UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE CIÊNCIAS EXATAS E DA NATUREZA

DEPARTAMENTO DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Simulação baseada em atores como ferramenta de ensino de organização e arquitetura de computadores

André Luís de Lucena Torres

João Pessoa - PB

Março de 2012

André Luís de Lucena Torres

Simulação baseada em atores como ferramenta de ensino de organização e arquitetura de computadores

Dissertação de Mestrado submetida ao curso de Pós-Graduação em Informática da Universidade Federal da Paraíba como requisito parcial para obtenção do título de Mestre em Informática.

Orientador: Prof. Dr. Alisson V. Brito

João Pessoa - PB
Março de 2012

T693s Torres, André Luís de Lucena.

Simulação baseada em atores como ferramenta de ensino de organização e arquitetura de computadores / André Luís de Lucena Torres.- João Pessoa, 2012.

112f. : il.

Orientador: Alisson V. Brito

Dissertação (Mestrado) – UFPB/CCEN

1. Informática. 2. Arquitetura de computadores. 3. Ensino. 4. Simulação. 5. Ptolemy.

ADICIONAR ATA DA DEFESA DE DISSERTAÇÃO DE MESTRADO

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de André Luís de Lucena Torres, candidato ao Título de Mestre em Informática na Área de Sistemas de Computação, realizada em 01 de março de 2012.

Ao dia primeiro do mês de março do ano dois mil e doze, às quatorze horas, no Auditório do CCEN - da Universidade Federal da Paraíba, reuniram-se os membros da Banca Examinadora constituída para examinar o candidato ao grau de Mestre em Informática, na área de "*Sistemas de Computação*", na linha de pesquisa "*Sinais, Sistemas Digitais e Gráficos*", o Sr. André Luís de Lucena Torres. A comissão examinadora foi composta pelos professores doutores: Alisson Vasconcelos de Brito (DCE-UFPB), Orientador e Presidente da Banca Examinadora, e Alexandre Nóbrega Duarte (DI-UFPB), como examinadores internos e Elmar Uwe Kurt Melcher (COPELE-UFCG), como examinador externo. Dando início aos trabalhos, o Prof. Alisson Vasconcelos de Brito, cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse, oralmente, a exposição do trabalho de dissertação intitulado "SIMULAÇÃO BASEADA EM ATORES COMO FERRAMENTA DE ENSINO DE ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES". Concluída a exposição, o candidato foi argüido pela Banca Examinadora que emitiu o seguinte parecer: "*aprovado*". Assim sendo, deve a Universidade Federal da Paraíba expedir o respectivo diploma de Mestre em Informática na forma da lei e, para constar, eu, professora Tatiana Aires Tavares, Coordenadora deste Programa, servindo de secretária lavrei a presente ata que vai assinada por mim mesmo e pelos membros da Banca Examinadora. João Pessoa, 01 de março de 2012.

Tatiana Aires Tavares

Prof. Dr. Alisson Vasconcelos de Brito
Orientador (DCE-UFPB)

Prof. Dr. Alexandre Nóbrega Duarte
Examinador Interno (DI-UFPB)

Prof. Dr. Elmar Uwe Kurt Melcher
Examinador Externo (UFCG)

AGRADECIMENTOS

Agradeço primeiramente a Deus, pelo dom da vida, saúde, paz e pela capacidade que me deu. A realização deste trabalho não teria sido possível sem a força que encontrei Nele durante esse tempo.

Agradeço aos meus pais, Ramilson e Maria José, por tudo que fizeram e fazem por mim, toda dedicação, amor, carinho, paciência, pelos bons exemplos que são para mim, pelo desprendimento nos investimentos que fazem em mim, e por serem os melhores pais do mundo. Amo muito vocês! Agradeço também aos meus irmãos, Tatiana e Romeu, com quem eu sei que posso contar para qualquer coisa.

Agradeço à minha namorada Dayse, que me compreendeu e me ajudou nos momentos mais difíceis desse mestrado, que me ajuda a ser cada dia melhor, em todos os sentidos. Nela encontro muita paz, amor, compreensão e tudo mais que procuro numa pessoa com a qual pretendo compartilhar todos os momentos da minha vida.

Agradeço ao meu orientador Prof. Alisson V. Brito, por todos os direcionamentos necessários à conclusão desse trabalho com êxito, pelos ensinamentos passados ao longo desse período, pela sua amizade e confiança. Agradeço também a todo corpo docente do PPGI, pela melhoria contínua que vêm propiciando a esse curso.

Agradeço aos amigos que fiz nesse mestrado, Carlos, Judson e Ramon. Essas pessoas contribuíram muito para dias mais alegres no laboratório, bem como com inúmeras discussões que me ajudaram na execução deste trabalho.

Agradeço aos meus amigos de todos os dias e de toda a vida, Josias, Paulo Márcio, Anderson, Sávio, Pablo, Jether, Caetano, Luciana, Yane e muitos outros mais, que apesar de distância de alguns, sei que posso contar e levá-los-ei sempre no coração.

Agradeço a toda minha família, em especial aos meus padrinhos, os quais serei eternamente grato por todo amor que me deram nos primeiros anos de vida. E em especial a Dona Jackeline, uma pessoa que entrou na minha vida para iluminar o meu caminho.

Resumo

A informática educativa se faz cada vez mais presente nas atividades pedagógicas. Nesta nova realidade, várias aplicações visam facilitar a construção do conhecimento por parte dos professores em relação aos alunos através de métodos dinâmicos, expondo aulas para múltiplos ramos sem haver grande esforço ou repetições desnecessárias. Na área da Computação, existe a necessidade de utilização de aplicações que facilitem a aprendizagem. Pois, se tem observado que os ensinos de alguns conceitos introdutórios em disciplinas essenciais costumam apresentar um nível de abstração que prejudica o aprendizado dos alunos de cursos de informática que já possuem uma grande dificuldade em lidar com disciplinas da área de hardware. A utilização de simuladores na educação se faz cada vez mais presente nas atividades pedagógicas. Neste sentido, este trabalho apresenta os resultados alcançados com a aplicação de uma extensão desenvolvida numa ferramenta de modelagem e simulação de sistemas concorrente baseada em atores, denominada Ptolemy. A extensão foi criada para contribuir com o processo de ensino-aprendizagem da disciplina de Organização e Arquitetura de Computadores com alunos da graduação.

Palavras-chave: Arquitetura de Computadores, Simulação, Ensino, Ptolemy.

Abstract

The educative informatics has become more present in pedagogical activities. On this new reality, many applications tend to make the knowledge construction a easier tool from the teachers to the students by dynamic methods, exposing multi-branch subjects with no great efforts or unnecessary repetitions. In computing area, the use of applications that facilitate learning is mandatory. Thus, it has been observed that the teaching of some introductory concepts on essential subject used to present an abstraction level that harms the instruction of students of computing courses that have difficulties on hardware related subjects. The use simulators in education have become more present in pedagogical activities. Thus, this work presents the achieved results of an extension developed on a simulation and modeling tool of concurrent systems based in actors, named Ptolemy. The extension was developed to contribute with the teaching-leaning process in the graduation course of Computers Architecture and Organization.

Keywords: Computer Architecture, Simulation, Education, Ptolemy.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

AA – *Architecture Advanced*
AB – *Arquitetura Básica*
ALU – *Arithmetic Logic Unit*
CI – *Circuitos Integrados*
CI – *Component Interaction*
CL – *Clock Level*
CPI – *Ciclos de clock por instrução*
CSP – *Communicating Sequential Processes*
DDE – *Distributed Discrete Events*
DDF – *Dynamic Data Flow*
DE – *Discrete Events*
DIMIPS – *Didact Interactive MIPS Simulator*
DT – *Discrete Time*
EX – *Execute*
FSM – *Finite State Machine*
ID – *Instruction Decode*
IDE - *Integrated Development Environment*
IF – *Instruction Fetch*
IL – *Level of Instruction*
IR – *registrador de instrução*
MA – *Memory Access*
MARS – *MIPS Assembly and Runtime Simulator*
MIMD – *Multiple Instruction Multiple Data*
MIPS – *Microprocessor without Interlocked Pipe Stages*
MoC – *Model of Computation*
MP – *Memória Principal*
PBL – *Problem Based Learning*
PC – *contador de programa*

PL – *Level Program*
RDM – *Registrador de Leitura*
REM – *Registrador de Escrita*
RISC – *Reduced Instruction Set Computer*
SDF – *Synchronous dataflow*
TC – *Continuous Time*
UCP – *Unidade Central de Processamento*
VHDL – *VHSIC Hardware Description Language*
VLSI – *Very Large Scale Integration*
WB – *Write Back*
XML – *Extensible Markup Language*

1 Introdução

A informática educativa tem tido uma relevância cada vez maior para o ensino e aprendizagem, onde o avanço das novas tecnologias no âmbito educacional é constante. Nesta nova realidade, várias aplicações visam facilitar a transmissão do conhecimento de professores para alunos através de métodos dinâmicos, massificando uma aula para múltiplos ramos sem haver grande esforço ou repetições desnecessárias [Falcão11].

Na área da Computação, existe uma necessidade na utilização de aplicações que facilitem a aprendizagem. Pois se tem observado nos alunos de cursos de informática uma grande dificuldade e indiferença às disciplinas da área de hardware [Sobreira07].

Entre as disciplinas voltadas para hardware, o estudo da disciplina de Arquitetura e Organização de Computadores é um campo desafiador para pesquisadores, professores e alunos. Pois entender como determinada arquitetura de computador funciona é uma tarefa difícil, principalmente embasado apenas na teoria. Para melhorar esse entendimento, são usados os simuladores que reproduzem com perfeição toda a dinâmica de execução das instruções em uma arquitetura de computador.

A visualização é uma palavra-chave necessária e uma idéia para melhorar o nível de compreensão do aluno. Por exemplo, quando o instrutor ensina seus alunos sobre o computador, eles querem usar ferramentas eficazes de ensino. Essas ferramentas devem ter algum tipo de função para simular e visualizar o que é difícil de entender [Imai07].

As técnicas de simulação vêm facilitando cada vez mais o processo de ensino aprendizagem do funcionamento de sistemas reais, como os microprocessadores, onde são usados para reproduzir ou prever o comportamento de tais sistemas, possibilitando o entendimento e previsão de possíveis falhas. Na área da ciência da computação existem

simuladores que auxiliam no ensino de várias disciplinas, como redes de computadores, técnicas de programação, arquitetura de computadores e sistemas operacionais.

1.1 Motivação

Atualmente, diversas ferramentas para o auxilio ao ensino de Arquitetura de Computadores vem sendo desenvolvidas, afim de sanar dificuldades apresentadas pelos alunos de graduação e que apenas tem acesso a teoria. Para diminuir tal dificuldade, a utilização de simuladores é uma abordagem benéfica para que os alunos entendam o conteúdo e possam visualizar a execução de programas para montagem real de uma arquitetura.

A grande vantagem da simulação, como metodologia de ensino/aprendizagem, é o fato de conseguir proporcionar ao aluno, dentro e fora do espaço das salas de aula, uma aproximação muito consistente entre a teoria e a prática [Moure99].

Analisando tais fatos, podemos destacar alguns problemas importantes que foram à motivação inicial desse trabalho, como: i) Falta de motivação dos alunos para o estudo da teoria e a não aplicação dos conhecimentos adquiridos em atividades práticas realísticas [Martins03]; ii) Dificuldade para ensino e principalmente para aprendizado dos conceitos, técnicas relacionadas e exemplos de sistemas computacionais quando se usavam apenas os métodos tradicionais baseados em leitura da teoria e alguns exercícios manuais; iii) Deficiências e inadequação dos métodos didáticos tradicionais usados atualmente no ensino e aprendizado de Arquitetura de Computadores.

A simulação de sistemas concorrentes baseada em atores é um exemplo de metodologia que possui um grande potencial educacional ainda pouco explorado. Onde um modelo de atores é definido como um modelo matemático de computação concorrente que trata “atores” como sendo os elementos primitivos da computação concorrente digital [Lee07].

1.2 Objetivos

Dada as limitações das soluções e propostas existentes, este trabalho propõe uma metodologia para ensino de Organização de Arquitetura de Computadores apoiada pela simulação de sistemas concorrentes baseada em atores pela ferramenta Ptolemy II, desenvolvida pela Universidade de Berkeley. O objetivo geral desse trabalho é prover

um simulador que possibilite um ambiente didático onde os alunos possam criar seus próprios modelos de arquitetura de maneira simples, interativa e didática.

Para que o Ptolemy possa ser utilizado com êxito para fins educacionais alguns aspectos devem ser considerados para que alcancemos o objetivo geral, como:

- Adicionar funcionalidades ao Ptolemy que permitam a elaboração de atividades e correção automática, auxiliando a tarefa do professor.
- Disponibilizar um conjunto de atividades didáticas a serem utilizados pelos professores em sala de aula. Para validar a metodologia criada, a mesma será aplicada com alunos da própria universidade, seguida por avaliações quantitativas e qualitativas.
- Adaptar didaticamente a ferramenta Ptolemy II, propondo melhorias para sua interface, visando torná-la mais amigável, com instruções em português e com modelos simuláveis, melhorando sua aplicação no ensino de Organização e Arquitetura de Computadores.
- Desenvolver atores personalizados que correspondam a diferentes níveis de abstração de hardware, criando assim um repositório de atores específicos para aplicação na disciplina de Organização e Arquitetura de Computadores. Possibilitando a criação de exercícios pelo professor, resolvidos pelos alunos e com o auxilio do Ptolemy verificados e avaliados.

1.3 Contribuições

A disciplina de Arquitetura e Organização de Computadores é fundamental na formação de alunos nos cursos de Ciência da Computação e de Engenharia da Computação, onde tais alunos devem, ao final da disciplina, conhecer conceitos básicos da Arquitetura física de sistemas computacionais. Para tal, os alunos precisam possuir as habilidades básicas para: compreender os modelos existentes, distinguir os conceitos básicos da arquitetura de computadores, além de diferenciá-las.

Com a elaboração deste trabalho, pretende-se oferecer aos alunos, um ambiente que permita a eles a criação dos seus próprios modelos computacionais, além da possibilidade de combinarem novos elementos, tornando-os elementos ativos na

formação do conhecimento [Brito09]. Destacam-se ainda, as principais contribuições resultantes desta dissertação de mestrado:

- Especificar num framework científico e industrial, uma extensão para fins educacionais que simule componentes básicos da disciplina em questão, possibilitando ao professor, demonstrar na prática os temas abordados em sala de aula;
- Utilizar um repositório próprio de atores que representem determinados componentes relacionados ao estudo da disciplina em questão;
- Utilizar uma técnica de verificação automática, com a apresentação de *feedback* aos alunos e contribua para a conclusão de exercícios, além de possibilitar um retorno imediato na correção.

1.4 Metodologia

Este trabalho se baseia no uso do Ptolemy, criado e mantido na Universidade de Berkeley, como uma ferramenta de ensino de Organização e Arquitetura de Computadores. Mas para conseguir êxito em tal objetivo, foram verificadas algumas barreiras para a utilização da ferramenta: i) Ptolemy é um software para fins científicos e industriais com interface pouca amigável para usuários inexperientes. ii) Ptolemy não oferece nenhum tipo de ajuda sobre como o aluno deverá montar seus ambientes. iii) Caso um professor configure um ambiente para que seja trabalhado por seus alunos, ele não terá controle sobre o que os alunos fizeram com o ambiente ou com os atores.

Além destas barreiras, foi realizada uma análise de usabilidade no framework Ptolemy, buscando compreender e apontar as necessidades do software para torná-lo uma ferramenta de ensino eficaz, para auxílio na disciplina de Arquitetura de Computadores, de forma mais simples e mais amigável.

Buscando atender às necessidades do software para torná-lo uma ferramenta apropriada para o ensino de Organização e Arquitetura de Computadores, o trabalho descreve duas melhorias: i) tornar sua interface mais amigável, com instruções em português, e para que instruções passo a passo sejam dadas aos alunos na montagem de seus ambientes na forma de atividades; ii) Verificar automaticamente os exercícios feitos pelos alunos e dar *feedback* sobre o que pode ser melhorado.

De forma a especificar a nova arquitetura do Ptolemy para fins educacionais, foram utilizados padrões de projeto de interface [Koscianski07], estudo de algumas ferramentas educacionais e abordagens para software educativo.

Inicialmente foram desenvolvidos atores para representação dos componentes da arquitetura do processador MIPS como estudo de caso. Com a aplicação da ferramenta almeja-se estender sua aplicação para outras arquiteturas de processadores, memórias, dispositivos de Entrada e Saída etc.

1.5 Organização do Trabalho

O restante deste trabalho está organizado da seguinte forma. O Capítulo 2 introduz os conceitos fundamentais que envolvem o ensino de Arquitetura de Computadores. O Capítulo 3 apresenta os conceitos fundamentais relacionados à utilização de Simuladores, com enfoque no framework Ptolemy II. Posteriormente, o Capítulo 4 comenta a utilização de ferramentas no ensino de Arquitetura de Computadores e na aprendizagem em sala de aula. No Capítulo 5 é apresentado detalhes de implementação e a descrição do estudo de caso, explorando as características e funcionalidades do modelo proposto. No Capítulo 6 é apresentada uma análise dos experimentos e o desenvolvimento de um roteiro didático. Por fim, o Capítulo 7 apresenta as considerações finais, evidenciando as contribuições, limitações e trabalhos futuros.

2 Organização e Arquitetura de Computadores

Neste capítulo são descritos conceitos fundamentais que envolvem o ensino de arquitetura de computadores, apresentando inicialmente as questões que enfatizam o quão importante a disciplina é para a computação e estudantes da área. Em seguida temos uma visão geral do escopo de conteúdo da disciplina, além de relatar alguns problemas que ainda dificultam o aprendizado da disciplina.

2.1 Estudo da disciplina de Organização e Arquitetura de Computadores

Dado o dinamismo acentuado da área de informática, recomenda-se que os cursos da área possuam flexibilidade para avançarem em consonância com a evolução que lhe é inerente. Logo, estudar o funcionamento de um computador e de seu processador é fundamental na formação de alunos de cursos de graduação em Computação [Morandi06].

A disciplina Organização e Arquitetura de Computadores têm esse objetivo, sendo ela fundamental na formação de alunos dos cursos de Ciência da Computação e de Engenharia da Computação, fornecendo os conhecimentos básicos para a compreensão dos conceitos apresentados durante o curso, fornecem subsídios fundamentais para a aprendizagem e compreensão da lógica de programação [Vieira10].

Para um melhor entendimento do escopo da disciplina, podemos dividi-la em:

- Arquitetura: atributos vistos pelo programador (conjunto de instruções, números de bits utilizados pelos dados, mecanismos de entrada e saída, técnicas de endereçamento, etc). Exemplo: Há uma instrução de multiplicação?

- Organização: como as funcionalidades são implementadas (sinais de controle, interfaces, tecnologia de memória, etc). Exemplo: Há uma unidade multiplicadora em hardware ou isso é feito por sucessivas adições?

2.1.1 Disposições gerais da disciplina

De acordo com as Diretrizes Curriculares de Cursos da Área de Computação e Informática [MEC98], “o termo arquitetura de computadores refere-se às características existentes em um projeto de máquina para executar as tarefas escritas em uma linguagem de programação, ou seja, arquitetura de computadores se refere ao estudo das máquinas que executam programas (computadores) [...]”.

Os objetivos da disciplina Arquitetura de Computadores são os de apresentar ao aluno os princípios básicos de projeto e funcionamento de computadores e os princípios básicos de funcionamento dos demais dispositivos eletrônicos que compõem um sistema computacional, de forma que o aluno possa estar apto a projetar novos computadores, fazer uso eficiente de recursos computacionais, organizar sistemas de computação eficientes e confiáveis e entender as implicações da arquitetura de computadores em um sistema computacional.

Nos dias atuais, os programadores que quiserem ganhar cada vez mais desempenho em seus programas devem conhecer mais detalhadamente a organização de computadores. Isso se deve ao fato de que o processamento paralelo está cada vez mais inserido no hardware atual, proporcionando um maior poder computacional. Mas para que esse poder possa ser bem aproveitado, o desenvolvedor de software deve saber lidar com a hierarquia de memória e com o paralelismo. Portanto, para que a tarefa de melhorar o desempenho dos softwares em um computador moderno não seja um martírio, devemos saber responder as seguintes perguntas:

- Como os programas escritos em uma linguagem de alto nível, como C ou Java, são traduzidos para a linguagem de máquina e como o hardware executa os programas resultantes?
- O que determina o desempenho de um programa e como um programador pode melhorar o desempenho?

- Que técnicas podem ser usadas pelos projetistas de hardware para melhorar o desempenho?

2.2 Dificuldades no ensino e aprendizagem

A disciplina de Organização e Arquitetura de Computadores busca fornecer aos alunos os conhecimentos básicos e técnicas de projetos em microeletrônica, capacitando-os na avaliação e comparação entre diferentes arquiteturas, além de capacitá-los a apontar e diagnosticar problemas relacionados ao desempenho de sistemas ou sub-sistemas. Porém, com a evolução da tecnologia o seu ensino tem se tornado uma tarefa difícil.

Em sala de aula o ensino de arquitetura de computadores encontra algumas dificuldades, como:

- Necessidade de ferramentas gráficas para exemplificar e ilustrar os caminhos de dados e de controle quando uma instrução é executada [Felix06];
- Compreensão abstrata dos princípios de funcionamento de microprocessadores ou de sistemas computacionais completos;
- Dificuldade de se exemplificar as diversas estruturas da hierarquia de memória utilizando os métodos convencionais [Coutinho06];
- Cursos de computação com foco maior em desenvolvimento de softwares;
- Recursos didáticos limitados [Wolffe02];

Além desses problemas, há necessidade de pesquisas sobre métodos de aprendizagem para a disciplina e materiais atualizados que correspondam aos ambientes estudados.

No próximo capítulo será abordada a utilização de simuladores na educação, com foco no auxílio à disciplina de arquitetura de computadores, possibilitando amenizar os problemas relatados nesta seção.

2.3 Considerações finais

Vital comenta [Vital10] que a disciplina de Organização e Arquitetura de Computadores tende a apresentar ao aluno as arquiteturas computacionais, de maneira que sua compreensão ofereça uma visão mais clara e abrangente possível da natureza e das características dos sistemas de computadores modernos. Entretanto, durante a evolução do computador digital moderno foram projetados e construídos centenas de diferentes

tipos de computadores, o que aliado a atual expansão da variedade de computadores, tornou o objetivo da disciplina ainda mais complexa.

3 Simuladores em Arquitetura de Computadores

Neste capítulo são descritos conceitos fundamentais relacionados à utilização de Simuladores, como também alguns exemplos de simuladores destinados a disciplina de Arquitetura de Computadores. Em seguida são apresentadas as características do framework Ptolemy II, sendo feita uma descrição sobre o projeto orientado a atores (base dos modelos computacionais no ambiente), como também, uma apresentação dos principais domínios, em particular ao domínio DE. O estudo sobre tais características possibilitam a implementação de melhorias, consequentemente, adequando a ferramenta para propósitos específicos, no caso, sua utilização em modelagem e simulação de ambientes na disciplina de Arquitetura de Computadores.

3.1 Simulação na computação

Uma simulação é a imitação, durante determinado período de tempo, da operação de um sistema ou de um processo do mundo real. Feita a mão (raramente) ou em um computador (quase sempre), a simulação envolve a geração de uma história artificial do sistema, e a partir desta história artificial a inferência de como o sistema real funcionaria [Santos99].

A simulação é um processo de projetar um modelo computacional de um sistema real e conduzir experimentos com este modelo com o propósito de entender seu comportamento e avaliar estratégias para sua operação [Pegden90]. Contribuindo para a aquisição de um conjunto de conhecimentos e competências que permitam, perante uma situação específica, desenvolver um modelo de simulação, implementá-lo, validá-lo e utilizá-lo de forma correta.

Para a disciplina de Organização e Arquitetura de Computadores, a simulação é muito atrativa, principalmente quando se analisa a diversidade e a complexidade das arquiteturas. A modificação das funcionalidades e características de determinada arquitetura podem ser refletidas no programa de simulação, tendo seu impacto no desempenho verificado sem requerer grandes esforços ou custos maiores.

O uso de simuladores possibilita o estudo e a criação de experimentos de várias arquiteturas sem a necessidade de adquirir equipamentos caros, além de propiciar um ambiente mais conveniente do que uma máquina real. Através deles é possível verificar a existência de erros e prever a necessidade de mudanças, podendo realizá-las com muito mais facilidade, uma vez que o projeto é construído primeiro no simulador e não fisicamente.

Inicialmente, para ser realizada uma simulação, é necessário construir um modelo computacional que corresponda à situação real que se deseja simular.

3.2 Modelagem

A modelagem é a principal ferramenta no estudo do comportamento de sistemas complexos [Pritsker98]. Onde sua utilidade, conforme [Shruti and Loui08], é demonstrada em modelos computacionais usados extensivamente em ciência, engenharia, negócios e governo para fenômenos reais, onde os projetistas efetuam a modelagem a fim de adequar o modelo à realidade.

Para colaborar com o entendimento de modelagem em simulação, Moreira define modelo como [Moreira01]:

> "Uma representação simplificada da realidade ou das principais características de um sistema. Ele é composto por um conjunto de relações que podem ser expressas sob a forma de palavras, diagramas, tabelas de dados, gráficos, equações matemáticas ou qualquer combinação desses elementos e que possibilite a simulação de fenômenos observados empiricamente ou não."

Para Eduard Lee [Lee05] a modelagem é o ato de representar um sistema ou subsistema formalmente. Podendo tal modelo ser construtivo, caso em que se define um procedimento computacional que imita um conjunto de propriedades do sistema. Os modelos construtivos são frequentemente usados para descrever o comportamento de

um sistema em resposta a estímulos externos, podendo tais modelos ser chamados de modelos executáveis, ou simuladores.

O principal passo em um processo de modelagem é abstrair as características do sistema em um modelo, definindo as mais essenciais e estabelecendo uma relação entre essas características.

O processo de desenvolvimento de uma simulação [Bruschi03] envolve três fases: Desenvolvimento, Teste e Análise. Tais fases foram baseadas nos estudos de Pegden et. al. [Pegden90].

No **Desenvolvimento** os propósitos e objetivos do estudo devem ser claramente definidos. Para isso, a *descrição do sistema* é a primeira atividade, onde são levantadas características relevantes e filtrados os detalhes desnecessários que podem resultar em futuros erros.

Um Sistema é definido como um grupo de objetos que estão juntos em alguma interação ou interdependência, objetivando a realização de algum objetivo [Santos99].

Devem ser respondidas questões do tipo: i) Por que o problema está sendo estudado? ii) Quais serão as respostas que o estudo espera alcançar? iii) Quais são os critérios para avaliação do desempenho do sistema?

Conforme [Bruschi03], uma vez que o sistema a ser simulado está bem definido é necessário abstraí-lo em um modelo, o qual irá representar uma visão particular deste. A atividade de *Abstração do Sistema* e *Descrição do Modelo* tem esse papel, onde são levantadas perspectivas do modelo do sistema.

Após a descrição do modelo, é verificada a necessidade da definição dos tipos de dados que irão fazer parte do sistema a serem modelados e simulados. *A Coleta de Dados* é formada por fatos, informações e estatísticas fundamentais, que em geral são valores hipotéticos (quando não se possui referência) ou baseados em alguma análise preliminar.

Com os requerimentos do sistema e da representação dos seus dados já coletados a *Seleção do Método de Análise* é o próximo passo.

Para finalizar esta fase, após a decisão por simulação como método de análise, ocorre o *Desenvolvimento do programa de simulação*. Onde é necessário transformar a

descrição do modelo em programa de simulação. Segundo Santana [Santana90], pode-se seguir um dos seguintes enfoques:

- Linguagem de programação convencional: linguagem de programação C, entre outras;
- Utilização de um pacote de uso específico para o desenvolvimento do programa de simulação: voltados para avaliação de sistemas particulares, onde a formulação é construída na própria ferramenta;
- Uso de linguagens de simulação de uso geral: são linguagens que já contém todas as estruturas necessárias para a criação de um ambiente de simulação. São classificadas em: Orientadas a evento, a processo e atividade (descritas respectivamente nas seções 3.4.1 a 3.4.3);
- Uso de uma extensão funcional de uma linguagem de programação de uso geral: inclusão de bibliotecas inseridas em linguagens convencionais para composição do ambiente de simulação.

O objetivo do **Teste** é encontrar defeitos, revelando que o funcionamento do sistema em uma determinada situação não está de acordo com o esperado [Koscianski07]. Nesta fase, é preparado um conjunto de dados com a finalidade exclusiva de se testar os sistemas, sendo formado por três atividades: Depuração, Verificação e Validação.

Conforme Bruschi [Bruschi03] a *Depuração* ocorre pela busca e redução de defeitos no sistema, a fim de produzir resultados coerentes e sem erros. A *Verificação* consiste em verificar se o programa de simulação é uma implementação válida do modelo, ou seja, se o modelo operacional (programa) representa o modelo conceitual (modelo) [Bruschi03] apud [MacDougall87]. Podendo, conforme [Banks98], ser realizada: seguindo os princípios da programação estruturada (top-down ou programação modular); documentando muito bem o sistema; analisando o sistema por mais de uma pessoa; verificando se os valores de entrada estão sendo usados apropriadamente e utilizar um depurador.

Na *Validação*, também conhecida como calibração do modelo [Santos99], verifica se o modelo opera de acordo com a intenção do analista (sem erros de sintaxe e lógica) e que os resultados por ele fornecidos possuam crédito e sejam representativos

dos resultados do modelo real. As principais questões da Validação são: i) O modelo gera informações que satisfazem os objetivos do estudo? ii) As informações geradas são seguras e confiáveis? iii) A aplicação está isento de erros de programação?

Na **Análise** o sistema é colocado em produção e os dados obtidos são analisados, podendo envolver a execução, várias vezes, do modelo, variando-se os dados e os parâmetros de entrada, de forma a verificar se os resultados obtidos estão condizentes com os esperados. Caso sejam encontradas discrepâncias podemos ter que voltar à etapa de construção do modelo.

São exemplos de modelos para simulação:

- **Simulação Orientada a Eventos:** organizado como um conjunto de rotinas ou seções de eventos, onde a tarefa do modelador é determinar os eventos que podem causar mudança no estado do sistema e então desenvolver a lógica com cada tipo de evento.

- **Simulação Orientada a Processo:** como na Simulação Orientada a Evento, na Simulação Orientada a Processo também existe um mecanismo de escalonamento e uma estrutura de lista, com a diferença de que cada entrada na lista define um processo e seu ponto de reativação [Bruschi03].

- **Simulação Orientada a Atividades:** nesta abordagem, o modelador descreve cada atividade possível para cada entidade do sistema, definindo as condições que causam seu início e término.

- **Simulação Orientada a Atores:** sua principal característica é a decomposição de um sistema em ações, onde um ator é um conjunto de ações encapsuladas e parametrizadas, que possui dados de entrada, produz dados de saída e os dados são transmitidos através de portas bem definidas. As portas e os parâmetros são a interface de um ator, que define atividades locais sem referência implícita para outros atores [Liu02].

A Figura 1 [Zhou08] apresenta a comparação entre simulação orientada a objeto e orientada a ator.

Figura 1: Simulação (Orientado a Objeto *versus* Orientado a Ator)

Na simulação orientada a objetos, as interações entre os objetos são chamada de métodos e o que flui através de um objeto é o controle seqüencial. Na simulação orientada a ator, as interações entre os atores são as mensagens com base na passagem dos dados de entrada e saída e o que flui através de um ator é fluxos de dados.

A simulação orientada a atores é descrita detalhadamente nas seções a seguir devido a ser a base para o desenvolvimento deste trabalho, assim como o domínio de Eventos Discretos – *Discrete Event* (DE). É importante comentar que nesta técnica de simulação os blocos agendam chamadas, que são armazenadas em uma fila na forma de eventos. Cada evento possui um tempo de execução. O núcleo do simulador (*kernel*) gerencia os eventos, retirando os eventos da fila no instante de tempo apropriado, entregando-o para o bloco para o qual o evento se destina [Kuller03].

3.3 Simuladores

Os simuladores computacionais são recursos tecnológicos que podem auxiliar em diversas áreas. No contexto abordado neste trabalho, os simuladores proporcionam formas dos professores criarem cenários para ensino e aprendizagem pedagogicamente interessantes para o nível de ensino que deseja. Do ponto de vista pedagógico, o simulador computacional é um software útil e simples, uma vez que levam os alunos a interagirem com os mesmos, o que lhes facilita a compreensão de conceitos abstratos, leis e características do mundo físico.

Segundo Felix [Felix06], na disciplina de Organização e Arquitetura de Computadores são notáveis as necessidades de ferramentas gráficas e didáticas para auxiliar o aprendizado. Especificamente para explicar a arquitetura de processadores,

pois são necessárias muitas figuras para ilustrar os caminhos de dados e de controle quando uma instrução é executada.

Conforme comentado sobre necessidades da disciplina, podemos citar algumas ferramentas voltadas para o campo da informática educativa, especificamente em uma das áreas correlatas (processadores) ao tema da pesquisa. Dentre os vários processadores, dos mais simples aos mais complexos, destacamos o processador MIPS (*Microprocessor without Interlocked Pipe Stages*). O MIPS é uma arquitetura de microprocessador projetada com objetivo de oferecer alto desempenho na execução de código compilado. Suas instruções são todas de 32 bits, mesmo nos processadores de 64 bits, como forma de manter a compatibilidade.

Atualmente, podemos verificar vários simuladores para o processador MIPS. A seguir, são descritos alguns exemplos e suas características em comum com os outros simuladores.

O WebMIPS [Branovic04] possui uma plataforma gráfica visualmente agradável para simular a arquitetura MIPS com pipeline. Ele possui a vantagem de ser uma plataforma disponível na Internet, proporcionando uma interface dinâmica, a qual o estudante pode ter acesso sem ter que instalar módulos do simulador.

O DIMIPSS [Felix et al.06] (*Didact Interactive MIPS Simulator*) é um software multiplataforma de simulação da execução (caminho de dados e de controle) das instruções do MIPS Monociclo. Ele recebe um programa em linguagem de montagem (assembly), o converte para a linguagem de máquina e representa graficamente o comportamento do caminho de dados e de controle durante a sua execução.

O MARS (*Mips Assembly and Runtime Simulator*) desenvolvido por Vollmar e Sanderson [Vollmar05] é uma ferramenta em Java com a IDE gráfica implementada usando Swing que simula um conjunto de instruções do MIPS32. A versão atual é a 3.8 e implementa 98 instruções, 36 pseudo-instruções e 17 chamadas de sistemas.

Outra ferramenta é o WinMIPS64 [Scott06] que simula a arquitetura pipeline do microprocessador MIPS64, apresentando um conjunto de 32 registradores para números inteiros, e 32 para números representados em pontos flutuantes.

O WinMIPS64 possibilita a visualização da instrução passando pelos pipelines, e ainda estatísticas relativas à execução, como, por exemplo, quantidade de ciclos que

determinada instrução consumiu para passar por todos os estágios do pipeline. E por último, descrevemos o MipsIt [Brorsson02] que é um ambiente de desenvolvimento integrado e que também realiza a simulação do processador MIPS, possibilitando a programação em alto nível e permitindo relacionar conceitos de programação com organização e arquitetura de computadores.

A Tabela 1 descreve uma análise comparativa entre ferramentas de simulação. Os tópicos abordados baseiam-se nos estudos de [Nikolic et al09] para avaliar o potencial educacional dos simuladores.

Tabela 1: Análise comparativa entre as ferramentas de aprendizagem

Ferramentas	Ambiente e Complexidade (A)	Suporte ao Projeto (B)	Apresentação Visual (C)	Nível de Simulação (D)	Ensino a Distância (E)	Detalhamento de Implementação (F)
WebMIPS	AA	N	S	IL	S	S
DIMIPSS	AB	N	S	IL	N	N
MARS	AB	N	S	PL	N	S
WinMIPS64	AB	N	N	IL	S	S
MipsIt	AA	N	S	IL/PL	N	S
Ptolemy	AB/AA	S	S	CL/IL	S	S

Onde (A) avalia se o simulador apresenta noções básicas da introdução à Organização e Arquitetura de Computadores. Os níveis de avaliação são Arquitetura Básica (AB) e Arquitetura Avançada (AA). O critério (B) avalia se o simulador inclui ferramentas de apoio à elaboração de modulos informatizados reutilizáveis ou apenas simula sistemas pré-definidos.

Para o critério (C) existem aqueles simuladores que detalham visualmente o funcionamento interno de um sistema computacional seguindo um fluxo de dados (descrevendo o caminho percorrido pela instrução), ou aqueles onde apenas são apreserntados os resultados (forma textual). Ainda sobre os critérios utilizados, o (D) diz respeito ao nível de detalhamento do comportamento de um objeto exposto, com base no nivel mais baixo de granularidade suportado durante uma seção de simulação. Podendo ser: Nível de Clock (CL), Nível de Instrução (IL) ou Nível de Programa (PL). Para o critério (E), *c*om base na disponibilidade de suporte para ensino à distância, os simuladores podem ser divididos entre aqueles com e sem suporte para o critério. E finalizando o (F) avalia se os simuladores fornecem detalhes da execução visíveis durante a simulação.

Após a análise das ferramentas verificou-se que o Ptolemy é o único que possibilita suporte ao projeto (B), pois permite que o aluno crie seus próprios elementos e possa reutilizá-los em diferentes níveis de abstração, como também em outros projetos e arquiteturas. Além disso, o Ptolemy possui algumas vantagens em relação às outras ferramentas, como por exemplo: i) no WebMIPS [Branovic04] as modificações que ocorrem nos caminhos de dados e controle durante a execução das instruções não são apresentadas; ii) O DIMIPSS [Felix et al.06] não é uma ferramenta web, dificultando o ensino a distância. Além de só implementar um pequeno número de instruções do MIPS monociclo; iii) o MARS não possui suporte a depuração e não permite que o usuário desfaça etapas de execução; iv) o WinMIPS64 [Scott06] não possui uma área para edição do código de execução pelo usuário; v) já o sistema MipsIt [Brorsson02] a arquitetura do processador MIPS, ilustrada no MipsIt, possui complexidade acima do desejável para ilustração de conceitos aos alunos que estão ingressando em cursos de Computação.

Podemos concluir que apesar do framework Ptolemy não ser um simulador didático (não possuindo um procedimento para o ensino a distância e de ter uma interface gráfica complexa), ele oferece a funcionalidade de criação de um repositório de atores, possibilitando ao aluno criar seus próprios cenários, como também seus atores, além de ser uma ferramenta gratuita e com o código fonte aberto, possibilitando que novas funcionalidades e mudanças na sua interface sejam realizadas para torná-la uma ferramenta apropriada para fins educacionais.

Muitas técnicas de Ensino de Arquitetura de Computadores utilizam de simuladores de arquiteturas reais ou hipotéticas [Yurcik02]. Para Brito [Brito09], estas técnicas, porém, não permitem uma abordagem construtivista onde os alunos possam criar suas próprias arquiteturas.

A utilização do Ptolemy como ferramenta de simulação neste projeto é devido a possibilidade de criar seus próprios ambientes, simular praticamente qualquer sistema concorrente e estimular a formação de novos atores através da composição de atores já existentes, tendo como principais características o incentivo à criatividade e ao entretenimento [Brito09].

Nas próximas seções são descritos os passos para um projeto no Ptolemy.

3.4 Projeto Orientado a Atores

Para Liu [Liu02], um Projeto Orientado a Atores é a forma de interação entre os componentes, utilizando-se da noção de modelo de computação – *Model of Computation* (MoC) que define a semântica de comunicação entre as portas e o fluxo de controle entre os atores e cuja implementação representa a estrutura onde o ator está inserido.

Liu [Liu02] defende a utilização do conceito de hierarquia, onde uma rede de atores pode ser visto como um único ator, possibilitando dividir um complexo modelo em uma árvore de submodelos e de cada nível formar uma rede de interação de componentes.

A Figura 2 [Lee05] representa o ator ***Sinewave***, contido num modelo orientado a atores, de forma a ilustrar a abstração hierárquica da imagem maior, composta por outros atores (Ramp, Const, Add/Substract e TrigFunction).

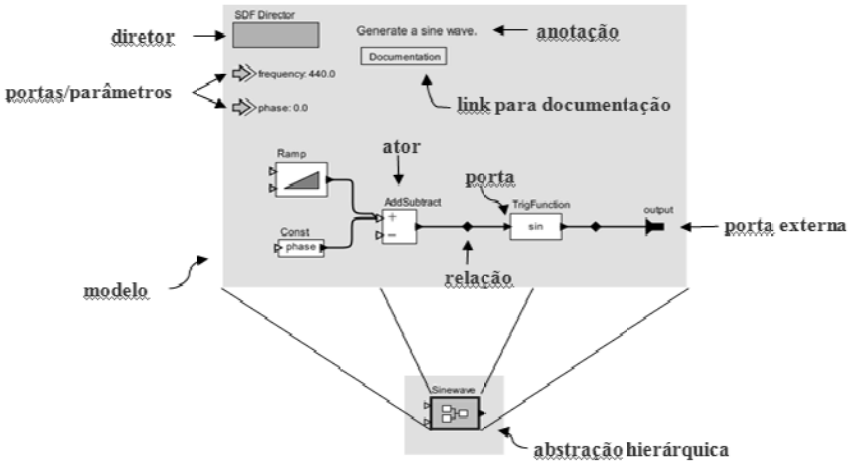

Figura 2: Modelo Orientado a Atores
Fonte: Lee08

Uma das vantagens de utilizar abstração hierárquica é que podemos modelar sistemas heterogêneos mais complexos, integrando diferentes modelos de computação, especificados nos diferentes níveis hierárquicos [Liu02].

O Modelo Orientado a Atores se qualifica como uma ferramenta adequada para a modelagem de ambientes estudados pelas ciências básicas vistas no ensino médio e fundamental, isso por apresentar duas características principais, a concorrência (os atores serão executados simultaneamente) e a orientação a mensagens (atores

configurados para receber mensagens, reagindo a essas e possibilitando gerar novas mensagens na saída de suas portas para outros atores) [Brito09].

Uma vez que os atores tenham sido modelados, ambientes podem ser organizados de forma que vários atores interajam trocando mensagens, reagindo e evoluindo de acordo com os estímulos gerados pelos próprios atores do mesmo ambiente de forma.

A seguir apresentaremos o ambiente Ptolemy II, descrevendo suas definições, funcionalidades e conceitos básicos para o desenvolvimento de modelos orientado a atores.

3.4.1 Projeto Ptolemy

Este trabalho se baseia na utilização do framework Ptolemy II criado do Projeto Ptolemy, que é mantido por um grupo de pesquisadores da Universidade da Califórnia, em Berkeley desde 1990. O projeto inicial foi desenvolvido utilizando a linguagem C++, já com o Ptolemy II houve a migração para Java.

O Ptolemy II é uma coleção de classes e pacotes Java, com capacidades específicas, cujo núcleo suporta uma sintaxe abstrata, definida como uma estrutura hierárquica de entidades com portas e interconexões. Essas simulações podem envolver sistemas complexos que executam diferentes operações como processamento de sinais e redes de computadores.

O framework descrito por [Lee03] é baseado no Modelo de Atores. Este consiste na definição de entidades denominadas “atores”, 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.

O Ptolemy II possibilita a modelagem de sistemas embarcados, particularmente aqueles que mesclam tecnologias, como dispositivos eletrônicos analógicos e digitais, dispositivos eletromecânicos, hardwares dedicados e como foco de pesquisa atual, para sistemas heterogêneos e com complexidade nos vários níveis de abstração.

Existem muitas maneiras de utilizar o Ptolemy II. Ele pode ser utilizado como uma estrutura para montagem de componentes de software, como modelagem e ferramenta de simulação, como um editor de diagrama de blocos, como uma aplicação de nível de sistema de prototipagem rápida, como um kit de ferramentas baseado em

design apoiando à investigação de componentes, ou como um conjunto de ferramentas para construção de aplicações em Java. Esse trabalho apresenta sua utilização como ferramenta de modelagem e simulação.

Inicialmente, a algumas dificuldades no manuseio do framework, logo superadas pelas vantagens oferecidas. Tais como:

- Um grande conjunto de mecanismos de interação entre os domínios heterogêneos, forçando os desenvolvedores de componentes a pensarem no padrão como os outros componentes do 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 diferentes;
- Possui ferramentas para a simulação dos modelos, permitindo a execução das mesmas como aplicações locais ou *web*, podendo ser utilizadas para apresentações à distância. É importante ressaltar tal ponto, em virtude da crescente utilização de ferramentas virtuais para ensino.
- Possui uma linguagem própria de marcação (*Markup Language*), baseada em XML (denominada 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.

Na próxima seção apresentamos mais detalhes sobre a orientação a atores que serve de base para a modelagem utilizada no Ptolemy II.

3.4.1.1 Projetando atores

Basicamente, atores são definidos como entidades que processam dados presentes nas suas portas de entrada ou que criam e enviam dados para outras entidades por meio de suas portas de saída. Como ilustra a Figura 3, os atores *Constant* e *Constant2* enviam os dados de suas portas de saídas para o ator *Add or Substract* através de suas conexões, denominadas de *relations*.

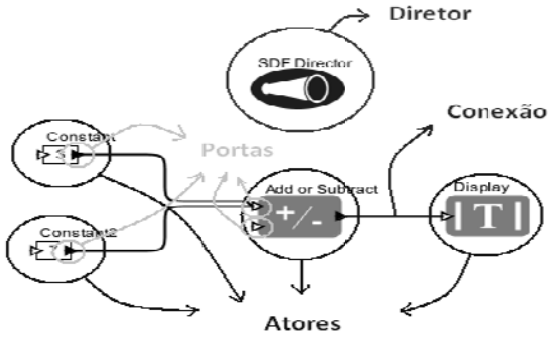

Figura 3: Anatomia de um ator

Entretanto, atores somente têm conhecimento da disponibilidade de dados nas suas portas de entrada e como processar estes dados e mandá-los para suas portas de saída? A tarefa de transportar dados é responsabilidade das portas, já que um ator não sabe com quais atores ele está conectado.

Para exemplificar tal comportamento, a Figura 4 [Lee05] ilustra a o comportamento entre os atores E1 e E2.

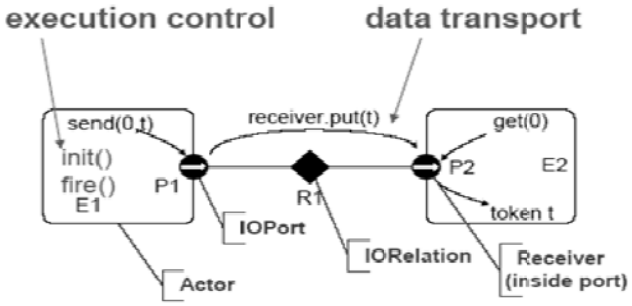

Figura 4: Comportamento de uma conexão entre atores

O ator **E1** tem uma porta de saída **P1**, indicada na Figura com uma flecha na direção do fluxo do ***token t***. O ator **E2** tem uma porta de entrada **P2**. Para transportar os dados para o ator **E2, E1** convoca o método ***send()*** de **P1** para enviar um ***token t***. A porta obtém a referência ao receptor remoto, através da relação **R1** (***IORelation***) e chama o método ***put()*** do receptor, passando-lhe o ***token***. O ator **E2** recupera o ***token t*** convocando o método ***get()*** de sua porta de entrada, a qual, por sua vez, convoca o método ***get()*** do receptor designado.

A Figura 5 ilustra os passos para projetar atores, detalhando suas características e funcionalidades. A fase de Especificação detalha a funcionalidade do ator a ser criado, como também são informadas as entidades contidas na classe, como: nome, tipo

(variável, porta, etc.), dado (string, inteiro, etc.), direção (entrada, saída ou interna) e a ação/função. Na fase de Implementação, o desenvolvedor escreve o código Java e adiciona a classe ao projeto Ptolemy. Sendo a fase Testes a responsável por verificar se o ator corresponde ao comportamento desejado, utilizando para isso os dados nas portas de I/O.

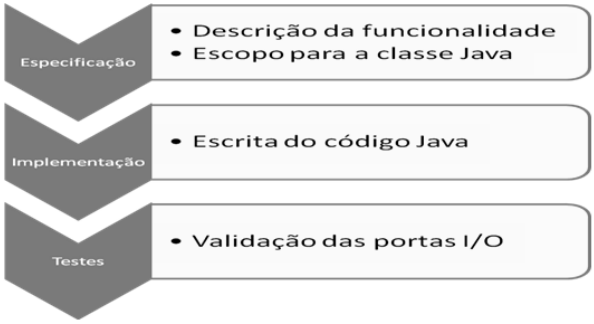

Figura 5: Fases para projeto de atores personalizados

Após as fases de especificação e implementação, devemos adicionar a classe referente ao ator no arquivo XML que deverá conter a lista de atores do modelador. A Tabela 2 exemplifica a utilização do arquivo *myactors.xml* descrevendo as classes dos atores criados.

Tabela 2: Conteúdo do arquivo myactors.xml

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE plot PUBLIC "-//UC Berkeley//DTD MoML 1//EN"
   "http://ptolemy.eecs.berkeley.edu/xml/dtd/MoML_1.dtd">
<entity name="MyActors" class="ptolemy.moml.EntityLibrary">
 <configure>
   <?moml
     <group>
 <doc>Atores criados por Aluno</doc>
          <entity name="ALU" class="ptolemy.actor.myactors.ALU"></entity>
          <entity name="Add" class="ptolemy.actor.myactors.Add"></entity>
     </group>
   ?>
 </configure>
</entity>
```

Após tal procedimento, deve-se adicionar na biblioteca de utilização do Ptolemy (*ptII/ptolemy/actor/lib/myactors*) o arquivo.

É recomendável verificar a configuração dos atores criados, de forma que fiquem disponíveis na área de criação dos novos projetos no ambiente gráfico.

Todos os arquivos criados nesse projeto são informados no arquivo *myactors.xml*, baseados nos arquivos (.java) dos atores criados. O arquivo deverá conter o nome do ator, como também sua classe Java, conforme abaixo:

```
<enttly name=”actor” class =”ptolemy.actor.myactors.lib.actor”></entity>
```

Após a criação/edição do arquivo na pasta informada, deve-se colocar todos os novos atores na pasta, como também editar o arquivo *myactors.xml.*

O próximo passo é editar o arquivo *defaultFullConfiguration.xml* (localizado na pasta */ptolemy/configs*), informando no arquivo o caminho para o arquivo *myactors.xml***,** conforme exposto na seleção da Figura 6.

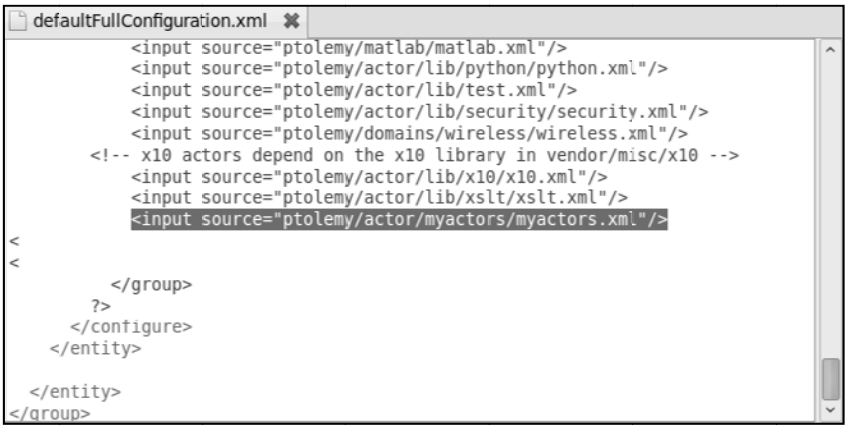

Figura 6: Visualização do arquivo *defaultFullConfiguration.xml*

a. Anatomia de um ator

Para [Lee05], cada ator consiste em um código fonte escrito em Java e compilado no arquivo *makefile* do seu diretório. Assim, ao criar um novo ator, é necessário acrescentar o seu nome para o *makefile* local.

O Vergil é a ferramenta de design gráfico utilizada para compor os atores e outros componentes, disponibilizando o ator em uma das bibliotecas do ator. Permitindo arrastar o ator para a área de design.

As bibliotecas são arquivos XML e muitas delas estão no endereço /PtII Ptolemy/ator/lib. A estrutura básica de um ator é mostrado na Figura 7.

```
/** Javadoc comment for the actor class. */
public class ActorClassName extends BaseClass implements MarkerInterface {

    /** Javadoc comment for constructor. */
    public ActorClassName(CompositeEntity container, String name)
            throws NameDuplicationException, IllegalActionException  {
        super(container, name);
        // Create and configure ports, e.g. ...
        portName = new TypedIOPort(this, "portName", true, false);
        // Create and configure parameters, e.g. ...
        parameterName = new Parameter(this, "parameterName");
        parameterName.setExpression("0.0");
        parameterName.setTypeEquals(BaseType.DOUBLE);
    }

    ////////////////////////////////////////////////////////////////////
    ////                     ports and parameters                   ////

    /** Javadoc comment for port. */
    public TypedIOPort portName;

    /** Javadoc comment for parameter. */
    public Parameter parameterName;

    ////////////////////////////////////////////////////////////////////
    ////                         public methods                     ////

    /** Javadoc comment for fire method. */
    public void fire() {
        super.fire();
        ... read inputs and produce outputs ...
    }

    /** Javadoc comment for initialize method. */
    public void initialize() {
        super.initialize();
        ... initialize local variables ...
    }

    /** Javadoc comment for prefire method. */
    public boolean prefire() {
        ... determine whether firing should proceed and return false if not ...
        return super.prefire();
    }

    /** Javadoc comment for preinitialize method. */
    public void preinitialize() {
        super.preinitialize();
        ... set port types and/or scheduling information ...
    }

    /** Javadoc comment for postfire method. */
    public boolean postfire() {
        ... update persistent state ...
        ... determine whether firing should continue to next iteration and return false if not ..
        return super.postfire();
    }

    /** Javadoc comment for wrapup method. */
    public void wrapup() {
        super.wrapup();
        ... display final results ...
    }
}
```

Figura 7: Estrutura de um ator

No projeto Ptolemy II 7.0, sua base de desenvolvimento tem como princípios a orientação a atores. Logo seus principais componentes são:

- **Diretor:** componente que controla a execução do *workflow*, gerenciando outros componentes (atores, portas etc.). Define o modelo computacional a ser utilizado (síncrono, paralelo, distribuído, etc.), sendo obrigatória a sua presença.

- **Ator:** componente do *workflow* que representa um dado ou serviço. Podem ser conectados com outros atores por meio de portas e ter parâmetros configuráveis.
- **Porta:** cada ator pode conter uma ou mais portas, utilizadas no consumo e na produção de dados assim como na comunicação com outros atores envolvidos no *workflow*. Atores são conectados entre si por meio de portas. A conexão que representa o fluxo de dados entre um ator e outro é chamada de canal. As portas podem ser divididas em três tipos diferentes:
    - **Entrada:** usada para consumo de dados;
    - **Saída:** destino dos dados produzidos pelo ator;
    - **Entrada/Saída:** para consumo e saída de dados.

Além disso, as portas podem ser configuradas como simples ou múltiplas (multiportas). Uma porta de entrada simples pode estar conectada a um único canal. Já uma porta de entrada múltipla pode estar conectada a vários canais simultaneamente.

- **Relação (***Relations***):** permitem replicar fluxos de dados. Assim, o mesmo dado pode ser mandado para vários lugares do *workflow*.
- **Parâmetro:** são valores configuráveis que podem fazer parte de um *workflow*, diretor ou ator.

b. Atores Compostos

Uma funcionalidade a mais do Ptolemy é permitir a composição de vários atores em apenas um ator composto, evitando com isso um cenário muito complexo para o entendimento, além da criação de **"***caixas pretas***"** para representarem grande quantidade de componentes. Seguindo os seguintes passos:

- Criação de atores isoladamente;
- Adicionar o ator (*CompositeAtor*), localizado na opção *Utilities* do menu da ferramenta Ptolemy, e renomear para o nome desejado para o ator composto. Logo após, selecionar os atores que irão fazer parte do ator composto (Figura 8).

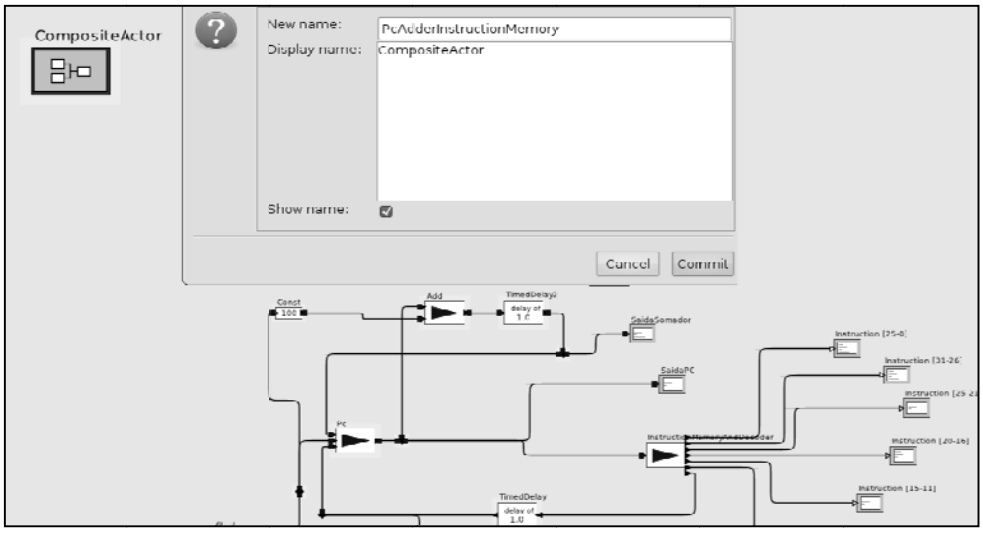

Figura 8: Exemplo de criação de ator composto (CompositeActor)

- Ao selecionar e recortar os atores (*Opção Edit - > Cut*), o próximo passo é clicar com o botão direito sobre o ator composto criado e utilizar a opção *Open Actor*. Será aberta outra janela de simulação contendo os atores que irão fazer parte do ator composto (*Datapath1*). Sendo preciso agora incluir novas portas de entrada e de saída (Figura 9).

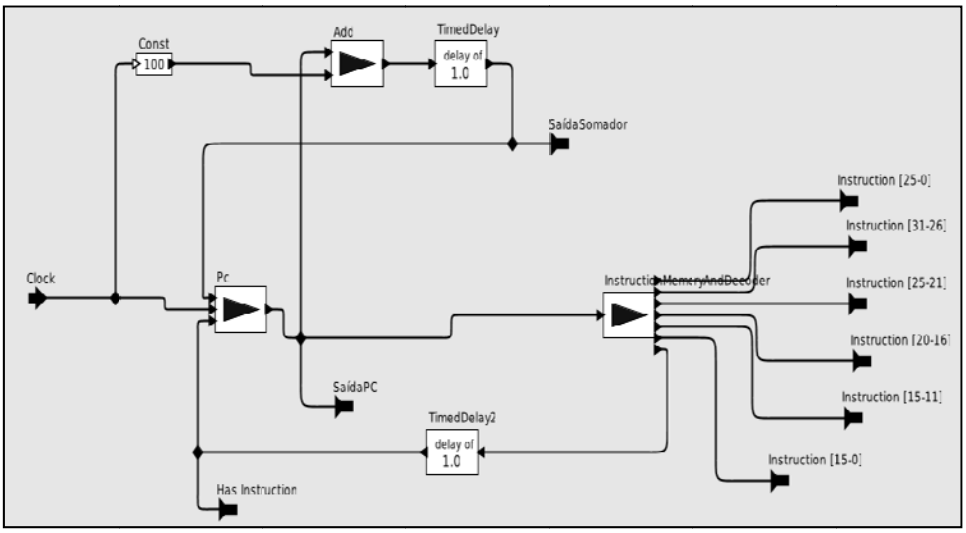

Figura 9: Composição do ator composto Datapath1

- Finalizando, devemos agora conectar os *Displays* e *Clocks* ao ator composto criado. O ator terá o mesmo comportamento do caminho de dados realizados pelos atores primários. A Figura 10 representa o ator composto.

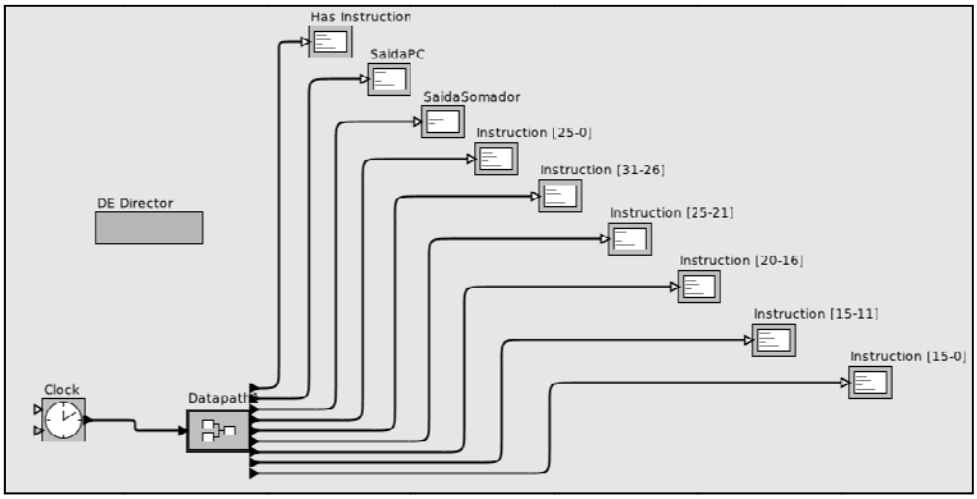

Figura 10: Representação do ator composto

c. Portas

As portas são definidas por Lee [Lee05], como componentes dos atores que representam um conjunto de entrada e saída de canais através dos quais os *tokens* podem passar os dados processados num ator para os demais atores do cenário.

As portas declaradas (*portName*) no ator são instâncias de *TypedIOPort*, declarado como: *public TypedIOPort portName*. A menos que necessitem de serviços de domínio específico, caso em que eles podem ser instâncias de uma subclasse de domínio específico, como *DEIOPort*. A porta é realmente criada no construtor pela linha: *portName = new TypedIOPort(this, "portName", true, false).*

O primeiro argumento para o construtor é o contêiner da porta deste ator. O segundo é o nome da porta, que pode ser qualquer ***string***. O terceiro argumento especifica se a porta é de entrada e o quarto argumento especifica se a porta é de saída. Podendo ocorrer de termos uma porta que seja de entrada e saída.

Um ator pode declarar suas portas com a seguinte linha.

```
<entity name="A" class="classname">
<port name="out"/>
</entity>
```

Na linha de código acima, nenhuma classe é declarada para a porta. Logo é necessário ser declarada para que a mesma exista na classe para a entidade A. Como alternativa, pode especificar um nome de classe, como na linha de código abaixo.

```
<entity name="A" class="classname">
<port name="out" class="classname"/>
</entity>
```

Neste caso, como a porta não existe, ela é criada. Onde a porta pré-existente deve ser uma instância da classe declarada. No Ptolemy II, o nome da classe típica de uma porta seria *ptolemy.actor.TypedIOPort.*

É uma importante informar na declaração da porta se a mesma é de entrada, de saída, ou de dupla funcionalidade. Para fazer isso, é informada na classe referente a porta uma propriedade chamada "entrada" ou "saída", ou ambas, como no exemplo a seguir:

```
<port name="out" class="ptolemy.actor.IOPort">
<property name="output"/>
</port>
```

d. Construtores

Tem como objetivo a criação e configuração das portas e dos parâmetros, chamar o método *super (container, name)* e convocar as bibliotecas de exceções *NameDuplicationException* e *IllegalActionException*. Esta última é a exceção mais utilizada, e muitos métodos de atores o declaram. O primeiro é acionado se o container especificado já contém um ator com o nome especificado.

e. Parâmetros

Os valores da maioria dos parâmetros dos atores podem ser dados como expressões. Suas variáveis na expressão se referem a outros parâmetros que estão no escopo, que são aqueles contidos no mesmo recipiente ou algum recipiente acima na hierarquia, podendo ainda referenciar variáveis e extensões de atributos em um escopo.

Para [Lee05] é possível definir um parâmetro que também é uma porta. No caso a biblioteca *PortParameter* fornece um valor padrão, que é especificado como o valor de qualquer outro parâmetro, mas quando a porta correspondente recebe dados o valor padrão é substituído com o valor fornecido na porta. Assim, esse objeto funciona como um parâmetro e uma porta.

Um exemplo de um ator que usa uma biblioteca *PortParameter* é o ator *Sinewave*, que é encontrado na biblioteca de fontes do *Vergil*. É mostrado na Figura 11 [Lee03] como efetuar tal procedimento, que no caso, deve-se dar um duplo clique sobre esse ator, definindo os valores padrão de freqüência e fase. Mas esses valores também podem ser definidos pelas portas correspondentes, os quais são mostrados com cinza de preenchimento.

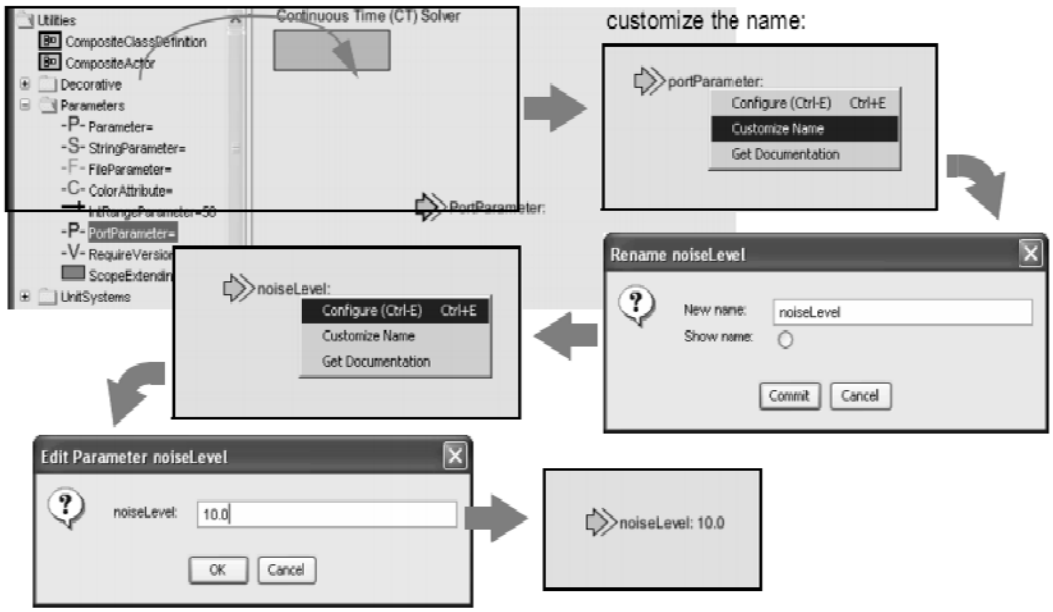

Figura 11: Exemplo de utilização da Biblioteca *PortParameter*

Alguns parâmetros têm valores que são sempre strings de caracteres. Tais parâmetros apóiam um mecanismo de substituição simples onde o valor da seqüência pode receber parâmetros de referência no âmbito de outros pelo nome usando a sintaxe $ name, onde name é o nome do parâmetro de sua abrangência.

f. Diretores

Para dar semântica ao modelo computacional a ser criado, o Ptolemy II requer a especificação de um diretor associado a um modelo, uma entidade, ou uma classe. Para [Brooks08] um diretor rege a execução de uma entidade composta e um gerente (*Manager*) rege a execução global do modelo.

Um exemplo do uso dessas classes é mostrado na Figura 12. Onde um ator de nível superior, E0, tem uma instância do Diretor (D1), servindo como seu diretor local. Onde um diretor local é responsável pela execução dos componentes dentro do cenário

completo. Ele irá executar qualquer programação que possa ser necessária, e convocar os elementos que precisam ser iniciados, gerando o código que precisa ser gerado, etc. No exemplo, D1 também serve como um “diretor executivo” para a E2. O diretor executivo que é associado a um ator é o diretor responsável por disparar o ator. Um ator de composição que não está no nível superior pode ou não pode ter o seu próprio diretor local. Na Figura 22 [Brooks08], E2 tem um diretor local e E3 não. O conteúdo de E3 está diretamente sob o controle da D1. Mas os conteúdos de E2 estão sob o controle de D2, que por sua vez está sob o controle de D1.

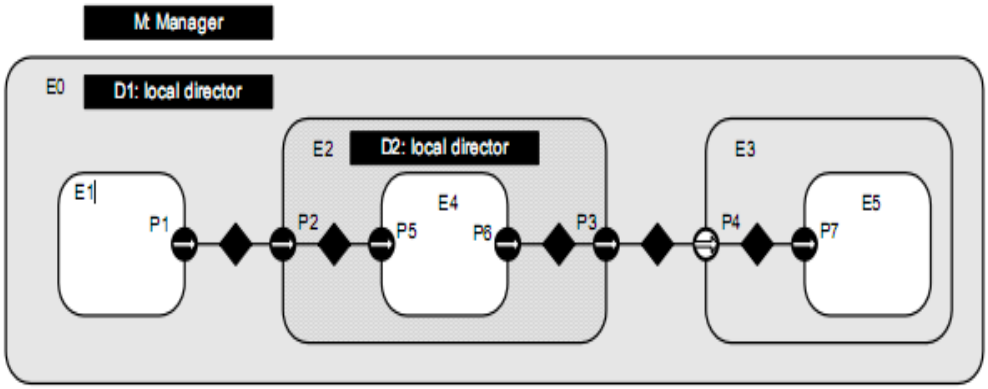

Figura 12: Exemplo de utilização de Diretores

O diretor é uma propriedade do modelo. O exemplo a seguir dá a semântica de eventos discretos para um modelo no Ptolemy II.

```
<entity name="top" class="ptolemy.actor.TypedCompositeActor">
<property name="director"
 class="ptolemy.domains.de.kernel.DEDirector">
<property name="stopTime" value="100.0"/>
</property>
...
</entity>
```

No exemplo acima também podemos definir as propriedades do diretor, sendo que o nome do diretor não tem muita relevância, exceto pelo fato de não poder coincidir com o nome de qualquer outra propriedade no modelo.

g. Links/Relações

Conforme a Figura 13 [Zhou08], um ator (*Entity*) pode ser comunicar com outros atores por meio de uma relação (*Relation*) conectada a suas portas (Port). A conexão da relação com as portas dos atores é denominada *Link*.

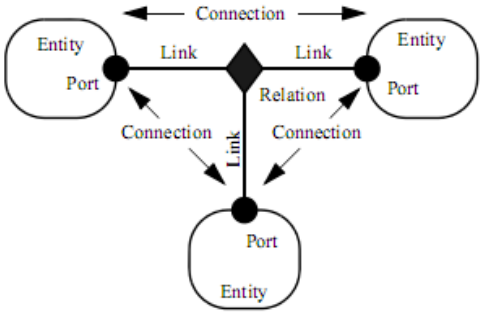

Figura 13: Comunicação entre atores

Para [Lee05], *links* é uma associação de uma porta com qualquer número de relações (*relations*), onde existe uma instância denominada, *NamedList*, associada a o ator que é usada para agregar as portas. E uma relação é associada a qualquer número de portas. A Figura 14 descreve a topologia de atores com a utilização de *relations*.

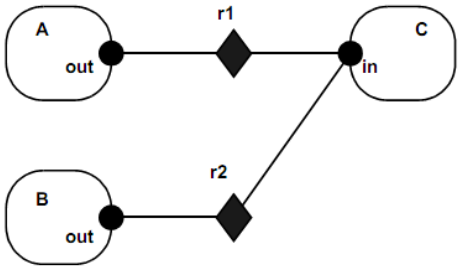

Figura 14: Topologia de Atores

Do lado da porta, os links têm uma ordem. Eles são indexados de 0 a n, onde n é o número retornado pela *numLinks()* da Porta. O código abaixo descreve como conectar os atores da Figura 13, criando as relations e links.

```
<entity name="top" class="classname">
<entity name="A" class="classname">
<port name="out"/>
</entity>
<entity name="B" class="classname">
```

```
<port name="out"/>
</entity>
<entity name="C" class="classname">
<port name="in">
<property name="multiport"/>
</port>
</entity>
<relation name="r1" class="classname"/>
<relation name="r2" class="classname"/>
<link port="A.out" relation="r1"/>
<link port="B.out" relation="r2"/>
<link port="C.in" relation="r1"/>
<link port="C.in" relation="r2"/>
</entity>
```

No Ptolemy II, o nome típico para classe de uma *Relation* é *ptolemy.actor.TypedIORelation*. O atributo de classe pode ser omitido se a *Relation* já existe na classe do ator.

Observe na Figura 14 que existem dois *links* distintos para a porta *in* do ator C originados de duas *Relation* diferentes. Dizemos que C possui dois *links*, indexado 0 e 1. O elemento de ligação explicitamente pode dar o número do índice no qual é inserido o *links*. Por exemplo, poderíamos ter conseguido o mesmo efeito acima dizendo:

```
<link port="C.in" relation="r1" insertAt="0"/>
<link port="C.in" relation="r2" insertAt="1"/>
```

Sempre que a opção *insertAt* não for especificada, a *Relation* é sempre acrescentados ao final da lista de *links*. Quando a opção *insertAt* for especificada, o *link* será inserido nessa posição, portanto, qualquer laços pré-existentes, com índices maiores terão seus números de índice incrementado. Com por exemplo, se fizermos:

```
<link port="C.in" relation="r1" insertAt="0"/>
<link port="C.in" relation="r2" insertAt="1"/>
<link port="C.in" relation="r3" insertAt="1"/>
```

Em seguida, haverá um *link* para r1 com o índice 0, um *link* para r2 com índice 2 e um *link* para r3 com o índice 1.

3.4.1.2 Domínios

O Ptolemy II é constituído de diferentes domínios cada qual com seu modelo computacional. O núcleo do Ptolemy assegura que diferentes modelos computacionais se comunicam de maneira definida e consistente. Cada domínio tem uma semântica de interação que determina o comportamento e os sistemas onde melhor se aplicam.

Atualmente os domínios mais maduros são o de eventos discretos – *Discrete Event* (DE), e o *Synchronous Dataflow* (SDF) utilizado para modelagem do comportamento de sistemas síncronos como processamento de sinais, onde o escalonamento é estático. O Ptolemy II oferece também domínios para geração de código, utilizados em sistemas embarcados.

Como já informado anteriormente, o domínio utilizado para os modelos computacionais na extensão Ptolemy foi o DE. Esse modelo tem como objetivo reproduzir atividades das entidades que compõe o sistema e efetuar o aprendizado sobre o comportamento e desempenho do sistema.

Buscando entender e analisar um Sistema de Eventos Discretos, alguns termos precisam ser definidos:

- **Sistema:** Conjunto de entidades que interagem ao longo do tempo para conseguirem um ou mais objetivos;
- **Entidade:** Objeto de interesse do sistema. Exemplos: Sistema de Tráfego (carros, semáforos e ruas) e supermercado (clientes, caixas e estacionamento);
- **Atributo:** Denotam a propriedade de uma entidade. Exemplos: Sistema de tráfego - carros (Velocidade, tamanho e posição na fila);
- **Atividade:** Intervalo de tempo (Exemplo: tempo de serviço), que consome certa quantidade de tempo e é conhecido quando começa;
- **Estado:** É definido como a coleção de variáveis necessárias para descrever o sistema (atributos) em um dado instante;

- **Evento:** É definido como a ocorrência instantânea que pode mudar o estado do sistema. Exemplo: Cliente entrou no supermercado
- **Processo:** É a sequência de transformações pela qual passa uma ou várias entidades.

A Figura 15 ilustra o gerenciamento de vôos de um aeroporto para exemplificar o Sistema de Eventos Discretos. A mudança de estado neste tipo de simulação é determinada pela ocorrência de um evento em um tempo determinado.

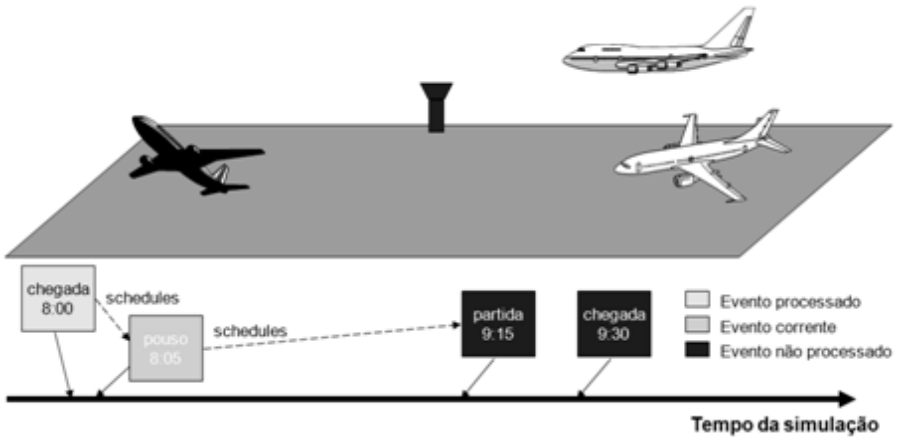

Figura 15: Exemplo de Sistema de Evento Discreto (Sistema de vôos)

Este exemplo pode ser generalizado para outros sistemas com uma fila onde os usuários ficam a espera e um servidor que realiza atendimento de um usuário de cada vez, como por exemplo, o controle de um estacionamento de um shopping ou de uma agência bancária com um único caixa.

Conforme [Lee05] o domínio Eventos Discreto – *Discrete Events* (DE) criado por Lukito Muliadi [Muliadi99], os atores se comunicam através de seqüências de acontecimentos ao longo de uma linha de tempo real.

Para [Liu01] o domínio DE possibilita a comunicação de atores através de eventos que são organizados em uma linha do tempo real. O DE é muito utilizado para a especificação de hardware digital e para a simulação de sistemas de telecomunicações, e tem sido realizadas em um grande número de ambientes de simulação, linguagens de simulação, e linguagens de descrição de hardware, incluindo *VHSIC Hardware Description Language* (VHDL) e Verilog.

No artigo "*Discrete Event Modeling in Ptolemy II*" [Muliadi99] comenta que o domínio DE típico consiste de uma rede de atores na qual o diretor DE rege a execução do modelo.

No DE, os atores contêm as portas e as portas são conectadas umas às outras através de relações, enviando sinais através das portas (de entrada, portas de saída, ou ambos). Os *tokens* são enviados por uma porta de saída e recebidos por todas as portas de entrada conectadas à porta de saída através das relações, empacotados como um evento e armazenados na fila de eventos globais.

Uma vantagem da utilização do Ptolemy com o domínio DE, em relação aos demais simuladores é o fato do domínio DE possibilitar a semântica determinística para eventos simultâneos, diferentemente da maioria dos simuladores para eventos discretos.

A Tabela 3 descreve os demais domínios utilizados no Ptolemy.

Tabela 3: Domínios no Framework Ptolemy

Domínio	**Descrição**
Interação de Componentes – *Component Interaction* (CI)	É utilizado nos sistemas de modelos que misturam os estilos baseados em dados e orientados pela busca. O Domínio tem como foco os modelos de interação em sistemas distribuídos, tais como a cotação das ações, serviços, informações meteorológicas ou de sistemas de trânsitos. Desenvolvido por Xiaojun Liu e Zhao Yang
Processo de Comunicação Sequencial - *Communication Process Sequence* (CSP)	Permite que os atores representem simultaneamente a execução de processos, implementados como threads Java. Onde os processos se comunicam de forma "atômica". Uma característica chave do domínio CSP é a sua capacidade para modelar interações em etapa única e interrupta, além de incluir uma extensão experimental cronometrada. Criado por [Smyth98].

Tempo Contínuo – *Continuous Time* (TC)	Inclui um conjunto extensível de solucionadores de equações diferenciais. O domínio, portanto, é útil para modelar sistemas físicos com Álgebra linear ou não linear, descrições de equações diferenciais, circuitos analógicos e muitos sistemas mecânicos. Criado por [Liu98].
Distribuição de Eventos Discretos – *Distributed Discrete Events* (DDE)	É utilizado como uma variação dos domínios DE e PN, abordando a imposição de um ponto central de controle na fila global do evento em um modelo, limitando enormemente a capacidade de distribuir um modelo em uma rede. Criado por [Davis00].
Fluxo de Dados Dinâmicos – *Dynamic Data Flow* (DDF)	É a coleção dos domínios (SDF) e (BDF), onde um ator pode alterar as taxas de produção e consumo após cada disparo de evento. Criado por [Zhou08].
Tempo Discreto – *Discrete Time* (DT)	Manipula sistemas de *feedback* da mesma forma que a SDF faz, mas com restrições adicionais em fichas iniciais.
Máquina de Estados Finitos – *Finite State Machine* (FSM)	As entidades neste domínio não representam atores, mas sim o estado, e suas conexões representam as transições entre os estados. A execução é uma seqüência estritamente ordenada de transições de estado. Criado por [Liu03].
Synchronous dataflow (SDF)	É um caso especial de fluxo de dados, onde o fluxo de controle é suficientemente regular para ser completamente previsível em tempo de compilação. Criado por Steve Neuendotffer.

3.4.1.3 Arquitetura do Framework Ptolemy II

O Ptolemy II é uma arquitetura modular e escalável baseada na tecnologia Java. A arquitetura oferece uma infra-estrutura unificada que permite a implementação de um grande número de modelos computacionais.

Fazem parte desta arquitetura, conforme [Bhattacharyya05] os pacotes: *core packages, UI packages, library packages e domain packages*. Onde:

- *Core Packages*: Formam a estrutura de classes central do Ptolemy II, suportando os modelos de dados, ou sintaxe abstrata dos diagramas definidos pelo usuário. Seus principais pacotes são: *kernel, data, actor, copernicus, graph, math, mathlab e util*;
- *UI packages*: Oferecem suporte tanto para a manipulação de arquivos XML do framework (MoML), como também para a interface gráfica (Vergil) utilizada na construção dos workflows. Seus principais pacotes são: *actor.gui, gui, media, moml e vergil*;
- *Library Packages*: São Bibliotecas utilizadas para definir atores polimórficos, uma vez que eles são capazes de operar independentemente em um grande número de domínios. Seus principais pacotes são: *actor.lib.comm, actor.lib.conversions, actor.lib.gui, actor.lib.hoc, actor.lib.image, actor.lib.io, actor.io.comm, actor.lib.jai, actor.lib.javasound, actor.lib.jmf, actor.lib.jxta, actor.lib.logic, actor.lib.net e actor.lib.python.*
- *Domain Packages*: São os pacotes dos domínios de aplicação, onde cada um deles pode implementar um único modelo de computação.

Outras Caracteristicas relevantes sobre a Arquitetura Ptolemy II são:

- Sistema extensível a uma linguagem de interpretação;
- Pacote gráfico (apresenta vários tipos de algoritmos gráfico-teóricos que são utilizados no sistema por programadores nos domínios individuais);
- *Code Generation* (Mecanismo de transformação dos modelos de forma sistemática por meio de gráfico de reescrita, usado para a criação de *Applets* e outras aplicações).

3.4.1.4 Interface Gráfica (Vergil)

O projeto Ptolemy inclui uma ferramenta gráfica (Vergil), interface extensível ao usuário criada por Steve Neuendotffer, utilizada para executar as simulações com melhor usabilidade [Brito03]. Na Figura 16 é apresentada a tela de modelagem na ferramenta Vergil.

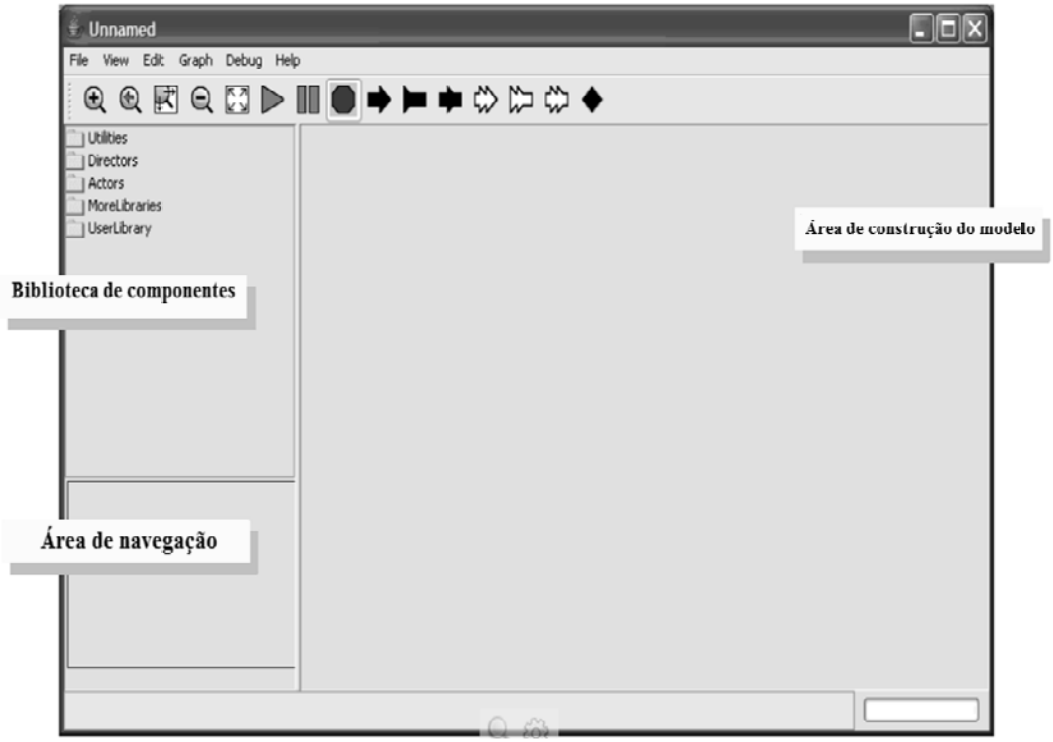

Figura 16: Modelagem no Vergil

Ainda sobre a Figura 16, podemos verificar que á esquerda localizam-se os atores, diretores e utilitários disponíveis para serem utilizados em qualquer modelo. As modificações implantadas no Vergil são descritos no capítulo sobre a extensão do Ptolemy para fins educacionais.

3.5 Considerações finais

Dentre as técnicas disponíveis para avaliar o desempenho dos sistemas computacionais, a simulação é a mais flexível e de baixo custo. Para o desenvolvimento de uma simulação é necessário abstrair o sistema em um modelo computacional, podendo seguir os seguintes enfoques: evento, processo, atividade e atores. A simulação deve representar o sistema com fidelidade, por isso o processo de desenvolvimento de uma

simulação deve seguir técnicas, metodologias e atividades, divididas em três fases: desenvolvimento, testes e análises.

O Modelo Orientado a Atores se qualifica como uma ferramenta adequada para a modelagem do ambiente foco deste trabalho, onde a concorrência e a orientação a mensagens possibilitam que os atores modelados, possam ser organizados de forma que vários atores interajam trocando mensagens, reagindo e evoluindo de acordo com os estímulos gerados pelos próprios atores do mesmo ambiente de forma Fazendo do Ptolemy II uma ferramenta com um grande potencial ainda pouco explorado como ferramenta educacional. Pois, Segundo [Brito09], uma vez que os atores tenham sido bem modelados e desenvolvidos, professores podem gerar ambientes multidisciplinares que mostrem aos alunos de forma interativa como os atores se relacionam e geram efeitos no ambiente e nos atores vizinhos.

4 O Ptolemy como Ferramenta de Ensino de Arquitetura de Computadores

Neste capítulo são apresentados e discutidos conceitos fundamentais relacionados à utilização de ferramentas no ensino de Arquitetura de Computadores com enfoque na interface hardware/software e na aprendizagem em sala de aula.

4.1 Visão geral

O ensino da disciplina de Arquitetura de Computadores apresenta-se como um grande desafio na formação de um profissional da área da Computação. O foco desta disciplina está relacionado com a necessidade do conhecimento de como a organização do hardware pode influenciar a construção de sistemas computacionais.

De forma a sanar tal lacuna, as ferramentas voltadas para o ensino de Arquitetura de Computadores tem possibilitado um grande acervo de aplicações educacionais onde conceitos discutidos e apresentados em sala de aula, podem ser detalhados e apresentados de forma gráfica. Para Mattos [Mattos99], o uso deste recurso, quando convenientemente explorado, permite a construção de elementos que facilitam a comunicação eficaz entre o aluno e o professor.

4.2 Contexto e Ambiente de Utilização

Tem-se observado nos últimos anos um aumento significativo na utilização e complexidade dos sistemas de informação, principalmente na complexidade de especificação, projeto, implementação, validação, programação e aprendizado da teoria dos sistemas computacionais [Martins03].

Consequentemente existe um grande número de estudantes de cursos de computação tradicionalmente voltados ao desenvolvimento de soluções em software, que às vezes apresentam indiferença às disciplinas da área de hardware. Esta situação talvez seja justificada pelo escasso tempo normalmente destinado a estas disciplinas durante os cursos de graduação [Sobreira07].

Analisando tais fatos, podemos destacar alguns problemas importantes que foram a motivação inicial desse trabalho, como:

- Aumento da complexidade, arquitetural e tecnológica, dos sistemas computacionais;
- Falta de motivação dos alunos para o estudo da teoria e a não aplicação dos conhecimentos adquiridos em atividades práticas realísticas [Martins03];
- Dificuldade para ensino e principalmente para aprendizado dos conceitos, características, técnicas relacionadas e exemplos de sistemas computacionais quando se usavam apenas os métodos tradicionais baseados em leitura da teoria e alguns exercícios manuais;
- Deficiências e inadequação dos métodos didáticos tradicionais usados atualmente no ensino e aprendizado de Arquitetura de Computadores.

Logo, diante do processo crescente no uso de tecnologias aplicadas à educação torna-se marcante a presença de ferramentas educacionais que possibilitem potencializar o ensino dos mais diversos conceitos, mas é importante ressaltar não só a contribuição dos recursos computacionais existentes, como também, a necessidade de estudar-se a forma ideal para sua aplicação, com a utilização correta de métodos e técnicas de aprendizagem.

4.3 Metodologias e Técnicas de aprendizagem

Muitas técnicas e métodos de ensino de Organização e Arquitetura de Computadores podem ser utilizados para sua aprendizagem, como por exemplo: o aprendizado baseado em problemas – *Problem Based Learning* (PBL) e a simulação de sistemas concorrentes baseadas em atores, que abordam as seguintes características:

- Expositiva: pode ser implementada através de hipertexto e a utilização de animações para descrever os conceitos fundamentais [Ferreira03];
- Demonstrativa: visualização de exemplos práticos do uso dos conceitos introduzidos [Ferreira03];
- Interativa e lúdica: realização de escolhas pelo usuário, elaboração de descrição ou realização de cálculo [Ferreira03] [Brito09];
- Prática: geração de projeto [Ferreira03];
- Instrucionista: modo passivo e sem postura analítica e crítica para o recebimento de conhecimento [Martins03];
- Didática: baseado na participação de aulas teóricas, leitura da teoria e realização de exercícios manuais [Martins03];
- Construtivista: os alunos podem criar suas próprias arquiteturas [Brito09];

Segundo [Souto09], diversas ferramentas para o auxilio ao ensino de Organização e Arquitetura de Computadores vem sendo desenvolvidas, porém existe uma dificuldade de se encontrar um processo para validação da aprendizagem que permita ao professor acompanhar o aprendizado de cada um dos alunos e medir o seu conhecimento prévio.

4.3.1 Aprendizado Baseado em Problemas – PBL

É uma abordagem na qual o aprendizado é auto dirigido e centrado no estudante, que participa como um construtor do seu próprio conhecimento. No PBL, conforme [Sobreira07] o aluno é que gerencia o ritmo do aprendizado de acordo com o seu interesse, possuindo liberdade na escolha das estratégias que podem vir a ser aplicadas na solução de seus projetos [Christensen06].

4.3.2 Simulação de sistemas concorrentes baseada em atores

Para [Brito09] a simulação de sistemas concorrentes baseada em atores possui um grande potencial educacional ainda pouco explorado. Onde um modelo de atores é definido como um modelo matemático de computação concorrente que trata “atores” como sendo os elementos primitivos da computação concorrente digital [Lee 2001].

Nesta abordagem a concorrência implica em dizer que os atores serão executados simultaneamente e que a orientação é o mecanismo no qual os atores serão configurados para receber mensagens em suas portas, reagir a essas mensagens e gerar novas mensagens na saída para outros atores.

4.4 Potencial do uso do Ptolemy como ferramenta educacional

O Ptolemy não é uma ferramenta preparada para o auxilio ao ensino de Organização e Arquitetura de Computadores. Entretanto, como ele é uma ferramenta de código aberto, as modificações sugeridas buscam torná-lo uma ferramenta educacional.

Para verificar se uma ferramenta atende os requisitos para ser caracterizada como um software educacional, algumas características de qualidade técnica e aspectos educacionais devem ser levantados, como citado em [Castro08]:

- Características pedagógicas: viabilidade de utilização do software em situações educacionais, adaptabilidade a metodologia de ensino e adequação a uma proposta de educação mais construtivista;
- Tradução: preparação do software para internacionalização;
- Abrangência: Capacidade de execução em mais de uma plataforma (Linux, Windows, etc.);
- Atividades sugeridas: a concepção do propósito de sua utilização. Onde se procura dar exemplos e fazer experiências com o software em situações educacionais;
- Usabilidade: facilidade de uso do software (clareza nos comandos pedidos pelo programa, presença de recursos e meios de interação do usuário com o software, mensagens de erros claros e indicadores do caminho correto a ser seguido pelo aluno), manual disponível e a possibilidade de configuração pelo usuário ou professor.

4.4.1 Problemas

Conforme descrito no capitulo e comentado por [Brito09], existem algumas barreiras para a utilização da ferramenta, que variam desde a quantidade de desenvolvedores avançados na equipe Ptolemy [Gordenis05] até as limitações de sua arquitetura.

De forma a se obter um bom retorno do potencial da ferramenta e procurar sanar tais problemas, foram verificadas para o projeto as seguintes necessidades:

a) Conhecimento da documentação do sistema;

b) Conhecimento de Java e XML;

c) Conhecimento dos domínios e atores;

Com relação ao item *a*, sabe-se que a operação correta de qualquer software computacional, passa pelo conhecimento de sua documentação (manuais, ajudas, descrição de funcionalidades, etc.). Entretanto, em virtude do público alvo do Ptolemy, as funcionalidades são muito variadas e em constante ampliação, tornando essa necessidade ainda maior.

Tratando-se do item *b*, o conhecimento de Java e XML serão primordiais para a modelagem de novas tecnologias ou sistemas, ou ainda para aplicação da ferramenta em situações específicas, como no ensino de arquitetura de computadores, pois haverá a necessidade de criação de novos atores.

E por fim, para efetuar simulações e modelagens, utilizando-se da grande biblioteca de atores já criados, precisa-se compreender os domínios, que implementam os modelos computacionais, definindo a semântica das relações entre os atores, e compreender propriamente os atores, para utilização correta de suas portas e parâmetros.

De forma a tratar tais questões, buscou-se através da participação do aluno Adelson Lopes da Nóbrega do Centro Universitário de João Pessoa – UNIPÊ, relatar os problemas encontrados através do uso de critérios de usabilidade/ergonomia. Sendo assim possível avaliar a ferramenta [Nóbrega11] e verificar as mudanças necessárias para que o Ptolemy II esteja apto para sua utilização como ferramenta de ensino de Arquitetura de Computadores.

4.4.2 Usabilidade no Ptolemy II

A Usabilidade representa basicamente o quão fácil é o usar o software, sendo esta a característica mais difícil de ser tratar [Koscianski07], devido a esta ser a característica que envolve a maior carga de fatores subjetivos durante sua análise. Por isso Cybis declara [Cybis03]:

> Um sistema pode proporcionar boa usabilidade para um usuário experiente, mas péssima para novatos, ou vice e versa. Pode ser fácil de operar se o sistema for usado esporadicamente, mas difícil, se for utilizado frequentemente, no dia a dia. Uma interface bonita pode dar prazer se o site for acessado por conexões rápidas, mas causar ansiedade insuportável se ele for acessado de casa, via modem.

Nesta seção é efetuada a análise da usabilidade do Ptolemy II, buscando compreender e apontar as necessidades do software para torná-lo uma boa ferramenta de ensino, para auxílio na disciplina de arquitetura de computadores, mais simples e mais amigável.

4.4.2.1 Avaliação através do Ergolist

Na realização da avaliação de usabilidade, foi utilizada a técnica diagnóstica de inspeção via *checklist*, com o auxílio da ferramenta *Ergolist*[1], desenvolvida pelo Laboratório de Utilizabilidade da UFSC/SENAI (LabiUtil), coordenado pelo Prof. Dr. Walter de Abreu Cybis. Essa ferramenta possui dezoito *checklists*, totalizando 193 questões.

Na Figura 17 é visualizado um gráfico que ilustra as estatísticas da avaliação Ptolemy II pela ferramenta Ergolist. No eixo horizontal apresenta os critérios de avaliação e o eixo vertical apresenta o percentual de questões conformes, não conformes e não aplicáveis em cada critério.

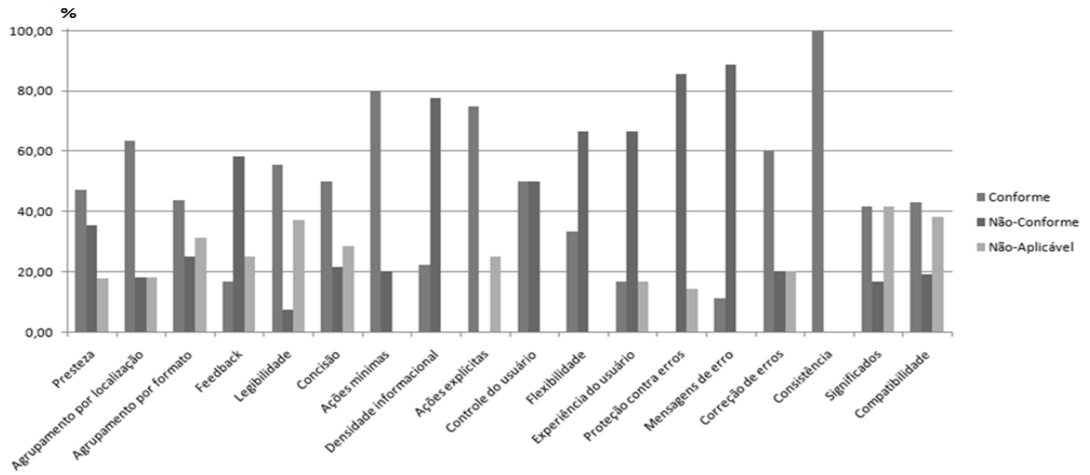

Figura 17: Avaliação da Usabilidade do Ptolemy

[1] Para acessar o Ergolist, visite: http://www.labiutil.inf.ufsc.br/ergolist/check.htm

Após a avaliação, podem-se destacar problemas de usabilidade nos seguintes critérios, cujo percentual de questões não-conformes se sobressaíram:

- *Feedback*: esse critério avalia a qualidade do *feedback* imediato às ações do usuário e o Ptolemy II apresentou quase 60% de questões não-conformes, tendo em vista que, por existir uma documentação muita extensa, muitas tarefas executadas não apresentam um *feedback* apropriado para o usuário ficar mais seguro do que está fazendo;
- Densidade informacional: esse critério avalia a carga de memorização que o usuário está exposto para operar um software e necessidade de executar tarefas complexas. Nesse critério o Ptolemy II apresentou quase 80% de questões não-conformes, pois apresenta uma vasta biblioteca de atores, domínios e utilitários, para simular diversos sistemas heterogêneos, de complexidades variadas;
- Flexibilidade: o sistema deve disponibilizar diversas formas de se realizar uma tarefa e possibilitar efetuar modificações. Com quase 67% de questões não-conformes, no Ptolemy II só é possível essa flexibilidade se o usuário conhecer Java e XML;
- Experiência do usuário: avalia se os usuários com diferentes níveis de experiência têm iguais possibilidades de obter sucesso no sistema. Novamente com 67% de questões não-conformes, a avaliação mostra que os usuários iniciantes terão dificuldades no Ptolemy II;
- Proteção contra erros: esse critério diz respeito aos mecanismos de detecção e prevenção de erros de entrada de dados ou comandos e ações não recuperáveis. A boa usabilidade diz que é preferível detectar os erros na digitação, do que na validação, por isso o Ptolemy II obteve mais de 85% de questões não-conformes, pois a maioria dos erros é detectada na validação dos arquivos XML e classes Java;
- Mensagens de erro: a qualidade das mensagens favorece o aprendizado, informando a natureza do erro cometido e sobre as ações para correção. Nesse critério o Ptolemy II alcançou o pior resultado com quase 89% de questões não-conformes (ANEXO A), visto que a maioria das mensagens

de erros são exceções Java, exibindo para o usuário uma pilha de classes a depurar.

Importante comentar que o Ptolemy II foi avaliado por quatro alunos mestrandos que utilizaram o Ptolemy como ferramenta meio ou fim em suas dissertações de mestrado aplicadas ao Programa de Pós Graduação em Informática (PPGI), e por um aluno de graduação que participou relatando os problemas encontrados através do uso de critérios de usabilidade/ergonomia. Além disso, convém salientar a importância da avaliação seja feita por outros avaliadores, fazendo comparação dos resultados, chegando num parecer mais coeso e reduzindo a subjetividade que pode prejudicar as avaliações individuais.

4.4.2.2 Melhorias

Após analisar as características do Ptolemy citadas neste trabalho, foi verificado seu potencial para o ensino, devido a permitir ao aluno e professor simular ambientes construídos pelos mesmos e ter a possibilidade de avaliar o conhecimento para o tema estudado na disciplina de Organização de Arquitetura de Computadores. Entretanto, o ambiente Ptolemy II ainda necessita de algumas adequações, como:

a) Melhorias na usabilidade da interface;
b) Criação de biblioteca com atores apropriados à disciplina;
c) Inclusão de opção para correção de exercícios;

Com relação ao item *a*, o objetivo é tornar sua interface mais amigável, com instruções passo a passo em português e a montagem de seus ambientes na forma de atividades, além de melhorar o layout dos atores criados para que fiquem visualmente semelhantes ao componente a ser simulado;

Tratando-se do item *b*, inicialmente foram desenvolvidos atores, seguindo a metodologia aplicada por [Patterson et al05], que simulassem o ciclo de processamento da arquitetura MIPS e execução de aplicações reais sobre o simulador.

Uma vez que os atores tenham sido modelados, ambientes podem ser organizados de forma que vários atores interajam trocando mensagens, reagindo e evoluindo de acordo com os estímulos gerados pelos próprios atores do mesmo ambiente de forma independente. Tudo isso pode ser exibido aos alunos passo-a-passo,

com animações, uso de diferentes cores, desenhando gráficos e reproduzindo sons, gerando recursos mais atrativos [Brito09].

A Figura 18 representa os componentes principais da arquitetura MIPS, onde cada um contém portas de entrada/saída e características próprias que serão detalhadas posteriormente.

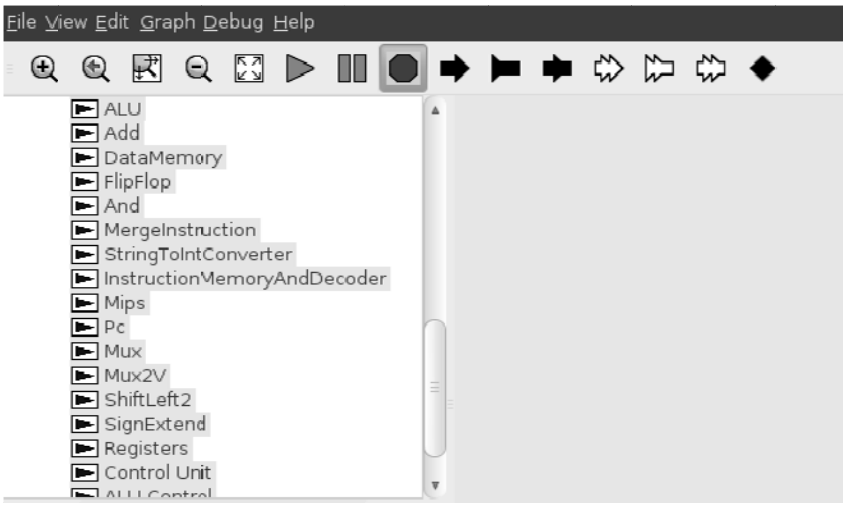

Figura 18: Repositório de atores referente ao processador MIPS

Na área das bibliotecas, ficarão disponíveis apenas os atores e domínios utilizados nas simulações para o conteúdo da disciplina de Arquitetura de Computadores.

O item *c* trata da criação de um método de verificação automática que possibilite oferecer *feedback* e mensagens de erros claras e indicadoras do caminho correto a ser seguido pelo aluno. O projeto *PtolemyModelChecker* desenvolvido em Java tem essa finalidade, possibilitando verificar na atividade respondida pelo aluno alguns pontos avaliativos do roteiro didático criado pelo professor. É Importante comentar que essa atividade se dá pela verificação dos arquivos XML que são gerados pelo Ptolemy para os ambientes simulados.

As melhorias desenvolvidas pelo *PtolemyModelChecker* são:

- Verificar os atores que são mandatórios: o professor poderá informar quais atores deverão está presentes na atividade do aluno, lhe informando os atores pendentes no cenário;

- Verificar as relações entre os atores: funcionalidade na qual são verificadas as conexões dos atores no cenário do professor e comparados com as conexões da atividade do aluno. Tais relações são muito importantes para que os alunos não conectem um ator numa porta indevida, podendo prejudicar o resultado final da simulação;
- Verificar as propriedades dos atores: possibilita verificar se o valor da propriedade do ator no cenário do aluno é equivalente (seguindo regras pré estabelecidas para a propriedade) ao ator no cenário do professor;
- Incorporar o *PtolemyModelChecker* a nova interface.

Com as melhorias propostas e implantadas no Ptolemy II, a ferramenta possuirá as seguintes características sugeridas para uma ferramenta educacional:

- Clareza nos comandos pedidos pelo programa.
- Controle das sequências reprodutoras do evento pelo aluno, facilitando a simulação da realidade.
- Facilidade de leitura da tela.
- Apresentação dos resultados ao aluno tanto parcialmente quanto ao final da simulação.
- Possibilidade de inclusão de novas estruturas e/ou segmentos de programa a fim de manter o conteúdo sempre atualizado e reproduzindo a realidade.
- Possibilidade de correção de erros realizados pelo aluno e detectados pelo próprio antes do registro.

4.5 Considerações finais

Simuladores didáticos são amplamente utilizados na disciplina de Arquitetura de Computadores para mostrar aos alunos, de uma forma prática, o funcionamento interno de determinadas arquiteturas. Entretanto, ainda há necessidade de pesquisas para auxiliar o processo de ensino, como metodologias, abordagens e ferramentas similares.

A abordagem utilizando o Ptolemy como ferramenta de ensino de Arquitetura de Computadores com a simulação baseada em atores, possibilitará que os alunos

assimilem mais facilmente os conteúdos ministrados em sala de aula, pois terão a oportunidade de exercitar nas aulas práticas os modelos passados nas aulas teóricas.

Permanecendo a situação atual (sem o auxílio da ferramenta Ptolemy com as modificações sugeridas), o aluno irá estudar o conteúdo da disciplina através de aulas teóricas ou, no máximo, utilizando ferramentas que não oferecem os recursos adequados ao aprendizado.

O desenvolvimento do projeto proporcionará aos alunos um ambiente amigável e de fácil usabilidade para que os mesmos possam, de forma construtiva, confeccionar cenários descritos em sala de aula e aperfeiçoá-los de acordo com o objetivo didático esperado pelo professor. Além de oferecer um método de verificação e avaliação dos cenários em conformidade com o tema abordado pela disciplina.

5 Desenvolvimento e Cenário de Aplicação da ferramenta

Este capítulo apresenta detalhes do desenvolvimento da proposta de uma nova interface gráfica e do método de verificação/avaliação automática do roteiro didático, incluindo a apresentação das principais tecnologias envolvidas, bem como as justificativas para uso das mesmas. Na fase de implementação, é descrito o novo ambiente criado com o software, que pode permitir novas abordagens no desenvolvimento de atividades na modelagem educacional.

5.1 Especificação da ferramenta

Anteriormente à fase de desenvolvimento de um sistema, ocorre a fase de análise onde são levantados os requisitos do sistema para atender às necessidades do usuário. No caso específico do Ptolemy, não foi feito um levantamento das necessidades junto a um usuário especificamente, mas um estudo do seu uso para fins educacionais.

Para [Koscianski07] os padrões de projeto para interface apresentam métodos de interação com os usuários, exibindo informações, expondo funcionalidades ou solicitando entradas de dados. Onde são representadas maneiras de responder aos problemas de alguns aspectos importantes, como:

- Encadeamento de ações para o usuário;
- Forma de apresentação de informações;
- Forma de atrair a atenção e enfatizar informações;
- Forma de expor funcionalidades.

O ambiente Ptolemy insere-se no contexto apresentado disponibilizando um ambiente de desenvolvimento integrado – *Integrated Development Environment* (IDE) que possibilita a modelagem, execução e simulação de vários ambientes heterogêneos. Podendo o Ptolemy ser dividido em dois módulos: i) Modelagem: permite ao usuário

criar cenários utilizando o repositório de atores, configurá-los e utilizá-los em vários domínios; e ii) Simulação: permite simular os cenários desenvolvidos.

5.2 Modificações introduzidas no Ptolemy

A hipótese dessa dissertação é de que a disponibilidade de modelos pré-determinados, de uma interface amigável e de um método de verificação automática trará uma maior facilidade ao trabalho do professor para criar roteiros didáticos e atividades de modelagem na disciplina de Organização e Arquitetura de Computadores.

O Ptolemy, por se tratar de um framework de código aberto e desenvolvido em Java, permitiu introduzir as modificações que foram feitas seguindo as características de um software educacional [Castro08] e sugestões a serem seguidas no projeto de uma interface [Koscianski07].

5.2.1 Aperfeiçoamento do Vergil para fins didáticos

Um ponto inicial a ser revisto na interface Vergil foi à tradução do software para o português, visando facilitar sua utilização por usuários com pouca experiência na língua inglesa e preparação do software para internacionalização.

Na tela inicial do Vergil (Figura 19), a barra de menu (opções *File* e *Help*) foi modificada para a língua portuguesa, editando o arquivos contidos nas pastas **ptolemy.gui** (*top.java*) e **ptolemy.configs.full** (*intro.htm*), em seguida foram retirados da opção (*File - > New*) alguns tipos de modelos computacionais que representam sistemas.

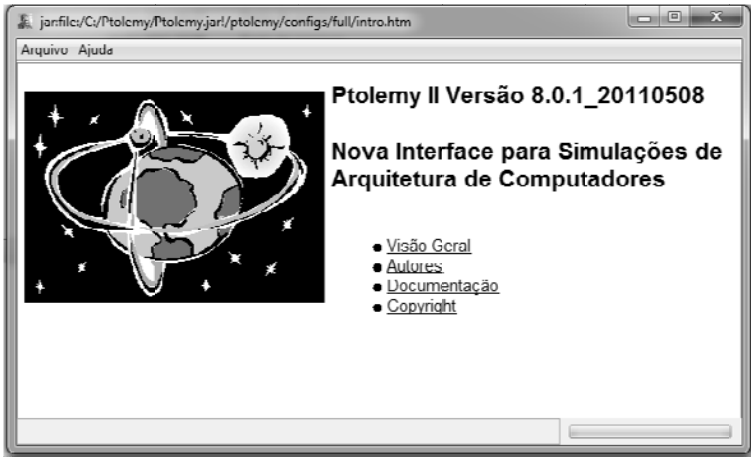

Figura 19: Tela inicial do Vergil

Para a opção *New* foram traduzidas e mantidas apenas as opções *Graph Editor* e *Text Editor*, retirando as demais opções. Essa modificação se dá para reduzir a quantidade de informação que os usuários precisam dominar. Como também, utilizar apenas a funcionalidade de modelagem e simulação.

Os demais modelos não serão usados devido a serem específicos para outras atividades na ferramenta Ptolemy, como: criação de ícones para os atores (*Icon Editor*) e criação de máquinas de estados (*FSM Editor*, *Modal Model*, *Ptera Model*), descrição em UML (*Model Transformation*) e não fazem parte do escopo do trabalho.

Para a remoção das chamadas das classes Java foi necessário editar o arquivo *defaultFullConfiguration. xml* (contido em **ptolemy.configs**), conforme a Figura 20.

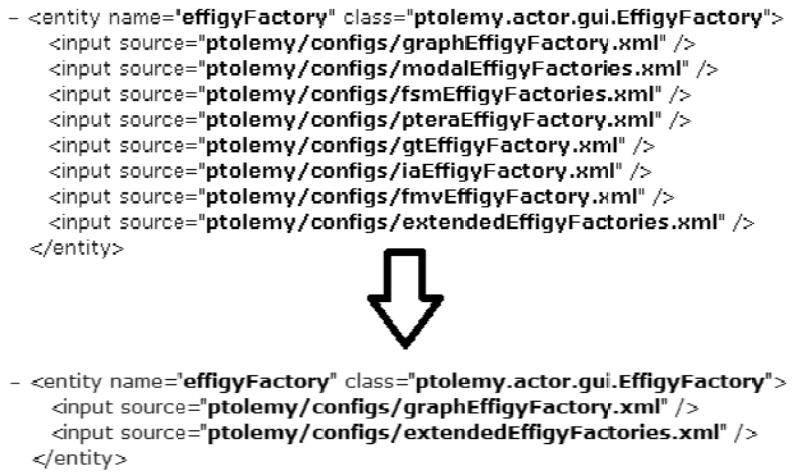

Figura 20: Edição do arquivo *defaultFullConfiguration.xml*

Foram mantidas as entradas dos seguintes arquivos XML:

- Modelagem e Simulação: referente à chamada do menu *Graph Editor* (**ptolemy.configs.graphEffigyFactory.xml**);
- Editor de Texto: referente à chamada do menu *Text Editor* (**ptolemy.configs.extendedEffigyFactories.xml**)

Após as mudanças na tela inicial do Vergil, o próximo passo foi modificar o ambiente de modelagem e simulação.

Seguindo as recomendações exposta em [Koscianski07], foram utilizados alguns padrões para efetuar tais mudanças, como:

- Quantidade de comandos: redução do número de opções disponíveis simultaneamente ao aluno;
- Disposição: tornar clara ao usuário à diferença entre rótulos e entradas de dados;
- Vocabulário: revisão das telas verificando se não há erros de digitação ou de português, com termos claros e tecnicamente corretos;
- Filtro: permitir a um usuário localizar uma informação entre um grande número de possibilidades.

O ambiente de modelagem e simulação do Vergil é composto por: a) Bibliotecas de componentes; b) Barra de menu; c) Barra de ferramentas (ícones); d) Área de navegação; e) Área de construção de modelos; e f) opções (botão direito do mouse).

Primeiramente, em (*a)* foi alterado o nome das pastas (*Utilities, Directors, Actors, MoreLibraries e UserLibrary*) que fazem parte das bibliotecas de componentes (Figura 21), editando o arquivo *basicLibrary.java* (**ptolemy.configs**).

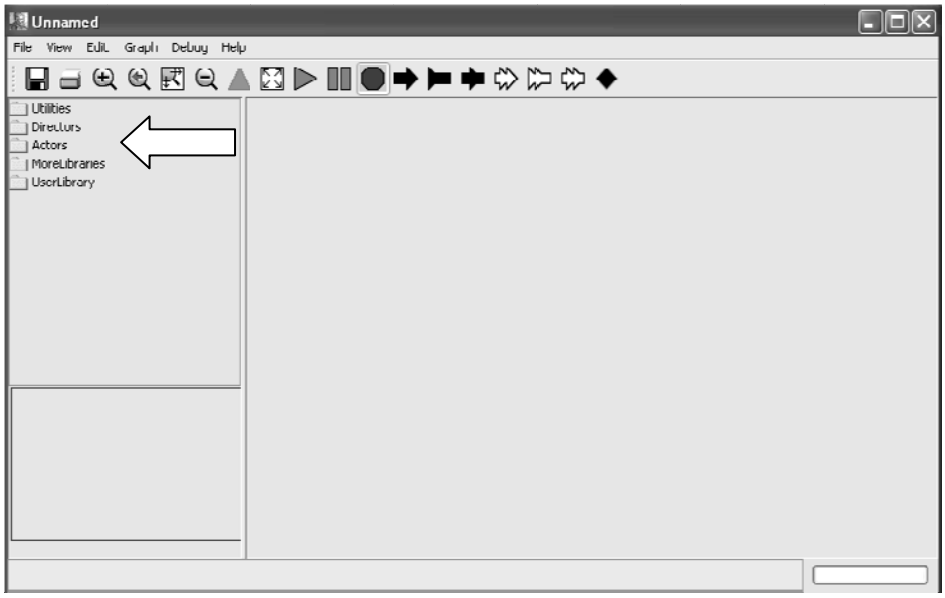

Figura 21: Biblioteca de componentes

Ainda sobre o item *(a)*, foram adicionados atores na pasta Atores - >Arquitetura. Logo após, foram declarados no arquivo *arquitetura.xml* (**ptolemy.actor.lib.arquitetura**), conforme a Figura 22.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE plot PUBLIC "-//UC Berkeley//DTD MoML 1//EN"
    "http://ptolemy.eecs.berkeley.edu/xml/dtd/MoML_1.dtd">
<entity name="Arquitetura" class="ptolemy.moml.EntityLibrary">
  <configure>
    <?moml
        <group>
        <doc>Atores criados por André Lucena</doc>
    <entity name="ALU" class="ptolemy.actor.lib.arquitetura.ALU"></entity>
    <entity name="Add" class="ptolemy.actor.lib.arquitetura.Add"></entity>
    <entity name="DataMemory" class="ptolemy.actor.lib.arquitetura.DataMemory"></entity>
    <entity name="FlipFlop" class="ptolemy.actor.lib.arquitetura.FlipFlop"></entity>
    <entity name="And" class="ptolemy.actor.lib.arquitetura.And"></entity>
    <entity name="MergeInstruction" class="ptolemy.actor.lib.arquitetura.MergeInstruction"></entity>
    <entity name="InstructionMemoryAndDecoder" class="ptolemy.actor.lib.arquitetura.InstructionMemoryAndDecoder"></entity>
    <entity name="Mips" class="ptolemy.actor.lib.arquitetura.Mips"></entity>
    <entity name="Pc" class="ptolemy.actor.lib.arquitetura.Pc"></entity>
    <entity name="Mux" class="ptolemy.actor.lib.arquitetura.Mux"></entity>
    <entity name="Mux2V" class="ptolemy.actor.lib.arquitetura.Mux2V"></entity>
    <entity name="ShiftLeft2" class="ptolemy.actor.lib.arquitetura.ShiftLeft2"></entity>
    <entity name="SignExtend" class="ptolemy.actor.lib.arquitetura.SignExtend"></entity>
    <entity name="Registers" class="ptolemy.actor.lib.arquitetura.Registers"></entity>
      <entity name="Control Unit" class="ptolemy.actor.lib.arquitetura.ControlUnit"></entity>
      <entity name="ALU Control" class="ptolemy.actor.lib.arquitetura.AluControl"></entity>
     </group>
    ?>
  </configure>
</entity>
```

Figura 22: Arquivo *Arquitetura.xml* para declaração de atores

Para ficarem disponíveis na biblioteca de componentes, dois arquivos precisam conter a declaração do arquivo *arquitetura.xml*. O arquivo *makefile* (Tabela 4) criado na pasta (**ptolemy.actor.lib.arquitetura**) declara a chamada do arquivo e as classes Java referentes aos atores. E o arquivo *basicActorLibrary.xml* (Tabela 5) localizado na pasta (**ptolemy.configs**).

Tabela 4: Makefile

```
#  PT_COPYRIGHT_VERSION_2

#  COPYRIGHTENDKEY

ME = ptolemy/actor/lib/arquitetura

DIRS =   test

# Root of the Java directory

ROOT =        ../../../..

CLASSPATH =    $(ROOT)

# Get configuration info

CONFIG =       $(ROOT)/mk/ptII.mk

include $(CONFIG)

# Used to build jar files
```

```
PTPACKAGE =     arquitetura

PTCLASSJAR =    $(PTPACKAGE).jar

# Keep this list alphabetized.

JSRCS = \

        Add.java \

        ALU.java \

        AluControl.java \

        And.java \

        ControlUnit.java \

        DataMemory.java \

        FlipFlop.java \

        GerenteDeExecucao.java \

        InstructionMemoryAndDecoder.java \

        MergeInstruction.java \

        Mips.java \

        Mux.java \

        Mux2V.java \

        MuxSemClock.java \

        Pc.java \

        Registers.java \

        Shiftleft2.java \

        SignExtend.java

OTHER_FILES_TO_BE_JARED = \

        arquitetura.xml

EXTRA_SRCS =   $(JSRCS) $(OTHER_FILES_TO_BE_JARED)

# Sources that may or may not be present, but if they are present, we don't
```

```
# want make checkjunk to barf on them.

# Don't include demo or DIRS here, or else 'make sources' will run 'make demo'

MISC_FILES = test

# make checkjunk will not report OPTIONAL_FILES as trash

# make distclean removes OPTIONAL_FILES

OPTIONAL_FILES = \

        demo \

    doc

JCLASS = $(JSRCS:%.java=%.class)

all: jclass

install: jclass jars

# Get the rest of the rules

include $(ROOT)/mk/ptcommon.mk
```

No arquivo *basicActorLibrary.xml* (Tabela 5) se declara a chamada dos arquivos XML que contem as classes java correspondentes aos atores.

Tabela 5: basicActorLibrary.xml

```
<?xml version="1.0" standalone="no"?>

<!DOCTYPE entity PUBLIC "-//UC Berkeley//DTD MoML 1//EN"

   "http://ptolemy.eecs.berkeley.edu/xml/dtd/MoML_1.dtd">

<group>

  <input source="ptolemy/actor/lib/arquitetura/arquitetura.xml"/>

  <input source="ptolemy/actor/lib/sources.xml"/>

  <input source="ptolemy/actor/lib/sinks.xml"/>

  <input source="ptolemy/actor/lib/array.xml"/>

  <input source="ptolemy/actor/lib/io/io.xml"/>

  <input source="ptolemy/actor/lib/logic/logic.xml"/>
```

```
 <input source="ptolemy/actor/lib/math.xml"/>
 <input source="ptolemy/actor/lib/matrix.xml"/>
 <input source="ptolemy/actor/lib/realtime.xml"/>
 <input source="ptolemy/actor/lib/string/string.xml"/>
</group>
```

Ainda sobre o item *(a)*, a biblioteca *UserLibrary* passou a se chamar Biblioteca do Aluno, seguindo recomendações de [Koscianski07] para adaptabilidade. Ela será usada para que o aluno salve os atores que ele mais utiliza, criando atalho para os atores mais utilizados por ele. Para isso, o mesmo deverá efetuar um clique com o botão direito sobre o ator, clicar sobre a opção *Save Actor In Library* (Salvar na Biblioteca) e salvar o arquivo referente a sua biblioteca.

Em relação ao item *(b)* foram traduzidos para português as opções da barra de menu (Figura 23) e removidos algumas opções:

- No menu *File* foi removido a opção *Open URL* (edição do arquivo *Top.java* em **ptolemy.gui**);
- No menu *View* foram removidos as opções: *Tree View*, *JVM Properties* e *Interface Window* (edição do arquivo *graphTableauFactory.xml* em **ptolemy.configs**);
- No menu *Debug* foi retirado a opção *Listen to Director* contido no arquivo *ActorGraphFrame.java* (**ptolemy.vergil.actor**)

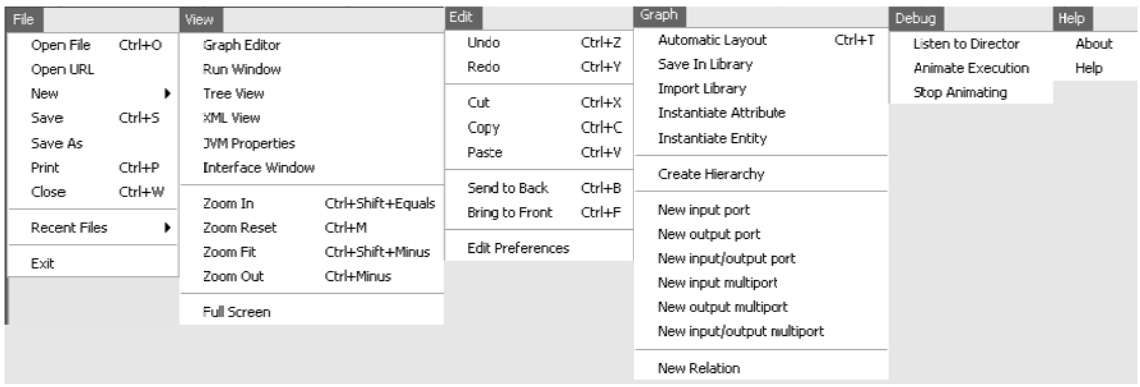

Figura 23: Barra de menu do Vergil (antes e depois)

Ainda sobre o item *(b)*, foi adicionada a funcionalidade Verificação automática (que utiliza o método *PtolemyModelChecker*, melhor descrito na seção 5.2.2). O arquivo *ActorGraphFrame.java* (**ptolemy.vergil.actor**) foi editado incluindo a chamada as opções para o menu com a seguinte linha de comando:

```
_checkerMenu = new JMenu("Verificação Automática")

 _menubar.add(_checkerMenu);
```

No item *(c)* foram alterados os ícones disponíveis na barra de ferramentas (Figura 24), editando os arquivos *RunnableGraphController*, *ActorEditorGraphController* e *BasicGraphFrame*.

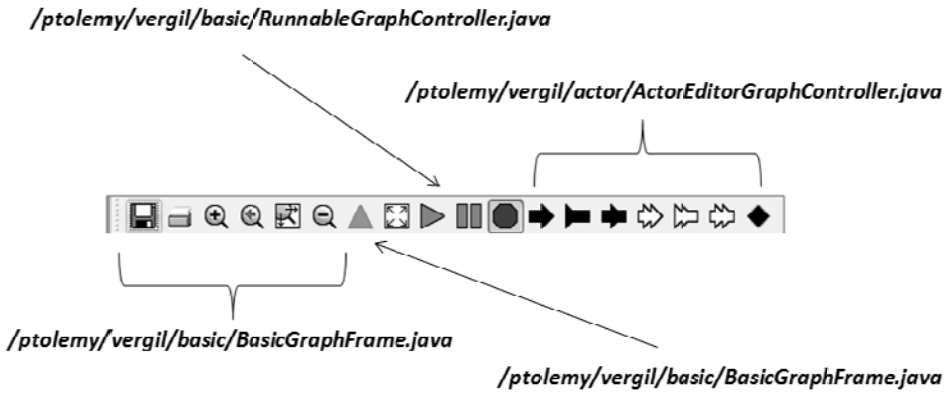

Figura 24: Barra de Ferramentas do Vergil

Para o item *(d)* e *(e)* não foram realizadas mudanças. É importante comentar o padrão “apresentação todo-detalhe” [Koscianski07] presente no item *(d)*, que permite apresentar uma visão geral e, ao mesmo tempo, um zoom sobre a área de interesse.

Finalizando, o item *(f)* é referente às opções disponíveis para comandos rápidos no ator, na porta do ator e no ambiente de modelagem, clicando com o botão direito sobre algum desses itens. Foram editados os arquivos:

/ptolemy/vergil/actor/ActorEditorGraphController.java

/ptolemy/vergil/actor/ActorController.java

/ptolemy/vergil/actor/ActorInstanceController.java

/ptolemy/vergil/basic/BasicGraphController.java

/ptolemy/vergil/basic/GetDocumentationAction.java

/ptolemy/vergil/basic/CustomizeDocumentationAction.java

/ptolemy/vergil/basic/RemoveCustomDocumentation.java

/ptolemy/vergil/fsm/StateController.java

/ptolemy/vergil/modal/StateController.java

/ptolemy/vergil/toolbox/EditIconAction.java

/ptolemy/vergil/kernel/AttributeController.java

/ptolemy/vergil/kernel/ListenToAction.java

A barra de título também foi traduzida, pela substituição da palavra “Unnamed” por “Sem título”. Foram editados os arquivos abaixo:

/ptolemy/actor/gt/controller/View.java

/ptolemy/actor/gt/Model/View.java

/ptolemy/actor/gui/Configuration.java

/ptolemy/actor/gui/Effigy.java

A nova interface (Figura 25) foi implementada na versão 8.0.1 do Ptolemy. Seu desenvolvimento foi na linguagem de programação Java, utilizando o compilador *Eclipse Helios Service Release 2*.

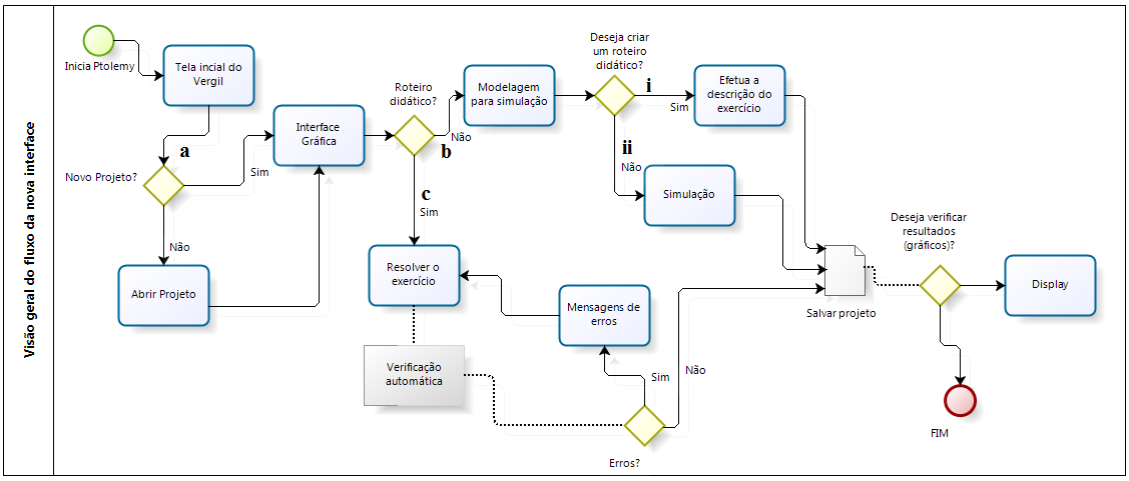

Figura 25: Visão geral da estrutura da nova interface

Conforme [Torres11], ao acessar a interface gráfica o usuário decidirá se deseja abrir um ambiente de modelagem para: *(a)* criar um novo projeto ou abrir um projeto já desenvolvido, podendo este ser *(b)* um projeto salvo no computador ou *(c)* um roteiro didático também salvo.

No caso *(a)* o usuário utilizará a interface para modelagem e simulação, onde terá duas opções: *i)* criar um roteiro didático (Figura 25), utilizando os atores da biblioteca, selecionando os obrigatórios, suas propriedades e as conexões entre eles, além de salvá-lo e/ou simulá-lo ou *ii)* criar seu próprio modelo utilizando a biblioteca, salvá-lo e/ou simulá-lo.

Em relação ao item *(b)* o usuário poderá abrir projetos existentes, adicionar/editar atores, salvá-los e simulá-los por último, o item *(c)* possibilita que ele abra roteiros didáticos existentes, resolvendo os exercícios propostos, verifique possíveis erros através da opção *Verificação Automática* e receba um *checklist* da correção do exercício.

Nas próximas seções, será explicado como funciona a opção de *Verificação Automática* na nova interface.

5.2.2 Adição do método de verificação automática nos cenários

Em geral, além destas barreiras para o ensino da disciplina, o professor encontra dificuldades de avaliar todos os exercícios de todos os alunos em um curto prazo, tornando difícil fornecer um rápido *feedback* e disponibilizar as correções para os alunos. Logo, verificou-se a possibilidade de incluir um método de verificação

automática das atividades desenvolvidas pelo aluno. Tal método efetua um *checklist* avaliativo da atividade do aluno comparando- o com o cenário descrito pelo professor.

Almejasse com a inclusão do método, possibilitar ao aluno um passo a passo da atividade descrita pelo professor. Sendo assim, a presença do professor junto ao aluno não é mais essencial, já que a ferramenta será capaz de gerar mensagens de erros e com isso colaborar para que o aluno desenvolva suas atividades.

A visualização das mensagens de erros pode ser configurada em conformidade com o roteiro didático feito pelo professor, podendo o mesmo informar quais componentes serão checados ou obrigatórios no cenário em questão. Tal possibilidade indica ao professor caminhos para aplicar ao aluno tarefas sequenciais, onde o mesmo terá o nível de dificuldade aumentado com a diminuição de mensagens de erros, ou com cenários mais complexos.

Tal método foi abordado devido ao sucesso obtido em pesquisas sobre o uso de verificação automática [Lino07] [Saikkonen01], em que o próprio sistema é um dos atores do processo [Moreira09]. Os principais benefícios para o educador são entre outros:

- Menor esforço, uma vez que conta com o auxílio da ferramenta;
- Melhor monitoramento dos estudantes e de suas tarefas;
- Melhor rastreamento individual dos estudantes, através das mensagens de erros;
- Melhor qualidade de ensino, devido o maior tempo de prática;

Na fase atual do trabalho, o método utiliza um pacote de classes em Java adicionado a ferramenta Ptolemy ***(ptII8.0.1\br\ufpb\ccae\projetoptolemy\checker),*** verificando os erros na atividade pela comparação de dois arquivos XML. A Figura 26 ilustra o processo no qual o método é utilizado no Ptolemy.

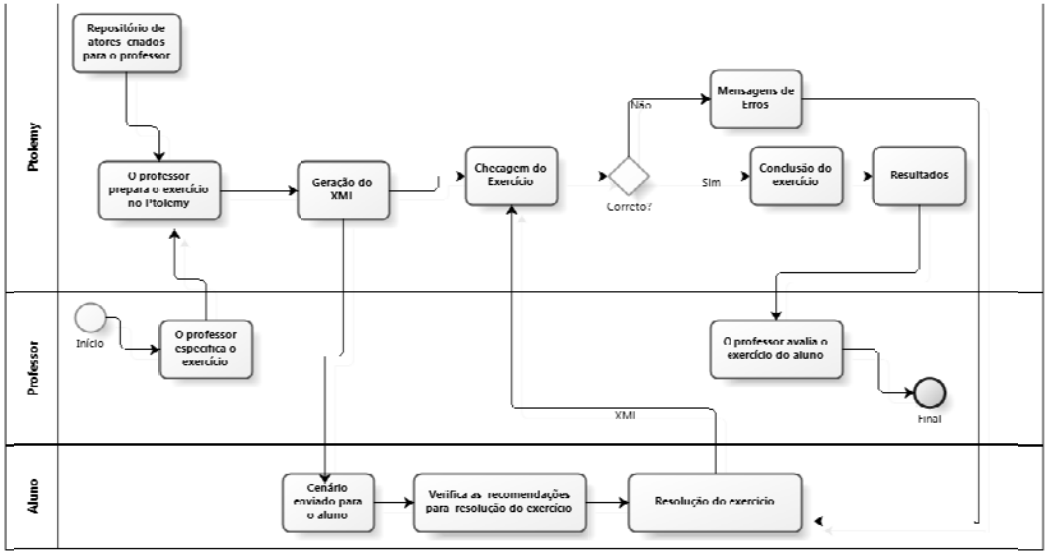

Figura 26: Processo de verificação do roteiro didático

Ainda sobre a Figura 26, o processo inicia-se quando o professor (utilizando o repositório de atores) cria a atividade para o aluno, gerando um arquivo XML que será utilizado pelo aluno como roteiro didático. Após a geração do roteiro, o professor poderá alterar a extensão do arquivo para RESP, formato criado para servir de gabarito ao aluno.

Com as orientações do roteiro, o aluno irá responder a atividade seguindo os seguintes passos: i) introdução dos atores especificados, ii) efetuar as conexões entre os atores e iii) efetuar configuração das propriedades especificadas aos atores. De forma a validar tais informações, a *Verificação Automática* possibilita o *feedback* ao aluno, disponibilizando o *checklist* de acordo com o XML criado pelo professor.

A Figura 27 ilustra as mensagens de erros que servirão para que o aluno conclua seu exercício.

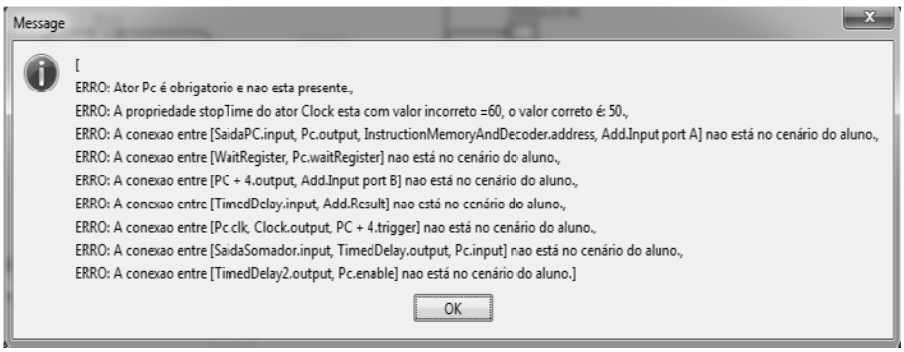

Figura 27: Mensagens dos erros encontrados no cenário

Nas próximas seções, as verificações que geram as mensagens acima são detalhadas para o melhor entendimento do seu funcionamento.

5.2.2.1 Verificação de atores obrigatórios

De forma a verificar se todos os atores definidos pelo professor como obrigatórios estão contidos na atividade desenvolvida pelo aluno, a funcionalidade busca a chamada "*entity*" nos arquivos XML, correspondendo aos atores dos cenários. Tal funcionalidade é efetuada pela classe *Checker.comparaXmlMandatory* que compara os arquivos *("exercicioProfessor.xml","exercicioAluno.xml").*

A Tabela 6 descreve o XML do professor onde foi adicionado manualmente o texto **mandatory="true"** para que o *Checker.comparaXmlMandatory* ao varrer os XML´s fosse capaz de comparar a chamada aos atores.

Tabela 6: Descrição do ator (XML do professor)

```
<entity name="Clock" class="ptolemy.actor.lib.Clock" mandatory="true">
        <property name="stopTime" class="ptolemy.data.expr.Parameter"
require="equals" value="40">
        </property>
        <doc>Create periodic timed events.</doc>
        <property name="_location"
class="ptolemy.kernel.util.Location" value="[-400.0, 200.0]">
        </property>
    </entity>
```

A Classe *Checker.comparaXmlMandatory* ao varrer os XML´s, compara se as coleções "actors" verificadas pelo método *parseDocumentMandatory* no XML do professor (M1) estão presentes também na varredura feita pelo método *parseDocument* em *modelStudents.getActors* no XML do aluno (M2) e verifica se cada ator "mandatory" aparece pelo menos uma vez na coleção do aluno, tal comparação é feita pela condição (*m1*.equivalentMandatory(*m2*)), conforme a Figura 28.

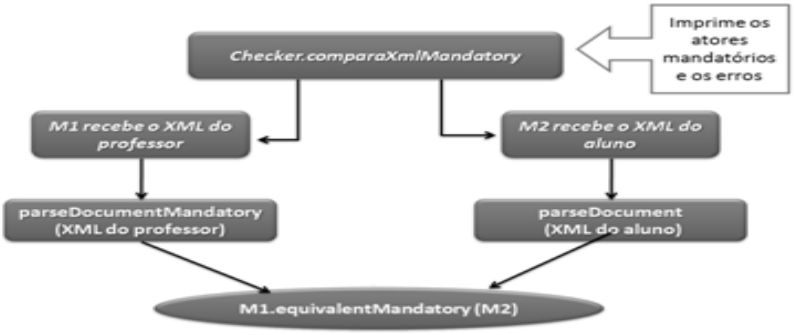

Figura 28: Varredura por atores obrigatórios

A funcionalidade de verificação de atores obrigatórios lista os atores definidos como **<mandatory="true">** no XML (M1) e o compara com os atores contidos no XML do projeto desenvolvido pelo aluno (M2). No exemplo da Figura 29, o ator Clock está contido no XML (M1), mas não foi encontrado no XML (M2), consequentemente é disparada uma mensagem de erro informando ao aluno que o ator em questão não foi encontrado no seu modelo. A intenção é proporcionar ao aluno um *feedback* com os atores pendentes no seu projeto.

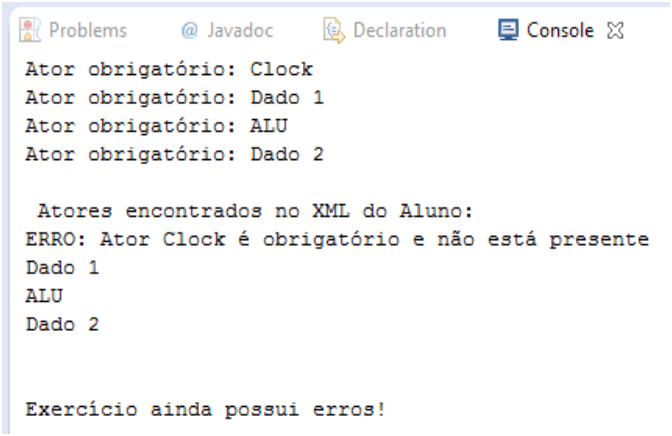

Figura 29: Mensagem da verificação de atores obrigatórios

Na aplicação em sala de aula, o professor envia ao aluno o roteiro didático com a extensão RESP, que é referente ao arquivo XML preparado pelo professor para aplicação do exercício.

5.2.2.2 Verificar conexão entre os atores

Outra questão solucionada pela verificação automática é a dos alunos que efetuam as conexões equivocadas entre os atores. Logo, o aluno poderá ter um *feedback* sobre a relação correta entre um ator e outro.

Nos cenários criados no ambiente Ptolemy, as relações são declaradas no XML conforme exposto na Tabela 7. E através das declarações "*relations*" é possível efetuar uma varredura entre os XML´s e compará-los, a fim de verificar quais conexões sugeridas pelo professor não foram realizadas pelo aluno.

Tabela 7: Declaração das relações entre os atores

```
</entity>
    <relation name="relation5" class="ptolemy.actor.TypedIORelation">
        <property name="width" class="ptolemy.data.expr.Parameter"
value="1">
        </property>
```

```
        <vertex name="vertex1" value="[-360.0, 200.0]">
        </vertex>
    </relation>
.
.
.
</relation>
    <link port="Clock.output" relation="relation5"/>
    <link port="Dado 1.trigger" relation="relation5"/>
    <link port="Dado 3.trigger" relation="relation5"/>
    <link port="Operação da ULA.trigger" relation="relation5"/>
    </entity>
```

Uma das tarefas que o aluno irá realizar é conectar os atores corretamente a outros, de acordo com o do professor. Logo, todas as relações entre atores propostas pelo professor devem estar presentes no trabalho do aluno. Sendo assim, o sistema utilizando o método *checkrelations* compara todos os pares de relações existentes, lembrando que A – B é equivalente a B – A.

A funcionalidade ao ser ativada efetua uma varredura nos projetos e captura as relações e as portas contidas dentro de cada relação, criando uma *Arraylist* de relações para cada projeto. Em seguida, verifica se as portas contidas numa relação X no projeto do professor encontrassem em alguma relação do projeto do aluno. Na Figura 29 a equivalência entre as relações é ilustrada pela mensagem *"relationX (PROF) tem as mesmas portas de relationX (ALUNO)"*.

Entretanto, se o aluno conectou os atores de forma equivocada e as relações do seu projeto contem portas diferentes das portas contidas nas relações do professor, são informadas mensagens de erros com quais conexões entre as portas dos atores estão faltando no projeto aluno, baseando-se no arquivo XML do professor. Então mesmo que em alguma relação do projeto do aluno exista uma das portas contidas numa relação do projeto do professor, ele será orientado pelas mensagens de erros a conectar as portas.

A Figura 30 traz dois exemplos de erros que podem acontecer e que o aluno cometeu. Onde não foi conectado a porta "*Dado1.output*" a porta "*ALU.Input_port_A*" e sim a porta "*Add.Input_port_A*"

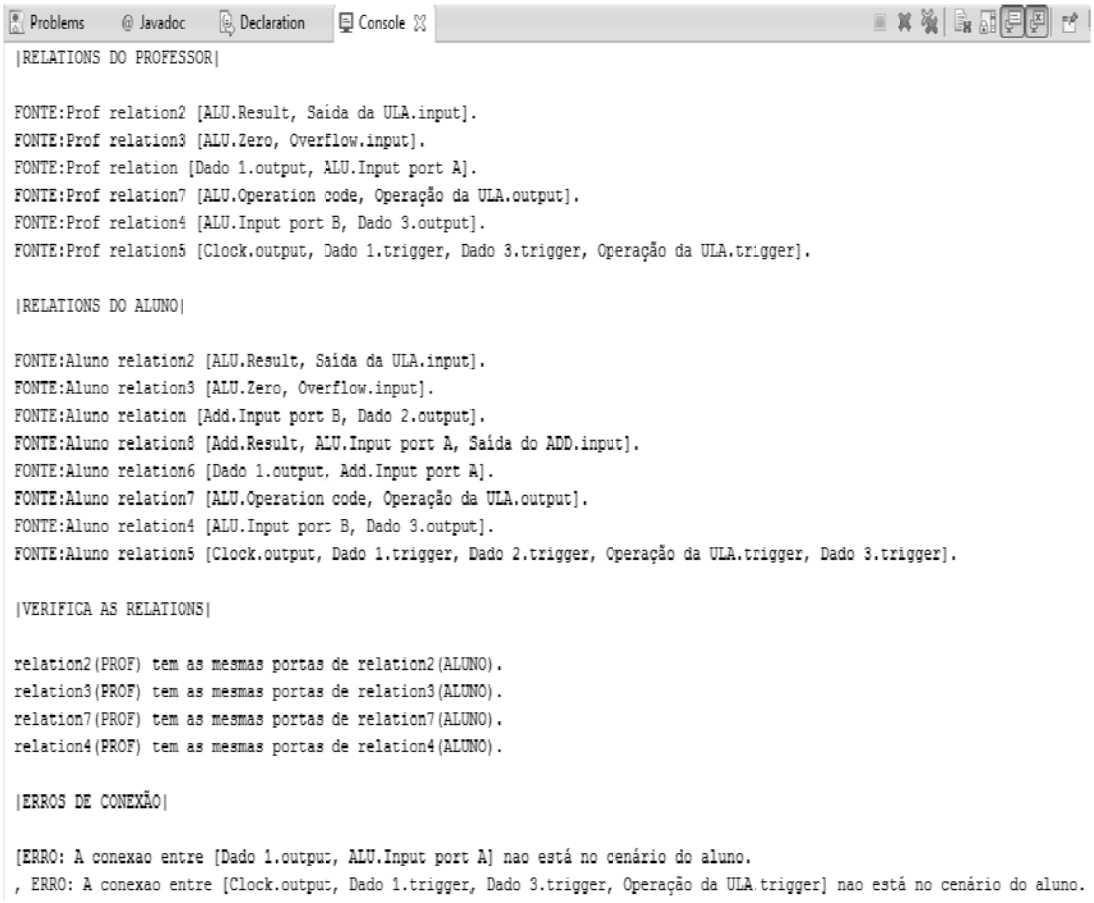

Figura 30: Mensagem de verificação das conexões entre atores

5.2.2.3 Verificar as propriedades dos atores obrigatórios

A funcionalidade de verificação das propriedades dos atores obrigatórios possibilita ao professor especificar valores para as propriedades e parâmetros dos atores. Logo, na descrição de um experimento onde o professor solicita a alteração de algum parâmetro do modelo é possível verificar se o aluno efetuou tal passo.

Uma dificuldade encontrada no XML gerado pelo Ptolemy é que o ator possui tipos heterogêneos de propriedades, como: valor, localização, ícone, nome de atributo, entre outros. Sendo assim, para verificar se os atores possuem propriedades equivalentes entre os modelos, foi adicionado o texto (*require="**restrição de valores**" value=" ")* no XML do professor para que fosse possível comparar as propriedades dos atores apenas relacionadas a valores.

Além do tipo de propriedade, foram adicionados ao método algumas restrições de valores, podendo o professor delimitar as condições que o aluno deverá seguir para introduzir os valores referentes às propriedades dos atores, conforme a Tabela 8.

Tabela 8: Restrições dos valores para as propriedades

Restrição	Condição
Equals	= =
More	>
Less	<
Moreoreguals	>=
Lessoreguals	<=

A verificação nas propriedades do ator iniciasse verificando os atores obrigatórios utilizando o método descrito na seção 5.2.2. Em seguida efetua a varredura das propriedades do ator obrigatório encontrado e armazena o valor contido na variável *"value"*. Finalizando, o valor da propriedade encontrada no ator do aluno que corresponde a um ator obrigatório e comparado com a condição configurada.

A Figura 31 ilustra um exemplo de tal funcionalidade utilizando a restrição "*equals*" e informa que o aluno informou um valor indevido para a propriedade "Stoptime" do ator "*Clock*".

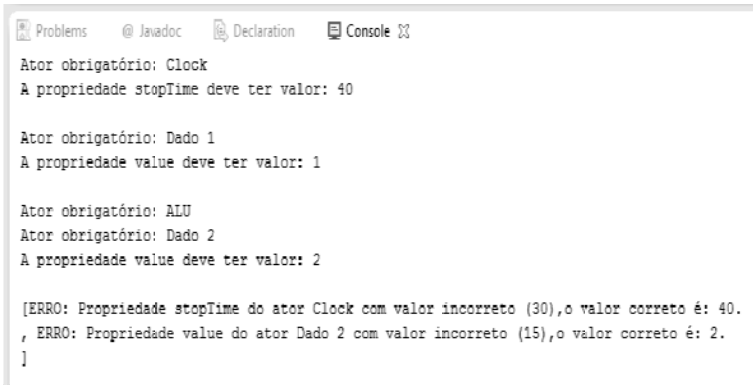

Figura 31: Mensagem de verificação das propriedades dos atores

5.2.3 Metodologia para a criação de roteiros didáticos

Para a execução de um projeto na ferramenta Ptolemy, inicialmente foi definido uma metodologia para a modelagem do ambiente a ser estudado, com a criação de atores e de materiais didáticos. Na Figura 32 são ilustradas as atividades sugeridas por [Brito09] e seguidas neste trabalho.

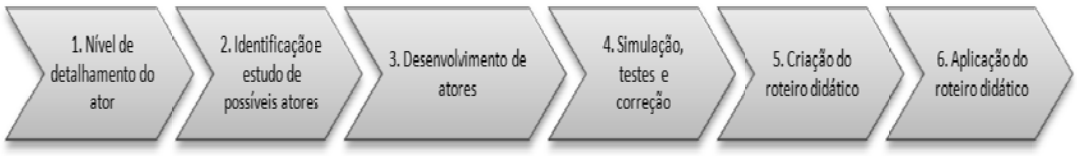

Figura 32: Metodologia para aplicação do roteiro didático

A primeira atividade é decidir o nível de abstração do ator (1). Como exemplo, citamos a Unidade Central de Processamento (CPU) que num nível mais a baixo de abstração possui a Unidade de Controle (UC) e que esta possui vários sub-nivéis, como: memória de controle, registrador e decodificador de controle e lógica de sequenciação. No (2) identificamos possíveis atores que correspondem aos componentes da arquitetura estudada e se busca compreender como esses atores funcionam em detalhes. Sendo desenvolvidos em Java (3), simulados e testados isoladamente (4) e utilizados juntamente com outros atores num cenário (5). Para finalizar o roteiro é aplicado aos alunos (6).

Após a criação dos atores, que servirão de repositório para aplicação em disciplinas posteriores, e criação do roteiro didático, sua aplicação é ilustrada na Figura 40, descrevendo o processo de sua verificação.

De forma a validar a metodologia, a simulação dos atores criados e o método de avaliação automática na nova interface gráfica, os próximos passos foram utilizar a ferramenta para a simulação da arquitetura MIPS, utilizando como material de apoio os livros de Hennessy e Patterson, respectivamente na segunda e terceira edições.

5.3 Extensão do Ptolemy para Simulação de Arquitetura de Computadores

Todas as modificações introduzidas no Ptolemy, detalhadas nas seções anteriores, tiveram como objetivo preparar o ambiente para simulação de modelos que representem Arquiteturas de Computadores visando validação e avaliação através da simulação.

A extensão desenvolvida permite que o usuário modele suas próprias arquiteturas utilizando o repositório de atores. Inicialmente foram desenvolvidos atores para representação dos componentes da arquitetura do processador MIPS, mas com a aplicação da ferramenta almejasse estender sua aplicação para outras arquiteturas de

processadores, como: Von Neumann, *Multiple Instruction Multiple Data* (MIMD), entre outras.

A nova interface contribuirá para que o professor possa demonstrar na prática aos alunos tópicos abordados na ementa da disciplina, como: hierarquia de memória, registradores, barramentos, dispositivos de entrada/saída e a abordagem estrutural no estudo de computadores.

5.3.1 Visão Geral

A extensão criada para a disciplina de Arquitetura de Computadores pode ser ativada através da tela inicial do Vergil (Figura 33), utilizando-se a opção "*Modelagem e Simulação*" do menu (Arquivo - > Novo).

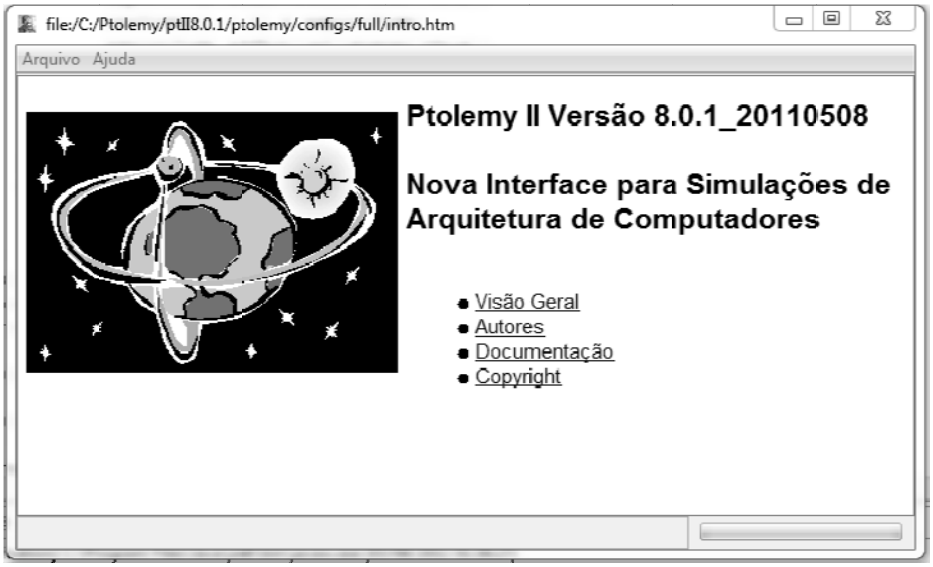

Figura 33: Tela inicial da nova interface do Vergil

Ao ativar o Vergil um novo ambiente é disponibilizado para que o usuário crie atividades ou abra um modelo existente, disponibilizando ferramentas especificas para a simulação de Arquitetura de Computadores (Figura 34). Sendo que as principais alterações estão na barra de menu e na paleta de componentes.

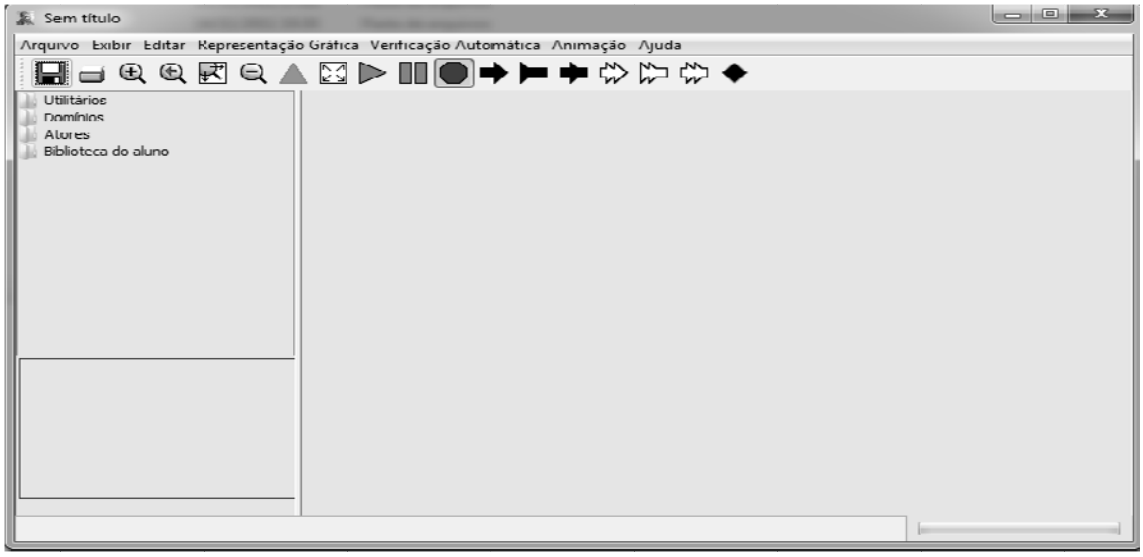

Figura 34: Nova interface gráfica em português

Para exemplificar as características desta funcionalidade utilizamos dois modelos desenvolvidos na nova interface, onde o primeiro é referente ao roteiro didático desenvolvido pelo professor e o segundo é a atividade do aluno baseado no roteiro.

A Figura 35 ilustra o roteiro didático sobre o comportamento da ALU.

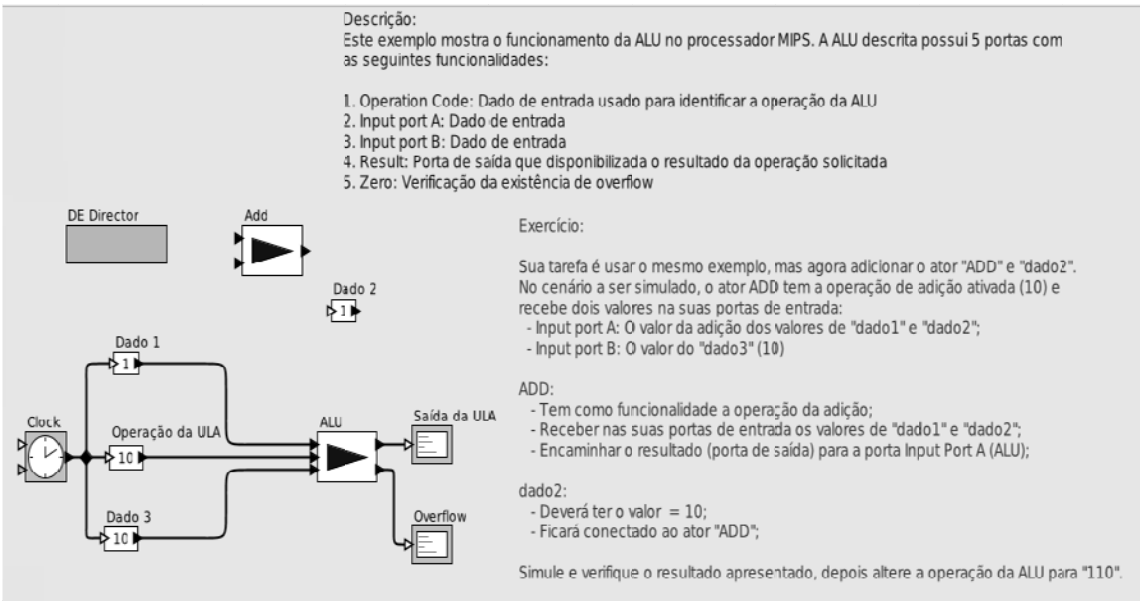

Figura 35: Roteiro didático do funcionamento da ULA (*ExercícioULAProfessor.xml*)

No Código 1 do arquivo XML abaixo, destacasse o nome do projeto, a localização das classes Java referentes aos atores, suas propriedades e as conexões entre

os atores no projeto. O arquivo é referente ao projeto desenvolvido por um professor que serviu de roteiro para as a criação do projeto exposto na Figura 35.

Código 1: ExercícioULAProfessor.xml

```
<?xml version="1.0" standalone="no"?>
<entity name="exercicioULAProfessor"
.
.
.
    <entity name="Clock" class="ptolemy.actor.lib.Clock"
mandatory="true">
        <property name="stopTime" class="ptolemy.data.expr.Parameter"
require="equals" value="40">
.
.
    <entity name="Dado 1" class="ptolemy.actor.lib.Const"
mandatory="true">
        <property name="value" class="ptolemy.data.expr.Parameter"
require="equals" value="1">
.
.
    </entity>
    <entity name="ALU" class="ptolemy.actor.myactors.ALU"
mandatory="true">
.
.
    </entity>
    <entity name="Dado 2" class="ptolemy.actor.lib.Const"
mandatory="true">
        <property name="value" class="ptolemy.data.expr.Parameter"
require="equals" value="2">
        </property>
.
.
    </entity>
    <entity name="Dado 3" class="ptolemy.actor.lib.Const">
        <property name="value" class="ptolemy.data.expr.Parameter">
        </property>
.
.
    </entity>
    <entity name="Operação da ULA" class="ptolemy.actor.lib.Const">
        <property   name="value"   class="ptolemy.data.expr.Parameter"
value="10">
        </property>
.
.
    <entity name="Saída da ULA" class="ptolemy.actor.lib.gui.Display">
.
.
    <entity name="Overflow" class="ptolemy.actor.lib.gui.Display">
.
.
    <entity name="Add" class="ptolemy.actor.myactors.Add">

    <relation name="relation5" class="ptolemy.actor.TypedIORelation">
.
.
    <relation name="relation7" class="ptolemy.actor.TypedIORelation">
```

```
.
.
    <relation name="relation2" class="ptolemy.actor.TypedIORelation">
.
.
    <relation name="relation3" class="ptolemy.actor.TypedIORelation">
.
.
   <relation name="relation" class="ptolemy.actor.TypedIORelation">
.
.
    <relation name="relation4" class="ptolemy.actor.TypedIORelation">
.
.
    </relation>
    <link port="Clock.output" relation="relation5"/>
    <link port="Dado 1.output" relation="relation"/>
    <link port="Dado 1.trigger" relation="relation5"/>
    <link port="ALU.Input port A" relation="relation"/>
    <link port="ALU.Operation code" relation="relation7"/>
    <link port="ALU.Input port B" relation="relation4"/>
    <link port="ALU.Result" relation="relation2"/>
    <link port="ALU.Zero" relation="relation3"/>
    <link port="Dado 3.output" relation="relation4"/>
    <link port="Dado 3.trigger" relation="relation5"/>
    <link port="Operação da ULA.output" relation="relation7"/>
    <link port="Operação da ULA.trigger" relation="relation5"/>
    <link port="Saída da ULA.input" relation="relation2"/>
    <link port="Overflow.input" relation="relation3"/>
</entity>
```

Na Figura 36 é ilustrada a atividade realizada pelo aluno, seguindo as orientações do roteiro didático feito pelo professor.

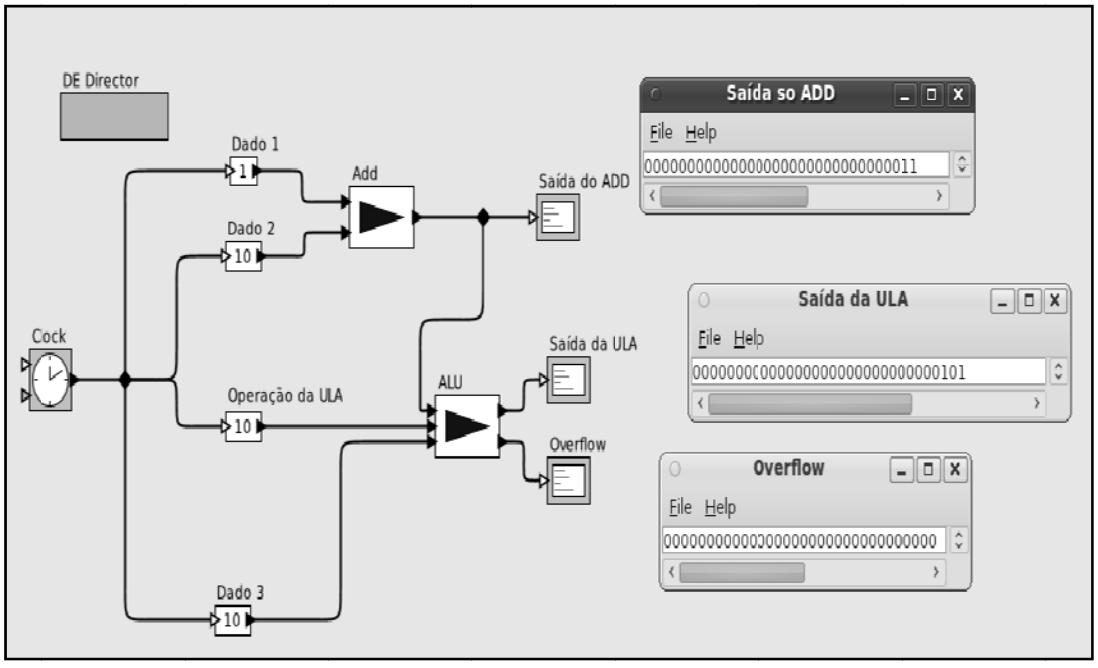

Figura 36: Resolução do exercício (*ExercícioULAAluno.xml*)

O código 2 é referente ao arquivo XML da Figura 36.

Código 2:ExercícioULAAluno.xml

```
<?xml version="1.0" standalone="no"?>
<entity name="exercicioULAAluno"
.
.
    </property>
     <entity name="Clock" class="ptolemy.actor.lib.Clock">
        <property  name="stopTime"  class="ptolemy.data.expr.Parameter"
value="30">
.
.
   </entity>
    <entity name="Dado 1" class="ptolemy.actor.lib.Const">
        <property name="value" class="ptolemy.data.expr.Parameter"
value="1">
.
.
    </entity>
    <entity name="Add" class="ptolemy.actor.myactors.Add">
.
.
    </entity>
    <entity name="ALU" class="ptolemy.actor.myactors.ALU">
.
.
    </entity>
    <entity name="Dado 2" class="ptolemy.actor.lib.Const">
        <property name="value" class="ptolemy.data.expr.Parameter"
value="15">
.
.
   </entity>
    <entity name="Operação da ULA" class="ptolemy.actor.lib.Const">
        <property name="value" class="ptolemy.data.expr.Parameter"
value="10">
.
.
    </entity>
    <entity name="Dado 3" class="ptolemy.actor.lib.Const">
        <property name="value" class="ptolemy.data.expr.Parameter"
value="10">
.
.
    </entity>
    <entity name="Saída da ULA" class="ptolemy.actor.lib.gui.Display">
.
.
    </entity>
    <entity name="Overflow" class="ptolemy.actor.lib.gui.Display">
.
.
    </entity>
    <entity name="Saída do ADD" class="ptolemy.actor.lib.gui.Display">
.
.
    </entity>
    <relation name="relation5" class="ptolemy.actor.TypedIORelation">
```

```
.
.
   <relation name="relation6" class="ptolemy.actor.TypedIORelation">
.
.
   <relation name="relation7" class="ptolemy.actor.TypedIORelation">
.
.
    <relation name="relation2" class="ptolemy.actor.TypedIORelation">
.
.
    <relation name="relation3" class="ptolemy.actor.TypedIORelation">
.
.
    <relation name="relation8" class="ptolemy.actor.TypedIORelation">
.
.
    <relation name="relation4" class="ptolemy.actor.TypedIORelation">
.
.
    </property>
    </relation>
    <link port="Clock.output" relation="relation5"/>
    <link port="Dado 1.output" relation="relation6"/>
    <link port="Dado 1.trigger" relation="relation5"/>
    <link port="Add.Input port A" relation="relation6"/>
    <link port="Add.Input port B" relation="relation"/>
    <link port="Add.Result" relation="relation8"/>
    <link port="ALU.Input port A" relation="relation8"/>
    <link port="ALU.Operation code" relation="relation7"/>
    <link port="ALU.Input port B" relation="relation4"/>
    <link port="ALU.Result" relation="relation2"/>
    <link port="ALU.Zero" relation="relation3"/>
    <link port="Dado 2.output" relation="relation"/>
    <link port="Dado 2.trigger" relation="relation5"/>
    <link port="Operação da ULA.output" relation="relation7"/>
    <link port="Operação da ULA.trigger" relation="relation5"/>
    <link port="Dado 3.output" relation="relation4"/>
    <link port="Dado 3.trigger" relation="relation5"/>
    <link port="Saída da ULA.input" relation="relation2"/>
    <link port="Overflow.input" relation="relation3"/>
    <link port="Saída do ADD.input" relation="relation8"/>
</entity>
```

As linhas do código correspondem respectivamente aos atores, suas propriedades e suas conexões, primordiais para que o Checker efetue a verificação automática.

No capitulo 6 a seguir, será apresentado um roteiro didático a fim de detalhar como serão realizadas as atividades com os alunos e validar sua aplicação como ferramenta de aprendizagem.

5.4 Potenciais e Limitações

O modelo pedagógico clássico utilizado na maioria dos cursos do ensino fundamental, médio e superior no Brasil ainda possui como base o modelo Instrucionista [Machado04]. A educação com tal modelo é baseada na transferência da maior quantidade possível de informação entre um mestre ativo e um aprendiz passivo. Logo, um dos potenciais deste trabalho é a utilização de outro método de ensino, adotando os conceitos da teoria construtivista nos roteiros didáticos utilizados.

No Construtivismo, o professor tem a função de ser um facilitador no processo de aprendizagem e não apenas um transmissor do conhecimento. A extensão Ptolemy oferece ao docente o ambiente para motivar o aluno no espírito crítico-investigativo, além de orientá-lo em trabalhos cooperativos com os demais colegas na busca da construção do conhecimento.

Outro potencial da extensão e que proporciona ao aluno o desenvolvimento cognitivo na busca das soluções dos roteiros didáticos é a verificação automática. Segundo Piaget [Machado04], o erro construtivo proporciona um desequilíbrio, mas possibilita uma nova ação intelectual na busca de um novo equilíbrio.

A verificação automática informa ao aluno os erros encontrados no seu cenário, baseados no arquivo resposta do professor. Entretanto, uma limitação da funcionalidade é que o aluno ao criar seus próprios modelos e buscar novas soluções para determinada questão pode utilizar componentes diferentes dos utilizados pelo professor e alcançar em determinados momentos a mesma solução, mas com isso o objetivo da pesquisa também será alcançado, visto que o aluno desenvolveu o conhecimento esperado pelo professor.

5.5 Considerações Finais

Para que o Ptolemy possa ser utilizado com êxito para fins educacionais as modificações em seu simulador e na sua interface gráfica deverão ser realizadas e validadas pela metodologia descrita por [Brito09]. Assim como o desenvolvimento dos componentes o objetivo de auxiliar a tarefa do professor com a criação de roteiros práticos poderão ser utilizados em sala de aula. O próximo passo será o de validar a

ferramenta aplicando atividades com alunos de Graduação em Computação e Sistemas de Informação da própria universidade, avaliando os mesmo com questões quantitativas e qualitativas.

6 Desenvolvimento de um roteiro didático

Para ilustrar o roteiro didático aplicado aos alunos, nesta seção é detalhado um experimento onde foram abordados os conhecimentos descritos no tutorial desenvolvido na disciplina Arquitetura de Computadores no Mestrado em Informática do DI – UFPB, intitulado: “Modelagem e Simulação do Processador MIPS na Ferramenta Ptolemy”. O material serve de base para os conceitos básicos de utilização do framework e definição dos componentes estudados.

O objetivo deste experimento foi o de estimular os alunos a criarem seus próprios modelos de arquitetura de maneira simples, interativa e didática, baseados num roteiro didático exposto pelo professor, ao mesmo tempo, possibilitando que eles verifiquem de uma forma automática os erros no seu cenário.

Inicialmente, vamos descrever a sequência didática utilizada para preparar um exercício para os alunos.

6.1 Tema: Unidade Lógica Aritmética (ALU)

Na descrição da sequência didática, começamos descrevendo o componente, que nesta é a Unidade Lógica Aritmética – *Arithmetic Logic Unit* (ALU). A ALU é a unidade do processador que executa as operações aritméticas e lógicas referenciadas pelos códigos de operações (*opcodes).*

O ator criado para representar a ALU suporta seis operações (AND, OR, ADD, SUB, SLT, E NOR) em um circuito combinacional que calcula uma saída de 32 bits baseada em duas entradas de 32 bits e uma entrada de 4 bits que especifica a operação da ALU a ser executada além de computar bits para verificar se existiu overflow.

Na Tabela 9, A e B representam as duas entradas de 32 bits para a ALU, e F(output) se refere ao resultado de 32 bits.

Tabela 9: Operações da ALU

Função	Controle da ALU	Semântica
E (and)	0000	F = A & B
OU (or)	0001	F = A | B
Adição (add)	0010	F = A + B
Sub	0110	F = A – B
Slt	0111	F = (A < B) ? 1:0
NOR	1100	F = ~ (A | B)

Após especificar as portas e variáveis do ator, criamos seu relacionamento de entrada e saída, conforme a Figura 37. No cenário abaixo, foi escolhida a operação de adição (10) e feito o teste para três dados de entrada, Dado 1 e Dado 2 (nas portas de entrada do ADD) e Dado 3 (conectado direto na porta Input B da ALU).

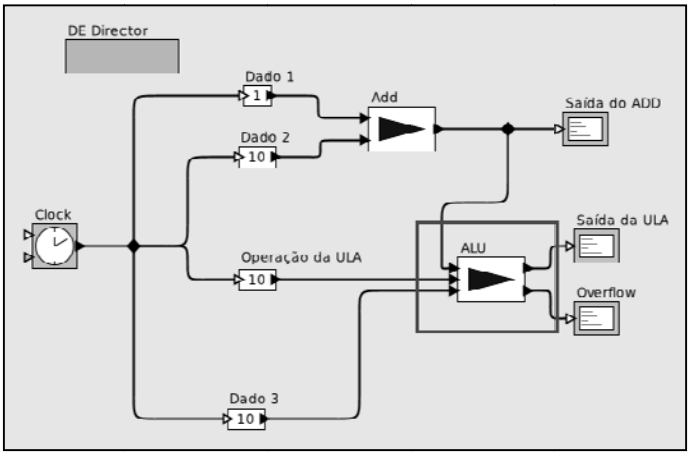

Figura 37: Simulação do Ator ALU

Para efetuar o teste para as funcionalidades do ator, a descrição da ALU foi a seguinte:

- Operação da ALU: Representa as operações de 4 bits.
- Saída do ADD: Representa a primeira entrada de 32 bits (inputA);
- Dado 3: Representa a segunda entrada de 32 bits (inputB);
- Saída da ALU: Representa a saída de 32 bits (result);
- Overflow: Representa os três bits finais que verificam overflow (zero).

Conforme exposto na Tabela 9, o valor “10” representa a função adição, onde a mesma efetua o cálculo, em binário, dos valores da porta de saída do ator ADD e do ator Dado 3. Tendo a ALU o resultado descrito por uma sequência de 32 bits e o valor “101” como resposta a operação e aos operandos.

O professor deverá descrever o experimento para o aluno utilizando o objeto “Annotation” contido na paleta (Utilitários / Decorativos), sendo aberta a tela de edição da anotação (Figura 38).

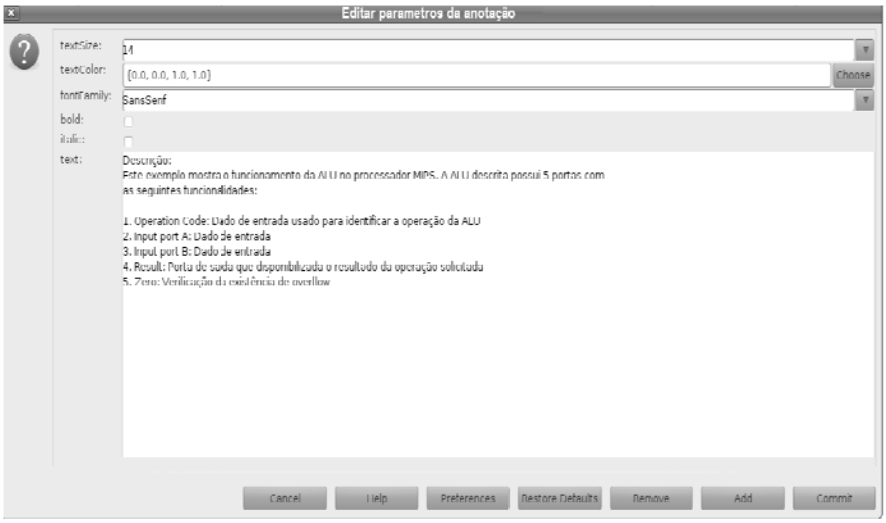

Figura 38: Tela de edição das anotações do roteiro didático

Como exercício proposto, o aluno deverá adicionar o ator ADD (efetua a adição de valores) ao cenário do professor, recebendo os valores dos atores “Dado 1” e “Dado 2” e tendo sua saída conectada a *Input port A* da ALU. Continuando a ter a operação de adição (10) na porta *Operation Code* e o ator “Dado 3” conectado a *Input port B*. Verificando se o comportamento da ALU é o esperado, o aluno seguirá para a conclusão do exercício.

6.2 Aplicação do experimento

Neste experimento iremos verificar como as operações lógica/aritméticas de transferência de dados e de controle são efetivamente executadas em processadores. Para tanto, se faz o uso do framework Ptolemy, criado e mantido na Universidade de Berkeley, para que os alunos modelem seus experimentos.

Na aplicação do experimento, os alunos deveriam criar um modelo que simulasse um dos estágios do processador MIPS, que foi a Busca da Instrução -

Instruction Fetch (IF) que deveria ter a seguinte premissa: que o modelo buscasse a próxima instrução na memória utilizando o endereço contido no registrador PC (contador de programa) e armazenasse na Memória de Instruções.

O objetivo é que os alunos entendam as principais micro-instruções presentes em qualquer processador ocorrem.

Para o experimento, foi utilizada como referência, a construção do caminho de dados descrita por Patterson para o processador MIPS. Onde a execução de uma instrução em um processador pode ser dividida em vários estágios. A quantidade de estágios e o propósito de cada um deles são diferentes em cada projeto. O processador MIPS utiliza cinco estágios conforme detalhados na Tabela 10.

Tabela 10: Estágios do Processador MIPS

IF	*Instruction Fetch* (busca da instrução): neste estágio a próxima instrução é buscada na memória utilizando o endereço contido no registrador PC (contador de programa) e armazenada no IR (registrador de instrução).
ID	*Instruction Decode* (decodificação da instrução): neste estágio a instrução contida no IR é decodificada, o valor do próximo PC é calculado, e os operandos necessários para execução da instrução são lidos dos registradores.
EX	*Execute* (execução): aqui a instrução é de fato executada; todas as operações da ALU são executadas neste estágio. (A ALU é a unidade lógica e aritmética e é responsável por operações como adição, subtração, deslocamentos de bits, etc).
MA	*Memory Access* (acesso a memória): este estágio faz os acessos à memória requisitados pela instrução atual. Desta forma, para instruções tipo load, é lido um operando da memória. Para instruções tipo store, é armazenado um operando na memória. Para todas as outras, nada é feito.
WB	*Write Back* (escrever de volta): Para instruções que têm algum resultado (registrador destino), este estágio escreve o resultado no registrador especificado. Isso inclui quase todas as instruções, exceto a instrução *nop* (instrução que simplesmente não executa nada) e instruções tipo *store*.

A Figura 39 mostra a parte do caminho de dados usado na busca de uma instrução e no incremento do PC.

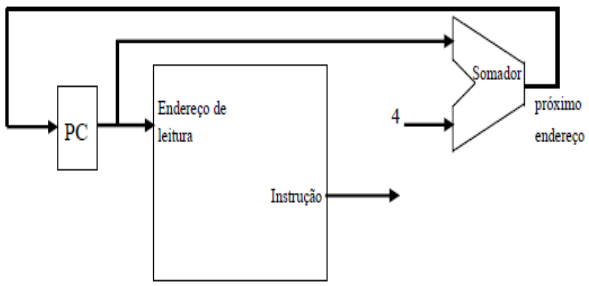

Figura 39: Envio da Instrução do PC para memória

Um ponto importante do exercício foi o aprendizado do funcionamento e da importância da temporização de um sistema. Um sinal de clock foi adicionado (ator mais a esquerda na Figura 40), e atores especiais (Time Delay) foram utilizados para gerar um atraso inerente ao processo entre enviar a instrução e esperar seu resultado ficar pronto na saída.

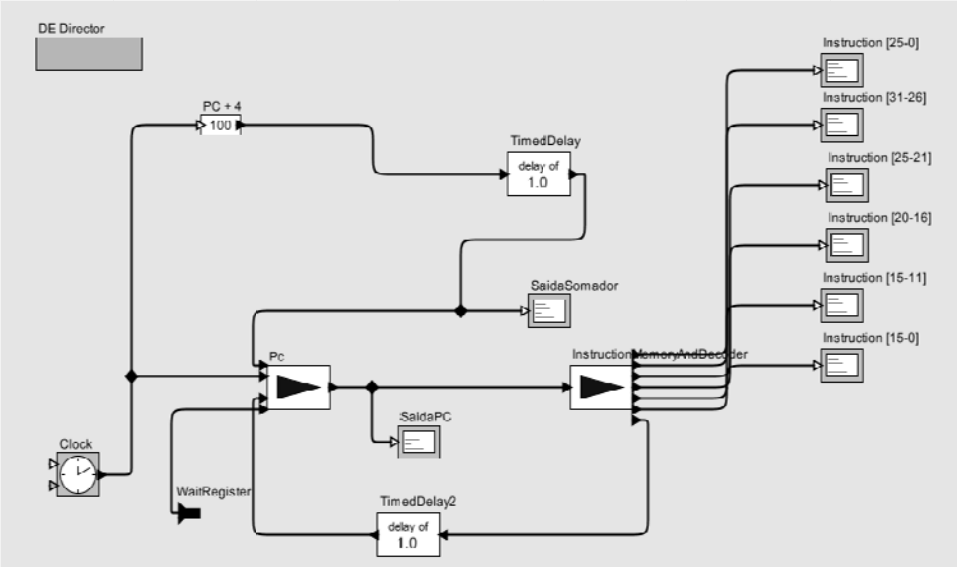

Figura 40: Modelo criado em sala de aula

Ainda sobre o exercício, podemos verificar que o ator *InstructionMemoryAndDecorder* consiste em receber na sua porta de entrada o valor de 32 bits originados do *PC* e usá-lo com index para selecionar uma instrução na memória de instrução, enviando para as suas saídas segmentos da instrução de entrada, de forma que cada segmento seguirá para um destino diferente, seguindo a ordem abaixo:

- Instrução 25 -0: Esses 26 bits seguem para o ator *Shift Left2* que tem o papel de efetuar o deslocamento dos elementos em dois bits à esquerda.
- Instrução 31 -26: A Unidade de Controle receberá 6 bits da instrução;
- Instrução 25 – 21: Esses 5 bits da instrução irão direto para o banco de registradores;
- Instrução 20 – 16: Esses 5 bits irão para o Multiplexador, como também para o banco de registradores;
- Instrução 15 -11: Esses 5 bits irão para o Multiplexador, antes de chegar no banco de registradores;

- Instrução 15 – 0: Esses 16 bits seguirão para a Extensão de Sinal, sendo que 6 bits desses irão para a *ALU Control*.

Na resolução do exercício, são informadas as três instruções de 32 bits que correspondem ao index originado do PC (Figura 41). Logo, o aluno ao receber o feedback das tarefas que faltavam ser realizadas pelos mesmos e não encontrar mais erros no seu cenário, poderá analisar que na saída do ator *InstructionMemoryAndDecorder* irão aparecer os segmentos de instruções, que seguirão para os outros componentes.

Figura 41: Resultado do experimento

Importante comentar que não seria possível a verificação automática do roteiro didático na ferramenta Ptolemy II sem a extensão criada. Tanto o aluno como o professor teriam apenas condições de criar seus atores, seus cenários para simulação e verificar seus resultados.

6.3 Resultados do experimento

A inclusão do experimento e suas atividades práticas na disciplina de Arquitetura de Computadores vêm suprir uma necessidade dos alunos em fixar melhor o aprendizado, simulando situações reais em que elas acontecem. Além disso, estas atividades práticas podem servir como motivadores iniciais dos tópicos a serem estudados dando um estímulo aos alunos que os levará a entender melhor os assuntos.

Os alunos que participaram do experimento estão no 4° Período do Curso de Ciência da Computação, dos turnos vespertino e noturno. Pela análise da ementa conclui-se que a disciplina de arquitetura e organização de computadores tem o foco

para os aspectos básicos, tanto de baixo quanto de alto nível, relacionados aos recursos computacionais.

O planejamento do experimento é descrito no Anexo B deste trabalho.

Durante o experimento ficou claro que os mesmos possuem uma grande dificuldade em relação a tais aspectos, como também, as características dos componentes e do caminho de dados entre eles.

Apenas a extensão do Ptolemy foi avaliada para verificar sua aplicação e usabilidade para fins educacionais. Os alunos preencheram um formulário online (http://twixar.com/YEWuLspLAT6h) onde puderam informar quais foram às dificuldades encontradas na resolução do experimento.

Como se tratava de turmas pequenas, apenas 16 alunos participaram das aulas e responderam à avaliação aplicada. A seguir, são apresentados alguns destes resultados.

6.3.1 Forma de interação no ambiente

Na Figura 42 é informado o resultado da questão respondida pelos alunos referente a forma de interação no ambiente.

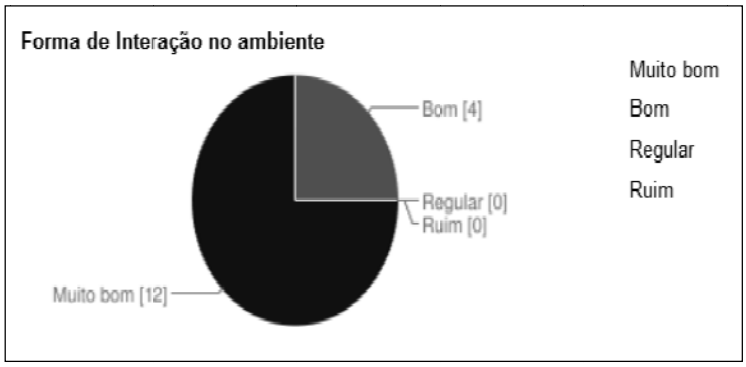

Figura 42: Forma de interação no ambiente

Nesta questão, 12 alunos consideraram com “muito bom” os aspectos de simulação e de acompanhamento do fluxo de dados.

6.3.2 Aspectos que contribuem para a aprendizagem

Para essa questão, o aluno deveria assinalar as características da extensão que mais contribuíram para a aprendizagem segundo sua interpretação. Na Figura 43 duas

características se destacaram: a funcionalidade de animação (94%) e a funcionalidade de verificação automática (88%).

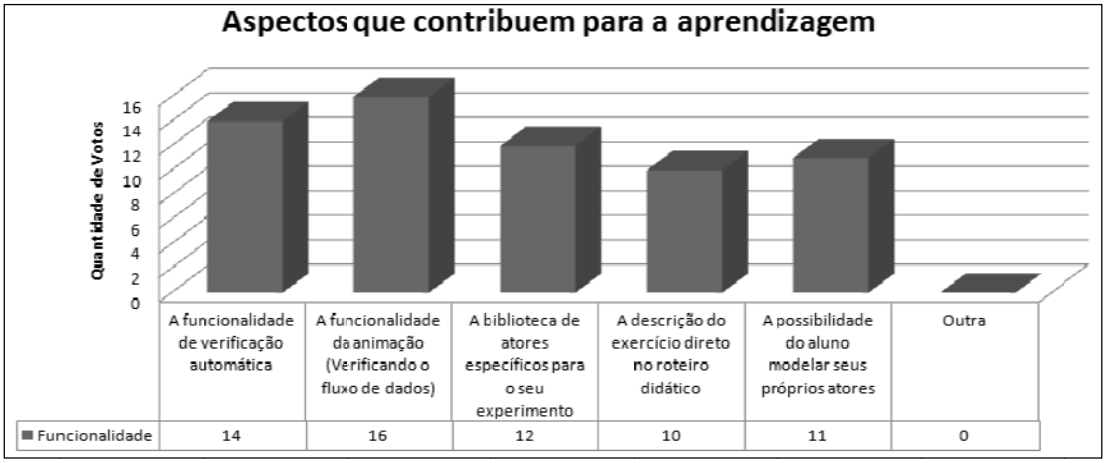

Figura 43: Caracteristicas da Extensão Ptolemy

A característica de animação possibilita aos alunos acompanharem o fluxo de dados e das instruções envolvidas no experimento passo-a-passo, facilitando o entendimento pela passagem das instruções pelos atores. Já a verificação automática, possibilita o feedback ao aluno.

6.3.3 Avaliação da interface com o usuário

A Figura 44 ilustra os resultados para as questões que tratavam da avaliação geral da interface com o usuário.

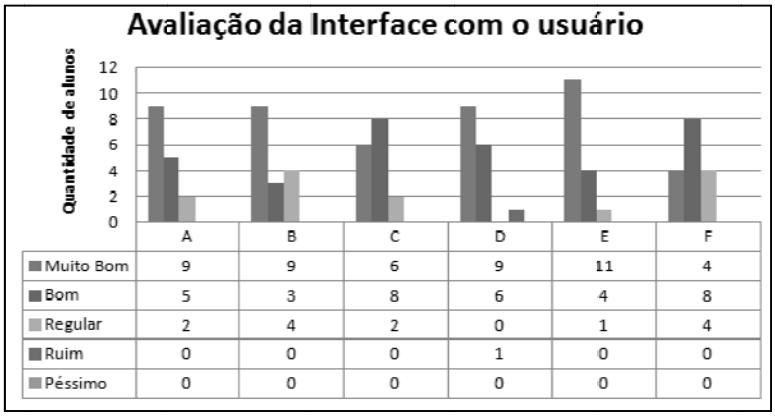

Figura 44: Avaliação da interface com o usuário

Nestas questões foram avaliados alguns itens. São eles: (A) Disponibilidade das tarefas; (B) Interface do programa; (C) Apresentação dos menus; (D) Tamanho e tipo de

fontes utilizadas; (E) Forma de despertar o interesse do aluno; (F) Navegação pelo sistema.

Em relação aos itens (A) e (B), mais de 56% da turma os consideraram como “muito bom”. Nestas questões, os alunos foram questionados pela forma que as tarefas ficavam disponíveis na área de trabalho da Extensão do Ptolemy e como os mesmos analisaram inicialmente as mudanças feitas na interface do programa.

Em relação ao item (C), 88% dos alunos consideram como “muito bom” ou “bom” a localização dos menus, atalhos e ícones.

Para o item (D), o percentual de 56% ficou satisfeito com o tamanho e tipo de fontes utilizadas, classificando como “muito bom”.

O item (E) teve o maior percentual de aprovação pelos alunos, com 69%. Nesta questão, os alunos avaliaram como “muito bom” a forma que a Extensão do Ptolemy possibilita despertar interesse. O fato da criação de atores específicos para a disciplina possibilita a eles entender melhor seu funcionamento, suas características e como cada ator se relaciona com outros no cenário.

Outra característica que colabora para um alto conceito para a questão é a verificação automática, onde é possível passar para o professor como o aluno está realizando (ou realizou) as atividades escritas pelo mesmo, como também colaborar para o aluno resolva o experimento sem a presença do professor Finalizando as questões referentes à avaliação da interface com o usuário, o item (F) teve 75%, quando se considera as respostas como: “muito bom” ou “bom”.

O formulário de avaliação aplicado com os alunos aborda tópicos levantados anteriormente para avaliar a usabilidade do Ptolemy. Logo, podemos concluir que os critérios que apresentaram não conformidade no Ptolemy obtiveram um bom percentual de conformidade com a extensão criada para fins educacionais.

7 Considerações Finais

Após analisar as características do Ptolemy citadas neste trabalho, verificamos seu potencial para o ensino/aprendizagem pela possibilidade do professor e dos alunos criarem seus próprios modelos, como também combinarem novos elementos e ampliarem seus estudos.

O simulador receberá mais funcionalidades e espera-se incorporá-lo em atividades práticas em tópicos curriculares na ementa da disciplina de Arquitetura de Computadores. Esse novo ambiente proporcionará ao aluno combinarem novos elementos e ampliarem seus estudos, tornando agora, elementos ativos na formação do conhecimento, onde eles próprios deverão buscar novas formas de criar os seus próprios modelos.

Atualmente utilizamos como bibliografia básica o livro Arquitetura de Computadores de [Patterson05] apresentando uma arquitetura hipotética própria para o ensino, divida em sequências de atividades e componentes.

Para as próximas avaliações, será realizado um novo experimento no qual os alunos serão avaliados.

Como requisito de um software educacional, a extensão do Ptolemy possui agora algumas características pedagógicas, como: viabilidade de utilização do software em situações educacionais, adaptabilidade a metodologia de ensino e adequação a uma proposta de educação mais construtivista.

Para avaliar os alunos, serão abordadas atividades com a concepção do propósito de sua utilização. Onde se procura dar exemplos e fazer experiências com o software em situações educacionais.

8 Anexos

ANEXO A

Questão 1 de 9 – Mensagem de Erro

As mensagens de erro ajudam a resolver o problema do usuário, fornecendo com precisão o local e a causa específica ou provável do erro, bem como as ações que o usuário poderia realizar para corrigi-l o?

☐ Sim ☑ Não ☐ Não aplicável ☐ Adiar resposta

Questão 2 de 9 – Mensagem de Erro

As mensagens de erro são neutras e polidas?

☐ Sim ☑ Não ☐ Não aplicável ☐ Adiar resposta

Questão 3 de 9 – Mensagem de Erro

As frases das mensagens de erro são curtas e construídas a partir de palavras curtas, significativas e de uso comum?

☐ Sim ☐ Não ☑ Não aplicável ☐ Adiar resposta

Questão 4 de 9 – Mensagem de Erro

As mensagens de erro estão isentas de abreviaturas e/ou códigos gerados pelo sistema operacional?

☐ Sim ☑ Não ☐ Não aplicável ☐ Adiar resposta

Questão 5 de 9 – Mensagem de Erro

O usuário pode escolher o nível de detalhe das mensagens de erro em função de seu nível de conhecimento?

☐ Sim ☑ Não ☐ Não aplicável ☐ Adiar resposta

Questão 6 de 9 – Mensagem de Erro

A informação principal de uma mensagem de erro encontra-se logo no início da mensagem?

◉ Sim ○ Não ○ Não aplicável ○ Adiar resposta

Questão 7 de 9 – Mensagem de Erro

Quando necessário, as informações que o usuário deve memorizar encontram-se localizadas na parte final da mensagem de erro?

○ Sim ◉ Não ○ Não aplicável ○ Adiar resposta

Questão 8 de 9 – Mensagem de Erro

Em situações normais as mensagens de erro são escritas em maiúsculo/minúsculo?

○ Sim ◉ Não ○ Não aplicável ○ Adiar resposta

Questão 9 de 9 – Mensagem de Erro

As mensagens de erro têm seu conteúdo modificado quando na repetição imediata do mesmo erro pelo mesmo usuário?

○ Sim ◉ Não ○ Não aplicável ○ Adiar resposta

ANEXO B

Aplicação do Experimento

O experimento será aplicado com alunos do quarto período do curso de Ciências da Computação e tem como conteúdo programático a Unidade I da disciplina (Organização de Computadores) e o tema do experimento é Processadores: organização da CPU e a execução de instruções.

Duração em aulas

A duração do experimento é de 03 aulas.

Aula 1: Explanação sobre o conteúdo didático, Introdução ao Ptolemy e explicação do experimento (1 hora e 30 Minutos);

Aula 2: Aplicação do experimento e avaliação dos resultados (1 hora);

Descrição das etapas e seus procedimentos

Etapa 1 – Conhecimentos básicos do Ptolemy

Prezado(a) estudante, a finalidade deste guia de laboratório é ajudá-lo(a) na utilização dos recursos do ambiente Ptolemy II (Ptolomeu II) através da interface gráfica Vergil. O manual completo do Ptolemy II encontra-se na pasta Documentação (Tela de Boa Vindas do Ptolemy).

Após clicar sobre o ícone do Vergil, apareça uma janela de inicialização conforme a Figura 33. Você pode aproveitar esse momento para explorar os Links desta janela e aumentar o conhecimento sobre a ferramenta.

Abertura de ambiente gráfico de modelagem: clicar na opção *Arquivo* > *Novo* > *Modelagem e simulação*;

Para gravar o trabalho em disco, selecione na barra de Menus a opção *Arquivo* > *Salvar*. Na primeira vez que a operação for executada aparecerá uma tela onde deve ser informado o nome do arquivo e a unidade de disco utilizada. É recomendável que se crie uma pasta com o nome Experimentos onde serão guardados os experimentos;

Numa próxima vez, para abrir um trabalho, selecione na barra de Menus a opção *Arquivo* > *Abrir Arquivo*. Localize a pasta onde o trabalho foi salvo (Experimentos no exemplo acima) e clique Abrir;

Criação de um modelo executável: Na esquerda da tela está a paleta de objetos que podem ser arrastados sobre a página. Os objetos selecionados serão colocados à direita, na área cinza, que será a nossa área de trabalho. A paleta contêm 4 pastas: Utilitários, Domínios, Atores e Biblioteca do Aluno (Figura 34).

Escolhendo atores: Conforme for solicitado pelo roteiro didático do professor, você deverá acessar a pasta Atores, localizar o ator desejado e pressioná-lo com o botão esquerdo do mouse, arraste-o sobre a página e coloque-o na área de trabalho do Ptolemy;

Conexão dos elementos: Localize o cursor sobre a porta de saída do Ator e arraste o mouse até a porta de entrada do ator que deseja efetuar a conexão. Dê um clique com o cursor sobre a área de trabalho. Está criado um fio entre os atores. Repita a operação para criar um fio entre os demais atores. Caso seja necessário apagar um fio, clique em cima deste. Quando aparecer dois quadradinhos azuis nas extremidades deste fio, apague-o com *Delete*.

Alterando os parâmetros: Coloque o curso sobre o ator que deseja configurar e clique sobre ele com o botão esquerdo. Ou clique com o botão direito, escolhendo a opção *personalizar* > *configurar*.

Execução do modelo: Na opção *Play* você poderá verificar o fluxo de dados do projeto, como também, ao colocar um display de saída, verificar o resultado.

Etapa 2 – Experimento

Etapa 3 – Verificação Automática

Avaliação

A avaliação dos alunos será realizada em dois momentos:

Momento 1: Avaliação do tempo gasto no experimento e o total de acertos com o cenário do professor;

Momento 2: avaliação da ferramenta pelo formulário online: http://twixar.com/YEWuLspLAT6h

Referências

[Banks98] Banks, J. Principles of Simulation. In: BANKS, J. Handbook of Simulation. Georgia, Atlanta: John Wiley & Sons, Inc., p. 3-30, 1998.

[Borges06] Borges, J. A. S.; Silva, G. P. NeanderWin - Um Simulador Didático para uma Arquitetura do Tipo Acumulador. Apresentado no Workshop sobre Educação em Arquitetura de Computadores (WEAC) – Simpósio Brasileiro de Arquitetura de Computadores. SBAC, Ouro Preto, 2006.

[Branovic04] Branovic, I, Giorgi, R e Matinelli, E. WebMIPS: A New Web-Based MIPS Simulation Environment for Computer Architecture Education, Proceedings of the 31st Annual International Symposium on Computer Architecture, Munchen, Germany, 2004.

[Brito03] Brito, A. V. Sistema de Transmissão de vídeo para Vigilância utilizando Bluetooth. Dissertação de mestrado – Centro de Ciências e Tecnologia – Universidade Federal da Paraiba, Campina Grande – PB, 2003.

[Brito09] Brito, A. V. Simulação Baseada em Atores para o Ensino de Arquitetura de Computadores. Apresentado no Workshop sobre Educação em Arquitetura de Computadores (WEAC) – Simpósio Brasileiro de Arquitetura de Computadores. SBAC, São Paulo, 2009.

[Brorsson02] Brorsson, M. MipsIt – A Simulation and Development Environment Using Animation for Computer Architecture Education. Proceedings of the workshop on Computer architecture education (WCAE '02). Held in conjunction with the 29th International Symposium on Computer Architecture, New Work, USA, 2002.

[Bruschi03] Bruschi, S. M. ASDA – Um Ambiente de Simulação Distribuída Automático. Tese de Doutorado. Instituto de Ciências Matemáticas e de Computação (ICMC – USP). São Paulo. pp.7 – 246. 2003.

[Castro08] Castro, V. G. RoboEduc: Especificação de um software educacional para o ensino da robótica as crianças como ferramenta de inclusão digital. Dissertação de Mestrado em Engenharia Elétrica e de Computação – CT – UFRN, Natal – RN, 2008.

[Christensen06] Christensen, J. E.; Ribu, K. Integration of Students in the Teaching Process. In 9th International Conference on Engineering Education, 2006.

[Coutinho06] Coutinho, L. M. N; Mendes, J. L. D; Martins, C. A. P. S. Web-MHE: Ambiente web de auxílio ao aprendizado de hierarquia de memória. Apresentado no Workshop sobre Educação em Arquitetura de Computadores (WEAC) – Simpósio Brasileiro de Arquitetura de Computadores. SBAC, Ouro Preto - MG, 2006.

[Davis00] Davis, J. Order and Containment in Concurrent System Design," Ph.D. thesis, Memorandum UCB/ERL M00/47, Electronics Research Laboratory, University of California, Berkeley, 2000.

[Falcão11] Falcão, E. L; Borges, E. V. C; Andrezza, I. L. P; Silva, G. S.; Walderley, K. G; Cavalcante, E. S; Silva, H. S. Ambiente de Simulação Gráfica 3D para Ensino de Arquitetura de Processadores. XIX Workshop sobre Educação em Computação (WEI CSBC11), Natal –RN, 2011.

[Ferreira03] Ferreira, R. S.; Freitas, G.; Chieppe, U. ; Biancardi, C. Software Livre no Ensino de Sistemas Digitais e Arquiteturas de Computadores. In: Workshop Peruano de Educacion en computacion e informatica (WECI03), Lima – Peru, 2003.

[Felix06] Felix, A. F; Pousa, C. V. e Carvalho, M. B. DIMIPSS: Um simulador didático e interativo do MIPS . Apresentado no Workshop sobre Educação em Arquitetura de Computadores (WEAC) – Simpósio Brasileiro de Arquitetura de Computadores. SBAC, Ouro Preto - MG, 2006.

[Hexsel06] Hexsel, R. A, Carmo, R. Ensino de Arquitetura de Computadores com enfoque na interface Harware/Software. Apresentado no Workshop sobre Educação em Arquitetura de Computadores (WEAC) – Simpósio Brasileiro de Arquitetura de Computadores. SBAC, Ouro Preto - MG, 2006.

[Imai07] Imai, Y.,Hori, Y., Kaneko, K. e Nakagawa, M. A web-based visual computer simulator and its evaluation through a real education. Proceeding International Conference Applied Computing (IADIS2007), Salamanca, Spain, 2007.

[Koscianski07] Koscianski, A., Soares, M. S. Qualidade de Software: Aprenda as metodologias e técnicas modernas para o desenvolvimento de software. Editora: Novatec. São Paulo – SP. 2007.

[Kuller03] Kuller, R. L. Simulador de redes orientado a eventos para o sistema operacional Linux. Dissertação para obtenção do título de mestre em Engenharia Elétrica. Universidade Estadual de Campinas – Unicamp. Campinas – SP. 2003.

[Lee03] Lee, E. A. Overview of the Ptolemy Project, Technical Memorandum UCB/ERL M01/11, University of California, Berkeley, EUA. 2003.

[Lee04] Lee, E. A. and Neuendorffer, S. Concurrent Models of Computation for Embedded Software, *Technical Memorandum UCB/ERL M04/26*, University of California, Berkeley, CA 94720, July 22, 2004.

[Lee05] Lee, A. et al. Heterogeneous Concurrent Modeling and Design in Java (Volume 1: Introduction to Ptolemy II). Technical Report No. UCB/EECS-2005, 2005.

[Lee07] Lee, A. et al. Tutorial: Building Ptolemy II Models Graphically. Technical Report No. UCB/EECS-2007-129, 2007.

[Lee08] Lee, A. et al. Heterogeneous Concurrent Modeling and Design in Java (Volume 2: Ptolemy II Software Architecture). Technical Report No. UCB/EECS-2008-29, 2008.

[Lee08] Lee, A. et al. Heterogeneous Concurrent Modeling and Design in Java (Volume 3: Ptolemy II Domains). Technical Report No. UCB/EECS-2008-37, 2008.

[Lino07] Lino, A.D.P; Harb, M.P.A.; Brito, S.R.; Silva, A.S.; Favero, E. "Avaliação automática de consultas SQL em ambiente virtual de ensino-aprendizagem". 2ª Conferência Ibérica de Sistemas e Tecnologias de Informação. Porto, Portugal, 2007.

[Liu98] Liu, J. Continuous Time and Mixed-Signal Simulation in Ptolemy II, MS Report, UCB/ERL Memorandum M98/74, Dept. of EECS, University of California, Berkeley, CA 94720, 1998.

[Liu01] Liu, J., Xiong, Y. and Lee, E. A. The Ptolemy II Framework for Visual Languages. Proceedings of the IEEE Symposia on Human-Centric Computing Languages and Environments (HCC'01), 2001.

[Lopes07] Lopes, C. L. V, Ferreira, R. P. M, Silva. G. R. L. Modelo de Simulação Multiagente de uma Central de Tele atendimento. VI Encontro Nacional de Inteligência Artificial – XXVII Congresso da SBC. Rio de Janeiro. 2007.

[MacDougall87] Macdougall, M. H. Simulating Computer Systems Techniques and Tools. The MIT Press, 1987.

[Machado04] Machado, F. B, Maia, L.P. Um framework construtivista no aprendizado de Sistemas Operacionais – uma proposta pedagógica com o uso do simulador SOsim. XII Workshop sobre Educação em Computação (WEI04) – XXIV Congresso da Sociedade Brasileira de Computação (CSBC04), Salvador – BA, 2004.

[Martins03] Martins, C. A. P. S; Pousa, C. V.; Carvalho, M. B.; Penha, D. O. Computer Architecture Education: Application of a New Learning Method Based on Design and Simulator Development. Proceedings of the Frontiers in Education Conference (FIE) - 2003

[Mattos99] Mattos, M. M., Tavares, A. C. VXt: Experiência de desenvolvimento cooperativo de um ambiente didático. Congresso Iberoamericano de Educação Superior em Computação (VII CIESC). Assunção – Paraguai, 1999.

[Mec98] MEC. Diretrizes Curriculares de Cursos de Computação. Versão final disponível em www.mec.gov.br/Ftp/sesu/diretriz/Computa.doc, 1998.

[Moreira01] Moreira, G. S. A Utilização de um Ambiente de Modelagem Computacional no Ensino/Aprendizagem de Economia. Dissertação de Mestrado IM-NCE: Universidade Federal do Rio de Janeiro – UFRJ/DCC/IM/NCE.pp.17. 2001.

[Moreira09] Moreira, M. P; Favero, E. L. Um Ambiente para Ensino de Programação com Feedback Automático de Exercícios. XVII Workshop sobre Educação em Computação (WEI09) – XXIX Congresso da Sociedade Brasileira de Computação (CSBC09), Bento Gonçalves – RS, 2009.

[Moure99] Moure, M. et al. Educational apllication of virtual instruments based on reconfigurable logic. In: IEEE International 119 Confererence on Microeletronic System Education, Arlington, 1999.

[Muliadi99] Muliadi, L. Discrete Event Modeling in Ptolemy II," MS Report, Dept. of EECS, University of California, Berkeley, CA 94720, 1999.

[Morandi06] Morandi, D.; Pereira, M. C.; Raabe, A. L. A.; Zeferino, C. A. Um processador básico para o ensino de conceitos de arquitetura e organização de computadores. Revista Eletrônica PUCRS - In Hífen, Uruguaiana, v. 30, número 58, páginas 73-80, 2006.

[Nóbrega11] Nóbrega, A. L. Proposta de melhorias na usabilidade da interface do Ptolemy II, como ferramenta de ensino em arquitetura de computadores. Monografia apresentada como pré-requisito para obtenção do grau de Bacharel em Ciências da Computação no Centro Universitário de João Pessoa (UNIPÊ). João Pessoa – PB, 2011.

[Oyamada99] Oyamada, M. S. Estudo de ambientes para Co-Simulação. Dissertação de Mestrado – Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, 1999.

[Patterson05] Patterson, D. A; Hennessy, J. L. Organização e Projeto de Computadores – A Interface Hardware/Software. 3 edição. Rio de Janeiro: Elsevier Editora, 2005.

[Pegden90] Pegden, C.D. et al. Introduction to simulation using SIMAN. NY: McGraw-Hill, 2nd ed, 1990.

[Pereira08] Pereira, D. C; Filho, J. W. B; Souza, P. S. L; Souza, S. R. S. Amnésia: Simulador de Hierarquia de Memória. Workshop sobre Educação em Arquitetura de Computadores (WEAC08), Campo Grande - MS, 2008.

[Pritsker98] Pritsker, A. A. B. Principies of simulation modeling. In Banks, J (ed.) Handbook of simulation – principles, metholdoly, advances, applications and practice. New York, 1998.

[Saikkonen01] Saikkonen, R.; Malmi, L.; Korhonen, A. “Fully Automatic Assessment of Programming Exercises”. Annual Joint Conference Integrating Technology into Computer Science Education. Canterbury, Reino Unido. 2001.

[Santana90] Santana, M. J. Advanced Filestore Architecture for a Multiple Lan Distributed Computing System. Tese (Doutorado), University of Southampton. Southampton, 1990.

[Santos99] Santos, M. P. “Introdução à Simulação Discreta”. Tutorial Técnico. Departamento de Matemática Aplicada. Universidade do Estado do Rio de Janeiro. Vol. 1. pp 7-165. Rio de Janeiro, 1999.

[Scott06] Scott, M. WinMips64, version 1.5, School of Computing, Dublin City University, Ireland, 2006.

[Shruti and Loui08] Shruti, K., Loui, M. C. Ethical Issues in Computational Modeling and Simulation. Undergraduate Summer Internship Program at the Information Trust Institute, University of Illinois at Urbana-Champaign. 2008.

[Smyth98] Smyth, N. Communicating Sequential Processes Domain in Ptolemy II, MS Report, UCB/ERL Memorandum M98/70, Dept. of EECS, University of California, Berkeley, CA 94720, 1998.

[Sobreira07] Sobreira, P. L, Dórea, M. A. M, Lima, M. E, Torres, M., Motta, T. O. , Wanderley, V. C., Sena, A. C., Alencar, C. S. Competição como uma Técnica Motivacional no Ensino de Arquitetura de Computadores. Apresentado no Workshop sobre Educação em Arquitetura de Computadores (WEAC) – Simpósio Brasileiro de Arquitetura de Computadores. SBAC, Gramado-RS, 2007.

[Souto09] Souto, A. V. M.; Duduchi, M. Um processo de avaliação baseado em ferramenta computadorizada para o apoio ao ensino de programação de computadores. Workshop sobre Educação em Informática (WEI09), Bento Gonçalves – RS, 2009.

[Spolon94] Spolon, R. Um editor gráfico para um ambiente de simulação automático. Dissertação de Mestrado. ICMSC – USP. São Paulo. 1994.

[Torres11] Torres, A. L. L, Brito, A. V. Extensão do Ptolemy para o ensino de Organização e Arquitetura de Computadores, Apresentado no Workshop sobre Educação em Arquitetura de Computadores (WEAC) - Simpósio Brasileiro de Arquitetura de Computadores. SBAC, Vitória - ES, 2011.

[Vieira10] Vieira, P. V; Raabe, A. L. A; Zeferino, C. A. Bipide: Ambiente de Desenvolvimento Integrado para Arquitetura dos Processadores BIP. Revista Brasileira de Informática na Educação, Volume 18, Número 1, 2010.

[Vollmar05] Vollmar, K. and Sanderson, P., A MIPS Assembly Language Simulator Designed For Education. The Journal of Computing Sciences in Colleges, Vol. 21, No. 1, 2005.

[Wolffe02] Wolffe, G. S; Yurcik, W; Osborne, H; Holliday, M. A. Teaching Computer Organization/Architecture With Limited Resources Using Simulators. National Science Foundation, SIGCSE - ACM. Covington, Kentucky, EUA, 2002.

[Yurcik02] Yurcik, W. e Gehringer, E. F. A survey of web resources for teaching computer architecture. Proceedings of the 2002 Workshop on Computer Architecture Education (WCAE) - 29th International Symposium on Computer Architecture. ACM, Nova York, 2002.

[Zhou08] Zhou, G. Partial Evaluation for Optimized Compilation of Actor-Oriented Models. Technical Report No. UCB/EECS-2008-53, 2008.