous Data Flow: Modular Code Generation from Hierarchical SDF Graphs”, Center for Hybrid and Embedded *Software* Systems (CHESS) at UC Berkeley,2000.

[48] Fujimoto, R. “Parallel and Distributed Simulation Systems”. *In: Wiley & Sons*. 2000.

[49] D'Angelo, G. “Parallel and Distributed Simulation from Many Cores to the Public Cloud”. *In: arXiv:1105.2301v3*. 2011

[50] Law, A. M., Kelton, D. M. “Simulation Modeling and Analysis”. *In: McGraw-Hill Higher Education*, 3rd edition, 1999.

[51] Egea-Lopez, E., Vales-Alonso, J., Martinez-Sala, A., Pavon-Mario, P., Garcia-Haro, J. “Simulation scalability issues in wireless sensor networks”. *In: Communications Magazine*, IEEE, 44(7):64 – 73, july 2006.

[52] Fujimoto, R. M. “Parallel discrete event simulation”. *In: Proceedings of the 21st conference on Winter simulation*, WSC ’89, pages 19–28, New York, NY, USA, 1989. ACM.

[53] Lamport, L., “Time, clocks, and the ordering of events in a distributed system”. In: Commun. ACM, 21:558–565, July 1978.

[54] Hill, M. D., Marty, M. R. “Amdahl’s Law in the Multicore Era”. 2007

[55] Perumalla, K. “Tutorial: Parallel and distributed simulation: Traditional techniques and recent advances”. 2007.

[56] Fujiimoto, R. M. “Parallel and Distributed Simulation Systems” *In: Wiley Series on Parallel and Distributed Computing*. 2000

[57] Zeng, X., Bagrodia, R. , Gerla, M. “Glomosim: a library for parallel simulation of large-scale wireless networks”. *In: SIGSIM Simul. Dig., 28(1):154–161*, 1998.

[58] Jun, Y., Raczy, C., Tan, G. “Evaluation of a sort-based matching algorithm for ddm”. *In: Proceedings of the sixteenth workshop on Parallel and distributed simulation, PADS* ’02, pages 68–75, Washington, DC, USA, 2002. IEEE Computer Society.

[59] Tay, S., Tan, G., Shenoy, K. “Piggy-backed time-stepped simulation with ’super-stepping’”. *In: Simulation Conference, 2003. Proceedings of the 2003 Winter*, volume 2, pages 1077 – 1085 vol.2, dec. 2003.

[60] Eick S. G., Greenberg A. G., Lubachevsky B.D.,Weiss, A. “Synchronous Relaxation for Par-allel Simulations with Applications to Circuit-Switched Networks” *In: ACM Transactions on Modelingand Computer Simulation*, Volume 3(4), pp. 287-314. 1993

[61] Misra, J. “Distributed Discrete-Event Simulation”. In*: Computing Surveys, Vol. 18,* No.1, pp. 39-56, March 1986

[62] Donath, U., Gruschwitz, R., Haase, J., Kurth, G., Schwarz, P. “Parallel Multi-Level Simulation with a Conservative Approach”. 1995

[63] Jefferson, D. “Virtual time.” *In: ACM Transactions Program. Lang. Sys*t., 7(3):404–425, 1985.

[64] Quaglia, F., Santoro, A. “Nonblocking checkpointing for optimistic parallel simulation: description and an implementation.” *In: Parallel and Distributed Systems*, IEEE Transactions on, 14(6):593 – 610, june 2003.

[65] Raytheon RTI NG Pro. Disponível em: http://www.raytheon.com/capabilities/products/rti/, 2011. Acesso em: 26 de junho de 2013.

[66] FDK - Federated Simulations Development Kit. Disponível em: http://www.cc.gatech.edu/computing/pads/fdk/, 2011. Acesso em: 26 de junho de 2013.

[67] MAK Technologies. Disponível em: http://www.mak.com/products/rti.php, 2011. Acesso em: 26 de junho de 2013.

[68] Pitch Technologies. Disponível em: http://www.pitch.se/products/prti, 2011. Acesso em: 26 de junho de 2013.

[69] OpenSkies - Cybernet. Disponível em: http://www.openskies.net/features/features.html, 2011. Acesso em: 26 de junho de 2013.

[70] Magnetar Games - Chronos. Disponível em: http://www.magnetargames.com/, 2011. Acesso em: 26 de junho de 2013.

[71] Portico Project. Disponível em: http://www.porticoproject.org/, 2011. Acesso em: 26 de junho de 2013.

[72] IEEE Standard No.1516-2000, "IEEE Standard for Modeling and Simulation (M&S) *High Level Architecture* (HLA) - Framework and Rules"

[73] IEEE Standard No 1516.1-2000, "IEEE Standard for Modeling and Simulation (M&S) *High Level Architecture* (HLA) - Federate Interface Specification"

[74] IEEE Standard No 1516.2-2000 "IEEE Standard for Modeling and Simulation (M&S) *High Level Architecture* (HLA) - Object Model Template (OMT) Specification"

[75] US Department of Defense, "*High Level Architecture* Interface Specification", Version 1.3, 2. April 1998.

[76] Kuhl, F., Weatherly, R., Dahmann, J. "Creating Computer Simulation Systems : An Introduction to the *High Level Architecture*". *Published by: Prentice Hall PTR*. 1999

[77] Shaw, M., Garlan, D. "*Software* Architecture: Perspectives on an Emerging Discipline". *Published by: Prentice Hall PTR*. 1996

[78] Chaudron, J. B., Noulard, E., Stron, P. "Design and modeling techniques for real-time RTI time management". *In: Onera / Dtim.*

[79] Wilson, A. L., Weatherly, R. M. "The aggregate level simulation protocol: an evolving system". *In: WSC '94: Proceedings of the 26th conference on Winter simulation*. 1994.

[80] Fujimoto, R. M. "Time Management in the *High Level Architecture*". *In: Simulation 71, pp 388-400*, December 1998.

[81] Defense Modeling and Simulation Office DMSO: "*High Level Architecture* Interface Specification", Version 1.3 NG, Washington D.C., 1998.

[82] Chandy, K. M., Misra, J. "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs". *In: Software Engineering, IEEE Transactions*. 1979.

[83] Amdahl, G. M. “Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities”. *In: AFIPS Conference Proceedings*, pages 483–485, 1967.

[84] Véras, F. C. L. "Modelagem e Simulação do Deslocamento de Pessoas para Estimativa de Formação de Grupos". *Dissertação Mestrado*. 2013

[85] Wireshark. Disponível em: http://www.wireshark.org/ . Acesso em: 18 de junho de 2013.

[86] Sigar API. Disponível em: http://www.hyperic.com/products/sigar . Acesso em: 20 de junho de 2013.