Augusto Garcia Fabbri

Modelagem e Simulação das Atividades da Arquitetura Empresarial Utilizando um Sistema Multiprocessado Não Preemptivo

Jataí-Goiás

2019

Augusto Garcia Fabbri

Modelagem e Simulação das Atividades da Arquitetura Empresarial Utilizando um Sistema Multiprocessado Não Preemptivo

Monografia apresentada

Universidade Federal de Jataí

Instituto de Ciências Exatas e Tecnológicas (ICET)

Bacharelado em Ciências da Computação

Orientador: Joslaine Cristina Jeske de Freitas

Jataí-Goiás

2019

Augusto Garcia Fabbri

Modelagem e Simulação das Atividades da Arquitetura Empresarial Utilizando um Sistema Multiprocessado Não Preemptivo/ Augusto Garcia Fabbri. – Jataí-Goiás, 2019-

52 p. : il. (algumas color.) ; 30 cm.

Orientador: Joslaine Cristina Jeske de Freitas

Monografia (Graduação) – Universidade Federal de Jataí
Instituto de Ciências Exatas e Tecnológicas (ICET)
Bacharelado em Ciências da Computação, 2019.

1. Palavra-Chave1. 2. Palavra-Chave2.

Augusto Garcia Fabbri

Modelagem e Simulação das Atividades da Arquitetura Empresarial Utilizando um Sistema Multiprocessado Não Preemptivo

Monografia apresentada

Trabalho aprovado. Jataí-Goiás, data da defesa: 07/06/2019

Joslaine Cristina Jeske de Freitas
Orientador

Franciny Medeiros Barreto
Avaliador

Márcio Moraes Lopes
Avaliador

Jataí-Goiás
2019

Este trabalho é dedicado à ...

AGRADECIMENTOS

Agradeço à ...

“Epígrafe.
(Autor da Epígrafe)

RESUMO

Resumo na língua vernácula, elaborado em folha separada, deve indicar, concisamente, os pontos relevantes do trabalho: objeto de estudo, problema, tema objetivos, justificativas, metodologia, resultados esperados ou obtidos, o valor científico do trabalho e sua originalidade, contendo, no máximo 400 palavras e no mínimo 200. Deve ser composto de uma sequência de frases concisas e não deve ser elaborado na forma de tópicos. Deve ser seguido das palavras-chave ou descritores, precedido de dois espaços simples, tendo no mínimo de 3 e no máximo de 5 palavras, isto é, inserir palavras que mais representam o conteúdo do trabalho, ressaltando que as mesmas devem vir separadas por ponto. O resumo deve ser em texto corrido e sem parágrafo, digitado em espaço simples.

Palavras-chaves: *PalavraChave1;Palavra-Chave2; Palavra-Chave3.*

ABSTRACT

Abstract (Resumo na língua estrangeira),contém as orientações do resumo em língua estrangeira, digitado em folha separada. Por ser elaborado em idioma de comunicação internacional (inglês – Abstract). Deve, também, ser seguido das palavras-chave (inglês - Keywords). Os critérios de formatação do Abstract seguem os mesmos do Resumo na língua vernácula.

Key-words: *Keyword1; Keyword2; Keyword3.*

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CCM	Como Construir Monografias

SUMÁRIO

1 INTRODUÇÃO

A importância da arquitetura dentro de uma empresa segundo (HOOGERVORST, 2004) remete ao sucesso ou a falha do negócio, onde a integração, capacidade de mudança e a agilidade são de extrema importância para que uma boa estratégia seja definida em que todos seus elementos estejam integrados.

Arquitetura empresarial ou *Enterprise Architecture* (EA) é definida por um dos pioneiros da área, John A. Zachman, como: "conjunto de representações descritivas que são relevantes para a descrição de uma empresa, que podem ser produzidas de acordo com os requisitos da administração e mantidas durante o período de sua vida útil"(ZACHMAN, 1997). Porém não há apenas uma única definição.

Segundo (HOOGERVORST, 2004) EA é a coletividade dos princípios de *desing* que estão associados a quatro perspectivas centrais, formando então os quatro diferentes domínios da EA.

A *Business Architecture* pode ser definida como um conjunto logicamente consistente de princípios e padrões que orientam como um determinado campo do empreendimento será explorado, por exemplo os canais a serem utilizados para entrega de produtos e serviços aos clientes. Já a *Organization Architecture* é o conjunto lógico e consistente de princípios que orientam como as atividades intencionais dentro de um campo particular de esforço são realmente organizadas, como o caso de uma linha de montagem de automóveis, que pode ser organizada através de uma linha de montagem ou então equipes de montagens mais autônomas. A *Information Architecture* diz respeito aos princípios e padrões que orientam como a informação deve ser gerenciada.

O relacionamento dessas perspectivas centrais apresenta que os princípios da *Information Architecture* devem seguir ou estar ligados aos princípios da *Business Architecture* e *Organization Architecture*, e também têm uma forte relação com a *Technology Architecture*, uma vez que depende dela para gerenciar a informação. Os princípios da *Information Architecture* ainda definem limitações e oportunidades pertinentes às áreas de *Business Architecture* e *Organization Architecture*. Já para (LAPALME, 2012), EA é a forma de executar e operar a estratégia global da empresa para manter uma vantagem competitiva.

A EA traz algumas vantagens quando implementada, (BROWN, 2004) apresenta seis delas:

1. Documentação prontamente disponível da empresa.

2. Capacidade de unificar e integrar os dados em toda a empresa e estabelecer ligações

com parceiros externos.

3. Capacidade de unificar e integrar processos de negócios em toda a empresa.
4. Redução do tempo da entrega de uma solução e custos de desenvolvimento, maximizando o uso de modelos empresariais.
5. Maior agilidade nas mudanças do negócio.
6. Capacidade de criar e manter uma visão comum do futuro de forma compartilhada para as áreas de negócios e de TI (Tecnologia da Informação).

Utilizada para produzir resultados pontuais com qualidade e gerenciar mudanças em produtos, a EA é a base de uma corporação para alavancar as inovações tecnológicas. Visando obter um aumento de rentabilidade e melhorias de processos internos a EA passa a ser uma proposta consistente para integrar as atividades corporativas aos sistemas de informação que a compõe.

1.1 Motivação e Justificativas do Trabalho

Na busca por soluções para as empresas evolverem no mercado, este trabalho é motivado pelos avanços tecnológicos e a necessidade das empresas definirem uma grande estratégia corporativa onde todos os elementos (recursos, negócios e organização) estejam alinhados, a fim de se evitar conflitos estruturais durante esse processo. Embasado no trabalho de (PASHAZADEH; NIYARI, 2014), no qual foi proposto um modelo onde os processos e atividades são organizadas em blocos escalonados em apenas um único processador, este trabalho, através da modelagem das atividades utilizando um sistema multiprocessado não preemptivo tem por justificativa trazer os avanços tecnológicos para o ambiente empresarial, utilizando um núcleo de processamento mais poderoso do que no artigo supracitado. Assim, tornando mais simples a introdução da estratégia corporativa em sua cultura e ajudando na execução da mesma.

1.2 Objetivos

Por conseguinte, este trabalho tem como objetivo geral fornecer uma base para modelagem da EA com a finalidade de introduzir a estratégia corporativa na cultura organizacional, possibilitando executá-la de maneira eficaz. Para atingir o objetivo geral, os seguintes objetivos específicos foram definidos:

1. Estudo de redes de Petri e da ferramenta CPN Tools.
2. Modelar o planejamento e execução das atividades da EA.

3. Construir a modelagem das atividades utilizando a ferramenta CPN Tools.
4. Executar as atividades utilizando o sistema multiprocessado contruído em CPN Tools.
5. Analisar os resultados.

2 REFERENCIAL TEÓRICO

Neste capítulo, para melhor compreensão dos termos e definições usadas ao longo do trabalho, apresenta-se o referencial teórico divido em 4 seções, dispostas da seguinte maneira: seção 2.1 - Arquitetura Empresarial (EA) , seção 2.2 - Redes de Petri, seção 2.3 - Redes de Petri Coloridas e seção 2.4 - CPN Tools.

2.1 Arquitetura Empresarial - *Enterprise Architecture*

Segundo (ROSS; WEILL; ROBERTSON, 2006) a chave para a efetivadade da EA de uma empresa é a identificação de processos, informações, tecnologias e interfaces com os clientes.

Como dito anteriormente, a EA possui quatro componentes fundamentais: *Business Architecture*, *Information Architecture*, *Organization Architecture* e *Technology Architecture* (padrões tecnológicos e serviços de infra-estrutura sobre os quais estes estão construídos).

Esta seção dedica-se ao aprofundamento dos domínios da EA e as formas de modelagem já existentes para descrever tais camadas.

2.1.1 *Information Architecture*

Essa arquitetura segundo (ROSS; WEILL; ROBERTSON, 2006), é a composição dos repositórios de dados e fontes de informação da EA. A *Information Architecture* é um componente fundamental da EA e um fator que determina o grau de maturidade da arquitetura. Tal maturidade é determinada basicamente a partir da centralização dos repositórios de informação que atendem os outros sistemas da arquitetura, ou seja, essa maturidade depende da consolidação dessa camada. A modelagem de repositórios de dados ou informações depende do tipo de repositório em questão, mas pouco tem sido alterado desde a proposta de 1976,(CHEN, 1988).

Diferentes ferramentas e abordagens, como a modelagem semântica de (HULL; KING, 1987) têm sido usadas ao longo do tempo até que a evolução dos modelos de bases de dados, no sentido de acompanhar os projetos de aplicações, chegou ao formato Objeto-Relacional. Este formato, ainda baseado no modelo Entidade-Relacionamento (ER), pode ser modelado empregando-se a linguagem UML (*Unified Modeling Language*), (MARCOS; VELA; CAVERO, 2003).

A modelagem de banco de dados usando a linguagem UML é altamente difundida devido ao seu alto grau de aderência aos modelos e diagramas de especificação de aplicações

que serão descritos no item a seguir.

2.1.2 *Organization Architecture*

Evidentemente, dentro de uma organização a liberdade existe em relação ao arranjo de processos (operacionais), seus meios de apoio, estruturas de governança, uso de recursos humanos e assim por diante (HOOGERVORST, 2004). A *Organization Architecture* é definida como um arranjo ou conjunto intencional e estruturado de meios para produzir o resultado desejado de atividades.

Na modelagem de aplicações, especialmente após a difusão da modelagem orientada a objetos, proposta por (RUMBAUGH et al., 1991), também se usa a UML de forma muito ampla e difundida.

No processo de modelagem e especificação de um sistema, diversas etapas devem ser cumpridas, desde a modelagem de diagramas de caso de uso (mais alto nível de abstração), onde se mapeiam as atividades de interação entre usuários e módulos de sistemas, passando por modelagem de classes, na qual se estabelecem as estruturas lógicas e funções, além de diagramas de sequência, nos quais se especificam a ordem de acionamento de cada componente de um sistema.

Para fazer uma modelagem mais completa e com maior nível de detalhes, pode-se, ainda, usar diagramas de colaboração, estados e atividades e diagramas de componentes, em caso de arquiteturas orientadas a serviço (SOA), com reaproveitamento de unidades funcionais (componentes).

2.1.3 *Business Architecture*

Este domínio da EA é responsável pelos processos, procedimentos, ferramentas de negócios e até mesmo dependências para dar suporte às funções dentro de um negócio. Inclui aplicações necessárias nos níveis de operação, gerenciamento e planejamento estratégico.

Desde o início da década de 90, a iniciativa de aferir negócios usando técnicas de gerenciamento de processos tem proporcionado resultados muito bons, pois possibilita documentar claramente os processos de negócio e, como consequência, identificar os principais indicadores de sucesso do processo de negócios.

O gerenciamento de processos de negócio (*Business Process Managament* - BPM), segundo (ELZINGA et al., 1995), demanda quatro fases fundamentais: desenho/modelagem, implantação, execução e controle (medidas e avaliação) dos processos de negócio.

Ao longo do tempo, somente um ciclo fechado de modelagem/melhoria, implantação, execução e controle, pode gerar vantagens competitivas sustentáveis e duradouras - um genuíno ciclo de vida de processo do negócio. O sucesso de tal procedimento, como indicado

por (ELZINGA et al., 1995), depende da orientação de processos da organização e da continuidade com a qual esta é aplicada.

A modelagem ou desenho de um processo de negócio consiste em identificar, de forma geral, o propósito do negócio como um todo, seus principais processos e respectivos indicadores de sucesso e depois, detalhá-los em sub-processos, até chegar ao nível de atividades canônicas, ou individuais.

Com o aumento da velocidade das mudanças dos processos de negócios e a necessidade de revisões frequentes desses processos, aumentou também a complexidade dos sistemas necessários para suportar estas mudanças, (SMITH; FINGAR, 2003) ressaltam a dependência atual do BPM em relação a TI e a tendência de aumento desta dependência.

2.1.4 *Technology Architecture*

O aumento da necessidade e importância da tecnologia da informação no suporte e viabilização dos processos de negócio faz com que a disponibilidade dos serviços de informação seja essencial para o bom desenvolvimento dos negócios.

A *Technology Architecture* é a plataforma que dá suporte às atividade e operações dos outros domínios da EA, pois nela se encontram sistemas de *software*, recursos de armazenamento de dados, redes, interfaces, *hardware* e algumas outras tecnologias, ou seja, toda a infra-estrutura de tecnologia da informação do negócios.

Quando o assunto tange a disponibilidade de serviços, a escolha de componentes de infra-estrutura e a definição dos serviços de suporte adequados são fundamentais para se atingir os resultados requeridos. Neste cenário, o aumento do número de pontos de dependência de *hardware* aumenta, significativamente, a probabilidade de um erro não esperado acontecer e interromper o processo de negócio.

Assim como o BPM (*Business Process Management*) e os métodos de especificação e desenvolvimento de *software*, a tecnologia de *hardware* também evoluiu muito, principalmente nos últimos 20 anos, tanto no quesito de processamento e miniaturização de recursos, quanto no quesito de redes e telecomunicações.

De forma geral, decisões sobre os investimentos em infra-estrutura são tomadas considerando três fatores principais: necessidade mínima de desempenho de componentes da infra-estrutura para o atendimento a sistemas, necessidade de conectividade entre empresas geograficamente distribuídas através da utilização de sistemas de telecomunicações e limitações financeiras para adquirir estes recursos (WEILL; BROADBENT, 1998).

Não basta, porém, apenas definir uma boa infra-estrutura de TI para garantir o suporte aos processos dos negócios, segundo (NOAKES-FRY; DIAMOND, 2003), ferramentas e táticas, para gerenciar o processo de planejamento de continuidade de negócios e

necessidades de retomada em caso de desastres precisam ser usadas em conjuntos com planejamento financeiro e governança corporativa de Sistemas de Informação (SI).

Algumas perguntas importantes surgem quando equipes técnicas analisam esses aspectos são: Como a empresa pode integrar plano de continuidade com o plano geral de riscos corporativos? Como a empresa pode usar ferramentas de planejamento de continuidade para preparar tomadores de decisões para pensarem de forma criativa e flexível no gerenciamento de crises? Segundo (NOAKES-FRY; DIAMOND, 2003), responder essas perguntas requer entendimento profundo dos serviços que suportam a continuidade do negócio, ou seja, o relacionamento entre as quatro camadas da arquitetura corporativa.

2.2 Redes de Petri

As redes de Petri têm origem na tese de Carl Adam Petri intitulada Comunicação com Autômatos apresentada em 1962 na Universidade de Darmsdadt. Apesar de ser considerada uma teoria relativamente jovem, a teoria se adapta bem a um grande número de aplicações em que as noções de eventos e de evoluções simultâneas são importantes (CARDOSO; VALETTE, 1997). Redes de Petri são consideradas como uma ferramenta gráfica e matemática de representação formal que permite modelagem, análise e controle de sistemas a eventos discretos que comportam atividades paralelas, concorrentes e assíncronas (MURATA, 1989). Uma rede de Petri pode ser vista em particular como um grafo bipartido e direcionado com dois tipos de nós chamados de lugares e transições. Os nós são conectados por meio de arcos direcionados. Conexões entre dois nós do mesmo tipo não são permitidas. Dessa forma, os elementos básicos que permitem a definição das redes de Petri são:

- Lugar: em uma rede de Petri um lugar é representado por um círculo. Um lugar pode ser interpretado como uma condição, um estado parcial, um procedimento, a existência de um recurso, etc..
- Transição: uma transição é representada graficamente por uma barra ou retângulo. As transições são associadas a eventos que ocorrem em um sistema.
- Ficha: a ficha é representada por um ponto em um lugar. É um indicador significando que a condição associada ao lugar é verificada, como, por exemplo, um recurso disponível em um certo processo.

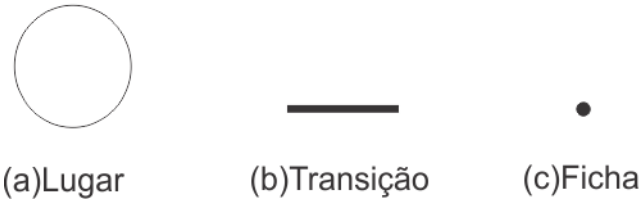

Figura 1 – Elementos Básicos de uma Rede de Petri

As redes de Petri são formalmente definidas da seguinte forma (CARDOSO; VALETTE, 1997) e (MURATA, 1989):

Definição 1 : Redes de Petri - *Uma rede de Petri é uma 4-upla R = <P, T, Pré, Pós> onde:*

(a) *P é um conjunto finito de lugares de dimensão n*;

(b) *T é um conjunto finito de transições de dimensão m*;

(c) *Pré: P × T → N é a aplicação da entrada (lugares precedentes ou incidência anterior), com N sendo o conjunto de número naturais*;

(d) *Pós: T × P → N é a aplicação de saída (lugares seguintes ou incidência posterior).*

Um lugar P é chamado de lugar de entrada de uma transição T, se existe um arco direcionado de P para T. Já um lugar é chamado de saída de uma transição se existe um arco direcionado de T para P.

Em uma rede de Petri, a ocorrência de um evento no sistema é representada pelo disparo da transição no qual o evento está associado. Uma transição T só pode ser disparada se em cada lugar de entrada de T existe pelo menos uma ficha, A ficha indica se a condição associada no lugar está ou não satisfeita. O disparo de uma transição, segundo (CARDOSO; VALETTE, 1997), consiste de: (1) retirar as fichas dos lugares de entrada e (2) depositar fichas em cada lugar de saída. Retirar as fichas de um lugar indica que esta condição não é mais verdadeira após a ocorrência do evento.

A característica marcante encontrada nas redes de Petri é a capacidade de representar graficamente relações e visualizar vários conceitos como: a representação de uma operação (Figura 2), uma sequência de operações (Figura 3), evoluções paralelas (Figura 4, caminhos alternativos (Figura 5), junção de caminhos alternativos (Figura 6), sincronização do tipo Rendez-vous (Figura 7), comunicações assíncronas do tipo semáforo (Figura 8).

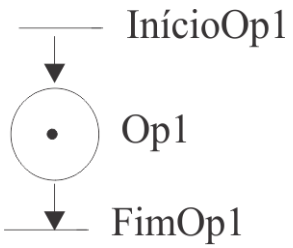

Figura 2 – Operação Básica

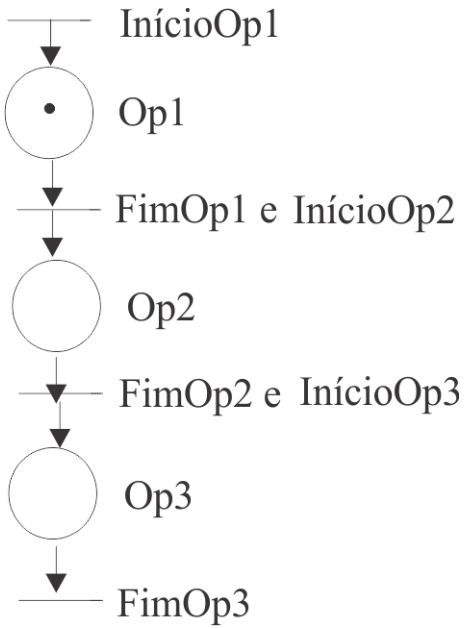

Figura 3 – Sequência de Operações

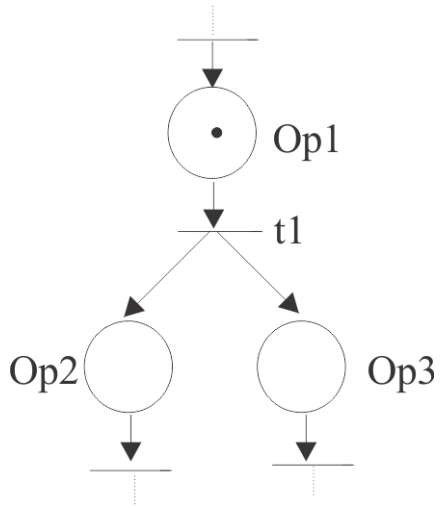

Figura 4 – Evoluções Paralelas

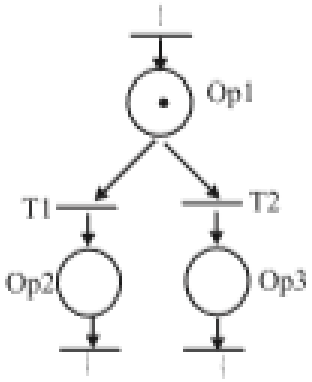

Figura 5 – Caminhos Alternativos

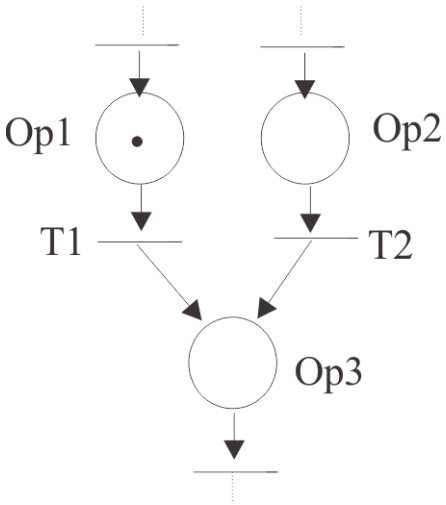

Figura 6 – Junção de Caminhos Alternativos

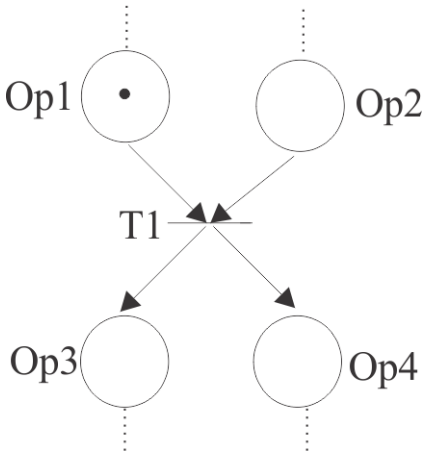

Figura 7 – Sincronização Rendez-vous

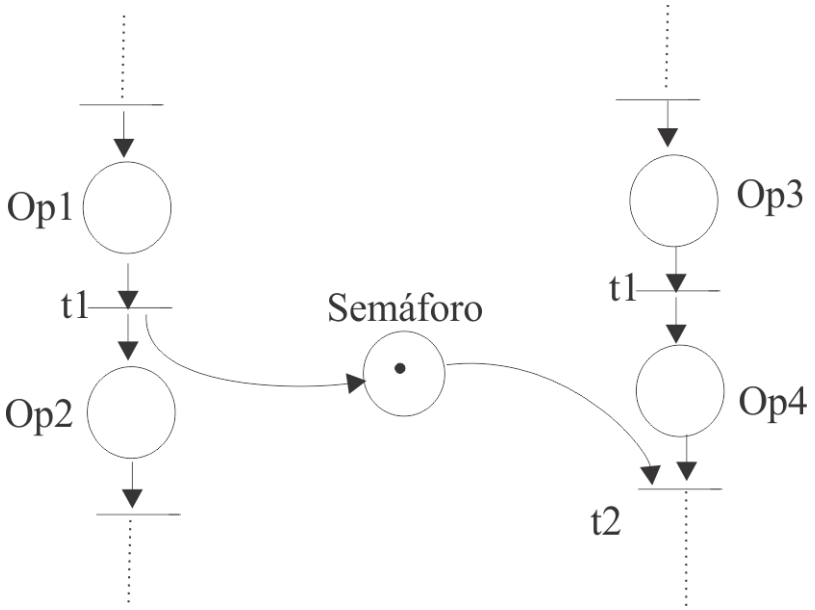

Figura 8 – Comunicação Assíncrona tipo Semáforo

2.3 Redes de Petri Coloridas

(KLEM; FRANTZ, 2016), como uma definição simples, alegam que as redes de Petri coloridas possuem alto nível de abstração, assim os modelos ficam menores, possibilitando infinitas modelagens com maior facilidade em comparação as outras estruturas de redes de Petri. Além disto, as redes de Petri coloridas primam por reduzir os modelos, atribuindo a cada elemento uma cor diferente, sendo que desta forma em uma mesma rede podem ser descritos diferentes recursos e processos. É descrito também por (MARRANGHELLO, 2005) que as redes de Petri coloridas possuem três partes, que são estrutura, declarações e inscrições, onde a estrutura é um grafo dirigido com vértices (lugares e transições), as declarações são as declarações de variáveis e de conjuntos de cores e as inscrições podem variar entre componentes de rede diferentes. A figura 9 mostra um exemplo de rede de Petri colorida.

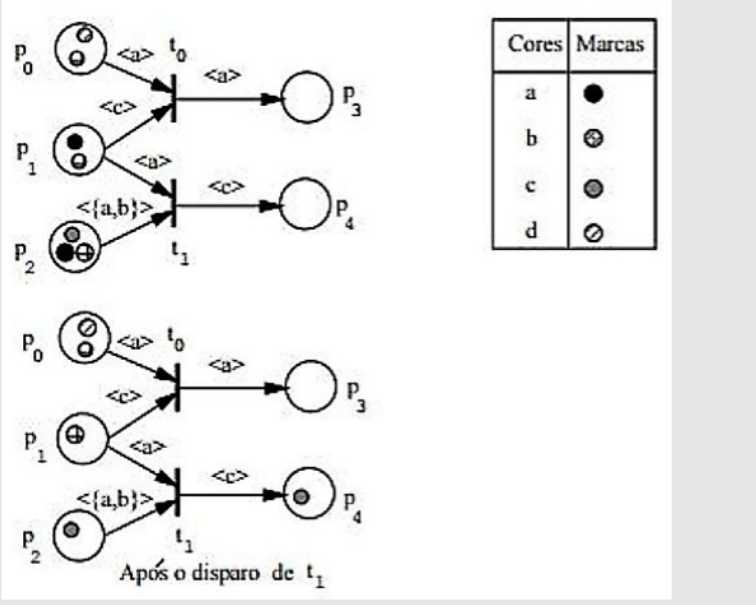

Figura 9 – Rede de Petri Colorida, (KLEM; FRANTZ, 2016).

Um exemplo de problema que pode ser representado pelas redes de Petri coloridas, é o conhecido problema do Jantar dos Filósofos, que de acordo com (PERES, 2013), consiste no seguinte:

- Existem 5 filósofos, que em todo o tempo pensam e comem. Estes filósofos se sentam em uma mesa redonda de cinco lugares e no centro está um recipiente com arroz.
- Em cima da mesa existem também 5 garfos, porém para que um filósofo consiga comer, é necessário que ele pegue 2 garfos, um deles está a sua frente, o outro ele pode pegar da direita, ou da esquerda.
- Enquanto está pensando um filósofo não pode comer, mas de tempos em tempos o mesmo para de pensar, pega 2 garfos e come, logo depois coloca os garfos sobre a mesa novamente e volta a pensar.
- Um filósofo não pode tirar um garfo da mão de outro, os garfos só podem ser pegos quando estão sobre a mesa, sem uso.

A figura 10 demonstra esta situação através de uma rede de Petri colorida, onde (PENHA; FREITAS; MARTINS, 2004) descrevem o seguinte: Cada filósofo e cada garfo caracterizado por uma cor diferente. Inicialmente, a rede possui cinco marcas (também chamadas de fichas, ou *tokens*) de cores distintas no lugar F pensando (filósofos pensando) e cinco marcas de cores distintas no lugar G livres (garfos livres). A função left(filósofo) seleciona o garfo em frente ao filósofo i (garfo i) e a função right(filósofo) seleciona o garfo à direita do filósofo i (garfo i+1).

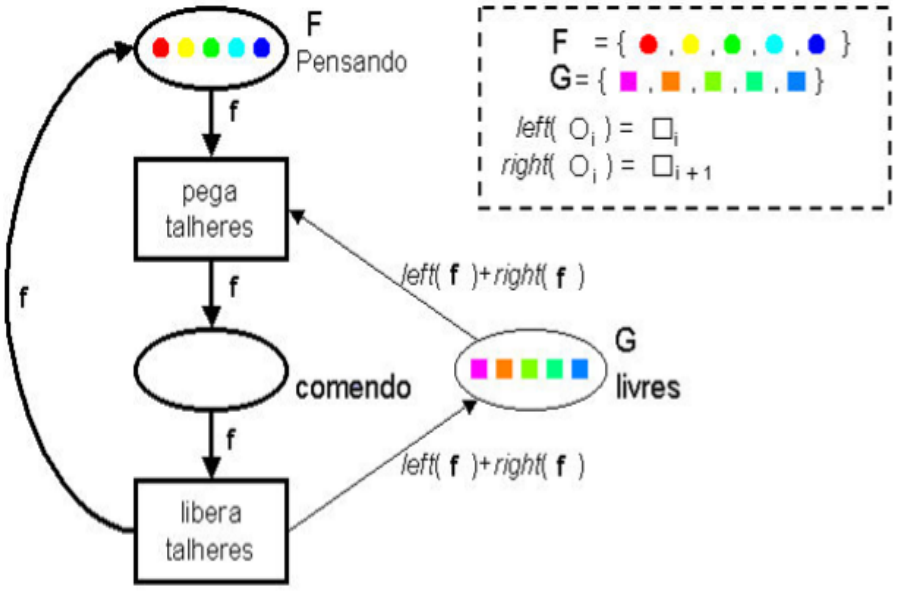

Figura 10 – Problema do Jantar dos Filósofos representado em rede de Petri colorida, (PENHA; FREITAS; MARTINS, 2004)

2.4 CPN Tools

As redes de Petri coloridas foram idealizadas por Kurt Jensen, pesquisador da Universidade de Aarhus, na Dinamarca. Seu artigo, datado do ano de 1981, intitulado *Coloured Petri Nets and the Invariant Method* foi um trabalho promissor, que induziu uma série de estudos sobre coloração das fichas.

Para melhores resultados de modelagem e análise das redes de Petri é importante a utilização de ferramentas que otimizem a criação, manipulação e simulação dos modelos. A ferramenta CPN Tools é considerada apropriada para tais tarefas (JENSEN; KRISTENSEN, 2009).

Várias vantagens do uso da ferramenta CPN Tools juntamente com as redes de Petri são apresentadas por (JENSEN; KRISTENSEN, 2009):

(a) O uso de redes de Petri fornece uma análise para correção do modelo por meio de, por exemplo, uma análise do *state space*. É importante salientar que a análise de estado é realizada a partir do modelo de alto nível através da replicação de cenário e não do estudo do modelo de rede de Petri ordinário subjacente. Isso é então um tipo de análise de propriedade de um modelo de alto nível que é o equivalente a um programa produzido numa linguagem de programação de alto nível.

(b) É muito fácil fazer pequenas adaptações dos modelos e depois comparar os resultados, dadas várias propostas de modelagem.

(c) A integração de processos e de dados complexos é essencial para modelagem de

processos complexos. CPN Tools oferece a possibilidade de integrá-los em um único modelo.

(d) CPN Tools contém um ambiente de simulação, em que se pode seguir passo-a-passo uma sequência de disparo. Simulação de tipo Monte Cario (com replicação de um mesmo cenário aleatório e o cálculo de estimativas de valores médios que pertencem a intervalos de confianças calculados automaticamente pelo *software*).

CPN Tools é uma ferramenta desenvolvida pelo grupo CPN da Universidade de Aarhus (2000-2010), segunda maior universidade da Dinamarca, capaz de trabalhar com vários tipos de redes de Petri, como as redes Temporizadas, redes Clássicas e as redes de Petri Coloridas, possibilitando a criação, simulação de modelos e extração de relatórios e probabilidades que determinam se estes modelos são viáveis, ou se são possíveis. Atualmente o CPN Tools é administrado pelo grupo AIS da Universidade Tecnológica de Eindhoven, que fica nos Países Baixos.

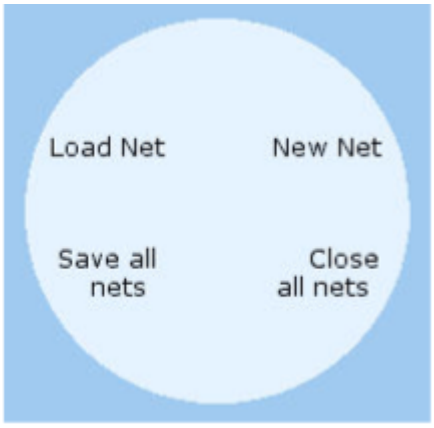

Figura 11 – Menu de marcação de rede.

O CPN Tools trabalha com uma série de estruturas e menus que determinam a ação a ser tomada, estes menus são chamados de menus de marcação Se tem acesso a eles clicando com o botão direito do mouse sobre o local desejado. A figura 11 mostra o menu de marcação de rede, onde é possível criar, carregar, salvar e fechar redes de Petri.

Figura 12 – Paleta de Redes

Todas estas opções podem ser encontradas também na paleta de ferramentas *Net*, que fica no menu à esquerda, figura 12.

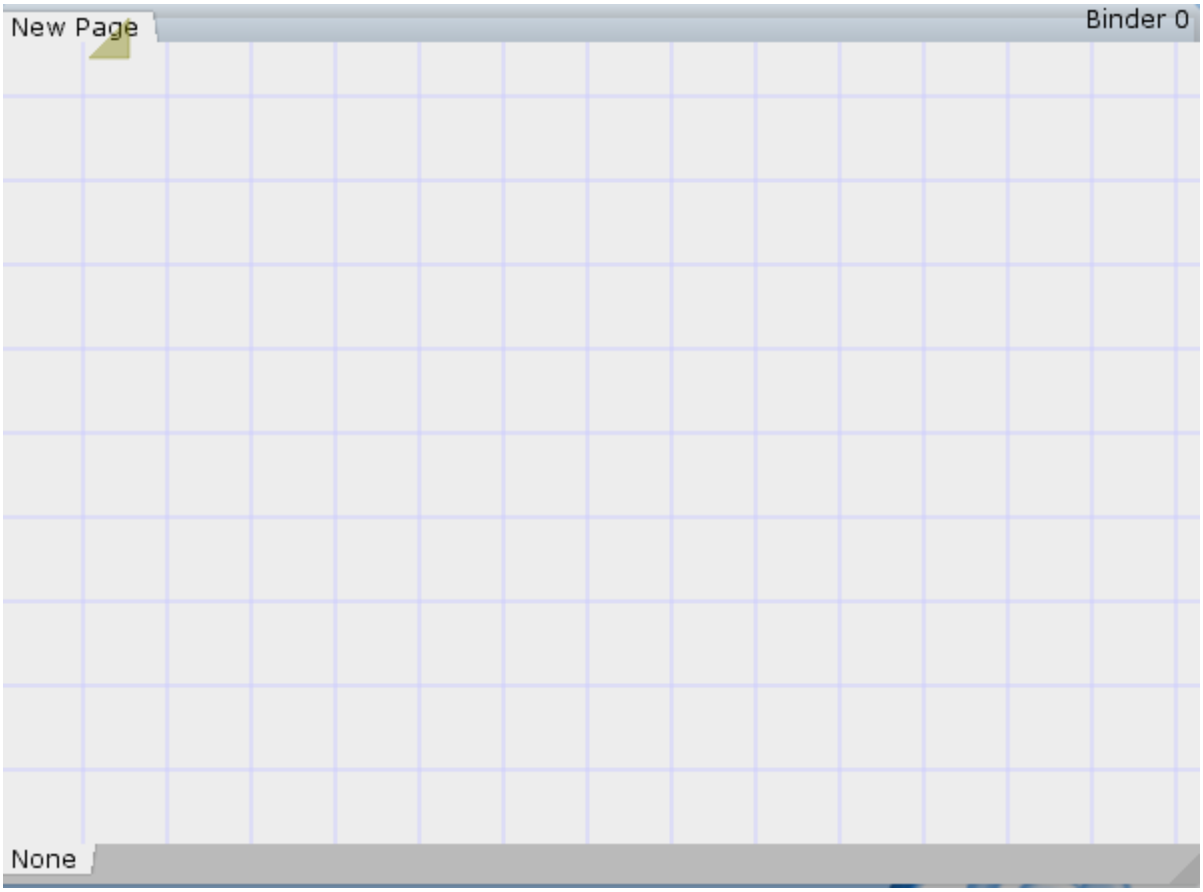

Figura 13 – Nova Página

Ao escolher a opção *Create a new net* é mostrada a estrutura da figura 13, onde é possível iniciar a criação dos modelos.

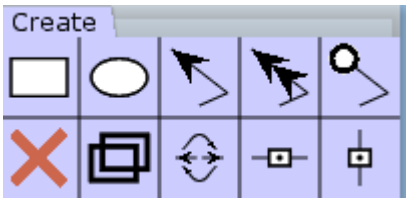

Figura 14 – Paleta *Create* - ferramenta de criação

A paleta de ferramentas *Create* é também uma das principais, visto que ela é usada para criar os lugares, as transições e os arcos. Esta paleta está representada na figura 14.

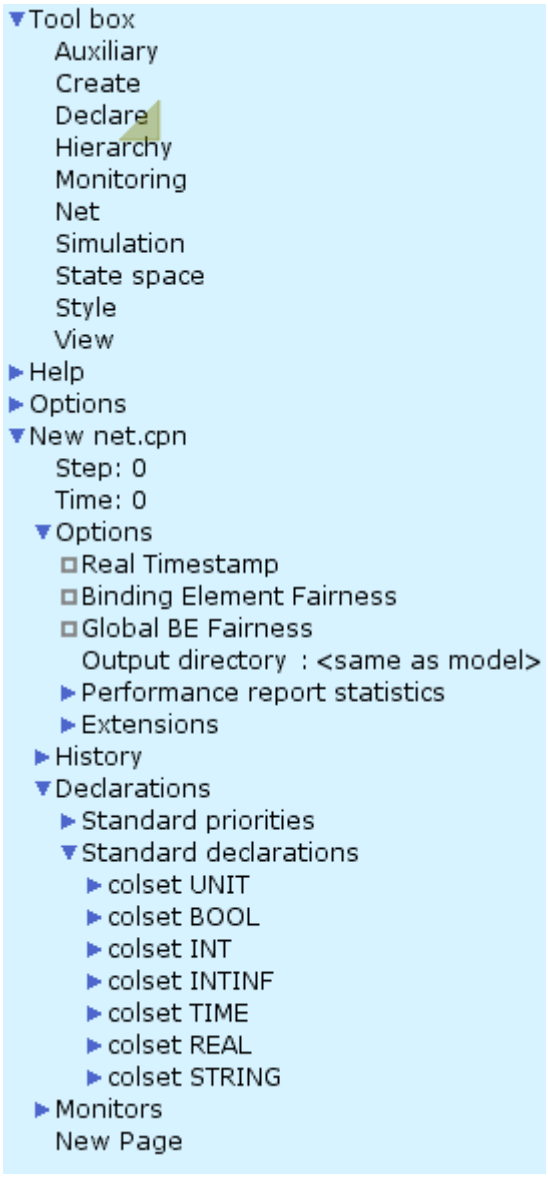

Figura 15 – Índice da ferramenta

Ao lado esquerdo da ferramenta CPN Tools temos o Índice, onde se encontram todos os painéis de ferramentas, além de todos os elementos da rede utilizada, como as declarações e os monitores, (Figura 15).

Um *colset*, como visto nas declarações da figura 15 é a nomenclatura de como são declarados os conjuntos de cores. eles são os tipos de dados em uma rede de Petri colorida. Segue abaixo um exemplo de declaração e alguns dos tipos de dados possíveis para as variáveis *colset* segundo (CPNTOOLS, 2018):

```
colset name = unit [with new_unit]  =  Exemplo de declaração.
```

Unit: O conjunto de cores da unidade compreende um único elemento. *Boolean*: Aceita valores *true* ou *false*. *Integer*: Aceita apenas números inteiros. *LargeInteger*: Aceita

números inteiros grandes. *Real*: Aceita números não inteiros, ou seja, que possuem casas decimais após a vírgula. *Time*: Tipo que guarda valores temporais. *String*: Aceita cadeias de caracteres.

Ainda segundo (CPNTOOLS, 2018), ao simular redes de Petri coloridas, é útil em muitas situações poder analisar as marcações e os elementos de ligação que ocorrem, para extrair periodicamente informações das marcações e elementos de ligação e, em seguida, usar a informação para diferentes fins, tais como:

- Parando uma simulação quando um determinado local está vazio.
- Contando o número de vezes que ocorre uma transição.
- Atualizando um arquivo quando ocorre uma transição com uma variável vinculada a um valor específico.
- Calculando o número médio de fichas em um lugar.

O CPN Tools tem um elemento que provê todas estas funcionalidades, que é o monitor. O mesmo é usado para observar, inspecionar, controlar ou modificar uma simulação de uma rede de Petri colorida. Muitos monitores diferentes podem ser definidos para uma determinada rede. Os monitores podem inspecionar as marcações de lugares e os elementos de ligação que ocorrem durante uma simulação, e eles podem tomar determinadas ações com base nas observações.

Como um exemplo de modelo de simulação completo, a figura 16 mostra o mesmo problema do Jantar dos Filósofos descrito anteriormente, modelado através da ferramenta CPN Tools.

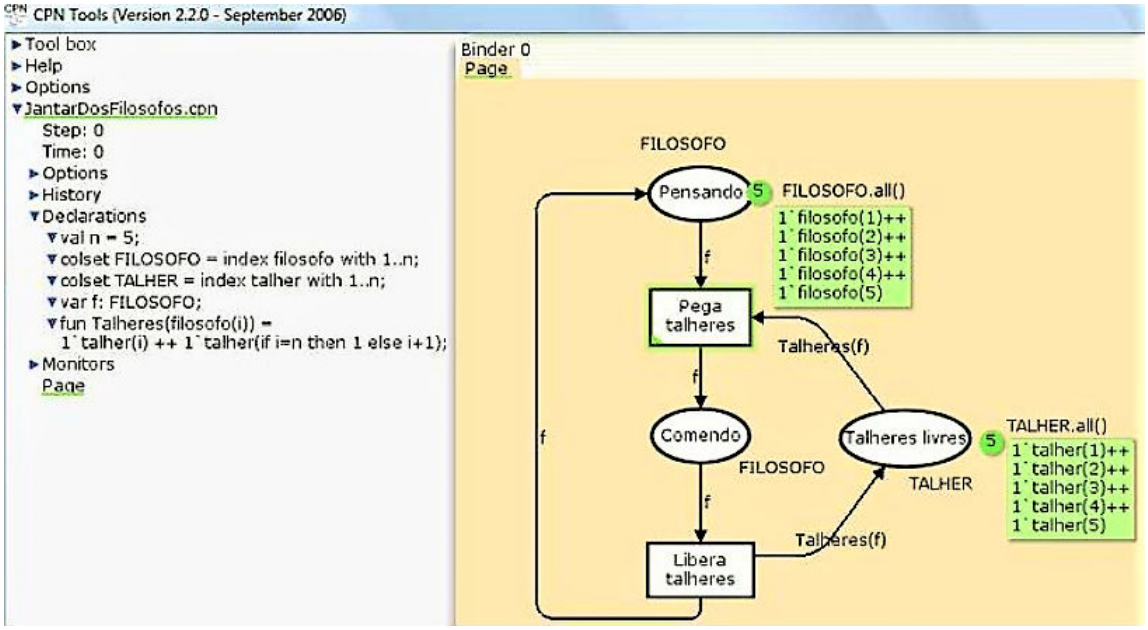

Figura 16 – Jantar dos Filósofos Modelado Utilizando CPN Tools, (OLIVEIRA;MORAIS, 2011)

CPN combina rede de Petri colorida com uma linguagem de programação funcional para obter uma linguagem de modelagem para sistemas concorrentes. Redes de Petri fornecem a base formal para modelar concorrência e sincronização; uma linguagem de programação fornece as primitivas para modelar manipulação de dados e criação de modelos compactos e parametrizáveis. Sendo assim, em CPN as fichas possuem valores que são conhecidos como cores. Essas cores não significam apenas cores ou padrões, elas podem representar tipos de dados complexos (JENSEN; KRISTENSEN, 2009). Dessa forma, as CPNs foram projetadas para tornar mais compactos modelos de sistemas complexos, permitindo a individualização de fichas (usando as cores que lhe são atribuídas). Assim, diferentes processos ou recursos podem ser representados em uma mesma estrutura gráfica.

2.4.0.1 Marcações

Um lugar em uma CPN pode conter diversas fichas, então, para representar todas as fichas de um lugar são utilizados os chamados *multisets*. um *multiset* na linguagem CPN é denotado como:

$$x'_1 v_1 + x'_2 v_2 + ... + x'_n v_n$$

Ou seja, existem x'_1 fichas de valor v_1 no lugar em questão, x'_2 fichas de valor v e assim sucessivamente.

A Figura 17 mostra três situações diferentes. Na primeira os lugares p_1 e p_2 tem conjuntos de cores INT e STR, respectivamente, mas não contém fichas. A Figura 17b

mostra os mesmos lugares, porém com uma ficha e consequentemente multisets definidos. A Figura 17c mostra os lugares com seis fichas e de acordo com a expressão definida o lugar p_1 contém uma ficha de valor 2 e cinco fichas de valor 4. O mesmo acontece com p_2 que contém uma ficha de valor "Jonh"e cinco fichas de valor "Sara".

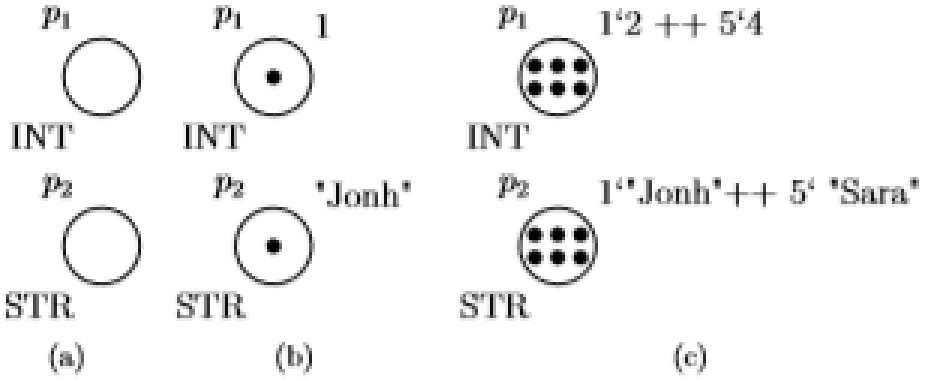

Figura 17 – *Multisets*

3 TRABALHOS RELACIONADOS

Apesar da quantidade de trabalhos voltados para os diferentes tipos de modelagem de EA, há dificuldade de se encontrar pesquisas com o intuito de analisar o comportamento de um modelo de atividades de EA sobre um sistema multiprocessado. As obras apresentadas abordam os seguintes assuntos: CPN Tools ou modelagens utilizando a ferramenta, EA, e por fim, trabalhos de modelagem de EA onde a ferramenta CPN Tools é utilizada.

3.1 Critérios de Busca

As obras apresentadas foram filtradas por meio do sistema de busca de periódicos da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Google Acadêmico e IEEE *Explore Digital Library*. Os resultados das buscas foram: artigos científicos, monografias, dissertações de mestrado e teses de doutorado.

Quanto aos idiomas dos trabalhos, pode-se perceber que a maior concentração de conhecimentos se encontram em inglês, porém existem trabalhos nacionais relevantes, por isso foram usados trabalhos de língua estrangeira e nacional para mostrar o estado da arte.

A busca nas duas bases de dados retornaram uma quantidade massiva de trabalhos quando os assuntos foram pesquisados separadamente, quando o assunto pesquisado era diretamente relacionado a este trabalho, um retorno de 5950 trabalhos foi obtido, o que causou a necessidade da criação de uma metodologia para diminuir a quantidade total de trabalhos.

3.2 Metodologia de Análise

Nesta seção será exposta a metodologia utilizada para filtrar os trabalhos relacionados, composta de 3 critérios sendo eles:

(a) **Critério 1** (C1) - Trabalhos que descrevam das mais variadas formas a EA.

(b) **Critério 2** (C2) - Trabalhos que envolvem CPN Tools ou modelagens (qualquer tipo de modelagem) utilizando esta ferramenta.

(c) **Critério 3** (C3) - Trabalhos que se relacionam diretamente com o propósito deste, modelagem de EA utilizando a ferramenta CPN Tools.

3.3 Trabalhos Analisados

De acordo com os critérios descritos na seção 3.2, foram analisados 4 trabalhos, sendo eles:

Trabalho 1 - *Enterprise Architecture: Enabling Integration, Agilty And Change, (HOOGERVORST, 2004).*

Este trabalho tem como objetivo apresentar os núcleos relevantes dos negócios modernos e definir a arquitetura e sua importância a partir dessas perspectivas. Ele diz que as organizações podem ser definidas como sistemas sociotécnicos adaptativos e que esses sistemas podem ser vistos de duas maneiras diferentes: Funcional, que diz respeito ao gerenciamento do negócio e trata da integração perfeita entre clientes e processos operacionais, agilidade e a capacidade de mudança, que são relevantes para uma execução bem-sucedida das escolhas estratégicas; Construtiva, como o sistema deve ser projetado e contruído, onde o conhecimento sobre a operação interna do sistema é essencial.

Segundo esta pesquisa a EA pode ser definida como a coletividade dos princípios de *design* já mostrados anteriormente e sua importância está relacionada com a capacidade de realizar operações integradas dos sistemas humanos e tecnológicos, que remete a noção de integração. A integração, por sua vez se torna necessária e está diretamente relacionada com a estratégia corporativa, onde todos os elementos devem estar alinhados para que a estratégia funcione.

A EA também é importante no campo das mudanças segundo este trabalho, salvaguardando e possibilitando mudanças emergentes, e por fim esta obra se refere a uma necessidade crescente de agilidade empresarial e organizacional afim do negócio não ficar limitado a ações imediatas, referindo-se então a importância da EA em fornecer agilidade.

Os quatro princípios de *desing* que formam a EA: *Business Architecture*, *Organization Architecture*, *Information Architecture* e *Technology Architecture* são descritas de forma detalhada e necessitam estar integradas para que o negócio seja competitivo. Cada um desses princípios possui sua própria arquitetura e são descritos em *frameworks* nesta obra, descrevendo as características inerentes a cada tipo de arquitetura e seus domínios. Com isto ficou claro a definição de EA e sua importância no universo das organizações, atendendo assim ao critério C1.

Trabalho 2 - *Enterprise Architecture as strategy: Creating a foundation for business execution, (ROSS; WEILL; ROBERTSON, 2006).*

Neste livro (ROSS; WEILL; ROBERTSON, 2006) a definição de EA é apresentada como "A organização lógica dos processos de negócios e da infra-estrtura de TI, refletindo os requisitos de integração e padronização reqeridos pelo modelo operacional de uma empresa, que é composta por quatro compenentes fundamentais, sendo eles: *Data/Information*

Architecture, *Aplication Architecture*, *Technology Architecture* (serviços de infra-estrutura e padrões tecnológicos sobre os quais os serviços são construídos) e *Business Process Architecture*". Essas quatro são detalhademente descritas no trabalho.

O trabalho também descreve a comunicação entre os processos de negócio de alto nível e os requisitos de TI do modelo operacional de uma empresa, e que uma EA eficiente é o ponto de partida para definir uma boa fundação para a execução do negócio. A consolidação deste modelo se deu em 2006. Embora existam outras formas para a modelagem das quatro camadas arquiteturiais, não existe nenhuma maneira unificada de modelagem.

Desta forma pode-se entender claramente os conceitos de EA descritos nesse trabalho, atendendo ao critério C1.

Trabalho 3 - *Quantitative modeling of power performance trade off son extreme scale systems, (YU et al., 2015).*

Este trabalho argumenta que a computação de alto desempenho (HPC - *High Performance Computing* evolve cada dia mais em complexidade e escala e a energia consumida nesse processo se torna uma restrição fundamental. Com o avanço da tecnologia para a computação *exascale* (sistemas computacionais aptos de operar em torno de um quintilhão de operações de ponto flutuante por segundo) não é mais possível deixar de observar os fatores que envolvem esse avanço, como a energia consumida por esses sistemas.

Este artigo exibe uma modelagem de desempenho/energia denominada PuPPET (*Power Performance Petri Net*). Nesta modelagem são usadas as redes de Petri coloridas, tornando essa modelagem rápida e escalável para as diferentes medidas de energia e desempenho. O PuPPET permite escalar centenas de milhares de núcleos de processador e concomitantemente, fornecer altos níveis de precisão de modelagem, o que o torna adequado para análise de eficiência energética de sistemas de escala extrema.

O PuPPET proporciona uma correlação entre aumento do consumo de energia e aumento de processamento, apresentando o nível ótimo de consumo de energia em relação ao desempenho e também quando esse consumo é muito maior que o ganho de performance. É possível determinar as interações dificultosas entre fatores que de alguma maneira causaram uma deficiência energética devido aos recursos avançados das redes de Petri coloridas. Este modelo foi criado a partir da ferramenta CPN Tools, uma ferramenta eficiente na criação de modelos computacionais. Sendo assim este trabalho atende ao critério C2.

Trabalho 4 - *Modeling Enterprise Architecture Using Timed Colored Petri Net: Single Processor Scheduling, (PASHAZADEH; NIYARI, 2014).*

Este artigo presente, se mostrou o melhor correlacionado ao propósito desse projeto. Esta obra teve como objetivo modelar EA utilizando CPN Tools utilizando o método nao

preemptivo, escalonando e executando os processos baseados em alguns algoritmos não preemptivos, que são: Método do trabalho mais curto (SJF - shortest job first), método primeiro a entrar, primeiro a sair (FIFO- first in first out), método da taxa de resposta mais próxima (HRRN - highest response ratio next). Após isso, este trabalho fez a comparação entre o desempenho dos escalonadores e teve por conclusão que não existe uma abordagem geral para a modelagem da EA e que a mesma pode ser feita utilizando vários modelos e métodos diferentes sem causar grandes impactos no resultado final.

Este trabalho ainda nos traz alguns conceitos de EA e cita a importância do planejamento dentro de uma organização, aumentando a sua produtividade. Para isso foi criada uma rede de Petri utilizando a ferramenta CPN Tools, segundo ele uma boa ferramenta para criação dos modelos de simulação, facilitando a tomada de decisões quanto a arquitetura dos sistemas de informação.

Esta obra é semelhante ao presente projeto, atendendendo a todos os critérios, principalmento o critério C3, entrentanto o diferencial deste projeto está na simulação dos escalonadores utilizando mais de um processador, em função dos avanços dos recursos computacionais, o que já demonstra sua contribuição científica, obtendo novos resultados após a simulação e montando uma nova análise dos dados obtidos.

4 ARQUITETURA DO SISTEMA MONO-PROCESSADO

Este capítulo destina-se a elucidar o funcionamento do modelo monoprocessado proposto por (PASHAZADEH; NIYARI, 2014), suas variáveis, *color sets* e funções implementadas.

4.1 *Color Sets*

Os *color sets* utilizados no modelo são definidos como segue abaixo:

```
colset RT = product INT*INT;

colsetProcess = record PI=INT* IT:INT * ST:INT * WT:INT * ES:INT * PR:RT timed;

colsetProcList = list Process timed;

colset SCHTYPE = with FCSF | SJF | PR | HRRN timed;
```

O *color set* RT é definido como um par de números inteiros, representando os dois campos de prioridade dos processos, sendo o primeiro campo a prioridade primária, e o segundo campo a prioridade secundária.

o *color set* Process contém seis campos. o primeiro campo, PI, é do tipo inteiro, representando o indíce do processo. O segundo campo, denotado por IT, representa o tempo de chegada do processo no sistema para ser executado em um número inteiro. O terceiro campo, ST, é do tipo inteiro e representa o tempo que o processo ficará em execução. O quarto campo, WT, representa o tempo de espera do processo até ser executado através de um número inteiro. O quinto campo chamado de ES, é do tipo inteiro e representa quando o processos entrou no processador para ser executado. O último campo, PR, é do tipo RT, e representa a prioridade de um processo.

O *color set* ProcList define uma lista de elementos do tipo *Process*. Já o *color set* SCHTYPE é usado para representar os quatro métodos de escalonamento do sistema, SFJ (*Shortest Job First*), HRRN (*Highest Responde Ratio Next*), PR (*Priority Based*) e FCFS (*First Come First Serve*).

4.2 Marcações Iniciais e Variáveis

As marcações iniciais do modelo definem os processos e seus atributos, e são definidas conforme segue:

```
    val InitialProcess =

[{PI=1, IT=6, ST=4, WT=0, ES=0, PR=(2,0)},
 {PI=2, IT=7, ST=3, WT=0, ES=0, PR=(1,0)},
 {PI=3, IT=8, ST=2, WT=0, ES=0, PR=(2,0)},
 {PI=4, IT=5, ST=2, WT=0, ES=0, PR=(3,0)},
 {PI=5, IT=9, ST=3, WT=0, ES=0, PR=(1,0)},
 {PI=6, IT=1, ST=3, WT=0, ES=0, PR=(4,0)},]
```

A tabela abaixo mostra os seis processos descritos acima, mostrando o indíce, tempo de entrada dos processos no sistema, tempo de serviço e prioridade.

Índice dos Processos	Tempo de Entrada	Tempo de serviço	Prioridade
P1	6	4	2
P2	7	3	1
P3	8	2	2
P4	5	3	3
P5	9	3	1
P6	1	3	4

As variáveis utilizadas no modelo estão descritas abaixo:

```
var sm: SCHTYPE;
var process: Process:
var allproc, run, rq, rp, proc, queue: ProcList;
```

4.3 Modelo do Sistema

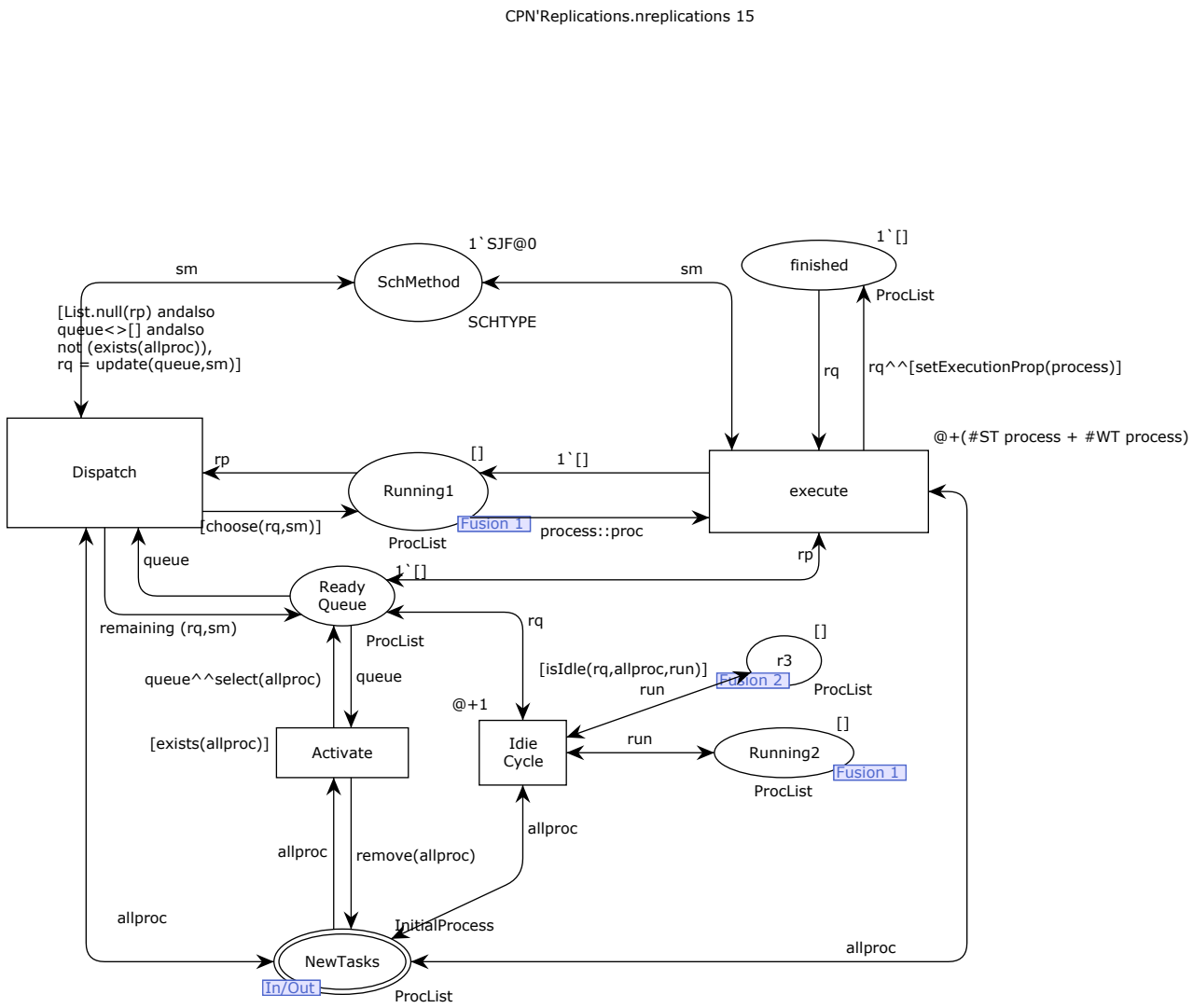

Figura 18 – Modelo Monoprocessado

4.4 Funcionamento do Modelo

Nesta seção será detalhado o funcionamento do modelo proposto por (PASHAZADEH; NIYARI, 2014), apresentando as transições, guardas de transições, funções, inscrições dos arcos e lugares.

4.4.1 Funções do Modelo

```
fun curTime(): int = IntInf.toInt(time());
```

A função *curTime* não possui parâmetro de entrada e retorna o tempo atual de simulação do modelo.

```
fun onTime(q:Process): bool = #IT(q)<=curTime();
```

A função *onTime* recebe como parâmetro de entrada um processo e chama a função *curTime*. Se o tempo de simulação do processo for menor ou igual ao tempo atual de simulação, retorna verdadeiro como resultado, caso contrário, retorna falso.

```
fun exists(z:ProcList): bool =List.exists onTime z;
```

A função *exists* recebe como entrado uma lista de processos e produz um valor booleano como saída. Essa função checa a lista de entrada inteira. Chamando a função *onTime*, se algum processo tem o seu tempo de entrada atingido baseado do tempo atual de simulação, a função retorna verdadeiro, caso contrário, falso.

```
fun udpdatePR (sm:SCHTYPE, q:Process) =
    case sm of
        FCFS => Process.set_PR q(#IT q,0)
    | SJF => Process.set_PR q(#ST q, #IT q)
    | PR => q
    | PR => Process.set q(#1(#PR q), #IT q)
    | HRRN => Process.set_PR q(((#ST q+ #WT q)*100)div #ST q, 0)
```

A função *updatePR* recebe dois parâmetros de entrada, sendo eles o método de escalonamento e um processo, respectivamente. Ela calcula a prioridade dos processo a ser processados de acordo com o método de escalonamento e atualiza seu atributo de prioridade. No método FCFS, a prioridade de cada processo é baseada de acordo com o seu tempo de entrada no sistema, caso a prioridade de dois processos forem iguais, então o segundo campo do atributo prioridade será utilizado, dando maior prioridade ao processos mais rápido. Já no método de escalonamento SJF, o processo mais rápido terá uma prioridade maior, caso a prioridade de dois processos sejam igual, o processo que entrou primeiro terá a maior prioridade. No método PR, as prioridades dos processos são definidas estaticamente no começo, e caso dois processos tenham a mesma prioridade, o processo com o atributo inputTime menor terá a maior prioridade. O metódo de escalonamento HRRN calculará a prioridade dos processos de acordo com a fórmula (serviceTime + waitingTime)/ serviceTime. Caso dois processos tenham a mesma prioridade, aquele cujo indíce é menor (definido no começo da simulação) terá a maior prioridade.

```
fun updateProcWait(q:Process)= Process.set_WT q(curTime()-#IT q)
```

Essa função recebe como entrada um processo e atualiza o parâmetro *waitingTime*. Seu funcionamento começa chamando a função *curTime* para obter o tempo atual de

simulação e decrementa o parâmetro *inputTime* para obter o tempo de espera do processo e retorna o processo atualizado como saída.

```
fun updateWait(L:ProcessList)= List.map updateProcWait L;
```

A função *updateWait* recebe uma lista de processos como entrada e atualiza todos os tempos de espera dos processos da lista chamando a função *updateProcWait*.

```
fun update(L:ProcList, sm:SCHTYPE)=

let
    val aw = updateWait(L)
    fun upd L = updatePR(sm,L)
in
 List.map upd aw
end;
```

A função *update* recebe dois parâmetros de entrada, uma lista de processos que estão esperando para ser selecionados para execução e o método de escalonamento sendo utilizado no modelo, respectivamente. A função atualiza o tempo de espera dos processos da lista de entrada chamando a função *updateWait* e a prioridade dos processos chamando a função *updatePR*.

```
fun cmp(a:INT, b:INT, sm:SCHTYPE): INT=
 if (sm=FCFS orelse sm=SJF) then
   if a<b then 1
   else if a=b then 0 else ~1
 else
   if a>b then 1
   else if a=b then 0 else ~1
```

A função *cmp* recebe três parâmetros de entrada. O primeiro parâmetro é a prioridade calculada do primeiro processo, o segundo é a prioridade calculada do segundo processos e o terceiro parâmetro é o método de escalonamento do modelo. Se o método for FCSF ou SJF, um valor numérico menor representa uma prioridade maior. Caso o método seja o HHRN, um valor numérico maior representa uma prioridade maior. Se o primeiro processos possui uma prioridade maior que o segundo, a função retorna o valor 1, caso os processos tenham a mesma prioridade a função retorna o valor 0, caso contrário retorna o valor -1.

```
fun compare(p1:Process, p2:Process, sm:SCHTYPE): INT =
 (*if relation > be true for two product then 1
 if two product be equal then 0
 if relation < be true for two produc then ~1 *)

 let
    val r = ref 0
    val major = cmp(#1(#PR p1), #1(#PR p2), sm)
    val minor = cmp(#2(#PR p1), #2(#PR p2), sm)
in
    if (major<>0) then r:=major
    else if (minor<>0) then r:=minor
        else if (#PI p1 < #PI p2) then r:=1
            else r:=~1;
!r
end
```

A função *compare* recebe três parâmetros de entrada, sendo os dois primeiros processos e o terceiro o método de escalonamento do modelo. Os atributos de prioridades atualizados de cada processos são guardados em formas de pares (valores inteiros) no campo PR dos processos. O primeiro campo é utilizado a prioridade principal enquanto que o segundo é a prioridade secundária. Essa função compara ambos os campos de prioridade chamando a função *cmp*. Caso o campo primário de prioridade de ambos os processos sejam iguais, então o campo secundáro será utilizado para determinar a prioridade relativa dos processos. Se ambos os campos de prioridade, primário e secundário, forem iguais, então o processo cujo indíce é menor, terá a maior prioridade. Se o primeiro processo tem a prioridade maior, então a função retornatá o valor 1, caso contrário a funçao retornará o valor -1 como resultado.

```
fun elect(q:ProcList, maxi:int, c:int, sm:SCHTYPE):INT =
let
    val len = List.lenght(q)
in
    if(c=len) then
        maxi
    else
        let
            val hr1 = List.nth(q,maxi)
            val hr2 = List.nth(q,c)
            val cmp = compare(hr1,hr2,sm)
```

```
        in
            if(cmp=1) then elect(q,maxi,c+1,sm)
            else elect(q,c,c+1,sm)
            end
end;
```

A função *elect* possui quatro parâmetros de entrada e é executada recursivamente. O primeiro parâmetro é uma lista de processos que estão prontos e aguardando para serem executado. O segundo parâmetro é o índice do processo da lista de entrada que possui a maior priorade até o momento. È importante observar que o índice dos processos na lista começo em 0, ou seja, na primeira chamada da função o segundo parâmetro será 0, significando que o primeiro processo será considerado com a maior prioridade. Já o terceiro parâmetro funciona como um ponteiro. Representa o índice de um processo da lista na chamada atual da função que será comparado com o processos do segundo parâmetro dessa função. O quarto parâmetro é o método de escalonamento do modelo. Essa função chama a função *compare*, seleciona o processo que possui a maior prioridade e retorna o seu índice como resultado. Caso múltiplos processos possuam a mesma prioridade, ela retornará o índice do processo que está na cabeça da lista.

```
fun choose(q:ProcList, sm:SCHTYPE) =
let
    val mx = elect(q,0,1,sm)
in
    List.nth(q,mx)
end;
```

A função *choose* recebe dois parâmetros de entrada, sendo o primeiro uma lista de processos que já estão prontos para serem executados, e o segundo o método de escalonamento do modelo. Chamando a função *elect*, escolhe o processo com maior prioridade da lista de processos e retorna seu índice, esse índice será o retorno da função, como sendo o processo escolhido para ser executado.

```
fun remaining(q:ProcList, sm:SCHTYPE) =
let
    val mx = elect(q,0,1,sm)
in
    List.take(q,mx)^^List.drop(q,mx+1)
end;
```

A função *remaining* recebe como parâmetro de entrada uma lista dos processos que estão prontos para execução e o método de escalonamento do modelo, respectivamente. Para obter a saída da função, ela chama a função *elect* para obter o índice do processos escolhido para ser executado e cria uma lista de processos como saída, sendo essa lista a mesma da entrada, exceto pelo processo que foi escolhido para ser executado.

```
fun setExecutionProp(q:Process): Process = Process.set_ES q(curTime())
```

A função *setExecutionProp* recebe como parâmetro um processo e chama a função *curTime*. Ao chamar essa função, ele atualiza o tempo de início do processo com o tempo atual de simulação retornado pela função *curTime*, e retorna o processo atualizado como saída.

```
fun isIdle(rq,init,run) =
if(run=[] andalso rq=[] andalso init<>[]) then
    if(not(exists(init))) then true
    else false
else
    false;
```

Essa função recebe 3 parâmetros de entrada. O primeiro parâmetro é uma lista de processos que estão prontos para serem executados. O segundo parâmetro é uma lista de processos que entrarão no sistema mais tardar. O terceiro parâmetro é uma lista de processos que estão rodando no momento, o tamanho dessa lista será no máximo 2, visto que temos apenas 2 processadores no modelo. Essa função indica se um processador está ocioso ou não.

Se algum processo está rodando no momento ou se a lista de processos prontos nao está vazia, então o processador não pode estar ocioso, retornando então o valor falso como resultado. Caso nao haja nenhum processos sendo executado e a lista de processos prontos esteja vazia e nenhum processo na lista de espera tenha atingido seu tempo de entrada no sistema baseado no tempo atual de simulação, então o valor retornado será verdadeiro. Chamando a função *exists* e o retorno for um valor verdadeiro, significa que algum processo atingiu o tempo para entrar no sistema, caso contrário, retornará falso.

4.5 Operação Detalhada do Modelo

Nesta seção do trabalho será explicitado o funcionamento do modelo apresentado, detalhando as transições, inscrições de arcos e lugares.

Ao iniciar a simulação do modelo, o lugar *NewTasks* mantém a lista de processos definidos como marcação inicial. A transição *Activate* será ativado com esse lugar conter algum processo cujo input time foi atingido em relação ao tempo atual de simulação, tal condição será checada pela função *exists*. Após o disparo da transição *Activate*, a função *select* extrairá os processos da lista inicial cujo tempo de entrada no sistema foi atingido, adicionado esses processos no lugar *ReadyQueue*, enquanto a função *remove* retorna os processos que nao atingiram o tempo de entrada para o lugar inicial do modelo.

O lugar *ReadyQueue* contém uma lista de processos que estão esperando para serem executados, e os dois processadores possuem clones, chamados de *fusion places*, colocados apenas para melhor visualização da operação ocorrida no modelo. Os processos serão executados apenas um por vez em cada processador, porém ao adicionar um núcleo de processamento, conseguimos eliminar o tempo de espera de alguns processos, reduzindo o tempo total final do modelo.

A transição *Dispatch* será ativada quando três condições forem verificadas. Quando nenhum processo estiver sendo executado, a lista de processos prontos para serem executados nao for vazia e quando não existir nenhum processo cujo tempo de entrada no sistema foi atingido, porém não se encontra na lista de processos pronto. Essa terceira condição é descrita para que nenhum processo seja omitido durante a execução, pois os processos podem atingir o seu tempo de entrada durante a execução de outro processos. Sendo assim, quando algum processo atingir o tempo de entrada no sistema durante alguma execução, a transição *Dispatch* será desativada e a transição *Activate* ativada para que possa colocar esse processo no lugar de processos prontos.

A condição de guarda da transição *Dispatch* atualiza as prioridades dos processos chamando a função *update*. Quando a transição for ativada, a função *choose* elegerá o processo com maior prioridade para ser executado, (essa parte da simulação será executada de acordo com o tamanhao da lista de processos prontos) com maior prioridade e inserir na lista do processador para ser executado. A função *remaining* retira os processos que não foram escolhidos para serem executadas e retorna a lista atualizada para o lugar *ReadyQueue*.

Quando um processo aparece na lista do processador, isso significa que ele foi selecionado para ser executado. Então a transição *Execute* será ativada, isso representa que um processo foi rodado e computado pelo sistema, este processo entao irá para o lugar Finished e o tempo de simulação do modelo avançará a mesma quantia que o tempo de simulação do processos (*Service Time*.

5 ARQUITETURA DO MODELO MULTIPROCESSADO

Esta seção tem como intuito de descortinar o modelo proposto neste trabalho, mostrando as diferenças entre os dois modelos, funções inplementadas e variáveis iniciais.

5.1 *Color Sets*, Variáveis e Marcações Iniciais

As váriaveis utilizadas no modelo multiprocessado são as mesmas apresentadas anteriormente no Capítulo 4, assim como os *color sets*.

A diferença entre os dois modelos começa a surgir nas marcações iniciais, onde o atributo *input time* de todos os processos fora reduzido, para que fosse possível criar uma fila maior de processos prontos durante o tempo de execução do modelo, afim de demonstrar a diferença de poder de processamento e obter uma comparação.

A tabela abaixo exibe os processos utilizados no sistema multiprocessado assim como seus atributo:

Índice dos Processos	**Tempo de Entrada**	**Tempo de serviço**	**Prioridade**
P1	1	4	2
P2	2	3	1
P3	3	2	2
P4	4	3	3
P5	5	3	1
P6	1	3	4

Tabela 1 – Marcação Inicial do Modelo Multiprocessado

5.2 Modelo do Sistema

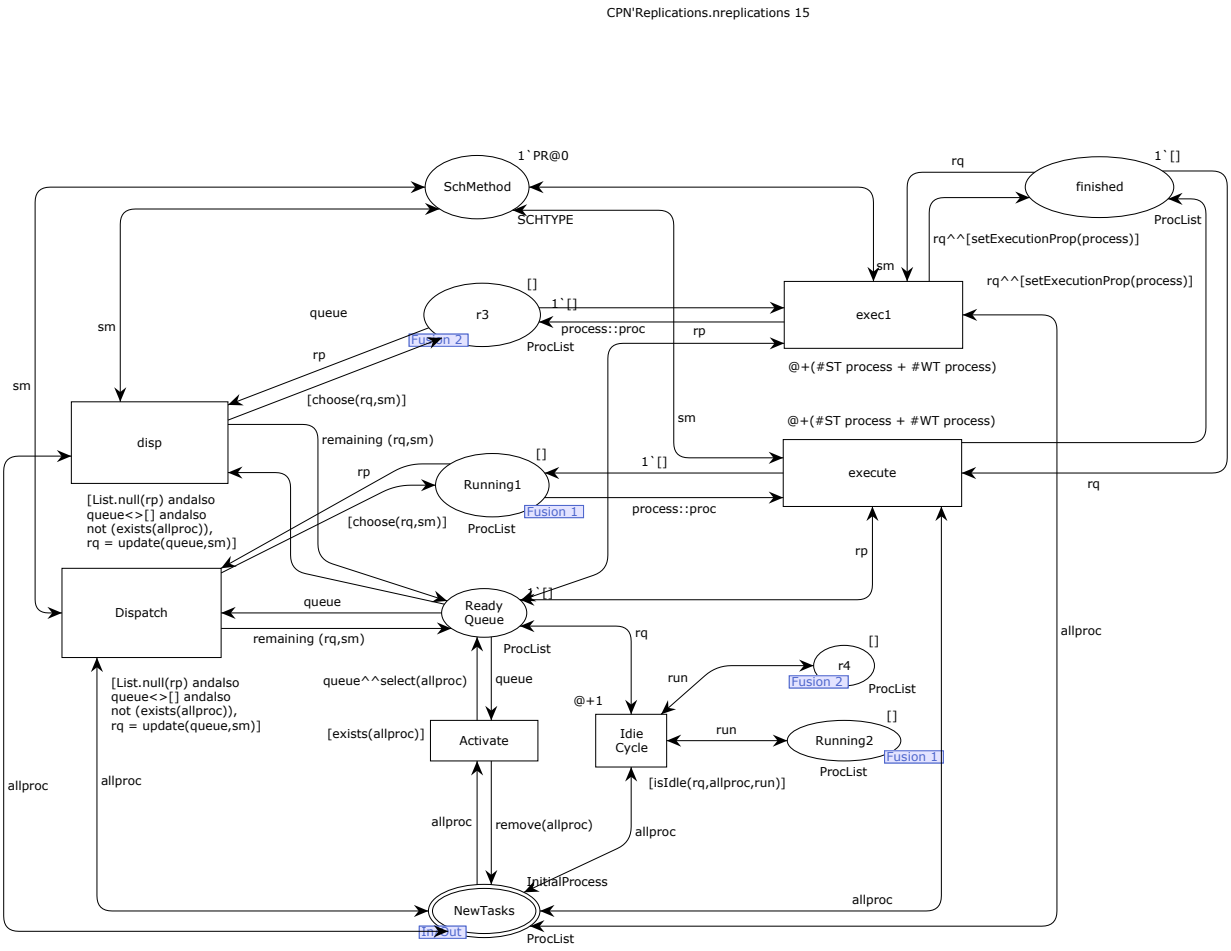

Figura 19 – *Modelo Multiprocessado*

5.3 Funcionamento do Modelo

Esta seção tem por objetivo apresentar as funções implementadas para o modelo multiprocessado e a diferença em tempo de execução dos modelos.

5.3.1 Funções do Modelo

As funções utilizadas foram as mesmas apresentadas por (PASHAZADEH; NIYARI, 2014), exceto pelas funções *select* e *remove* que serão explicadas abaixo.

```
fun select(Li:ProcList) : ProcList =
    let val n = List.length(Li)
        val L = ref []
        val i = ref 0
    in
```

```
        while !i< n do (
            let val p = List.nth(Li,!i)
            in if onTime(p) then
                    if List.length(!L) = 0 then
                        L :=[p]
                    else L := !L ^^ [p]
                else ()
            end;
            i := !i + 1 );      (* while i *)
        !L
End
```

A função *select* recebe como entrada uma lista de processos e produz uma lista de processos como saída. Ela percorre todos os elementos dessa lista, para cada elemento a função *onTIme* é chamada para verificar se o tempo de entrada do processo foi atingido baseado no tempo atual de simulação, caso atingido, ele é adicionado à lista de saída.

```
fun remove (Li:ProcList) : ProcList =
    let val n = List.length(Li)
        val L = ref[]
        val i = ref 0
    in
    while !i< n do (
        let val p = List.nth(Li,!i)
        in if onTime(p)=false then
                if List.length(!L) = 0 then
                    L :=[p]
                else L := !L ^^ [p]
            else ()
        end;
        i := !i + 1 );    (* while i *)
    !L
End
```

A função *remove* recebe uma lista de processos como entrada e produz uma lista de processos como saída. Ela checa todos os processos da lista de entrada chamando a função *onTime*. Caso algum processo não tenha atingido o seu tempo de entrada baseado no tempo atual de simulação, será adicionado à lista de saída.

5.3.2 Operação Detalhada do Modelo

Ao iniciar a simulação do modelo, assim como no modelo monoprocessado, todo os processos estarão no lugar *NewTasks*. Todos os processos terão seu atributo *input time* checados pela função *exists*, quando alcançado, ou seja, pronto para entrar no sistema e ser executado, eles irão para o lugar *Ready Queue*. Então a diferença entre os dois modelos emerge.

Com a introdução do processador no modelo, os processos que estão prontos para serem executados, serão escolhidos de acordo com a função *choose* e então serão designados arbitrariamente pela ferramenta aos núcleos de processamento. Portando essa parte da simulação pode ocorrer de duas maneiras diferentes explicitadas abaixo:

1. Dois processos diferentes podem ser executados em um núcleo, enquanto apenas um processo será executado no outro núcleo
2. Os processos podem ser executados alternadamente nos núcleos

Assim como no modelo monoprocessado, apenas um processo poderá ser computado por vez. O resto da simulação ocorre da mesma maneira detalhada na seção 4.5.

6 RESULTADOS

Para que a comparação fosse feita, o modelo monoprocessado recebeu as mesmas marcações iniciais definidas anteriormente na Tabela 1. Ambos os modelos foram replicados quinze vezes.

É importante ressaltar que, diferentemente do modelo proposto por (PASHAZADEH; NIYARI, 2014), o modelo multiprocessado não computa o mesmo tempo durante as replicações, portanto os resultados obtidos são a média aritmética das replicações, resultados que são mostrado na tabela a seguir:

Método de Escalonamento	**Multiprocessado**	**Monoprocessado**
	Tempo final de simulação	
SJF	76	115
HRRN	88	184
PR	82	132
FCFS	98	192

Tabela 2 – Comparação dos Tempos Finais

O lugar *Finished* também sofreu alterações, passando a computar o tempo de espera somado ao tempo de serviço dos processos. Essa mudança se deu devido ao modelo proposto por (PASHAZADEH; NIYARI, 2014) computar apenas o tempo de serviço do processos, tornando o resultado igual toda vez que simulado.

7 CONCLUSÕES E TRABALHOS FUTUROS

Com o intuito de introduzir os avanços tecnológicos no ambiente empresarial e ajudar as empresas a definir uma estratégia global de operação, esse trabalho apresentou um modelo multiprocessado não preemptivo construído utilizando redes de Petri colorida e a ferramenta CPN Tools.

Ao utilizar o CPN Tools, foi possível combinar o poder de representação provido pelas redes de Petri juntamente com a linguagem de programação ML, obtendo um diferentes tipos de dados e uma grande abstração dos mesmos. Para simplificação do modelo e aumentar a capacidade de abstração, foram usados diferentes tipos de *color sets* complexos.

Após a comparação dos resultados obtidos, pode-se observar que ao introduzir um núcleo de processamento, houve uma diminuição de aproximadamente 56% do tempo final. Isso mostra que é capaz de se obter um grande avanço com essa mudança, ajudando a introduzir uma estratégia corporativa melhor e aperfeiçoar os resultados no ambiente dos negócios.

Como trabalho futuro, os autores farão a validação do modelo empregando a ferramenta *State Space*, que permite a extração de relatórios que após serem analisados, permitem saber se o modelo corresponde às especificações desejadas.

REFERÊNCIAS

BROWN, T. The value of enterprise architecture. *Zachman Institute for Framework Advancement (ZIFA), www. zifa. com*, 2004. Citado na página 13.

CARDOSO, J.; VALETTE, R. *Redes de petri*. [S.l.]: Editora da UFSC, 1997. Citado 2 vezes nas páginas 19 e 20.

CHEN, P. P.-S. The entity-relationship model—toward a unified view of data. In: *Readings in artificial intelligence and databases*. [S.l.]: Elsevier, 1988. p. 98–111. Citado na página 16.

CPNTOOLS. *Documentations*. 2018. <http://cpntools.org/2018/01/16/documentation-2/>. Acessado em 05/07/2018. Citado 2 vezes nas páginas 28 e 29.

ELZINGA, D. J. et al. Business process management: survey and methodology. *IEEE transactions on engineering management*, IEEE, v. 42, n. 2, p. 119–128, 1995. Citado 2 vezes nas páginas 17 e 18.

HOOGERVORST, J. Enterprise architecture: Enabling integration, agility and change. *International Journal of Cooperative Information Systems*, World Scientific, v. 13, n. 03, p. 213–233, 2004. Citado 3 vezes nas páginas 13, 17 e 33.

HULL, R.; KING, R. Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys (CSUR)*, ACM, v. 19, n. 3, p. 201–260, 1987. Citado na página 16.

JENSEN, K.; KRISTENSEN, L. M. *Coloured Petri nets: modelling and validation of concurrent systems*. [S.l.]: Springer Science & Business Media, 2009. Citado 2 vezes nas páginas 25 e 30.

KLEM, L. F.; FRANTZ, F. R. Análise do comportamento de uma solução de integração por meio de simulação. *Salão do Conhecimento*, v. 2, n. 2, 2016. Citado 3 vezes nas páginas 9, 23 e 24.

LAPALME, J. Three schools of thought on enterprise architecture. *IT professional*, IEEE, v. 14, n. 6, p. 37–43, 2012. Citado na página 13.

MARCOS, E.; VELA, B.; CAVERO, J. M. A methodological approach for object-relational database design using uml. *Software and Systems Modeling*, Springer, v. 2, n. 1, p. 59–72, 2003. Citado na página 16.

MARRANGHELLO, N. Redes de petri: Conceitos e aplicaçoes. *São Paulo: DCCE/IBILCE/UNESP*, 2005. Citado na página 23.

MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, IEEE, v. 77, n. 4, p. 541–580, 1989. Citado 2 vezes nas páginas 19 e 20.

NOAKES-FRY, K.; DIAMOND, T. Business continuity and disaster recovery planning and management: Technology overview. *Gartner, Inc.(11 July 2003)*, v. 2, 2003. Citado 2 vezes nas páginas 18 e 19.

OLIVEIRA;MORAIS. *Jantar dos Filósofos*. 2011. <http://di.uern.br/cpnsamples/?page_id=287>. Acessado em 11/07/2018. Citado 2 vezes nas páginas 9 e 30.

PASHAZADEH, S.; NIYARI, E. A. Modeling enterprise architecture using timed colored petri net: Single processor scheduling. *arXiv preprint arXiv:1404.2939*, 2014. Citado 6 vezes nas páginas 14, 34, 36, 38, 46 e 49.

PENHA, D. O. d.; FREITAS, H.; MARTINS, C. Modelagem de sistemas computacionais usando redes de petri: aplicação em projeto, análise e avaliação. *Pontifícia Universidade Católica de Minas Gerais, Relatório técnico*, 2004. Citado 3 vezes nas páginas 9, 24 e 25.

PERES, D. B. R. *Threads, Semáforos e Deadlocks - O Jantar dos Filosofos*. 2013. <https://www.revista-programar.info/artigos/threads-semaforos-e-deadlocks-o-jantar-dos-filosofos/2/>. Acessado em 03/07/2018. Citado na página 24.

ROSS, J. W.; WEILL, P.; ROBERTSON, D. *Enterprise architecture as strategy: Creating a foundation for business execution*. [S.l.]: Harvard Business Press, 2006. Citado 2 vezes nas páginas 16 e 33.

RUMBAUGH, J. et al. *Object-oriented modeling and design*. [S.l.]: Prentice-hall Englewood Cliffs, NJ, 1991. v. 199. Citado na página 17.

SMITH, H.; FINGAR, P. *Business process management: the third wave*. [S.l.]: Meghan-Kiffer Press Tampa, 2003. v. 1. Citado na página 18.

WEILL, P.; BROADBENT, M. *Leveraging the new infrastructure: how market leaders capitalize on information technology*. [S.l.]: Harvard Business Press, 1998. Citado na página 18.

YU, L. et al. Quantitative modeling of power performance tradeoffs on extreme scale systems. *Journal of Parallel and Distributed Computing*, Elsevier, v. 84, p. 1–14, 2015. Citado na página 34.

ZACHMAN, J. A. Enterprise architecture: The issue of the century. *Database Programming and Design*, v. 10, n. 3, p. 44–53, 1997. Citado na página 13.