UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de São José do Rio Preto

Fernanda Fernandes Peronaglio

Modelagem de relações entre tarefas usando interfaces gráficas aplicada em sistemas de tempo-real

São José do Rio Preto
2019

Fernanda Fernandes Peronaglio

Modelagem de relações entre tarefas usando interfaces gráficas aplicada em sistemas de tempo-real

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.

Financiadora: FAPESP - Proc 2017/06737-6

Orientador:

Prof. Dr. Aleardo Manacero Jr.

São José do Rio Preto
2019

P453m

Peronaglio, Fernanda Fernandes
Modelagem de relações entre tarefas usando interfaces gráficas aplicada em sistemas de tempo-real / Fernanda Fernandes Peronaglio. -- São José do Rio Preto, 2019
122 f. : il., tabs.

Dissertação (mestrado) - Universidade Estadual Paulista (Unesp), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto
Orientador: Aleardo Manacero

1. Ciência da computação. 2. Sistemas de computação. 3. Analise de sistemas. 4. Simulação (Computadores digitais). I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

Fernanda Fernandes Peronaglio

Modelagem de relações entre tarefas usando interfaces gráficas aplicada em sistemas de tempo-real

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.

Financiadora: FAPESP - Proc 2017/06737-6

Comissão Examinadora:

Prof. Dr. Aleardo Manacero Jr.
UNESP - Câmpus de São José do Rio Preto
Orientador

Prof. Dr. Henrique Dezani
FATEC - São José do Rio Preto

Prof. Dr. Rodrigo Capobianco Guido
UNESP - Câmpus de São José do Rio Preto

São José do Rio Preto
16 de agosto de 2019

Aos meus pais José Carlos e Rosa

À minha irmã Flávia

Agradecimentos

Agradeço primeiramente a Deus por me guiar durante toda a caminhada, permitindo-me alcançar mais esta realização e tornando possível minha chegada até aqui.

Agradeço aos meus pais, José Carlos e Rosa, por serem sempre minha base de vida, fornecendo-me apoio e suporte durante todos os meus dias. A minha irmã, Flávia, por tornar minha vida mais divertida, trazendo companheirismo e leveza a todos os momentos.

Ao meu orientador, Prof. Dr. Aleardo Manacero Jr., pela amizade e por todos os ensinamentos e direcionamentos que tornaram possível a concretização deste projeto e incrementaram meu crescimento como profissional, agradeço por toda a confiança no meu trabalho. À Prof. Dra. Renata Spolon Lobato, pela amizade e apoio durante todo este período, pela coorientação e por todos os incentivos fornecidos.

Agradeço também aos professores doutores Alexandro Baldassin e Stéphane Julia por suas contribuições apresentadas nas fases iniciais deste trabalho, por ocasião das bancas de Estudos Especiais e Qualificação.

Aos amigos do GSPD agradeço pela amizade, trabalho em equipe e compartilhamento de experiências ao longo dos meus anos como integrante do grupo.

Agradeço aos professores do programa por toda a dedicação, aos funcionários do Ibilce por toda ajuda e à Unesp por todo o suporte fornecido.

Agradeço à FAPESP pelo apoio por meio da concessão de bolsa de pesquisa, sob o processo nº 2017/06737-6, Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP).

Resumo

A análise de sistemas computacionais por meio de simulação depende fortemente de como as tarefas a serem executadas serão modeladas. Isso é ainda mais importante quando se fala de sistemas de tempo-real, em que tarefas devem ser atendidas dentro dos intervalos corretos e a análise por simulação evita perigos possivelmente presentes se executadas no ambiente real. Entretanto, simuladores de tarefas tipicamente as modelam como blocos independentes de carga de trabalho. Embora isso seja correto ou aceitável em muitas situações, isso não é sempre verdade em sistemas de tempo-real, pois normalmente aplicações desse tipo apresentam relações de dependência entre si. Em alguns simuladores isso é resolvido com a escrita do modelo de tarefa em alguma linguagem de modelagem específica, o que evidentemente dificulta o trabalho de simulação se comparado com abordagens gráficas. Desse modo, aqui se propõe uma abordagem que permita a modelagem de interações entre tarefas, em especial de tempo-real, por meio de interfaces visuais, sem comprometer a corretude do motor de simulação. A modelagem visual permite que o usuário se concentre no problema a ser avaliado e não na programação de seus detalhes, melhorando a eficiência desse processo. Na abordagem proposta se parte do conceito de árvores hierárquicas para estabelecer associações entre tarefas, elementos de processamento e recursos do sistema. As relações previstas entre tarefas incluem precedência, sincronismo, paralelismo e exclusão mútua. A validação dessa abordagem ocorre com sua aplicação no simulador de algoritmos de escalonamento RTsim, que já apresenta características interessantes de simulação, como interfaces para modelagem de escalonadores e de auxílio ao ensino. Assim como outros simuladores, o RTsim originariamente trata tarefas como blocos independentes de carga de trabalho, o que facilita a verificação dos efeitos desta abordagem na modelagem. Os resultados obtidos mostram que os procedimentos de modelagem de relações entre tarefas, na forma como definidos aqui, são aplicáveis em simuladores de eventos discretos, conforme avaliações de sua correção e usabilidade.

Palavras-chave: Modelagem de Tarefas, Escalonamento, Sistemas de Tempo-Real

Abstract

Analysis of computing systems through simulation is strongly dependent of how tasks to be executed are modeled. This is even more important in real-time systems, where tasks must be executed before their deadlines, and their simulation avoid possible risks present in the real environment. However, simulators of such tasks usually model them as independent load blocks. Although this is correct for many systems, it is not always true for real-time systems, where applications usually have dependence relationships between tasks. In some simulators this is solved by enforcing the user to write the model in some modeling or simulation language, clearly making the simulation harder when compared to graphical approaches. Therefore, a graphical approach for task modeling is proposed. This approach is specially aimed to real-time tasks and does not compromise simulation accuracy. Graphical, that is visual, modeling allows for the user to focus on the problem to be evaluated and not in programming details of the system, improving the efficiency of this process. This proposal is based on the hierarchical tree concept to establish associations between tasks, processing elements and system resources. Chosen relationships include precedence, synchronism, parallelism and mutual exclusion. This proposal is validated through its application in a simulator of real-time scheduling algorithms named RTsim, which already offers interesting modeling/simulation characteristics such as interfaces to model schedulers and to teaching. RTsim, as others simulators do, models its tasks as independent blocks, what facilitates the verification of this approach effects to modeling. The results achieved with the tests show that procedures for modeling relationships between tasks, as defined in this dissertation, are applicable to discrete event simulators, according to the evaluations of this correctness and usability.

Keywords: Task Modeling, Scheduling, Real-Time System

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AURTSS	AU Real Time Scheduler Simulator
BPMN	Business Process Modeling Notation
CHESF	Companhia Hidro Elétrica do São Francisco
CTTE	Concurrent Task Tree Environment
CTT	Concur Task Trees
DSM	Design Structure Matrix
EDF	Earliest Deadline First
GSPD	Grupo de Sistemas Paralelos e Distribuídos
HCPN	Hierarchical Coloured Petri Nets
JVM	Java Virtual Machine
LAAS-CNRS	Laboratoire d'analyse et d'architecture des systèmes
LOTOS	Language Of Temporal Ordering Specification
LST	Least Slack Time First
RTsim	Real-Time simulator
SDL	Specification and Description Language
SimSo	Simulation of Multiprocessor Scheduling with Overheads
UML	Unified Modeling Language

Sumário

Capítulo 1

Introdução

O crescente aumento na capacidade de sistemas computacionais tem possibilitado a sua aplicação em áreas cada vez mais diversas, incluindo aplicações complexas executando várias tarefas simultâneas e concorrentes. A análise de tais sistemas antes de sua aplicação efetiva depende, muitas vezes, de sua simulação. Para que os resultados dessas simulações seja efetivo é preciso que o modelo do sistema seja bastante preciso, sendo que essa precisão depende da qualidade da caracterização de sua carga de trabalho. No caso da simulação de sistemas computacionais, a carga de trabalho é representada pelo conjunto de tarefas que serão executadas no ambiente real. Esta dissertação apresenta uma metodologia para modelar tarefas e interações entre tarefas que é precisa e fácil de realizar. Os objetivos do trabalho proposto são apresentados após a descrição da relevância de sistemas concorrentes, em especial aqueles de tempo-real

1.1 Relevância da análise de sistemas concorrentes

Desde o início da informatização de processos, a computação tem se tornado cada vez mais presente no cotidiano das pessoas. A crescente demanda por agilidade e praticidade na execução de atividades de rotina faz com que diversas atividades, comuns ou complexas, sejam estudadas e analisadas para que se alcance a automação de um número cada vez maior de processos, e a consequente facilidade na obtenção de resultados. Sistemas computacionais de automação abrangem desde tarefas simples, como a substituição de atividades manuais ou rotinas auxiliares, até sistemas complexos, que executam cálculos em tempo e quantidade superior à capacidade humana ou realizam controle e monitoramento com alta precisão, como por exemplo veículos autoguiados (MURRAY, 2007).

A grande vantagem no uso de computadores na automação de processos está na velocidade de sua execução e também na sua capacidade de repetibilidade, eliminando a limitação da capacidade humana para execução de certas atividades e reduzindo o risco de possíveis falhas. Porém, ao trocar um dos componentes de um *software* complexo é

preciso verificar se a mesma muda o comportamento esperado da aplicação. Isso implica no uso de ferramentas, como simuladores, para a prévia análise do novo componente, de modo que o desenvolvedor possa identificar corretamente a decisão a ser tomada em cada situação antes de sua implementação.

Os componentes mencionados no parágrafo anterior controlam, na verdade, partes isoladas de uma atividade real. Assim, para tornar factível o processo de automação, o que se faz é tratar componentes como sendo tarefas computacionais, que serão executadas de modo sincronizado para obter os resultados esperados. Tarefas definem quais procedimentos um sistema deve executar durante uma operação, estabelecendo também quais as decisões que devem ser tomadas de acordo com a ocorrência de cada evento. Tarefas e seus relacionamentos tornam possível o estabelecimento de uma lógica sobre o seu comportamento e também na definição de um sistema, sendo característica primordial para a obtenção de resultados (FARINES; FRAGA; OLIVEIRA, 2000).

Esta proposta se justifica na crescente demanda por automação de atividades e pela grande presença de aplicações envolvendo sistemas concorrentes com compartilhamento de recursos. Como na maioria dos sistemas é inviável fornecer um recurso para cada uma das tarefas que precise executar de forma exclusiva sobre esse recurso, surgem as chamadas condições de corrida, em que atividades executadas disputam um dado recurso. A programação concorrente é a área de computação que engloba os mecanismos para tratamento de condições de corrida. Esses mecanismos é que permitem o compartilhamento de recursos e a execução de várias tarefas simultaneamente (MAZIERO, 2014). No entanto, sistemas concorrentes precisam de mais atenção e cuidado na modelagem de suas tarefas, pois devem levar em consideração a interação entre as tarefas e o uso de recursos, evitando problemas de travamento por espera (*deadlock*) ou de impossibilidade de executar por falta de recurso (*starvation*).

1.2 Objetivo

Como visto, o uso de sistemas concorrentes é cada vez maior, tornando-se presente na maioria das aplicações. Entre os vários elementos que fazem parte destes sistemas destacam-se as tarefas. São elas que realizam as atividades pretendidas e é por meio de sua modelagem, definindo suas características e possíveis formas de interação, que é possível modelar corretamente o comportamento do sistema.

Diante da importância das tarefas para o correto funcionamento do sistema, o **objetivo principal** deste trabalho foi **especificar uma metodologia eficiente para modelar de forma simples as interações entre tarefas computacionais na simulação de sistemas**. Para obter essa simplicidade, mantendo a correção do modelo, o ideal é o uso de interfaces gráficas, com manipulação visual das interações.

A avaliação dessa metodologia ocorreu com sua aplicação na simulação de tarefas de tempo-real, que sabidamente é uma área importante de sistemas concorrentes. Essa aplicação envolve a implementação das interações especificadas no simulador RTsim (MANACERO JR.; MIOLA; NABUCO, 2001). A escolha por sistemas de tempo-real se justifica pelo intenso uso desses sistemas nos sistemas computacionais modernos, em especial os embarcados, e pela característica das tarefas de tempo-real de apresentarem relações marcantes de precedência e sincronismo.

Para se atingir o objetivo descrito, incluindo a sua avaliação, foram necessários atingir alguns outros objetivos acessórios, gerando resultados adicionais ao trabalho, como:

1. Definição de métodos para converter o modelo gráfico desenhado na interface em uma especificação que possa ser simulada, contendo as características e parâmetros necessários. Essa etapa foi fundamental para a avaliação do modelo proposto, permitindo que os gráficos do modelo fossem tratados e analisados para observação dos resultados.

2. Especificação de um modelo de particionamento entre os métodos de interpretação e análise do modelo. Para tornar o modelo uma implementação simples de ser adicionada em outros tipos de ferramenta, é preciso separar ações que dizem respeito ao desenho das tarefas e suas interações das ações que geram o modelo simulável. Essa separação, com a consequente modularização do código, facilita os processos de implantação e manutenção. Além disso, é possível evitar mudanças excessivas em simuladores que não estejam preparados para trabalhar com este tipo de relacionamento.

3. Melhoria na capacidade de simulação do RTsim. Como o RTsim foi a base de validação desta metodologia, uma consequência imediata é que o mesmo passará a ter uma maior flexibilidade quanto aos tipos de sistemas e tarefas que poderão ser simulados.

1.3 Organização do texto

Este texto está organizado em sete capítulos, contando com esta introdução. No segundo capítulo são abordados sistemas concorrentes e de Tempo-Real, incluindo tarefas de tempo-real e seus simuladores, contextualizando este projeto. No terceiro capítulo se apresenta um estudo sobre a modelagem de tarefas, preparando o leitor para o entendimento desta metodologia. Os capítulos quatro e cinco apresentam o trabalho desenvolvido neste projeto, primeiro com sua especificação e depois sua implementação. O sexto capítulo apresenta os testes e a validação do projeto. Por fim, o sétimo capítulo apresenta as conclusões e trabalhos futuros.

Capítulo 2

Sistemas Concorrentes e Sistemas de Tempo-Real

Como visto no Capítulo 1, este projeto busca facilitar a modelagem e simulação de tarefas concorrentes, principalmente as de tempo-real. Portanto, antes que se discuta trabalhos semelhantes ao aqui apresentado, é necessário que se compreenda as características destas tarefas e como funcionam as suas relações.

2.1 Processos e escalonadores

A evolução de sistemas computacionais, com a crescente necessidade por maior processamento e aproveitamento dos recursos, aumentou a demanda pela construção de sistemas concorrentes. Estes são capazes de responder e gerenciar atividades simultâneas, compartilhando tempo de processador e demais recursos, melhorando o desempenho de aplicações (BACON, 1998). Exemplos típicos de sistemas concorrentes envolvem aplicações como:

- Sistemas operacionais
- Sistemas de tempo-real
- Gerenciadores de banco de dados e transações
- Computação em nuvem

Em particular, os sistemas de tempo-real são aplicações concorrentes que possuem rígida restrição em seu tempo de execução e na finalização de suas tarefas. Essa característica os torna um exemplo completo para estudo de tarefas concorrentes e suas características. Essa é uma das razões que levaram à escolha de sistemas de tempo-real como o campo para aplicação inicial da modelagem de tarefas aqui proposta.

As outras aplicações aqui apresentadas apresentam características diversas. Sistemas operacionais, por exemplo, envolvem condições de corrida entre seus mecanismos internos

e a provisão de mecanismos para que aplicações possam executar concorrentemente. Gerenciadores de banco de dados, por sua vez, são caracterizados como sistemas concorrentes devido ao seu gerenciamento de transações em consultas, feito entre vários usuários na base de dados que alteram, adicionam, excluem ou apenas consultam dados no repositório, o que de certo modo limita a possibilidade de concorrência. Aplicações em nuvem, por fim, acrescentam detalhes de concorrência relativos aos mecanismos de *middleware* que não são simples de caracterizar, embora possam facilmente ser mapeados para as mesmas estruturas de sistemas de tempo-real.

2.1.1 Condição de corrida e região crítica

O principal problema em sistemas concorrentes é a chamada condição de corrida, em que dois ou mais processos disputam o acesso a um determinado recurso. A existência de condição de corrida leva ao conceito de região crítica, que é parte do código com tratamento de sincronismo e que permite apenas um processo executando por vez. Para esta região, uma solução eficiente precisa garantir (SILBERSCHATZ; GALVIN; GAGNE, 2014):

- Exclusão Mútua - Se um processo P_i está executando sua região crítica, então nenhum outro processo pode executar a região.
- Espera Limitada - Garante um limite no número de processos que iniciam a execução de uma região crítica após outro processo já ter realizado o pedido. Fornece uma garantia de tempo finito de espera.
- Progresso - Deve ser permitida a entrada na região crítica se nenhum outro processo está utilizando.

2.1.2 Escalonadores de processos

Um elemento fundamental para o funcionamento de sistemas concorrentes é o escalonador, que é responsável por realizar o escalonamento e coordenar a ocupação do processador pelos processos (FARINES; FRAGA; OLIVEIRA, 2000). O escalonador realiza este controle seguindo um algoritmo preestabelecido, que deve conter uma lógica para troca de tarefas e atribuição de recursos. São vários tipos de algoritmos, que levam em conta fatores característicos do seu conjunto de tarefas para tomar uma decisão. Portanto, a escolha por um determinado escalonador depende do conhecimento dos atributos que caracterizam as tarefas a serem executadas.

Exemplos tradicionais de algoritmos, e as características de tarefas neles consideradas, são apresentados por Silberschatz, Galvin e Gagne (SILBERSCHATZ; GALVIN; GAGNE, 2014):

- *First-Come, First-Served*: Um dos algoritmos mais simples, funciona como uma fila comum e leva em consideração apenas a ordem de chegada, atendendo às tarefas mais antigas primeiro.

- *Shortest-Job-First*: Este prioriza as tarefas que utilizam o recurso por menos tempo, independente da ordem de chegada.

- Prioridade: Classe de algoritmos muito utilizada em sistemas de tempo-real, em que algum parâmetro da tarefa, como sua periodicidade, indica sua prioridade de execução perante as outras.

- *Round-Robin*: Faz a atribuição alternada de um tempo limite de execução para cada tarefa, levando à preempção nos casos que este tempo é ultrapassado.

Na literatura podem ser encontradas propostas bastante diversificadas de algoritmos de escalonamento. Isso ocorre pois cada aplicação prioriza um determinado comportamento, levando a algoritmos diferentes se o sistema for otimizado para uma ou outra aplicação. Essa situação é ainda mais marcante para sistemas de tempo-real, em que o atendimento de *deadlines* é essencial.

Escalonamento em Sistemas de Tempo-Real

Dada a importância do atendimento aos prazos neste tipo de sistema, é possível entender a necessidade do bom funcionamento de escalonadores de tarefas de tempo-real. O funcionamento eficiente depende das políticas de escalonamento adotadas, ou seja, das regras que devem ser seguidas na escolha de qual tarefa ocupará o processador e sobre quais condições uma ação deve ser tomada. No caso de escalonamentos de tempo-real é necessário que se entenda o comportamento do sistema para a execução do escalonador, sendo que segundo Jane Liu (LIU, 2000) a forma de execução permite separar políticas de escalonamento nas seguintes classes:

- Algoritmos Estáticos: São aqueles que estabelecem relações de prioridades em seu conjunto de tarefas, usando um determinado parâmetro, ao iniciar a execução e não as alteram durante o processo.

- Algoritmos Dinâmicos: Estes ordenam seu conjunto de tarefas no início da execução, mas podem mudar as relações de prioridade de acordo com as ocorrências do sistema.

Para a análise da eficiência de uma determinada política de escalonamento se deve definir alguns parâmetros relativos ao conjunto de tarefas a serem escalonadas e aos resultados de seu escalonamento. Assim, se pode definir:

- Tempo de Folga: Este parâmetro está relacionado à ocupação do processador, sendo que esse tempo é o período em que o processador fica ocioso, ou seja, não há tarefas aguardando para executar e também não há tarefas em execução.
- Factibilidade: Se todas as tarefas de um escalonamento podem executar e também ter seus *deadlines* atendidos, o escalonamento é dito factível, ou seja, existe escalonamento possível para o conjunto de tarefas.
- Fila de Pronto: É a estrutura em que são armazenados os dados para execução de tarefas no sistema, desde que essas tarefas estejam prontas para executar.
- Tempo de chaveamento: É o tempo consumido na troca de tarefas no processador.

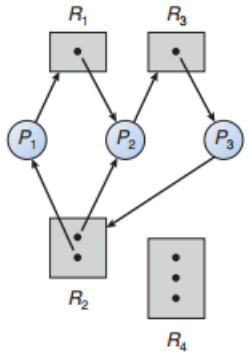
2.1.3 Processos e recursos

Para melhor compreender as características das tarefas existentes nestes sistemas, é importante conhecer como elas executam suas atividades. Para utilizar um recurso o processo deve o solicitar, utilizar e devolver ao final e, em caso de indisponibilidade, entrar em um estado de espera. Porém, esse estado de espera pode tornar-se indefinidamente longo devido ao bloqueio dos recursos, caracterizando um problema conhecido como *deadlock* (SILBERSCHATZ; GALVIN; GAGNE, 2014).

Deadlock é uma situação que surge apenas em sistemas concorrentes, pois está relacionada à disputa por recursos. Ela deve ser identificada e evitada para que permita a correta execução dos processos e não afete o resultado final. As condições necessárias para sua ocorrência são:

- Exclusão Mútua - situação em que apenas um processo pode solicitar e utilizar um recurso do sistema.
- Segura e Espera - em que pelo menos dois processos são bloqueados ao solicitarem um recurso sem liberar aqueles que estão em sua posse até obter o que solicitaram.
- Não-preempção - característica que não permite remover recurso que está em posse de algum processo, ou seja, este não pode ser removido ou ter sua execução interrompida.
- Espera Circular - relacionada com a segunda condição, ocorre quando os processos formam uma lista de espera circular, aguardando recursos que são utilizados pelo elemento seguinte da lista. Por exemplo, caracteriza espera circular quando três processos P_1, P_2 e P_3 estão esperando recursos e P_1 solicita o que está em posse de P_2, P_2 o que está em posse de P_3 e P_2 o que está sendo usado por P_1, como visto na Figura 2.1, em que ilustra-se uma situação na qual há ocorrência de *deadlock*.

Figura 2.1: Grafo de alocação de recursos com *deadlock*



(Extraído de (SILBERSCHATZ; GALVIN; GAGNE, 2014))

Em alguns casos, mesmo sem a ocorrência *deadlock*, o tempo de espera por um recurso pode se tornar extremamente alto, não permitindo que um processo, bloqueado ou não, tenha a chance de executar. Esta situação recebe o nome de *starvation*, e assim como o *deadlock* é comum em sistemas concorrentes, devendo ser tratada com o objetivo de não afetar o funcionamento correto do sistema. Observe-se que para sistemas de tempo-real a ocorrência de *starvation* causa problemas tão sérios quanto os causados por *deadlocks*.

2.2 Gerência de tarefas e sincronismo

É importante compreender como funciona o gerenciamento das tarefas para que se possa definir como elas podem interagir na execução de uma dada atividade. Maziero (MAZIERO, 2014) classifica sistemas como sendo Monotarefa (apenas uma tarefa é carregada e concluída por vez), Multitarefa (permitindo mais tarefas executando, com preempção de tarefas) e de Tempo Compartilhado (solucionando o problema de tarefas que monopolizam o processador). Para os dois últimos é interessante ter outros modos de interação entre tarefas que não seja apenas o tratamento de condição de corrida. Isso envolve soluções de sincronismo entre tarefas, quando tarefas terão suas atividades coordenadas de acordo com atividades executadas por outras tarefas, como necessitam do resultado produzido após o término de outra, ou alteram simultaneamente os mesmos dados, causando problemas de inconsistência e resultados inesperados.

2.2.1 Sincronismo na execução

Algumas soluções adotadas para o problema de condição de corrida são estruturas que fornecem controle sobre a entrada e a saída da região crítica. A solução mais utilizada neste tratamento são os semáforos, propostos por Dijkstra no início dos anos 60 (DIJKSTRA, 196_).

Semáforos são variáveis modificadas apenas pelas primitivas de bloqueio e liberação de recursos. Esta modificação é indivisível, ou seja, não pode ser feita simultaneamente por dois processos, garantindo o controle de acesso e coordenando a entrada e saída de processos da região crítica.

Por mais que semáforos sejam uma solução robusta, não são simples de utilizar, sendo comuns erros de programação e no uso das primitivas por programadores de sistemas. Alguns erros são difíceis de detectar, acontecendo em casos particulares de execução e tornando a correção mais difícil, o que torna mais interessante a sua modelagem para análise de um sistema.

Para diminuir a complexidade de programação, e consequente possibilidade de erros, outras soluções foram propostas, como por exemplo monitores (HOARE, 1974), que são abstrações de regiões críticas para o encapsulamento da lógica de bloqueio. Com isso, programadores não precisam codificar a sincronização explicitamente, melhorando os resultados obtidos.

2.2.2 Sincronismo na comunicação

Além do sincronismo no uso de recursos pelas tarefas, também se deve pensar no sincronismo utilizado na comunicação e na troca de informações entre elas. Algumas tarefas com relação de dependência entre si precisam de uma resposta ou um dado produzido para continuar sua execução. A comunicação necessária nesses casos pode assumir, segundo Maziero (MAZIERO, 2014), uma das seguintes formas:

- Síncrona - Também conhecida como comunicação bloqueante, faz o bloqueio do emissor da mensagem até que o receptor confirme o seu recebimento, e também força o receptor a ficar em espera até o recebimento da mensagem.

- Assíncrona - Ou comunicação não bloqueante, apenas retorna uma informação de erro caso um dos participantes não esteja pronto para receber a mensagem. Nestes casos é adequada a criação de um *buffer* para armazenar as informações trocadas, pois são raros os casos em que ambos estão disponíveis ao mesmo tempo para receber a mensagem.

- Semissíncrona - Semelhante a comunicação assíncrona, porém, neste caso, é estabelecido um prazo para as tentativas de transmitir e receber.

2.3 Comunicação entre processos

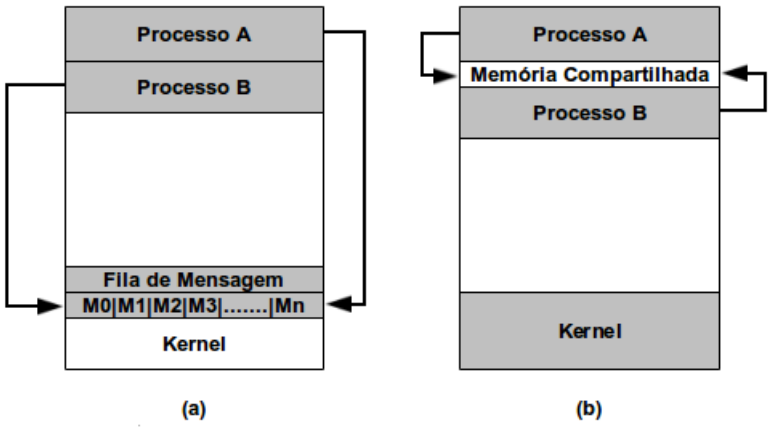
Na maioria dos sistemas concorrentes as tarefas que devem ser executadas são divididas em subtarefas interdependentes, que precisam cooperar entre si para atingir o objetivo da

execução. Para que essa cooperação ocorra é necessário que estas tarefas se comuniquem e troquem informações sobre seus resultados e seu processamento, justificando a necessidade de estabelecer métodos de comunicação. Assim, independente da necessidade de comunicação síncrona ou assíncrona, é preciso definir a estrutura a ser usada para efetivar a comunicação.

2.3.1 Formas de comunicação

De acordo com Silberschatz, Galvin e Gagne (2014), quando se trata de comunicação entre processos executando em um sistema, há duas formas fundamentais para troca de informações: Memória Compartilhada e Troca de mensagens. Na Figura 2.2 são ilustrados os dois modelos de comunicação, permitindo observar o contraste entre eles. Nela é possível verificar que em sistemas com memória compartilhada é preciso uma região em comum da memória, em que seja permitida a troca de informações (2.2b), enquanto sistemas com troca de mensagens criam canais para acesso por cada processo de forma isolada (2.2a).

Figura 2.2: Modelos de Comunicação. (a) Troca de Mensagem (b) Memória Compartilhada (SILBERSCHATZ; GALVIN; GAGNE, 2014)



(Traduzido e Adaptado pelo autor)

Sistemas com memória compartilhada necessitam de uma região comum da memória em que processos são autorizados a ler e escrever seus dados. Além disso, os processos são responsáveis pelo controle de simultaneidade das operações dentro destas regiões, garantindo que não ocorra inconsistência. Para isso, são adotados mecanismos de coordenação tratados na seção anterior, como a exclusão mútua e o sincronismo.

Ambientes com compartilhamento de memória são mais rápidos na execução, pois dados são trocados sem excessivas chamadas de sistema, o que acelera o processamento. Porém, como a informação migra entre vários níveis de cache, estes sistemas apresentam como desvantagem o problema de coerência de cache.

Sistemas de troca de mensagem adotam um mecanismo que permite a troca e sincronização de suas ações sem a necessidade de manter uma região compartilhada da memória. Esse modelo é amplamente utilizado para aplicações distribuídas, em que os processos comunicantes residem em computadores diferentes, conectados por uma rede.

Estes sistemas possuem no mínimo duas operações básicas: enviar mensagem e receber mensagem. Sendo que o tamanho das mensagens pode ser fixo ou variável, o que respectivamente implica na complexidade de implementação da tarefa e de sistema. Este modelo acrescenta alguns conceitos e características que são detalhados na seção seguinte, como por exemplo o canal de comunicação.

2.3.2 Características da comunicação

Segundo Maziero, a comunicação entre tarefas/processos pode ocorrer de duas formas: direta ou indireta. O modelo em que as primitivas possuem o formato *enviar(destino, mensagem)* e *receber(origem,mensagem)* são caracterizados pela identificação específica do emissor e do receptor da mensagem, estabelecendo a comunicação direta. Já o modelo indireto não precisa desta relação estrita entre processos, pois utiliza um canal de comunicação criado pelo sistema operacional. Desta forma, as primitivas se relacionam diretamente com o canal, como *enviar(canal,mensagem)* e *receber(canal,mensagem)*.

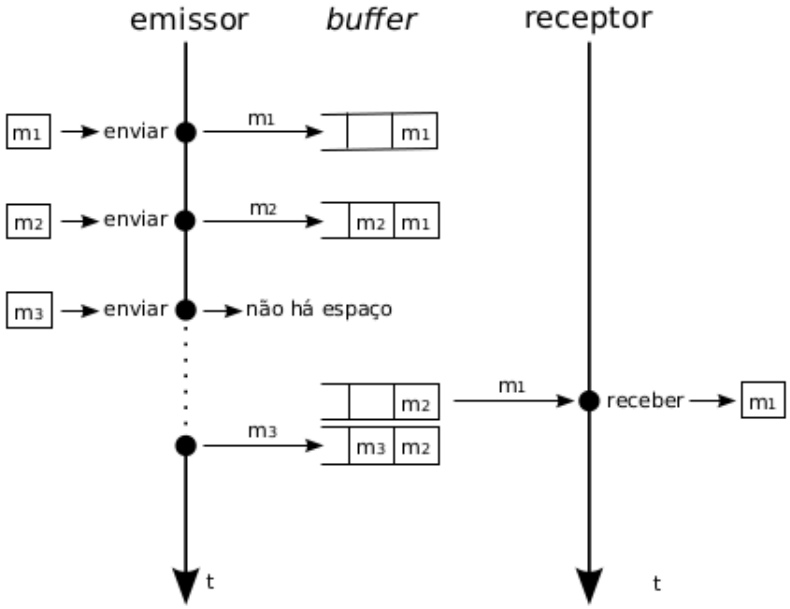
As mensagens nestes canais podem ser tratadas como sequência de pacotes. Esses pacotes podem ser tratados e recebidos de forma independente ou como um fluxo contínuo de dados. Nesse caso eles são mantidos em *buffers* no canal e lidos a critério do receptor, obedecendo sempre a sequência de envio para que seja garantida sua ordem lógica. O uso de canais de comunicação demanda a especificação de níveis de capacidade, sendo:

- Capacidade Nula: Não é possível armazenar mensagem, sendo que esta deve ser enviada diretamente ao receptor.
- Capacidade Infinita: Emissor sempre poderá enviar mensagens. Apesar de não existir na prática, é adotada para estudos e modelagem de algoritmos.
- Capacidade Finita (N): Emissor pode enviar até o limite N sem que o receptor consuma alguma informação. Ao atingir o limite o emissor será bloqueado ou avisado que o *buffer* está cheio.

Na Figura 2.3 é abordado um canal de comunicação com capacidade 2 no *buffer* de mensagens com modelo de comunicação síncrona. É possível observar que ao atingir a capacidade do *buffer* o emissor fica bloqueado até que o receptor inicie a recepção das mensagens enviadas.

Além destas características, os canais que transportam as mensagens são categorizados de acordo com a confiabilidade do sistema. Os erros possíveis de acontecer no transporte

Figura 2.3: Comunicação síncrona de canal com capacidade 2



(Extraído de (MAZIERO, 2014))

pelo canal envolvem perda de dados, perda de integridade ou perda de ordem. Nesse sentido sistemas podem ser classificados como confiáveis se mantém os dados íntegros e a ordem na qual foram enviados.

2.4 Sistemas de Tempo-Real

Considerando que o campo de aplicações concorrentes escolhido para a aplicação da proposta de modelagem de tarefas aqui apresentada envolve sistemas de tempo-real, é preciso uma análise mais cuidadosa sobre tempo-real.

2.4.1 Conceitos básicos

Com o aumento do número de atividades que necessitam de computação gerou-se a necessidade de utilizar diversos tipos de sistemas, cada um ajustado para a função que irá executar. Para atividades com restrição no tempo de execução percebeu-se a necessidade de uso de sistemas que tenham um controle rígido sobre o instante de execução de suas tarefas e sejam capazes de administrar o atendimento a todas elas.

Um sistema de tempo-real envolve aplicações que possuem restrições em seu tempo de execução, tendo prazos de atendimento para cada tarefa. Esses sistemas são normalmente usados para aplicações críticas (com tempo rígido para atendimento), que podem ter consequências graves em caso de falha ou quando o tempo de execução não é atendido

(SHIN; RAMANATHAN, 1994). Exemplos desses sistemas incluem o monitoramento de pacientes ou o controle de aeronaves (FARINES; FRAGA; OLIVEIRA, 2000). Segundo Nissanke (NISSANKE, 1997), estes sistemas podem ser classificados de acordo com sua necessidade de produzir resultados corretos atendendo seu prazo, sendo:

- ***Soft Real-Time***, ou sistemas não críticos, que em caso de ocorrência de falhas os impactos são toleráveis.
- ***Hard Real-Time***, ou sistemas críticos, em que não são admitidas falhas, ou seja, qualquer funcionamento incorreto pode gerar consequências catastróficas.

Tarefas em sistemas de tempo-real possuem todas as características dos processos concorrentes que foram exibidos no início deste capítulo. Além delas surgem características específicas ao tratamento de *deadlines* e restrições associadas, como a sua frequência de execução, que pode categorizar tarefas em:

- Periódicas: Este é o tipo de tarefa que ocorre sempre respeitando o mesmo intervalo pré-determinado de tempo entre suas ocorrências.
- Aperiódicas: Tarefas deste tipo não possuem instante de ocorrência conhecido, podendo ocorrer a qualquer momento.

Além da frequência de execução tarefas de tempo-real possuem parâmetros clássicos para sua caracterização, que são:

- Carga (ou tempo de execução): É o tempo necessário para que a tarefa complete a sua execução. Somente após a execução de toda a carga é que será considerada como completa.
- Tempo de Chegada (ou instante de ocorrência): Instante de tempo em que a tarefa se torna pronta para executar.
- Período: Intervalo de tempo em que a tarefa ocorrerá novamente, sendo relevante apenas para tarefas periódicas.
- *Deadline*: Um dos parâmetros mais importantes para tarefas de tempo-real. É o prazo em que a tarefa deve completar sua execução antes de ser considerada como falha. Existem dois tipos de *deadline*, o *deadline* relativo (intervalo de tempo entre a chegada da tarefa e seu prazo máximo para terminar) e o *deadline* absoluto (instante exato em que a tarefa deve terminar sua execução, sendo a soma do tempo de chegada com o *deadline* relativo).

2.4.2 Relações de prioridade

As tarefas de tempo-real recebem tratamento por prioridade. Esse tipo de tratamento ocorre quando cada algoritmo estabelece um parâmetro que será priorizado e classifica as tarefas de acordo com seus valores relacionados a característica que foi escolhida. Formalmente, Nissanke (NISSANKE, 1997) indica que dado um conjunto de tarefas a ordenação é estabelecida seguindo uma relação de prioridade, $\overset{p}{>} : TAREFA \leftrightarrow TAREFA$. A leitura dessa relação indica que tendo duas tarefas T_i e T_j, com $\overset{p}{>} : Ti \leftrightarrow Tj$, então T_i tem maior prioridade do que T_j.

De forma prática, a execução segue a prioridade de cada tarefa, sendo que a tarefa de maior prioridade que estiver na Fila de Pronto terá a sua execução antes das restantes, podendo ocorrer inclusive preempção de uma tarefa de menor prioridade.

2.4.3 Relações de precedência e exclusão mútua

Em alguns sistemas de tempo-real as tarefas são tratadas de forma independente entre si, o que faz com que sua ordenação dependa apenas da relação de prioridade. Porém, se houver dependência entre tarefas é preciso que o modelo de execução estabeleça relações de precedência entre tarefas executadas (FARINES; FRAGA; OLIVEIRA, 2000).

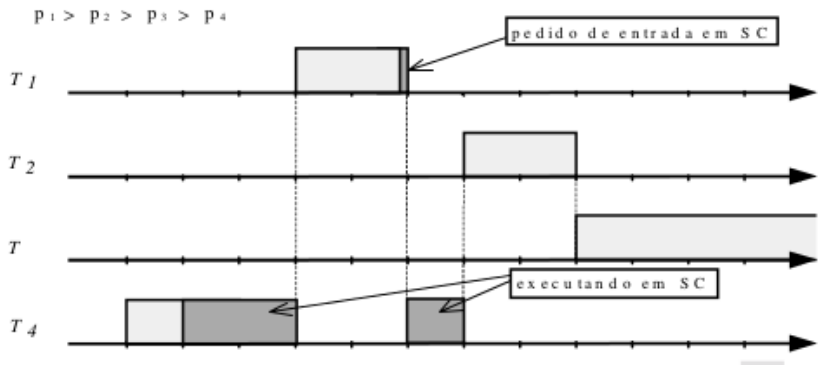
Formalmente é possível dizer que se a tarefa T_i precede a tarefa T_j ($T_i \rightarrow T_j$), então a execução de T_j é possível apenas após o término de T_i. Essa ordenação parcial pode representar dependência de informação produzida ou situações de sincronismo. São representadas normalmente por grafos orientados que determinam as relações.

Outra forma de relação entre tarefas é a de exclusão mútua, fundamental em sistemas concorrentes e com impacto importante para os escalonamentos em tempo-real. A necessidade de exclusão mútua ocorre em ambientes com compartilhamento de recursos, em que uma tarefa exclui a outra ao executar uma região crítica e ganhar o uso exclusivo de um recurso. Para sistemas de tempo-real a necessidade de exclusão pode levar à inversão das prioridades estabelecidas, permitindo que tarefas de maior prioridade sejam bloqueadas por tarefas que estão dentro desta região.

Um exemplo de inversão de prioridades pode ser visto na Figura 2.4. Ele traz um conjunto de quatro tarefas ordenadas por seu período, sendo que período de $T_1 > T_2 > T_3 > T_4$, e que T_1 é a tarefa mais prioritária. Neste exemplo há um compartilhamento de recursos entre T_1 e T_4, com T_4 entrando primeiro na região crítica, e bloqueando T_1 quando esta solicita o recurso. Este bloqueio pode levar a uma inversão de prioridade caso as tarefas T_2 ou T_3 passem a executar. A solução típica para esse problema envolve a herança de prioridades, elevando-se a prioridade de T_4 até o término de sua região crítica.

Com tarefas dependentes esta alteração de prioridade é inevitável para o tratamento, o único problema é manter um limite entre a quantidade de trocas para não tornar o escalonamento imprevisível.

Figura 2.4: Ocorrência de inversão de prioridades



(Extraído de (FARINES; FRAGA; OLIVEIRA, 2000))

Um exemplo real sobre como a inversão de prioridades pode causar resultados imprevistos é o incidente com o projeto Mars PathFinder (JONES, 1997). Naquele caso a sonda espacial, enviada para estudar o solo do planeta Marte e devolver informações para a Terra, passou a enfrentar problemas com repentinas reinicializações de sistema.

O software da sonda foi implementado sobre o kernel VxWorks, que trabalha com escalonamento preemptivo com base em relações de prioridade fixa atribuídas às tarefas. De maneira simplificada, o problema ocorreu em um caso particular envolvendo três tarefas, com prioridades baixa, média e alta, em que duas delas compartilhavam um barramento para troca de informações.

As tarefas eram responsáveis por controlar a área de transferência (Tg - Prioridade alta e curta duração), coletar dados meteorológicos (Tm - Prioridade baixa e curta duração) e transmitir dados para a Terra (Tc - Prioridade média e longa duração). Além disso, o sistema possuía um sistema de *watchdog* para evitar *deadlocks* ou *starvation*, que monitorava o uso da área de transferência, causando o reinício do sistema se estivesse bloqueada por muito tempo (MAZIERO, 2014). Na situação problemática a sequência de eventos ocorria da seguinte forma:

- A tarefa Tm perdia o processador para Tc, sem liberar a área de tranferência;
- Tg esperava a liberação do recurso, que só iria ocorrer após a exexução de Tm;
- Tm não consegue interromper Tc, que possui longa duração;
- *Watchdog* força reinicialização do sistema após um prazo determinado.

Esse é um caso particular de inversão de prioridade em que não foi possível prever a falha. A solução foi obtida a partir da adoção de um protocolo de herança, permitindo que Tm herdasse a prioridade de Tg e corrigisse o problema.

2.4.4 Escalonamento de tempo-real

A compreensão das relações entre tarefas é fundamental no estudo dos algoritmos de escalonamento, cujo funcionamento é determinante na eficiência de sistemas de tempo-real. A influência do conjunto de regras e tarefas sobre cada algoritmo pode ser evidenciada na análise de diferentes propostas, como os algoritmos Taxa Monotônica (LIU; LAYLAND, 1973), Least Slack Time First (LST) (LEUNG, 1989), Deadline Monotônico (LEUNG; WHITEHEAD, 1982), Earliest Deadline First (EDF) (LIU; LAYLAND, 1973), Servidor por Deferência (STROSNIDER; LEHOCZKY; SHA, 1995), Míope (RAMAMRITHAM; STANKOVIC; SHIAH, 1990) e Troca de Prioridade (SPRUNT; LEHOCZKY; SHA, 1988). Desses serão detalhados aqui os algoritmos Taxa Monotônica, Least Slack Time First, Míope e Troca de Prioridade. As características mais relevantes de cada algoritmo são tratadas nas próximas seções.

Taxa Monotônica (LIU; LAYLAND, 1973)

É um algoritmo estático desenvolvido para tarefas com prioridade fixa, ou seja, as tarefas recebem a atribuição de prioridade no início da execução e não alteram sua ordem durante o processo. Por ser um algoritmo simples e de fácil entendimento se tornou fundamental em todos os estudos de tempo-real.

O Taxa Monotônica ordena suas tarefas de acordo com seus períodos, sendo que tarefas com menor período terão maior prioridade. Também se considera que o *deadline* de cada tarefa é equivalente ao seu próprio período e que as cargas devem ser conhecidas. É um algoritmo preemptivo, ou seja pode remover uma tarefa do processador se receber uma de maior prioridade na fila de pronto. Ideal para tarefas periódicas e independentes.

Least Slack Time First (LST) (LEUNG, 1989)

Este é um algoritmo com prioridade dinâmica, sendo que a cada chegada de uma tarefa na Fila de Pronto o algoritmo recalcula a ordem de execução. A prioridade é calculada de acordo com o tempo de folga para execução da tarefa, sendo também um algoritmo preemptivo e idealizado para tarefas independentes.

Como os algoritmos dinâmicos muitas vezes apresentam um desempenho melhor no escalonamento, principalmente por terem a habilidade de mudar seu comportamento e ajustar melhor o escalonamento à situação atual, é importante que possa fazer parte da simulação. Mesmo sendo importante entre algoritmos dinâmicos, o LST não está implementado de modo nativo na atual versão do RTsim. Apesar disso, com o uso do módulo de geração automática de escalonadores é possível gerar o LST de modo bastante simples.

Míope (RAMAMRITHAM; STANKOVIC; SHIAH, 1990)

Os algoritmos anteriores são para ambientes monoprocessados, ou seja, ambientes com apenas uma unidade de processamento. O algoritmo Míope foi desenvolvido a partir de algoritmos heurísticos e é destinado a sistemas multiprocessados. Recebe este nome por "enxergar" apenas as tarefas que estão em sua janela de escalonamento K, que é o número de tarefas que o algoritmo considera a cada passo. O algoritmo de busca é refinado com o uso de *backtracking* caso os *deadlines* das tarefas dentro da janela não possam ser atendidos.

Para este algoritmo as tarefas são independentes, periódicas ou aperiódicas. Podem ou não utilizar recursos fornecidos por seus processadores. O Míope não é preemptivo e também não permite migração de tarefas entre processadores.

O funcionamento básico do algoritmo consiste em ordenar a lista de tarefas pendentes em ordem crescente de *deadline* relativo, seguindo com a aplicação da função heurística para os K primeiros elementos da lista. A partir daí o algoritmo cria um nova lista com as tarefas ordenadas pela função aplicada, se não for possível atender aos prazos esperados, a função é aplicada em *backtracking*, buscando pela melhor solução possível. Caso seja possível atender os *deadlines* é alocada a primeira tarefa da lista e os passos se repetem para as tarefas seguintes.

Troca de Prioridade (SPRUNT; LEHOCZKY; SHA, 1988)

Este algoritmo é uma das variações do Protocolo de Herança de Prioridade, tratando de um conjunto de tarefas dependentes com relação de exclusão mútua no acesso a recursos. O Troca de Prioridade escalona suas tarefas usando uma política qualquer, como por exemplo o Taxa Monotônica, enquanto não houver disputa por uma região crítica.

Quando uma tarefa solicita a entrada em uma região crítica que está ocupada por uma tarefa de menor prioridade, esta última herda a prioridade maior até que termine o uso do recurso. Na saída da região crítica a prioridade inicial é restabelecida e a tarefa com maior prioridade é liberada e segue sua execução.

2.5 Simuladores de algoritmos de Tempo-Real

Simulação é uma área fundamental dentro da computação. Por meio dela é possível prever o funcionamento de sistemas e realizar testes sem ter que efetuar sua construção real, evitando gastos desnecessários. A situação não é diferente para sistemas de tempo-real, em que simular é indispensável, principalmente no desenvolvimento e avaliação de sistemas críticos. Dada essa demanda vários simuladores de escalonadores de tempo-real foram apresentados ao longo do tempo. Alguns deles são discutidos a seguir, buscando evidenciar a necessidade para uma melhor modelagem das tarefas em sua carga de trabalho.

2.5.1 Uma revisão geral de simuladores de tempo-real

Esta seção apresenta os simuladores de escalonadores de tempo-real mais encontrados na literatura da área, discutindo suas características principais. É destacado também o tratamento que cada um deles dá ao processo de modelagem de relacionamentos entre tarefas.

STRESS

O STRESS (AUDSLEY et al., 1994) foi desenvolvido na Universidade de York por um grupo de pesquisa especializado em sistemas de tempo-real. Este simulador é orientado para a simulação de sistemas de tempo crítico, possuindo uma linguagem especificamente desenvolvida para modelagem de tempo-real no simulador. O software é utilizado no meio industrial e acadêmico.

A interface gráfica do STRESS permite a entrada de dados e características do sistema a ser simulado, sendo que suas três principais funções são: análise, simulação e exibição de resultados. Para que o usuário possa realizar uma simulação é necessário codificar as características do sistema em uma linguagem própria da ferramenta. Além disso, este simulador permite a manipulação de dependências no conjunto de tarefas, porém estas também devem ser codificadas.

SPARTS

O SPARTS (NIKOLIC; AWAN; PETTERS, 2011), desenvolvido em Portugal, é uma ferramenta flexível para simulação. Seu funcionamento é dividido em quatro módulos principais, sendo: Gerador de conjunto de tarefas, responsável por receber as características comportamentais do conjunto de tarefas; Gerador de tarefas, em que são definidos os parâmetros individuais de cada tarefa; Sequenciador de tarefas, módulo responsável por organizá-las para a execução; e, por fim, Ambiente de execução, em que se pode observar o escalonamento e simulação.

A estrutura modular do simulador permite seu uso para diferentes propósitos, sendo que seus módulos podem ser usados separadamente. Para utilização da ferramenta, o usuário determina características das tarefas e pode escolher entre algoritmos que estão disponíveis de forma nativa, mas não são tratadas questões de dependência.

Realtss

O Realtss (DIAZ; BATISTA; CASTRO, 2007) foi desenvolvido no Instituto de Tecnologia do México, como uma ferramenta aberta, que pode ser usada em ambientes Linux ou Windows por qualquer pessoa que deseje. Sua estrutura modular permite a inclusão de mais algoritmos na ferramenta, trabalhando com linguagens como Tcl, C e C++. O

objetivo deste simulador é testar as políticas de escalonamento existentes, sendo uma ferramenta importante de ensino e pesquisa.

Da mesma forma que o SPARTS, este disponibiliza um conjunto nativo de escalonadores para uso. A interface gráfica deste software é simples e muito explicativa, não sendo de difícil utilização e permitindo uma boa interação com o usuário. Também não há previsão de estabelecimento de dependências entre tarefas no conjunto.

AURTSS

Diferente dos simuladores vistos até aqui, o AURTSS (*AU Real Time Scheduler Simulator*) (YAASHUWANTH; RAMESH, 2010) é uma ferramenta para plataforma *Web*. Foi inicialmente projetado com o objetivo de ensinar sistemas de tempo-real e também permitir a prática de profissionais. A ferramenta possui uma interface bem intuitiva, construída de modo que qualquer usuário possa entender o que precisa ser inserido nos campos.

O simulador recebe os dados das tarefas como entrada, e as escolhas de escalonamento, como alocação de recursos ou ativar/desativar preempção, e fornece como saída o gráfico de ocorrências no tempo. Permite o uso apenas dos escalonadores já implementados no AURTSS, e recebe as tarefas como blocos independentes de carga de trabalho.

SimSo

O SimSo (*Simulation of Multiprocessor Scheduling with Overheads*) é um simulador para ambiente multiprocessado desenvolvido no LAAS-CNRS (Laboratoire d'analyse et d'architecture des systèmes) em Toulouse (CH'ERAMY; D'EPLANCHE; HLADIK, 2013). A linguagem Python foi adotada para o desenvolvimento por ser considerada pelos autores como fácil de ser aprendida e utilizada.

Para que a simulação possa ocorrer é necessário a programação manual do modelo, ou seja, é preciso que o usuário descreva o simulador por meio de uma classe programada em Python. Também considera apenas tarefas independentes.

Yartiss

O Yartiss (CHANDARLI et al., 2014) foi desenvolvido na Université Paris-Est Marne-la-Vallée, França e é um simulador para escalonamento de algoritmos multiprocessados para sistemas de tempo-real.

O simulador conta com uma interface gráfica projetada para ser intuitiva aos seus usuários. Esta ferramenta permite que o usuário estabeleça relações entre as tarefas que serão simuladas e insira novas políticas de escalonamento no simulador, porém as especificações precisam ser programadas manualmente.

2.5.2 RTsim

O RTsim (Real-Time simulator) (MANACERO JR.; MIOLA; NABUCO, 2001) é um simulador para escalonamento de tarefas de tempo-real mantido pelo Grupo de Sistemas Paralelos e Distribuídos da UNESP - GSPD, estando disponível para carregamento na página do grupo (GSPD, 2016)[1]. Seu desenvolvimento teve início em 1996, com o intuito de ser uma ferramenta de simulação que permitiria aos desenvolvedores de sistemas de tempo-real testar seus algoritmos e verificar se eram adequados para a aplicação. Após o desenvolvimento de seu protótipo observou-se uma nova aplicação para a ferramenta, o ensino de sistemas de tempo-real, resultando em um novo foco em seu desenvolvimento.

Breve Histórico

A versão inicial, implementada em linguagem C, possuía cinco algoritmos nativos para ambientes monoprocessados, recebendo mais cinco para multiprocessados após alguns anos (KEHDY, 1999). Sua interface gráfica para interação com o usuário era implementada por meio da biblioteca Xforms.

Embora propiciasse uma interface eficiente, o uso do Xforms criava problemas com portabilidade. Isto ocorria devido a dependências de outras aplicações e de gerenciamento de janelas. O principal problema é que ela dependia de gerenciadores de janelas que, em sua maioria, eram encontrados apenas em sistemas Linux. Para resolver esse problema e tornar o RTsim uma ferramenta de maior alcance, a biblioteca Xforms foi removida e foram integradas interfaces com o mesmo funcionamento da anterior, mas agora implementadas em linguagem Java, comunicando-se com o código C existente (MIOLA, 2001).

Essa solução foi descartada em seguida pela lentidão introduzida com o carregamento da JVM (Máquina Virtual Java), que ocorria a cada vez que se abria uma janela. Assim, todo o RTsim passou a ser implementado em Java, eliminando os problemas de carregamento e de portabilidade (GONÇALVES, 2005).

Estado Atual

Apesar de ter sido inicialmente pensado como um simulador para desenvolvedores de aplicações de tempo-real, hoje seu uso tem se concentrado em ensino. É utilizado principalmente para os alunos da disciplina "Sistemas de Tempo-Real" do curso de graduação em Ciência da Computação do Instituto de Biociências, Letras e Ciências Exatas (IBILCE/UNESP), tendo módulos adaptados para auxiliar o processo de aprendizagem. A versão atualmente disponível possui, de modo nativo, cinco algoritmos monoprocessados e três multiprocessados, vistos na Tabela 2.1

Além dos algoritmos implementados de forma nativa, a ferramenta recebeu um módulo

[1] http://www.dcce.ibilce.unesp.br/spd

Tabela 2.1: Algoritmos nativos no RTsim

Ambientes Monoprocessados	Ambientes Multiprocessados
Taxa Monotônica	Míope
Servidor por Deferência	Leilão
Servidor Esporádico	Leilão Focado
Topo de Prioridade	
Troca de Prioridade	

de geração automática de escalonadores (PERONAGLIO, 2017; PERONAGLIO et al., 2017). Nele é permitido, por meio de uma interface gráfica, que o usuário selecione alguns parâmetros de escalonamento pré-estabelecidos e os relacione na forma de uma equação matemática que representa a política do escalonador. O módulo então gera automaticamente o código Java para esse escalonador. Com isso é possível definir, implementar e simular novas políticas de escalonamento no simulador sem a necessidade de codificá-las.

O RTsim pode ser executado de três formas distintas, sendo que o usuário escolhe uma delas em algum momento antes de solicitar a produção de resultados. As formas de execução são:

- Plotar Gráfico: aqui é gerado um gráfico detalhado do escalonamento, para que o usuário possa acompanhar as ocorrências das tarefas. A interface é completa tanto no ambiente monoprocessado como no multiprocessado, sendo que neste se separa os gráficos por processador. Também oferece opções de *zoom*, régua e salvar o gráfico.

- Análise: Interface de saída que apresenta apenas estatísticas sobre as ocorrências, como por exemplo, tarefas que perderam *deadline*.

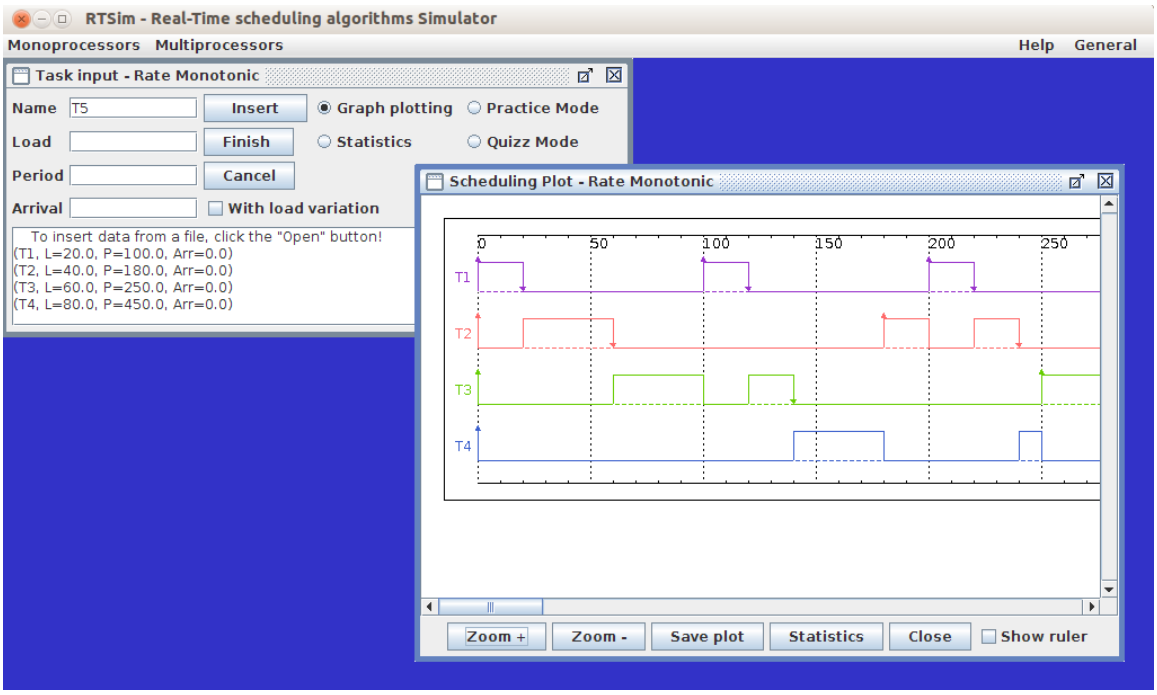
- Modo Aprendizado e Modo Teste: São as funções mais importantes para o uso em ensino. Com elas é possível que o usuário entre manualmente com o escalonamento e receba a correção do mesmo. É possível exibir os erros que o usuário cometeu ao tentar escalonar e também fornecer ajuda durante a entrada do escalonamento no modo aprendizado.

O uso durante as aulas ministradas com o apoio do simulador tem se mostrado de grande auxílio para os estudantes. O conteúdo visto em aula pode ser praticado e testado na ferramenta, o que torna a compreensão do comportamento dos escalonadores mais fácil, principalmente por possibilitar a visualização gráfica das tarefas sendo executadas.

A Figura 2.5 exemplifica a entrada de dados do Algoritmo Taxa Monotônica (parte superior) e a saída resultante da opção “Plotar Gráfico” (parte inferior). No canto esquerdo da Figura 2.5 está a entrada de dados, na qual é possível observar os parâmetros de entrada que são solicitados, as opções que podem ser escolhidas e também é possível ver um registro de todas as tarefas que já foram inseridas. No canto direito está o gráfico

do escalonamento que exibe as execuções de cada tarefa, apresentando ainda botões para controle da visualização e estatísticas para auxiliar o usuário.

Figura 2.5: Interfaces do RTsim para o algoritmo Taxa Monotônica



(Gerada pelo autor)

A entrada de dados no RTsim pode ser feita pela inserção de cada tarefa manualmente, usando janelas diferentes, ajustadas para os diversos tipos de tarefa, ou pela leitura de arquivos contendo as informações necessárias. Já a saída de resultados é feita tanto de forma gráfica como em arquivos. Dois arquivos são produzidos automaticamente pelo simulador, sendo um com o rastro da simulação e outro com dados das tarefas de entrada. Este último pode ser usado futuramente como arquivo de entrada de uma simulação.

2.6 Análise qualitativa dos simuladores

Matsubara *et al.*, ao proporem uma nova ferramenta de simulação (MATSUBARA et al., 2012), especificam uma série de características relevantes na simulação de sistemas de tempo-real. Em seu artigo fazem uma análise sobre o oferecimento dessas características em alguns simuladores. Eles elencam um conjunto de seis requisitos práticos, abrangendo desde a modelagem até o resultado final.

Do estudo feito por eles e da análise vista na seção 2.5.1 foi possível identificar que os simuladores deixam de atender requisitos diversos para um sistema de tempo-real. Isso mesmo considerando que existem simuladores com diferentes propósitos e diversas características, cada um atendendo a um tipo de demanda de simulação. A falta de atendimento das características definidas, principalmente no que diz respeito às tarefas,

pode-se concluir que a abrangência e funcionalidade dos simuladores é diminuída.

O primeiro requisito identificado por Matsubara *et al.* trata da modelagem de uma tarefa na simulação. Um simulador deve permitir a configuração dos parâmetros de uma tarefa, como por exemplo as suas execuções e suas variáveis. De formas diferentes, todos os simuladores listados neste capítulo permitem a inserção de um conjunto de tarefas e a configuração de suas características, seja por meio de uma interface gráfica ou programação em linguagem específica.

A simulação de relações de dependência entre tarefas, que já teve a sua importância destacada neste trabalho, é o segundo requisito. Este aborda a necessidade de acomodar as tarefas dependentes e suas características (variáveis compartilhadas, exclusão mútua, entre outras) para trazer mais funcionalidade ao simulador. Dos simuladores listados apenas STRESS e Yartiss atendem a este requisito atualmente, porém com o custo de que tais relações devem ser manualmente programadas. Nenhum simulador oferece esta funcionalidade de forma gráfica, o que seria interessante para facilitar o processo de modelagem.

O terceiro requisito trata da modelagem de eventos assíncronos, como interrupções e troca de mensagens. Para atender esta demanda é necessário manter o controle do usuário sobre os tempos de ocorrência dos eventos. Apesar de pela descrição do STRESS deixar a entender que se pode modelar troca de mensagens, não existe indicação de como isso seria realizado pelo usuário. Os demais simuladores não apresentam essa funcionalidade. Já do ponto de vista da modelagem de interrupções, o que se percebe é que permitem a ocorrência de preempções no escalonamento, mas não a existência de um evento de interrupção.

Os demais requisitos identificados em (MATSUBARA et al., 2012) abordam a existência de escalonadores nativos ou a possibilidade de inserir facilmente novas políticas de escalonamento; facilidade de modelagem da aplicação de tempo-real; e a forma de análise dos resultados da simulação. Os seis requisitos se justificam na busca por um simulador que atenda uma demanda maior por informações e que seja condizente com o que ocorre na real execução de um sistema de tempo-real.

Na análise feita sobre os simuladores descritos na seção 2.5.1 é possível perceber que uma lacuna mais intensa diz respeito ao modo como se modelam tarefas. A Tabela 2.2 resume as características principais identificadas nesses simuladores. Nela é possível observar que quanto ao tratamento de interações entre tarefas, apenas os simuladores STRESS e Yartiss permitem a manipulação de tarefas dependentes, porém exigindo sua programação em uma linguagem específica. A necessidade de codificação do modelo de tarefas ocasiona uma perda no foco principal da modelagem, que é o comportamento do conjunto escalonador-carga de trabalho.

O trabalho proposto aqui busca atender a estes requisitos para modelagem e simulação de sistemas concorrentes, mas também oferecer isto por meio de uma interface gráfica.

Tabela 2.2: Características básicas dos simuladores avaliados

Simulador	Escalonadores Nativos	Inserção de escalonadores	Tipos de tarefas	Modelagem de tarefas
SimSO	sim	script	independentes	GUI/script
STRESS	sim	script	independentes/ dependentes	script
Yartiss	sim	script	independentes/ dependentes	script
Sparts	sim	script	independentes	script
Realtss	sim	script	independentes	GUI
AURTSS	sim	não	independentes	GUI
RTsim (GSPD)	sim	GUI	independentes	GUI

Isso torna a tarefa de criar um modelo mais intuitiva e menos trabalhosa. A construção em ambiente gráfico permite uma rápida adaptação aos elementos do modelo, sem a necessidade de compreender detalhes do processo de simulação, permitindo manter o foco na análise do sistema concorrente sendo simulado.

2.7 Considerações finais

A partir do exposto neste capítulo é possível observar a importância que interações entre as tarefas apresentam para permitir o correto entendimento de um sistema. Essas interações, que envolvem desde condições de corrida até relações de dependência e troca de informações, representam um problema adicional quando se faz a modelagem e simulação de sistemas concorrentes.

Embora modelar tarefas como blocos independentes de carga de trabalho satisfaça os objetivos de boa parte dos analistas, é necessário observar que algumas aplicações precisam modelar tarefas com base nas relações de dependência e comunicação. Não foi possível encontrar na literatura específica um simulador que atenda de modo simples a esse requisito, o que leva ao trabalho aqui apresentado, em que se propõe um modelo gráfico para descrever estas relações e interações.

Capítulo 3

Um Estudo sobre Modelagem de Tarefas

No capítulo anterior ficou claro que o tratamento de dependências entre tarefas melhora os resultados de uma simulação. Ficou claro também que poucos simuladores habilitam esse tratamento e, quando o fazem, o fazem por meio de programação adicional para modelar dependências. Nesse capítulo se apresenta um panorama sobre a modelagem de tarefas no cenário atual, seguindo com uma revisão de projetos inseridos na mesma proposta e da solução adotada por este trabalho.

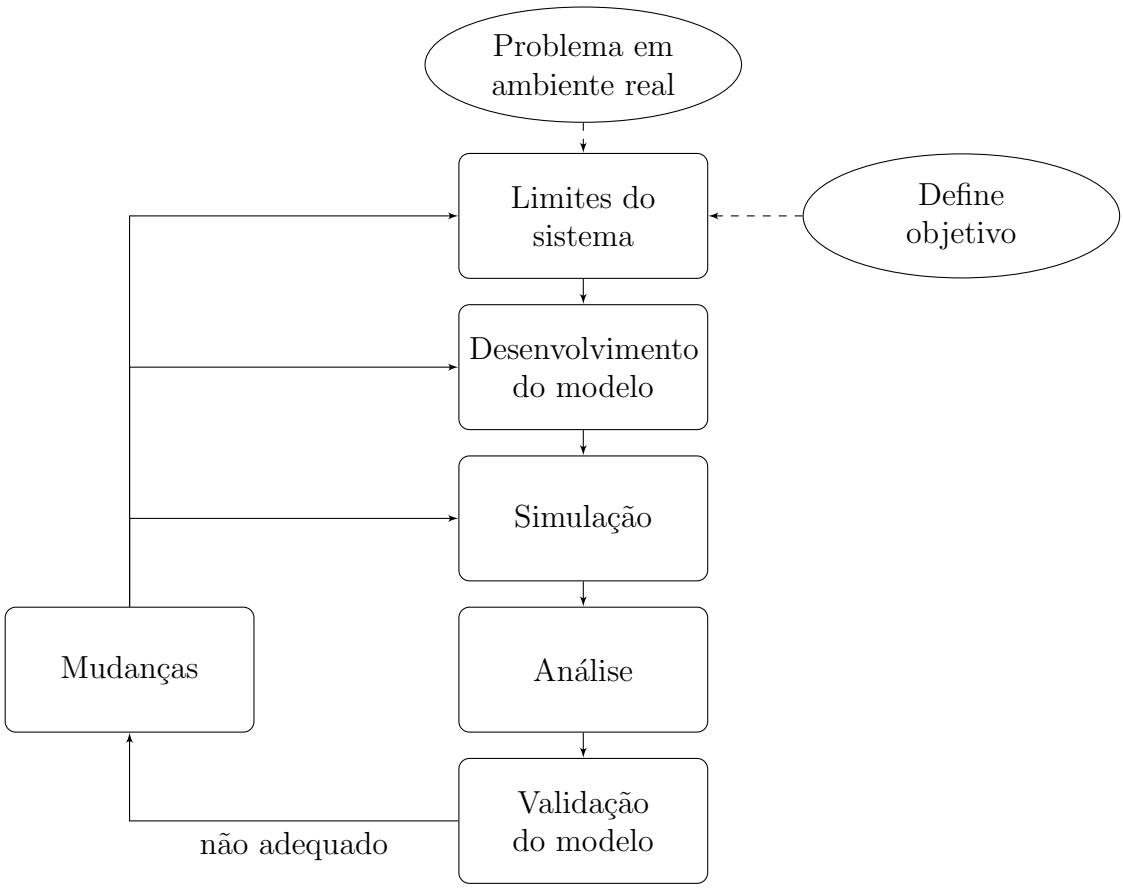
3.1 Conceituação de modelagem

O processo de modelagem é a recriação do comportamento do sistema por meio de um modelo abstrato, correspondendo à descrição do funcionamento de um sistema real ou de seus elementos por meio de algum tipo de ferramenta. A modelagem pode ocorrer usando variadas formas de representação, como por exemplo modelos matemáticos, protótipos ou lógicos (CHATURVEDI, 2017).

Todas as técnicas de modelagem buscam a melhor descrição do sistema real, de modo a imitá-lo com precisão. A construção de modelos é feita para se possa fazer a análise, usualmente por simulação, de um sistema sem a execução de sua implementação real. Este processo é aplicável mesmo em casos em que a exploração do sistema em ambiente real é inviável, permitindo que se faça testes do sistema em diferentes condições, ou que se conheça o funcionamento de um sistema antes de efetivamente construí-lo.

O processo de análise de um sistema envolvendo modelagem pode ser visto na Figura 3.1. Nela é exibida a metodologia básica para a construção de um modelo do sistema real, incluindo etapas para simulação e modificações no modelo. Deve ser observado que os tipos de modelo variam de acordo com a necessidade dos objetivos de análise, podendo ter vários tipos de representação.

Figura 3.1: Processo de modelagem (CHATURVEDI, 2017)



(Traduzido e Adaptado pelo autor)

Neste trabalho o foco é na análise de tarefas concorrentes por simulação. Isso caracteriza sistemas a eventos discretos, que têm seu comportamento definido por ocorrências de eventos em determinado tempo no seu espaço de execução. De modo geral, eventos em um sistema são ações específicas que o levam de um estado a outro (CASSANDRAS; LAFORTUNE, 2009).

Há várias técnicas e formas de modelagem para este caso de estudo, como por exemplo as Redes de Petri (CAO; HO, 1990). A revisão apresentada a seguir envolve principalmente técnicas de modelagem de dependência entre processos, de modo a possibilitar comparações entre este projeto e outras construções existentes.

3.2 Metodologias para modelagem de dependência entre processos

As metodologias abordadas nesta seção tem objetivo semelhante ao proposto neste trabalho, servindo para compreender as variações e as necessidades dentro da área de modelagem de dependências. Técnicas de modelagem com objetivo final mais próximo da

proposta deste trabalho serão apresentadas nas próximas seções.

Grafos de fluxo de execução

Uma primeira abordagem se refere a técnicas de análise de dependência por meio dos grafos de fluxo de execução (CHEN et al., 2000). Tais grafos são frequentemente utilizados para representar o comportamento de um sistema. Neste caso em particular os grafos servem para a representação de programas concorrentes desenvolvidos em linguagem Ada, que pela sua característica permitiria adaptações da proposta para outras linguagens.

A construção dos grafos não pode ser aplicada diretamente a sistemas concorrentes, principalmente pela existência de compartilhamento de recursos e relações entre tarefas que devem ser respeitadas. Por este motivo a proposta sugere uma análise particular dos grafos para esta demanda, com o desenvolvimento de um algoritmo para análise dos elementos do grafo.

O trabalho desenvolvido por Chen *et al.* (CHEN et al., 2000) traz uma análise matemática que pode ser aplicada a programas concorrentes. Apesar de não ter desenvolvido uma construção em ambiente gráfico, é importante destacar que este tipo de proposta serve para otimização, análise e teste de programas concorrentes como um todo.

Diagramas de dependência

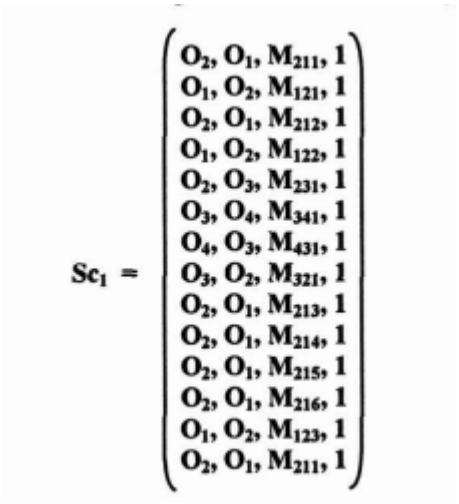
O desenvolvimento dos diagramas de dependência (VASILACHE; TANAKA, 2005) é uma abordagem mais relacionada à análise comportamental das interações do sistema. Estes diagramas foram desenvolvidos para que se pudesse representar interações entre cenários de execução, sendo que cada cenário é um conjunto de eventos que podem ocorrer durante uma operação particular do sistema.

Com base em outros modelos Vasilache e Tanaka propuseram uma notação gráfica para representar seu diagrama considerando interações e relacionamento entre seus cenários. As dependências tratadas são divididas em: Dependência temporal, em que há uma relação de tempo entre a execução dos cenários; Causa-Efeito, que relaciona a execução de um cenário a uma condição pré-estabelecida; e por fim, Generalização, em que há uma abstração do cenário, pois este parte de um elementos maior.

Cada cenário é representado por uma sequência de interações, ou trocas de mensagem, representada matricialmente. A Figura 3.2 trás um exemplo dessa matriz, em que para cada linha os elementos O_i e O_j representam participantes de uma troca; M_{ijk} representa a mensagem k trocada; e o número 1 identifica mensagens síncronas.

A partir desta representação em matriz é possível seguir um algoritmo para construir a representação em máquinas de estado, sendo uma para cada cenário tratado. O procedimento também envolve a síntese de todas as máquinas construídas em um resultado final.

Figura 3.2: Matriz de troca de mensagens



(Extraído de (VASILACHE; TANAKA, 2005))

A modelagem de interações desenvolvida por este método foca no comportamento do sistema como um todo e na sua consequente representação como máquina de estado. Com isso ele apoia a análise de requisitos e o processo de desenvolvimento do sistema, permitindo a observação de quais cenários são afetados por mudanças. Um efeito adicional é que permite também a geração de diversos tipos de teste a partir das máquinas de estado.

Modelagem por grafos de dependência

O trabalho desenvolvido por Ferrante *et al.* (FERRANTE; OTTENSTEIN; WARREN, 1987) faz a modelagem por grafos de dependência, buscando sua aplicação em otimização. A pesquisa trata das características comportamentais deste grafo e da sua representação no controle da dependência de dados entre os elementos.

Pode-se destacar desta pesquisa exemplos de otimização na modelagem de dependências pelo grafo. As otimizações envolvem a detecção de elementos paralelos e a determinação de particionamento entre nós. Essas operações apoiam a modelagem multiprocessada sem prejudicar a dependência, entre outras técnicas.

Statecharts

De um modo mais amplo, o trabalho desenvolvido por Harel (HAREL, 1987) traz uma abordagem baseada no diagrama de transição de estados para representação de sistemas. A abordagem por *statecharts* propõe um modelo formal de diagrama, capaz de abordar conceitos de hierarquia, concorrência e comunicação.

Sem a ideia explícita de modelo hierarquizado, *statecharts* tratam o conceito de representação de hierarquia por encapsulamento, colocando um estado dentro de outro para representar uma relação. O conceito de ortogonalidade no diagrama é usado para tratar

independência/concorrência entre tarefas. Neste modelo são permitidos arcos direcionais para representar as ações do sistema.

Apesar da abordagem permitir que seja modelado o funcionamento de um sistema complexo, o refinamento de relações apenas entre as tarefas não é destacado neste trabalho, permitindo uma visão apenas dos estados do sistema.

Diagramas UML

Uma abordagem muito conhecida para a modelagem é a utilização de diagramas UML (*Unified Modeling Language*) (CLARK; EVANS, 1997). Esta é uma linguagem específica para modelar objetos de um sistema. Muito utilizada em processos de engenharia de software, o conjunto de diagramas produzidos por esta linguagem permitem a análise do projeto e construção do sistema, permitindo que o modelo forneça visualizações do comportamento dos eventos.

Com representações gráficas específicas para cada tipo de diagrama, representam modelos estáticos e comportamentais dos sistemas. Os modelos comportamentais são responsáveis por representar as ocorrências dinâmicas no sistema, como as trocas de mensagem. A especificação UML é extremamente completa, sendo um modelo de representação consagrado.

Seguindo essa linha Kohler *et al.* (KöHLER et al., 2000) propuseram uma metodologia para modelagem de processos em ambiente de controle de produção. A proposta utiliza uma associação da linguagem de descrição SDL (*Specification and Description Language*) com diagramas UML, visando criar uma linguagem de programação visual que possa ser executada, fornecendo geração de código. Cada elemento associado é mutuamente dependente do outro na associação, controlando o comportamento de cada diagrama.

A especificação alcançada por este trabalho permitiu uma modelagem adequada para o cenário proposto, extraindo as vantagens de cada tipo de diagrama. Foi permitida também a geração de código Java com base no diagrama, por meio de ferramenta específica.

DSM - Matriz de estrutura de projeto

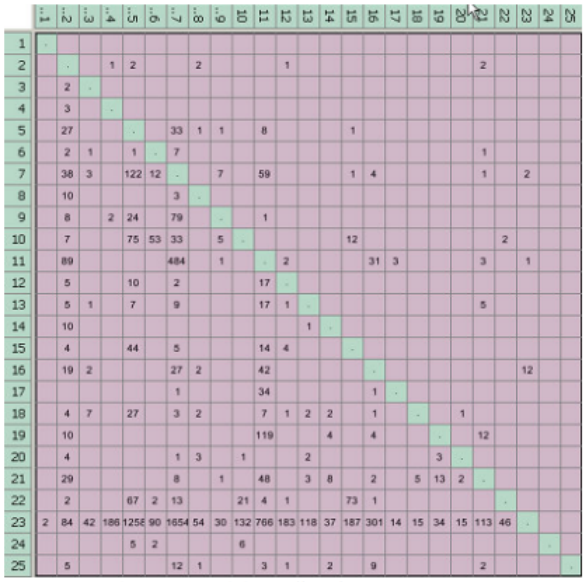
Um modelo DSM (*Design Structure Matrix*) é um tipo de representação visual do sistema na forma de uma matriz quadrada, buscando representar dependências entre atividades diversas (EPPINGER; BROWNING, 2016; DSM, 2019). Como DSM tem uma aplicabilidade bastante aberta, é possível usar sua metodologia em aplicações de desenvolvimento de software que, como já mencionado, apresentam semelhanças com o escalonamento de tarefas computacionais.

Nessa linha se encontra o trabalho de Breivold *et al.*—, que faz a proposta de uma abordagem para modelar o desenvolvimento de arquiteturas de software com dependência de atividades (BREIVOLD et al., 2008). Neste caso se aplica modelos de dependência

para identificar pontos de correlação entre módulos de uma arquitetura de software. O trabalho se concentra nesses pontos pois a modularização é um processo importante de produção, fornecendo suporte a mudanças.

A matriz apresentada na Figura 3.3 aborda as relações resultantes da aplicação da abordagem por eles proposta em um caso de estudo. O processo de reestruturação ocorre após a análise do modelo de dependências.

Figura 3.3: Exemplo de matriz gerada para análise de dependência



(Extraído de (BREIVOLD et al., 2008))

Ainda usando DSM se encontra o trabalho desenvolvido por Sangal *et al.* (SANGAL et al., 2005) para tratar dependência entre processos relacionada a arquitetura de software. Ela utiliza uma matriz DSM para rastrear e analisar os tipos de dependência entre os componentes no processo de modelagem.

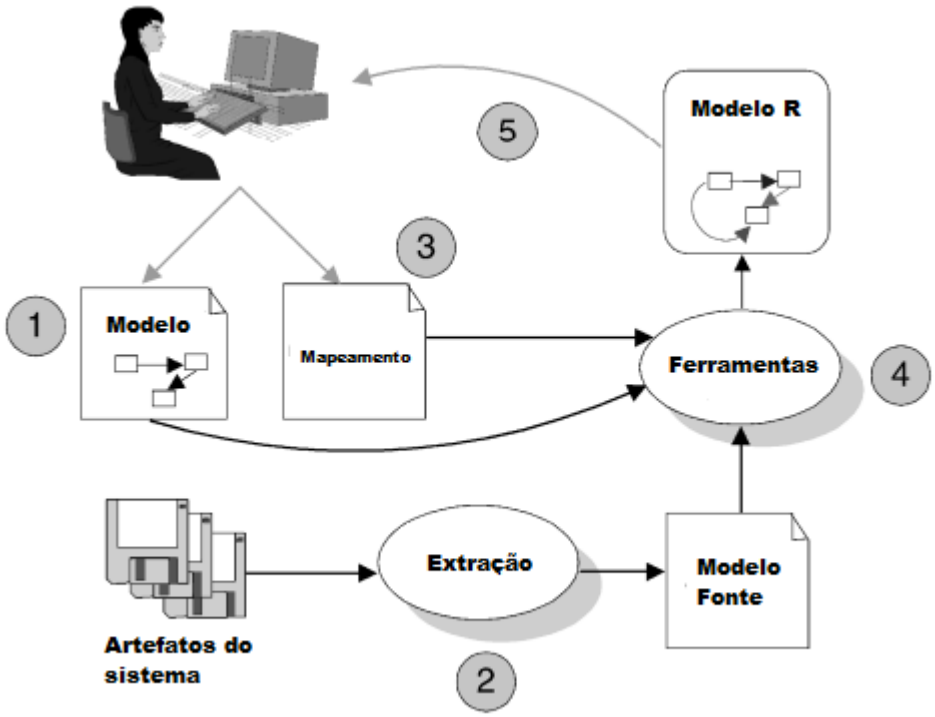
A construção da matriz permite um rastreamento da arquitetura do software, o que gera uma melhor compreensão dos objetivos do projeto e sobre as regras de implementação. Em particular, a proposta apresenta trata da aplicação de algoritmos de particionamento e agrupamento sobre a matriz gerada, reduzindo ciclos e dependências.

Modelos de reflexão de software

A técnica conhecida como *Software Reflexion Model* também é um método de modelagem para apoiar engenheiros de software durante o processo de manipulação de componentes (MURPHY; NOTKIN; SULLIVAN, 2001). O procedimento envolve um mapeamento estrutural do software e uma construção de modelo de alto nível, trabalhando para inferir características comuns de configuração para o caso de estudo.

Na Figura 3.4 exibe-se uma representação do processo de modelagem por esta técnica.

Figura 3.4: Aplicação de *Software Reflexion* (MURPHY; NOTKIN; SULLIVAN, 2001)



(Traduzido e Adaptado pelo autor)

Os passos iniciais são a criação de um modelo base pelo usuário, a extração das estruturas do sistemas e o consequente mapeamento, fornecido pelo usuário, entre um e outro. A aplicação de ferramentas específicas produz o resultado esperado da análise.

Esse processo de análise tem foco semelhante ao do uso das matrizes DSM. Relacionados a engenharia de software estes métodos de modelagem permitem gerar um conhecimento sobre o comportamento do código e da estrutura dos programas, além de permitir a identificação de pontos de melhoria.

Por mais que os objetivos sejam diversos do que se propõe neste trabalho, a apresentação destas técnicas permite a visualização do cenário de modelagem em diversas áreas de aplicação, compreendendo a demanda por técnicas e ferramentas.

Análise formal

O trabalho apresentado por Ivkovic e Kontogianiss (IVKOVIC; KONTOGIANNIS, 2006) aborda o uso de análise formal para modelar o comportamento do sistema. Esse uso visa identificar os pontos de dependência de forma automática, tornando mais simples a análise do código. O desenvolvimento do método inclui a criação de metamodelos e modelos intermediários, com a utilização do formalismo para descrever as dependências, para que possa ser validado.

Automação de dependências entre rotinas

Voltado para apoiar a manutenção de processos de software, o trabalho de Loyall e Mathisen (LOYALL; MATHISEN, 1993) faz uma análise sobre as dependências entre rotinas de um sistema e propõe uma ferramenta para automatizar esta atividade. A proposta do sistemas é fornecer a análise de dependências, o que indicaria as partes afetadas por mudanças, execução automatizada de testes, checagem e avaliação dos métodos de teste.

O foco principal da proposta era a de promover uma fácil manutenção do software, considerando as técnicas de identificação de dependência. O protótipo desenvolvido foi testado em códigos específicos, desenvolvidos em Ada.

3.2.1 Considerações sobre as metodologias apresentadas

Os trabalhos apresentados nesta seção fazem parte do levantamento de técnicas e ferramentas para modelagem de processos. Como pode ser observado, as metodologias são aplicadas em diversas áreas, porém com maior frequência na Engenharia de Software. Uma conclusão que se pode tirar é que a utilização de modelagem em processos computacionais é bastante útil e que a dependência entre tarefas é um problema importante a ser resolvido para testes e construção do sistema real.

Entretanto, a proposta deste projeto é permitir a modelagem de dependências de forma simplificada. Diferente destes trabalhos, aqui se busca a expressão destas relações de modo mais direto, usando interfaces gráficas para tanto. Dentre as abordagens apresentadas se percebe que várias fazem uso de grafos, ou variações destes. O problema delas é que os processos de modelagem descritos não fazem uso de recursos gráficos para a construção das dependências.

3.3 Modelagem usando Redes de Petri

Uma abordagem importante para modelagem de tarefas concorrentes, e principalmente das tarefas de tempo-real, envolve as Redes de Petri (PN). O uso dessa ferramenta para a modelagem é relativamente natural em virtude de que sua proposta original (PETRI, 1966) tratar explicitamente de processos de comunicação entre processos concorrentes. Embora sejam muitas as propostas de modelagem usando PN, serão apresentadas aqui apenas duas delas, uma vez que o foco deste trabalho não envolve as PN.

Modelagem com redes de Petri coloridas hierárquicas

Neto, Turnell e Santoni (NETO; TURNELL; SANTONI, 2007) propuseram uma metodologia prática para a modelagem de atividades de uma companhia. Mais especificamente, a modelagem foi aplicada sobre tarefas de manobra operacionais em uma subestação

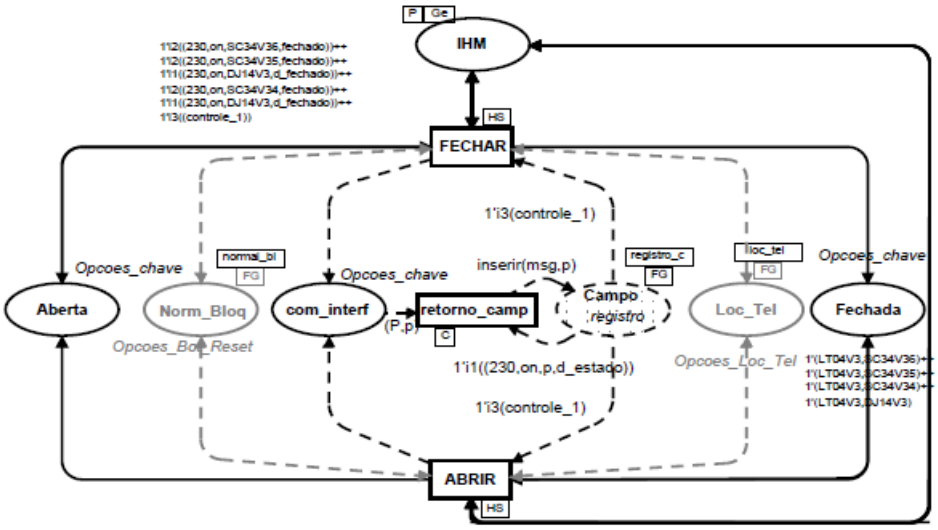
elétrica, em projeto da CHESF. Manobras são ações tomadas para manter o sistema em funcionamento normal ou por intervenção em algum evento, podendo ser feitas manualmente por um operário ou por meio de sistemas automatizados.

O objetivo era conseguir observar os erros e simular cenários possíveis nas diversas aplicações de manobras, obtendo esta análise por meio do processos de modelagem. Para isso, a simulação deveria ocorrer sobre um modelo construído, que seria composto pela modelagem das manobras realizadas e as suas relações.

O processo de modelagem adotou o modelo de Redes de Petri Coloridas Hierárquicas (HCPN) (JENSEN, 2013). Para este trabalho, seu principal foco foi permitir a antecipação de erros e a análise de variações na execução de tarefas. Em particular o método foi aplicado em uma manobra para liberação de disjuntor.

A manobra foi analisada e mapeada para um modelo específico, para que fosse possível simular as diversas situações, como visto na Figura 3.5. Para esta HCPN é possível observar a representação de ações do operador representadas por fichas e as transições como eventos executados sobre a chave.

Figura 3.5: Modelo de elemento construído para o trabalho



(Extraída de (NETO; TURNELL; SANTONI, 2007))

O modelo foi construído com a ajuda do DesignCPN, que é uma ferramenta para trabalhar com redes coloridas. A análise realizada pelos pesquisadores aplicou algoritmos de busca por caminhos adequados, encontrando algumas variações para realização da manobra. Foi possível observar erros e ações desnecessárias em determinados fluxos de execução, destacando o potencial de falhas humanas e erros de rotina de manobra. Do ponto de vista da facilidade de modelagem é necessário dizer que a técnica demandou conhecimentos aprofundados em Redes de Petri e suas regras de construção.

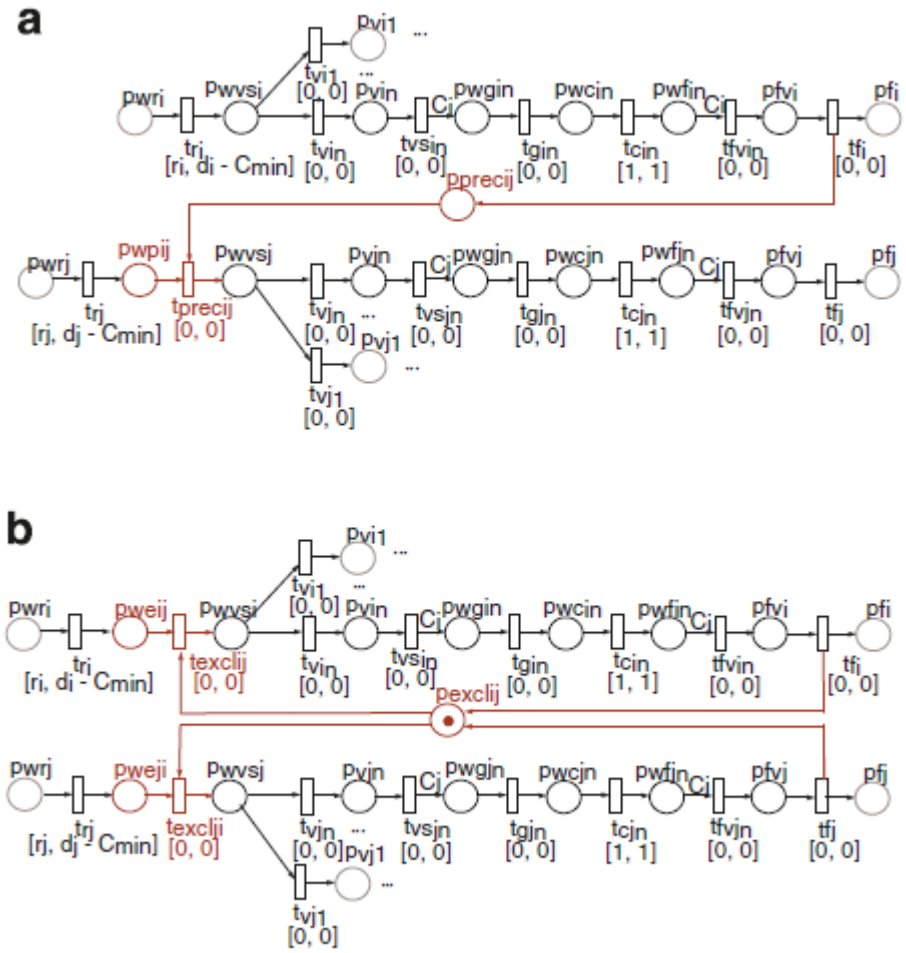
Modelagem com redes de Petri temporizadas

Ainda na linha de modelagem com Redes de Petri é possível encontrar a proposta de Tavares, Maciel e Silva (TAVARES; MACIEL; SILVA, 2008). A técnica adotou as redes de Petri temporizadas para conseguir uma descrição formal do sistema e construir seu escalonamento considerando, entre outras características, relações entre tarefas.

Os objetivos deste trabalho estavam relacionados a produzir, por meio de rede de Petri temporizada, modelos para tratamento de tarefas periódicas críticas de tempo-real que se relacionam e têm restrições de consumo de energia. O modelo simulado deve atender as restrições de execução das tarefas e reduzir o consumo de energia.

As relações entre tarefas previstas são a precedência e a exclusão mútua, que foram modeladas como pode ser visto na Figura 3.6 para duas tarefas. A parte (a) representa a relação de precedência e a parte (b) a de exclusão mútua. No caso de exclusão mútua o *token* no lugar P_{exclij} garante a exclusão mútua entre as tarefas.

Figura 3.6: Modelos para tratamento de relações de precedência (a) e exclusão mútua (b)



(Extraída de (TAVARES; MACIEL; SILVA, 2008))

A abordagem foi aplicada em um software que recebe como entrada parâmetros de tarefas, e permite a avaliação de precedência e exclusão por indicação do usuário em lista específica. Embora faça a modelagem para eventos semelhantes aos tratados neste trabalho, essa ferramenta não permite a modelagem visual do comportamento das tarefas, recebendo-as como entradas estáticas de texto.

3.4 Árvore de tarefas concorrentes

Para tornar a modelagem de tarefas e relações uma atividade mais intuitiva é preciso torná-la visual, com o usuário passando graficamente as relações que o simulador deve interpretar. Para isso é necessário uma interface gráfica capaz de compreender modelos construídos a partir de ícones representando cada componente e identificadores para as relações. Esta abordagem permite a construção de modelos que sejam claros para a especificação do usuário, mas principalmente que sejam objetivos para a compreensão do simulador, evitando principalmente ambiguidades.

Esse tipo de abordagem não foi encontrado na literatura que trata sobre simuladores de tempo-real ou de sistemas concorrentes. Uma proposta semelhante, embora aplicada em tarefas de projeto de sistemas interativos, é encontrada no trabalho de Mori, Paternò e Santoro (MORI; PATERNÒ; SANTORO, 2002). Nele se usa árvore de tarefas concorrentes (PATERNÒ; MANCINI; MENICONI, 1997) para a criação de um simulador.

O modelo em árvore busca especificar a ocorrência das tarefas de interação humano-computador seguindo um modelo gráfico, capaz de representar os tipos de tarefas e as suas relações. A decomposição lógica segue o formato de uma árvore para evitar interpretações ambíguas. O sistema proposto recebeu o nome de CTTE e foi moldado como um editor para especificação das interações humano-computador, permitindo representar no modelo atividades e características que serão interpretadas posteriormente. O trabalho destaca a crescente demanda por modelagem de tarefas e cita algumas aplicações de modelos durante o processo de desenvolvimento de software, como a análise de requisitos, o projeto de aplicações interativas e a avaliação de usabilidade.

O modelo de árvore de tarefas concorrentes (CTT) é baseado na linguagem LOTOS (BOLOGNESI; BRINKSMA, 1987), que é um método de descrição formal que permite descrição dirigida a eventos e modificações de estado. As principais características do modelo CTT são:

- Foco na atividade, essencial para a etapa de projeto de software, o que permite a concentração nas atividades do usuário;
- Estrutura hierárquica, o que torna a construção do modelo mais intuitiva;
- Sintaxe gráfica, facilitando a manipulação dos elementos;

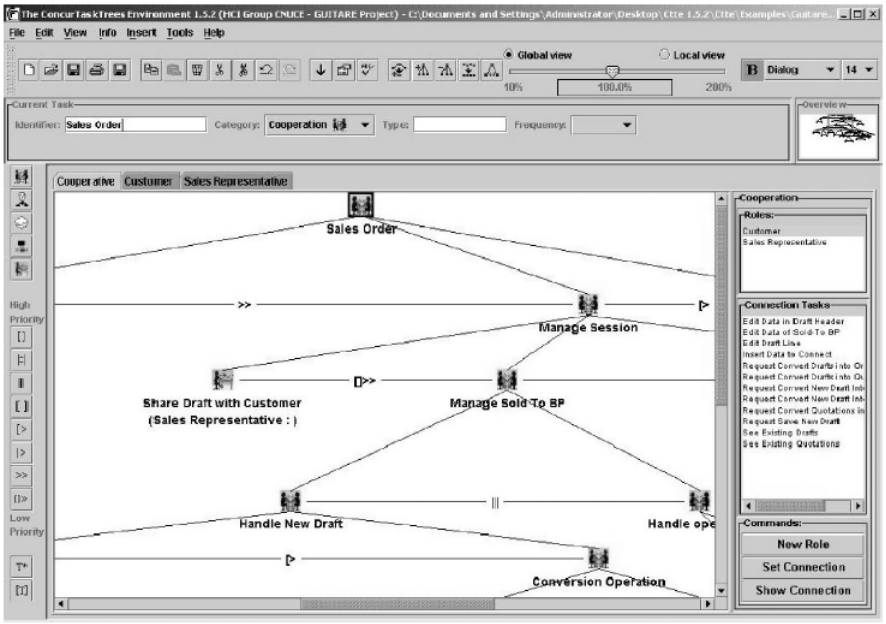
- Notação concorrente, com relacionamentos temporais para interação entre elementos;
- Alocação de tarefas, divididas em tipos específicos para o escopo da técnica;
- Objetos, que fazem parte do domínio das tarefas.

O modelo é composto por quatro tipos de tarefas: usuário, aplicação, interação e abstrata. Todas representam relações entre usuário e interface do sistema, respeitando o domínio do modelo. Além das tarefas, os relacionamentos entre tarefas foram criados para simbolizar a interação entre os elementos do software, como por exemplo precedência.

A ferramenta CTTE procura facilitar a manipulação desta técnica de modelagem, permitindo construir modelos diretamente na interface e simular as interações. Cada tipo de tarefa recebe uma especificação feita pelo projetista do sistema (usuário). A partir daí é possível realizar uma análise sobre o desenho e a especificação, identificando erros no modelo.

Uma visualização da interface disponibilizada pelo editor aparece na Figura 3.7. Na área central desta figura é possível observar o modelo gráfico construído, incluindo desde a disposição de tarefas até o estabelecimento de relações. As regras e especificações de cada elemento são feitas pelo usuário de forma descritiva como, por exemplo, a declaração de objetos ligados a cada tarefa.

Figura 3.7: Interface principal do CTTE



(Extraído de (MORI; PATERNÒ; SANTORO, 2002))

Apesar das definições dos operadores temporais estarem ligadas à área de interação humano-computador, elas abordam relações fundamentais de concorrência, como pre-

cedência por exemplo. Este tipo de relacionamento se aplica a outros tipos de tarefas concorrentes, e servem como base para compreender o funcionamento de uma interação, o que também será tratado na abordagem apresentada nos próximos capítulos.

A justificativa para o desenvolvimento do CTTE se dá pela perspectiva de facilidade de uso. É destacado que engenheiros de software, adaptados à construção de modelos formais, consideraram o modelo fácil de aprender e utilizar, apesar de nas avaliações feitas usuários sem experiência apresentarem mais dificuldades. Como o escopo da aplicação envolver um segmento particular entre as áreas de interface humano-computador, o processo de modelagem precisa do conhecimento de conceitos dessa área.

3.5 Considerações finais

Este capítulo abordou o cenário de modelagem, tratando também da modelagem específica de tarefas concorrentes. Foi possível perceber a demanda dentro da área de aplicação do projeto por uma técnica de modelagem por interface gráfica, que é o objetivo deste trabalho.

Embora o objetivo do CTTE ser diverso do que se propôs atingir neste trabalho, ele serviu como base para a especificação da estrutura hierárquica e do modo como são tratadas as relações e inserções de dados de tarefas dependentes. As características da metodologia proposta serão tratadas nos próximos capítulos e sua validação ocorreu com sua implementação no simulador RTsim, apresentado na seção 2.5.2.

Capítulo 4

Modelagem Visual de Interações Entre Tarefas

Nos capítulos anteriores se evidenciou a importância em se ter métodos corretos e práticos para modelar tarefas e suas interdependências dentro do processo simulação de sistemas computacionais. Daquela análise se estabeleceu o objetivo principal deste trabalho, que é o de propor uma metodologia visual para modelar dependências entre tarefas. Esta proposta é descrita ao longo deste capítulo.

4.1 Modelo visual para estabelecimento de relações entre tarefas

Os estudos descritos anteriormente levaram à identificação da abordagem usada no CTTE como sendo a mais indicada para a especificação de relações entre tarefas. Com isso, a metodologia seguida aqui envolveu, inicialmente, a proposição de um modelo para especificar relações de dependência e sincronismo entre tarefas de forma visual e simplificada. Em seguida foram especificadas as técnicas de interpretação do modelo gráfico para uma representação simulável.

A proposição de um modelo para especificar relações entre tarefas buscou determinar características genéricas de tarefas concorrentes. Dentro desse escopo se deu também ênfase para características específicas de tarefas de tempo-real, já que essa é a área de aplicação do RTsim.

Assim como o CTTE, a proposta estabelece um modelo baseado em árvores hierarquizadas, principalmente por esta estrutura permitir a eliminação de ambiguidades na interpretação de relacionamentos, como será verificado nas próximas seções. Isso torna o manuseio dos modelos de tarefas mais simples e intuitivo. Para o objetivo deste trabalho foram acrescentados componentes e relações característicos de sistemas concorrentes e de tempo-real.

4.1.1 Elementos fundamentais de modelagem

Para modelar sistemas concorrentes é necessário usar dois componentes básicos, um representando recursos do sistema e outro representando tarefas executadas nesses recursos. Naturalmente restringir modelos a esses dois elementos diminuiria bastante sua qualidade. Assim, recursos serão representados por elementos que caracterizem sua funcionalidade e tarefas serão representadas por elementos que definam características básicas de sua operação. Esses elementos são descritos a seguir.

- **Processador**

  O processador é um componente fundamental para a correta especificação deste modelo, representando os elementos de processamento no sistema (de qualquer tipo), podendo ou não ter tarefas especificamente associadas a ele. Sua representação gráfica é feita pelo ícone mostrado na Figura 4.1.

  Figura 4.1: Representação do Processador

  (Gerada pelo autor)

  Na organização do modelo hierárquico os elementos de processamento são colocados na posição de raiz da árvore. Isso permite uma melhor organização gráfica caso haja elementos (tarefas) que serão atribuídos a eles. Esta atribuição permite que os métodos de interpretação construam o conjunto de tarefas inicialmente atribuído a cada processador, para que possa ocorrer a simulação do escalonamento.

  Neste modelo as tarefas graficamente associadas a cada processador formarão o conjunto inicial de tarefas para simulação no processador. Não foram criadas representações ou sinalizações de troca de tarefas entre processador pelo fato desta ser uma decisão atrelada ao algoritmo de escalonamento adotado, não sendo necessária a especificação de forma gráfica.

- **Tarefas Periódicas**

  Tarefas formam o centro do modelo de carga de trabalho para a simulação de um sistema concorrente. Para representar relações de dependência entre tarefas é necessário ter um elemento que as represente. Isso é necessário pois são as tarefas que representam as atividades que serão executadas e as características necessárias para a simulação. É por meio das tarefas e seus relacionamentos que é possível desenhar, e compreender visualmente, o comportamento do sistema e do conjunto simulado.

A representação gráfica de tarefas periódicas é feita por um retângulo com contorno de linha cheia, como visto na Figura 4.2.

Figura 4.2: Representação de Tarefas Periódicas

(Gerada pelo autor)

Tarefas periódicas aqui representadas são comuns em aplicações de tempo-real. Este tipo de tarefa possui como característica principal a regularidade em seus instantes de ocorrência (período de execução). Por ter um número infinito de execuções e um intervalo regular de ocorrência, o que gera uma previsibilidade na performance, são normalmente consideradas tarefas críticas (FARINES; FRAGA; OLIVEIRA, 2000), estando constantemente presentes no conjunto de tarefas para simulação.

Do ponto de vista do modelo de tarefa, além da determinação de relações de dependência, é importante que as tarefas periódicas recebam seus parâmetros característicos de simulação. Por este motivo, a cada componente Tarefa Periódica são definidos seis parâmetros (Nome, Carga, Período, *Deadline*, Tempo de Chegada e Prioridade) e uma entrada para configuração de recursos (caso haja recurso atribuído à tarefa em questão). Esses parâmetros, com exceção do Nome, são valores de entrada para simulação do sistema.

O atributo Nome de cada componente funciona como um identificador global de elemento na interface. Assim, para evitar que se configure dois elementos com mesmo nome, este parâmetro é gerado automaticamente pela ferramenta, não sendo configurável pelo usuário. A forma como ocorre esta configuração pela interface será tratada adiante.

- **Tarefas Aperiódicas**

Tarefas aperiódicas são uma característica importante em sistemas de Tempo-Real, representando atividades ou eventos não previsíveis. Do ponto de vista de modelagem devem ser diferenciadas das tarefas periódicas e por isso são representadas por um retângulo com contorno tracejado, como visto na Figura 4.3.

Figura 4.3: Representação de Tarefas Aperiódicas

(Gerada pelo autor)

Os parâmetros de execução de aperiódicas também são diferentes, compondo cinco valores e uma interface para configuração de recurso. Os cinco valores são: Nome, Carga, *Deadline*, Intervalo Mínimo entre Chegadas e Prioridade. Deve-se observar que o intervalo mínimo entre chegadas é um parâmetro qualitativo, que pode indicar se o simulador tem comportamento semelhante ao do sistema real, isto é, as frequências de ocorrências de aperiódicas é condizente ao que se espera delas no mundo.

- **Grupo de Tarefas**

Quando há a necessidade de estabelecer relações entre várias tarefas o modelo pode se tornar confuso, ou mesmo ambíguo. Para manter a clareza do modelo se criou o componente *Grupo de Tarefas* que representa um agrupamento de tarefas na interface, e recebe o relacionamento a elas aplicado. Como de certo modo o conceito é semelhante ao de uma caixa preta, o ícone representativo de grupo pode ser visto na Figura 4.4.

Figura 4.4: Representação de Grupos de Tarefas

(Gerada pelo autor)

O objetivo deste componente de modelagem não é representar um elemento de sistemas concorrentes, mas sim tratar um problema existente na construção das árvores hierárquicas, que é o de ambiguidade. O problema da ambiguidade será explicado na página 60, após a descrição das relações de dependência e seu funcionamento, para que fique mais clara a função deste elemento.

- **Recurso**

O último componente definido para a modelagem de sistemas concorrentes envolve os recursos do sistema. Tarefas utilizam diversos recursos durante a execução, como por exemplo barramentos de comunicação, periféricos, etc. Assim foi necessário criar um elemento gráfico para representar estes recursos externos e diferenciá-los do processador. Deve-se salientar que apesar do processador também ser um tipo de recurso, como é de uso obrigatório adotado pelas tarefas preferiu-se sua definição separada.

Para representação dos recursos foram adotadas formas circulares como a da Figura 4.5.

Figura 4.5: Representação de Recursos

(Gerada pelo autor)

Para maior clareza no processo de modelagem foi definido que a configuração de recursos é atrelada às interfaces de cada tarefa. Assim, os recursos não aparecem no mesmo plano de tarefas e processador, sendo visíveis apenas na configuração de cada tarefa. Além disso, o tratamento de recursos em interfaces separadas permite considerar um comportamento independente para cada tarefa. Permite ainda o estabelecimento de relações de precedência na captação de recursos. Com isto, foi possível visualizar uma nova oportunidade no modelo, criando um estabelecimento de relação entre os recursos presentes no sistema.

Justificando a simbologia adotada

Para a escolha dos símbolos dos componentes foram estudadas ferramentas de modelagem de processos existentes e quais características seriam vantajosas ao projeto. Primeiramente foram adotadas formas geométricas para que os desenhos do modelo fossem similares a desenhos feitos a mão, o que gera uma visão mais amigável da interface. Além disto foram adotadas formas existentes em modelos consagrados, como o Business Process Modeling Notation (BPMN) (WHITE; MIERS, 2008), que apesar de ter função diversa, também trata da representação gráfica de processos e seus relacionamentos.

Assim como no BPMN 2.0 e em outros modelos adotou-se estruturas de retângulo para representação de tarefas. Uma característica que deve ser observada no modelo é a diferenciação entre os tipos de tarefas apenas pelo preenchimento do contorno. Esta padronização foi escolhida pois evita o uso de formatos gráficos distintos para representar objetos semelhantes, tornando o modelo mais simples.

Os outros formatos (hexágono para processadores, caixa preta para grupos e oval para recursos) foram escolhidos principalmente com o objetivo de criar uma diferença visual considerável entre os componentes. Assim, fica fácil identificar cada componente representado durante a leitura do modelo.

4.1.2 Estabelecimento de relações de dependência

Como o objetivo fundamental é a possibilidade de especificar relações entre tarefas de modo gráfico, a representação dessas relações é essencial para o funcionamento do modelo. Como cada componente é representado por um ícone, o estabelecimento de uma relação se dá por meio de uma linha ligando o componente de origem ao componente destino,

associada a um símbolo identificador. Esse símbolo identificador, caracterizado pelos símbolos listados na Tabela 4.1, é quem define qual a relação modelada. As relações que são consideradas para a simulação de tarefas são:

- **Precedência**

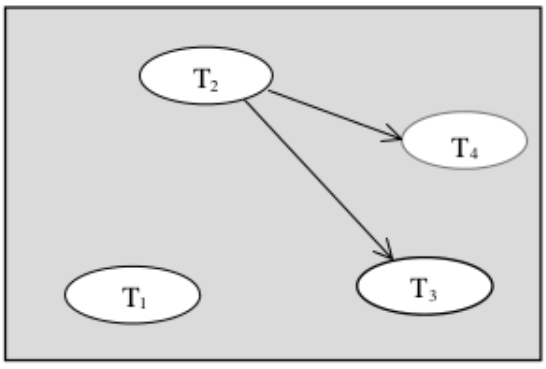
  Precedência é uma relação básica quando se trata de tarefas concorrentes com relações de dependência entre si. O conceito deste relacionamento é a ideia de uma tarefa executando antes de outra, e isso está relacionado à existência de ordem na ocorrência das atividades de um sistema. Formalmente tem-se que a tarefa T_i precede T_j ($T_i \rightarrow T_j$) se a execução de T_j é possível apenas após o término de T_i, como visto na seção 2.4. Essa ordenação parcial pode representar dependência de informação produzida ou situações de sincronismo.

  Durante um escalonamento tarefas com relação de precedência tem que, obrigatoriamente, seguir uma ordem entre a ocorrência das suas atividades. Essa ordem pode ser estabelecida por um *offset*, que é o cálculo prévio do pior tempo de liberação da tarefa, sendo que este parâmetro só permite a execução da tarefa após o tempo determinado por ele. Pode também ser utilizado o *jitter*, que é um parâmetro associado a troca de mensagens e a liberação é autorizada após a sinalização que a tarefa precedente terminou.

  Em termos de análise de escalonabilidade ambos são idênticos, pois consideram sempre o pior caso para a liberação. A diferença ocorre em tempo de execução, uma vez que *offset* é uma atribuição fixa que leva sempre a execução do pior caso e *jitters* são dinâmicos, tornando o pior caso uma ocorrência eventual.

  Este tipo de relação é normalmente tratada pela representação em grafos orientados, como visto na Figura 4.6. Nela é ilustrado um conjunto de tarefas e suas relações representadas por grafos, em que os índices das tarefas representam sua prioridade, ou seja, T_1 tem prioridade igual a 1.

Figura 4.6: Grafo de precedência (FARINES; FRAGA; OLIVEIRA, 2000)



tarefas	J_i	C_i	P_i	D_i
T_1	1	10	40	40
T_2	3	10	80	25
T_3	-	5	80	40
T_4	-	10	80	80

No exemplo podem ser extraídas as relações presentes no conjunto e o entendimento da influência do *jitter*. A tarefa T_1 é a mais prioritária sendo a primeira a executar e não sofrendo interferência de nenhuma outra, seu tempo de resposta é obtido pela soma da carga com o seu *jitter*. Já T_2 sofre a influência da execução de T_1 devido à prioridade estabelecida e só poderá executar após o final de T_1.

A tarefa T_2 não é considerada como interferência em T_3 e T_4, pois estas só podem executar depois de T_2 finalizada (relação de precedência estabelecida pelos arcos do grafo). A tarefa T_3 tem seu tempo calculado considerando a execução de T_1, e seu valor de *jitter* é o tempo de resposta calculado para T_2. Por fim a tarefa T_4 sofre influência de T_1 e também de T_3, tendo como *jitter* o valor de T_2.

Abordando o cálculo da precedência

As equações 4.1 e 4.2 são utilizadas para o teste de escalonabilidade do conjunto. A primeira trata da janela de tempo para ativação de uma tarefa, considerando o caso de interferência entre as atividades e a segunda do tempo de resposta com consideração do *jitter*.

$$W_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i + J_j}{P_j} \right\rceil C_j \quad \text{, e} \tag{4.1}$$

$$R_i = W_i + J_i \tag{4.2}$$

As tarefas do conjunto mostrado na Figura 4.6 podem ser testadas por estas equações da seguinte maneira:

T_1: Não sofre interferência, logo: $R_1 = 10 + 1 \rightarrow R_1 = 11$

T_2: Sofre interferência de T_1, logo:
$W_2^0 = C_2 = 10$
$W_2^1 = 10 + \sum_{j \in hp(i)} \left\lceil \frac{10+1}{40} \right\rceil 10 = 20$
$W_2^2 = 10 + \sum_{j \in hp(i)} \left\lceil \frac{20+1}{40} \right\rceil 10 = 20$
$R_2 = 20 + 3 \rightarrow R_2 = 23$

T_3: Sofre interferência por T_1 e possui o *jitter* de T_2, logo:
$W_3^0 = C_3 = 5$
$W_3^1 = 5 + \sum_{j \in hp(i)} \left\lceil \frac{5+1}{40} \right\rceil 10 = 15$
$W_3^2 = 5 + \sum_{j \in hp(i)} \left\lceil \frac{15+1}{40} \right\rceil 10 = 15$
$R_3 = 15 + 23 \rightarrow R_3 = 38$

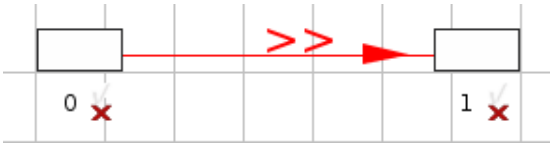
T_4: Sofre interferência por T_1 e também de T_3 devido a sua prioridade, recebendo também o *jitter* de T_2 pela relação de precedência, logo:

$$W_4^0 = C_4 = 10$$
$$W_4^1 = 10 + \sum_{j \in hp(i)} \left\lceil \frac{10+1}{40} \right\rceil 10 + \sum_{j \in hp(i)} \left\lceil \frac{10+23}{80} \right\rceil 5 = 25$$
$$W_4^2 = 10 + \sum_{j \in hp(i)} \left\lceil \frac{25+1}{40} \right\rceil 10 + \sum_{j \in hp(i)} \left\lceil \frac{25+23}{80} \right\rceil 5 = 25$$
$$R_4 = 25 + 23 \rightarrow R_4 = 48$$

Analisando os valores de *deadline* relativos de cada tarefa e o tempo de resposta obtido pelo teste pode-se concluir que todas são escalonáveis.

Para a representação no modelo adotou-se o identificador “>>” para esta relação. Ele é atribuído a uma linha direcional, em que a origem é a tarefa precedente e o destino a tarefa precedida, como na Figura 4.7.

Figura 4.7: Representação de Precedência no Modelo



(Gerada pelo autor)

A adoção deste símbolo a uma linha direcional visa dar mais flexibilidade para manipular os componentes na interface, pois a seta segue o componente ao qual está ligada independente da posição em que este seja movimentado, mantendo o direcionamento correto para a interpretação e leitura do modelo.

- **Precedência com passagem de informação**

  Esta relação tem um funcionamento semelhante ao item anterior, valendo-se do mesmo comportamento base para a execução. Precedência com passagem de informação ocorre quando uma tarefa, além de aguardar a execução completa de sua precedente, necessita uma informação/mensagem que será recebida para que sua execução possa começar normalmente.

  Esta relação é interessante para tarefas que trocam informações entre si ou necessitam dados resultantes de outras operações para realizar suas atividades corretamente. Isso ocorre, por exemplo, com tarefas de ajuste de temperatura que recebem a medição da temperatura feita por outra tarefa do sistema, e com isso são capazes de realizar o ajuste necessário.

  O processo de comunicação entre tarefas do sistema e sua sincronização foi tratado com detalhes no Capítulo 2. Para o modelo gráfico foi adotado o símbolo “[ ] >>”, que foi pensado para remeter ao símbolo da precedência, trazendo também a ideia de carregar informação. O conteúdo da mensagem a ser passada é irrelevante, sendo responsabilidade do modelo apenas ilustrar e sinalizar que haverá este tipo de relação envolvendo um conjunto de tarefas.

Na Figura 4.8 é ilustrado o relacionamento com este identificador entre duas tarefas. Assim como a relação de precedência, esta também é atribuída a uma seta direcional, pois é importante que o modelo identifique quem produzirá e quem receberá a informação, além da relação de precedência entre elas.

Figura 4.8: Representação de Precedência com passagem de informação no Modelo

(Gerada pelo autor)

- **Prioridade**

  Prioridade é um parâmetro básico quando se trata de tarefas de tempo-real, indicando de modo explícito a preferência de execução entre elas. Embora do ponto de vista de modelagem a prioridade já seja um dos atributos de tarefas, a adoção de uma relação de prioridade entre tarefas permite associar duas tarefas cuja única forma de interação seja pela prioridade que uma tenha em relação à outra.

  Tarefas no conjunto que seguem apenas relações de prioridade entre si são tarefas independentes, porque dependem apenas da escolha de ordem de execução feita pelos parâmetros do escalonador. A escolha de abordar esta relação no modelo mesmo ela não sendo um relacionamento de dependência é devido a necessidade de ilustrar todo o conjunto de tarefas na interface, incluindo as que são independentes, pois isto permite uma visão gráfica completa de todas as tarefas e seu comportamento.

  O símbolo adotado para representar a prioridade no modelo foi $[P]$, como visto na Figura 4.9. Como a relação de prioridade existe apenas para demarcar ausência de dependência, o símbolo é associado a uma linha simples.

Figura 4.9: Representação de Prioridade no Modelo

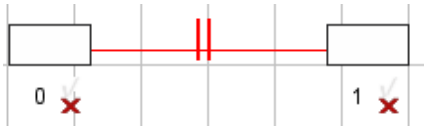
(Gerada pelo autor)

- **Paralelismo**

  Uma relação comum entre tarefas concorrentes é a possibilidade de executar algumas atividades em paralelo. Assim, é importante definir a relação de paralelismo entre tarefas para completar as opções de modelagem. A indicação do paralelismo permite que se modele tarefas que poderão ser executadas simultaneamente. Na Figura 4.10

é ilustrado como esta relação é representada graficamente no modelo, sendo marcada pelo símbolo "| |".

Figura 4.10: Representação de Paralelismo no Modelo



(Gerada pelo autor)

Esta relação é ainda mais importante para sistemas multiprocessados, permitindo sinalizar tarefas que podem executar juntas desde que com recursos de processamento disponíveis. Além disso, este tipo de relação é importante para algoritmos de tempo-real que permitam que tarefas troquem de processador (migração) para evitar perda de *deadline*, pois a ideia de paralelismo já permite estabelecer esse modelo de execução.

- **Independência de Ordem**

  Esta relação é importante para sistemas concorrentes de modo geral. A independência de ordem é aplicada para tarefas que compartilhem algum recurso de acesso exclusivo, e desta forma acabam impedindo a execução das outras durante a sua ativação. Quando há este relacionamento entre tarefas, em que não exista estabelecimento de prioridade, o sistema interpreta que não há ordem de ativação entre elas. Isso implica que as tarefas podem ser executadas em qualquer ordem, porém a ativação de uma delas desabilita a outra quanto ao uso do recurso, impedindo sua utilização até a liberação do mesmo.

  Essa relação pode ser comparada ao operador lógico "OU EXCLUSIVO", em que não podemos ter duas relações Verdadeiras/Válidas ao mesmo tempo. Na Figura 4.11 é abordado o relacionamento no modelo, sendo "|=|" o símbolo associado a ele.

  Este tipo de relacionamento é adequado para modelagem de tarefas concorrentes que compartilham recursos e que devem ser tratadas de forma indistinta pelo sistema, o que impede a priorização explícita de uma delas. O problema do "Jantar dos Filósofos" (DIJKSTRA, 2002) é um bom exemplo desta situação, pois neste caso não há prioridade entre os filósofos, mas há compartilhamento de recursos, os *hashis*, com seus vizinhos. Nele a relação de independência de ordem entre os vizinhos é evidente, pois não importa para o problema qual filósofos usará os recursos primeiro, impedindo os vizinhos de o usarem simultaneamente.

De forma resumida a Tabela 4.1 apresenta os símbolos adotados para as relações do modelo e seus respectivos significados.

Figura 4.11: Representação de Independência de Ordem no Modelo

(Gerada pelo autor)

Tabela 4.1: Símbolos para ilustrar relações entre tarefas

Relação	Identificador
Precedência	>>
Precedência com passagem de informação	[ ] >>
Prioridade	$[P]$
Paralelismo	| |
Independência de ordem	|=|

Quanto à adoção dos símbolos para relações é possível observar que são caracteres simples, presentes em teclados de computadores e em qualquer língua ocidental. A escolha por caracteres visou manter a simplicidade na descrição e desenho visual, facilitando também a representação em artigos e em desenhos de diversos tamanhos. Além disso, alguns desses símbolos também estão presentes na simbologia do CTTE, embora nem sempre com a mesma funcionalidade e objetivos.

4.1.3 O problema da ambiguidade e a função do grupo de tarefas

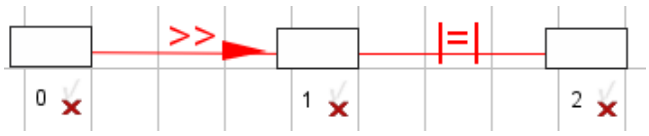
O uso da estrutura em árvore traz clareza para a modelagem de dependências entre tarefas. Entretanto, a interpretação de modelos construídos apenas com elementos básicos (processadores, recursos e tarefas periódicas e aperiódicas) pode criar ambiguidades.

O problema da ambiguidade na leitura do modelo surge da forma em que é construída a relação entre as tarefas de forma visual. Essa construção ocasiona duas formas diferentes de interpretar o que foi desenhado, levando ao erro do sistema na leitura da interface.

Na Figura 4.12 é ilustrado um modelo em que aparece ambiguidade. A primeira leitura que pode ser feita, mais imediata, seria que a Tarefa 0 precede a Tarefa 1, e que a Tarefa 1 tem uma independência de ordem em relação a Tarefa 2, que não tem relação com a Tarefa 1. Porém, observando o modelo, também é possível identificar que a Tarefa 1 é independente da Tarefa 2 e as duas são precedidas pela Tarefa 0. Deve-se observar que esta segunda leitura seria mais natural se existisse também uma relação de precedência direta entre as tarefas 0 e 2.

Para evitar a necessidade de inclusão de vários arcos para definir relações desse tipo é que se criou o elemento grupo de tarefas. O uso desse elemento elimina ambiguidades sem causar poluição visual, como visto na Figura 4.13. Nela é possível realizar de imediato a

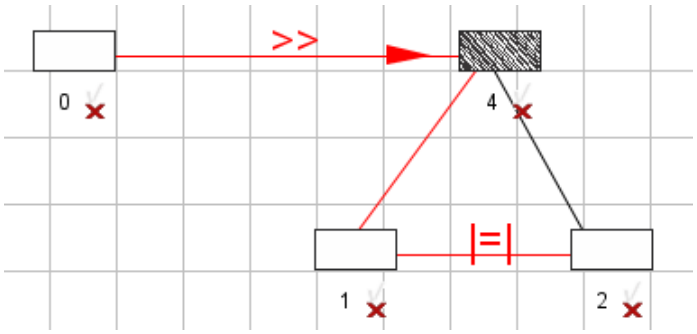
Figura 4.12: Representação de relacionamento com ambiguidade no Modelo.



(Gerada pelo autor)

leitura dos relacionamentos, identificando que as tarefas 1 e 2 são precedidas pela 0.

Figura 4.13: Representação de relacionamentos usando grupo de tarefas.



(Gerada pelo autor)

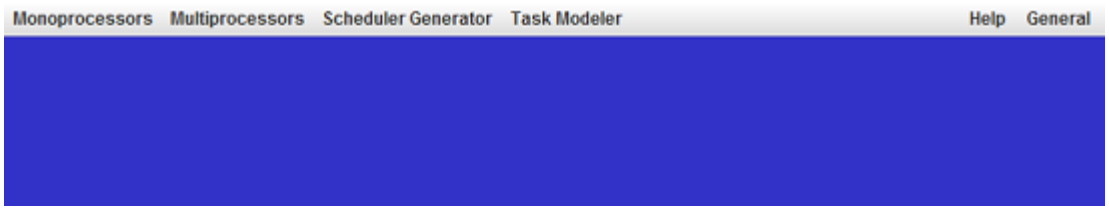
4.2 Interface gráfica para composição do modelo

Os componentes e relações para modelagem apresentados na seção anterior permitem projetar uma interface gráfica para a composição de modelos de interações entre tarefas. A interface de desenho foi adicionada como um módulo separado dos outros módulos do RTsim, de forma que fosse de fácil manipular seus códigos para manutenção. Isso permitiu também manter as entradas tradicionais já implementadas no simulador como opção ao usuário. Este tipo de opção permite que o construtor do modelo observe as diferenças entre tratamento de uma carga de trabalho representada por tarefas independentes e representada com relações entre as tarefas.

Na Figura 4.14 é exibida a interface inicial do simulador já com o menu de modelagem adicionado ("Task Modeler"). Esta opção permite o acesso à interface para desenho e geração do modelo. Modelos que forem interpretados e gerados pelos métodos adicionados podem ser simulados pelos escalonadores presentes nas opções de Monoprocessados e Multiprocessados.

O processo de modelagem das tarefas começa apenas após a escolha de uma pasta para salvamento dos modelos que serão desenvolvidos. Isto é importante para fazer a ligação entre a interface principal, composta de tarefas, e a interface secundária de recursos associados a cada elemento Tarefa.

Figura 4.14: Interface do RTsim com adição do menu de modelagem (*Task Modeler*).



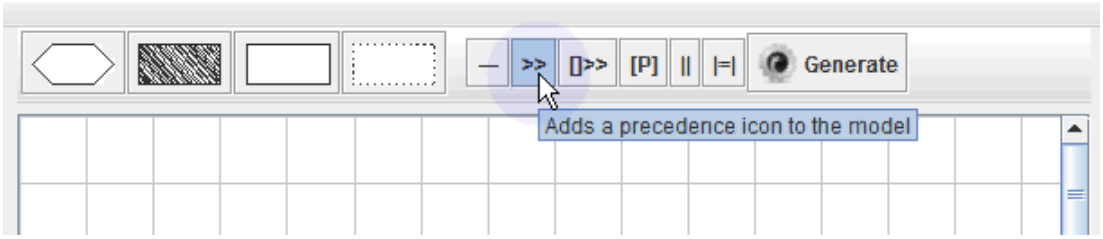
(Gerada pelo autor)

Para ilustrar melhor o funcionamento da interface principal como um todo, são descritas a seguir as cinco partes que compõem a tela de desenho e quais as suas funções.

1. Painel de Componentes e Relações:
   O painel principal, e mais utilizado para o desenho, é aquele que contem os componentes e as relações que serão utilizadas para compor o modelo. Eles s ao colocados de maneira próxima para que seja fácil ter uma visão de todas as opções possíveis, como visto na Figura 4.15, os componentes (Tarefas, Processador, e outros) aparecem mais a esquerda e as relações mais a direita.

   O botão "Generate", responsável por dar início 'a interpretação do modelo e geração do arquivo simulável, também aparece no lado direito. Na Figura 4.15 é possível observar uma das mensagens disparadas pelos botões, o que facilita a associação do elemento ao seu significado durante a criação do modelo, principalmente nos momentos de primeiro contato com a ferramenta.

Figura 4.15: Painel de ações presente na tela de desenho

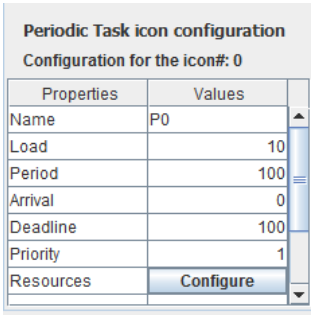


(Gerada pelo autor)

2. Painel de Configurações Associado a cada Componente:
   Neste painel são configurados cada componente. Por exemplo, ao inserir uma tarefa periódica na interface é necessário configurar parâmetros como período, carga, "deadline" e prioridade, entre outros, como visto na Figura 4.16. Foi criado um tipo de painel para cada componente da interface, visando atender as características de cada um, porém mantendo sempre o padrão de interface para simplificar a utilização.

   Esse painel também é exibido na parte lateral da área de desenho, como um item não configurável, ou seja, ele apenas exibe as características que já foram inseridas.

Figura 4.16: Painel de configurações presente na tela de desenho



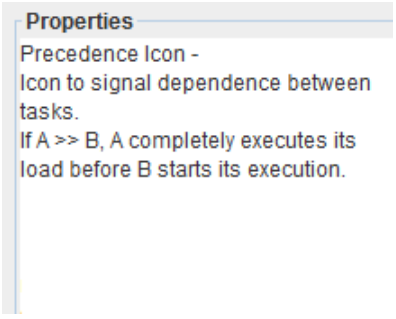
(Gerada pelo autor)

A edição é feita quando o painel é exibido em uma janela de confirmação externa, associada a cada elemento. Nessa janela ele pode receber as configurações e salvá-las após a confirmação.

3. Painel de Propriedades:
   O painel de propriedades permite ao usuário entender as informações que está introduzindo no modelo de tarefas. Nele são exibidas informações sobre o que fazer para configurar o componente ou qual é o significado atribuído à relação que está sendo utilizada. Este painel foi adicionado à parte lateral da tela principal, sendo interessante para apoiar os usuários que estão começando a trabalhar com o ambiente de modelagem.

   Na Figura 4.17 é ilustrada a exibição das propriedades associadas ao item de precedência. Pode ser observado que o painel é composto pelo nome da relação e por uma breve descrição de sua característica.

   Figura 4.17: Painel de propriedades presente na tela de desenho



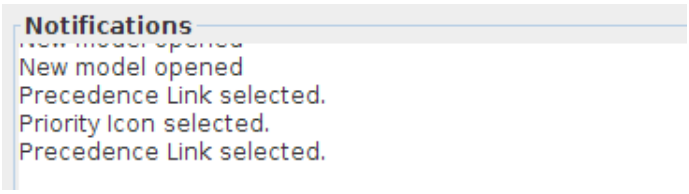
(Gerada pelo autor)

4. Painel de Notificações:
   Este painel tem um funcionamento simples, apenas listando quais ações foram tomadas na área de desenho, como por exemplo qual item foi selecionado ou adicionado na interface. Sua função é servir como um registro das ações executadas pelo usuário,

complementando os painéis criados para tornar a construção do modelo assistida pela ferramenta.

O painel de notificações, com algumas informações de exemplo, é visto na Figura 4.18. É possível por meio deste painel saber qual foi a ordem de construção escolhida pelo usuário, o que ajuda a compreender a lógica do construtor do modelo.

Figura 4.18: Painel de notificação presente na tela de desenho



(Gerada pelo autor)

5. Área de Desenho:
   A área de desenho ocupa a posição central da interface principal, sendo construída para ser semelhante a um desenho manual do modelo. Para que a construção seja semelhante a um desenho manual, os componentes são inseridos selecionando o ícone e o local em que será adicionado na área. Já as relações são adicionadas apenas com a seleção do item origem seguido do item destino. Deve ser observado que os elementos incluídos no modelo podem ser movidos dentro da área de desenho, sempre que isso for conveniente.

Na Figura 4.19 é exibida a interface completa do módulo de modelagem. É possível observar os cinco componentes apresentados anteriormente dispostos na tela principal, de forma que fiquem com suas informações disponíveis facilmente para o usuário.

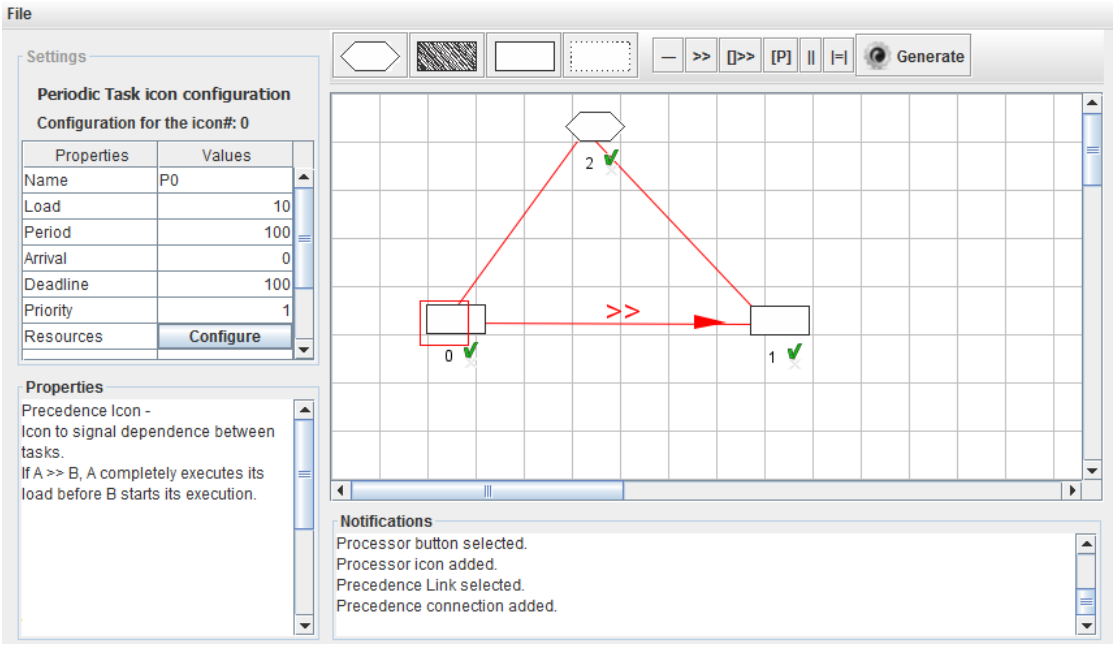
4.2.1 Interface para modelagem de recursos

Além das interfaces descritas é necessário uma interface para a configuração dos recursos que uma tarefa utiliza. Essa interface é acionada pelo botão *"Resources"*, encontrado no painel de configurações (visto na Figura 4.16).

Como dito anteriormente, para não causar uma poluição visual que prejudicaria a organização e interpretação do modelo, foi adotada a ideia de manter a configuração de recursos em interfaces separadas e individuais a cada tarefa, o que gerou também a oportunidade de trabalhar com relacionamentos entre os recursos.

Na Figura 4.20 se exibe a caixa de diálogo ativada, após o acionamento do botão de recurso, para interagir com o usuário sobre as configurações. Este painel é composto por três escolhas de ações, que são:

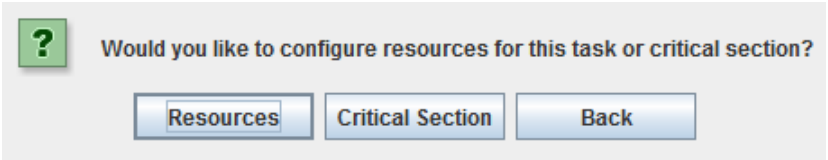
Figura 4.19: Interface Principal para modelagem



(Gerada pelo autor)

- **Resources**: disponibiliza uma nova interface exclusiva para modelagem dos recursos, que será vista a seguir com mais detalhes de funcionamento.
- **Critical Section**: esta opção funciona para aqueles modelos que precisam estabelecer regiões de exclusão mútua com semáforos, desde que em função do compartilhamento do processador.
- **Back**: Opção básica que sinaliza apenas que não haverá modelagem de recursos para esta tarefa, retornando para a janela de desenho original.

Figura 4.20: Caixa de diálogo acionada com o botão *Resources*

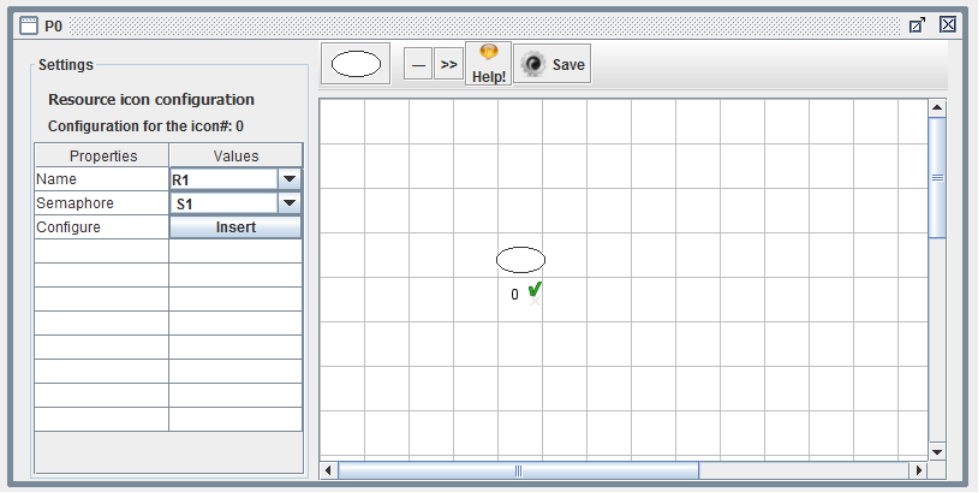


(Gerada pelo autor)

A interface para configuração dos recursos foi desenvolvida para ser uma versão simplificada da tela principal de modelagem. Isto foi feito porque ambas têm funções semelhantes, e torná-las parecidas gera uma melhor experiência de utilização para o usuário.

É possível observar na Figura 4.21 a interface desenvolvida para a modelagem de recursos. Esta não possui todos os painéis da tela principal e seus componentes também são mais simples, pois o foco é apenas a modelagem exclusiva dos recursos e semáforos.

Figura 4.21: Interface para modelagem de recursos



(Gerada pelo autor)

Assim como acontece na interface principal o painel de configurações tem sua visualização na lateral esquerda e uma caixa de diálogo para edição. Para recursos os parâmetros solicitados são o nome que será atribuído ao recurso e o o nome do semáforo que será utilizado. Além destes, o botão *Insert* leva a uma interface para configuração da seção crítica, como por exemplo, o tempo de entrada na região.

O botão *Save* presente na parte superior da tela é responsável por duas ações. A primeira é o salvamento do modelo gráfico em diretório e formato específico, para que possa ser reaberta posteriormente pelo usuário; e a segunda é a gravação das configurações inseridas no modelo interpretado.

A relação de precedência entre os recursos

A escolha por manter a modelagem de recursos em uma interface externa permitiu criar também a função de relacionamento entre os recursos da tarefa. Precedência entre recursos é bastante presente na literatura de sistemas concorrentes, sendo que para sua representação foram mantidos o mesmo nome e identificador do relacionamento de precedência entre tarefas.

A precedência entre os recursos, quando estabelecida, garante que a tarefa requisitará primeiro o recurso precedente, e após ter posse deste, fará a seleção pelo precedido. A posse dos recursos pode ser simultânea, desde que as requisições atendam a relação dada. Deve ser observado que isso difere da precedência entre tarefas, quando uma aguarda a completa execução de sua precedente para iniciar. Finalmente, caso seja necessário modelar um recurso que aguarde a completa execução de seu precedente, basta controlar os valores de tempo de entrada na região crítica inserido no painel de configuração.

Alguns problemas clássicos de concorrência, como o “Jantar dos Filósofos”, ilustram

casos em que é necessário manter uma ordem para a captação de recursos, mesmo que devam ser utilizados em conjunto. Isso mostra a motivação para introduzir este tratamento no trabalho e tornar a ferramenta mais completa para as necessidades de sistemas concorrentes.

4.3 Composição de relações entre tarefas

As seções anteriores trataram das definições atribuídas a cada elemento presente em um modelo e suas características. Com isso, é possível observar o comportamento esperado nas relações definidas na interface. Nesta seção será tratada a composição de vários elementos e como essa composição deve ser interpretada.

Modelo de árvore hierarquizada

A construção do modelo em um formato hierárquico permite que se estabeleça uma ordem lógica de construção. Assim se começa pela raiz da árvore, que é representada pelo processador neste modelo, e vai se adicionando as tarefas executadas. De um modo geral, o objetivo é ter tarefas ligadas ao processador (raiz) para que visualmente seja possível identificar de imediato as tarefas que compartilham o mesmo recurso de processamento.

O processo de interpretação ocorre de maneira recursiva, em que após identificar o processador, a leitura segue para os próximos níveis de hierarquia, encontrando todas as tarefas que estão ligadas ao processador. Se houver um caminho do elemento raiz até a tarefa na árvore, então esta será atribuída à lista de tarefas no processador.

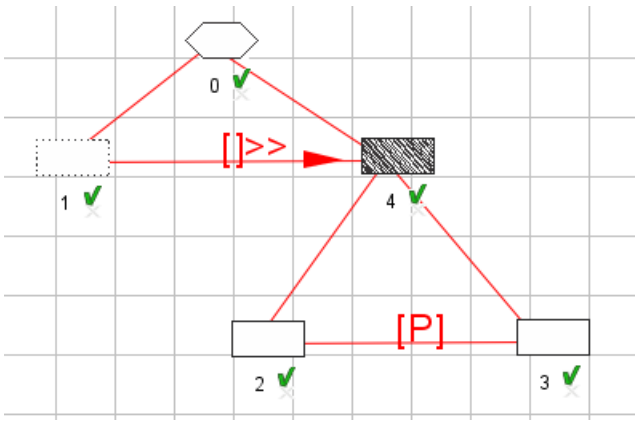
A Figura 4.22 permite uma visualização da hierarquia e da atuação do grupo de tarefas. Para a visualização do modelo é mais simples observar como a relação de precedência com passagem de informação atua sobre as demais tarefas com a presença do grupo. O grupo é a raiz da subárvore iniciada por ele, e segue o mesmo modelo de leitura adotado pleo processador. A hierarquia aplicada pelo modelo resolve o grupo distribuindo a relação entre as tarefas filhas, como pode ser compreendido graficamente. Esse processo de interpretação é representado (virtualmente) pelo modelo mostrado na Figura 4.23.

Precedências

As relações de precedência e precedência com passagem de informação são modeladas da mesma forma e tratadas com a mesma lógica. Sua diferenciação ocorre apenas no método responsável pela troca de mensagem. Em ambas se define um sentido para a relação, o que é fundamental para a identificação dos lados precedente e precedido.

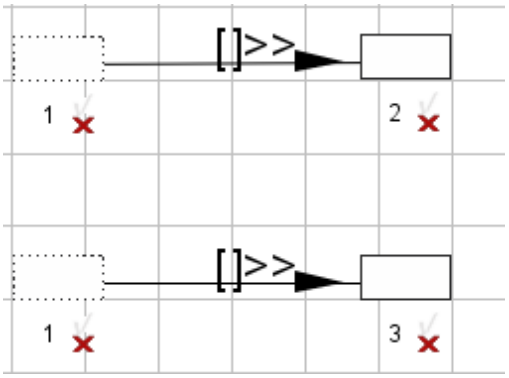
Na Figura 4.24(a) exibe-se um modelo com relacionamentos de precedência. Visualmente é possível identificar o fluxo de execução construído por meio dos grupos de tarefas, iniciando com a execução da Tarefa 1 e finalizando com a Tarefa 4. O fluxo de execução

Figura 4.22: Modelo com relação de precedência com passagem de informação



(Gerada pelo autor)

Figura 4.23: Interpretação da dependência com o uso de grupos de tarefas.



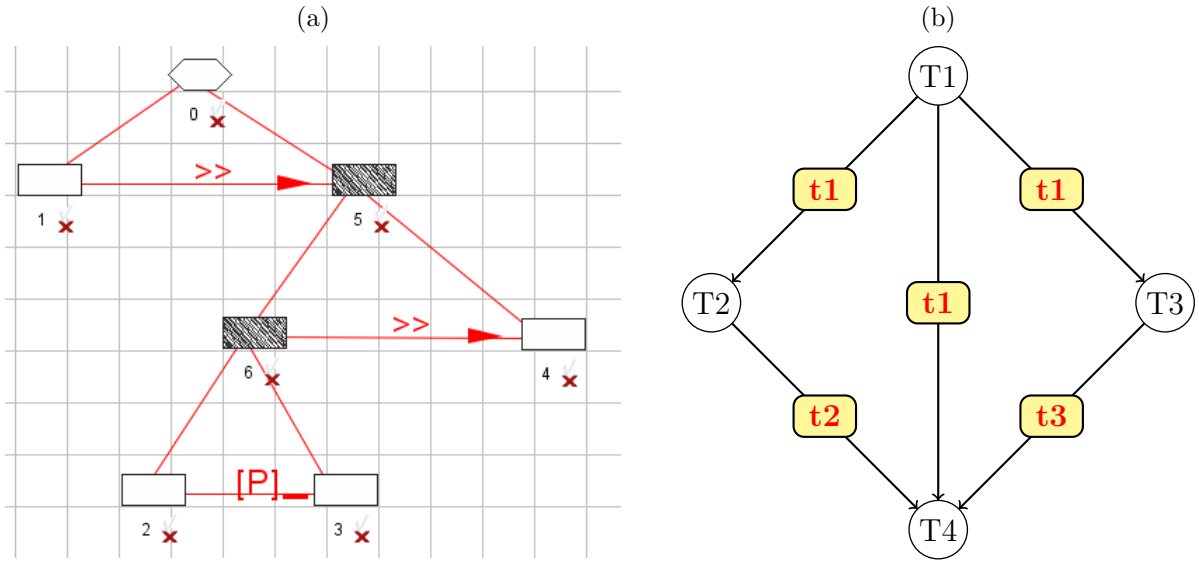
(Gerada pelo autor)

interpretado para modelo é ilustrado na Figura 4.24(b), que traz o grafo de precedência entre as tarefas.

Os vértices do grafo representam as tarefas marcadas pelos mesmos números identificadores no modelo. Suas arestas indicam a direção da ativação de cada elemento, destacando as dependências entre tarefas retiradas do modelo. As caixas em cada aresta representam o tempo de execução consumido antes da ativação da tarefa consequente. Iniciando pela tarefa T1, a única sem dependência inicial, e supondo que a prioridade de T2 é maior que T3, os tempos de ativação podem ser interpretados como:

- A ativação de T2 ocorre após transcorrer o tempo t1, que representa a execução de sua precedente T1. A finalização de T1 libera a execução de T2 e T3, mas pela relação de prioridade, a tarefa T3 deve aguardar a finalização de T2;
- Apesar de não compor o grafo de precedência, a relação de prioridade pode ser vista no modelo gráfico e é considerada na interpretação do modelo como parâmetro de configuração.

Figura 4.24: Modelo com interações de precedência entre as tarefas



(Gerada pelo autor)

- A tarefa T3 estará pronta para executar após a finalização de T2, tendo como tempo de ativação t1, e de execução t1+t2;
- Finalmente, T4 estará ativa após a passagem de t1+t2+t3, sendo uma tarefa precedida por todas as outras do sistema.

Dependência entre recursos

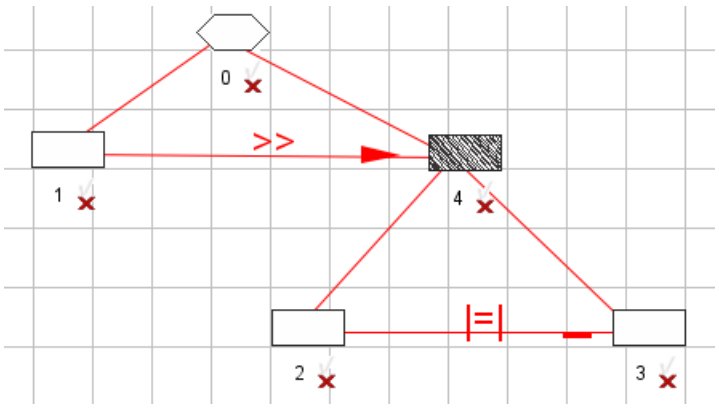
Como tratado anteriormente, a precedência em relação aos recursos tem comportamento básico semelhante ao apresentado aqui, apenas com algumas mudanças de funcionalidade. Um grafo de dependência entre os recursos pode ser tratado da mesma forma como foi visto para tarefas, porém com sua execução ordenada pelo acesso aos semáforos.

É importante destacar que o relacionamento de tarefas não afeta diretamente o relacionamento de recursos, pois são tratados separadamente pelo modelo. As relações entre recursos só estão ativas quando a tarefa que utilizar o recurso estiver executando. Por exemplo, supondo um sistema com três tarefas, como modelado na Figura 4.25, e que utilizam os recursos R1 e R2, requisitados como descrito a seguir, se tem a seguinte interpretação:

- T1 - Não utilizará recurso
- T2 - R1 >> R2
- T3 - R2 >> R1

O grafo de dependência das tarefas pode ser visto na Figura 4.26(a), construído da mesma forma como visto anteriormente. Já as Figuras 4.26(b) e 4.26(c) representam

Figura 4.25: Modelo com dependência de recursos.

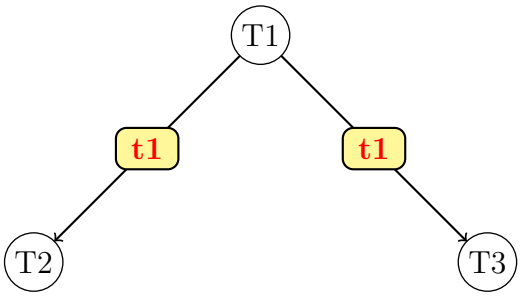


(Gerada pelo autor)

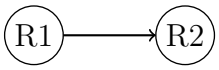
respectivamente os recursos relacionados em T2 e T3, que possuem ordem de precedência definida de forma inversa.

Figura 4.26: Grafos de dependência de tarefas e recursos

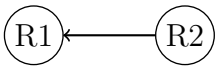
(a) Dependência entre tarefas



(b) Dependência de recursos para T2



(c) Dependência de recursos para T3



(Gerada pelo autor)

O fluxo de execução segue a seguinte sequência:

- T1 inicia sua execução por ser a única habilitada no instante inicial;
- T2 e T3 são independentes de ordem e se tornam ativas após a execução de T1;
- Com a ativação de uma das duas tarefas as relações definidas para os recursos por ela utilizados se tornam ativas e são consideradas de acordo com as chamadas de semáforos.

O que deve ser destacado deste exemplo é que o fluxo de execução das tarefas não é afetado pelo que foi definido para o relacionamento dos recursos. Isso ocorre pois estas relações são tratadas de modo independente, com os recursos sendo utilizados pelas tarefas apenas se estas estiverem ativas.

As questões referentes a bloqueio ou disponibilidade de recursos são tratadas diretamente no escalonamento, cabendo ao modelo e sua interface a configuração do comportamento e características de execução.

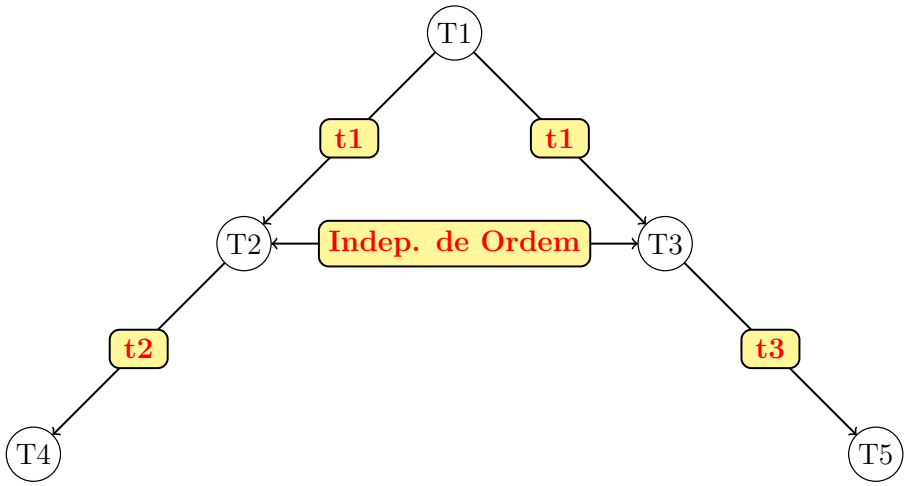
Fluxos independentes de execução

As outras formas de relacionamento definidas consideram independência total ou parcial entre as tarefas. Por este motivo, os fluxos de execução destas relações são construídos de maneira distinta. Esse é o caso da relação de independência de ordem, que não apresenta a execução determinística das relações de dependência.

A imprevisibilidade no sistema surge pois o bloqueio de uma tarefa em favor de outra é resolvido apenas em tempo de escalonamento. Na simulação isso é tratado por meio de listas de bloqueio e liberação e pela ordenação da Fila de Pronto.

O entendimento da construção desta relação é fundamental para o correto funcionamento do sistema. Esse entendimento é importante pois, além do compartilhamento de recursos, as subtarefas dependentes deste relacionamento serão afetadas pela decisão do sistema. Para exemplificar esta situação pode-se considerar o seguinte conjunto de tarefas e relações, correspondentes ao grafo de tarefas visto na Figura 4.27:

Figura 4.27: Grafo de execução considerando a independência de ordem



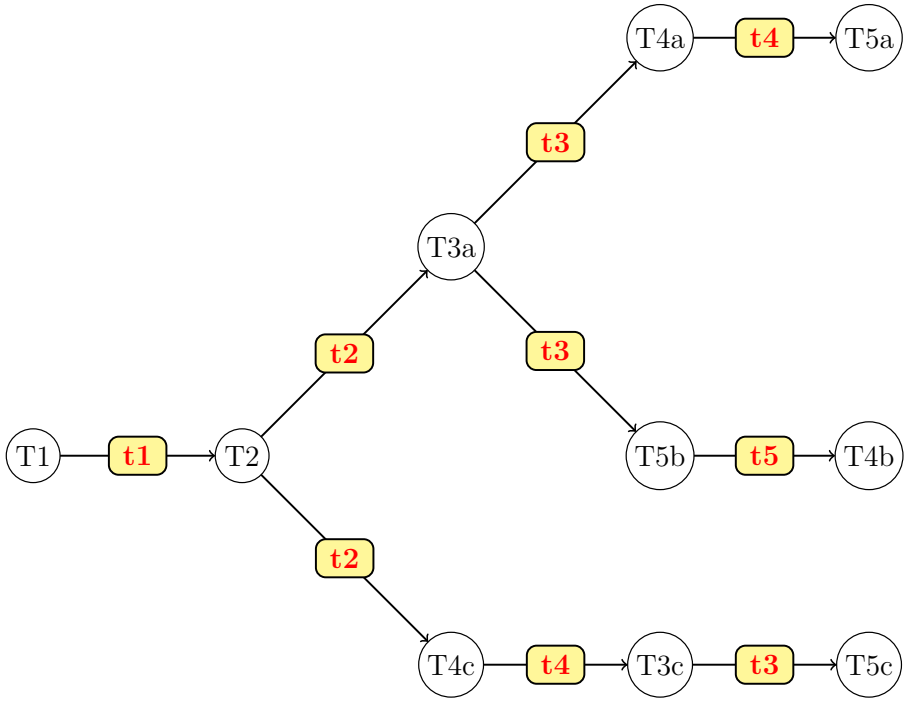
(Gerada pelo autor)

- Tarefas T1, T2, T3, T4 e T5;
- Relações:
  - T1 >> T2 e T1 >> T3;
  - T2 |=| T3;
  - T2 >> T4;
  - T3 >> T5

No grafo da Figura 4.27 as tarefas T2 e T3 são marcadas como independentes de ordem, sendo portanto independentes. Isso não implica, entretanto, que possam ocorrer ao mesmo tempo pela sua definição.

Na interpretação do modelo é preciso identificar esta relação e a subárvore de tarefas envolvidas, controlando quais fluxos são alterados pela ativação desta relação de independência. Para observar a influência deste relacionamento no modelo, na Figura 4.28 são exibidas as ramificações de comportamento que são tratadas pelos métodos desta relação Os fluxos possíveis estão separados pelos identificadores a, b e c.

Figura 4.28: Grafo de execução considerando a variação no fluxo de execução



(Gerada pelo autor)

O modelo construído pelo grafo considera em sua construção o bloqueio de T3 em favor de T2, permitindo que esta execute completamente a sua carga. Existem outras possibilidades de variação que são tratadas pelos interpretadores do modelo, esta foi escolhida apenas para exemplificar o comportamento da relação. Os possíveis caminhos de execução podem ser construídos de acordo com a seguinte lógica:

1. A execução começa com T1 seguida por T2;
2. Fluxo (a) de execução
   - Entre as tarefas habilitadas para executar (T3a e T4c) escolhe-se T3;
   - Entre as tarefas habilitadas para executar (T4a e T5b) escolhe-se T4;
   - Por fim, resta T5a para finalizar o fluxo;
3. Fluxo (b) de execução

- Após a escolha por executar T3, torna-se habilitada T5;
- Entre as tarefas habilitadas para executar (T4a e T5b) escolhe-se T5b;
- A tarefa restante passa a ser T4b;

4. Fluxo (c) de execução

- Este é o fluxo mais simples para a execução, escolhendo T4c após T2;
- Com T5c desabilitada, T3c é a única possibilidade de sequência;
- Finalizando a execução com T5c;

Outra observação direta que pode ser retirada das informações do grafo é a variação nos tempos de ativação de cada tarefa, que são diferentes para cada caminho. Por exemplo, o tempo de ativação da Tarefa T4 pode variar no seguinte formato: T4a = t1 + t2 + t3; T4b = t1 + t2 + t3 + t5; T4c = t1 + t2. Essa mudança de tempo afeta diretamente no funcionamento do modelo, podendo atrasar ou adiantar a execução de uma tarefa, demandando sua interpretação correta pelo simulador.

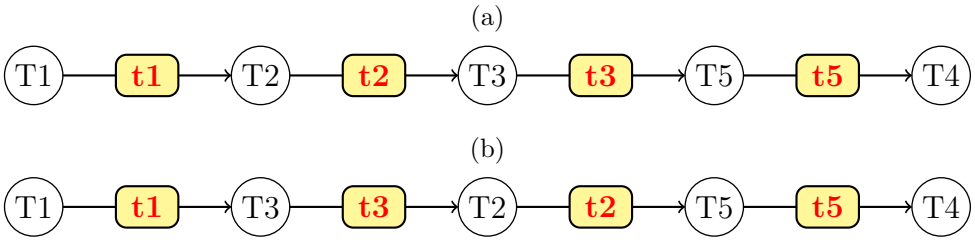
Este tipo de relacionamento é mais aplicado para tarefas concorrentes de modo geral. Para tarefas de tempo-real, que possuem restrições rígidas de execução, é necessário fazer o controle de ordenação pela indicação de prioridade entre as tarefas. Isso reduz as possibilidades de variação no tempo de ativação descritas no parágrafo anterior.

O uso de prioridades permite também direcionar os resultados de tarefas com Independência de Ordem, apesar da imprevisibilidade associada. Por exemplo, é possível alcançar o resultado exibido pelo fluxo (b) respeitando as prioridades definidas na Tabela 4.2, apesar de ainda se manter a imprevisibilidade entre T2 e T3. Nesse caso as duas únicas possibilidades de fluxo restantes são apresentadas na Figura 4.29.

Tabela 4.2: Atribuição de prioridades às tarefas

Tarefas	Prioridade
T1	1
T2/T3	2
T4	4
T5	3

Figura 4.29: Grafo de execução com controle de prioridade entre as tarefas



(Gerada pelo autor)

Relação de paralelismo

Finalmente, a relação de paralelismo também precisa ser considerada. Ela define um fluxo independente de execução, sinalizando no modelo que as tarefas podem ocorrer ao mesmo tempo, dependendo apenas da disponibilidade de recursos de processamento. A construção de um modelo com paralelismo segue a mesma forma de árvores hierarquizadas, porém replicadas pelo número de tarefas paralelas.

As árvores podem ser construídas normalmente na interface. Como o elemento processador demarca a raiz de cada árvore, o fluxo inicial de execução é feito de forma separada, controlando os relacionamentos existentes em cada uma delas.

Em ambientes com um processador esta relação continua sendo aplicada para indicar tarefas que podem acontecer juntas, dividindo a utilização do processador. Nesses casos o paralelismo seria feito por compartilhamento do processador no tempo, principalmente em situações em que o controle por prioridade não pode ser aplicado ao conjunto de tarefas.

Situações como a descrita anteriormente ocorrem em problemas em que há compartilhamento de recursos entre algumas tarefas no conjunto, e pela definição do problema elas devem ser tratadas como iguais para a execução, o que impede a indicação de prioridade. Para estas situações, as tarefas que não compartilham recursos podem ser sinalizadas como paralelas, o que será interpretado como um fluxo independente de carga de trabalho.

4.4 Considerações finais

Neste capítulo se apresentou o projeto desenvolvido e a especificação da proposta de modelagem de dependências entre tarefas concorrentes. Ao longo do capítulo foram justificadas as escolhas feitas na proposta e apresentados exemplos de como suas características e regras são aplicadas. No próximo capítulo será descrita a metodologia usada para transformar a proposta em uma implementação prática, com a sua aplicação no RTsim.

Capítulo 5

Métodos de Interpretação para Simulação

A metodologia para modelagem de relações entre tarefas descrita no capítulo anterior apenas faz sentido se for possível sua aplicação. Neste capítulo se descreve a implementação desta proposta no RTsim, o que também foi realizado utilizando a linguagem Java. Para manter a modularização na ferramenta, os métodos para geração da interface e interpretação de modelos foram adicionados como pacotes Java, separados por sua funcionalidade. Na sequência se apresenta a lógica de construção do tratamento adotado para cada relação do modelo e as implementações necessárias para efetivar a simulação.

5.1 Método de interpretação do modelo

A análise dos métodos de interpretação está baseada na descrição da lógica que foi utilizada para tornar possível a simulação do modelo gráfico. Nesse processo é preciso compreender quais informações compõem o modelo textual recebido pelo simulador e como elas são encontradas nos parâmetros informados na interface gráfica.

A estrutura do arquivo é simples, composta por seções que tratam aspectos bastante específicos. Cada elemento dos modelos gera um conjunto distinto de seções, sendo que no máximo são geradas cinco seções. As seções definidas são:

1. **Seção de Tarefas:** responsável pela principal parte da carga de trabalho simulada, as tarefas. As tarefas são exibidas na forma de listas, separadas por periódicas e aperiódicas, contendo seus parâmetros de execução para serem inseridos no simulador. O formato de representação adotado aqui, visto a seguir, é semelhante ao que era usado RTsim, facilitando a operação de leitura e inserção de dados.

   `Name = [nome da tarefa]`,
   `Load = [valor inserido na interface]`,
   `Period = [valor de período dado pelo usuário]`, caso seja periódica,

`Arrival = [instante de chegada]`, se periódica,
`Deadline = [deadline indicado no modelo]`,
`Prioridade = [prioridade inserida na interface]`

Para a inserção de dados referentes a tarefas aperiódicas são feitas algumas alterações em relação aos tipos de parâmetros, porém a estrutura é a mesma.

2. **Seção de Recursos:** Nesta seção aparecem as características relativas aos recursos necessários para a execução de uma tarefa, como por exemplo dados relativos às regiões críticas que guardem um recurso. O formato é semelhante ao das tarefas, divergindo quando aos tipos de informação, como visto a seguir.

   `Name = [Nome da semáforo]`,
   `Task = [Tarefa a qual este recurso foi atribuído]`,
   `Semaphore = [Semáforo utilizado para a região crítica]`,
   `CSL = [Carga utilizada na região crítica]`,
   `CST = [Instante de entrada na região crítica]`

   Embora recursos sejam associados à tarefas, optou-se por separar dados sobre recursos de suas tarefas. Isso foi feito para manter a leitura mais clara da entrada e permitir a associação de um recurso à várias tarefas pela associação das réplicas dos dados dos recursos. Isso evita também que se tenha uma estrutura mais carregada para a listagem de tarefas.

3. **Seção de Relações entre Recursos:** Uma vez listadas as informações sobre tarefas e recursos se passa ao detalhamento desses dados do ponto de vista de relações. A primeira categoria diz respeito às relações entre os recursos. Esses dados são incluídos com uma linha para cada tarefa e relação entre recurso. O formato seguido para este estabelecimento foi pensado para ser claro inclusive quando lido pelo usuário, sendo composto por:

   `[Nome tarefa]: [Nome recurso] < relação > [Nome recurso]`

   Sendo `[Nome tarefa]` o identificador da tarefa que recebeu em sua interface a configuração do relacionamento exibido.

4. **Seção de Elementos de Processamento:** Esta seção contem os processadores adicionados ao modelo e seu objetivo principal é o controle de tarefas associadas a cada elemento deste tipo. As informações aqui apresentadas exibem o número de processadores existentes no modelo, seguidos por uma listagem de informações.

   Para esta listagem, é fornecido o identificador do elemento de processamento e os identificadores de cada tarefa atribuída a este elemento pela interface. Para compor esta seção é que são feitas as ligações entre processador e seus componentes na

interface principal, pois desta forma é possível separar o conjunto de tarefas recebido na entrada para cada processador associado.

5. **Seção de Relações entre Tarefas:** Por fim, a quinta seção exibe o relacionamento entre as tarefas, deixada separada ao final do arquivo, pois, dependendo do tamanho do modelo construído, pode ter um grande número de relacionamentos para comportar.

   Semelhante a estrutura de relação entre recursos, esta também é separada em linhas e contem um identificador de cada lado da relação. Esta forma bilateral de leitura é preparada para o simulador, sendo que neste arquivo já está resolvida a hierarquia de grupo presente nas relações.

Com as informações recebidas no arquivo de entrada, os métodos de escalonamento estão prontos para executar o que foi proposto. A seguir serão vistas as formas de tratamento separadas por tipo de relação, para permitir uma visão mais clara do tratamento.

5.2 Métodos para interpretação dos elementos

A formatação de dados descrita na seção anterior é a base do processo de interpretação do modelo gráfico para a estrutura a ser simulada pelo RTsim. O processo de interpretação, para cada tipo de elemento ou relação, é descrito a seguir.

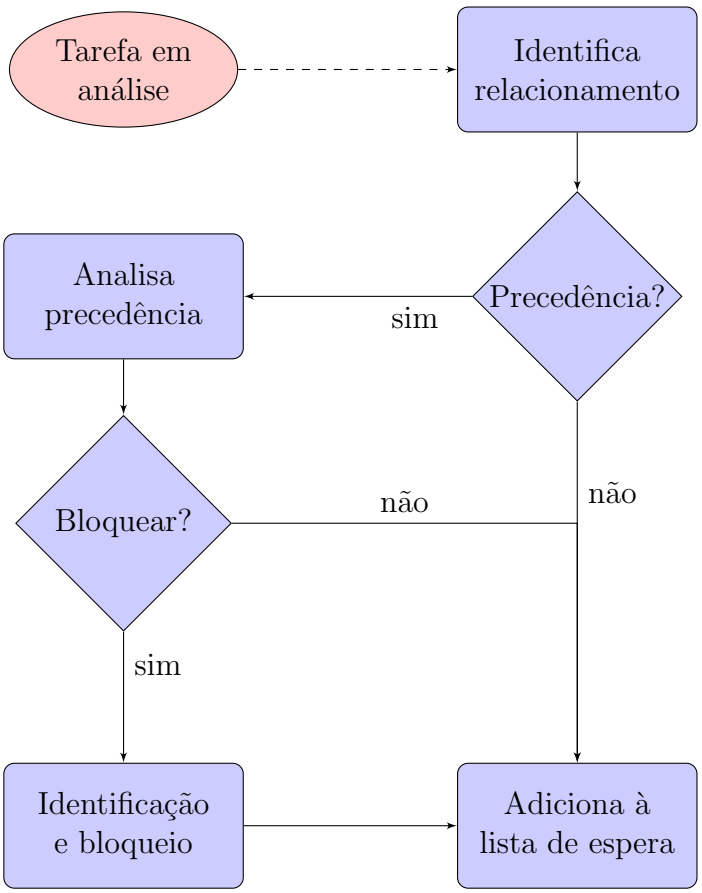
Precedência

Se for identificada uma relação de precedência, a primeira ação a ser tomada é a indicação disso nos objetos "Tarefa" envolvidos. Cada objeto suporta uma lista de dependência, cujo preenchimento indica quais são as tarefas que devem estar completamente executadas antes da ocorrência desta tarefa precedida. Estas informações são acrescentadas à lista geral de eventos que será usada pelo simulador, recebendo o nome de lista de espera.

A lista de espera é usada no processo de escalonamento das tarefas sendo simuladas. Esse escalonamento é executado pela função *escalonar(tempo)*, começando pela chamada do método *gerarListadeEspera(tempo,proximoTempo)*, que já existia no RTsim, mas foi alterada para considerar a situação de precedência para adicionar tarefas a esta lista.

O funcionamento da geração da lista de espera é visto no fluxograma apresentado na Figura 5.1. Para cada tarefa identificada pela lista de espera como pronta para executar, é realizado um chamado pelo método de identificação de dependência, que fará o bloqueio caso esta tarefa seja precedida por alguma outra que ainda não finalizou sua execução. Com a adição de métodos para validação da habilitação da tarefa a função de escalonamento pode ser mantida na forma original, evitando modificações excessivas no simulador.

Figura 5.1: Fluxograma do tratamento de precedência no modelo



(Gerada pelo autor)

Adicionar tarefas ainda não habilitadas para execução na lista de espera evita alterar a lista a cada término de uma tarefa. Assim, quando ocorrer o término da tarefa da qual uma tarefa bloqueada é dependente, esta já se encontra pronta para executar.

O método *escalonar* original do RTsim sofreu o mínimo de modificações possível, alterando apenas a direção de suas execuções e a inserção de ações diante de um bloqueio e liberação. Os trechos que sofreram modificação são os responsáveis pela escolha de tarefas para executar e pelas ações a serem tomadas após o encerramento. Os passos são:

i. Ordena lista de espera por prioridade;

ii. Captura a tarefa no topo desta lista;

iii. Se a escolhida estiver bloqueada, percorre a lista até encontrar uma tarefa habilitada para execução;

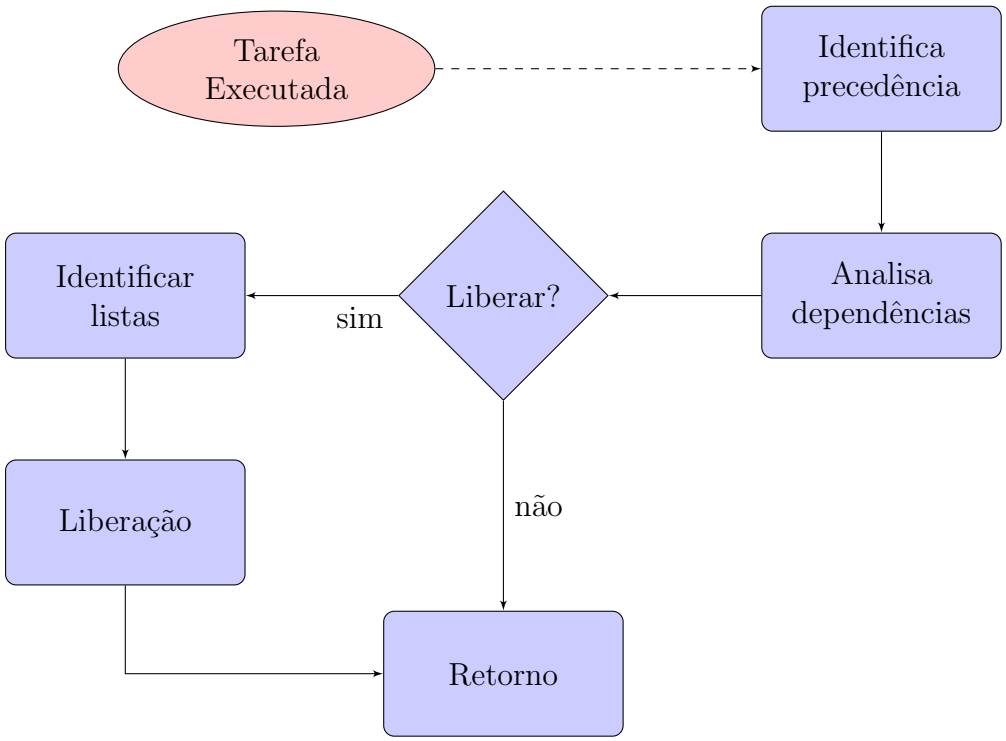
   - Se toda a lista estiver bloqueada, avança o relógio do simulador;

iv. Segue funções de execução com a tarefa escolhida;

v. Ao finalizar a execução de toda a carga, ativa os métodos de liberação se necessário.

O método de liberação é ativado se a tarefa que finalizou sua execução preceder alguma tarefa na lista. O objetivo desta função é liberar tarefas que foram bloqueadas e aguardam para executar. Seu fluxograma é apresentado na Figura 5.2 em que é exibida a construção deste método, que busca apenas identificar corretamente as tarefas e remover a variável de bloqueio.

Figura 5.2: Fluxograma do tratamento de precedência no modelo - Função de Liberação de bloqueio



(Gerada pelo autor)

Para os casos em que há mais de uma tarefa precedente à que foi liberada, e esta ainda depende do bloqueio ocasionado por uma segunda tarefa ainda não executada, haverá a manutenção de seu bloqueio na lista. Essa manutenção ocorre pela reanálise das tarefas dependentes durante a atualização da lista de espera. Deve ser observado que esses detalhes são tratados internamente, bastando que o usuário indique as várias relações de dependência.

Entender o tratamento exibido aqui para a relação de precedência apoia a compreensão das outras relações implementadas. Isso ocorre porque estas trabalham com o mesmo conceito de bloqueio e liberação de tarefas, alterando-se apenas a forma de tratamento da relação.

Precedência com passagem de informação

Esta relação se diferencia da precedência simples pelo trecho para troca de informação entre tarefas. É preciso lembrar que não se exibe, nem se formaliza, a troca de informação, que é tratada apenas como um evento instantâneo que ocorre em algum momento da execução da tarefa precedente. Isso indica que os mesmos procedimentos vistos nas Figuras 5.1 e 5.2 poderia ser utilizado. Entretanto optou-se por criar um método distinto, para que já se tenha o método de tratamento e exibição de troca de informações pronto.

A diferença é o acréscimo de uma lista, em que tarefas passam a ter uma lista de tarefas precedidas que receberão informação. Como a função *escalonar(tempo)* é a mesma para todas as relações, sua primeira ação é gerar a lista de espera. Nesta função de geração há um chamado para o método que verifica a existência de passagem de informação na tarefa selecionada, gerando-se a lista de precedidas nesse caso.

Para manter a homogeneidade do simulador usou-se aqui a mesma estrutura do método apresentado na Figura 5.1, ficando as principais alterações ligadas ao metodo de liberação de um tarefa. Nele se implementou alterações para tratar tarefas deste tipo de relação, considerando as listas de informações apropriadas para sua simulação. O método segue o mesmo formato para alcançar o bloqueio da tarefa.

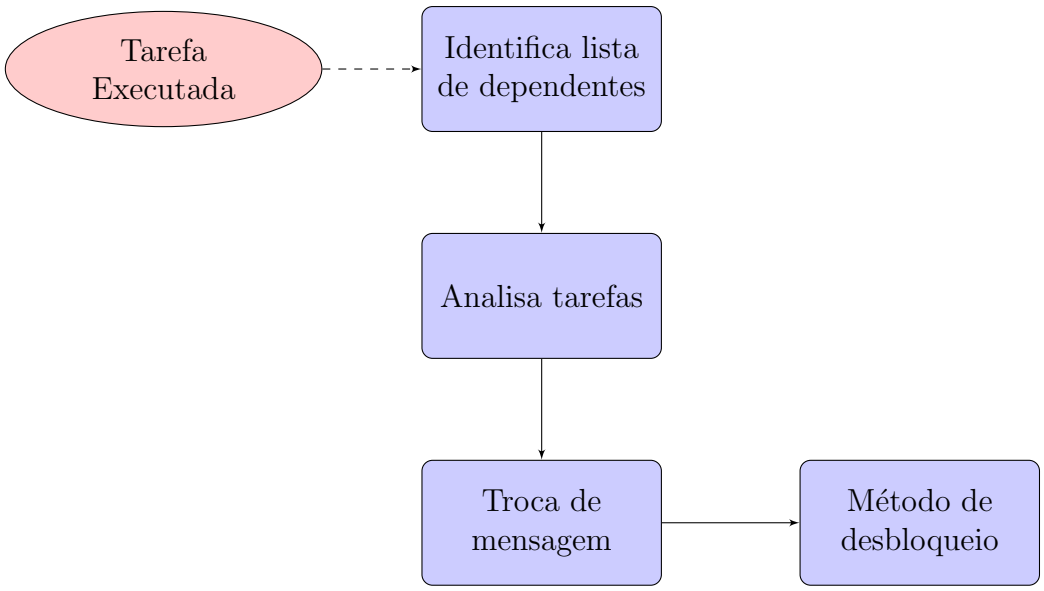
A análise de dependências da tarefa, e seu eventual bloqueio, é realizada de forma equivalente quer exista ou não troca de mensagens. Da mesma forma que o método anteriormente exibido, se for necessário o bloqueio da tarefa, esta ainda assim será adicionada à lista de espera, porém com o estado alterado para impedir a sua execução.

Além disso, o escalonamento também segue a mesma ideia do tratamento de tarefas precedentes. Como as tarefas são adicionadas como bloqueadas na lista de espera, a função não faz diferenciação quanto ao motivo do bloqueio, seguindo seu fluxo padrão de execução. Ao selecionar uma tarefa para execução, o método identifica se ela está bloqueada observando a variável de controle, e prossegue sua execução normalmente, sem a necessidade de considerar qual relação ou tarefa ocasionou o bloqueio.

Este tipo de decisão durante a implementação garantiu que a função *escalonar(tempo)* recebesse o mínimo de modificações em seu código, mantendo o padrão existente no RT-sim. Isso é importante pois este projeto não visava alterar desnecessariamente sua implementação original. Embora não tenha sido tentado, é possível inferir que ter menos especificidades facilita o uso desta metodologia de modelagem em outros simuladores.

A Figura 5.3 traz o fluxograma da função de liberação, que é a principal modificação em relação ao realizado para dependência simples, visando a demarcação do instante de troca de mensagem. É um pouco mais extensa que as outras exatamente por realizar o controle sobre dois tipos de listas estruturais nas tarefas, tratando liberação e troca de mensagem de forma separada. A estrutura dividida em duas partes permite visualizar os tratamentos principais separadamente.

Figura 5.3: Fluxograma do tratamento de precedência com passagem de informação no modelo - Trecho de sinalização da troca



(Gerada pelo autor)

O diferencial nesta implementação é o bloco que trata da troca de informação, controlando as tarefas presentes na lista de habilitação da tarefa executada. Esse bloco concentra seu trabalho na identificação das funções que devem receber a informação produzida e na sinalização desta troca para acompanhamento do usuário. Embora a interface do simulador não traga ainda sinalizações das passagens no seu gráfico de escalonamento, este método de tratamento foi preparado para adaptação as necessidades futuras.

Independência de Ordem

A independência de ordem tem um propósito diferente das relações tratadas até aqui. Elas são consideradas como independentes na simulação até que uma delas ocasione o bloqueio da outra. Sua funcionalidade está ligada ao uso de recursos, embora implementada também para algoritmos que não consideram recursos em seu escalonamento. Isso permite que o usuário possa analisar seu comportamento em cargas de trabalho mais simples.

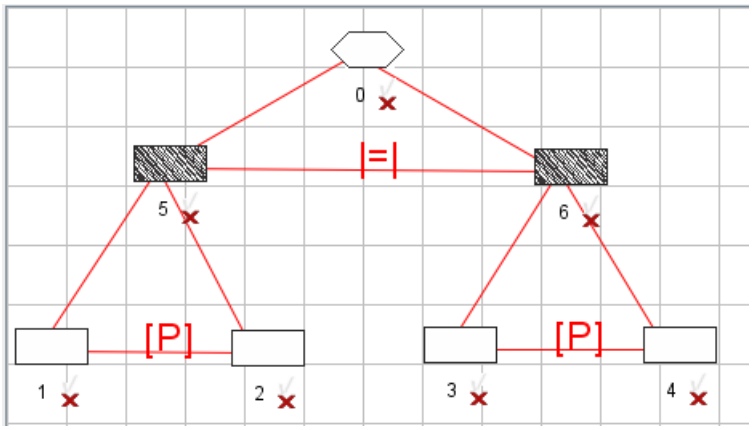
Esta relação, por ser bijetora, precisou de mudanças no processo de identificação das tarefas para ter um controle sobre seu comportamento. Desta forma, quando a independência de ordem é identificada entre duas tarefas, cada uma delas inclui a outra tarefa em uma lista específica, composta apenas por tarefas que tem essa relação. Por exemplo:

- Identificada a relação P1 |=| P2 ;
- Para P1: adicionar a tarefa P2 em sua lista individual de tarefas independentes;
- Para P2: adicionar a tarefa P1 em sua lista individual de tarefas independentes;

Com a lista de espera pronta a função *escalonar(tempo)* inicia seu funcionamento normalmente. É válido lembrar que esta relação não causa sinalização de bloqueio em nenhuma das tarefas. Assim todas as tarefas podem compor a lista de espera apresentada ao escalonador, desde que estejam prontas, deixando o algoritmo livre para escolher qual delas executará primeiro. A função de bloqueio só é ativada quando uma delas for escolhida para iniciar a execução, seguindo o propósito da relação.

O modelo exibido na Figura 5.4 apoia o entendimento da estrutura que foi adotada. Neste exemplo será desconsiderada a relação de prioridade e a configuração dos elementos, destacando apenas a interpretação da relação de independência de ordem entre as tarefas do grupo.

Figura 5.4: Exemplo de modelo construído na interface



(Gerada pelo autor)

Com a resolução da hierarquia de grupos, a interpretação textual que irá compor o modelo, fornecerá as seguintes relações (desconsiderando as prioridades):

- P1 |=| P3
- P2 |=| P3
- P1 |=| P4
- P2 |=| P4

Supondo que o tratamento da relação desconsiderasse sua bijeção, as listas poderiam ser associadas a apenas um dos lados. No caso de associação ao lado contendo P1 e P2, então as listas produzidas seriam:

$$\overbrace{P3 \quad P4}^{P1} \qquad \overbrace{P3 \quad P4}^{P2}$$

Caso P3 entre em execução a estrutura de bloqueio precisa buscar sua ocorrência em todas as listas para encontrar os relacionamentos com a tarefa P1 e P2. Agora, se for considerada a bijeção da relação tem-se o seguinte conjunto de listas:

$$\overbrace{P3 \quad P4}^{P1} \qquad \overbrace{P3 \quad P4}^{P2} \qquad \overbrace{P1 \quad P2}^{P3} \qquad \overbrace{P1 \quad P2}^{P4}$$

Desta forma, a composição da lista de tarefas de um cada elemento já traz todas as informações para realizar o bloqueio corretamente. Assim, basta verificar a lista de P3 quando esta fosse colocada para executar e obter diretamente que P1 e P2 deveria ser bloqueadas.

Ao final da execução da tarefa selecionada, esta servirá como parâmetro para a verificação para liberar as tarefas que estão bloqueadas. Seguindo a lista de tarefas independentes mantida pela tarefa executada, inicia-se o processo de liberação dos bloqueios, que ocorre de forma semelhante ao descritos para as outras relações.

Paralelismo

A relação de paralelismo foi implementada para atender as necessidades dos simuladores que permitem a migração de tarefas entre seus processadores. Isso permite a utilização da abordagem para sistemas executando em ambientes multiprocessados. O funcionamento desta relação é bem simples, pois sua função principal é a sinalização das tarefas que podem ocorrer ao mesmo tempo.

Para implementação desta relação no simulador RTsim considerou-se que tarefas paralelas são de fato independentes. Assim não é necessário vincular estas tarefas a nenhumas das listas usadas pelas relações anteriores. Isso permite que um algoritmo que use migração apenas tenha que usar a lista de tarefas independentes para realizar a troca.

O momento para a migração de tarefas não é definido pelo usuário em seu modelo. A migração ocorre de acordo com a política de escalonamento adotada, possivelmente com a análise de folgas e das chances de perda de *deadline* associadas ao momento. É importante lembrar que forçar migrações específicas não é representativo para uma carga de trabalho efetivamente genérica.

Embora o paralelismo não seja objetivo de modelos de sistemas com apenas um processador, esta funcionalidade também está disponível. Para tais sistemas ela será entendida como uma relação entre tarefas independentes para fins de simulação. Isto foi adotado porque a sinalização do paralelismo torna o processo de modelagem do problema mais natural para os usuários, e deixa a interface mais amigável.

Em outros simuladores, com funcionamento que permita a relação do paralelismo apenas por grupos específicos de tarefas, a estrutura do modelo está preparada para trabalhar com listas de tarefas. Desta forma, as adaptações necessárias seriam relacionadas a adições de lista de tarefas que podem ser trocadas entre os elementos de processamento, pois os métodos de identificação desta relação já estão preparados nas funções de interpretação.

Prioridade

Como explicado nas seções anteriores, a relação de prioridade permite que tarefas independentes possam ser configuradas em conjunto com as outras relações. O RTsim, em

sua implementação original, trata as tarefas como independentes, apenas considerando a relação de prioridade entre elas para poder escaloná-las. Desta forma, a relação de prioridade passa a ser a sinalização necessária para que o simulador siga, com este grupo de tarefas, a sua implementação original para escalonamento.

É possível perceber que a estratégia de implementação adotada neste trabalho, visa modificar o tratamento das relações extras que estão compondo o modelo, separando estas em listas e adicionando variáveis de controle, mantendo o tratamento original quando necessário. Isto foi planejado para não causar modificações na função de escalonamento, preservando a estrutura da implementação original, pois uma vez que este trabalho se propôs como um modulo externo a ser adicionado, modificações na estrutura de tratamento do simulador alterariam este propósito.

Recursos

Os recursos receberam um tratamento especial neste trabalho, permitindo inclusive relacionamento de precedência entre eles. Estes foram escolhidos para compor o modelo por sua importância em sistemas concorrentes, em que se controla o acesso a diversos tipos de recursos para atendimento das tarefas. Assim se percebe a importância de tratar relacionamentos entre eles, adequando o modelo às necessidades das tarefas.

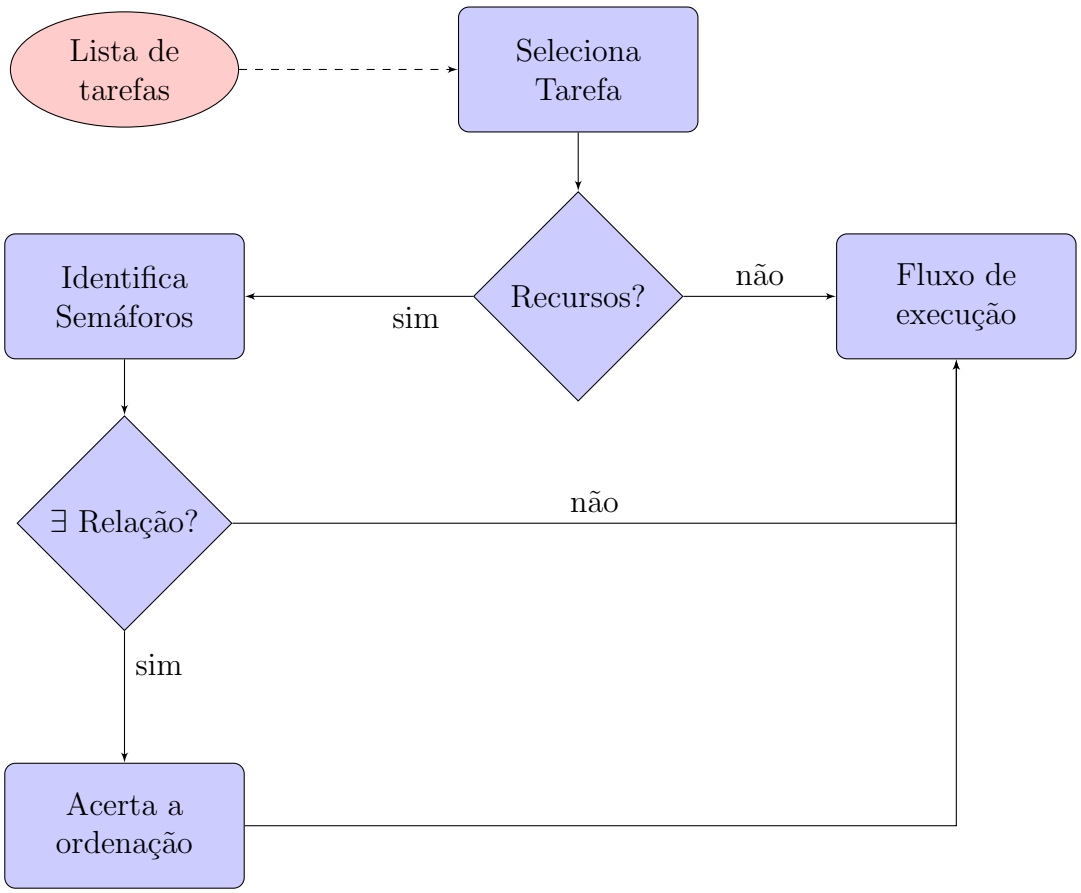
Os métodos para tratamento de recursos foram adicionados apenas aos escalonadores que tratam recursos explicitamente. Isso ocorreu pois a relação atua sobre o escalonamento quando este considera a existência de recursos nas tarefas. Para a adição de recursos às tarefas criou-se métodos para interpretar os parâmetros de entrada associados as tarefas e gerar as informações em listas específicas no escalonamento.

O fluxo da simulação de tarefas com recursos segue a mesma abordagem utilizada em outros algoritmos. Altera-se apenas o tratamento em casos em que a tarefa está bloqueada por falta de recurso. A estrutura de bloqueio, porém, funciona da mesma forma que a adotada para as tarefas precedentes.

A modificação principal é feita dentro da função *escalonar(tempo)*, com apoio de métodos externos, quando se verifica a existência de relação entre os recursos da tarefa selecionada. Na Figura 5.5 é abordada parte da lógica adotada pela função *escalonar* considerando os recursos modelados. O foco das alterações se concentra no comportamento do método que ordena as listas de recursos. O acerto da ordenação é feito por um método externo, que executa ações de ordenação da seguinte forma:

- Identifica os recursos associados a tarefa que está em análise;
- Identifica as relações e os recursos que sofrerão alterações;
- Busca na lista de controle por cada recurso e suas características;
- Adapta o instante de entrada na região crítica e sinaliza a existência de precedência;

Figura 5.5: Fluxograma do escalonamento com recursos modelados



(Gerada pelo autor)

Um outro método importante para este funcionamento é a busca na lista de semáforos. Com as informações atribuídas à tarefa, o método passa a buscar pelo semáforo que aguarda o início da execução, respeitando as relações estabelecidas pelo modelo. As estruturas de bloqueio funcionam de forma semelhante ao bloqueio de tarefas já apresentado. Cada recurso utilizado recebe uma variável sinalizando seu bloqueio e liberação, e é por esta estrutura que é possível controlar se a tarefa pode iniciar sua execução, executar sua carga completamente ou ser bloqueada até a liberação do recurso.

5.3 Configuração dos métodos de simulação

Dando maior clareza ao que foi implementado, nesta seção serão apresentadas algumas estruturas das classes e dos algoritmos implementados. Com isso, será possível compreender de maneira geral a implementação realizada e permitir um certo grau de repetibilidade do que foi proposto em outros simuladores. É válido lembrar que as funções e implementações apresentadas a seguir não estão totalmente representadas nos algoritmos. A ideia é apenas trazer os conceitos do funcionamento dos métodos, tornando a apresentação sobre a lógica do projeto mais didática e clara.

5.3.1 Configuração dos componentes da interface

Todo o processo de simulação depende de como as informações são capturadas da interface de desenho. Como a captura depende do tipo de tarefa, serão apresentados aqui os métodos para configuração das tarefas periódicas, escolhidos para servir como exemplo. É importante ressaltar que as estruturas básicas são as mesmas para todos os componentes do modelo, apesar das classes serem construídas individualmente, respeitando as necessidades individuais de cada elemento que será criado.

A estrutura da classe responsável pelo elemento *Tarefa Periódica* e seus parâmetros de configuração é descrita no Algoritmo 1. É possível observar que características como *identificador*, *nome*, entre outros, não estão presentes nesta estrutura. Esses parâmetros são mantidos em outra estrutura, da classe *IdentificadorItemGrade*, contendo parâmetros comuns entre as classes, o que facilita a organização do código e das chamadas.

Algoritmo 1: Classe de parâmetros para tarefas periódicas

Entrada: Dados que forem inseridos na interface
Saída: Variáveis da tarefa recebem os valores adicionados pelos métodos

```
1  início
2      Instanciação do objeto da classe IdentificadorItemGrade para geração dos
         parâmetros gerais
3      Inicialização de variáveis do componente
       /* Métodos para inserção e captura de valores                    */
4      início
5          setCarga();
6          getCarga();
7          setPeriodo();
8          getPeriodo();
           /* Entre outros métodos para cada tipo de componente         */
9      fim
       /* Funções de suporte à configuração e utilização dos parâmetros
          de entrada                                                    */
10     Verifica se já está configurado
11     Chamada da função gráfica para desenho na tela principal
12     Captura de valores para colocação no arquivo XML da interface
13 fim
```

A instanciação para todas as classes de configuração básica dos componentes do modelo segue este padrão. Isto facilita a compreensão do funcionamento do código e torna a implementação mais clara. Cada uma destas classes possui sua referente implementação dos parâmetros presentes no painel como classes de outro pacote, e são estas classes que realizam as chamadas dos métodos de captura e inserção tratados aqui.

No Algoritmo 2 são exibidos os métodos de captura e inserção, também para tarefas periódicas como exemplo. Esses métodos são responsáveis por controlar a estrutura vista

na interface do painel de configuração (Figura 4.16). É possível observar pela interface exibida que a estrutura de entrada dos parâmetros para cada componente segue um formato de tabela. Este tipo de estrutura permite uma melhor manipulação das células existentes, com controle realizado individualmente para cada tipo de elemento.

As decisões ilustradas no Algoritmo 2 são voltadas para controlar o recebimento de dados fornecidos pelos usuários e o correto preenchimento dos objetos instanciados. Deve-se observar que estas entradas representam características importantes para simulação das tarefas, e devem fazer parte do arquivo final gerado com a interpretação do modelo.

Destaca-se também que há um trecho de implementação dedicado à configuração exclusiva dos recursos na tarefa. Esta foi feita individualmente para cada componente tarefa por envolver identificação individual de cada objeto criado. Além disso, pode-se observar que há chamadas de funções criando novas interfaces, de acordo com as características de cada uma. Isso justifica a necessidade da pergunta inicial sobre a ação que o usuário irá tomar quanto a configuração do recurso.

Apesar da representação simplificada dos métodos presentes nos Algoritmos 1 e 2, é possível perceber que o segundo algoritmo tem a implementação mais completa quanto ao tratamento dos métodos. Essa característica é padrão também para os outros componentes, o que mantém a classe principal mais simplificada, com as funções necessárias apenas para inserção dos dados de simulação. Já a segunda absorve a complexidade de tratamento das interações com o usuário na interface e disponibilização de valores nas células.

5.3.2 Configuração das ligações de relacionamento da interface

Diferente dos componentes apresentados na seção anterior, a configuração de relações é mais simples, levando em consideração basicamente o seu posicionamento e função atribuída pelo modelo. Há uma classe básica que implementa as funções de posicionamento da relação, configurada como um modelo abstrato. Ela configura apenas as necessidades comuns a todos os tipos de relação, reservando as configurações de interface e modelo para as classes particulares de cada tipo de relação.

Como exemplo geral, se apresenta no Algoritmo 3 a configuração básica dos métodos para estabelecer a relação de precedência. Há classes semelhantes para cada tipo de relação, as quais são responsáveis por controlar a origem e destino da ligação, além das funções atribuídas às características da relação. Nesse algoritmo é possível observar que a configuração das relações é menos complexa que a referente aos componentes, principalmente por não receberem parâmetros de entrada para tratamento ou características inseridas pelos usuários.

Com esta classe e a classe base associada a ela, é possível estabelecer relações na interface de forma rápida, associando sempre um identificador do tipo de relacionamento.

Algoritmo 2: Classe de entrada de configuração do elemento Tarefa Periódica

Entrada: Inicia-se com a abertura do painel de configuração, e recebe como entrada os dados inseridos pelo usuário
Saída: Confirma ou atualiza as configurações do elemento

```
1  início
   /* Configuração do botão de adição de recursos                    */
2  início
3      if Já existe uma interface com modelo de recurso configurada then
4          Carrega a interface salva e abre para edição
5      else
6          if Escolha do usuário é para modelar o comportamento dos recursos
            then
7              Gera nova interface para modelar o recurso vinculada ao id da tarefa
8              Abre para configuração
9              Atualiza variável que controla a existência de modelo de recursos
                para as tarefas
10         else
11             if Escolha do usuário é para modelar seção crítica para elemento de
                processamento then
12                 Gera interface para receber configuração de semáforo e
                    parâmetros de execução da região crítica
13                 Abre para edição sinalizando que o recurso configurado é o
                    processador
14             else
15                 if Escolha do usuário sinaliza que não haverá configuração de
                    recurso then
16                     Segue a edição de parâmetros da tarefa normalmente
17             end
18         end
19     end
20 fim
   /* Métodos para exibição de parâmetros já configurados e para
      passagem de valores inseridos                                  */
21 Insere o valor de determinada posição selecionada no método específico que está
    implementado na classe principal do elemento início
22     Caso seja a linha correspondente ao nome, localize o elemento pelo
        identificador na grade e insira o conteúdo fornecido pelo usuário
23     Caso seja a linha correspondente a carga, acione o método setCarga()
        configurado para este elemento
       /* Entre as entradas configuradas para cada linha no painel   */
24 fim
25 Exibe o valor dos parâmetros já configurados na posição adequada.
   /* Funções para controle da estrutura de exibição em tabela       */
26 Sinaliza quais células são editáveis pelo usuário
27 Retorna nome a ser exibido na coluna
28 Retorna número de linhas e colunas
29 fim
```

Algoritmo 3: Classe de configuração para relação de precedência

Entrada: Chamada do método de manipulação da classe de interface para desenho da relação e configuração

Saída: Desenho gráfico e configurações feitas na relação

```
1  início
2  |  Instanciação do objeto da classe IdentificadorItemGrade para geração dos
   |   parâmetros gerais da relação
3  |  Inicialização de variáveis da ligação
4  |  Captura de valores para inserção no arquivo XML da interface
5  |  Método para desenho na interface principal, considerando posicionamento e
   |   identificador de relação.
   |  /* Métodos para captura de posicionamento                              */
6  |  início
7  |  |  getX()
8  |  |  getY()
9  |  fim
10 fim
```

Este identificador é passado para as informações geradas e serve para facilitar o processo de interpretação do arquivo final.

5.3.3 Configuração da classe de funções da interface principal

Para que a classe principal de desenho da interface não seja sobrecarregada é necessário que o tratamento das ações do usuário na tela principal fiquem em outra classe. Por este motivo, os métodos de tratamento dos objetos da interface de desenho foram concentrados em classes específicas de configuração, sendo construídas separadamente, uma para configuração dos elementos da interface principal e uma para os componentes da interface de recurso.

Como a interface de recursos envolve apenas manipulação de listas, como indicado na seção 4.2.1, apenas a classe de configuração da interface principal, conhecida como *DesenhoGrade.java*, será apresentada aqui. Deve ser observado entretanto que as duas interfaces apresentam funcionamento semelhante. Na interface principal se concentram as funções para chamadas de métodos das classes de configuração e preparação de arquivo final, os quais serão apenas indicados no algoritmo.

No Algoritmo 4 exibe-se a classe de configuração da tela principal, seus métodos se concentram em chamadas de métodos de configuração, o que torna esta classe fundamental para o correto funcionamento do modelo, executando o controle sobre as ações realizadas na interface.

Algoritmo 4: Classe de configuração da interface principal

```
1  início
2      Inicialização de variáveis da classe
3      Inicialização de botões de opção da tela, como por exemplo Remove, associado
         ao componente.
       /* Métodos para adição de aresta entre os componentes, como
          controle de origem destino e tipo de relação                    */
4      início
5          adicionarAresta(origem,destino);
           adicionarArestaPrecedencia(origem,destino);
           adicionarArestaParalelo(origem,destino);
           adicionarArestaPrioridade(origem,destino);
           adicionarArestaHabilitacao(origem,destino);
           adicionarArestaIndependencia(origem,destino);
           /* Com adição da relação na forma textual ao arquivo final    */
6      fim
7  fim
   /* Tratamento semelhante para remover uma relação da interface       */
8  Método para adição de vértice, com estrutura de dados para chamada da classe do
     componente de acordo com o tipo.
9  Métodos para configurações gerais dos itens da grade
10 Método para captura de valores da grade, getGrade() para inserção das
     informações no arquivo XML.
11 Método para exibição da tabela de parâmetros para os componentes
12 Métodos para tratar confirmação destes parâmetros, incluindo adição das tarefas e
     componentes já configurados a uma lista específica
13 Captura de valores para inserção no arquivo XML da interface
14 Método para desenho na interface principal, considerando posicionamento e
     identificador de relação.
15 Método para inserção de valores em setGrade, para recuperação de dados do XML
16 Método para verificação de configuração dos elementos
```

5.3.4 Configuração para conversão do modelo

Com as classes apresentadas anteriormente, a conversão se torna uma tarefa mais simples. Isso ocorre pois ações de tratamento e armazenamento das informações já foram realizadas nas classes anteriores. Assim, para as funções de conversão é necessário apenas a identificação dos dados e sua correta interpretação.

O acionamento do botão *Generate*, na interface principal, dá início aos métodos de captura dos dados e aos acertos necessários para compor o modelo textual do que foi desenhado na tela. Este modelo é construído em uma forma padronizada de modo a tornar possível sua interpretação, independente do conteúdo desenhado.

No Algoritmo 5 é tratada uma parte do tratamento para geração do modelo padrão. Neste se destaca o método *acertaModelo()*, que se ocupa da resolução de hierarquia dos grupos. Esse é o método responsável por produzir a interpretação correta da ordem de aplicação das relações inseridas de acordo com o que foi interpretado do desenho.

Algoritmo 5: Método para preparação do modelo padrão

```
1 início
      /* Verifica se os componentes estão corretamente configurados   */
2     validaDesenho();
3     Trata o processador e suas tarefas associadas, para que seja feita a correta
       captura do conjunto de tarefas atribuídas a cada processador
4     Trata grupos e as tarefas atribuídas a cada grupo
      /* Resolve a hierarquia, transformando grupos e tarefas e
         adicionando ao modelo as relações definidas para cada tarefa */
5     acertaModelo();
      /* Salva saída em arquivo texto                                  */
6     salvarSaida();
7 fim
```

Deve ser lembrado que o objetivo do método de acerto do modelo é resolver a hierarquia entre as relações, removendo a informação do grupo e resolvendo a relação de acordo com cada tarefa. Este método não será mostrado aqui na forma de um algoritmo, por se tratar quase que completamente de estruturas de repetição e checagem de componentes, além de ser apenas um método de suporte para resolver o problema dos conjuntos de tarefas.

5.3.5 Configuração nas classes de algoritmos de escalonamento já existentes na ferramenta

Como visto na Seção 2.5.2 o RTsim já possui algumas políticas de escalonamento nativas, trabalhando sobre tarefas independentes. Para tornar possível a simulação das tarefas dependentes, vindas da interface de modelagem, algumas alterações nas classes do motor de simulação foram necessárias. As classes receberam métodos e funções extras

para administrar o tratamento das relações, sendo implementados individualmente para cada política de escalonamento, respeitando suas características particulares e buscando facilitar manutenções futuras.

Para as classes de interface as novas configurações se concentraram na recepção e tratamento do arquivo textual contendo o modelo. Para esta entrada, os métodos se concentraram em identificar o conjunto de tarefas e suas características, e destiná-lo às listas de execução. Também foi preciso implementar a identificação para cada tipo de relação introduzida, extraindo as informações necessárias do arquivo de entrada.

As modificações de maior vulto se concentraram nos algoritmos de escalonamento de cada política, pois o motor de simulação não estava preparado para tratar estas relações, necessitando de adaptações. Os métodos implementados se concentram na preparação das tarefas e nos controles necessários para respeitar as relações estabelecidas. Apesar de serem implementações diferentes para cada classe, o funcionamento conceitual é semelhante para todos os escalonadores e está descrito no Algoritmo 6.

Deve ser observado que este algoritmo trata apenas de relações entre tarefas. Quando ocorrem relações envolvendo recursos utilizados pelas tarefas, as mesmas são tratadas por um outro método, implementado separadamente. Esse método não é apresentado aqui, mas seu funcionamento é baseado nas listas de controle da região crítica e seus semáforos.

5.4 Considerações finais

Este capítulo abordou a especificação de como a metodologia proposta pode ser implementada. Em particular foram apresentadas as técnicas para a execução das atividades de modelagem de cada tipo de relação entre tarefas e como as mesmas foram transformadas em componentes do RTsim. Com as descrições de algoritmos e classes realizadas aqui espera-se que a metodologia proposta tenha ficado mais clara, assim como seja possível instanciá-la para outros simuladores de sistemas concorrentes.

Algoritmo 6: Algoritmo para preparação do escalonamento

```
1  if Há tarefas para escalonar then
2  |  Lê as relações entre tarefas recebidas;
3  |  Estabelece as relações indicadas pelo usuário;
4  |  if Há relação do tipo precedência then
5  |  |  Prepara as tarefas precedidas para serem bloqueadas até a finalização de
   |  |   suas precedentes;
   |  |  /* Em caso de precedência com passagem de informação haverá
   |  |     sinalização de envio após o término da precedente          */
6  |  end
7  |  if Há relação de independência de ordem then
8  |  |  Indica as tarefas como independentes;
9  |  |  Prepara as tarefas para serem bloqueadas quando uma delas iniciar a
   |  |   execução;
10 |  end
11 |  if Há relação de paralelismo then
12 |  |  Indica as tarefas como independentes;
13 |  |  Em caso de existir mais de um processador e haver possibilidade de migrar
   |  |   tarefas realize a migração;
14 |  end
15 |  if Há relação de prioridade then
16 |  |  Indica as tarefas como independentes;
17 |  end
18 |  while Lista não está completamente ordenada do
19 |  |  Ordena as tarefas independentes pela prioridade estabelecida na entrada;
   |  |  /* Não considerar as tarefas bloqueadas nesta lista           */
20 |  end
21 end
```

Capítulo 6

Validação da Metodologia e Testes de Usabilidade

A validação deste trabalho ocorreu a partir de testes realizados após a implementação da metodologia proposta no RTsim. Os testes realizados para validar a metodologia procuram verificar tanto sua corretude quanto sua usabilidade. Ao longo deste capítulo serão apresentadas as condições em que os testes foram realizados e os resultados obtidos.

6.1 Avaliação de corretude

A avaliação de corretude da metodologia proposta contempla a verificação de como cada relação é modelada e se a sua simulação corresponde ao previsto. Desse modo, os testes a seguir apresentam um modelo com uma determinada dependência e os resultados obtidos com sua simulação no RTsim. Deve ser observado que o caso tratado na seção 6.1.4 apresenta uma situação bastante conhecida, que é o problema do jantar dos filósofos (SILBERSCHATZ; GALVIN; GAGNE, 2014), o que deve facilitar o entendimento do processo de modelagem utilizado.

6.1.1 Avaliação da relação de precedência

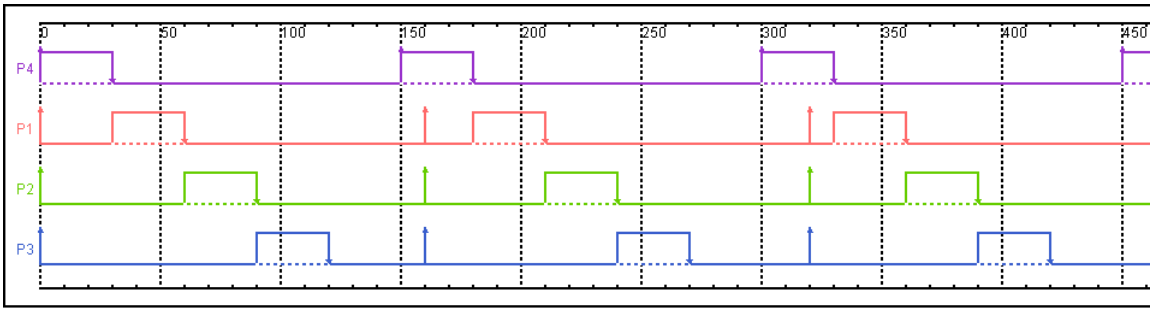
Para avaliar o impacto de relações de precedência será utilizado um conjunto de tarefas que será escalonado usando a política Taxa Monotônica. Os parâmetros básicos dessas tarefas são apresentados na Tabela 6.1. Usando o Taxa Monotônica, a tarefa com maior prioridade é aquela com menor valor de período (P4), tendo as demais tarefas prioridade equivalente. O escalonamento simulado para esse conjunto de tarefas tem o comportamento exibido pela Figura 6.1.

Com os valores de períodos iguais, as tarefas P1, P2 e P3 seguem o escalonamento por ordem de entrada na fila de execução. Este mesmo conjunto foi selecionado para ser

Tabela 6.1: Conjunto de tarefas periódicas

Tarefa	Período/Deadline	Carga	Chegada
P1	160	30	0
P2	160	30	0
P3	160	30	0
P4	150	30	0

Figura 6.1: Escalonamento produzido com o algoritmo Taxa Monotônica

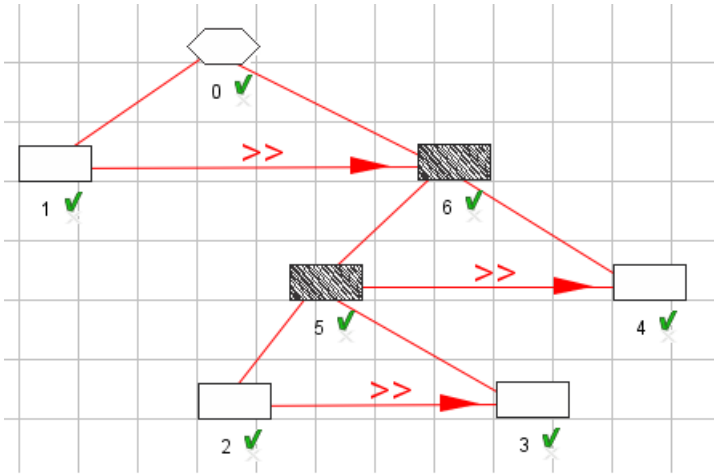


(Gerada pelo autor)

modelado pela interface e receber relacionamentos de precedência, verificando assim as diferenças geradas com a adição dos métodos.

Na Figura 6.2 exibe-se o conjunto de tarefas modelado pela interface. É possível perceber que as relações de precedência foram colocadas de forma a criar uma sequência de ações, sendo executadas respectivamente P1, P2, P3 e P4, ainda que P4 fosse mais prioritária seguindo apenas a política do Taxa Monotônica. Este exemplo foi projetado com o propósito de ilustrar a diferença de comportamento obtida com a adição das interações.

Figura 6.2: Modelo com interações de precedência entre as tarefas

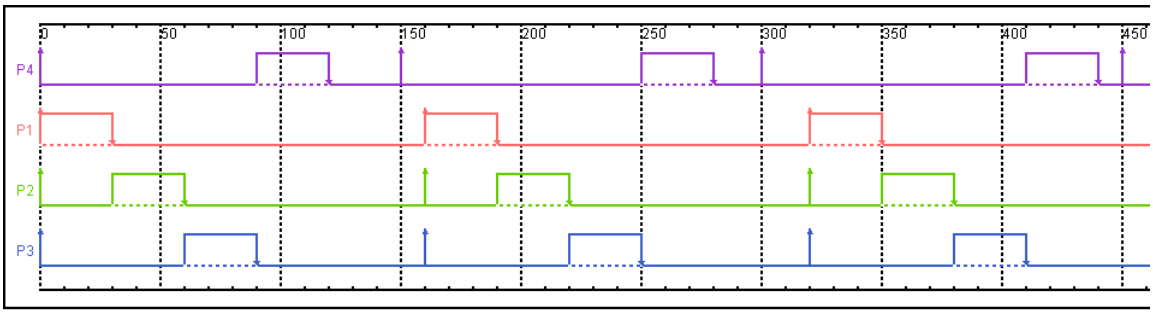


(Gerada pelo autor)

O escalonamento obtido com o modelo de precedência é ilustrado na Figura 6.3. É possível observar, já nos instantes iniciais, o respeito ‘a ordem de precedência, sendo P1 a primeira tarefa a iniciar sua execução. No instante T=150, a tarefa P4 não inicia

sua execução porque é feita uma análise por períodos de simulação, verificando-se que dentro deste determinado período haverá execuções de suas precedentes, ocasionando seu bloqueio.

Figura 6.3: Modelo com interações de precedência entre as tarefas



(Gerada pelo autor)

Todos os métodos de bloqueios e liberação apresentados para a relação de precedência foram utilizados para gerar o escalonamento final. Este tipo de controle permite obter resultados como o da Figura 6.3, que antes não era possível de ser obtido com a implementação original da ferramenta.

6.1.2 Avaliação da relação de precedência com passagem de informação

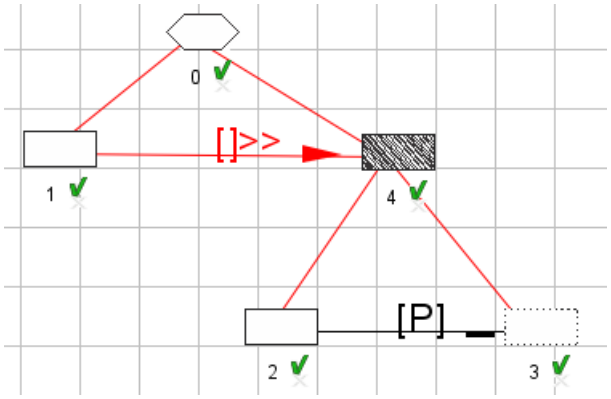
O conjunto de tarefas exibido na Tabela 6.2, composto por tarefas periódicas e aperiódicas, foi escolhido para ilustrar a simulação de precedência com um outro algoritmo do RTsim, o Servidor por Deferência. Na Figura 6.4 é exibido o modelo de tarefas com a relação já estabelecida. Para este exemplo, a relação de prioridade definida serve apenas para considerar as duas tarefas do grupo sem relacionamento entre si, de forma que a decisão por qual delas escalonar leva em consideração apenas a prioridade estabelecida pelo algoritmo.

Tabela 6.2: Conjunto de tarefas periódicas e aperiódicas

Tarefa	Período	Deadline	Carga
P1	100	100	10
P2	100	100	10
Tarefa	Ocorrência	Deadline	Carga
A3	0	190	30

Este conjunto de tarefas, seguindo a execução tradicional do algoritmo, iniciaria a execução da tarefa aperiódica A3 devido a sua ocorrência, prevista neste exemplo para o instante 0. Porém, neste modelo criou-se uma situação em que a tarefa aperiódica necessita de alguma informação fornecida por outra tarefa, o que causa o seu bloqueio até a execução completa de P1. Isso muda o comportamento do escalonamento, forçando um comportamento que respeita a relação estabelecida pelo modelo.

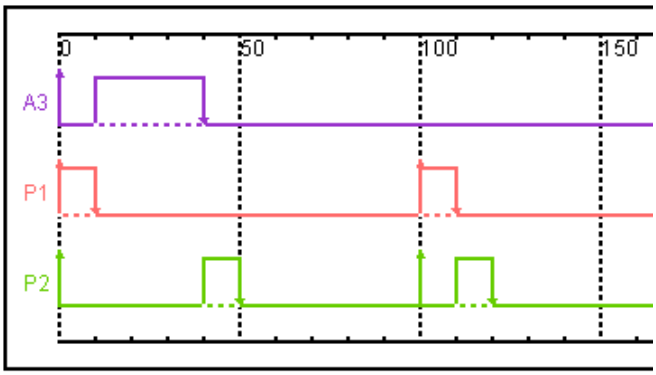
Figura 6.4: Modelagem de tarefas com relação de precedência com passagem de informação



(Gerada pelo autor)

Na Figura 6.5 é exibido o resultado da simulação. Neste trecho é possível observar o bloqueio das duas tarefas, P2 e A3 enquanto há execução de P1 no conjunto. Esta mesma situação não poderia ser configurada pela interface tradicional da ferramenta, pois a ocorrência de A3 logo em $t = 0$ forçaria o início de sua execução com prioridade sobre as demais.

Figura 6.5: Execução no algoritmo Servidor por Deferência



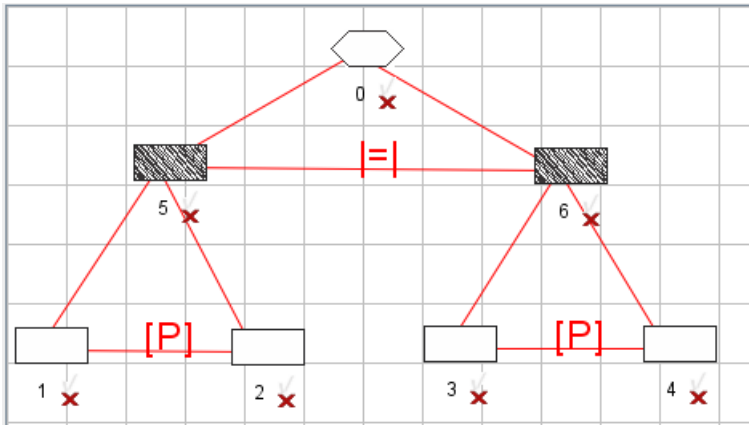
(Gerada pelo autor)

6.1.3 Avaliação da relação de independência de ordem

Para ilustrar o comportamento desta relação se usou o mesmo modelo apresentado no capítulo 5, representado na Figura 6.6 foi configurado para executar seguindo os parâmetros da Tabela 6.3. Neste caso, o exemplo construído foi executado aplicando-se o algoritmo Taxa Monotônica e, para ilustrar a relação em funcionamento, forçou-se a prioridade maior para as tarefas P3 e P4 por meio do valor de período.

Na Figura 6.7 é exibido o resultado da simulação este conjunto. Pelo conceito da relação, os grupos são independentes de ordem entre si, com isso, mesmo que tenha maior

Figura 6.6: Exemplo de modelo construído na interface



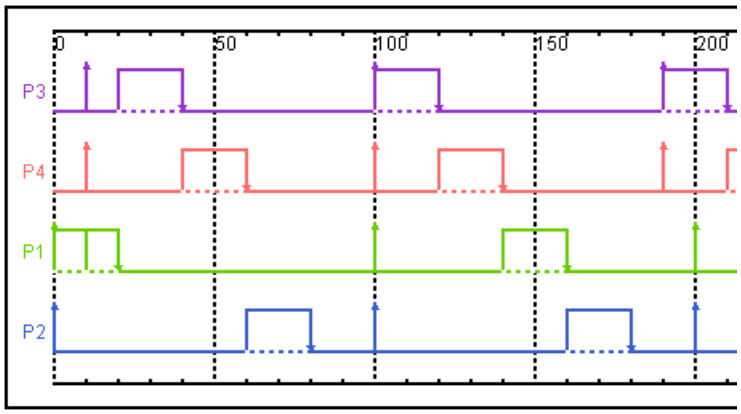
(Gerada pelo autor)

Tabela 6.3: Conjunto de tarefas periódicas para execução

Tarefa	Período	Chegada	Carga
P1	100	0	20
P2	100	0	20
P3	90	10	20
P4	90	10	20

prioridade, a tarefa não poderá executar até a finalização da execução de outra tarefa, permanecendo bloqueada. Como dito anteriormente, a independência de ordem atua sobre a utilização de recursos, sinalizando as tarefas que compartilham recursos e se bloqueiam durante a execução, executando este controle durante a execução do recurso.

Figura 6.7: Exemplo de modelo construído na interface



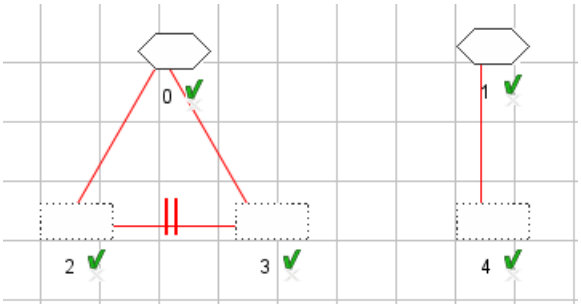
(Gerada pelo autor)

Avaliação da relação de paralelismo

Para ilustrar o comportamento desta relação, na Figura 6.8 exibe-se o modelo adotado para exemplificar este funcionamento. O modelo trabalha com dois processadores para

ilustrar a habilitação da troca de tarefas entre eles. Em particular as tarefas identificadas como 2 e 3 podem ocorrer em paralelo, o que resultará em uma execução no processador 1 se houver necessidade.

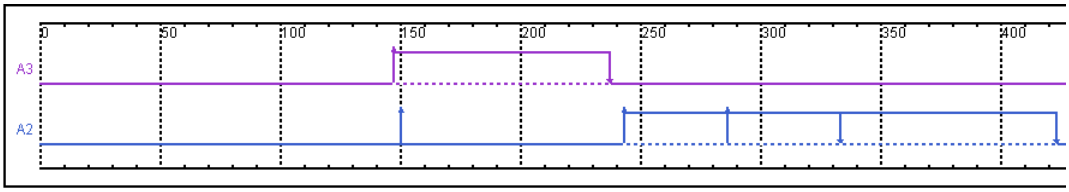
Figura 6.8: Exemplo de modelo construído na interface com relação de paralelo



(Gerada pelo autor)

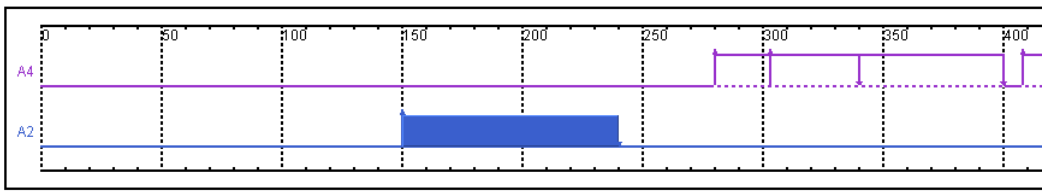
O conjunto projetado de tarefas e seus parâmetros de execução causaram uma sobrecarga no processador 0. Com isso a tarefa A2 foi escolhida pelo algoritmo para executar em outro processador. Nas Figuras 6.9 e 6.10 são exibidos respectivamente o escalonamento nos processadores 0 e 1.

Figura 6.9: Exemplo de execução no algoritmo Leilão (Processador 0)



(Gerada pelo autor)

Figura 6.10: Exemplo de execução no algoritmo Leilão (Processador 1)



(Gerada pelo autor)

No exemplo, é possível observar a tarefa A2 sendo executada ao mesmo tempo que A3, mas em processadores diferentes. Para a migração se respeitou a relação de paralelismo estabelecida pelo modelo, sendo a região sombreada a simbolização de tarefa de outro processador executando.

6.1.4 Avaliação da relação entre recursos

Como citado no Capítulo 2, o conhecido problema do "Jantar dos Filósofos" aborda o compartilhamento de recursos, sendo um caso de estudo para relações de concorrência. Para ilustrar o comportamento das relações entre os recursos do sistema preparou-se um modelo para representação deste problema considerando 4 filósofos com precedência entre seus recursos. Aqui esse problema foi adaptado para que fosse possível observar sua execução no algoritmo Troca de Prioridade.

Durante a modelagem deste problema é necessário considerar os Filósofos como tarefas periódicas, para que possam ser tratados adequadamente pelo algoritmo. Assim, os filósofos são as tarefas P1, P2, P3 e P4, e os garfos são tratados como os recursos R1, R2, R3 e R4 disponíveis. A escolha por tratar apenas quatro filósofos se deu para que a construção do modelo fosse tratada de maneira didática. Para diferentes quantidade de filósofos os modelos são semelhantes, variando-se apenas o tratamento da hierarquia e a composição dos relacionamentos.

Construção do modelo

A definição do problema indica que haverá desativação de tarefas devido aos bloqueios na utilização de recursos. O problema define que a quantidade de recursos disponíveis é igual ao número de filósofos tratados (quatro recursos disponíveis nesta modelagem), sendo que cada tarefa (filósofo) utiliza necessariamente dois deles.

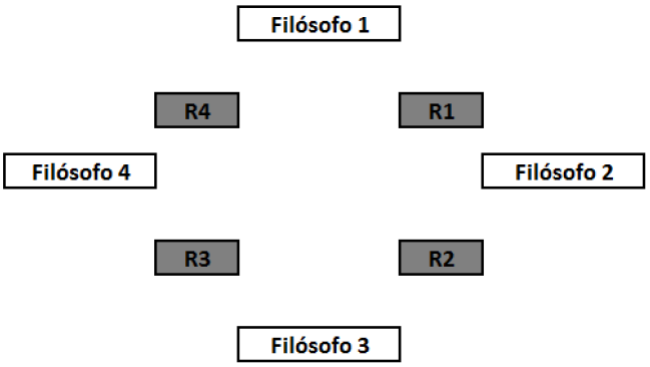
Cada tarefa compartilhará seus recursos com mais outras duas. Com isso, caso uma tarefa se torne ativa, suas dependentes (filósofos sentados imediatamente ao lado) devem ser bloqueadas na utilização dos recursos compartilhados.

Também haverá tarefas que são independentes quanto a execução. Por exemplo, supondo que P1 se torne ativa e inicie a utilização dos seus recursos, as tarefas que compartilham estes recursos não poderão utiliza-los até a liberação (P2 e P3 do modelo). Porém P4, que não depende desses recursos, estará livre para ativação.

Embora uma tarefa use os dois recursos associados ao mesmo tempo, a definição do problema aponta que os recursos são obtidos sequencialmente, em uma ordem de captação restrita. Por este motivo é necessário aplicar o tratamento de precedência entre recursos disponibilizado pela metodologia.

Para exemplificar a situação que será modelada, na Figura 6.11 está representado o posicionamento explicado anteriormente, sendo os filósofos as tarefas e os recursos representados pelas iniciais R. Para a modelagem do problema haverá uma ordem de obtenção dos recursos, sendo requisitado primeiro o recurso posicionado à direita do elemento e depois o recurso à esquerda. Por exemplo, o filósofo 1 (Tarefa 1) fará primeiro a requisição do recurso R4 para, depois de obtido este, requerer R1. Para eliminar problemas de *deadlock* haverá uma troca nessa ordem de captura para um dos filósofos.

Figura 6.11: Exemplo de posicionamento para modelagem



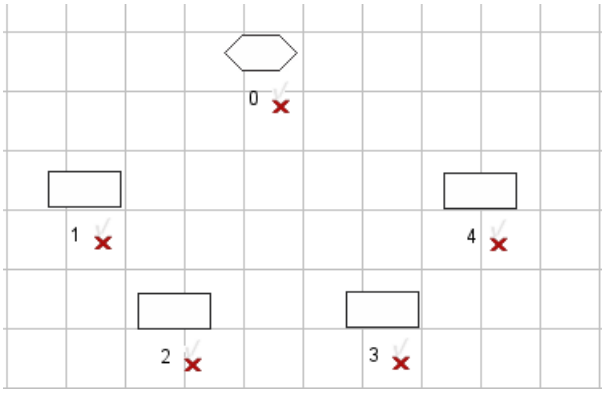
(Gerada pelo autor)

A ordem de requisição dos recursos foi definida da seguinte forma:

1. P1: R4 >> R1;
2. P2: R1 >> R2;
3. P3: R2 >> R3;
4. P4: R4 >> R3;

Para a modelagem do problema serão necessários quatro elementos para representação das tarefas e um elemento de processamento para a construção da árvore hierárquica do modelo (Figura 6.12). Por ser feito em interface gráfica, o posicionamento dos elementos pode ser alterado para se adequar à organização das relações.

Figura 6.12: Elementos para a modelagem

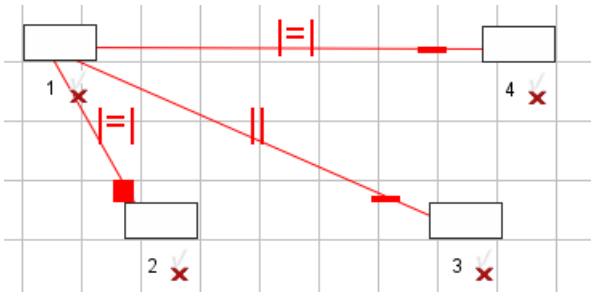


(Gerada pelo autor)

Observando o comportamento das tarefas definido para o problema é possível estabelecer relações de independência de ordem entre as tarefas que compartilham recursos. Desse comportamento se percebe que não há nenhuma prioridade de execução entre os filósofos. Das relações definidas neste documento a representação do paralelismo é a que melhor se adapta ao problema com tarefas independentes.

Com isso é possível observar que uma tarefa terá um relacionamento de independência com as duas tarefas vizinhas, sendo paralela com a tarefa restante. Por exemplo, P1 será independente de ordem de P2 e P4, e paralela a P3. Assim como a tarefa P4 será independente de ordem de P1 e P3, e paralela a P2. Este relacionamento, colocado de forma direta no modelo, tem um resultado como mostrado na Figura 6.13 (desconsiderando o processador e exibindo apenas as relações de P1).

Figura 6.13: Exemplo de estabelecimento de relações



(Gerada pelo autor)

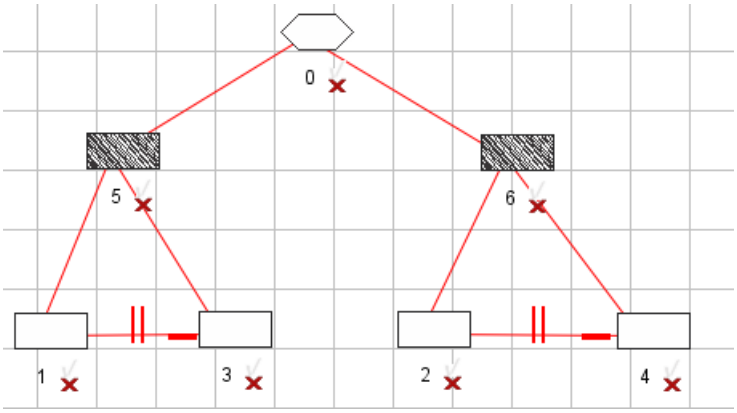
A modelagem hierárquica traz vantagens na exibição deste problema, principalmente para evitar o uso excessivo de arcos. É possível inserir grupos de tarefas nesta configuração, reduzindo graficamente o número de relações estabelecidas. Para isso, é necessário identificar os relacionamentos que formam conjuntos entre as tarefas.

Pela definição do problema, observando-se a Figura 6.11, é possível verificar que as tarefas paralelas não são vizinhas (dependentes), o que define:

$$P1 \parallel P3; \qquad P2 \parallel P4;$$

Organizando os elementos, separados em grupos de tarefas paralelas entre si, é possível obter um modelo como exibido na Figura 6.14. As tarefas internas ao grupo já estão definidas como paralelas, restando apenas o estabelecimento da independência.

Figura 6.14: Exemplo de modelagem com grupos de tarefas



(Gerada pelo autor)

As relações de independência são definidas entre as tarefas como:

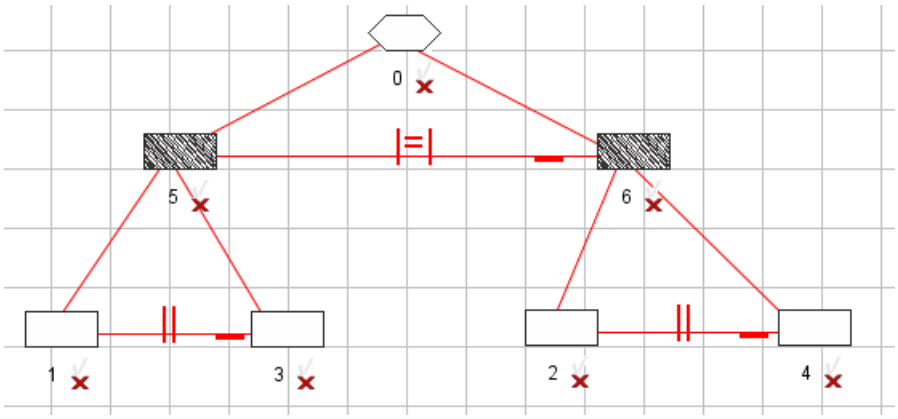
$$P1 \mid=\mid (P2 \wedge P4) \qquad P2 \mid=\mid (P1 \wedge P3)$$

$$P3 \mid=\mid (P2 \wedge P4) \qquad P4 \mid=\mid (P1 \wedge P3)$$

É possível observar que as tarefas paralelas entre si têm o mesmo conjunto de tarefas independentes (exemplo P2 e P4). Com esta informação, pode-se então definir que os grupos de tarefas são independentes entre si, o que se aplicará a sub-árvore de elementos.

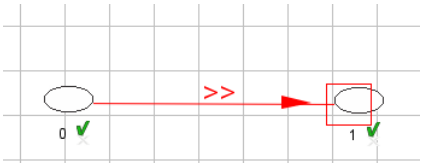
A modelagem adotada para o problema é vista na Figura 6.15. O modelo construído permite observar como a hierarquização fornece uma melhor visualização do comportamento do conjunto. A sub-árvore de cada grupo mantém as tarefas independentes entre si. Dentro da interface de cada tarefa foi configurada a relação de precedência entre os recursos, recém descrita, no formato ilustrado pela Figura 6.16.

Figura 6.15: Exemplo de modelagem para execução com recursos



(Gerada pelo autor)

Figura 6.16: Exemplo de modelagem para recursos na tarefa



(Gerada pelo autor)

Há ainda outras formas de construir o modelo na interface. A escolha por este formato visou utilizar a funcionalidade do elemento Grupo de Tarefas para obter uma simplificação no tratamento da Independência de Ordem.

Simulação na ferramenta

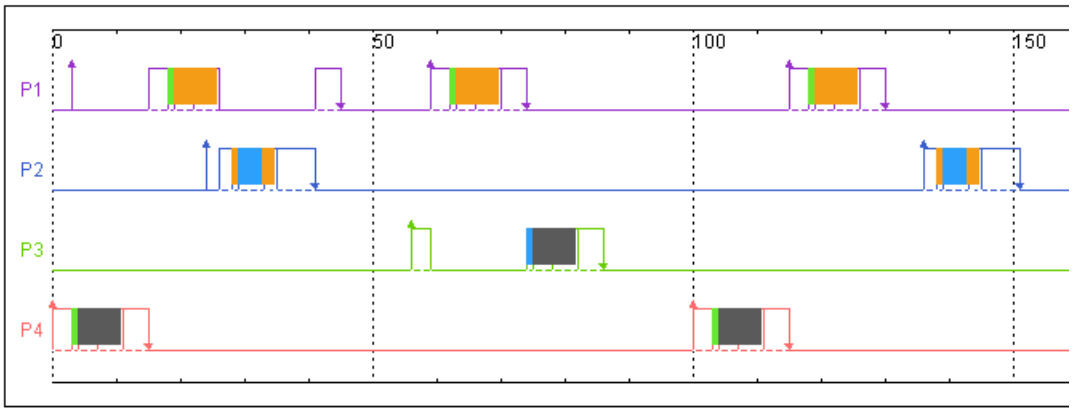
Na caracterização das tarefas adotou-se os seguintes valores para a requisição (entrada na região crítica) de cada recurso: R1=2; R2=3; R3=2; R4=3. Os instantes de entrada na

região crítica foram propositadamente definidos para ilustrar o funcionamento da relação de precedência. Por exemplo, no caso da Tarefa P1 os parâmetros indicam que o primeiro recurso utilizado seria R1. Porém há uma relação de precedência que prevalece entre os recursos, forçando R1 esperar pela obtenção de R4.

O escalonamento das tarefas modeladas, executando o algoritmo Troca de Prioridade, é apresentado na Figura 6.17. Nela é possível acompanhar a obtenção dos recursos pelas tarefas, sendo que a coloração no gráfico indica a posse de um determinado recurso. A correspondência entre recursos e cores é a seguinte:

- R1: laranja;
- R2: azul;
- R3: cinza;
- R4: verde.

Figura 6.17: Exemplo de escalonamento para execução com recursos



(Gerada pelo autor)

Neste exemplo é possível observar que a ordem de captura dos recursos em cada tarefa respeita a relação entre recursos. As áreas coloridas demarcam a utilização do recurso pela tarefa e ajudam a visualização dos eventos. Por exemplo, é possível observar o que foi dito anteriormente sobre a precedência dos recursos em P1 desde a sua primeira execução, próxima ao instante 15 do escalonamento, quando R1 (laranja) só é adquirido depois da obtenção de R4 (verde).

6.2 Testes de usabilidade

Com a avaliação apresentada na seção anterior foi possível perceber que as relações entre tarefas foram corretamente implementadas no simulador. Entretanto, considerando que o objetivo da proposta é fornecer uma metodologia que facilite o processo de modelagem

dessas relações, é importante avaliar como usuários diversos se comportam com seu uso. Desse modo nesta seção serão descritos testes realizados com alunos, visando observar sua interação com o resultado do projeto.

Para avaliar a experiência dos usuários com a interface de modelagem foram escolhidos grupos de estudantes do curso de bacharelado em ciência da computação e do programa de pós-graduação em ciência da computação. Entre os alunos de graduação os testes foram aplicados a alunos de terceiro e quarto ano, garantindo assim algum conhecimento em fundamentos de sistemas operacionais, em especial sobre gerência do processador. Já os alunos do programa de pós-graduação tinham formações diversas, porém com um perfil profissional mais avançado.

Para obter mais clareza na análise dos resultados, os testes foram divididos em três grupos de avaliadores. Como preparação para os testes o primeiro grupo recebeu uma aula introdutória, enquanto os dois outros grupos receberam um material escrito e uma aula de revisão dos conceitos utilizados nos testes. Os grupos foram:

- Grupo 1 - Formado por 31 alunos de graduação matriculados em uma turma da disciplina de "Microcontroladores". Estes já tinham conhecimentos básicos sobre sistemas concorrentes e receberam uma aula introdutória dos conceitos que seriam trabalhados nos testes. Todos os 31 realizaram o teste de modo supervisionado, o que permitiu mais discussão e esclarecimentos sobre a ferramenta.

- Grupo 2 - Composto por 27 alunos de graduação matriculados em uma turma da disciplina de "Linguagens Formais e Autômatos". Este grupo trazia conhecimentos sobre manipulação de diagramas e grafos, e apenas conceitos básicos de sistemas operacionais. O objetivo principal deste teste foi trabalhar com o grupo para ter uma avaliação do fluxo de execução do modelo e seus componentes, uma vez que este é construído como um diagrama.

- Grupo 3 - Formado por 14 alunos de disciplinas de pós-graduação. Com conhecimentos em concorrência e diagramas, além de experiências profissionais e níveis de escolaridade diversos. Este grupo avaliou se a ferramenta é funcional para pessoas do ambiente externo ao acadêmico.

Os testes foram compostos por exercícios que avaliavam três focos principais: interação com a interface de modelagem, vantagens de modelar as tarefas por meio gráfico e compreensão das relações de dependência entre tarefas.

Os formulários incluíram afirmações com cinco alternativas de concordância, seguindo uma escala Likert, sendo: Concordo Totalmente (CT), Concordo Parcialmente (CP), Nem Concordo Nem Discordo (NCND), Discordo Parcialmente (DP) e Discordo Totalmente (DT). Além das respostas para as afirmações foi solicitado para que os avaliadores indicassem pontos positivos e negativos a partir de listas de aspectos considerados importantes,

sem limite de itens a serem votados. Os resultados dessa avaliação são apresentados a seguir.

6.2.1 Avaliação sobre a intuitividade da interface para compor o modelo

Na Tabela 6.4 são exibidas as avaliações dos alunos sobre a experiência com a interface, as quais podem ser visualmente observadas na Figura 6.18. Os resultados apresentados representam a junção das respostas para três afirmações, que foram aglomeradas em virtude de terem basicamente o mesmo foco. As afirmações foram:

1. A interface principal de composição da interação entre as tarefas e elementos do modelo é intuitiva.

2. Os elementos que estão disponíveis na interface do modelo (tarefas, processador, etc.) são fáceis de compreender.

3. Atribuir relações entre tarefas no modelo é um processo simples.

Como as afirmações tratam da intuitividade da interface e seus componentes, é possível entender que com elas se identifica a primeira impressão sobre a interface do modelo. A grande maioria dos avaliadores respondeu de forma positiva, prevalecendo a concordância com as afirmações propostas. Por ser um primeiro contato com este tipo de interface os resultados foram satisfatórios.

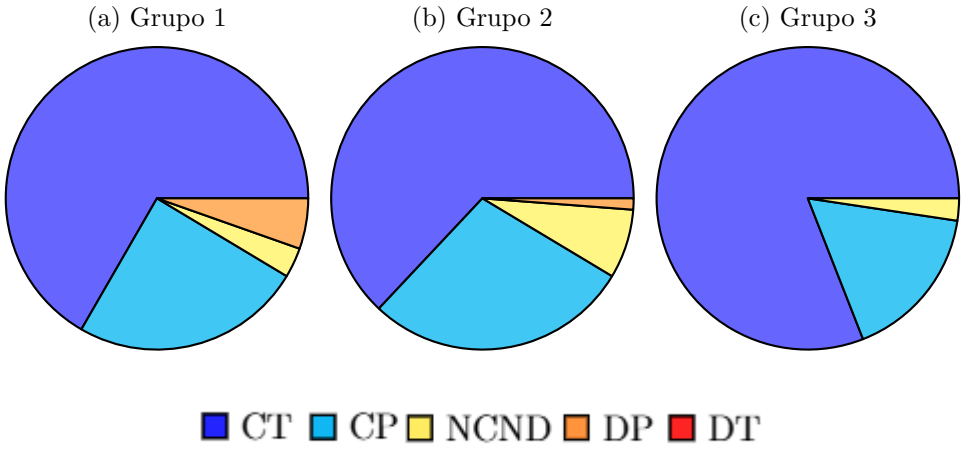
É possível entender também que por não existir contato anterior com as funcionalidades de cada relação, ou com tarefas de tempo-real, houve algumas avaliações de discordância das afirmações. Observe-se que dentre 72 avaliadores, houve apenas 16 respostas (de um total de 216) que indicaram não concordar ou discordar de alguma afirmação. A avaliação dos alunos de pós-graduação, que representam o público mais profissional, se destaca com resultados muito satisfatórios para a ferramenta, não apresentando dificuldades na manipulação da ferramenta e na compreensão dos elementos gráficos.

Tabela 6.4: Avaliação da interface (Valores em Porcentagem)

Grupo	CT	CP	NCND	DP	DT
Grupo 1	66,67 ± 5,48	24,73 ± 1,51	3,20 ± 0	5,40 ± 4,03	0
Grupo 2	63,00 ± 7,99	28,40 ± 6,35	7,40 ± 6,04	1,20 ± 1,74	0
Grupo 3	80,95 ± 12,13	16,68 ± 8,92	2,37 ± 3,35	0	0

A grande maioria dos avaliadores concordou que os elementos disponíveis na interface são fáceis de compreender, o que é um resultado positivo. Os elementos que compõe a interface são parte fundamental para a simulação e devem ser bem compreendidos para permitir a correta construção do modelo. Para reforçar a boa interação com a interface, na avaliação sobre pontos positivos a votação indicou que 74,2% dos alunos do Grupo

Figura 6.18: Gráfico de avaliações sobre a interface e componentes gráficos da ferramenta



(Gerada pelo autor)

1, 88,9% do Grupo 2 e 85,7% do Grupo 3 consideraram a interface gráfica intuitiva com sendo um aspecto positivo para a construção do modelo.

6.2.2 Avaliação da facilidade de interpretação dos problemas modelados

A Tabela 6.5, que sistematiza os gráficos da Figura 6.19, são exibidas as respostas para a seguinte afirmação:

1. Os modelos visuais construídos por meio da ferramenta são de fácil interpretação quando comparados ao entendimento do problema na sua representação textual.

O objetivo desse item foi comparar a sensação dos usuários em relação aos processos de modelar graficamente e modelar formalmente. Destaca-se nos resultados a concordância dos alunos com um dos objetivos do trabalho, que foi tornar o processo de construção de um modelo mais claro ao se fazer o desenho gráfico da situação.

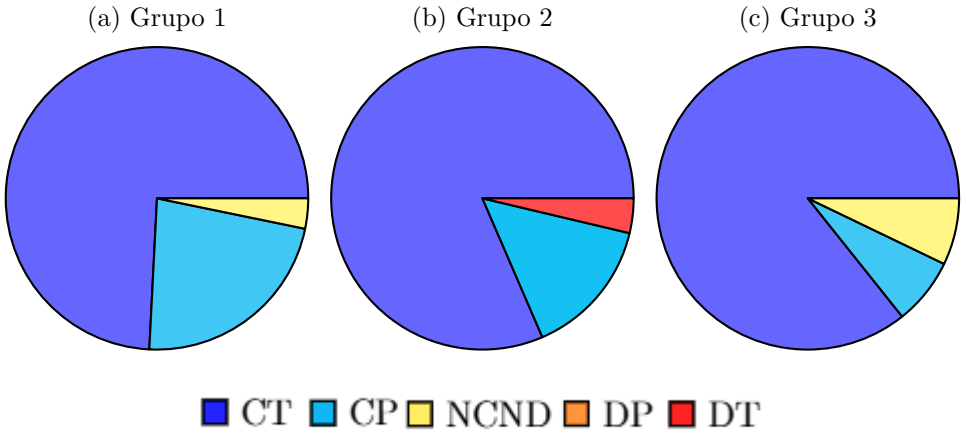
Tabela 6.5: Avaliação da interpretação (Valores em Porcentagem)

Grupo	CT	CP	NCND	DP	DT
Grupo 1	74,19	22,58	3,23	0	0
Grupo 2	81,49	14,81	0	3,70	0
Grupo 3	85,72	7,14	7,14	0	0

O aumento da concordância apresentado pelos Grupos 2 e 3 se deve ao fato de terem trabalhado mais ativamente e com mais contato na modelagem manual de tarefas concorrentes nos testes. Eles receberam como tarefa refazer o problema estudado no modelo proposto, e com isso conseguiram visualizar mais claramente a interpretação do problema.

Para reforçar este resultado, durante a votação em aspectos positivos 87,1% dos alunos do Grupo 1, 96,3% dos alunos do Grupo 2 e 92,8% do Grupo 3 elegeram “Possibilidade

Figura 6.19: Gráfico de avaliações sobre a interpretação do modelo gráfico



(Gerada pelo autor)

de visualizar graficamente as relações entre tarefas" como uma vantagem do modelo apresentado. Isto se deve principalmente a clareza alcançada na visualização das relações em forma de ligações no diagrama, o que permite verificar o comportamento estabelecido.

6.2.3 Avaliação da composição do modelo na interface

Para avaliar a experiência dos alunos ao comporem o modelo gráfico foram apresentadas as seguintes afirmações sobre a composição do modelo na interface gráfica:

1. Compor o modelo na interface gráfica é mais rápido do que escrever em um formato textual específico.

2. Compor o modelo na interface gráfica é mais simples do que escrever em um formato textual específico.

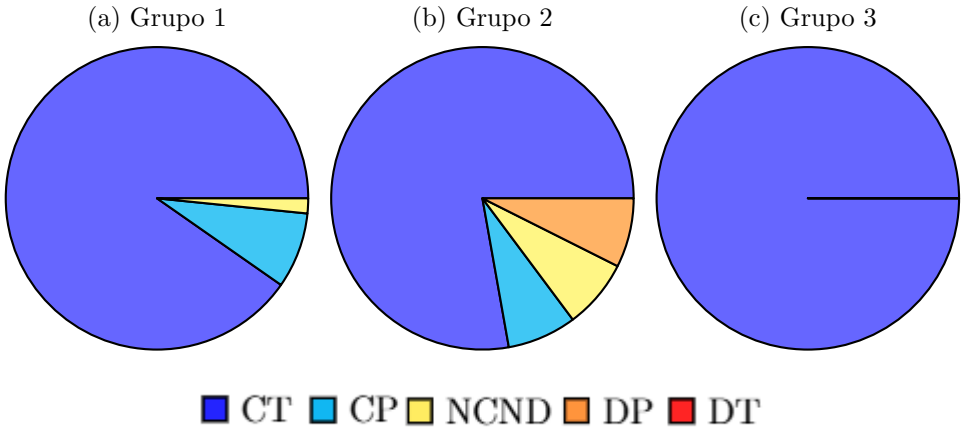
Como um dos objetivos do trabalho é eliminar a necessidade de programação do modelo, como ocorre nos simuladores em que é possível modelar interações, estas respostas refletem a experiência de melhoria alcançada do ponto de vista do usuário. A Tabela 6.6 apresenta a junção das respostas a estas duas afirmações, valores esses que podem ser melhor visualizados nos gráficos apresentados na Figura 6.20.

Tabela 6.6: Avaliação da composição do modelo (Valores em Porcentagem)

Grupo	CT	CP	NCND	DP	DT
Grupo 1	90,30 ± 3,20	8,10 ± 1,60	1,60 ± 1,60	0	0
Grupo 2	77,80 ± 0	7,40 ± 0	7,40 ± 0	7,40 ± 0	0
Grupo 3	100 ± 0	0	0	0	0

Os resultados apresentados na Figura 6.20 são satisfatórios para o trabalho, exibindo grande maioria de concordância com as vantagens da construção do modelo apresentado.

Figura 6.20: Gráfico de avaliações sobre a composição do modelo gráfico



(Gerada pelo autor)

A grande aceitação vista nos Grupos 1 e 3 pode ser justificada pela experiência prévia dos participantes. Os alunos já conheciam os conceitos de concorrência, e, ainda que conceitualmente, já tiveram contato com técnicas de programação destes sistemas.

6.2.4 Avaliação de vantagens na construção do modelo

Outras duas afirmações tratavam de vantagens obtidas na composição do modelo. As afirmações, listadas a seguir, buscavam identificar se a modelagem gráfica auxiliava no entendimento do funcionamento do sistema modelado. As avaliações recebidas são discutidas separadamente.

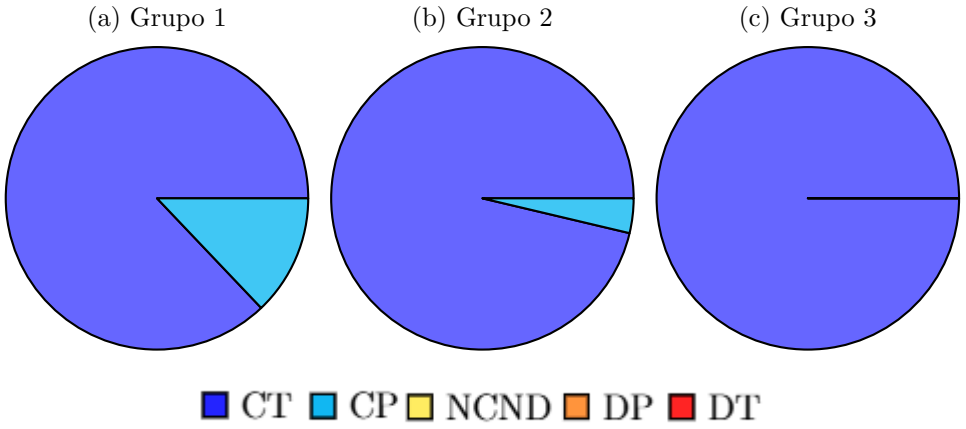
1. Construir o modelo graficamente auxilia na compreensão do problema estudado.
2. O modelo gráfico permite uma visão mais clara do comportamento das tarefas e de suas relações.

Na Figura 6.21 são exibidos os resultados obtidos nas respostas para a primeira afirmação (sistematizados na Tabela 6.7), que tratava da melhora na compreensão do problema estudado quando seu modelo era construído usando a metodologia proposta. Com ela se busca determinar o apoio da ferramenta para uma possível utilização em ensino, mostrando as vantagens de tratar problemas concorrentes graficamente.

Tabela 6.7: Avaliação sobre a compreensão do problema modelado (Valores em Porcentagem)

Grupo	CT	CP	NCND	DP	DT
Grupo 1	87,1	12,9	0	0	0
Grupo 2	96,30	3,70	0	0	0
Grupo 3	100	0	0	0	0

Figura 6.21: Gráfico de avaliações sobre a compreensão do problema



(Gerada pelo autor)

Não houve discordância desta afirmação entre os alunos, destacando-se ainda as taxas maiores de concordância total obtidas nos grupos 2 e 3, que tiveram mais contato com a modelagem do problema manual antes de ver o modelo gráfico correspondente. Este resultado ilustra uma grande vantagem em utilizar a ferramenta no ensino de problemas concorrentes, trazendo mais clareza ao problema estudado quando este é construído a partir de uma interface gráfica.

Já nos gráficos da Figura 6.22 são exibidas as avaliações sobre uma visão mais clara do relacionamento das tarefas a partir do modelo (sistematizadas na Tabela 6.8). Esta avaliação ilustra a vantagem do modelo gráfico para o estudo e análise do comportamento de tarefas, que podem ser vistos na forma de diagrama.

Tabela 6.8: Avaliação da análise de relações (Valores em Porcentagem)

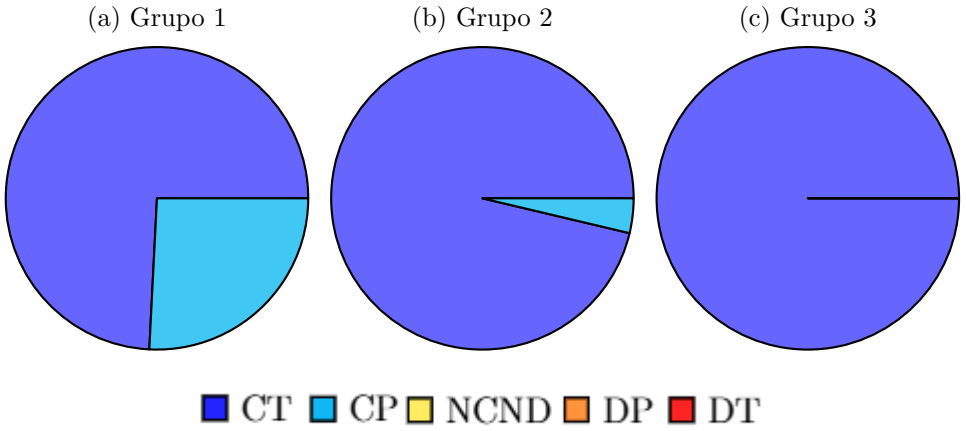
Grupo	CT	CP	NCND	DP	DT
Grupo 1	74,19	25,81	0	0	0
Grupo 2	96,30	3,70	0	0	0
Grupo 3	100	0	0	0	0

Mais uma vez não houve discordâncias com a afirmação. O objetivo da pergunta era ilustrar a capacidade de análise conceitual (uma estimativa) do conjunto de tarefas que pode ser obtida apenas pela sua construção gráfica. De acordo com as respostas a abordagem gráfica torna mais fácil a compreensão do comportamento de tarefas, uma vez que se tem uma visão prévia do fluxo de execução fornecida pelo diagrama do modelo.

6.2.5 Avaliação de pontos positivos e negativos

Como dito anteriormente, o questionário de avaliação tinha duas partes. A primeira com as afirmações já descritas e uma segunda em que os alunos poderiam escolher pontos positivos e negativos da ferramenta fornecidos em uma lista. Os pontos elencados buscavam

Figura 6.22: Gráfico de avaliações sobre a análise de relações do modelo gráfico



(Gerada pelo autor)

compreender a experiência do usuário com o processo de modelagem.

Para essa segunda parte não existia obrigação de resposta nem limite para a escolha dos pontos. Na Tabela 6.9 é exibida a quantidade de indicações feitas como sendo de pontos positivos da metodologia/simulador feitas em cada turma. É possível observar que, mesmo sendo opcional, a participação nesta etapa foi total em todos os grupos.

Tabela 6.9: Respostas do levantamento de aspectos positivos da metodologia e de sua aplicação no RTsim

Pontos positivos	Grupo 1	Grupo 2	Grupo 3
Total de respondentes	31	27	14
Interface gráfica intuitiva.	23	24	12
Visualização gráfica das relações.	27	26	13
Possibilidade de simular o modelo.	28	26	14
Estabelecer relações de forma automática.	14	15	7

Dessa tabela é possível confirmar que o uso de interfaces gráficas de fácil manipulação é considerado como um aspecto positivo da ferramenta. Isso é confirmado pelo elevado número de alunos, dos três grupos, que apontaram os dois primeiros aspectos como sendo positivos. Pode-se observar também que a maior quantidade de avaliações positivas nestes aspectos dentro do Grupo 2 é reflexo da adaptação dos alunos ao trabalho com grafos e autômatos, o que gerou familiaridade com a estrutura adotada pelo modelo.

Outro aspecto importante é que a quase totalidade dos alunos apontou que a possibilidade de simular o modelo era um ponto positivo. Isso reforça o conceito assumido de que a simulação é uma técnica importante para análise de sistemas computacionais. A simulação do modelo permite visualizar o comportamento desenhado para cada tarefa na interface, de forma simples e sem a necessidade de escrita de rotinas em alguma linguagem específica.

Por fim, o estabelecimento de relações de forma automática foi avaliado positivamente

por aproximadamente 50% dos alunos em cada grupo. Embora seja um resultado relativamente baixo, é possível inferir que os alunos não puderam perceber mais profundamente a contribuição de se modelar sem a escrita de códigos exatamente por nunca terem escrito modelos de tarefas antes. Na prática isso significa a ausência de um parâmetro de comparação. Mesmo assim, pelo menos metade dos alunos consideraram a automatização do processo de geração do modelo como sendo uma vantagem da ferramenta.

Apesar da forte indicação de pontos positivos, os alunos também foram questionados sobre eventuais pontos negativos, com alguns deles apontando algo. As respostas com os aspectos negativos apontados são mostradas na Tabela 6.10.

Tabela 6.10: Respostas do levantamento de aspectos negativos da metodologia e de sua aplicação no RTsim

Pontos negativos	Grupo 1	Grupo 2	Grupo 3
Total de respondentes	31	27	14
Interface gráfica não intuitiva.	6	2	1
Falta de outros componentes no modelo.	2	2	2
Dificuldade na composição do modelo.	11	4	3

O número de participantes que avaliaram negativamente o projeto é pequeno em todos os grupos. Em sua maioria, os aspectos negativos apontados podem ser atribuídos em parte ao pouco conhecimento da ferramenta e do processo de modelagem de uma tarefa. As respostas mais negativas vieram do Grupo 1, que apresentou mais dificuldades com a interface. Isso ocorreu principalmente por realizarem os testes com a primeira versão da interface, enquanto os demais grupos receberam uma interface melhorada, contendo dicas e técnicas de apoio para utilização.

Deve ser observado que a falta de conhecimento prévio acaba se refletindo também nas respostas recebidas para a dificuldade de composição do modelo e nas poucas respostas considerando a falta de outros componentes no modelo como ponto negativo. Em particular, é possível compreender que para perceber a ausência de alguma relação ou componente era preciso conhecer mais profundamente como modelar e simular um sistema concorrente (além da teoria envolvida). Um aspecto que deve ser mencionado aqui é que nenhum dos alunos que indicou a deficiência quanto a falta de componentes fez indicação, em campo específico, sobre qual elemento ou relação poderia ser adicionado ao modelo.

6.2.6 Considerações sobre usabilidade

Com base nas avaliações apresentadas nesta seção é possível perceber que a abordagem proposta, bem como suas funcionalidades, foi avaliada positivamente. Os problemas de concorrência tratados nos testes não eram previamente conhecidos pelos alunos, e construí-los na interface proporcionou um melhor entendimento do comportamento das tarefas.

Um outro ponto positivo apontado pelos alunos durante a avaliação é a possibilidade de simular o que foi modelado, obtendo resultados quantitativos (90.3% dos estudantes do Grupo 1, 96,3% do Grupo 2 e 100% do Grupo 3 consideraram esta característica como vantajosa). A integração da abordagem no RTsim tornou a interface ainda mais atrativa para os usuários, principalmente porque essa ferramenta já tinha uma interface gráfica do escalonamento preparada para ser clara e intuitiva. Esta característica, unida com as funcionalidades de modelagem, apoia o entendimento do problema ao permitir visualizar se o modelo construído está de acordo com o esperado.

Poucos estudantes apresentaram dificuldades durante a realização dos testes, porém estas não atrapalharam o objetivo das avaliações. As dificuldades encontradas se concentraram no entendimento das relações existentes em sistemas concorrentes, o que era esperado por ser um primeiro contato com essas características dos sistemas. Também apresentaram inicialmente alguma dificuldade para interpretar a forma textual de apresentação dos problemas que foram modelados, o que foi reduzido após a execução das primeiras modelagens.

6.3 Considerações finais

Os testes aqui descritos permitiram tanto a avaliação da corretude da aplicação da abordagem proposta em um simulador real, quanto a facilidade de seu uso por parte de analistas e desenvolvedores de sistemas concorrentes. De modo geral resultados foram positivos, com uma boa aceitação da metodologia implementada no RTsim e da forma de representação escolhida para as relações de dependência.

Capítulo 7

Conclusões e Trabalhos Futuros

Ao longo do texto desta dissertação se apresentou inicialmente a motivação que justifica um trabalho na linha do que foi realizado. Boa parte da motivação está alicerçada nas revisões sobre sistemas concorrentes e de tempo-real, vistos no Capítulo 2, em conjunto com a revisão sobre simuladores de escalonadores de tempo-real. A justificativa foi complementada com o estudo sobre algumas das abordagens propostas para modelagem de tarefas em sistemas computacionais apresentado no Capítulo 3. O trabalho proposto está descrito e fundamentado ao longo dos Capítulos 4 e 5, que foram seguidos dos testes para a validação da proposta, apresentados no Capítulo 6. Embora algumas conclusões tenham sido mencionadas durante a descrição dos testes, na próxima seção se cuida de apresentar as principais conclusões deste trabalho, ficando para a seção seguinte uma breve discussão sobre trabalhos futuros que podem ser agora vislumbrados.

7.1 Conclusões essenciais

O objetivo principal deste trabalho foi obter uma técnica que permitisse o estabelecimento de relações entre tarefas de forma visual, sem necessidade de codificação. A metodologia proposta neste trabalho permite representar relações fundamentais de sistemas concorrentes de forma gráfica, abordando características genéricas destes sistemas. Além dessas relações foram acrescidas relações específicas de tempo-real, permitindo a sua aplicação em simuladores deste tipo de sistema.

A composição de modelos de tarefas em uma interface gráfica simples e de fácil manipulação permitiu que usuários produzissem modelos corretos mais rapidamente. A construção e modificação de um modelo pode ser facilmente realizada com pequenas ações na interface, uma vez que todos os elementos e relações estão claramente disponíveis para acesso e edição. A interface implementada conseguiu manter uma aparência simples e concisa, com composição totalmente gráfica, sem a necessidade de inserção de códigos de configuração.

A especificação modularizada da metodologia, separando os métodos de composição e interpretação dos elementos na interface trouxe algumas vantagens. A primeira é que com isso se torna possível usar a especificação da composição para aplicar a metodologia em outros simuladores de algoritmos de escalonamento. Basta para tanto especificar como ocorrerá a conversão entre os dados obtidos da interface e os necessários para simulação. Isso pôde ser observado no próprio caso do RTsim, que não tinha estrutura preparada para realizar os tratamentos que foram implementados.

Uma segunda vantagem é facilitar todo o trabalho de implementação de novas relações ou funcionalidades. Ao se tratar separadamente de cada relação e dos impactos que elas trazem no comportamento do sistema, se torna bastante simples o processo de colocar uma nova relação ou mesmo modificar o comportamento de uma já existente.

A metodologia foi avaliada por meio de sua implementação no simulador RTsim. Embora tenha sido necessário diversas modificações no RTsim, a abordagem modular permitiu que ela ocorresse sem maiores problemas. Com o simulador funcionando foi possível verificar se as relações entre tarefas eram obedecidas durante os escalonamentos realizados e se a interface de fato trazia vantagens ao processo de modelagem.

Os testes de corretude mostraram os relacionamentos em ação durante o processo de simulação, com exemplos de casos comuns de execução de tarefas. Todas as situações avaliadas retornaram resultados de acordo com o esperado, com escalonadores mudando seu comportamento original para atender relações de dependência entre tarefas.

O teste de usabilidade, envolvendo o uso da metodologia, e do RTsim, por grupos variados de estudantes, também foi bastante positivo. Os resultados obtidos mostram uma boa adaptação e aceitação da metodologia por pessoas com diferentes níveis de conhecimento prévio. Ainda que os participantes não sejam atuantes em projetos de sistemas concorrente, eles conseguiram utilizar e compreender com facilidade a interface de modelagem e avaliaram bastante bem a ferramenta, com mais de 70% considerando a interface e o processo de modelagem intuitivos.

Logo é possível concluir que a metodologia visual para modelagem de interações entre tarefas aqui proposta conseguiu atingir plenamente os objetivos de facilitar o processo de modelagem, sem perda da precisão dos resultados simulados. O fato de modelar um sistema sem preocupação com adaptá-lo a uma linguagem de programação específica, permite manter o foco no objetivo principal do trabalho, facilitando a atuação dos usuários finais e agilizando a produção de resultado.

Além das vantagens para a realização de modelagem dos sistemas, um resultado adicional, porém bastante relevante, obtido foi o de aumentar as funcionalidades providas pelo RTsim. A nova capacidade de melhor modelar tarefas deve tornar a ferramenta mais atrativa em seu campo de utilização.

7.2 Trabalhos futuros

O trabalho desenvolvido trouxe um tratamento diferenciado para tarefas e suas relações, principalmente quando comparado aos trabalhos existentes na área. Como todo trabalho de pesquisa, dele também surgem novos problemas a serem resolvidos. Alguns deles envolvem melhorias nas interfaces gráficas do RTsim, porém estas melhorias não serão tratadas aqui por não estarem diretamente relacionadas ao processo de modelagem. Na linha de melhorias diretamente relacionadas com a metodologia proposta podem ser destacados os seguintes aspectos:

- Desenvolvimento de novos relacionamentos entre tarefas para outras áreas de aplicação;
- Aprimoramento da separação (modularização) entre as especificações das regras de relação, da interface e de sua implementação;
- Aplicação de técnicas de agrupamento de tarefas no modelo.

Embora sistemas concorrentes apresentem uma grande quantidade de relações entre tarefas, existem outras áreas da computação que apresentam uma ou outra relação que implica em comportamento diferenciado. Um destes casos envolve a modelagem de transações em bancos de dados, em que a execução de "commits" demanda um comportamento bastante distinto. Desta forma, o incremento de interações diversas nesta metodologia torna-a mais completa e adequada para diferentes finalidades.

Por outro lado, a modularização do projeto dos diversos elementos envolvidos na metodologia proposta permite sua implementação em outros simuladores sem alterações excessivas em sua implementação original. Como foi descrito no trabalho, os diversos métodos de interpretação e manipulação das interações foram criados de forma modularizada, porém é possível ampliar essa capacidade se forem criados módulos intermediários. Esses módulos intermediários seriam responsáveis por fornecer as interfaces gráficas para desenho das relações e interfaces de conversão entre as formas de representação de dados de cada simulador.

Por fim, embora o uso da relação "grupo de tarefas" permitir uma certa simplificação no processo de modelagem, a mesma é insuficiente para modelos mais complexos, ou modelos em que se queira desconsiderar alguma relação. Isso pode ser feito com a introdução de mecanismos para agrupamento no modelo. O agrupamento deixaria a atividade de modelagem mais confortável e poderia ser implementada aplicando técnicas já existentes na modelagem de processos de engenharia de software.

7.3 Publicações

O desenvolvimento deste trabalho é consequência de atividades desenvolvidas pela autora no grupo de pesquisa. Dessas atividades resultou o seguinte trabalho:

Peronaglio, F. F. et al., Modeling real-time schedulers for use in simulations through a graphical interface. In: *Proceedings of the 50th Annual Simulation Symposium*. Virginia Beach, USA. (ANSS'17), p10:1-10:12. 2017.

Os resultados do trabalho aqui descrito estão em artigo submetido ao PDCAT 2019 (20th Conf. on Parallel and Distributed Computing, Applications and Technologies), a ser realizado em Gold Coast, Australia, em dezembro de 2019. O título do trabalho é:

Modeling and simulation of tasks interactions using graphical interfaces.

Referências

AUDSLEY, N. et al. Stress: A simulator for hard real-time systems. *Software-Practice and Experience*, p. 543–564, 1994. Citado na página 30.

BACON, J. *Concurrent Systems - Operating Systems, Database and Distributed Systems: An integrated Approach*. [S.l.]: Addison-Wesley, 1998. Citado na página 16.

BOLOGNESI, T.; BRINKSMA, E. Introduction to the iso specification language lotos. *Computer Networks and ISDN systems*, Elsevier, v. 14, n. 1, p. 25–59, 1987. Citado na página 47.

BREIVOLD, H. P. et al. Using dependency model to support software architecture evolution. In: *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2008. (ASE'08), p. III–82–III–91. ISBN 978-1-4244-2776-5. Disponível em: ⟨https://doi.org/10.1109/ASEW.2008.4686324⟩. Citado 2 vezes nas páginas 41 e 42.

CAO, X.-R.; HO, Y.-C. Models of discrete event dynamic systems. *IEEE Control Systems Magazine*, IEEE, v. 10, n. 4, p. 69–76, 1990. Citado na página 38.

CASSANDRAS, C. G.; LAFORTUNE, S. *Introduction to discrete event systems*. [S.l.]: Springer Science & Business Media, 2009. Citado na página 38.

CHANDARLI, Y. et al. *YARTISS: A Generic, Modular and Energy-Aware Scheduling Simulator for Real-Time Multiprocessor Systems*. Tese (Doutorado) — UPE LIGM ESIEE, 2014. Citado na página 31.

CHATURVEDI, D. K. *Modeling and simulation of systems using MATLAB and Simulink*. [S.l.]: CRC press, 2017. Citado 3 vezes nas páginas viii, 37 e 38.

CHEN, Z. et al. An approach to analyzing dependency of concurrent programs. In: IEEE. *Proceedings First Asia-Pacific Conference on Quality Software*. [S.l.], 2000. p. 39–43. Citado na página 39.

CH'ERAMY, M.; D'EPLANCHE, A.-M.; HLADIK, P.-E. Simulation of real-time multiprocessor scheduling with overheads. In: *Intl Conf on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2013)*. [S.l.: s.n.], 2013. p. 5–14. Citado na página 31.

CLARK, A.; EVANS, A. Foundations of the Unified Modeling Language. In: SPRINGER. *Proceedigs of the 2nd Northern Formal Methods Workshop*. [S.l.], 1997. Citado na página 41.

DIAZ, A.; BATISTA, R.; CASTRO, O. Realtss: a real-time scheduling simulator. In: IEEE. *Electrical and Electronics Engineering, 2007. ICEEE 2007. 4th International Conference on*. [S.l.], 2007. p. 165–168. Citado na página 30.

DIJKSTRA, E. W. Over de sequentialiteit van procesbeschrijvingen. Circulated privately. 196_. Disponível em: ⟨http://www.cs.utexas.edu/users/EWD/ew00xx/EWD35.PDF⟩. Citado na página 20.

DIJKSTRA, E. W. Hierarchical ordering of sequential processes. In: _____. *The Origin of Concurrent Programming: From Semaphores to Remote Procedure Calls*. New York, NY: Springer New York, 2002. p. 198–227. ISBN 978-1-4757-3472-0. Disponível em: ⟨https://doi.org/10.1007/978-1-4757-3472-0_5⟩. Citado na página 59.

DSM. DSM model. Disponível em <https://dsmweb.org>, acessado em 09 de julho de 2019. 2019. Citado na página 41.

EPPINGER, S. D.; BROWNING, T. R. *Design Structure Matrix Methods and Applications*. [S.l.]: MIT Press, 2016. Citado na página 41.

FARINES, J.-M.; FRAGA, J. D. S.; OLIVEIRA, R. D. Sistemas de tempo real. *Escola de Computação*, v. 2000, p. 201, 2000. Citado 8 vezes nas páginas viii, 14, 17, 25, 26, 27, 52 e 55.

FERRANTE, J.; OTTENSTEIN, K. J.; WARREN, J. D. The program dependence graph and its use in optimization. *ACM Trans. Program. Lang. Syst.*, ACM, New York, NY, USA, v. 9, n. 3, p. 319–349, jul. 1987. ISSN 0164-0925. Disponível em: ⟨http://doi.acm.org/10.1145/24039.24041⟩. Citado na página 40.

GONÇALVES, T. *RTsim 2.0 - Reengenharia e Novas Funcionalidades*. 2005. Monografia de Projeto Final de curso, UNESP. Citado na página 32.

GSPD. Gspd's homepage. Disponível em <http://www.dcce.ibilce.unesp.br/spd/>, acessado em 15 de junho de 2016. 2016. Citado na página 32.

HAREL, D. Statecharts: A visual formalism for complex systems. *Science of computer programming*, Elsevier, v. 8, n. 3, p. 231–274, 1987. Citado na página 40.

HOARE, C. A. R. Monitors: An operating system structuring concept. *Commun. ACM*, ACM, New York, NY, USA, v. 17, n. 10, p. 549–557, out. 1974. ISSN 0001-0782. Disponível em: ⟨http://doi.acm.org/10.1145/355620.361161⟩. Citado na página 21.

IVKOVIC, I.; KONTOGIANNIS, K. Towards automatic establishment of model dependencies using formal concept analysis. *International Journal of Software Engineering and Knowledge Engineering*, World Scientific, v. 16, n. 04, p. 499–522, 2006. Citado na página 43.

JENSEN, K. *Coloured Petri nets: basic concepts, analysis methods and practical use*. [S.l.]: Springer Science & Business Media, 2013. v. 1. Citado na página 45.

JONES, M. What really happened on mars rover pathfinder. *The Risks Digest*, v. 19, n. 49, p. 1–2, 1997. Citado na página 27.

KEHDY, R. *Simulação de Algoritmos de Escalonamento de Tarefas de Tempo-Real em Sistemas Distribuídos*. 1999. Monografia de Projeto Final de curso, UNESP. Citado na página 32.

KöHLER, H. J. et al. Integrating uml diagrams for production control systems. In: *Proceedings of the 22Nd International Conference on Software Engineering*. New York, NY, USA: ACM, 2000. (ICSE '00), p. 241–251. ISBN 1-58113-206-9. Disponível em: ⟨http://doi.acm.org/10.1145/337180.337207⟩. Citado na página 41.

LEUNG, J. Y.-T. A new algorithm for scheduling periodic, real-time tasks. *Algorithmica*, Springer, v. 4, n. 1-4, p. 209–219, 1989. Citado na página 28.

LEUNG, J. Y.-T.; WHITEHEAD, J. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation*, Elsevier, v. 2, n. 4, p. 237–250, 1982. Citado na página 28.

LIU, C. L.; LAYLAND, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, ACM, v. 20, n. 1, p. 46–61, 1973. Citado na página 28.

LIU, J. W. S. *Real-Time Systems*. [S.l.]: Prentice Hall, 2000. Citado na página 18.

LOYALL, J. P.; MATHISEN, S. A. Using dependence analysis to support the software maintenance process. In: IEEE. *1993 Conference on Software Maintenance*. [S.l.], 1993. p. 282–291. Citado na página 44.

MANACERO JR., A.; MIOLA, M.; NABUCO, V. Teaching real-time with a scheduler simulator. In: *Proc. of the 31st Frontiers in Education Conference*. [S.l.]: IEEE Press, 2001. p. T4D15–19. Citado 2 vezes nas páginas 15 e 32.

MATSUBARA, Y. et al. An open-source flexible scheduling simulator for real-time applications. In: IEEE. *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2012 IEEE 15th International Symposium on*. [S.l.], 2012. p. 16–22. Citado 2 vezes nas páginas 34 e 35.

MAZIERO, C. *Sistemas Operacionais: Conceitos e Mecanismos*. [S.l.]: UFPR, 2014. Citado 5 vezes nas páginas 14, 20, 21, 24 e 27.

MIOLA, M. *Reengenharia aplicada ao RTsim*. 2001. Monografia de Projeto Final de curso, UNESP. Citado na página 32.

MORI, G.; PATERNÒ, F.; SANTORO, C. CTTE: support for developing and analyzing task models for interactive system design. *IEEE Transactions on software engineering*, IEEE, v. 28, n. 8, p. 797–813, 2002. Citado 2 vezes nas páginas 47 e 48.

MURPHY, G. C.; NOTKIN, D.; SULLIVAN, K. J. Software reflexion models: Bridging the gap between design and implementation. *IEEE Transactions on Software Engineering*, IEEE, v. 27, n. 4, p. 364–380, 2001. Citado 3 vezes nas páginas viii, 42 e 43.

MURRAY, R. M. Recent research in cooperative control of multivehicle systems. *Journal of Dynamic Systems, Measurement, and Control*, American Society of Mechanical Engineers, v. 129, n. 5, p. 571–583, 2007. Citado na página 13.

NETO, J. A. N.; TURNELL, M. D. F. Q. V.; SANTONI, C. Modelagem da rotina operacional de uma subestação elétrica em hcpn. *VIII SBAI - Simposio Brasileiro de Automacao inteligente*, 2007. Citado 2 vezes nas páginas 44 e 45.

NIKOLIC, B.; AWAN, M. A.; PETTERS, S. M. Sparts: Simulator for power aware and real-time systems. In: IEEE. *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*. [S.l.], 2011. p. 999–1004. Citado na página 30.

NISSANKE, N. *Realtime Systems*. [S.l.]: Prentice Hall, 1997. Citado 2 vezes nas páginas 25 e 26.

PATERNÒ, F.; MANCINI, C.; MENICONI, S. ConcurTaskTrees: A diagrammatic notation for specifying task models. In: SPRINGER. *Human-Computer Interaction INTERACT'97*. [S.l.], 1997. p. 362–369. Citado na página 47.

PERONAGLIO, F. F. *Simulação de Tempo Real: Implementação do método para escalonamento manual na ferramenta RTsim*. 2017. Monografia de Trabalho de conclusão de curso, UNESP. Citado na página 33.

PERONAGLIO, F. F. et al. Modeling real-time schedulers for use in simulations through a graphical interface. In: *Proceedings of the 50th Annual Simulation Symposium*. Virginia Beach, VI, USA: Society for Computer Simulation International, 2017. (ANSS '17), p. 10:1–10:12. Citado na página 33.

PETRI, C. A. *Communication with automata*. Tese (Doutorado) — Universität Hamburg, 1966. Citado na página 44.

RAMAMRITHAM, K.; STANKOVIC, J. A.; SHIAH, P.-F. Efficient scheduling algorithms for real-time multiprocessor systems. *Parallel and Distributed Systems, IEEE Transactions on*, IEEE, v. 1, n. 2, p. 184–194, 1990. Citado 2 vezes nas páginas 28 e 29.

SANGAL, N. et al. Using dependency models to manage complex software architecture. In: ACM. *ACM Sigplan Notices*. [S.l.], 2005. v. 40, n. 10, p. 167–176. Citado na página 42.

SHIN, K. G.; RAMANATHAN, P. Real-time computing: A new discipline of computer science and engineering. *Proceedings of the IEEE*, Citeseer, v. 82, n. 1, p. 6–24, 1994. Citado na página 25.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Operating system concepts essentials*. [S.l.]: John Wiley & Sons, Inc., 2014. Citado 6 vezes nas páginas viii, 17, 19, 20, 22 e 94.

SPRUNT, B.; LEHOCZKY, J.; SHA, L. Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm. In: IEEE. *Real-Time Systems Symposium, 1988., Proceedings.* [S.l.], 1988. p. 251–258. Citado 2 vezes nas páginas 28 e 29.

STROSNIDER, J. K.; LEHOCZKY, J. P.; SHA, L. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, IEEE, v. 44, n. 1, p. 73–91, 1995. Citado na página 28.

TAVARES, E.; MACIEL, P.; SILVA, B. Modeling hard real-time systems considering inter-task relations, dynamic voltage scaling and overheads. *Microprocessors and Microsystems*, Elsevier, v. 32, n. 8, p. 460–473, 2008. Citado na página 46.

VASILACHE, S.; TANAKA, J. Bridging the gap between analysis and design using dependency diagrams. In: IEEE. *Third ACIS Int'l Conference on Software Engineering Research, Management and Applications (SERA'05)*. [S.l.], 2005. p. 407–414. Citado 2 vezes nas páginas 39 e 40.

WHITE, S.; MIERS, D. *BPMN Modeling and Reference Guide: Understanding and Using BPMN*. [S.l.]: Future Strategies Incorporated, 2008. (Business Process Management Process Modeling). ISBN 9780977752720. Citado na página 54.

YAASHUWANTH, C.; RAMESH, R. Web-enabled framework for real-time scheduler simulator (a teaching tool). In: IEEE. *Computer Research and Development, 2010 Second International Conference on*. [S.l.], 2010. p. 826–830. Citado na página 31.

unesp Campus de São José do Rio Preto

TERMO DE REPRODUÇÃO XEROGRÁFICA

Autorizo a reprodução xerográfica do presente Trabalho de Conclusão, na íntegra ou em partes, para fins de pesquisa.

São José do Rio Preto, 03 / 09 / 19

Fernanda F. Peronglio

Assinatura do autor