UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ESTRATÉGIA DE MODELAGEM PARA CONTROLE DE FMS COMBINANDO REDES DE PETRI E UM SISTEMA FUZZY

FELIPE BEZERRA DOS REIS

ORIENTADOR: PROF. DR. ORIDES MORANDIN JUNIOR

São Carlos - SP

Julho/2012

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ESTRATÉGIA DE MODELAGEM PARA CONTROLE DE FMS COMBINANDO REDES DE PETRI E UM SISTEMA FUZZY

FELIPE BEZERRA DOS REIS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Inteligência Artificial.

Orientador: Dr. Orides Morandin Junior.

São Carlos - SP

Julho/2012

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

R375em

Reis, Felipe Bezerra dos.
Estratégia de modelagem para controle de FMS combinando redes de Petri e um sistema Fuzzy / Felipe Bezerra dos Reis. -- São Carlos : UFSCar, 2014.
142 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2014.

1. Sistemas. 2. Redes de Petri colorida. 3. Sistemas flexíveis de fabricação. 4. Controle de FMS. 5. Modelagem de FMS. I. Título.

CDD: 003.7 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

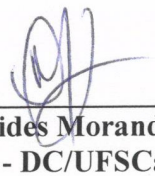
Programa de Pós-Graduação em Ciência da Computação

"Estratégia de Modelagem para Controle de FMS combinando Redes de Petri e um Sistema Fuzzy"

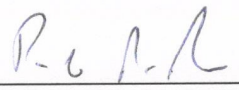
Felipe Bezerra dos Reis

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

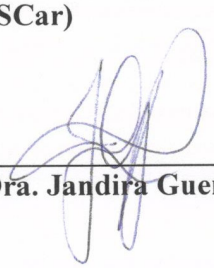
Membros da Banca:



Prof. Dr. Orides Morandin Junior
(Orientador - DC/UFSCar)



Prof. Dr. Paulo Rogério Politano
(DC/UFSCar)



Profa. Dra. Jandira Guenka Palma
(UEL)

São Carlos
Setembro/2012

Aos próximos.

AGRADECIMENTO

Todos os resultados trazidos por este trabalho vieram em função de pessoas que direta ou indiretamente contribuíram para que o objetivo fosse alcançado. Minha família tem um papel fundamental nesse desafio que foi vencido. Um dos caras que fizeram mágica para que o trabalho fosse realizado é meu pai Walter Bezerra dos Reis. A ele fica aqui o meu muito obrigado de coração, pra sempre trago comigo o seu carinho e respeito. Agradeço a serenidade em pessoa que é a minha mãe Maria do Socorro Rodrigues Reis. Os dois me educaram no sentido de equilibrar a minha vontade e loucura de fazer todas as coisas, e o respeito que a vida exige de nós.

Opa! Felipe, meu caro!!! Esse é um dos bordões clássicos do Orides Morandin Junior, meu orientador e uma pessoa espetacular. Me admira a sua política e tratamento. Um cara que me guiou com muita atenção até o final deste trabalho.

Tenho muito orgulho de ter sido morador das repúblicas Vopoulos Nopopoulos, e QBLZ. Todas as festas, todos os problemas, todas as contas divididas fizeram de nós amigos irmãos.

Raça São Carlos! Treinar com o time de polo aquático da UFSCar me trouxe a persistência e garra que eu precisava para terminar este e trabalho. A raça que talvez só teria ganho fazendo parte do exército eu consegui adquirir treinando com esses caras. Grandes amigos.

A todas as pessoas que fizeram parte da minha vida até então dedico os meus agradecimentos e espero que nos vejamos em outros desafios.

"Se, a princípio, a ideia não é absurda, então não há esperança para ela."

Albert Einstein

RESUMO

Os sistemas flexíveis de manufatura (FMS) são sistemas de produção automatizados capazes de produzir uma grande variedade de tipos de produto. Eles são compostos por máquinas, robôs e sistemas de transporte automático. Realizar o controle de um FMS é uma tarefa complexa devido aos vários subsistemas e elementos que o compõe, e também pela necessidade de responder a questões estratégicas que variam de acordo com as demandas do mercado. Muito tem sido feito em relação ao controle de FMSs. Vários trabalhos na literatura abrangem as questões de controle de FMSs. Apesar do alto número de trabalhos que abordam estas questões, os avanços tecnológicos e as exigências de mercado fazem com que novos desafios sejam lançados e novos estudos sejam realizados. Este trabalho se baseia em outros trabalhos sobre o controle de FMS para evoluir em algumas questões de modelagem e controle de FMSs. A proposta deste trabalho é indicar uma estratégia de modelagem baseada em redes de Petri que seja de fácil compreensão, que exija baixos esforços de adaptação para planos de produção diferentes, e que permita que o controle do modelo de FMS leve em consideração informações do estado corrente do sistema, nas tomadas de decisão para a resolução de conflitos. O trabalho é finalizado com a aplicação do método de modelagem proposto em um FMS hipotético. Foi construído um sistema de controle para fazer a leitura de variáveis do modelo de FMS e decidir como os conflitos do modelo em redes de Petri devem ser resolvidos. Para apoiar as tomadas de decisão do sistema de controle foi construído um sistema *Fuzzy*. Foram realizadas simulações para uma variedade de planos de produção nos quais o sistema de controle obteve respostas em tempo real para a resolução dos conflitos impedindo que o sistema atingisse condições de *deadlock*.

Palavras-chave: FMS, Redes de Petri Colorida, Controle de FMS, Modelagem de FMS.

ABSTRACT

The flexible manufacturing systems (FMS) are automated production systems capable of producing a wide variety of product types. They are composed by machines, robots and automated transport systems. Make the control of an FMS is a complex task due to the various subsystems and elements that compose it, and also by the necessity of respond to strategic issues that vary according to market demands. Much has been made in relation to the control of FMS. Several studies in the literature cover issues of control of FMS. Despite the high number of papers that address these issues, technological advances and market demands cause introduction of new challenges and new studies are conducted. This work based itself on other studies about the control of FMS to evolve into some issues of modeling and control of FMS. The purpose of this paper is to indicate a modeling strategy, based on Petri nets, that is easy to understand, requiring low efforts to adapt to different production plans, and allowing that the control of the FMS model takes into account information of the current state of the system in decision making for conflict resolution. The work ends with the application of the modeling method proposed in a hypothetical FMS. It was built a control system to make the reading of variables from the model of the FMS and decide how conflicts should be resolved on the Petri net model. To support the decision making of the control system, it was built a *Fuzzy* system. Simulations were conducted for a variety of production plans in which the control system achieved real-time responses to conflict resolution, preventing the system of reach deadlock conditions..

Keywords: FMS, Colored Petri Net, FMS Control, FMS Modeling.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AGV – *Veículos Autoguiados*

AMS – *Automated Manufacturing System*

ERP – *Enterprise Resource Planning*

FCPN – *Fuzzy Coloured Petri Net*

LAN – *Local Area Network*

MES – *Manufacturing Execution System*

MRP – *Material Requirement Planning*

PCP – *Planejamento e Controle da Produção*

POPN – *Process Oriented Petri Net*

ROPN – *Resource Oriented Petri Nets*

VPN – *Virtual Petri Nets*

DLL – *Dynamic Link Library*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Contextualização

Este trabalho dedicou esforços para avançar na modelagem e controle de sistemas de manufatura, tendo como ponto de partida os trabalhos (MORANDIN e KATO, 2003), (ARAÚJO, 2006), e (SGAVIOLI, 2010).

Estes trabalhos evoluíram em relação aos problemas de modelagem e controle de sistemas de manufatura, e responderam questões importantes levantadas pelos estudos relacionados a este tema. Apesar desta evolução, muito ainda precisa ser feito neste campo, pois ainda há questões a serem estudadas e respondidas, haja a vista que a modelagem e controle de sistemas de manufatura é um campo vasto a ser estudado e que a cada dia impõe novos desafios conforme as demandas de mercado crescem e tomam novos formatos.

O trabalho presente se baseia em questões, e desafios propostos pelos trabalhos citados anteriormente. Estas questões são a respeito de estratégias de modelagem e controle de sistemas de manufatura.

1.1.1 Sistemas Flexíveis de Manufatura

No contexto atual da economia, as indústrias de manufatura enfrentam um desafio crucial: aumentar a capacidade produtiva de seus sistemas para atender a demandas voláteis e ciclos de produção cada vez mais curtos. Os Sistemas Flexíveis de

Manufatura (FMS) proveem esta capacidade empregando automação programável para processamento e manuseio de material. Para que estes benefícios sejam plenamente extraídos, é necessário lidar com os problemas relacionados a modelagem e controle de FMSs.

As pesquisas em FMSs aparecem com frequência na literatura evidenciando a importância desta tecnologia para a indústria.

Em (SLACK, CHAMBERS e JOHNSTON, 1999) é observado que um FMS é mais do que uma tecnologia, um FMS tem tecnologias integradas em um sistema, que tem o potencial para ser melhor do que a soma de suas partes. Com efeito, um FMS é capaz de manufaturar um componente completo do inicio ao fim. Ainda em (SLACK, CHAMBERS e JOHNSTON, 1999) afirma-se que um FMS pode ser definido como uma configuração, controlada por computador, de estações de trabalho semi-independentes conectadas por unidades de manuseio de materiais e carregamento de máquinas automatizadas.

Em (KUO e HUANG, 2000) é dito que um FMS consiste em um número de sistemas de produção como ações, processos, dispositivos de manuseio de material, estocagem de materiais, unidades de controle, estações de inspeção e de medição, e mensagens de controle da produção incluindo informações do processo e informações sobre o produto. Os comandos de controle são roteados por meio de um sistema de comunicação, que consiste de um computador, uma unidade de controle e uma área local rede (LAN).

Em (TUNG, LIN e NAGI, 1999) os FMSs são apresentados como sistemas de produção automatizados capazes de produzir uma grande variedade de tipos de produtos. Este aumento de flexibilidade é devido ao uso de grupos de máquinas fisicamente próximas e dedicadas a produzir vários tipos de peças para uma família de produtos que compartilham operações comuns nestas máquinas.

Os FMSs estão inseridos dentro de um grande grupo chamado sistemas de manufatura automatizados (AMS). Consoante ao que é dito em (WU, ZHOU e HU,

2007), um AMS consiste em máquinas e um sistema de manuseio de materiais automático.

O fator flexibilidade dos FMSs é importante por ser ele o responsável pela atenção que a indústria tem dado para estes sistemas, haja vista a grande produtividade que os FMSs podem oferecer. Isto atrai investimentos para os projetos e gestão desta classe de sistemas. No entanto, tal flexibilidade atribui características de sistemas complexos aos FMSs que dificultam a sua modelagem e controle.

1.1.2 Controle

O controle automático representa um papel vital no avanço da engenharia e da ciência e tornou-se parte integrante e importante dos processos industriais e de manufatura modernos (OGATA, 1982).

Os sistemas de controle estão presentes na maioria dos processos industriais modernos. Em (LEME, 1974) é explicado que o controle corresponde a seguinte sucessão de ações:

- Considerar o que foi planejado;
- Considerar o que foi realizado;
- Confrontar o realizado com o planejado;
- Tomar providências quando o realizado e planejado não coincidirem.

Diz-se em (LEME, 1974) que um conjunto de regras que permite o exercício do controle constitui um "sistema de controle". Um sistema de controle deve ter no mínimo:

- Um plano;
- Regras para medir o realizado;

- Um instrumento de confronto do realizado com o planejado;
- Regras de providência (opcional).

Dentro da engenharia de controle e automação entende-se que a filosofia básica de um sistema de controle é unir o resultado da leitura dos elementos sensores com a ação dos elementos atuadores. Os sistemas de controle recebem as informações lidas dos sensores para saber o atual estado do processo, executam cálculos e lógicas pré-definidas (também chamadas de leis de controle) e enviam o resultado para os atuadores, de modo que a situação atual do processo seja modificada para que se atinja um ponto de operação próximo do desejado.

Em (OGATA, 1982) é dito que o controle automático é essencial em operações industriais para controle de pressão, temperatura, umidade, viscosidade e fluxo em processos industriais. Pode-se ainda considerar o texto em (SLACK, CHAMBERS e JOHNSTON, 1999) o qual mostra que o controle faz os ajustes que permitem que a operação atinja os objetivos que o plano estabeleceu, mesmo que as suposições feitas pelo plano não se confirmem.

Uma das características dos FMSs é a possibilidade de um produto em fabricação poder seguir roteiros de produção diferentes com máquinas variadas, e também ser transportado por veículos distintos ao longo do chão de fábrica. A decisão de qual veículo ou máquina deverá ser utilizada para realizar uma determinada tarefa quando mais de um elemento pode fazê-la é responsabilidade do sistema de controle.

1.1.3 Chão de fábrica

Chão de fábrica é o lugar físico no qual unidades operacionais, como máquinas, executam os trabalhos de concepção física das peças que compõe os produtos. Opõe-se aos escritórios que, por sua vez, fornecem alojamento para a gestão do negócio.

No trabalho (CHOI e LEE, 2001) ressalta-se que o chão de fábrica é um ambiente dinâmico no qual eventos inesperados ocorrem continuamente e impõem mudanças as atividades planejadas. Para lidar com este problema, um chão de fábrica deve adotar um sistema de controle apropriado que seja responsável por coordenar e controlar o fluxo físico da manufatura, bem como o fluxo de informação.

Ocasionalmente encontra-se na literatura o uso da palavra planta para denotar o chão de fábrica. Em (OGATA, 1982) é mencionado que uma planta é uma parte de equipamento, eventualmente um conjunto de itens de uma máquina, que funciona conjuntamente, cuja finalidade é desempenhar uma dada operação.

Os elementos de chão de fábrica de FMSs formam um sistema complexo caracterizado pelo compartilhamento de recursos como máquinas, equipamentos e veículos de transporte além de paralelização de tarefas. Estes sistemas complexos frequentemente apresentam situações de conflito.

1.1.4 Conflito

Dois eventos estão em conflito se a ocorrência de um evento desabilita a ocorrência o outro. Conflitos estão presentes na modelagem de várias situações básicas, como exclusão mútua, semáforos e compartilhamento de recursos. Usualmente, conflitos precisam de tratamento especial e podem requerer intervenções (GOMES, 2005).

Um contexto de indecisão ou conflito pode ser exemplificado por um produto que se encontra em condição de seguir para a próxima etapa de fabricação, porém duas ou mais máquinas que realizam a mesma tarefa estão disponíveis para processar este produto. Nesse caso deve-se fazer uma escolha que pode ser aleatória, ou de acordo com alguma regra. Em uma situação como esta, um sistema de controle interessado na maximização do desempenho da produção da fábrica deverá tratar a questão de decidir qual seria a máquina mais adequada para executar a tarefa. Neste caso o sistema de controle de chão de fábrica pode receber informações do sistema de gestão da produção

e entender qual máquina irá atender melhor a demanda de produção dado o contexto global do sistema de produção.

É possível usar estas informações de forma inteligente ponderando os parâmetros que influenciam de algum modo no alcance dos objetivos de produção. Neste trabalho foram consideradas informações sobre o comportamento do próprio FMS em tempo de execução, para a resolução dos conflitos do FMS.

1.1.5 *Deadlocks*

Os *deadlocks* são situações nas quais os processos de um sistema travam e assim, ficam impossibilitados de continuar a execução do sistema. Isto pode acontecer quando as seguintes condições acontecem simultaneamente:

- Processos não preemptivos: não é possível encerrar um processo que esteja usando determinado recurso para que outro processo use este recurso;
- Exclusão mútua: apenas um processo pode usar um recurso por vez;
- Espera circular: dado um conjunto de processos $P\{p_1, p_2, p_3, ..., p_n\}$ em espera, p_1 está esperando por p_2; p_2 está esperando por p_3 e assim sucessivamente até que p_n esteja esperando por um recurso alocado por p_1;
- Monopolização de recursos: quando mais de um processo acessam um determinado recurso de exclusão mútua e nenhum processo retrocede.

Deadlocks são considerados uma questão importante pelo motivo de poderem desabilitar operações de partes de um sistema de manufatura ou até mesmo o sistema integral.

Os efeitos que as situações de *deadlocks* podem causar para um sistema de manufatura podem ser o atraso de entrega de produtos, e perda de matéria prima, entre outros.

Para operar um sistema de manufatura efetivamente, é necessário coordenar e controlar o uso de recursos compartilhados a fim de evitar *deadlocks*.

1.1.6 Sistemas de Controle de FMSs

Ao longo do processo de modernização das indústrias, marcado pelo desenvolvimento da engenharia de controle, advento da robótica industrial, aumento da eficiência de sensores entre outras tecnologias, pesquisadores propuseram trabalhos relacionados à modelagem e controle dos sistemas de produção.

A dificuldade de elaborar os projetos de layout de fábricas complexas bem como de efetuar o controle de máquinas, o despacho de Veículos Autoguiados (AGV) e gerenciamento de suas rotas, geraram demanda para que pesquisas fossem realizadas no sentido de tornar estas tarefas mais simples, e de garantir resultados mais confiáveis.

Em (CHENGEN, JIANYING e ZHONGXIN, 1993) foi construído um modelo genérico de FMS usando redes de Petri Colorida. A estrutura do modelo em redes de Petri demonstrou matematicamente o comportamento dinâmico do FMS modelado. Um sistema especialista foi desenvolvido para o controle do FMS baseado no modelo em redes de Petri. O sistema especialista apresentou alta flexibilidade e pôde ser usado para controlar uma variedade de FMSs por meio da construção de base de dados distintas.

O trabalho (LEDUC, LAWFORD e DAI, 2006) demonstrou como é possível obter ganhos significantes do sistema aplicando uma arquitetura hierárquica, modular e baseada em interfaces na construção do sistema supervisório. O trabalho provê uma descrição detalhada de como a teoria pode ser aplicada para projetar e verificar células de FMS. O modelo utilizado para verificação da proposta foi baseado em uma célula de FMS específica.

Em (LEITAO e RESTIVO, 2008) é apresentado uma arquitetura de controle baseada em sistemas holônicos adaptativos. O objetivo foi aumentar a agilidade e capacidade de reconfiguração do sistema de produção. O trabalho descreve o desenvolvimento e a validação experimental de um sistema de controle de manufatura baseado em sistemas holônicos adaptativos, em um modelo de FMS específico usando tecnologia de sistemas multiagentes.

O trabalho (BANNAT, BAUTZE, *et al.*, 2011) propõe uma abordagem baseada em sistemas técnicos cognitivos para solucionar uma série de problemas relacionados aos desafios que são encontrados nas fábricas modernas, em especial as automobilísticas. Verificou-se que os sistemas técnicos cognitivos podem ser usados para o planejamento e controle da produção, usinagem autônoma, programação automatizada de robôs industriais, garantia de qualidade e controle de processos, além de sistemas guia para operadores em montagens manuais.

Em (MORANDIN e KATO, 2003) realizou-se um trabalho no qual foi proposto um método modular de modelagem para o planejamento e controle de tarefas de FMSs baseado em redes de Petri Virtuais. A contribuição deste trabalho se deu em relação à diminuição da dificuldade de modelar sistemas muito grandes ou complexos, e também, a diminuição da dificuldade de compreensão e análise do modelo quando os sistemas têm uma grande quantidade de elementos. Para o teste da proposta foi feita a modelagem de um FMS especifico. A proposta foi verificada via simulação deste modelo.

Em (ARAÚJO, 2006) foi proposta uma estratégia de modelagem para o controle e intertravamento de AMSs usando redes de Petri Virtuais. Este trabalho se baseou no trabalho realizado em (MORANDIN e KATO, 2003) para avançar nas questões de controle de sistemas de manufatura. Neste trabalho foi feita uma modelagem modular baseada em redes de Petri Virtuais de um FMS. No modelo foi incorporado o controle do sistema. Esta incorporação do controle do sistema no modelo representou o avanço em relação ao trabalho (MORANDIN e KATO, 2003). O modelo foi simulado para vários planos de produção com cinco tipos de produtos, cada um com mais de um roteiro de produção. O modelo realizou o controle do sistema garantindo a ausência de

deadlocks. Uma desvantagem deste trabalho foi o aumento da complexidade de compreensão ou leitura do modelo. Outra desvantagem é que os roteiros de produção foram fixados no modelo. Isto aumenta a dificuldade de alteração para que o mesmo possa comportar outros tipos de produtos ou diferentes roteiros de produção. Além disto, o controle deste modelo não consegue reagir, por exemplo, a uma quebra de máquina. Caso isso aconteça todos os roteiros de produção dependentes desta máquina não serão completados. Outra questão que pode ser levantada em relação a este trabalho é que o mesmo não leva em conta informações ou variáveis do sistema durante a realização do controle, para aumentar a eficiência do sistema de manufatura.

No trabalho (SGAVIOLI, 2010) foi apresentada uma modelagem de sistema de manufatura usando redes de Petri Colorida *Fuzzy* que soluciona conflitos do sistema considerando informações do sistema em tempo real. A característica de usar variáveis do sistema para solucionar conflitos em tempo real cobre uma das desvantagens do trabalho (ARAÚJO, 2006). Neste trabalho o controle é realizado de forma inteligente, pois considera informações do sistema como, por exemplo, a quantidade de produtos em *buffer*. Isto representa uma vantagem em relação ao trabalho (ARAÚJO, 2006). Uma desvantagem do trabalho (SGAVIOLI, 2010) é o aumento da complexidade de compreensão ou leitura do modelo. Outra desvantagem é a dificuldade de adequar este modelo para outros planos de produção. A configuração deste modelo em sistemas reais exige equipamentos de processamento de controle de alta capacidade. Isto também é uma desvantagem.

O presente trabalho tem uma contribuição pautada na cobertura de algumas desvantagens do trabalho de (ARAÚJO, 2006), a saber:

- A dificuldade de compreensão do modelo;
- A dificuldade de adequação do modelo para novos produtos e novos planos de produção;
- Não levar em conta informações em tempo real do sistema para realização do controle.

O esforço realizado para evoluir nestas questões forma a base dos objetivos deste trabalho.

1.2 Objetivos

Dando continuidade ao que tem sido realizado a respeito do controle de sistemas de manufatura, este trabalho realizou a construção de um sistema de controle de um modelo de um FMS. Tal sistema foi baseado em um FMS experimental que tem sido usado em vários trabalhos realizados no laboratório Tear.

O objetivo específico deste trabalho foi o desenvolvimento de um sistema de controle de um modelo de FMS que considera variáveis do sistema para realização do controle do modelo em tempo real.

O objetivo geral foi avançar no trabalho realizado em (ARAÚJO, 2006). Na proposta de (ARAÚJO, 2006) o controle é realizado e o sistema é livre de *deadlocks*. No entanto, não são consideradas informações ou variáveis do sistema como, quantidade de produtos nos *buffers*, e distância percorrida pelos AGVs.

Algumas mudanças foram feitas na estratégia de modelagem proposta em (ARAÚJO, 2006) a fim de diminuir a complexidade da modelagem do sistema e permitir que o controle fosse realizado por um sistema de controle externo ao modelo. Outra razão para as alterações na estratégia de modelagem de (ARAÚJO, 2006) é facilitar a adequação do modelo para novos produtos e novos planos de produção.

A estratégia de modelagem proposta em (ARAÚJO, 2006) também foi usada em (SGAVIOLI, 2010).

1.3 Justificativa e Motivação

No final do século passado houve um investimento notório em computadores, software e em tecnologia para aperfeiçoamento destes. Após esta onda a indústria dá um passo à frente concentrando esforços na integração destes sistemas, saindo de um cenário de investimento em equipamento para outro no qual a habilidade e capacidade de utilização deste parque de equipamentos seriam mais importantes que a aquisição de material com tecnologia de ponta.

Neste contexto os sistemas evoluíram e foram encontrando espaço em todos os departamentos e setores da indústria até o estágio de informatização completa da gestão da produção ou qualquer que seja a atividade da companhia.

Esses *software* eram muito grandes, sabidamente integravam todas as funções da organização e foram chamados de Sistemas de Planejamento de Recursos Empresariais ou ERP (VOLLMANN, BERRY, *et al.*, 2008).

Muitos dos sistemas ERP estavam baseados na lógica do planejamento das necessidades de material satisfeitas pelos sistemas Planejamento de Requerimento de Materiais (MRP) e na integração da fábrica que os sistemas MRPs já tinham estruturado. Contudo, os sistemas ERP muitas vezes substituíram os sistemas de Planejamento e Controle da Produção (PCP) já existentes ou foram implementados de forma paralela. O resultado é que os sistemas de PCP estão agora inseridos nos sistemas ERP em um grande número de organizações. Isso aumentou a capacidade gerenciamento eficaz e, ao mesmo tempo, aumentou a complexidade de integração exigida. Nenhuma atividade de PCP pode ignorar o alcance dos sistemas ERP (VOLLMANN, BERRY, *et al.*, 2008).

Uma grande força que criou a necessidade de reação em sistemas de planejamento e controle da produção foi a contínua descentralização da tomada de decisão para o chão de fábrica. Equipes de chão de fábrica estão organizadas para resolver problemas, administrar células de produção e melhorar a informação sobre a necessidade de processos nesse nível das operações da organização.

A globalização configura nos sistemas de produção uma mudança contínua e variável.

Em parte como consequência da internacionalização dos negócios, e em parte como reação à terceirização num momento em que as companhias estão enfocando nas suas competências essenciais, a interconectividade das empresas de produção aumentou substancialmente. A implicação disso é que as empresas estão frequentemente integradas como clientes dos seus fornecedores e integradas com os clientes aos quais elas fornecem de maneira complexa. Isso criou a necessidade de gerenciar algumas redes ou cadeias de suprimentos muito complexas. Cada vez mais, a empresa tem uma combinação de mercados industrial e de consumo que requer canais diferentes, o que também aumenta a complexidade das relações de produção. Essas relações devem ser incorporadas no sistema de PCP da empresa.

Todas estas exigências das partes envolvidas nos negócios de produção estão se tornando demanda para a tecnologia de automação dos sistemas produtivos.

Uma das respostas da tecnologia de automação para as demandas do mercado foi o conceito de FMS. A flexibilidade destes sistemas de manufatura impõe vários desafios aos sistemas de controle. Um destes desafios é a realização do controle em tempo real do sistema de manufatura.

Por este motivo, o presente trabalho dedicou-se aos problemas relacionados à modelagem para o controle de FMSs.

1.4 Estrutura do trabalho

No capítulo 2 procura-se salientar as características que são encontradas nos FMSs e os desafios que os sistemas que portam esta tecnologia enfrentam para alcançar os seus objetivos de produção. São observados os problemas relacionados à modelagem, simulação e análise dos FMSs. Além disto, são apresentadas algumas abordagens de

controle de FMSs encontradas na literatura. Por fim, são levantados trabalhos que apresentam abordagens para a definição de alocação de recursos de FMSs.

O capítulo 3 é dedicado à apresentação da proposta deste trabalho. São levantados os pontos específicos do trabalho. Neste capítulo a hipótese do trabalho é apresentada. Além disto, é apresentado, em detalhes, o esquema do FMS usado no trabalho. As variáveis consideradas na resolução dos conflitos do sistema em questão também são apresentadas e a forma com que foram tratadas também é explicada.

No capítulo 4 a validação da proposta é discutida. Neste capítulo é explorado o método de desenvolvimento do sistema de controle, os critérios de resolução de conflitos, o ambiente de validação e a metodologia. Também é abordada neste capítulo a cadeia de ferramentas usadas para o desenvolvimento tanto do modelo de FMS quanto do sistema de controle.

No capítulo 5 são feitas observações sobre a conclusão do trabalho. Neste capítulo são apresentados os resultados do trabalho e são feitos comentários a respeito dos resultados. Ainda neste capítulo, são apresentas algumas propostas para trabalhos futuros.

Este trabalho também possui um apêndice no qual são apresentadas as teorias sobre redes de Petri usadas neste trabalho.

Capítulo 2

CONTROLE DE SISTEMAS FLEXÍVEIS DE MANUFATURA

2.1 Considerações iniciais

O controle de um Sistema Flexível de Manufatura deve ser modelado de modo a satisfazer todas as restrições da linha de produção em tempo real. A metodologia de controle deve lidar primeiramente com os problemas relacionados à modelagem, simulação, análise e, finalmente com o controle em tempo real (ZHOU e VENKATESH, 1999).

Para lidar com os problemas de modelagem, simulação e análise é geralmente realizada a construção dos modelos do FMS e em seguida definida a estimativa das medidas de desempenho tais como utilização de robôs e máquinas, tamanho da fila de entrada das máquinas, entre outras. O propósito desta atividade é sugerir uma configuração ótima do FMS para as especificações necessárias, incluindo o arranjo físico, as rotas dos AGVs através das máquinas, as políticas de programação e sincronização das máquinas, entre outras tarefas (ZHOU e VENKATESH, 1999).

Deve-se também tratar o controle em tempo real do FMS, que tipicamente envolve coordenação, programação em tempo real, alocação e monitoramento dos

recursos. O objetivo do controle é manter uma alta utilização do sistema assim como satisfazer os prazos de produção (ZHOU e VENKATESH, 1999).

Segundo (GROOVER, 2007), as funções desempenhadas pelo controle de um FMS incluem as decisões para a combinação e avaliação da entrada de várias peças no sistema. São decisões baseadas de acordo com os dados do sistema, tais como índices de produção desejados e matéria-prima disponível. Decisões envolvendo o tráfego e o estado dos AGVs também devem ser regularizadas pelo controle do sistema. Como alguns caminhos no sistema permitem somente o trânsito de um veículo, o movimento destes deve ser controlado.

Outra função do controle é satisfazer algumas especificações como evitar *deadlocks,* considerar as relações de precedência de operações, limites dos *buffers*, e resolução de conflitos, entre outros problemas (GOLMAKANI, MILLS e BENHABIB, 2003).

A tarefa de programação da produção está relacionada com a tarefa de controle. Juntas estas tarefas governam a alocação dos recursos do sistema durante sua execução.

2.2 Abordagens para definição de alocação de recursos do sistema

2.2.1 Programação dinâmica

Programação da produção é um processo de tomada de decisão relacionado à alocação de recursos limitados entre as tarefas ao longo do tempo (TUNCEL, 2007). Esta tem como objetivo maximizar a taxa de produção e minimizar o tempo total do fluxo de operações através da ordenação de entrada dos lotes que serão executados na produção (AGUIRRE, 2007).

Enquanto a flexibilidade de um FMS possibilita um grande número de escolha de recursos e roteiros e uma maior produtividade, isso impõe um desafio à alocação dos

recursos em diferentes processos na produção de cada produto (LEE e DICESARE, 1994).

Um sistema com flexibilidade de roteiros de produção torna possível a produção de um produto por meio de vários roteiros alternativos. Este tipo de flexibilidade torna possível balancear a carga de trabalho e proporciona uma melhor sincronização das unidades operacionais, reduzindo a probabilidade de ócio na linha de produção quando houver empecilhos, como quebra ou manutenção das máquinas (CHANG, 2007).

Fica evidente que o desempenho do sistema é altamente dependente da seleção de uma eficiente política de programação para ser usada no controle do sistema. Para utilizar toda a capacidade do sistema, o sistema de controle tem que estar habilitado a cooperar com as mudanças no estado do chão de fábrica e mudanças nos objetivos operacionais do sistema.

Várias abordagens de programação aparecem na literatura. Neste trabalho é destacada a abordagem de programação dinâmica. Esta é uma representante das abordagens que consideram o estado atual do sistema e, portanto, consegue dar bons resultados mesmo com eventos inesperados do chão de fábrica como, por exemplo, uma quebra de máquina.

Programação dinâmica (também chamada de adaptativa ou online) não cria ou modifica uma programação da produção. Ao invés, métodos descentralizados de controle da produção despacham trabalhos quando necessário e usam informações disponíveis no momento do despacho. Estas abordagens usam regras de despacho e heurísticas para priorizar trabalhos que estejam esperando para serem processados por um recurso (VIEIRA, HERRMANN e LIN, 2003).

Outra abordagem de programação da produção é a programação reativa. A programação reativa é uma estratégia comum de reprogramação. Esta abordagem tem dois passos. Primeiramente, uma programação da produção estática é gerada. No próximo passo ela é atualizada, revisada e modificada em resposta as interrupções e

outros eventos para minimizar o impacto no desempenho do sistema (VIEIRA, HERRMANN e LIN, 2003).

As abordagens que usam somente programação estática usualmente resolvem o problema por abordagens ótimas, mas podem facilmente se tornarem inviáveis em um ambiente de manufatura real, já que assumem algumas suposições não realísticas (LEE, 2008). No contexto da programação estática, uma mudança no planejamento da produção ou falha nos recursos implica uma parcial ou total reconfiguração do sistema.

A programação dinâmica tem suas vantagens sobre a programação estática. Em um sistema de manufatura, uma programação estática pode se tornar obsoleta após um pequeno período de tempo, enquanto a programação dinâmica pode se adaptar as mudanças que ocorrem no sistema (OUELHADJ e PETROVIC, 2009), (SINREICH e SHNITS, 2006).

Programação dinâmica tem sido resolvida usando as seguintes técnicas: heurísticas, meta-heurísticas, sistemas baseados em conhecimento, lógica *Fuzzy*, redes neurais e sistemas multiagentes (OUELHADJ e PETROVIC, 2009).

Os trabalhos (TURGAY, 2009) e (XIANG e LEE, 2008) usam sistemas baseados em agentes para resolver o problema do controle e da programação dinâmica.

No trabalho (TURGAY, 2009) a modelagem de um sistema de controle baseado em agentes é representada por meio de redes de Petri e o desempenho é avaliado por meio do tempo associado aos lugares da rede. O objetivo é gerar uma programação eficiente para o processamento dos produtos. O controle de um sistema baseado em agentes é um mecanismo que monitora continuamente o estado e as condições do sistema e realiza decisões relacionadas à produção. Decisões como qual peça será processada e em quanto tempo, o robô ou o AGV que será usado e, qual plano de processamento será implementado com qual sequência de operações, são realizadas pelo agente.

Primeiramente, os dados e a base de regras são determinados considerando os componentes. Agentes independentes são considerados de acordo com o recurso do

sistema como agente supervisor, agente de peças, agente das máquinas, agente do robô e agente do AGV. O agente supervisor é o mecanismo que realiza as decisões. Esse agente decide, por exemplo, qual peça será processada e qual agente será usado no processamento.

O trabalho (XIANG e LEE, 2008) tem como objetivo construir uma programação dinâmica eficiente baseada em agentes para um sistema de manufatura. Um modelo baseado no algoritmo de colônia de formigas é proposto para ser combinado com um agente local para otimizar o desempenho do sistema baseado em alguns critérios. Os agentes do sistema foram modelados de forma autônoma de acordo com o conhecimento relacionado às suas funções e objetivos, como agente do chão de fábrica, agente do pedido de produção, agente das máquinas, entre outros. Diferentes agentes se comunicam e coordenam suas atividades.

Os trabalhos (LEE, 2008), (BILGE, FıRAT e ALBEY, 2008), (CAPRIHAN, KUMAR e STECKE, 2006), (CHAN, CHAN e KAZEROONI, 2003) e (SRINOI, SHAYAN e GHOTB, 2002) usam abordagens *Fuzzy* para solução de conflitos e seleção de roteiros.

O trabalho (LEE, 2008) propõe um método para extrair regras *Fuzzy* automaticamente de uma base de dados que é continuamente atualizada para resolver problemas de programação adaptativa. Uma base de regras é construída de acordo com um classificador de aprendizado dinâmico *Fuzzy* baseado em treinamento de dados acumulados por um método de simulação.

O método engloba um esquema para aquisição automática de conhecimento para construir uma base de regras *Fuzzy*. A simulação é usada para avaliar o desempenho das regras candidatas. Em um ponto de decisão, cada regra de despacho é avaliada e, a regra com melhor desempenho é adicionada na base. A decisão de despacho é selecionar uma tarefa entre àquelas que estão esperando para ser processada. Quando uma estação acaba de processar um peça, esta estação deve determinar qual peça será processada a seguir entre aquelas que esperam. Para a escolha do AGV a regra é escolher o mais próximo.

O trabalho (BILGE, FıRAT e ALBEY, 2008) desenvolve uma estratégia para seleção de roteiros durante o controle em tempo real de um FMS. Este trabalho adota uma abordagem em lógica *Fuzzy* para gerar uma relação de critérios. São apresentados três algoritmos que incorporam os três critérios em diferentes maneiras. Tempo estimado de término, tempo mínimo estimado de fluxo para cada alternativa e um algoritmo *Fuzzy*.

O trabalho (CAPRIHAN, KUMAR e STECKE, 2006) apresenta uma estratégia de despacho baseado em lógica *Fuzzy* para cooperar com os atrasos de informações. O sistema apresenta flexibilidade de roteiros. Assim, após cada operação uma decisão de controle *on-line* é solicitada para escalar uma máquina apropriada, para a qual a peça deve ser despachada. As variáveis para a decisão de despacho são primeiramente identificadas. Para cada alternativa de despacho, o grau de verdade é avaliado e a alternativa com maior grau é executada.

O trabalho (CHAN, CHAN e KAZEROONI, 2003) apresenta um sistema *Fuzzy* de tempo real para programação da produção em um FMS. A abordagem em lógica *Fuzzy* é proposta para melhorar o desempenho considerando várias medidas, como o número de máquinas bloqueadas, o tempo total de processamento, o número de etapas de processamento e a média de tempo de fluxo.

Sua abordagem foca no estado do sistema para atribuir prioridades as peças que estão esperando para ser processadas. Um modelo de simulação do FMS é construído, e em vários pontos do sistema, pontos de decisão são considerados para verificar o seu estado. Quando um AGV está livre, ele deve voltar à área de estacionamento. Assim, passa por vários pontos de interseção onde verifica solicitações. Se existe alguma solicitação para ele, ele vai até a estação que o requisitou.

O trabalho (SRINOI, SHAYAN e GHOTB, 2002) apresenta um modelo de programação baseado em *Fuzzy* para fazer o controle em tempo real para solucionar problemas da programação. O programador *Fuzzy* deve decidir qual a melhor rota entre as alternativas será selecionada para processar uma peça.

A seleção da rota depende de três fatores, o número de peças esperando no *buffer* de entrada de cada máquina considerando seu tempo de processamento total, o tempo restante para completar a operação requisitada e o tempo de viagem da peça através da rota escolhida.

Os trabalhos (TUNCEL, 2007), (ZHANG, JIANG e GUO, 2007), (DELGADILLO e LLANO, 2007) e (SHNITS e SINREICH, 2006) usam regras de despacho ou heurísticas para atribuir prioridades aos processos conflitantes.

No trabalho (TUNCEL, 2007), uma abordagem baseada em regras heurísticas para programação dinâmica e controle de FMS, que integra o carregamento, a entrada de peças e a escolha de roteiros é proposta.

Para atingir uma utilização eficiente dos recursos e encontrar uma sequência melhor de operações, uma política de alocação de recursos em tempo real é desenvolvida. As decisões consideram as condições do chão de fábrica. A modelagem do sistema é realizada usando redes de Petri de alto nível e considera a sequência de execução das transições para análise. Um conjunto de regras de produção "se-então" é construído com base na heurística desenvolvida. Para construção das regras são consideradas informações como o número de peças conflitantes, plano de processamento do produto, e número de operações restantes. Assim a peça é selecionada de acordo com regras de prioridade.

Em (ZHANG, JIANG e GUO, 2007) é proposto um mecanismo para programação em tempo real baseado em simulação no qual regras de despacho e estratégias de reparo variam dinamicamente de acordo com informações de tempo real. O estudo de caso examina o impacto de desempenho do método de acordo com alguns critérios como vazão, entrega dos produtos e tempo de processamento. Um algoritmo que combina várias regras de despacho é desenvolvido com base na teoria de restrições. Um usuário monitora o estado corrente do sistema e as medidas de desempenho de um intervalo prévio de produção e fornece os valores desejados para a produção do próximo período. O simulador avalia as regras de despacho e estratégias de reparo e seleciona a melhor combinação.

Em (DELGADILLO e LLANO, 2007) foi proposta uma abordagem para modelagem e programação de sistemas de manufatura usando redes de Petri e regras de despacho para resolver eventuais conflitos. A execução do algoritmo proposto corresponde aos disparos das transições até que o último estado seja alcançado. Um conflito operacional ocorre quando várias transições estão habilitas, ou seja, existe uma competição pelo mesmo recurso. As medidas de desempenho consideradas foram minimizar o tempo total de processamento, maximizar a utilização de recursos e atingir o prazo de entrega.

Em (SHNITS e SINREICH, 2006) é apresentada uma metodologia para controle multicritério de FMS. A metodologia é baseada em um mecanismo de decisões de dois níveis para desenvolver uma programação adaptativa. O primeiro passo é selecionar um critério de decisão e um conjunto de regras de programação usando um algoritmo baseado em regras de acordo com o estado do chão de fábrica, requisitos da produção e prioridades do sistema. O segundo passo é selecionar por simulação a regra de programação que obteve o melhor desempenho de acordo com o critério selecionado. Para selecionar o critério de operação são considerados o estado do chão de fábrica, requisitos da produção e prioridades do sistema. A metodologia proposta expande a programação adaptativa habilitando mudanças, não somente nas regras de despacho, mas também no critério de objetivo que governa as operações do sistema e afeta a seleção de uma regra apropriada.

2.2.2 Controle

No trabalho (TSINARAKIS, TSOURVELOUDIS e VALAVANIS, 2005) uma rede de Petri temporal é usada para modelar a flexibilidade de operações e de roteiros em um sistema de produção. Quatro modelos são construídos envolvendo diferentes níveis de flexibilidade. O modelo consiste em duas partes, um que define o estado das máquinas em cada instante de tempo e o outro que descreve os processos e as peças transportadas no sistema.

Em (ODREY e MEJÍA, 2005), a abordagem desenvolvida é baseada na integração de submodelos de redes de Petri com o modelo geral do sistema de manufatura. Os submodelos são os planos de recuperação que são incorporados nas estações de trabalho e consistem em um sequencia de passos necessários para fazer com que o sistema retorne a um estado normal.

Em (NOUREDDINE e MARTINEAU, 2005) é apresentada uma abordagem para modelar e analisar um FMS usado redes de Petri. Primeiramente é definida a descrição física do modelo e então construído o modelo conceitual. Cada recurso é considerado como uma entidade operacional associada a uma entidade de armazenamento. Entidades adjacentes se comunicam através do modelo genérico. O modelo conceitual é obtido a partir da notação genérica e dos processos de manufatura.

Em (BUYURGAN e SAYGIN, 2006) foram apresentados o projeto e desenvolvimento de um framework integrado de controle que prove um modelo de controle supervisionado em tempo real com capacidade de fazer previsões de como o FMS se comportaria em um determinado tempo no futuro. Os objetivos do controle e suas políticas foram modelados e caracterizados por uma base de regras *Fuzzy* integrada ao modelo de controle. Este *framework* consiste em um gerador de máquinas de estados finitas e um controlador. O controlador monitora medidas de desempenho do FMS e reage de acordo com as mudanças de estado do sistema a fim de manter as medidas de desempenho no nível desejado. A validação do *framework* foi dada por meio do desenvolvimento de uma plataforma de software. O desempenho do *framework* foi verificado em um FMS hipotético usando simulação.

O trabalho (MORANDIN, ARAUJO e KATO, 2006) propõe uma estratégia de modelagem para o planejamento, controle e intertravamento de um AMS usando Redes de Petri Virtuais, considerando a comunicação com níveis superiores através de um supervisório. Redes de Petri Colorida foram usadas para representar produtos e sinais. Redes de Petri temporal para representar o tempo de montagem e o de deslocamento.

O trabalho (WU, ZHOU e HU, 2007) compara dois métodos de modelagem, o modelo em redes de Petri orientada a processos (POPN – *Process Oriented Petri Net*) e

o modelo em redes de Petri orientada a recursos (ROPN – *Resource Oriented Petri Nets*). Para a modelagem em POPN são modeladas as sequências do processo de produção, os recursos, a solicitação de recursos para as operações. No primeiro passo as atividades são identificadas para o processamento de cada peça, incluindo a execução das operações e o armazenamento. Para a modelagem em ROPN são modelados os recursos do sistema e, para cada tipo de produto, é modelado o processo de produção em uma sub-rede. As sub-redes são unidas e cores são introduzidas para descrever os processos.

Em (ZHANG, JIANG e GUO, 2007) é apresentado um modelo de FMS em redes de Petri orientada a objetos e descreve as relações das peças do sistema. O modelo é particionado em quatro classes de objetos, peças, máquinas, transporte e armazenamento. Cada objeto é modelado em uma sub-rede que troca informações com outros objetos.

Em (TÜYSÜZ e KAHRAMAN, 2010) é apresentada uma abordagem para modelagem e análise de tempo de execução de uma célula flexível de manufatura usando redes de Petri estocástica com conjuntos *Fuzzy* para melhorar o poder de modelagem e análise de sistemas complexos. Este tipo de modelagem representa incertezas e o parâmetro de distribuição exponencial é representado por um número triangular *Fuzzy*.

Em (TUNCEL, 2007) foi proposta uma abordagem orientada a objetos para a modelagem e análise do problema de programação de chão de fábrica de FMSs usando rede de Petri de alto nível. Nesta proposta primeiramente foi construído o diagrama de modelagem orientada a objetos de um FMS e desenvolvida uma heurística baseada em regras para o problema de contenção de recursos e em seguida, foi formulado o comportamento dinâmico do sistema por meio de uma rede de Petri de alto nível.

O trabalho (ARAÚJO, 2006) propõe um método de modelagem de controle usando redes de Petri Virtuais para AMSs. O intertravamento das máquinas e dos AGVs são modelados separadamente em módulos. Uma vez que os componentes estão modelados, todos módulos são conectados por meio de redes de Petri Virtuais. O sistema

de controle é inserido no próprio modelo por meio de funções nos arcos da rede de Petri do modelo.

Em (SGAVIOLI, 2010) é sugerida uma abordagem de para modelagem de FMSs considerando pontos de conflito e tomada de decisão. Um dos aspectos do trabalho é a utilização de informação do FMS em tempo real para apoiar as decisões. Redes de Petri Colorida *Fuzzy* foram usadas para modelar a base de regras *Fuzzy* e um sistema de inferência *Fuzzy*. Com base nas informações em tempo real do estado do sistema, prioridades foram dadas aos processos em conflito. Quando um produto tem duas ou mais rotas, uma é selecionada com base nas entradas das regras *Fuzzy* do modelo. Um modelo de FMS existente foi utilizado como base para a proposta. A proposta foi verificada via *software* de simulação.

2.3 Considerações finais

A programação da produção e o controle são entidades isoladas e independentes. Não obstante, estas entidades são mutuamente colaborativas. Pode-se dizer que a programação da produção é responsável por elaborar uma agenda de alocação de recursos no tempo. Dado que dois ou mais recursos podem ser alocados para uma tarefa no mesmo tempo, segue que essa programação é geradora de conflitos. A figura do sistema de controle irá agir nestes conflitos de modo que eles sejam solucionados em tempo real.

Observa-se então, que existe um limiar entre a programação da produção e o sistema de controle. Este limiar pode ser justificado pelo fato de as duas atividades, programação e controle, serem processadas em tempos diferentes e produzirem saídas diferentes. Diferente da programação, o controle não pode gerar uma saída ou reposta que caracterize conflito. Uma vez que o controle foi acionado, entende-se que o sistema já está em conflito e uma decisão deve ser tomada para dar continuidade ao processo de produção. Esta tomada de decisão deve ser dada em tempo real, enquanto que o

processamento da programação da produção pode acontecer em qualquer ponto no tempo antes da execução desta programação.

Muitas abordagens para a realização do controle de FMSs e programação da produção foram propostas na literatura. Grande parte destes trabalhos faz uso de técnicas de modelagem baseadas em redes de Petri, orientação a objetos, *Fuzzy*, heurísticas, sistemas multi-agentes e também sistemas holônicos. Além disso, observa-se o uso de modelos de FMSs hipotéticos para validação das propostas.

Neste trabalho foi usado rede de Petri Colorida no desenvolvimento do modelo do FMS considerado. O FMS modelado, também usado nos trabalhos (ARAÚJO, 2006) e (SGAVIOLI, 2010), é hipotético. Um software foi desenvolvido para realizar as funções do sistema de controle e um sistema *Fuzzy* foi desenvolvido para apoiar a tarefa de resolução dos conflitos do FMS.

Capítulo 3

PROPOSTA

3.1 Hipótese

A hipótese deste trabalho é baseada na formulação provisória de que é possível realizar o controle de um de FMS considerando informações do sistema como o número de produtos e os tipos de produtos nos *buffers* de entrada e de saída, o número de nós ou pontos de entroncamentos que os AGVs terão que acessar para alcançar estes, bem como a distância que estes AGVs devem percorrer para atender as necessidades de transporte de produtos no FMS.

O raciocínio dedutivo que precede a hipótese deste trabalho é que, uma vez que estas informações são intrínsecas aos FMSs, é natural que um sistema de controle que as leve em consideração terá uma representatividade mais próxima da realidade do que um sistema de controle que não as considere e, portanto, um desempenho melhor. Tal desempenho pode ser medido por meio da diminuição tanto do *makespan*, quanto da ociosidade de recursos do sistema como máquinas e AGVs.

Assim, a hipótese trabalhada e mais tarde verificada é a possibilidade de realizar o controle de um FMS considerando informações do FMS aumentando a representatividade do sistema e garantindo que este FMS seja livre de *deadlocks*.

3.2 Proposta

A proposta deste trabalho é construir um sistema de controle para um modelo de FMS. O modelo do FMS é baseado na estratégia de modelagem proposta em (ARAÚJO, 2006) que, por sua vez, tem o sistema de controle interno ao modelo.

A estratégia de modelagem proposta em (ARAÚJO, 2006) garante um modelo livre de *deadlocks* para um ou mais planos de produção, porém, não faz tratamento das variáveis do sistema FMS citadas na seção anterior deste trabalho.

O FMS modelado para a validação desta proposta tem a mesma configuração daquele usado na validação da proposta de (ARAÚJO, 2006).

3.2.1 Descrição do FMS

O FMS é composto por seis máquinas, três AGVs, um estacionamento para AGVS, um armazém para produtos que serão processados pelas máquinas e produtos terminados, e uma rota circular para o tráfego de AGVs. O FMS produz três tipos distintos de produtos. Na Figura 1 o arranjo físico do FMS é apresentado.

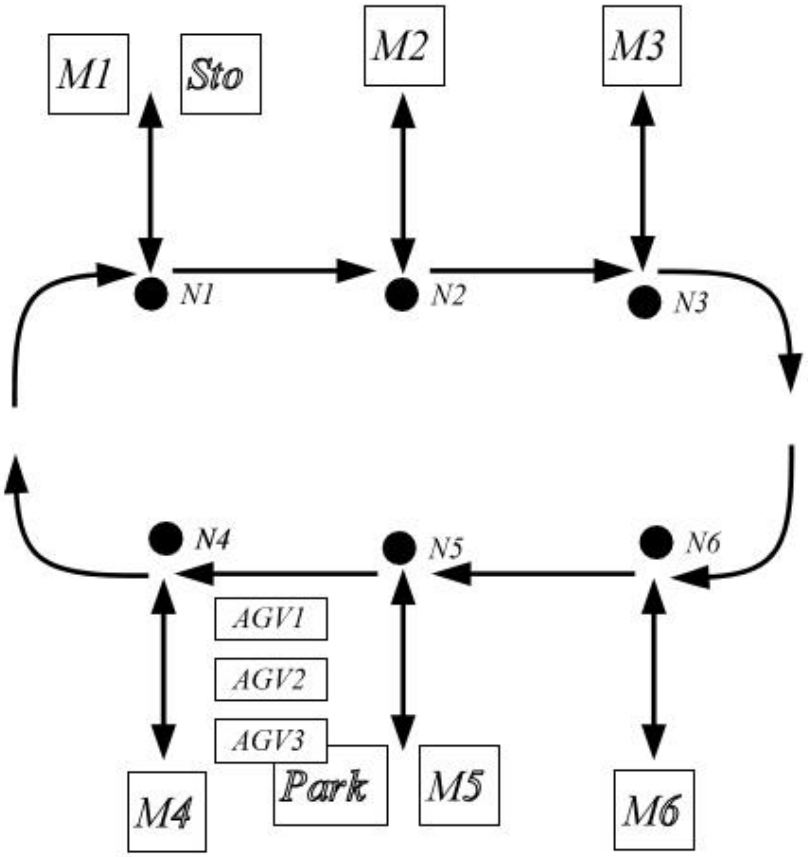


Figura 1 – Arranjo físico do FMS.

Na Figura 1 é mostrado a disposição das máquinas que estão indicadas por *M1, M2 M3*, *M4*, *M5*, e *M6*. São mostrados o estacionamento, indicado por *Park*, e o armazém indicado na figura por *Sto*. Os nós ou pontos de entroncamento são representados por *N1, N2, N3, N4, N5* e *N6*. O caminho dos AGVs está representado por setas que indicam o sentido de locomoção dos AGVs. Para acessar qualquer um destes recursos: máquinas, estacionamento e armazém, é necessário que o AGV se dirija até o nó localizado a frente do recurso obedecendo ao caminho dos AGVs. Na Figura 1 três AGVs estão posicionados no estacionamento.

Cada uma das seis máquinas possui dois *buffers*: um de entrada de produtos que serão processados pela máquina, e outro de saída de produtos que foram processados pela máquina e estão prontos para serem transportados, ou para uma próxima máquina,

que irá realizar o próximo estágio de produção daqueles produtos que não estiverem terminados, ou para o armazém caso estejam terminados.

Um *buffer* de entrada e um *buffer* de saída de uma máquina formam a área de *buffers* desta máquina. É permitido até três AGVs na área de *buffers* de uma determinada máquina.

O armazém, assim como as máquinas, possui dois *buffers*: um de saída que armazena os produtos serão transportados pelos AGVs para que as máquinas possam processá-los, e outro de entrada que armazena os produtos terminados ou prontos.

Os AGVs podem transportar apenas um produto por vez, esteja este produto em processo de produção ou terminado.

A rota circular dos AGVs possui seis nós ou pontos de entroncamento que permitem ao AGV acessar os *buffers* de entrada e saída das máquinas. Cada máquina está posicionada á frente de um dos pontos de entroncamento da rota dos AGVs.

Entre os pontos de entroncamento, dois destes, além de darem acesso a duas máquinas distintas, também dão acesso ao armazém e ao estacionamento de AGVs. Assim têm-se um ponto de entroncamento que além de dar acesso a uma máquina dá acesso ao estacionamento de AGVs, e outro que além de dar acesso a uma máquina dá acesso ao armazém. Na Figura 1 observa-se que o nó ou ponto de entroncamento *N1* dá acesso a máquina *M1* e ao armazém *Sto*. O nó *N5* dá acesso ao estacionamento *Park* e á maquina *M5*.

Cada *buffer* de máquina seja de entrada ou de saída possui capacidade para 100 produtos sejam eles de quaisquer tipos.

Os *buffers* de entrada e saída do armazém, assim como os *buffers* das máquinas, possuem capacidade para 100 produtos de quaisquer tipos.

O estacionamento de AGVs tem capacidade para três AGVs.

Os AGVs podem trafegar na rota circular do FMS apenas em sentido horário.

O caminho ou rota dos AGVs permite que apenas um AGV ocupe cada ponto de entroncamento. Para que um AGV acesse os de alguma máquina é necessário alcançar antes o ponto de entroncamento ao qual a máquina esteja posicionada à frente. Se houver pontos de entroncamento intermediários até o fim da rota do AGV para um *buffer* de saída ou *buffer* de entrada, o AGV deverá passar por todos estes pontos. Nos casos em que um AGV acessar um *buffer* de entrada ou de saída de alguma máquina o ponto de entroncamento desta máquina ficará livre. Assim, o caminho ficará livre para um AGV que estiver atrás possa seguir o seu caminho para qualquer máquina inclusive a máquina que está à frente deste ponto de entroncamento, haja vista que o esquema físico do FMS permite que até três AGVs ocupem a área de *buffers* de uma máquina.

3.2.2 Variáveis do FMS consideradas no sistema de controle

O sistema de controle proposto trabalha com variáveis do FMS como quantidade de produtos em *buffer* e distâncias das trajetórias dos AGVs, a fim de reduzir o makespan e a ociosidade dos recursos do FMS. A descrição e a forma de tratamento destas variáveis são definidas a seguir.

3.2.2.1 Espaço restante em *buffers* de entrada e saída

Para qualquer AGV do sistema são possíveis os seguintes estados com relação à carga de produto: livre, e carregado. No primeiro estado o AGV não está portando nenhum produto, e no segundo estado, inverso, o AGV está com algum produto em sua carga.

Quando o estado do AGV é livre, este AGV precisa tomar as decisões de qual *buffer* de saída irá atender, qual produto deste *buffer* de saída será transportado, e para qual *buffer* de entrada este produto será transportado e então depositado. Tomadas as decisões, o AGV se dirige para o *buffer* de saída escolhido, carrega o produto escolhido, mudando seu estado para carregado e, se dirige ao *buffer* de entrada escolhido onde deposita o produto que está sendo transportado. O AGV atinge então o estado livre novamente. O processo se repete até que todas as demandas de transporte de produtos sejam esgotadas.

Conforme o processo de produção acontece os AGVs do FMS precisam fazer decisões. Estas decisões configuram situações de conflito no sistema.

O número de produtos, e tipos de produtos em cada *buffer* de entrada e de saída são informações importantes para que essas decisões sejam tomadas e os conflitos resolvidos.

A proposta deste trabalho é que estas informações sejam usadas estrategicamente pelos AGVs para que estes possam tomar uma decisão eficiente em relação aos conflitos.

Um AGV que está livre e, portanto, em situação de conflito, deverá analisar todos os *buffers* de saída das máquinas do sistema. Nesta análise serão observados a quantidade de produtos e quais tipos de produto que cada *buffer* de saída está armazenando. Para cada tipo de produto serão observados os *buffers* de entrada das possíveis máquinas nos quais cada produto poderá ser descarregado levando em consideração o roteiro de produção de cada produto. Sabendo-se quais as possíveis máquinas para as quais cada produto pode ser transportado, pode-se avaliar a quantidade de produtos em cada um dos *buffers* de entrada das máquinas.

Com esta coleção de informações: número de produtos no *buffer* de saída, tipos de produtos nos *buffers* de saída e número de produtos no *buffer* de entrada das máquinas de destino, o AGV poderá escolher: um *buffer* de saída que esteja próximo da sua lotação impedindo que este *buffer* de saída atinja a sua lotação e sua máquina pare de trabalhar por não ter espaço para depositar os produtos processados; um produto que pode ser processado em uma máquina que esteja com a quantidade de produtos no *buffer* de entrada muito baixa e, portanto na iminência de parar de trabalhar por falta de produto para processar. Desta maneira, é possível diminuir o tempo de ociosidade das máquinas.

3.2.2.2 Número de nós

Com relação ao número de nós, que são pontos de entroncamento do caminho dos AGVs, quanto menor a quantidade de nós usados no transporte de um produto mais fluidez terá o sistema de transporte. Quanto maior o número de nós, maior a chance de haver um trecho da trajetória ocupado por outro AGV.

Isto sugere que o número de nós que um AGV irá percorrer do seu ponto atual ao ponto de carga, mais o número de nós entre o ponto de carga e o ponto de descarga do produto é uma informação importante e que deve ser considerada para melhorar o desempenho do sistema de transporte do FMS.

Neste trabalho o número de nós será considerado na resolução dos conflitos de escolha entre *buffers* de entrada e saída. Quanto menor for o número de nós de um possível atendimento a *buffer* de saída e de entrada, maior será a prioridade do AGV em fazer este atendimento.

3.2.2.3 Distância

Entre cada área de *buffers* (ponto de carga do AGV e ponto de descarga do AGV) há uma distância que deve ser considerada. Quanto maior a distância que um AGV leva para se descolar do seu ponto atual até o ponto de carga e em seguida até o ponto de descarga do produto, mais este AGV irá ocupar o caminho de transporte e, além disso, o tempo de execução do transporte será maior. Isto não é interessante para manter um *makespan* baixo. Portanto deslocamentos menores terão prioridade maior de atendimento.

3.2.3 Tratamento de informações do sistema pelo sistema de controle

Todas as variáveis, ou informações do FMS, citadas deverão ser ponderadas pelos AGVs a cada tomada de decisão em cada situação de conflito.

Os conflitos existirão para AGVs que estiverem em algum nó e no estado livre. Isso acontece porque sempre que um AGV entrar em estado de ocupado, este já terá ponderado todas as variáveis consideradas para decidir inclusive o *buffer* de entrada no qual será depositado o produto que possui em sua carga, não sendo mais necessário pensar até que este produto tenha sido depositado no devido *buffer* de entrada. Uma vez que o produto é depositado, o AGV seguirá para o ponto de entroncamento da máquina cujo *buffer* de entrada recebeu o produto. Assim o AGV entra no estado livre e na posição ponto de entroncamento. Este ciclo se repete enquanto houver produtos a serem transportados.

Os conflitos também vão existir quando um AGV estiver em um nó, independentemente de estarem livres ou carregados com algum produto. Nesse caso sempre será verificado se o AGV está em seu nó de destino. Em caso positivo a solução do conflito será enviar este AGV para a área de *buffers* da máquina que estiver à frente do nó. Caso contrário, a solução do conflito é enviar o AGV para o próximo nó até que o nó de destino seja alcançado.

Na Tabela 1 estão relacionadas as situações em que o sistema entra em conflito.

Tabela 1 – Possíveis estados de conflito do sistema.

Estado/Posição do AGV	Situação de conflito	Variáveis Consideradas
Livre/Ponto de entroncamento	Escolher *buffer* de saída Escolher produto Escolher *buffer* de entrada	Quantidade de produtos nos *buffers* de saída Produtos nos *buffers* de saída Número de nós Distância a percorrer Quantidade de produtos nos *buffers* de entrada
Livre/Dirigindo-se para o *buffer* de saída	Entrar na área de *buffers* Seguir para o próximo nó	Destino do AGV Nó atual
Ocupado/Ponto de entroncamento	Entrar na área de *buffers* Seguir para o próximo nó	Destino do AGV Nó atual

Ocupado/Dirigindo-se para o *buffer* de entrada	Entrar na área de *buffers* Seguir para o próximo nó	Destino do AGV Nó atual

Os valores das variáveis podem ter descrição linguística e podem ser ponderadas por um sistema *Fuzzy*. Neste trabalho é proposto que um sistema *Fuzzy* pondere todas estas variáveis a cada tomada de decisão, e entregue para o AGV que estiver em situação de conflito, a tarefa com maior prioridade de atendimento. O sistema *Fuzzy* fará parte do sistema de controle.

3.2.4 Comparativo entre a realização do controle considerando variáveis do FMS e sem considerar variáveis do FMS

O trabalho presente propõe uma melhoria, de acordo com a hipótese a ser verificada, para a estratégia de modelagem proposta por (ARAÚJO, 2006). Nesta sessão serão apresentados alguns aspectos que diferenciam as duas estratégias.

Em (ARAÚJO, 2006) não são consideradas variáveis do FMS como número de nós e distância das trajetórias dos AGVs, e quantidade de produtos em *buffers* de entrada e saída. A proposta do trabalho presente o FMS é o mesmo, porém considera variáveis do FMS na resolução de conflitos.

Com relação à modelagem, em ambos os trabalhos realizada em redes de Petri, o modelo trabalho de (ARAÚJO, 2006) possui uma quantidade menor de places em relação ao trabalho presente. No entanto a quantidade de funções ou pesos nos arcos em (ARAÚJO, 2006) é bastante elevada em relação aos arcos do modelo do trabalho presente que por sua vez não possui funções nos arcos. Esta quantidade alta de inscrições nos arcos é explicada pelo fato de o controle do modelo ser realizado no próprio modelo e, portanto todas as resoluções de conflitos são resolvidas pelos pesos e funções nos arcos. Isto confere uma menor legibilidade ao modelo. Realizar o controle considerando variáveis do FMS aumentaria significativamente a complexidade do modelo usado em (ARAÚJO, 2006). Por este motivo a característica de controle interno ao modelo proposta em (ARAÚJO, 2006) não foi adotada nesta proposta.

Com relação à escolha de *buffers* de entrada em (ARAÚJO, 2006) todas as escolhas são definidas nos arcos em forma de listas que representam o plano de produção. O sistema segue o plano de produção determinado nestas listas independentemente do comportamento do sistema conforme a execução do plano de produção. Isto significa que todos os conflitos estão previamente resolvidos e o sistema de controle, interno, não reage a qualquer comportamento indesejado do sistema. A intenção da modelagem realizada em (ARAÚJO, 2006) é garantir que o sistema fique livre de *deadlocks*, mesmo que o desempenho seja baixo. Já no trabalho presente todos os conflitos são resolvidos por um sistema de controle externo ao modelo que reage em tempo real, conforme a produção segue. As decisões para os conflitos são tomadas com a intenção de melhorar o desempenho do sistema no sentido de diminuir o *makespan* e diminuir a ociosidade dos recursos do sistema.

Em (ARAÚJO, 2006) a decisão tomada a respeito de qual *buffer* de entrada será escolhido para depositar o produto que o AGV transporta é definida em uma lista estática inscrita no modelo que, de acordo com o tipo do produto que será transportado a partir de uma determinada máquina ou *buffer* de saída, retorna o *buffer* de entrada ao qual o AGV deverá se dirigir e depositar o produto.

Tanto a lista de *buffers* de saída de produtos quanto a lista de *buffers* de entrada de produtos fazem parte do modelo, e formam os roteiros de cada produto. Com isso o sistema terá sempre uma execução estática que não reflete as eventuais necessidades do sistema. Por exemplo, caso queira-se simular a indisponibilidade de alguma máquina por quebra durante a execução de um plano de produção, todos os produtos que tiverem roteiros de produção que, previamente, dependem desta máquina não seguirão seu processo de produção a menos que o modelo seja alterado.

Em (ARAÚJO, 2006) a escolha dos produtos que serão transportados pelos AGVs segue uma regra que não considera nenhuma das variáveis do FMS citadas nesta proposta.

Na proposta do trabalho presente, os *buffers*, tanto de saída quanto de entrada, bem como os produtos que serão transportados pelos AGVs são eleitos automaticamente

pelo sistema de controle, que é externo ao modelo em rede de Petri, considerando a situação atual do sistema por meio de suas variáveis. Portanto, toda e qualquer mudança de roteiro pode ser feita pelo sistema de controle sem a necessidade de alteração do modelo.

3.3 Considerações finais

Foi definido o arranjo físico do FMS considerado neste trabalho que é composto por máquinas com áreas de carga e descarga de produtos, AGVs, estacionamento de AGVs e um armazém que comporta produtos que serão processados e produtos que foram terminados.

Foram levantadas as questões que o trabalho de (ARAÚJO, 2006) não contempla a fim de esclarecer a contribuição deste trabalho.

As informações, ou variáveis, do sistema FMS que serão observadas para a realização do controle foram caracterizadas.

Por fim em uma comparação entre sistemas de controle que usam informações do sistema FMS para a realização do controle e sistemas que não consideram essas informações foi possível observar as vantagens que o uso destas pode trazer para a realização do controle,

As definições feitas nesta proposta formam a base para o desenvolvimento do modelo usado, e também para o sistema de controle.

Capítulo 4

IMPLEMENTAÇÃO DA PROPOSTA

4.1 Introdução

Para a validação da proposta foram construídos três sistemas de *Software* para viabilizar a simulação do modelo: o Sistema de Interface de Comunicação com a Ferramenta *CPN Tools*, o Sistema Decisor *Fuzzy*, e o Sistema Controlador. Estes três sistemas são subsistemas de um sistema maior chamado Sistema de Controle. O Sistema de Controle é o responsável por resolver os conflitos do sistema FMS modelado.

Neste capítulo cada um destes sistemas são descritos detalhadamente tanto em termos de construção, as suas estruturas em nível de linguagem de programação, quanto em termos de funcionalidades, ou as suas responsabilidades dentro do conjunto de sistemas.

Ao longo do capítulo também são apresentadas o conjunto de ferramentas de *Software* usado na construção destes sistemas. As responsabilidades de cada sistema desenvolvido são mostradas por meio de diagramas de caso de uso.

Por fim é apresentado o procedimento passo a passo que deve ser executado para que todos os sistemas entrem em funcionamento para a realização do controle do modelo proposto neste trabalho.

4.2 Ferramentas

Para o desenvolvimento dos sistemas de *Software* usados para a validação do modelo proposto neste trabalho foram usadas as seguintes ferramentas:

- *CPN Tools*
  - *Software* de modelagem usado na construção do modelo em rede de Petri. Neste trabalho foi usada a sua versão *CPN Tools* 2.0.
- *Access/CPN*
  - Biblioteca de *Software* usada pelo Sistema de Controle para acessar funções da ferramenta *CPN Tools*. Neste trabalho foi usada a sua versão *Access/CPN* 2.1.0.
- *Matlab*
  - IDE (*Integrated Development Enviroment*) usada para desenvolver as funções matemáticas do Sistema *Fuzzy*. Neste trabalho foi usada a sua versão 2010A.
- *Visual Studio*
  - IDE usada para escrever código de alto nível na linguagem de programação C#. Neste trabalho foi usada a sua versão 2010.
- *Eclipse*
  - IDE usada para escrever código de alto nível na linguagem de programação *Java*.

4.3 Sistemas de *Software*

Aqui são descritas a estrutura de código e as funcionalidade de cada um dos três sistemas de *Software* que foi desenvolvido para formar o Sistema de Controle usado na validação do modelo proposto neste trabalho.

Uma vez que os sistemas foram desenvolvidos considerando o paradigma de Programação Orientada a Objetos nesta seção todas as classes, bem como membro destas classes, são apresentados e detalhados em termos de estrutura e de funcionalidade.

4.3.1 Sistema de Interface de Comunicação com *CPN Tools*

A ferramenta *CPN Tools* é responsável por auxiliar na construção e simulação do modelo em rede de Petri. Para que o Sistema de Controle possa interagir com a simulação do modelo, e assim realizar o controle do modelo, é necessário que haja um canal de comunicação entre o Sistema de Controle e o processo *CPN Tools* responsável pela execução da simulação do modelo. Esta tarefa pode ser simplificada por meio da biblioteca *Access/CPN*.

A biblioteca *Access/CPN* permite o controle externo para realizar as seguintes atividades:

- Disparar transições habilitadas no modelo em simulação;
- Ler as marcas dos *places* que compõe a rede de Petri que representa o modelo;
- Alterar as marcas dos *places* que compõe a rede de Petri que representa o modelo.

Foi criado um projeto baseado em *Java* usando a IDE *Eclipse* para o desenvolvimento das classes que estendem o *Access/CPN* e realizam o a comunicação com o modelo apresentado neste trabalho. Este projeto foi denominado *CPNControl*.

4.3.1.1 Projeto CPNControl

O projeto *CPNControl* é responsável por estabelecer comunicação entre o Sistema de Controle externo e o modelo simulado na ferramenta *CPN Tools*. No diagrama de caso de uso apresentado na Figura 2 observa-se que toda comunicação entre o Sistema Controlador e o *CPN Tools* envolve o caso de uso Estabelecer comunicação com o modelo que é provido pelo Sistema de Interface de Comunicação com o *CPN Tools*.

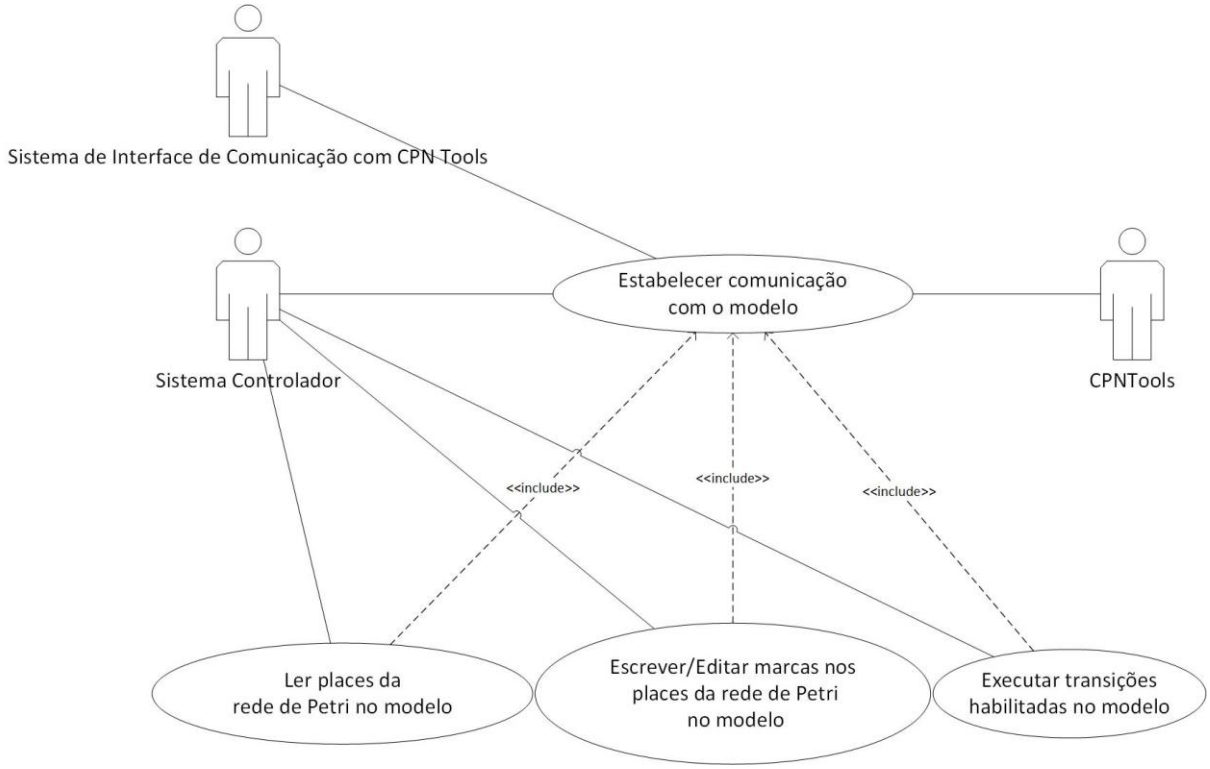


Figura 2 – Diagrama de caso de uso do sistema de Interface de Comunicação entre CPN Tools e Sistema Controlador.

A seguir são listadas as tarefas realizadas para criação da Interface de Controle com o *CPN Tools*:

- Criar projeto no *Eclipse* que estenda a biblioteca AccessCPN para abrir interface para os métodos:
  - *HighLevelSimulator.getMarking(PlaceNode);*
  - *HighLevelSimulator.setMarking(PlaceNode, String);*
  - *HighLevelSimulator.execute()*.
- No projeto criado no *Eclipse* deve ser criado um *WebService SOAP* baseado na classe que contenha os métodos estendidos da biblioteca *AccessCPN*.
  - Depois de criado o *WebService* deverá ser configurado com um parâmetro que mantenha o estado dos objetos da classe no escopo da aplicação para que não seja criado um objeto novo a cada chamada feita para o *WebService*.

O Projeto *CPNControl* é composto por quatro classes que estendem a biblioteca *Access/CPN*:

- *Simulator;*

- *PNWatcher;* e
- *PlaceWatcher*.

4.3.1.1.1 Classe Simulator

A classe *Simulator* tem a responsabilidade de referenciar a instância local do simulador do *CPNTools*. Uma vez que a classe *Simulator* faz referência ao simulador é possível que outras classes se comuniquem com o simulador, possibilitando o envio e o recebimento de sinais do e para o simulador.

Esta classe é formada pelos membros:

- *Simulator* – Este objeto é responsável por fornecer acesso a instância do simulador local do *CPNTools*.
- *proxyDaemon* – Este objeto é responsável pela localização do serviço de simulador do *CPNTools*. Por meio deste objeto o objeto *Simulator* é inicializado.
- *proxySimulator* – Este objeto realiza a comunicação entre o simulador e o *proxyDaemon*. O objeto *proxySimulator* recebe do objeto *proxyDaemon* uma instância do simulador local do *CPNTools* e então localiza a rede que está sendo simulada pelo *CPNTools*. Em seguida, já com a rede de Petri localizada e referenciada, o *proxySimulator* entrega o simulador do *CPNTools* e a rede de Petri para o objeto *Simulator* desta classe. O objeto *Simulator* então está pronto para realizar a comunicação com o simulador *CPNTools* local que está executando a rede de Petri.
- *Start()* – Este método é responsável por inicializar todos os objetos da classe até que o simulador local do *CPN Tools* seja localizado.

4.3.1.1.2 Classe PNWatcher

Esta classe tem a responsabilidade de fazer o reconhecimento da rede de Petri que está sendo simulada pelo *CPNTools* e localizar todos os *places* que compõe a rede de Petri.

Esta classe é formada pelos seguintes membros:

- *petriNet* – Objeto responsável por armazenar uma referência da rede de Petri que esteja sendo simulada no *CPNTools*.
- *Simulator* – Este objeto mantém uma referência ao simulador.

- *Iterator<PlaceNode>* - Este objeto é uma lista que é usada para iterar por todos os elementos da rede de Petri e encontrar os *places* que formam esta rede.
- *getPetriNetName()* – Este método recupera o nome da rede de Petri que está sendo simulada.
- *getPlaceNodes()* – Este método itera por todos os elementos da rede de Petri e inicializa a lista de *places* do objeto *Iterator<PlaceNode>*.

4.3.1.1.3 Classe PlaceWatcher

A classe *PlaceWatcher* é responsável por realizar as operações efetivas de alteração de marca de *place*, leitura de marca de *place*, e disparo das transições habilitadas na rede de Petri que está sendo simulada. Essas operações são realizadas com auxílio das classes *Simulator* e *PNWatcher*.

Esta classe é formada pelos seguintes membros:

- *pnWatcher* – Objeto responsável por manter uma referência da classe *PNWatcher*.
- *Simulator* – Objeto responsável por manter uma referência da classe *Simulator*.
- *getMark(String placeName)* – Este método retorna uma *string* contendo a marca de um dado *place* cujo nome é avaliado pelo parâmetro *placeName*.
- *setMark(String placeName, String Mark)* – Este método realiza a substituição da marca atual do *place* recebido como parâmetro por *placeName* pela marca recebido como parâmetro por *Mark*.
- *ExecuteSimulator()* – Este método envia um sinal para o simulador que executa todas as transições habilitadas na rede de Petri em um determinado momento.

4.3.1.2 *WebService* de interface *Access/CPN* para o Sistema de Controle

Com o objetivo de manter todo o sistema com maior independência de plataforma de *Software*, foi construído um *WebService* baseado nas classes que estenderam a

biblioteca *Access/CPN* para que estas classes de extensão pudessem ser acessadas por sistemas de diferentes plataformas por meio de protocolos de internet.

Para a construção do *WebService* foi usada a ferramenta *Eclipse*.

4.3.1.2.1 Criação do WebService

Para a criação do *WebService* a classe *PlaceWatcher* do projeto *CPNControl* foi usada como base haja vista que esta classe possui todas as operações necessárias para realizar o controle externo do modelo em rede de Petri simulado na ferramenta *CPN Tools*.

Os seguintes passos foram realizados para a criação do *WebService*:

- Com o projeto *CPNControl* aberto na ferramenta *Eclipse*, clique com o botão direito na classe *PlaceWatcher*. Um menu de contexto será apresentado.
- Em seguida, no menu de contexto apresentado, selecione o menu *WebServices* e então clique em Criar *WebService*.
- Complete o diálogo de criação de *WebService* da ferramenta e uma URI local será gerada para que outros sistemas possam acessar as operações expostas pelo *WebService*.

4.3.1.2.2 Adicionando referência ao Webservice no projeto ControlInterface

Para adicionar uma referência do *WebService* no projeto *ControlInterface* os seguintes passos foram executados:

- Com o a solução do *Visual Studio* que contém o projeto *ControlInterface* aberta, clique com o botão direito sobre o projeto *ControlInterface*.
- No menu de contexto que surge clique em adicionar referência de serviço.
- No dialogo informe a URI gerada durante a criação do *Webservice*.
- Todas as operações disponíveis no *Webservice* serão listadas. Selecione as operações e finalize o diálogo.
- Neste momento as operações *setMark*, *getMark*, e *ExecuteSimulator* já podem ser chamadas pelo sistema de controle.

4.3.2 Sistema Decisor *Fuzzy*

Durante a simulação do modelo surgem as situações de conflito. O papel do Sistema de Controle como um todo é solucionar estes conflitos. O subsistema Decisor *Fuzzy* tem papel fundamental neste processo pois ele irá receber o estado do sistema e devolver a resposta que contém a solução de transporte escolhida com base nas regras. Portanto antes de tirar o sistema da situação de conflito o Sistema Controlador irá consultar o caso de uso Verificar prioridade para cada atendimento de transporte possível. Este caso de uso é responsabilidade do Sistema Decisor *Fuzzy* e esta relação pode ser descrita pelo diagrama de caso de uso apresentado na Figura 3.

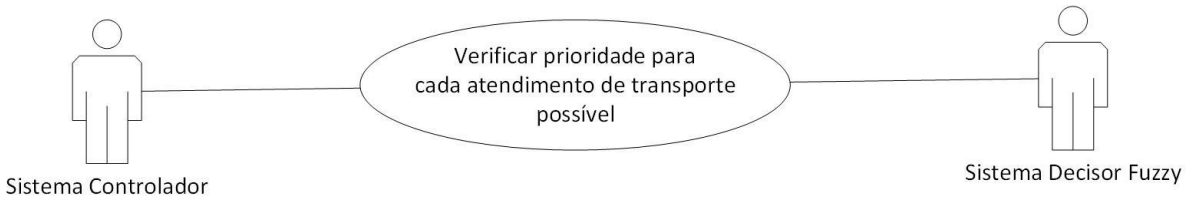


Figura 3 – Diagrama de caso de uso do sistema Decisor Fuzzy.

A biblioteca *Fuzzy* deve ser criada para atender as necessidades de resolução de conflito do modelo que será executado. Na tabela 2 são observadas as situações de conflito do modelo considerado neste trabalho.

Tabela 2 – Relação de conflitos.

Estado/Posição do *AGV*	Situação de conflito	Variáveis Consideradas
Livre/Ponto de entroncamento	Escolher *buffer* de saída Escolher produto Escolher *buffer* de entrada	Quantidade de produtos nos *buffers* de saída Produtos nos *buffers* de saída

		Número de nós Distância a percorrer Quantidade de produtos nos *buffers* de entrada
Livre/Dirigindo-se para o *buffer* de saída	Entrar na área de *buffers* Seguir para o próximo nó	Destino do *AGV* Nó atual
Ocupado/Ponto de entroncamento	Entrar na área de *buffers* Seguir para o próximo nó	Destino do *AGV* Nó atual
Ocupado/Dirigindo-se para o *buffer* de entrada	Entrar na área de *buffers* Seguir para o próximo nó	Destino do *AGV* Nó atual

Para criação do Sistema Decisor *Fuzzy* as seguintes tarefas devem ser executadas:

- No *Matlab* criar um projeto *Fuzzy* e neste projeto implementar a lógica *Fuzzy* que será usada para as decisões de quais *buffer*s de entrada e saída os *AGVs* do sistema irão atender.
    - No projeto devem ser criadas duas funções que farão a chamada para a lógica *Fuzzy* construída, e a passagem de parâmetros
        - *PrioridadeBufferIn*
            - Os parâmetros passados para esta função são:
                - Número de peças no *buffer* de entrada da área de *Load*;
                - Número de nós até a área de *Load*;
                - Distância até a área de *Load*.
        - *PrioridadeBufferOut*
            - Os parâmetros passados para esta função são:
                - Número de peças no *buffer* de saída da área de *Load*;
                - Número de nós até a área de *Load* do *buffer* de saída;
                - Distância até a área de *Load*;
                - Número de peças no *buffer* de entrada do *Load*.
    - No projeto criar um projeto do tipo *Deploy*. Um nome de classe deverá ser escolhido e em seguida os arquivos das funções *PrioridadeBufferIn* e *PrioridadeBufferOut* deverão ser adicionados a esta classe. Então o projeto de *Deploy* está pronto para gerar a biblioteca que será usada pelo sistema decisor.Para tal execute a opção *Build*.

- Este *deploy* irá gerar a biblioteca ou DLL que será usada para acessar a lógica *Fuzzy* pelo sistema de controle mais tarde. Esta biblioteca deverá ser reservada para referência futura.

4.3.3 Sistema Controlador

O Sistema Controlador é responsável pelas mudanças que ocorrem no modelo durante a simulação. Na Figura 4 é apresentada a relação que o Sistema Controlador estabelece com os outros subsistemas que compõem o Sistema de Controle.

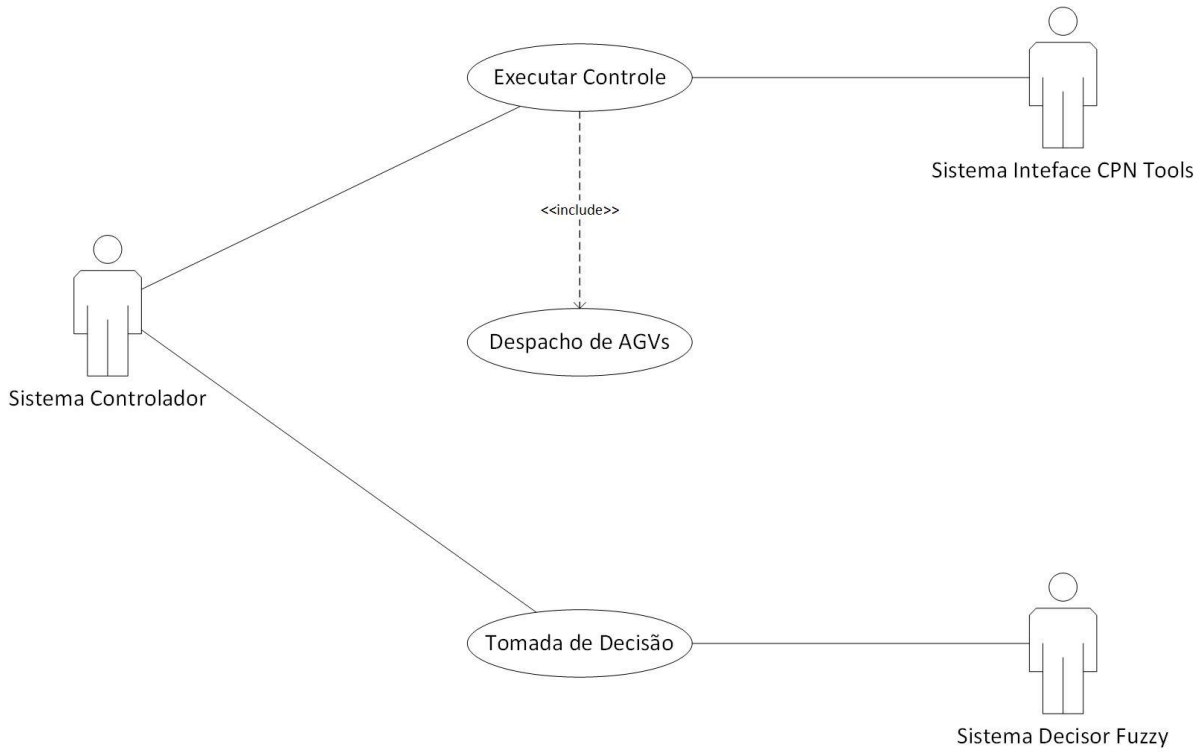


Figura 4 – Diagrama de caso de uso do sistema Controlador.

Tarefas para criação do Sistema Controlador

- No *Visual Studio* criar um projeto do tipo *Console Application*
- Neste momento a solução do *Visual Studio* terá apenas o projeto principal recém criado. Em seguida alguns projetos deverão ser criados na mesma solução. Os projeto que deverão ser criados são:
  - Projeto *Tear* do tipo *Console Application*
  - Projeto *Decision* do tipo *Class Library*
  - Projeto *ControlInterface* do tipo *Class Library*

- Projeto *AGVLoadControl* do tipo *Class Library*
- Projeto *AGVDispatch* do tipo *Class Library*

A seguir o detalhamento dos projetos criados na solução *Visual Studio* para a construção do Sistema de Controle

4.3.3.1 Projeto ControlInterface

O Projeto *ControlInterface* é responsável pelas comunicações entre o sistema de controle e o modelo simulado em *CPN Tools*. Este projeto faz uso do *WebService* criado para realizar esta comunicação. Por meio do *WebService* o projeto *ControlInterface* consegue resgatar as marcas nos *places* da rede de Petri, alterar as marcas da rede, e executar todas as transições habilitadas na rede.

Para que isso seja possível uma tarefa deve ser realizada:

- Deve se adicionar no projeto *ControlInterface* uma referência de serviço que aponte para o serviço criado pelo *WebService* do projeto do *Eclipse*.

Neste momento o projeto já consegue acessar o *WebService* que permite que as operações de rede de Petri no modelo sejam realizadas.

O *WebService* entrega e recebe as chamadas com marcas e de alterações de marcas em formato de texto. Isso é um fator limitante que reduz a produtividade. Por este motivo deve ser criado módulos ou *Namespaces* no projeto *ControlInterface* que tenham a responsabilidade de executar estas tarefas de tradução das informações, que chegam em formato texto vindas do *WebService*, em objetos que representam cada elemento do modelo representado pelos *places* da rede.

Namespaces são diretórios criados na estrutura do projeto que servem para organizar e dividir as responsabilidades das classes dentro do projeto. Devem ser criados dois *Namespaces* dentro do projeto: *Model* e *Control*. Para criar estes *Namespaces* deve-se realizar a seguinte tarefa:

- Criar dois novos diretórios no projeto *ControlInterface* com os nomes:

- *Model*
- *Control*.

No *Namespace Model* devem ser definidos todos os elementos existentes na rede de Petri com exceção das transições pois no escopo deste trabalho as informações contidas nas transições não tem influência na tomada de decisão e execução do controle. Estes objetos irão facilitar o trabalho ou o manuseio das informações vindas e enviadas para o modelo em rede de Petri.

Isto deve ser feito por meio da criação de classes. Para cada elemento na rede em *CPN Tools*: marca, *place*, e *MultiSet*; deverá ser criada uma classe possa criar objetos que representem estes elementos.

As classes devem ser escritas de modo a cumprir os seguintes objetivos:

- Classe *MultiSet* (*MultiSet*)
  - A classe *MultiSet* deve conter dois membros sendo um do tipo inteiro e outro do tipo *string*. O tipo *string* será responsável por guardar a Cor (redes de Petri Colorida) da marca. O tipo inteiro será responsável por guardar a quantidade de marcas de uma Cor representada pelo campo do tipo *string*.
- Classe *Mark* (marca)
  - A classe marca deve conter um campo do tipo lista de *MultiSet*. Essa lista ir registrar todas as marcas que o *place* possui bem como as cores e quantidade das cores dentro dessas marcas.
- Classe *Place* (*place*)
  - A classe *Place* deve conter um objeto que permita que a rede seja consultada e alterada a fim de possibilitar a recuperação e alteração de marcas nos *places* da rede. Este objeto deverá ser de um tipo que faça referência para o serviço referenciado dentro de *ControlInterface* já que a biblioteca gerada pelo projeto *ControlInterface* tem a responsabilidade de realizar as comunicações com a rede de Petri no *CPN Tools* por meio do *WebService* criado.
  - A classe *Place* deve ter também um membro do tipo *string* para representar o nome de um dado *place* na rede de Petri no *CPN Tools*.
  - A classe *Place* ainda deve ter um membro do tipo *Mark* para representar a marca que um dado *place* pode conter.
- Classe *MultiSetInfo*
  - A classe *MultiSetInfo* é uma classe estática criada para auxiliar na recuperação de marcas de um determinado *place*. Ela é responsável por recuperar as cores e a quantidade destas cores dentro das marcas nos

places. Esta classe faz uma leitura do *MultiSet* do *place* que está na rede de Petri.

Estas quatro classes completam a responsabilidade do módulo ou *Namespace Model* dentro do projeto *ControlInterface*.

O *Namespace Control* tem a responsabilidade de consumir *WebService* criado para acesso a rede em *CPNTools*. Ele é formado pelas classes:

- Classe *PlaceWatcher*
  - Esta classe é formada pelos membros:
    - A classe *PlaceWatcher* contém apenas um membro do tipo *Place* para representar o *place* que está sendo referenciado.
- Classe *Simulator*
  - Esta classe é formada pelos membros:
    - A classe *Simulator* possui um único membro que é uma instância do cliente de acesso ao *WebService* que foi criado dentro do projeto *ControlInterface*.

Com estes dois *Namespaces* construídos o projeto *ControlInterface* está pronto para gerar a biblioteca fornecendo os *Namespaces* que serão usados pelos outros projetos dentro da solução.

4.3.3.2 Projeto *TearFMS*

O projeto *TearFMS* é formado pelas classes que representam os elementos do modelo Tear de sistema de produção. Elementos como *AGV*, *Buffer* e máquina são representados pelas classes que devem ser construídas no projeto *TearFMS*.

Assim como no projeto *ControlInterface*, o projeto *TearFMS* é formado por *Namespaces*. São dois os *Namespaces* que formam este projeto: *Model* e *Control*.

Namespace Model

Este *Namespace* possui as definições de todos os elementos presentes na fábrica modelada. As classes que formam este *Namespace* deve conter as seguintes definições:

- Classe *AGV*

    - Esta classe é formada pelos membros
        - *Name*:*String* – representa o nome do *AGV* no sistema de produção. Serve como identificador único do *AGV*
        - *Status*:*AGVStatusTypes* – representa os possíveis estados que um *AGV* pode assumir, a saber, *Free* e *Busy*.
        - *Path*:*string* – representa o destino ou máquina de destino o qual o *AGV* deve seguir em uma determinada solicitação de transporte de peça ou produto.
        - *Product*:*Product* – representa o produto que o *AGV* está transportando
- Classe *AGVInfo*
    - Esta classe é formada pelos membros:
        - *AGVMark*:*Mark* – um método estático que retorna um valor do tipo *Mark* e recebe como parâmetro um valor do tipo *AGV*. Este método retorna a marca que representa o *AGV* na rede de Petri.
- Classe *AGVStatusTypes*
    - Esta classe é formada pelo membro:
        - *AGV*StatusType:*ENUM* – uma enumeração que representa os estados possíveis para o *AGV*:
            - *Free*
            - *Busy*
- Classe *AGV*Types
    - Esta classe é formada pelo membro:
        - *AGV*Types:*ENUM* – uma enumeração que representa os tipos de produto que podem ser transportados pelo *AGV*:
            - A
            - B
            - C
- Classe *Buffer* (*BufferLoad*)
    - Esta classe é formada pelos membros:
        - *Capacity*:Inteiro – representa a capacidade do *buffer*.
        - *Count*:Inteiro – representa a quantidade de produtos em um determinado momento no *buffer*.
        - *Place*:*Place* – representa o *place* na rede de Petri que está relacionado com este *buffer*.
- Classe *ControlPlace*
    - Esta classe é formada pelos membros:
        - *Place*:*Place* – representa o *place* na rede de Petri vinculado a este *place* de controle.
        - *Activate*:*Void* – um método que coloca uma marca de controle em um *place* de controle para que uma transação desejada seja disparada quando esta está em situação de *deadlock*. Este método

é chamado sempre que uma situação de conflito foi solucionada para dar continuidade com a execução da simulação do modelo.

- Classe *Load*
  - Esta classe é formada pelos membros:
    - *Name*:*String* – retorna o nome do *place* que representa na rede de Petri a área de *buffer*s de uma determinada máquina.
    - *Priority*:*Double* – representa a prioridade de atendimento que este *Load* tem em um determinado momento.
    - *Node*:*Node* – representa o nó ou entroncamento na linha que dá acesso a esse *Load* no modelo.
    - *AGVBuilder*:*AGVBuilder* – este objeto é usado para verificar na rede de Petri quais ou qual *AGV* estão em um determinado *place*. Quando um *AGV* chega em um *place* esse objeto é usado para descobrir qual *AGV* está naquele *place*.
    - *AGVs*:Lista<*AGV*> – guarda os *AGVs* presentes no *Load* em um determinado momento.
    - *BufferIn*:*BufferLoad* – representa o *buffer* de entrada do *Load*.
    - *BufferOut*:*BufferLoad* – representa o *buffer* de saída do *Load*.
    - *Place*:*Place* – representa o *place* que representa um determinado *Load* na rede de Petri.
- Classe *Machine*
  - Esta classe é formada pelos membros:
    - *Name*:*String* – representa o nome da máquina.
    - *AGV*:*AGV* – representa o *AGV* que está atendendo a máquina em um determinado momento.
    - *Load*:*Load* – representa o *Load* ou a área de carga e descarga da máquina.
- Classe *Node*
  - Esta classe é formada pelo membro:
    - *Point*:Inteiro – representa o número do ponto do entroncamento representado pelo nó.
- Classe *Product*
  - Esta classe é formada pelos membros:
    - *Type:ProductTypes* – representa o tipo deste produto.
    - *ID*:Inteiro – representa o identificador do produto.
    - *Priority*:*Double* – representa a prioridade que este produto tem dentro da linha de produção. Esta informação não é obrigatória.
    - *DueDate*:Inteiro – representa um fator que indica quanto tempo este produto tem para estar pronto para ser entregue e sair da fábrica.
    - *StringToProductTypes*(*String*):*ProductTypes* – um método que recebe um tipo de produto em *format string* e converte este tipo para a enumeração *ProductTypes*.

- Classe *ProductTypes*
  - Esta classe é formada pelo membro:
    - *ProductTypes*:*ENUM* – uma enumeração que representa os tipos de produto disponíveis.

Namespace Control

As classes responsáveis por alterar produto e status dos *AGVs*, descobrir quais *buffer*s precisam de atendimento de *AGV*, e por calcular as distâncias entre os nós entre uma máquina e outra dentro do sistema, estão definidas no *Namespace Control*. Suas classes têm as seguintes definições:

- Classe *AGVBuilder*
  - Esta classe é formada pelos membros:
    - *Place*:*Place* – representa o *place* na rede de Petri que será investigado para a descoberta de possíveis *AGVs* neste *place*.
    - *AGVs*:Lista<*AGV*> – representa a lista de todos *AGVs* que estão representados em uma marca de *place*.
    - *ExtractAGVsFromMark*(Lista<*MultiSet*>):Lista<*AGV*> – um método que recebe *MultiSet*s e extrai os *AGVs* dentro deste *MultiSet* (marca) retornando estes *AGVs* em uma lista de *AGVs*.
    - *RemoveAGVFromPlace*(*AGV*):*void* – um método que exclui a marca de um *AGV* de um determinado *place*.
    - *ChangeAGVPath*(*AGV*, *String*):*void* – um método que altera o destino de um *AGV*. O novo destino é passado como segundo parâmetro do tipo *string*.
- Classe *BufferIn*fo
  - Esta classe é formada pelos membros:
    - *IsBufferOutToAttend*:Lógico – verifica se há no sistema *buffer* de saída com produtos necessitando ser transportados e retorna verdadeiro ou falso.
    - *BufferOutWithProduct*(Lista<*Load*>):Lista<*Load*> – verifica todos os *Load*s cujo *buffer*s de saída contém produtos e retorna uma lista destes *Load*s.
- Classe *ControlPoint*
  - Esta classe é formada pelos membros:
    - *Place*:*Place* – representa o *place* na rede de Petri que representa um ponto de controle que também é representado por um *Node* ou nó.
    - *WhatToDoNext*:*Delegate* – este membro recebe um método como parâmetro. Este método é executado initerruptamente enquanto o sistema de controle estiver sendo executado. Sempre que um *AGV*

atingir um ponto de controle este método irá executar uma ação no *AGV*, por exemplo, alterar o destino do *AGV*.
        - *StartControl*():*void* – um método que executa o *Delegate WhatToDoNext* e em seguida executa o simulador que dispara todas as transições habilitadas.
        - *ExecuteSimulator*():*void* – um método que executa o simulador para disparar todas as transições habilitadas naquele momento.
- Classe *ControlPointWatcher*
    - Esta classe é formada pelos membros
        - *Place*:*Place* – representa o *place* na rede de Petri referente ao ponto de controle que está sendo assistido.
        - *AGVs*:Lista<*AGV*> – representa todos os *AGVs* que estão no *place* em um determinado momento.
        - *Load*:*ControlPlace* – representa a área de *buffer*s da máquina que está a frente deste ponto de controle.
        - *Storage*:*ControlPlace* – quando existe representa o armazém que pode ser acessado por meio do ponto de controle.
        - *Park*:*ControlPlace* – quando existe representa o estacionamento de *AGVs* que pode ser acessado por meio do ponto de controle.
        - *NextControlPoint*: *ControlPlace* – é uma referência para o próximo ponto de controle.
        - *WhatToDoNex*:*Delegate* – ponteiro para o método que será executado no loop de controle deste ponto de controle.
        - *Thread*:*Thread* – linha de processamento que será usada para executar o *loop* de controle.
        - *Simulator*:*Simulator* – referência de acesso ao simulador por meio do *WebService* localizado no projeto *ControlInterface*.
        - *Node*:*Node* – representa o nó ou entroncamento ao qual este ponto de controle pertence.
        - *StartWatching*():*void* – método que executa o loop de controle.
- Classe *Node*s
    - Esta classe é formada pelo membro:
        - *DistanceToNode*(Inteiro, Inteiro):Inteiro – retorna a distância entre dois nós ou entroncamentos.

De uma forma geral, o projeto *TearFMS* é útil para representar programaticamente cada elemento da fábrica que está sendo observado pelo sistema de controle. Com esta representação é possível ler e modificar o estado destes elementos de uma forma rápida. Uma outra vantagem é o aumento da legibilidade do código do sistema de controle. Isto contribui também para uma manutenção de código com poucos efeitos colaterais no restante do programa.

4.3.3.3 Projeto *Decision*

O projeto *Decision* contém o código responsável por receber dados das situações de conflito do sistema e enviar estes dados ao sistema de inferência *Fuzzy* para que sejam processados. O sistema de inferência *Fuzzy* devolve para o projeto *Decision* as prioridades das possíveis decisões a se tomar e o Projeto *Decision* então repassa esta informação para o sistema de Controle que irá atuar resolvendo o conflito de acordo com a decisão de maior prioridade do sistema de inferência. Os dados do sistema lidos pelo projeto *Decision* são: número de peças nos *buffer*s de entrada e saída das máquinas; número de nós para cada deslocamento de *AGVs*; e a distância a ser percorrida pelos *AGVs* a cada atendimento de transporte de peças.

No modelo considerado todas as tomadas de decisão se resumem a escolha de um entre os *buffer*s da fábrica que será atendido por um determinado *AGV*, por isso o Projeto *Decision* é formado por apenas uma classe: a classe *BufferDecision*. A seguir a definição desta classe:

- Classe *BufferDecision*
  - Esta classe é formada pelos membros:
    - *BufferInPriority*(número de peças no *buffer* de entrada, números de nós, distância):Prioridade – Um método que recebe os dados de uma possibilidade de atendimento em uma determinada situação de conflito, processa esta entrada no sistema de inferência *Fuzzy* e retorna um valor numérico que representa a prioridade ou o peso que a solução deve ter no processo de escolha de qual solução será usada para resolver o conflito.
    - *BufferOutChoose*(lista de *Load*s, nó atual):*Load* – Um método que recebe uma lista de *Load*s (cada *Load* é um par de *buffer* de entrada e de saída de uma máquina), o nó atual em que o *AGV* se encontra e retorna o melhor *buffer* de saída para atender dentro do sistema.
    - *BufferInChoose*(Lista<*Load*>, nó atual):*Load* – Um método que recebe uma lista de *Load*s, o nó atual em que o *AGV* se encontra e retorna o melhor *buffer* de entrada para atender dentro do sistema.
    - *BufferOut*Priority(número de peças no *buffer* de saída, número de nós, distância, número de peças no *buffer* de entrada):Prioridade

de atendimento – Um método que recebe uma possível solução para uma resolução de conflito na qual o *AGV* deve escolher um *buffer* de saída para atender. A possível solução é composta por: o número de peças que estão no *buffer* de saída que poderá ser escolhido; o número de nós que o *AGV* deverá percorrer para realizar o atendimento; a distância que o *AGV* deverá percorrer para realizar o atendimento; e o número de peças no *buffer* de entrada da máquina que forma par de *Load* com o *buffer* de saída lido no primeiro parâmetro deste método. O retorno do método é um valor numérico que indica a prioridade de escolha que deve ser dada para um *buffer* de saída.

- *HighestPriorityBufferOut*(Lista<*Load*>, nó atual):Prioridade – Um método que verifica dentro de uma lista de *Load*s qual *buffer* de saída deve ter a maior prioridade de atendimento.
- *HigherPriorityBufferIn*(Lista<*Load*>, nó atual):Prioridade – Um método que verifica dentro de uma lista de *Load*s qual *buffer* de entrada deve ter a maior prioridade de atendimento.
- *LoadToGoCollect*(Lista<*Load*>, prioridade mais alta, nó atual):*Load* – Um método que efetivamente escolhe o *buffer* de saída que será atendido dadas as condições do sistema.
- *LoadToGoRealease*(lista<*Load*>, prioridade mais alta, nó atual):*Load* – Um método que efetivamente escolhe o *buffer* de entrada que será escolhido para que um *AGV* deposite o produto que está sendo transportado.

O projeto *Decision* tem o papel de enviar e receber via *WebService* as decisões tomadas pelo sistema de Decisão *Fuzzy*. Este projeto faz a comunicação entre o Sistema de Controle e o Sistema de Decisão *Fuzzy*.

4.3.3.4 Projeto *AGV*Control*Load*

O Sistema de Controle acompanha todos os passos dos *AGVs* no sistema. Quando os *AGVs* entram em alguma área de *Load* de máquina o sistema de controle identifica este evento e realiza uma interferência no modelo no sentido de ativar ou não os *places* de controle que vão permitir que as transições a seguir sejam disparadas ou não.

A interferência realizada é no sentido de resolver um conflito no sistema: quando o *AGV* atinge uma área de *Load* é necessário saber se o *AGV* está carregado para então

enviá-lo para a área de *buffer* de entrada, ou saber se o *AGV* está livre para, neste caso, enviá-lo para o *buffer* de saída.

Cada entrada de *Load* possui dois caminhos: um para o *buffer* de entrada, e outro para o *buffer* de saída. Cada um dos caminhos é travado por um *place* de controle. O sistema de controle verifica o status do *AGV* para ativar o *place* de controle que dá acesso ao *buffer* de saída caso o *AGV* esteja vazio, ou para ativar o *place* de controle que dá acesso ao *buffer* de entrada quando o *AGV* estiver carregado com alguma peça.

O Projeto *AGV*Control*Load* é responsável por ouvir os eventos dos *AGVs* nestes pontos de entrada de *Load* (identificados no modelo em rede de Petri por *places* com o prefixo *AGV_PATH* no nome) e responder aos eventos liberando a passagem dos *AGVs* para os *buffer*s de entrada ou saída de acordo com a situação.

Este projeto é formado por duas classes:

- Classe *AGVOnLoadReleasingControl*
  - São membros desta classe
    - *Place* – Um tipo *place* que faz referência ao *place* real no modelo que representa a entrada da área de *Load* de uma máquina.
    - *AGVBuilder* – Um tipo auxiliar para realizar operações em nível de programação com o *AGV* responsável por disparar o evento.
    - *IsThereAGV*():lógico – Um método que retorna um valor verdadeiro/falso indicando se o *AGV* ainda se encontra na posição de entrada da área de *Load*.
    - *IsAGVFree*():lógico – Um método que retorna um valor verdadeiro/falso indicando se o *AGV* possui alguma peça em carga ou está livre.
    - *Control*() – Um método que verifica o status do *AGV* e então ativa o *place* de controle necessário para permitir que o *AGV* siga para a o *buffer* de saída ou de entrada.
- Classe *AGV*On*Load*Controle
  - São membros desta classe
    - *AGVOnLoadReleasingControl* – Um tipo auxiliar que permite que cada entrada de *Load* de máquina presente no sistema seja observada no sentido de responder aos eventos dos *AGVs*. Para cada entrada de área de *Load* existe um membro deste nesta classe.
    - *ControlPoint* – Um tipo auxiliar que faz uma chamada de execução das transições habilitadas no modelo após uma

determinada alteração nas marcas de um determinado ponto de controle. Este tipo recebe um método como parâmetro que será executado a todo momento enquanto o sistema de controle estiver operando. Este método deve ser responsável por realizar algum controle. Neste caso o controle realizado e verificar se o *AGV* deve ser dirigido para o *buffer* de entrada ou de saída do *Load* em questão. Cada *place* de entrada de *Load* do modelo deve ter um membro *ControlPoint* dentro desta classe.

- *Controller*() – Um método que chama todos os métodos passados como parâmetro para *ControlPoint* para realizar o controle dos pontos de entradas dos *Load*. Após a chamada do *Controller*() sempre que um *AGV* atingir o *place* de entrada de uma área de *Load* será realizado o teste que verifica se o *AGV* deve ir para a área de *buffer* de entrada ou de saída.
- *AGV_PathAction*() – Um método responsável por informar quais são os caminhos dos *buffer*s de entrada e de saída de cada área de *Load*. Existe um membro *AGV_PathAction*() para cada área de *Load* no modelo.

4.3.3.5 Projeto AGVDispatch

Este projeto é responsável pela circulação dos *AGVs* na via central do sistema de transporte. No sistema modelado os pontos de controle coincidem com os nós. Cada nó é um ponto de controle nos *places* do modelo. Estes nós ou pontos de controle dão acesso ás máquinas ao estacionamento e ao armazém.

Sempre que um *AGV* atinge um ponto de controle o sistema de controle observa a situação do sistema, inclusive o destino do *AGV*, e libera os pontos de controle seguintes para que o *AGV* siga o seu caminho.

Este projeto é formado pelas seguintes classes:

- *AGVDispatchingControl*
  - São membros desta classe:
    - *Place* – Um objeto para fazer referência ao *place* no modelo que representa o nó ou o ponto de controle em questão.
    - *AGVBuilder* - Um objeto para realizar operações em nível de programação com o *AGV* que atingir o ponto de controle.

- *AllLoads* – Uma lista com todas as áreas de *Load* do sistema. Esta lista é usada nos casos em que o *AGV* não possui destino e precisa ou carregar ou descarregar uma peça: nesta situação o sistema de controle envia os dados de todos os *Load*s para o sistema de decisão para que seja retornado o *Load* que será o novo destino do *AGV* para que este faça um atendimento de transporte.
- *ActualControlPoint* – Um número inteiro que representa o número que identifica aquele ponto de controle ou nó.
- *Load*sIn – Uma lista com *Load*s cujo os *buffer*s de entrada podem ser usados pelos *AGVs* para descarga de peças.
- *LoadLoad*s() – Um método que inicializa todos os *Load*s com seus respectivos *places* de *buffer*s de entrada e saída.
- *ActivateControlPlace*(*Place*) – Um método que ativa um *place* de controle de conflito colocando uma marca neste *place* para que uma determinada transição seja habilitada.
- *LoadToGoCollect*(Lista<*Load*>):*Load* – Um método que faz comunicação com o sistema de decisão para resolver qual *Load* terá seu *buffer* de saída atendido pelo *AGV* que está no ponto de controle, livre, e sem destino.
- *GoCollect*(*AGV*, Lista<*Load*>):*Load* – Um método que verifica qual *Load* deverá ter seu *buffer* de saída atendido e faz a alteração física na marca do *AGV* no sentido de alterar o seu destino para o *Load* que deve ser atendido. Este método retorna o *Load* que irá ser atendido pelo *AGV*.
- *LoadToGoRelease*(Lista<*Load*>): *Load* – Um método que faz a seleção dentre uma lista de *Load*s com *buffer*s de entrada os quais podem ser atendidos pelo *AGV*. Esta lista de *buffer*s de entrada é exatamente o roteiro que um determinado tipo de produto pode seguir a partir deste ponto de controle. Este método retorna o *Load* que deve ser usado para o *AGV* fazer a descarga do produto.
- *GoRelease*(*AGV*, Lista<*Load*>) – Este método verifica por meio do método *LoadToGoRelease*(Lista<*Load*>):*Load* qual *Load* deverá ser usado para que *AGV* deposite o produto e realiza a mudança na marca do *AGV* no modelo no sentido de alterar o destino deste *AGV* para que ele atenda o *Load* selecionado.

- Classe *AGVDispatchingController*
  - São Membros desta classe:
    - *cpPointDispatcher* – Um objeto que permite que métodos sejam passados como parâmetro para serem executados pelo sistema de controle. Cada nó ou ponto de controle no modelo possui um *cpPointDispatcher* relacionado nesta classe.
    - dispatcher – Um objeto que permite o controle dos *AGVs* nos pontos de controle. Cada ponto de controle está relacionado a um dispatcher.

        - *Controller*() – Este método inicia a execução dos controladores dos pontos de controle que são os métodos passados como parâmetro para os objetos do tipo *ControlPoint* dentro da classe.
        - *PointDispatcher*() – Este método é responsável pelo controle de cada ponto de controle. Ele é passado como parâmetro para os objetos do tipo *ControlPoint* dentro da classe. Para cada um ponto de controle do modelo existe um *PointDispatcher*. Neste método estão definidos os roteiros de cada produto a partir de cada máquina ou nó ou ainda ponto de controle.
- Classe ParkControl
    - São Membros desta classe:
        - *AllLoads* – Este objeto é usado para verificar o status de cada *buffer* de saída e entrada de *Load*s do sistema inclusive do armazém. Por meio dele é possível saber se o sistema possui ou não demandas de transporte.
        - *LoadLoad*s() – Método que inicializa cada *Load* do sistema com seus respectivos *places* no modelo.
        - *ControlAGVsReleasing*() – Método que verifica a existência de demandas de transporte e em caso positivo libera a saída de *AGVs* para realizar os atendimentos de transporte.
- Classe *ParkController*
    - São membros desta classe:
        - *ParkControlAction*() – Método que inicializa um objeto do tipo ParkControl e ativa o controle que será feito por este objeto.

A principal responsabilidade deste projeto é responder ao modelo por meio da alteração de marcas qual deverá ser o caminho que o *AGV* deve seguir a cada atendimento de transporte feito no sistema.

4.4 Procedimento para Execução do Sistema

4.4.1 Passos Iniciais

- Instalar o *Eclipse* Juno (instalador padrão no arquivo)
- Instalar apache *tomcat* 7 (instalador padrão no arquivo)
- Copiar o projeto *CPNControl* para o workspace do *Eclipse*
- Copiar as *lib* do *acesscpn* na pasta *lib* do *tomcat*
- Instalar o *Visual Studio* 2010 *ultimate* (instalador padrão no arquivo)
- Instalar o *CPNTools* (instalador padrão no arquivo)

- Instalar o *Matlab* R2012a (instalador padrão no arquivo)

4.4.2 Executando o Projeto

- Abrir o *Eclipse* e importe o projeto *CPNControl* do *workspace*
    - *File/import/general/existing projects into Workspace*
- Configurar o caminho de acesso as libs do *Access/CPN*
    - Botão Direito em *CPNControl / build path / configure build path / libraries / add external jars*
- Criar o *WebService*
    - *CPNControl / src / br.ufscar.dc.tear.cnptools /* clica com o direito em *PlaceWatcher.Java* / *WebService* / *create WebService*
    - Completar o diálogo de criação de *WebService*
- Configuração do escopo de aplicação, alterar o arquivo de configuração do *WebService*
    - WebServiceproject / *webcotent / webinf / server-config.wsld*
    - Incluir a linha *<ns1:parameter name="scope" value="application"/>*. A referência para incluir a linha é a linha em que é definida a operação "*setMark*"
- Com o *WebService* for criado, executar os seguintes passos
    - *start* no server *tomcat* e depois;
    - run em : *WebServiceprojectclient / webcontent / samplePlaceWatcherproxy/testcliente.jps*

- A fim de testar a comunicação do *WebService* com o modelo que será simulado:
    - Fazer chamada a algum *place* da rede a título de teste
        - *getMark (Java.lang.String)* / digita o nome do *place* / invoke
    - Abrir o arquivo da rede de Petri e aguardar até que ela seja verificada e carregada
- Após a realização do teste execute o *Visual Studio* 2012
    - Abrir o projeto tear
    - Atualizar as referências das referências as DLLs do *Matlab*
        - *MWarray.xml (../Matlab/R2012a/toolbox/dotnetbuilder/bin/win64/v2.0/)*
        - *Decision*Maker (no arquivo do projeto)
- No *Visual Studio* executar o projeto *Tear / Run*

4.5 Conclusão

Aqui foram apresentadas as ferramentas e os passos de execução necessários para a reprodução da validação da proposta apresentada neste trabalho.

O Sistema de Controle foi apresentado em subsistemas: Sistema de Interface de Comunicação, Sistema Controlador, e Sistema Decisor *Fuzzy*. Os recursos de programação de cada um destes bem como o seu papel dentro do Sistema de Controle foi descrito por meio de definições que seguem o paradigma de programção orientada a objetos e também por meio de diagramas da UML (*Unified Modeling Language*).

Capítulo 5

VALIDAÇÃO DA PROPOSTA

5.1 Considerações iniciais

O FMS adotado para a validação desta proposta é composto por: seis máquinas identificadas pelos nomes *M1*, *M2*, *M3*, *M4*, *M5*, e *M6*, cada uma delas está associada com um *buffer* de entrada e um *buffer* de saída; três AGVs; um estacionamento para AGVs; e um armazém com um *buffer* de entrada de produtos prontos e um *buffer* de saída de produtos que serão processados pelas máquinas. Este ambiente foi escolhido pelo fato de ser usado em diversos trabalhos realizados no laboratório Tear.

A modelagem do FMS descrito realizada por (ARAÚJO, 2006) sofreu algumas alterações para se adequar a proposta deste trabalho. Nesta seção os modelos de (ARAÚJO, 2006) serão apresentados e brevemente explicados. Na seção 2.2 Metodologia os modelos construídos e usados por este trabalho serão explicados detalhadamente. Em seguida, o sistema de controle será apresentado e explicado.

5.1.1 Modelos de (ARAÚJO, 2006)

A técnica *top-down* foi utilizada para a construção do modelo no trabalho de (ARAÚJO, 2006). Nesta técnica a principio é realizada a modelagem do esquema físico do FMS. Nesta primeira etapa é considerada a disposição dos recursos do FMS para que mais tarde seja elaborado o sistema de transporte.

O método *top-down* foi escolhido frente ao método *bottom-up*, pois confere a possibilidade de se abordar primeiramente o arranjo físico do sistema e então continuar a modelagem pelos elementos mais específicos. O detalhamento dos recursos do FMS é modelado em módulos distintos que estão ligados entre si.

Os componentes do sistema de manufatura foram modelados em Redes de Petri Colorida (CPN), como as rotas do AGVs, máquinas e *buffers*. Na Figura 5 é apresentado o modelo após a primeira etapa da modelagem.

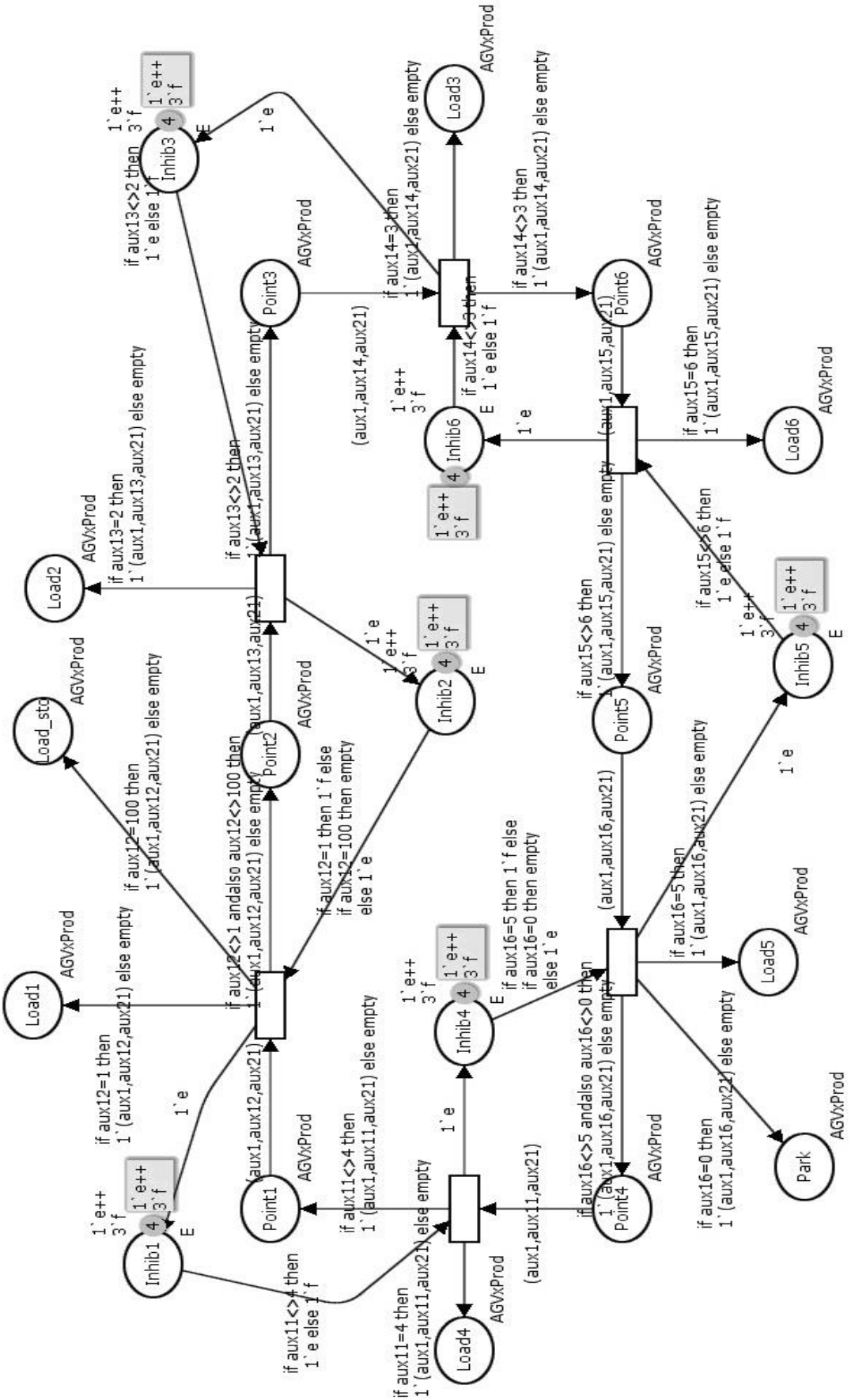


Figura 5 – Modelagem em CPN do arranjo físico do sistema (ARAÚJO, 2006).

Os caminhos possíveis para o AGV são determinados pelo arranjo físico do sistema. Pontos de controle são usados para restringir o número de AGVs no caminho.

O lugar *Load(i)* representa a área de entrada e saída da máquina *M(i)*, *Load_sto* representa o armazém, e *Park* representa o estacionamento dos AGVs.

Neste modelo, restrições referentes ao transporte são consideradas. Cada lugar nomeado *Point(i)* representa um nó ou ponto de entroncamento e tem um lugar associado *Inhib(i)*. *Inhib(i)* é uma estrutura usada para limitar a capacidade dos lugares. Assim, a presença de um único AGV em um nó e a presença de, no máximo, três AGVs na área de *buffers* de cada máquina, é garantida por essa estrutura. Marcas coloridas de cor *e* são usadas para o lugar de nó e marcas de cor *f* na área de *buffers* das máquinas.

Tipos de cores são definidos. A cor *AGVxProd* é uma tripla definida por *AGVxPROD* = (*N*, *D*, *TP*), em que *N* = nome do AGV, *D* = destino do AGV, *TP* = tipo de produto na carga do AGV.

Os possíveis valores para cada um dos elementos da tripla são:

- *N* {a, b, c}
- *D* {0, 1, 2, 3, 4, 5, 6, 100, 101}
    - 0 – O AGV não possui destino
    - 1 – O destino do AGV é a área de *buffers* da máquina *M1*;
    - 2 – O destino do AGV é a área de *buffers* da máquina *M2*;
    - 3 – O destino do AGV é a área de *buffers* da máquina *M3*;
    - 4 – O destino do AGV é a área de *buffers* da máquina *M4*;
    - 5 – O destino do AGV é a área de *buffers* da máquina *M5*;
    - 6 – O destino do AGV é a área de *buffers* da máquina *M6*;

- 100 – O destino do AGV é a área de *buffers* do armazém;
- 101 – O destino do AGV é o estacionamento;

- *TP* {*x*, *u*, *v*, *free*}.

Inscrições nos arcos são adicionadas no modelo para avaliar a situação e realizar o controle. As marcas referentes aos AGVs são adicionadas inicialmente no lugar referente ao estacionamento. As marcas são do tipo *AGVxProd*, mas neste estágio o tipo de produto é indicado por *free*, indicando que o AGV está livre.

As inscrições nos arcos com variáveis prefixadas por *aux(i)* estão relacionadas com informações de AGVs. Uma inscrição (*aux(i)*, *aux(y)*, *aux(z)*) representa uma marca *AGVxProd* que guarda informações de um AGV.

Na próxima etapa, o sistema foi dividido em outros modelos. Lugares de *fusão* foram usados para a construção do modelo em vários módulos por motivos de legibilidade e organização. Um lugar de *fusão* representa o mesmo lugar em módulos diferentes. Este é indicado pelo retângulo com o nome da fusão logo abaixo do círculo que representa o lugar. Como nem todos os caminhos possíveis foram modelados no modelo da Figura 5, alguns caminhos também foram representados nos módulos específicos do modelo. Módulos separados foram criados para cada máquina, para o armazém, e para o estacionamento dos AGVs.

Já com a quantidade, o tipo e ordem de peças definidas, os arcos ao longo do sistema são definidos para que possam apontar o caminho de uma peça ou veículo de transporte de acordo com os planos definidos previamente. As rotas dos produtos são definidas por meio das inscrições nos arcos.

No módulo do estacionamento, apresentado na Figura 6, a primeira solicitação é que um AGV vá até a estação de carga, pois é lá que ele será carregado com uma peça a fim de levá-la para o processamento em alguma máquina. O número inteiro que caracteriza essa solicitação é o número 100.

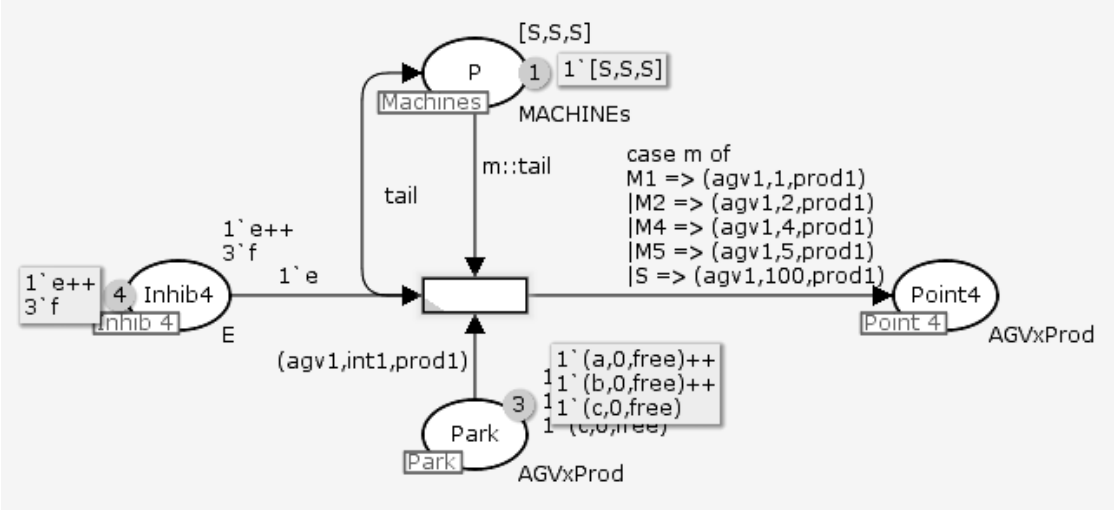


Figura 6 – Modelagem do estacionamento dos AGVs (ARAÚJO, 2006).

Nota-se que uma lista de requisições ou de roteiros está pré-definida na marca do lugar de nome *P*. A lista *[S,S,S]* representa os valores que serão associados ao destino do AGV. Quando a transição da Figura 6 for disparada, a variável livre *m* irá assumir como valor a cabeça da lista em *P* ao passo que um AGV em *Park* e uma marca e em *Inhib4* serão consumidos pela transição. No arco de entrada do lugar *Point4* uma função está pronta para avaliar a variável livre *m* e de acordo com o valor de m alterar o destino do AGV. Nesse caso o valor de *m* será *S* e, portanto, o destino do AGV será 100 que representa o armazém. Todos os movimentos dos AGVs estão pré-determinados em listas como a lista em *P* ao longo do modelo.

No módulo das máquinas, as estações de entrada e saída de produtos e os *buffers* de entrada e saída foram modelados. Na Figura 7 é apresentado o módulo do modelo para a máquina *M1*. Este módulo também foi usado para as outras máquinas com as alterações devidas.

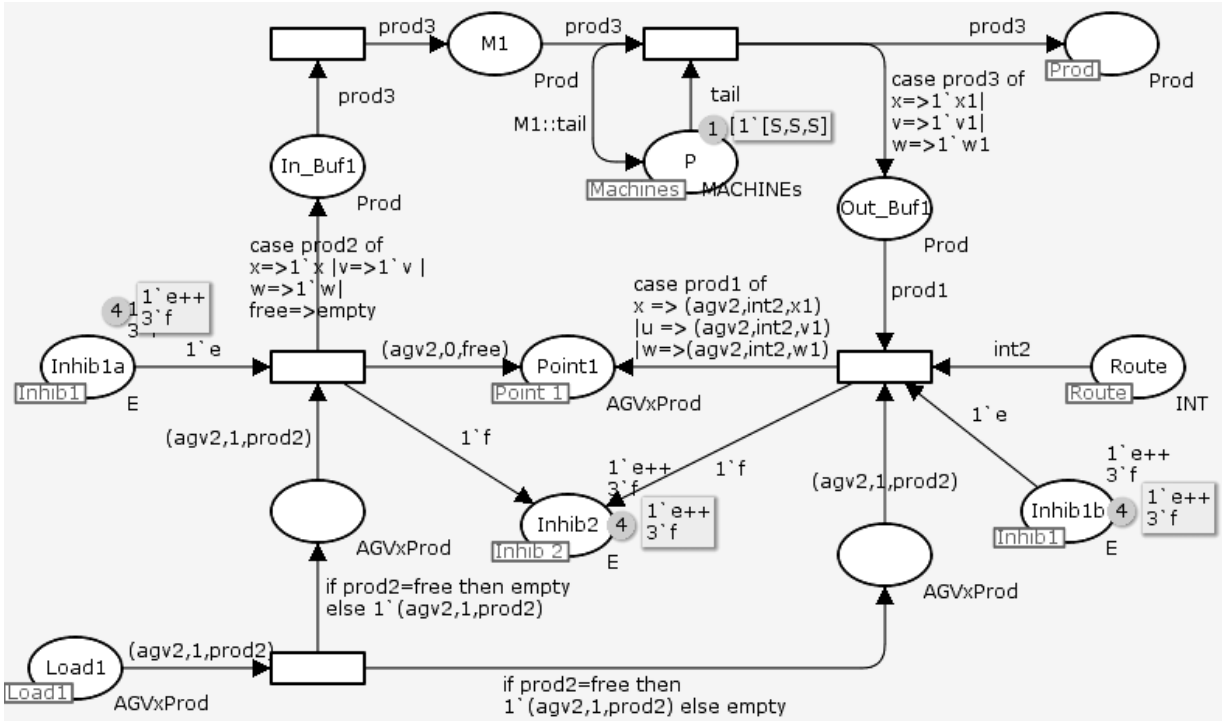


Figura 7 – Modelagem da Máquina *M1* (ARAÚJO, 2006).

Para determinar o próximo destino que o AGV deve levar o produto que está no *buffer* de saída da máquina em questão, o modelo da Figura 8 define, de acordo com cada matéria prima, uma lista de destinos.

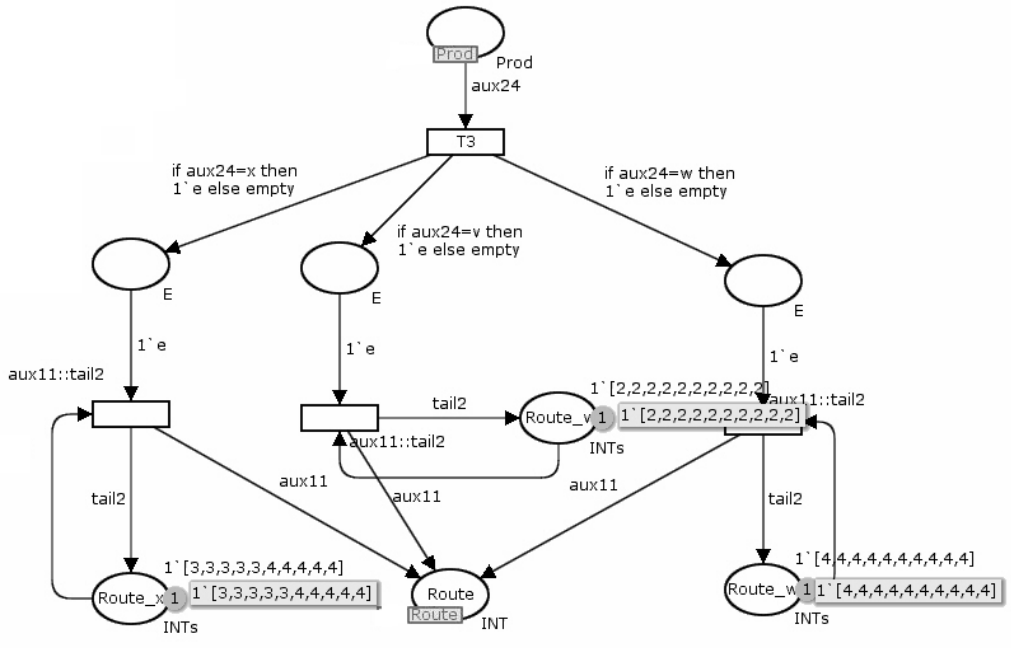


Figura 8 – Módulo de seleção do roteiro (ARAÚJO, 2006).

Na Figura 9, o módulo do armazém do modelo é representado. Neste módulo, um o AGV pode seguir dois caminhos de acordo com seu estado atual, caso o AGV esteja livre, este segue pelo caminho *Path_a*. Caso o AGV esteja carregado, este segue pelo caminho *Path_b*.

O lugar *Storage1* possui uma lista de produtos que deverão ser transportados pelos AGVs e processados pelas máquinas até que estejam prontos. Eles estão na quantidade e ordem em que devem ser produzidos. Com características semelhantes, o lugar *P* tem uma lista completa composta por marcas que representam máquinas e a ordem que elas serão usadas. Essas listas fazem parte do conjunto de elementos que compõem o sistema de controle do modelo de (ARAÚJO, 2006).

Assim que um AGV está pronto para receber a peça, a marca que encabeça a lista, mostrada em *Storage1*, será removida e fará parte da tripla do AGV. Isto também ocorre com a lista de marcas que representa as máquinas. Estas marcas têm a função de sinalizar ao AGV qual peça deve ser entregue em qual máquina.

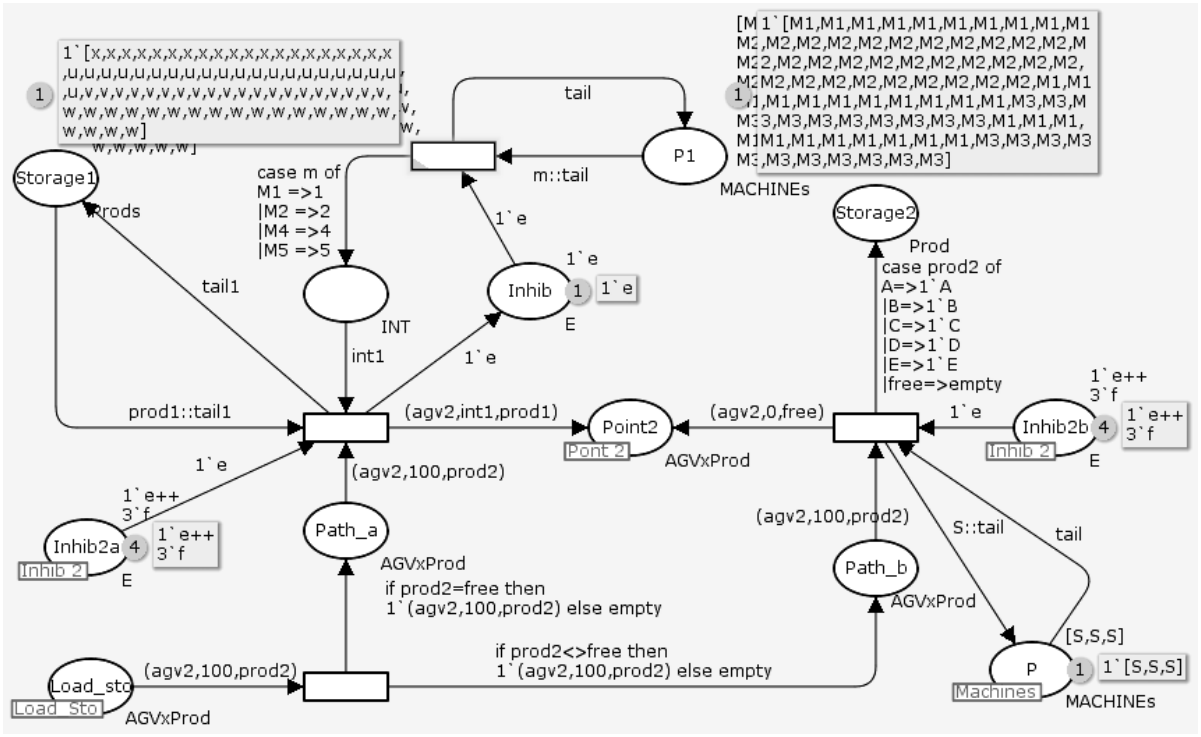


Figura 9 – Modelagem da Estação de Carga e Descarga (ARAÚJO, 2006).

O lugar *P* é um lugar presente também na modelagem das máquinas e do estacionamento. Este lugar é responsável por armazenar a lista de requisições para o uso dos AGVs.

5.2 Proposta

Para a realização deste trabalho foi feito um estudo de caso, cuja análise baseou-se no trabalho de (ARAÚJO, 2006). As etapas do processo de construção do modelo e do sistema de controle foram realizadas usando técnicas de modelagem e de engenharia de software. Estas técnicas serão descritas nas subseções seguintes.

5.2.1 Adaptação da estratégia de modelagem

Todas as etapas previstas na proposta de (ARAÚJO, 2006) foram realizadas para o FMS usado neste trabalho. Com isso foi alcançado um modelo de FMS que é a imagem do modelo apresentado em (ARAÚJO, 2006). Contudo o modelo sofreu adaptações para que a proposta do trabalho presente fosse verificada. Na Figura 10 é mostrado o resultado da modelagem do módulo principal após as adaptações.

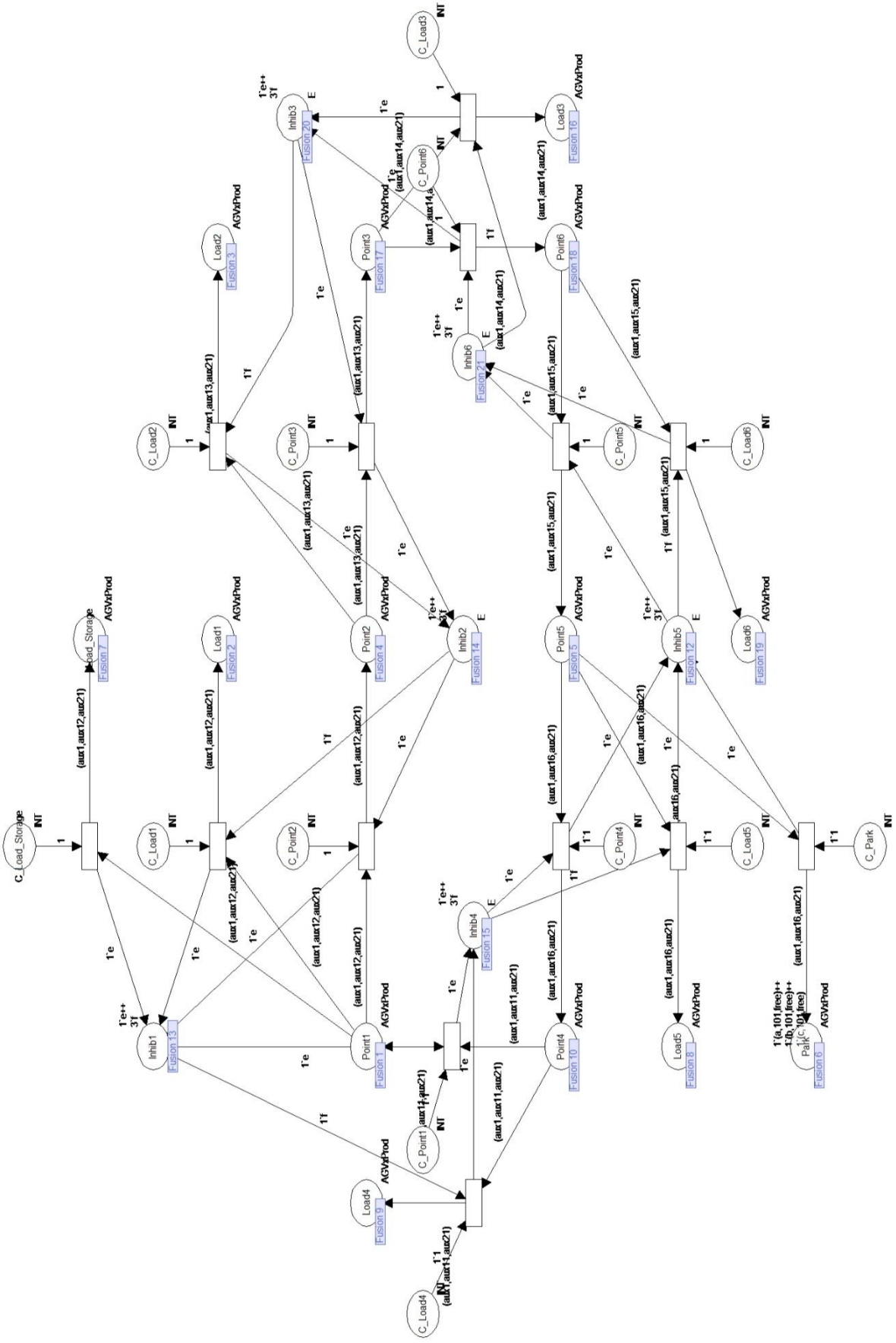


Figura 10 – Módulo principal do modelo do FMS usado na proposta.

A principio foi realizada a remoção de todas as transições de guarda dos lugares *Point(i)*. Estas transições têm arcos incidentes apenas nos lugares *Load(i)*, *Inhib(i)*, e no lugar *Point(j)* sucessor, com exceção da transição de guarda do *Point1* que, além de ter arcos incidentes nos lugares *Load1*, *Inhib1* e *Point2*, tem também um arco incidente no lugar *Load_sto*, e com exceção da transição de guarda do *Point5* que, além de ter arcos incidentes nos lugares *Load5*, *Inhib5* e *Point4*, tem também um arco incidente no lugar *Park*. Todos os arcos das transições removidas foram também removidos. Um exemplo da remoção da transição e dos arcos do lugar *Point1* pode ser observado na Figura 11.

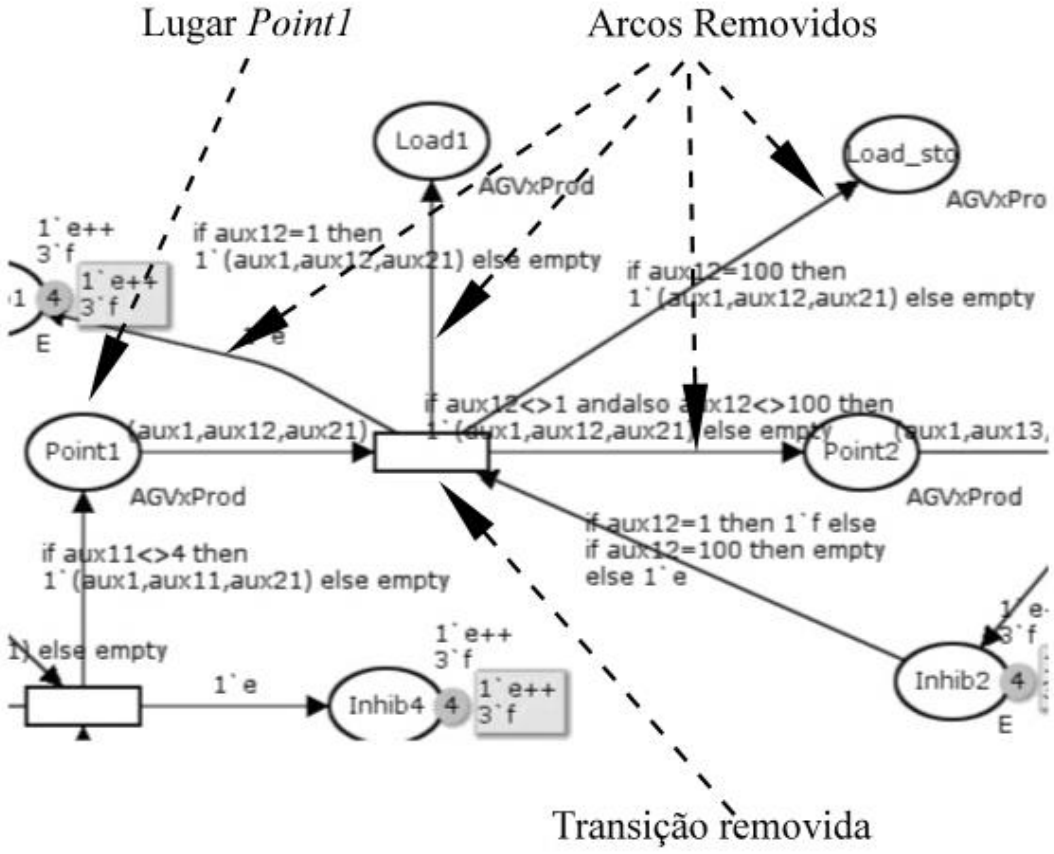


Figura 11 – Indicação da transição e dos arcos que serão removidos para o lugar *Point1*.

Para cada arco removido, com exceção do arco que incide sobre o lugar *Inhib(i)*, foi inserido um par de lugar e transição. A transição deste par recebe um arco incidente do lugar *Point(i)* e, a partir desta transição, incide um arco no lugar que, antes da remoção, recebia um arco da transição removida.

Sobre a transição do par adicionado foi colocado um arco incidente vindo do lugar do par. O lugar do par é um lugar de controle e a partir de cada transição dos pares adicionados incide um arco no lugar *Inhib(i)*.

No lugar *Inhib(j)* do lugar *Point(j)* que é sucessor do lugar *Point(i)* foi colocado, para cada par adicionado, um arco incidente sobre a transição deste par.

Na Figura 12 é observado um exemplo da adição de pares de lugar e transição para o *Point1*. No exemplo, o lugar *Inhib(j)* é o lugar *Inhib2* e o lugar *Point(j)* é o lugar *Point2*.

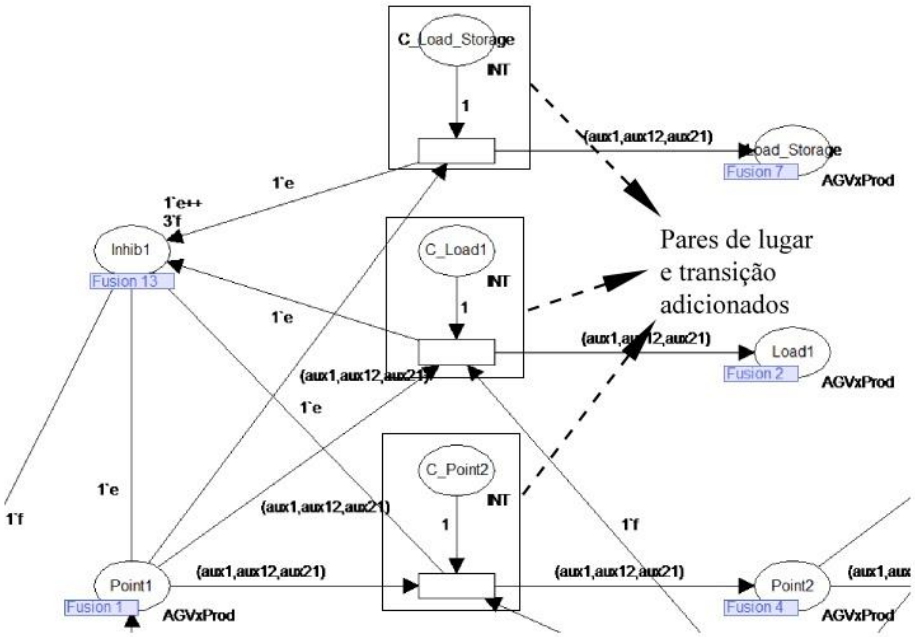


Figura 12 – Detalhe do modelo após a adição dos pares de lugar e transição.

Os lugares prefixados com *C_* são lugares de controle. O lugar de controle não representa nenhum recurso do FMS. Sua função é deixar a rede temporariamente parada em todas as situações de conflito. Este lugar de controle existe para cada uma das possíveis respostas ao conflito. Isto permite que, dada uma situação de conflito, um

sistema de controle externo avalie a melhor resposta para o conflito enquanto a rede está parada. Quando o sistema de controle obtiver a resposta de qual decisão será tomada para a resolução do conflito, este irá colocar uma marca no lugar de controle que representa a sua tomada de decisão e a rede do modelo voltará ao fluxo normal.

Esta abordagem substitui as funções presentes nos arcos do modelo proposto em (ARAÚJO, 2006), permitindo que o controle do modelo seja realizado por um sistema de controle que lê e imprime sinais no modelo resolvendo os conflitos e garantindo que o sistema não entre em *deadlock*. Na Figura 13 é apresentado o detalhe da região lugar *Point1* no modelo após a remoção da transição de guarda deste lugar e adição dos pares lugar e transição de controle. Na Figura 14 é apresentado o detalhe da região do lugar *Point1* no modelo do trabalho de (ARAÚJO, 2006).

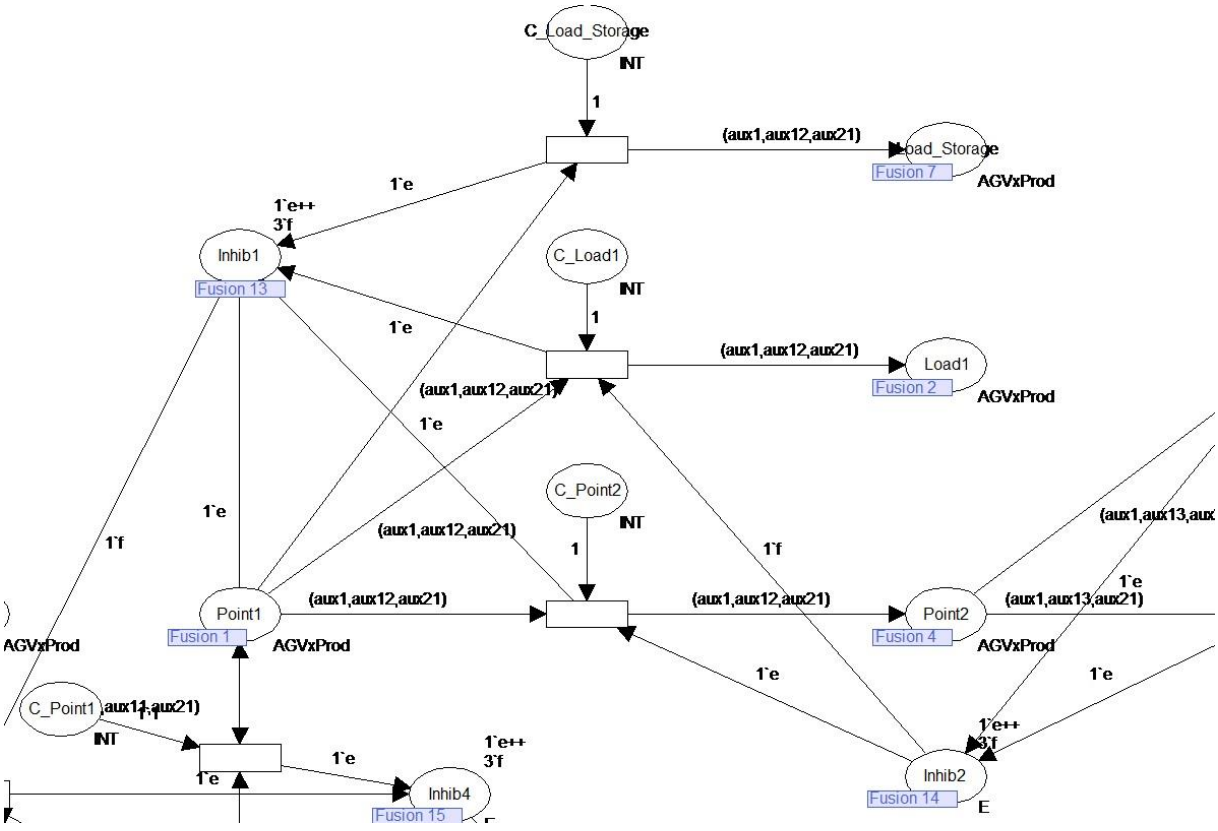


Figura 13 – Detalhe do modelo após a remoção da transição de guarda do lugar *Point1* e adição dos respectivos pares lugar transição.

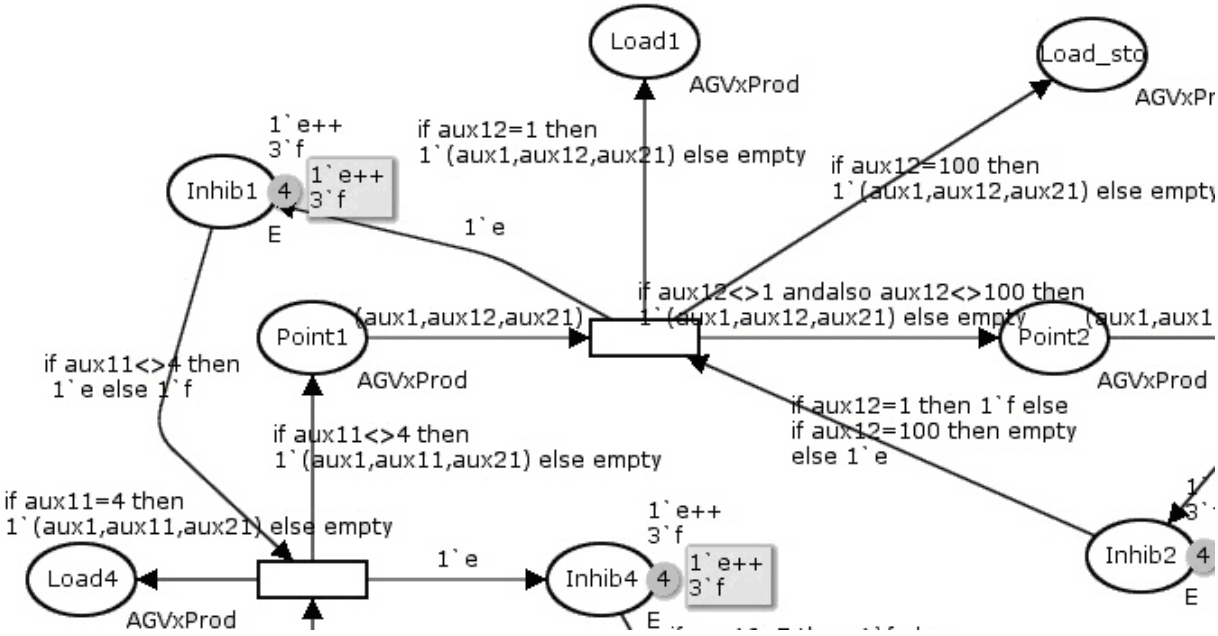


Figura 14 – Detalhe do lugar Point1 no modelo da proposta de (ARAÚJO, 2006).

Observa-se na Figura 13 um acréscimo de um lugar e uma transição para cada arco removido da transição de guarda do lugar *Point1* do modelo de (ARAÚJO, 2006). Este acréscimo é justificado pelo fato de que com essa abordagem é possível realizar alterações no controle sem alterar o modelo. Além disso, não há funções nos arcos como pode ser observado na Figura 13. Isto aumenta a legibilidade do modelo. Com a remoção das funções dos arcos, toda a responsabilidade de controle do modelo deverá ser realizada pelo sistema de controle externo.

Essa Abordagem foi adotada em todas as situações de conflito do modelo. Assim, foram removidas todas as funções dos arcos do modelo.

De acordo com o modelo apresentado na Figura 14, modelo de (ARAÚJO, 2006), um AGV que estivesse posicionado no nó *N1*, representado no modelo pelo lugar *Point1*, deveria avaliar as funções dos arcos de saída da transição de guarda do lugar *Point1* para seguir a sua rota. As possibilidades para este AGV seriam três: entrar na área de *buffers* da máquina *M1*, representada pelo lugar *Load_1*; entrar no armazém, representado pelo lugar *Load_sto*; ou seguir para o nó *N2* do caminho de AGVs, representado pelo lugar *Point2*.

Já no modelo apresentado na Figura 13, modelo da proposta deste trabalho, um AGV que acessar o nó *N1* enviará um sinal para o sistema de controle, que terá visão global do sistema incluindo todas as áreas de *buffers*, todas as máquinas e todos os produtos. Uma vez que recebe o sinal do AGV, e pondera as variáveis de todos os recursos do FMS, o sistema de controle devolve para o AGV a ação que este deverá realizar incluindo: coletar um determinado produto em algum *buffer* de saída, e depositar este produto em algum *buffer* de entrada.

Como foi dito, esta abordagem foi adotada em todos os pontos de conflito inclusive na escolha de produtos para o transporte.

Na Figura 15 é mostrado o detalhe de modelo do *buffer* de saída do armazém após a alteração e, na Figura 16 é apresentado o modelo do *buffer* de saída do armazém de acordo com o modelo de (ARAÚJO, 2006).

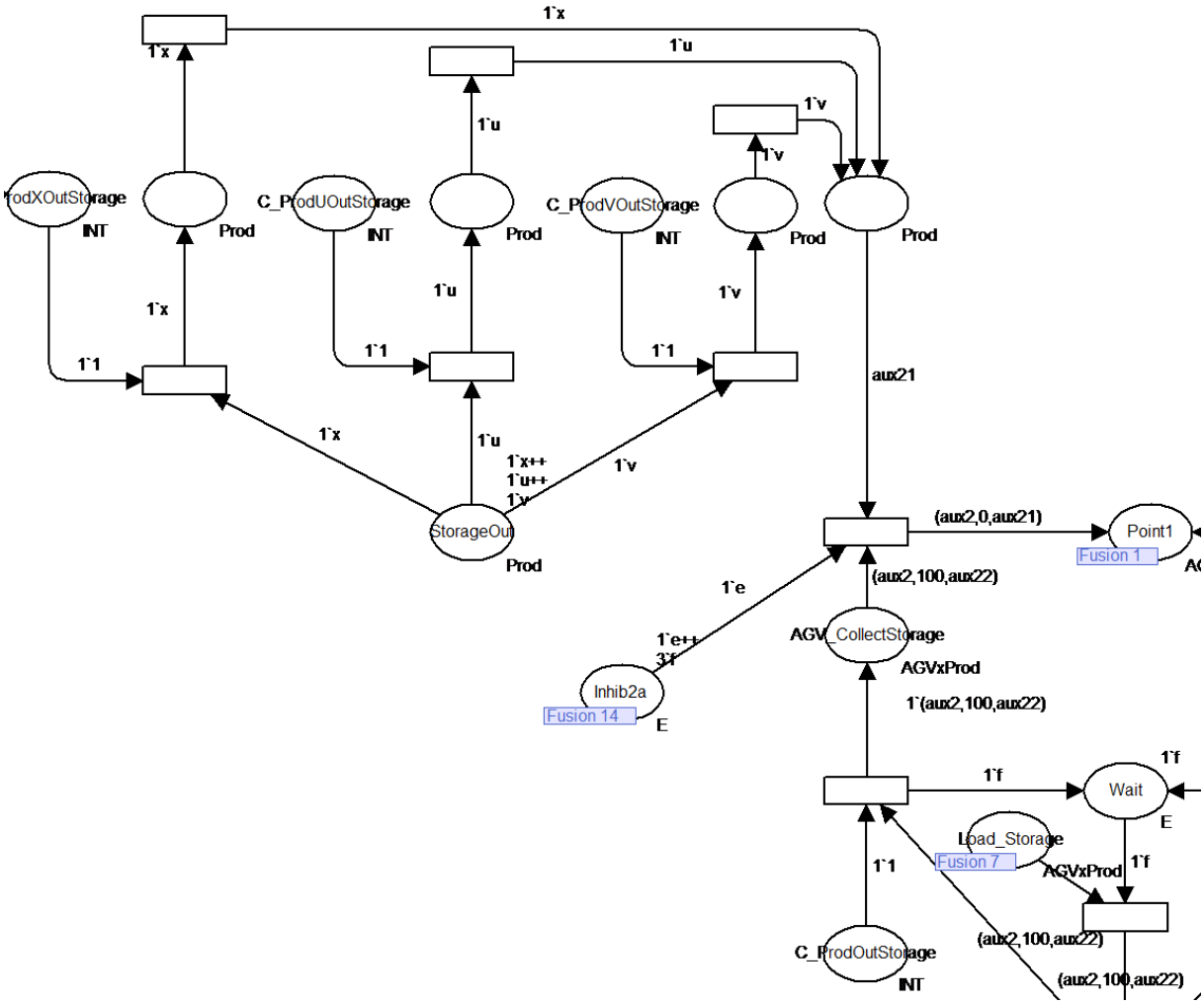


Figura 15 – *Buffer* de saída do armazém do modelo proposto neste trabalho.

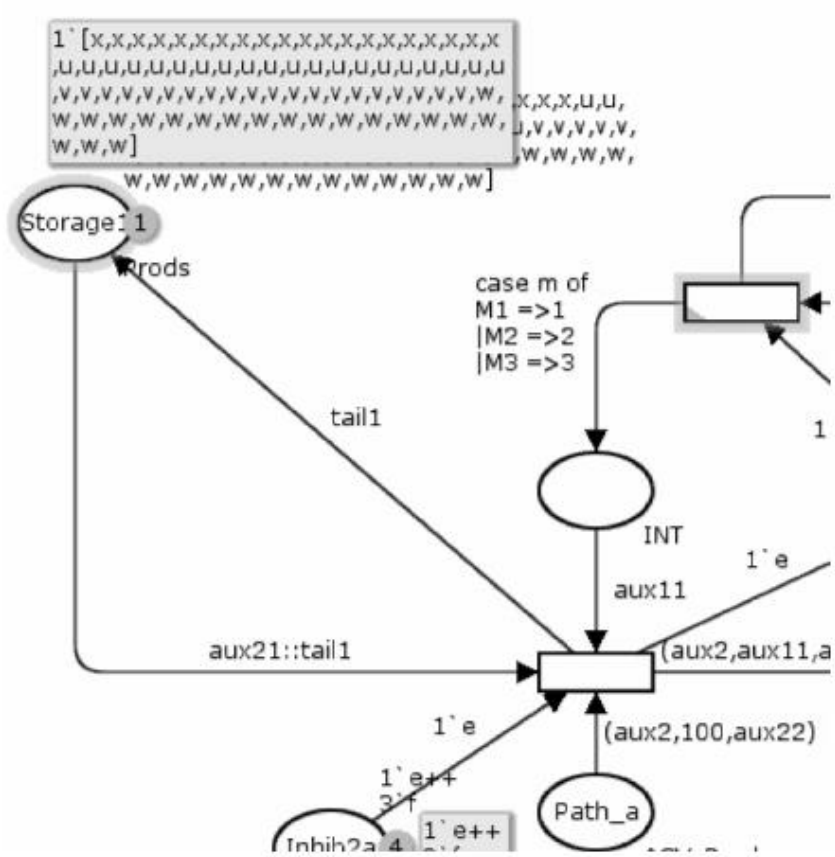


Figura 16 – *Buffer* de saída do armazém no modelo proposto em (ARAÚJO, 2006).

Como pode ser observado na Figura 15, o *buffer* de saída de produtos do armazém representado pelo lugar *StorageOut* não pode liberar qualquer produto de qualquer tipo sem que o sistema de controle adicione uma marca nos lugares de controle: *C_ProdXOutStorage*, *C_ProdUOutStorage*, ou *C_ProdVOutStorage*. Isto permite que o sistema de controle escolha qual produto será carregado por um determinado AGV. Caso o sistema de controle julgue que o melhor produto a ser carregado por um determinado AGV é do tipo *x*, este irá adicionar uma marca no lugar de controle *C_ProdXOutStorage*, dando assim condição para que um produto do tipo *x*, representado pela marca *x*, siga para a carga do AGV quando este AGV estiver à frente do *buffer* de saída do armazém, representado pelo lugar *AGV_CollectStorage*.

Na Figura 16 observa-se que uma lista no lugar *Storage*, que representa o *buffer* de saída do armazém, determina uma ordem estática. Neste caso os produtos são liberados para a carga seguindo uma fila pré-estabelecida pela lista.

Para os *buffers* de saída das máquinas a abordagem também foi adotada. Na Figura 17 é mostrado o *buffer* de saída da máquina *M1* no modelo proposto neste trabalho e, na Figura 18 é apresentado o *buffer* de saída da máquina *M1* no modelo da proposta de (ARAÚJO, 2006).

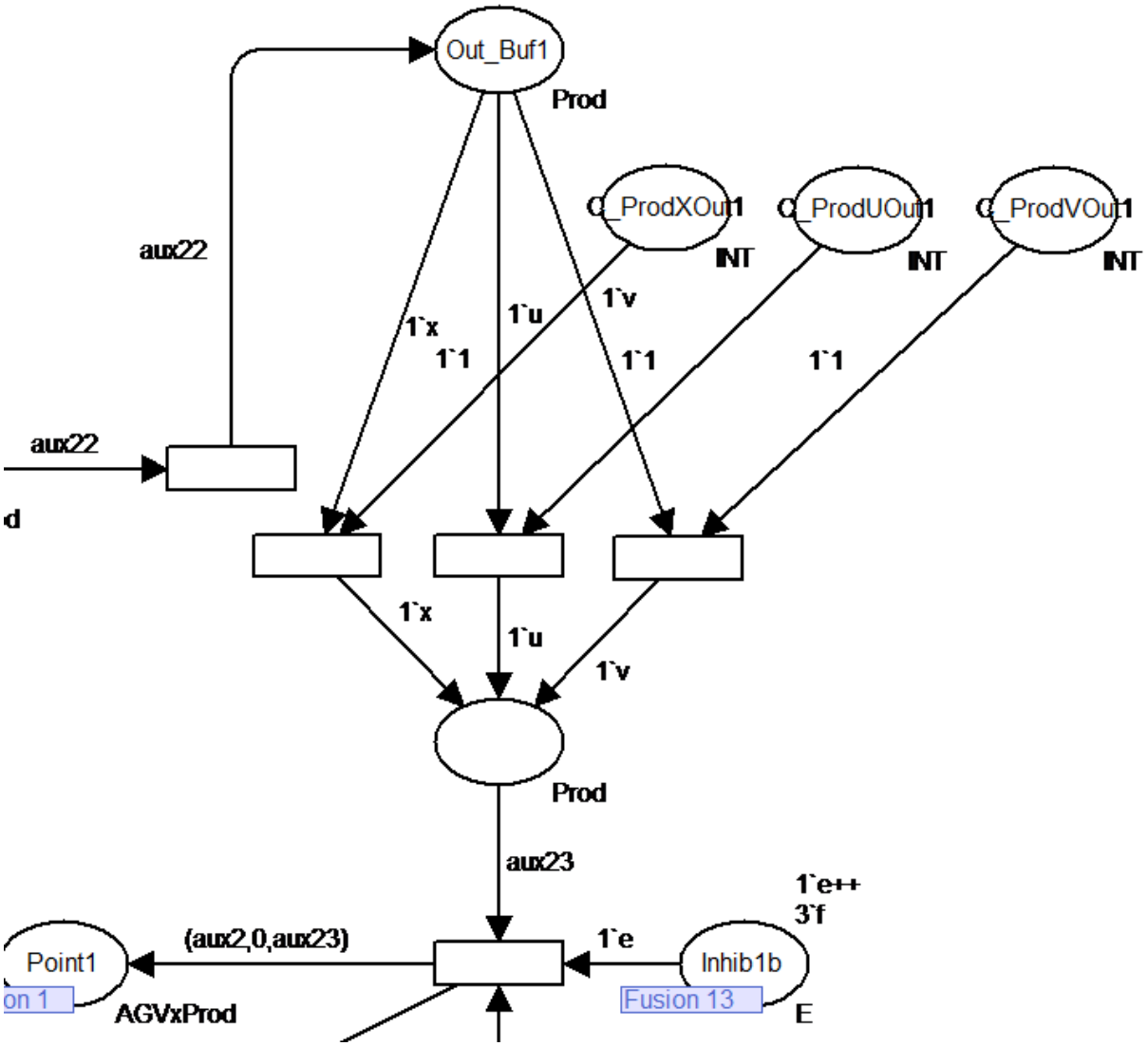


Figura 17 – *Buffer* de saída da máquina *M1* após alteração do modelo de (ARAÚJO, 2006).

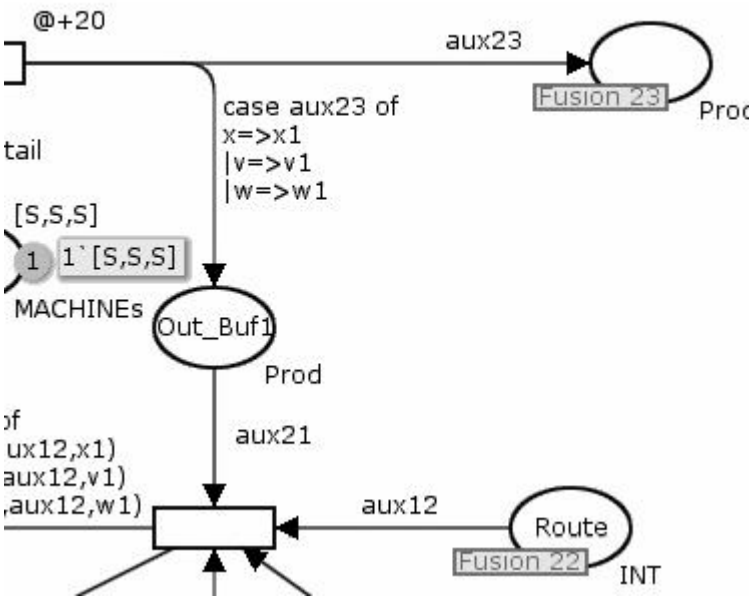


Figura 18 – *Buffer* de saída da máquina *M1* do modelo de (ARAÚJO, 2006).

Na Figura 18 observa-se que o *buffer* de saída da máquina *M1* representado no modelo pelo lugar *Out_Buf1* recebe lugares de controle para cada possibilidade de liberação de produto para um AGV. Estes lugares de controle são tratados pelo sistema de controle que irá definir qual produto deverá ser liberado em um determinado momento.

Na Figura 18, modelo de (ARAÚJO, 2006), o *buffer* de saída da máquina *M1* representado pelo lugar *Out_Buf1* tem uma função no arco de saída deste lugar que define qual produto irá ser transportado por um AGV que estiver à frente do *buffer* de saída desta máquina.

No modelo usado neste trabalho todos os conflitos de escolha de produtos, escolha de *buffers* de saída e de *buffers* de entrada serão resolvidos pelo sistema de controle. Por este motivo foi necessária a remoção das funções dos arcos do modelo de (ARAÚJO, 2006) que por sua vez resolve todos os conflitos e deixa o sistema livre de *deadlocks*. Esta proposta se baseia na hipótese de tratar variáveis do FMS na realização do controle do sistema. Para tanto, é necessário ter um sistema de controle que tenha

visão global do sistema a todo instante. Por motivos de simplicidade foi feita a remoção das funções dos arcos e a adição de pares de lugar de controle e transição.

Na subseção seguinte é apresentado o modelo completo do FMS em redes de Petri Colorida.

5.2.2 Modelo do FMS em Redes de Petri Colorida

O modelo desta proposta foi feito usando-se a ferramenta CPN Tools. A ferramenta foi escolhida por dar suporte a redes de Petri Colorida, permitindo assim que suas informações sobre os AGVs fossem persistidas nas marcas.

Seria possível modelar o FMS usando redes de Petri ordinárias sobre pena de um modelo com um voluma alto de lugares e transições. Isto diminuiria a legibilidade do modelo e diminuiria também, o nível de abstração do modelo.

Além disso, o CPN Tools permite que outros sistemas possam enviar e receber sinal de um modelo em rede de Petri. Dada a necessidade da construção de um sistema de controle externo que fizesse comunicação com o modelo, esta característica do CPN Tools daria condição para a realização do controle do modelo por este sistema de controle externo.

Na Figura 19 é apresentado o módulo principal do modelo. Neste módulo observam-se os nós dos caminhos dos AGVs representados pelos lugares identificados por *Point(i)*, em que i representa o nó *N(i)*. Neste módulo estão presentes também os lugares de controle que governam o fluxo de AGVs de um nó para outro, para as áreas de *buffers* das máquinas, para o armazém e também para o estacionamento. Os lugares identificados por *Inhib(i)* também podem ser observados aqui. Eles fazem parte das estruturas de controle que restringem a presença de até 1 AGV por nó, e de até 3 AGVs por área de *buffers* das máquinas. Para cada lugar *Point(i)* existe uma estrutura de controle *Inhib(i)* que restringe o número máximo de AGVs no nó *N(i)*, e na área de *buffers* da máquina *M(i)*.

Nos arcos estão as inscrições que representam informações da tripla que identifica cada AGV com seu nome, e informa o estado deste AGV com o seu destino e tipo de produto que está transportando. Nos arcos de saída de cada lugar *Inhib(i)*, e em seus arcos incidentes, estão inscritas as marcas *e* e *f*, que fazem parte da estrutura de controle de restrição de número máximo de AGVs em áreas de *buffers* de máquina e em nós. Sempre que um AGV acessar o nó *N(i)*, o lugar *Inhib(i)* perderá uma marca *e*, o que impedirá que outro AGV acesse o nó *N(i)* enquanto este estiver ocupado por pelo primeiro AGV. No momento em que um AGV abandona o nó *N(i)* uma marca *e* é devolvida para o lugar *Inhib(i)*. De maneira análoga, sempre que um AGV acessar a área de *buffers* de uma máquina *M(i)* que está localizada a frente do nó *N(i)*, o lugar *Inhib(i)* perderá uma marca *f*. Outros AGVs poderão acessar a área de *buffer* da máquina *M(i)*, se e somente se houver marcas *f* em *Inhib(i)*. Sempre que um AGV se desloca da área de *buffer* de uma máquina *M(i)* uma marca *f* é devolvida para o lugar *Inhib(i)*.

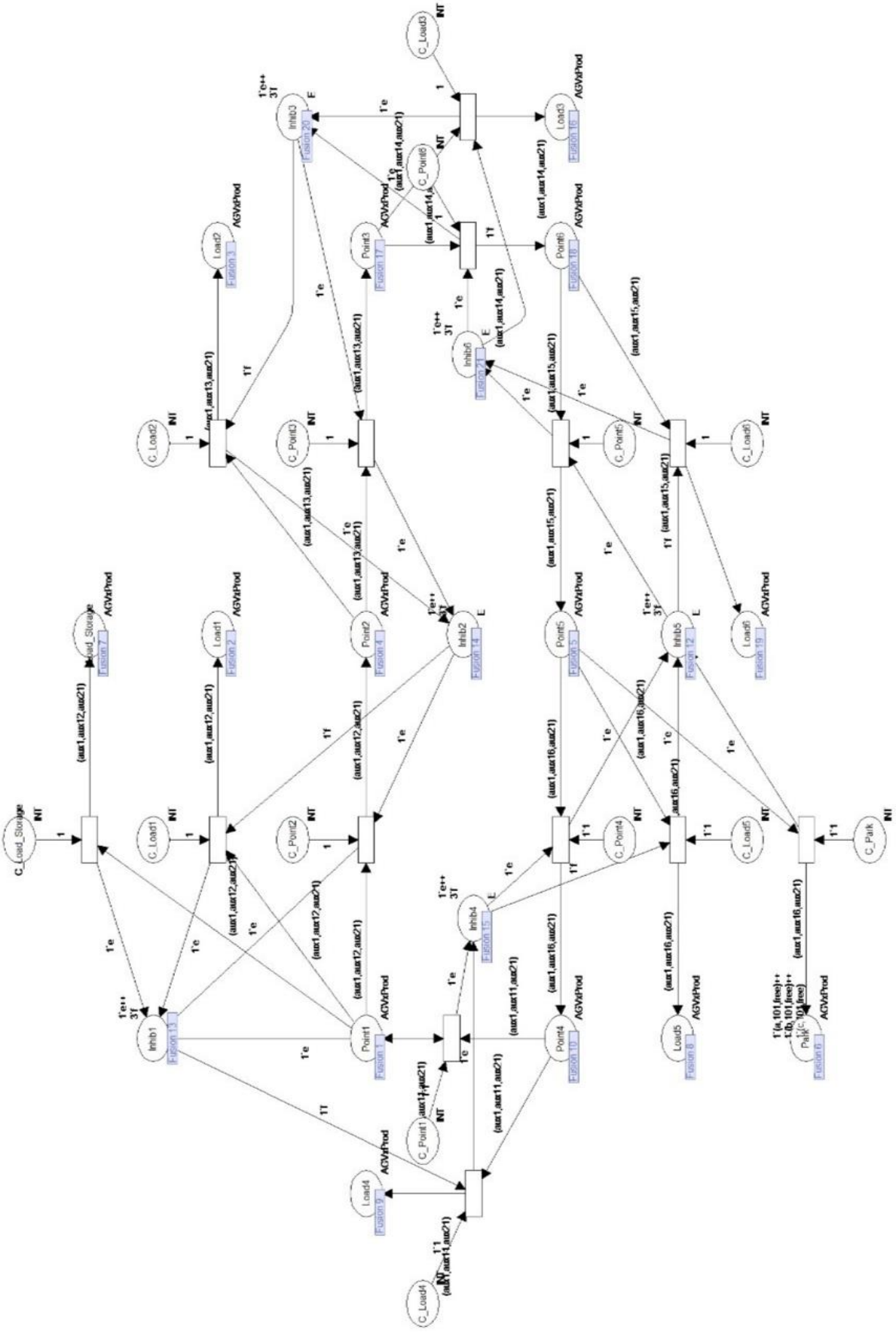


Figura 19 – Módulo principal do modelo proposto.

Na Figura 20 é apresentado o módulo de estacionamento dos AGVs. Neste módulo o lugar *Park* armazena os AGV até que o sistema inicie o processo de produção. Uma vez o processo iniciado os AGVs se dirigem às áreas de *buffers* das máquinas ou ao armazém para responder às requisições de transporte do sistema. Assim que não houver mais demanda de transporte no sistema os AGVs retornam ao lugar *Park*. Neste módulo encontra-se o lugar de controle *C_Release_AGV* que é responsável por liberar os AGVs sempre que houver demanda de transporte de produtos.

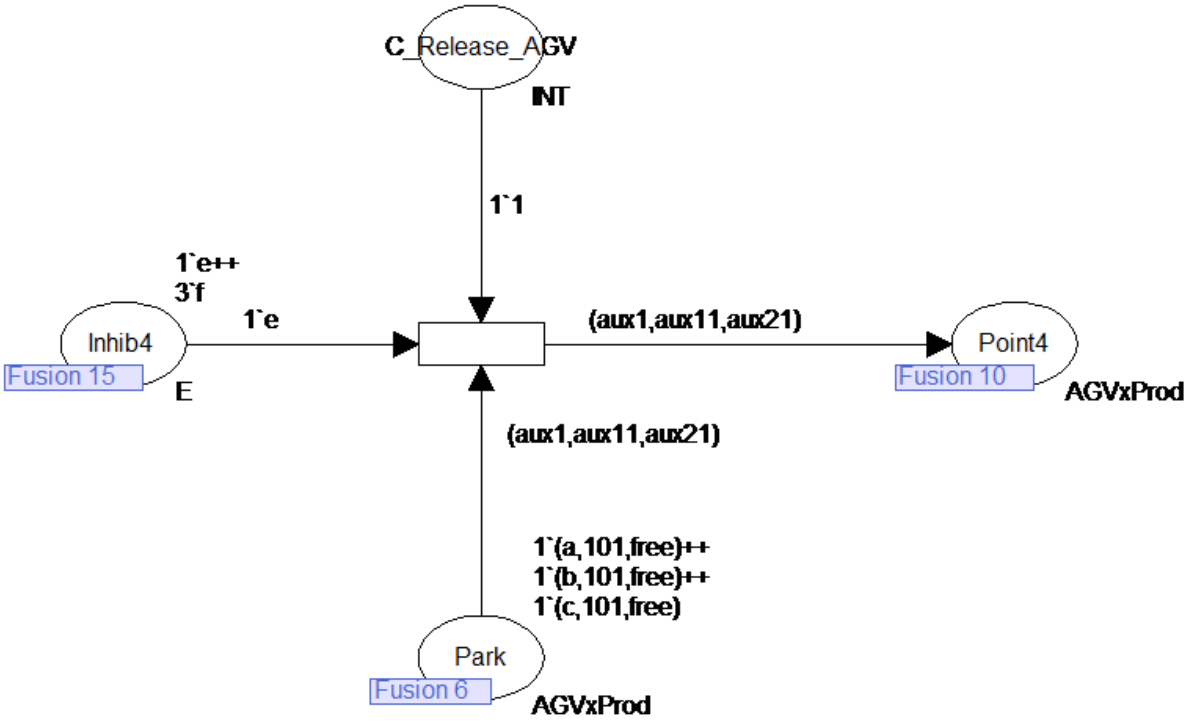


Figura 20 – Módulo de estacionamento do modelo.

O módulo do armazém é apresentado na Figura 21. O acesso do AGV ao armazém é dado por meio do lugar *Load_Storage*. Uma vez neste lugar o AGV irá aguardar qualquer outro AGV que esteja carregando ou descarregando produto. Este comportamento é garantido pelo lugar *Wait* que, assim como os lugares *Inhib(i)*, controla o número máximo de AGVs em um determinado lugar. Neste caso o lugar *Wait* está restringindo que apenas um AGV carregue, ou descarregue, por vez. Quando um AGV termina o processo de carregar ou de descarregar o produto, o lugar *Wait* recebe uma marca novamente permitindo que outro AGV que esteja no lugar *Load_Storage* possa carregar ou descarregar. A estrutura dos *buffers* de saída e de entrada não só do armazém, mas também de todas as máquinas foi construída desta forma para garantir

que cada tipo de produto seja atribuído a um determinado AGV. Quando um AGV chega em qualquer área de *buffers* para transportar um produto o sistema de controle já definiu qual tipo de produto ele irá transportar. Para que o sistema de controle garanta que este AGV colete o produto correto é necessário isolar cada AGV, uma vez que até três AGVs podem estar na área de *buffers* de uma máquina ou do armazém. Deste modo, quando o AGV acessar o lugar *AGV_CollectStorage* o sistema de controle será sinalizado e irá colocar uma marca em um dos lugares de controle: *C_ProdXOutStorage*, *C_ProdUOutStorage*, ou *C_ProdVOutStorage*, de acordo com o tipo de produto previsto para aquele AGV. Depois que o sistema colocar uma marca em algum dos lugares de controle. O lugar *Storage_Out* terá a condição necessária para executar a transição relacionada com o lugar de controle que recebeu a marca do sistema de controle. Assim o produto, do tipo determinado, representado pela marca que saiu do lugar *Storage_Out*, será carregado pelo AGV. Este AGV então seguira para o lugar *Point1* e irá realizar a sua jornada até a área de *buffers* determinada pelo sistema de controle.

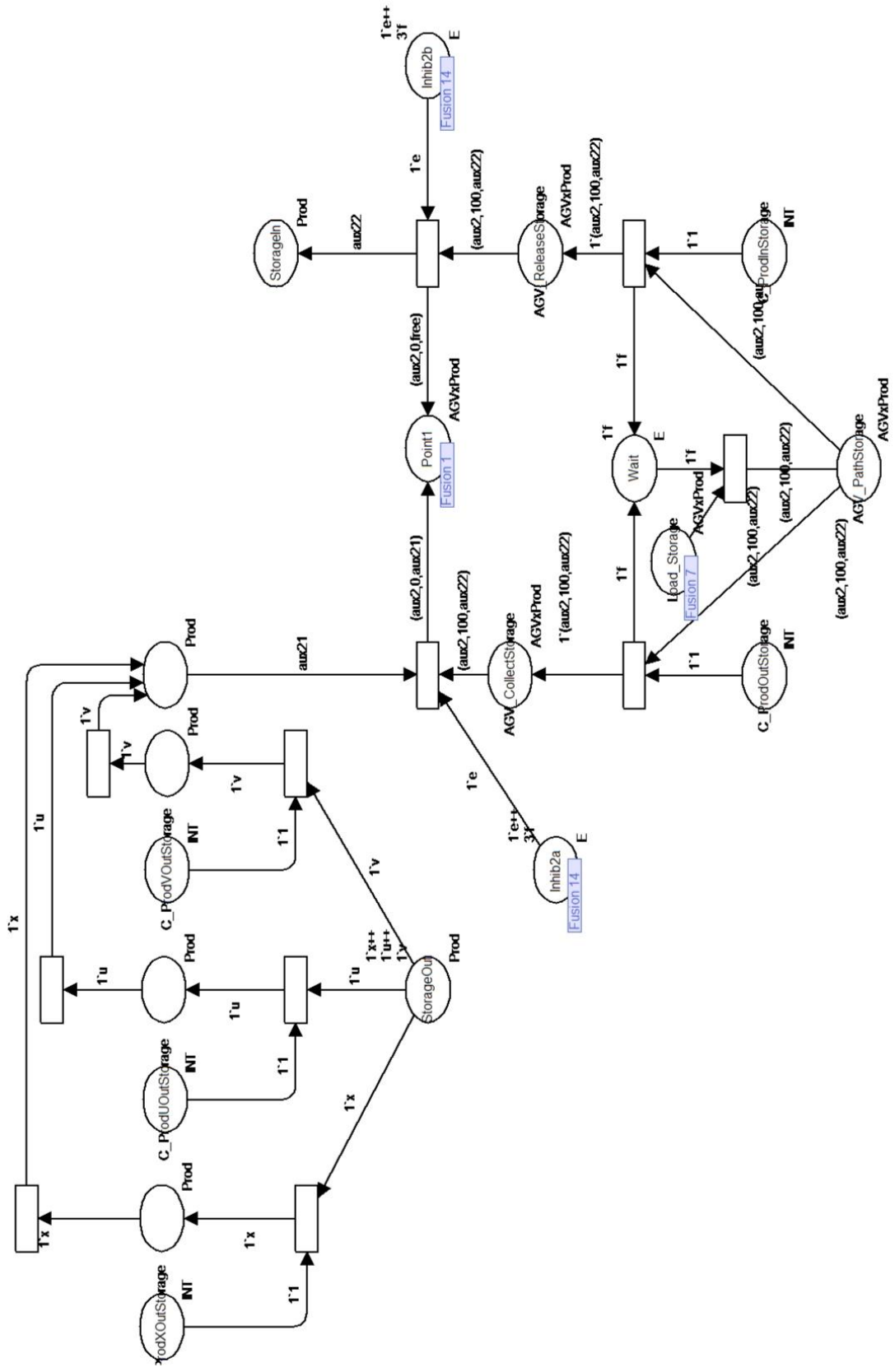


Figura 21 – Módulo do armazém do modelo.

A seguir, na Figura 22, é apresentado o módulo da máquina *M1* e sua área de *buffers*. O acesso do AGV para a área de *buffers* da máquina *M1* é feito por meio do lugar *Load1*. Este lugar permite a presença de até três AGVs, assim como todas as áreas de *buffers* das máquinas e do armazém. Quando um ou mais AGVs acessam este lugar, a transição de guarda do lugar não permite que este AGV siga para o *buffer* de entrada ou de saída, caso haja outro AGV em um desses *buffers*. O lugar *Wait* garante este comportamento, pois sempre que um AGV ocupar um dos *buffers*, o lugar *Wait* perderá sua marca *f* que é responsável por dar condição de disparo a transição de guarda do lugar *Load1*. Ao término do processo de carga ou descarga do AGV que está ocupando um dos *buffers* o lugar *Wait* ganha uma marca *f* que dará condição para que um próximo AGV acesse o *buffer* de entrada ou de saída da máquina.

Antes de acessar qualquer um dos dois *buffers* o AGV se posiciona no lugar *AGV_Path1*. Sempre que um AGV ocupa este lugar o sistema de controle é sinalizado. Um AGV nesta localização configura uma situação de conflito, uma vez que o AGV tem dois caminhos a seguir ou dois lugares: o lugar *AGV_Collect1*, ou o lugar *AGV_Release1*. Esta decisão é tomada pelo sistema de controle de acordo com o estado do AGV. Caso o AGV esteja livre ou sempre produto em sua carga o sistema de controle irá colocar uma marca no lugar de controle *C_ProdOut1* entendendo que o AGV está livre e deve seguir para o *buffer* de saída onde deverá coletar um produto de tipo determinado. Caso contrário, o sistema de controle irá colocar uma marca no lugar de controle *C_ProdIn1*, dando condição para que a transição que leva o AGV para o lugar *AGV_Release1* seja disparada. Neste último caso o AGV depositará o seu produto no *buffer* de entrada da máquina *M1* na qual será realizado um dos estágios de produção daquele produto.

Um AGV que cujo estado é livre e acessa o lugar *AGV_Path1* será enviado logo em seguida para o lugar *AGV_Collect1*. Neste ponto o sistema de controle será sinalizado de que determinado AGV está naquele lugar. Seja este AGV o AGV de nome *a*. O AGV *a* está com todas as condições para carregar um produto e realizar o transporte

deste. No entanto nenhum produto irá chegar até o AGV *a* dado que estes produtos estão no lugar *Out_Buf1* , que representa o *buffer* de saída da máquina, e os lugares de controle *C_ProdXOut1*, *C_ProdUOut1*, e *C_ProdVOut1* estão causando estado de *deadlock*, ou não possuem marca para dar condição para que uma marca de produto chegue até o AGV. Neste momento, sabendo da presença do AGV a no lugar *AGV_Collect1*, o sistema de controle irá consultar em uma tabela de ordens, a última ordem que foi data ao AGV *a*. Esta ordem é uma tupla definida por *O = (N, BS, TP, BE)* onde *N* = nome do AGV, *BS* = *buffer* de saída que o AGV deverá atender, *TP* = tipo do produto que o AGV deverá coletar, e *BE* = *buffer* de entrada onde a carga deverá ser depositada. A tabela de ordens é dinâmica e está acessível ao sistema de controle em qualquer instante da execução do sistema.

Sempre que um determinado AGV estiver em situação de conflito e sem ordem, isto é, em estado livre, sem destino e em um nó do caminho de AGVs o sistema de controle irá analisar o contexto global do sistema considerando as variáveis do sistema e, a partir desta análise, irá elaborar uma ordem para este AGV. Elaborada a ordem, o *AGV(n, d, free)* recebe um número inteiro *BS* que representa o destino do *buffer* de saída que este AGV deverá ir coletar o produto do tipo *TP* definido na ordem. Neste momento o *AGV(n, d, free)* tem seu estado alterado para *AGV(n, BS, free)*.

O AGV *AGV(n, BS, free)* chegará então ao lugar *AGV_Collect(BS)*. Assim, o sistema de controle irá consultar na tabela de ordens a última ordem para o AGV de nome *n*. A ordem retornada *O = (n, BS, TP, BE)* será usada pelo sistema de controle para colocar uma marca no lugar de controle *C_Prod(TP)Out(BS)*. Assim o novo estado do AGV será *AGV(n, BE, TP(p))* tendo como novo destino *BE* e como carga um produto do tipo *TP*. Este AGV acessará o caminho de AGV por meio do lugar *Point(i)* e seguirá até o nó de *BE*: seu novo destino onde depositará o produto em sua carga.

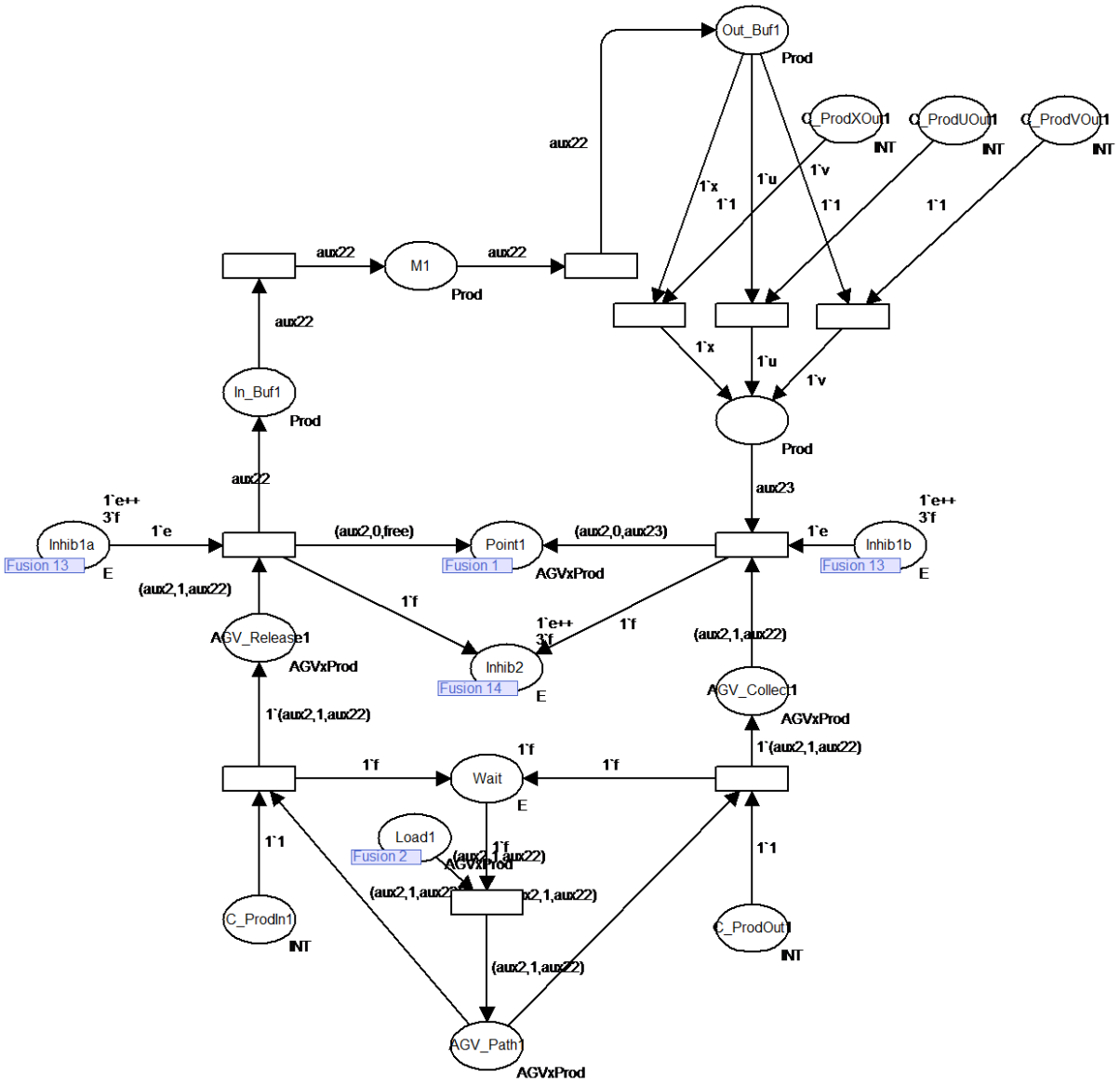


Figura 22 – Módulo da área de *buffers* da máquina *M1*.

Além destes módulos aqui apresentados existem outros cinco referentes às máquinas *M2*, *M3*, *M4*, *M5* e *M6*. Todos eles seguem a mesma estrutura de construção e comportamento do módulo da máquina *M1* e, portanto, dispensam detalhamento.

Nas figuras a seguir são apresentados os módulos das máquinas *M2*, *M3*, *M4*, *M5*, e *M6*.

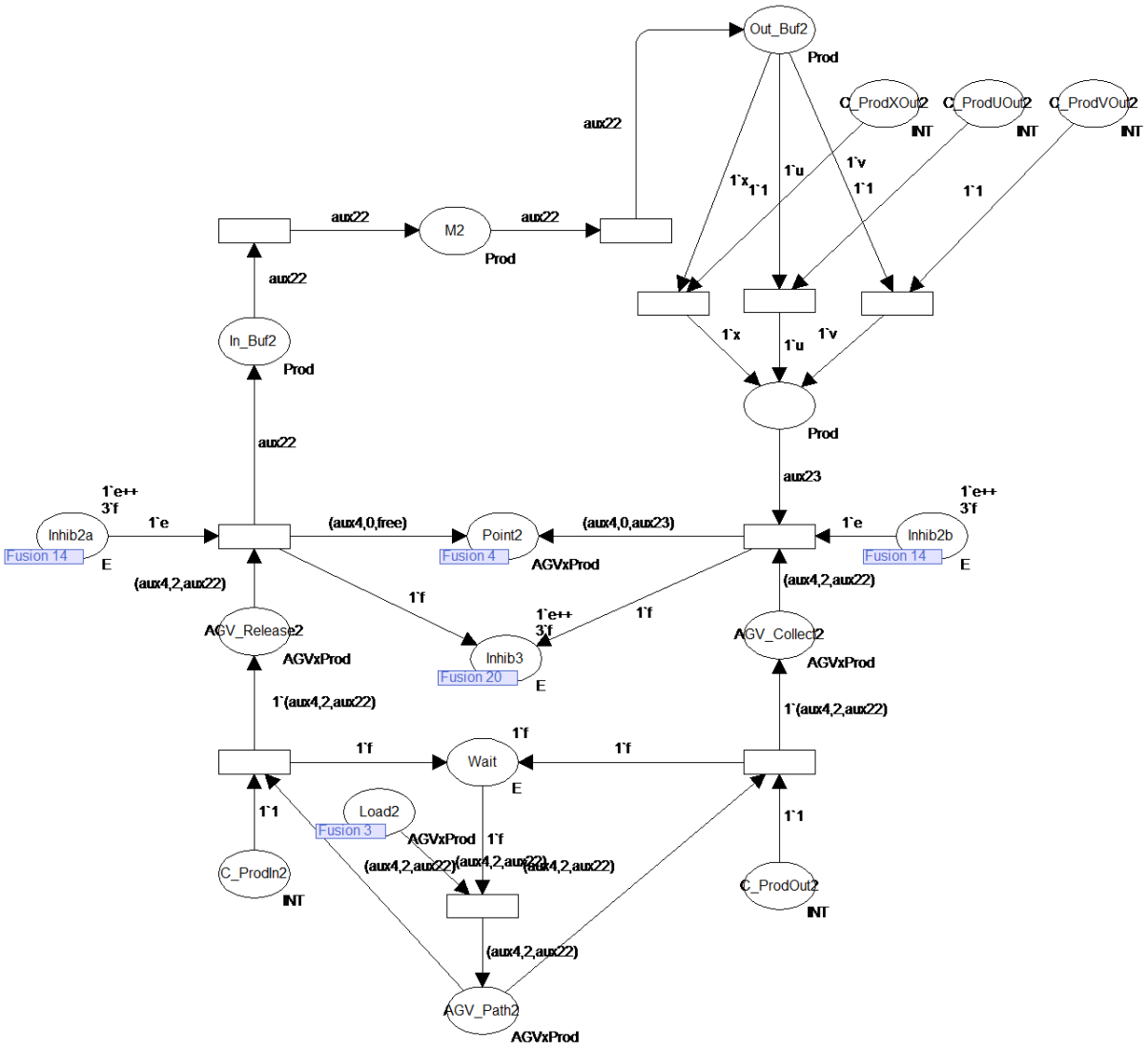


Figura 23 – Módulo da área de *buffers* da máquina *M2*.

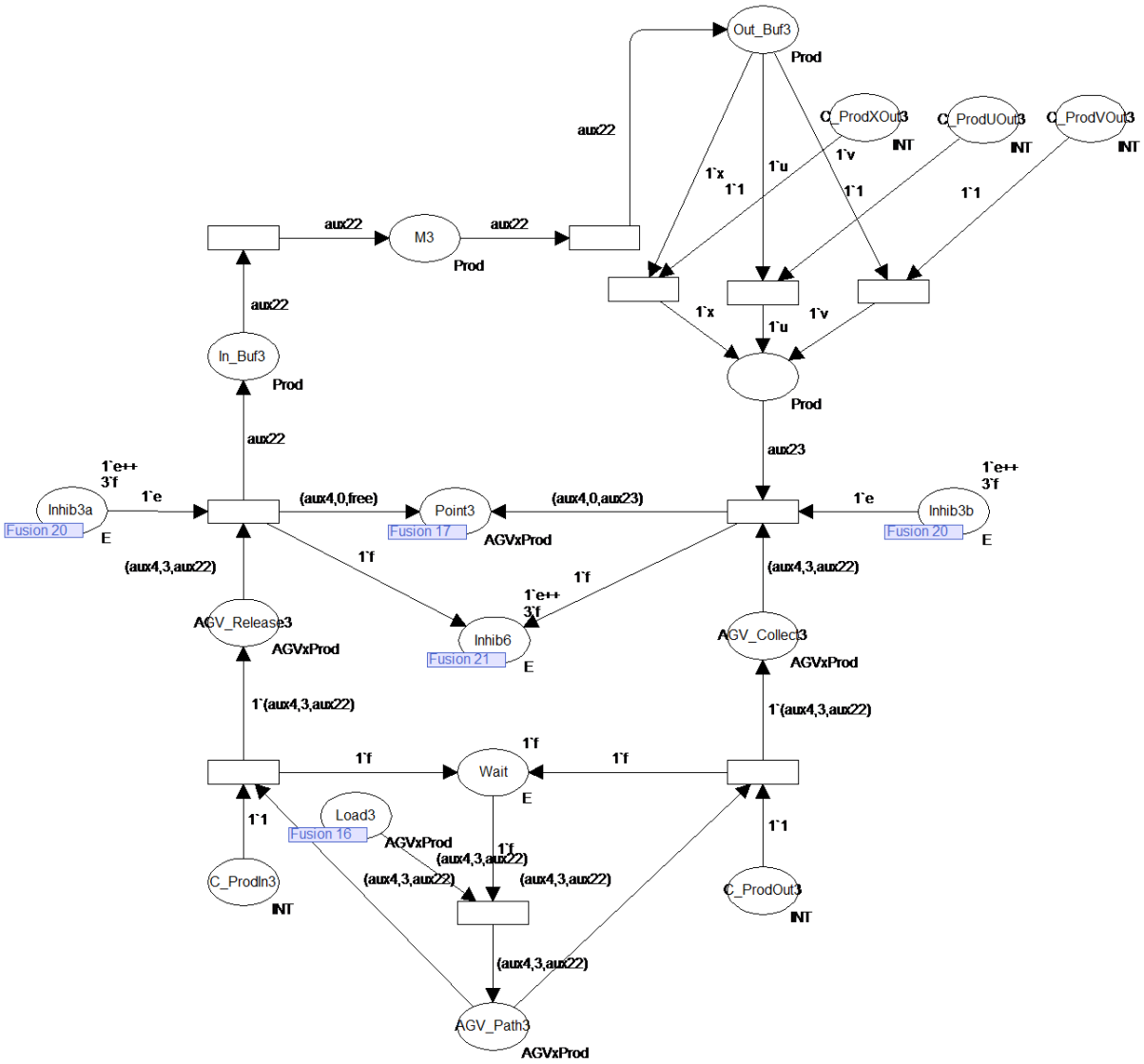


Figura 24 – Módulo da área de *buffers* da máquina *M3*.

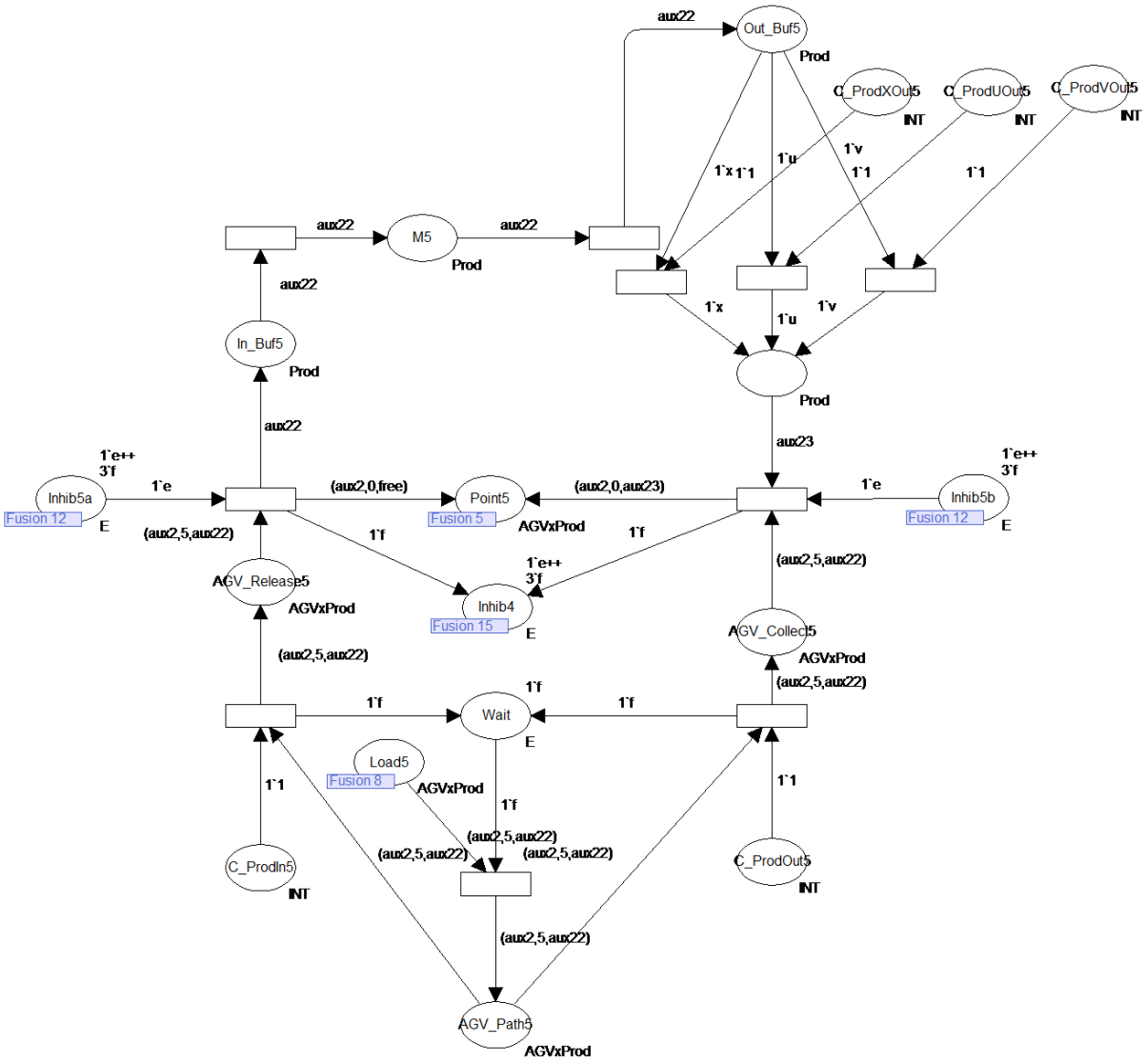


Figura 25 – Módulo da área de *buffers* da máquina *M4*.

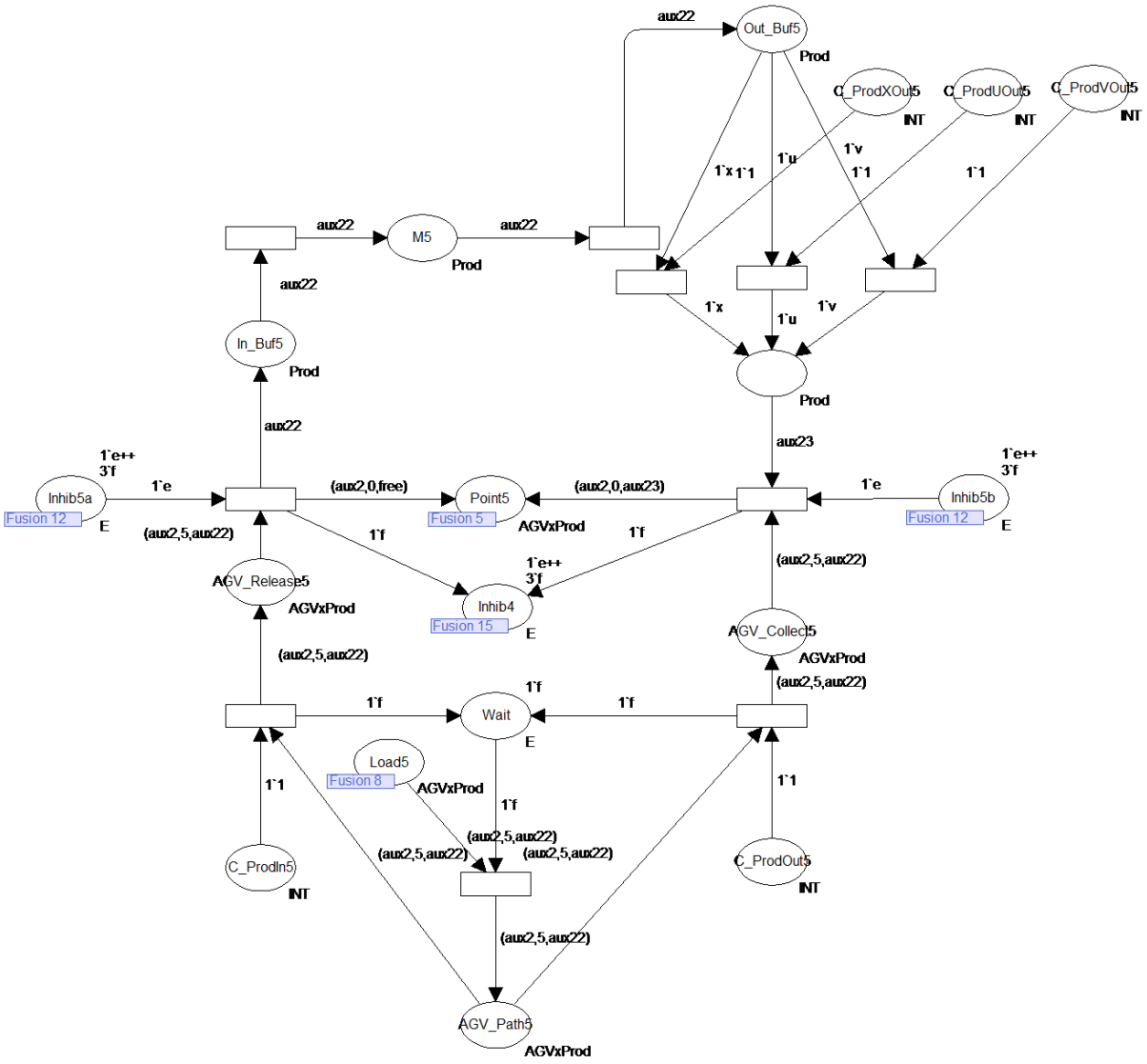


Figura 26 – Módulo da área de *buffers* da máquina *M5*.

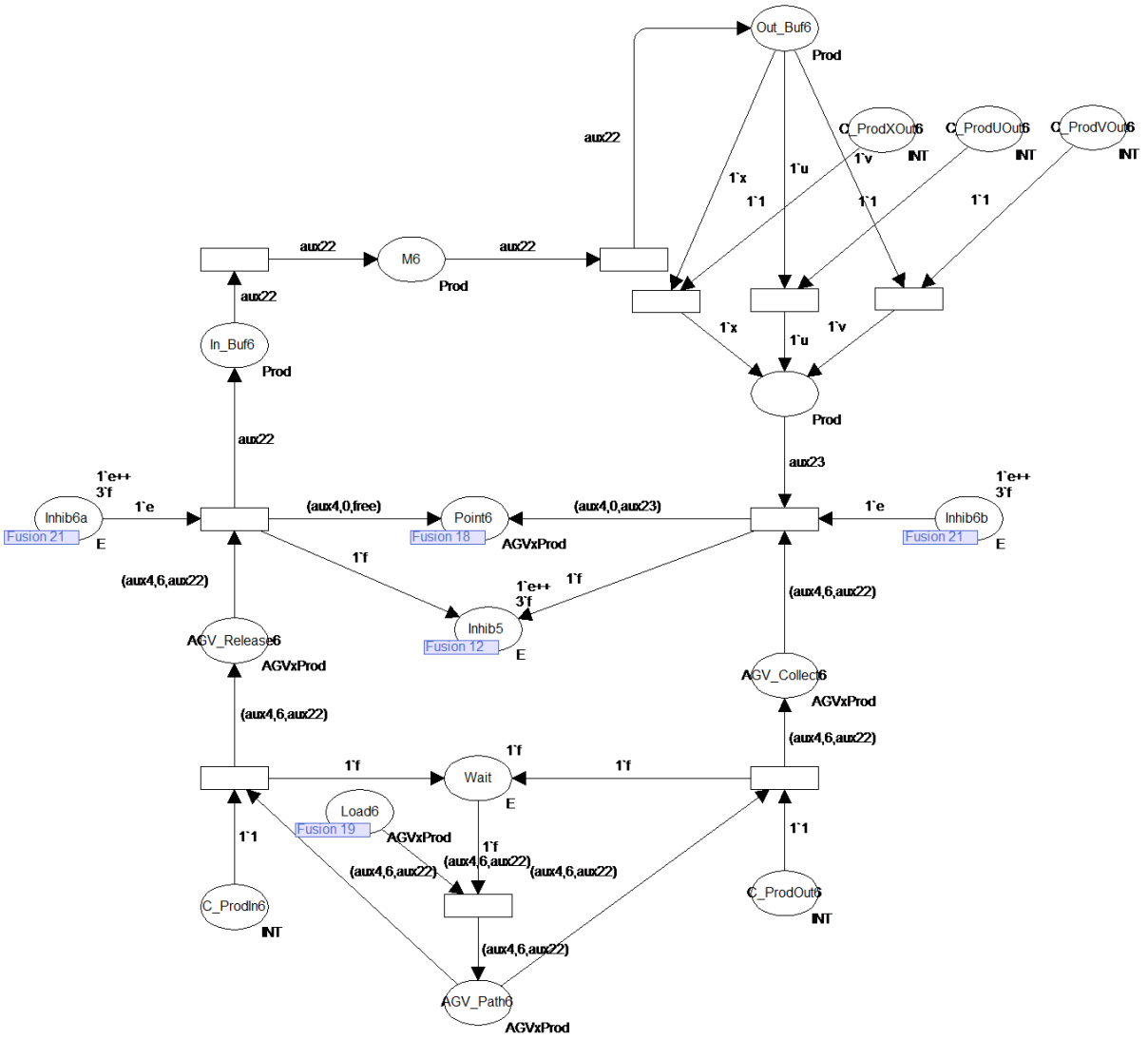


Figura 27 – Módulo da área de *buffers* da máquina *M6*.

5.2.3 Roteiros de Produção

O modelo de FMS em desta proposta possui três tipos de produto: *x*; *u*; *v*.

Cada um desses tipos de produto passa por três estágios de produção. Isto significa que cada um dos produtos será processado por uma máquina em cada estágio de produção. Quando o produto estiver pronto, este terá sido processado por três máquinas. Em cada estágio de produção um produto pode ser processado por uma ou mais máquinas. Cada conjunto de três máquinas, que processam o produto em cada

estágio de produção, forma a coleção roteiros de produção de um determinado tipo de produto.

Na Tabela 3 são apresentados os roteiros de produção de cada tipo produto.

Tabela 3 – Roteiros de produção.

Tipo do Produto	Roteiros
x	*M1, M3, M5* *M2, M3, M5*
u	*M1, M2, M5* *M1, M2, M6* *M1, M4, M5* *M1, M4, M6* *M3, M2, M5* *M3, M2, M6* *M3, M4, M5* *M3, M4, M6*
v	*M1, M4, M5* *M3, M4, M5*

A escolha do roteiro está relacionada ao comportamento global do sistema. O sistema de controle irá trabalhar no sentido de diminuir o *makespan* da produção e também de diminuir a ociosidade dos recursos do sistema.

Assim os roteiros serão escolhidos em tempo de execução conforme os AGVs atendem as ordens do sistema de controle. O roteiro de produção de um produto só será conhecido quando este produto estiver pronto, pois conforme o sistema evolui os AGVs recebem ordens do sistema de controle para que transportem os produtos de uma máquina para outra até que o produto esteja terminado e seja transportado para o armazém.

5.2.4 Resolução de conflitos

No modelo de FMS desta proposta existem pontos específicos nos quais se apresentam situações de conflito. Na Tabela 4 são listadas as possíveis situações de conflito do modelo.

Tabela 4 – Relação das possíveis situações de conflito do sistema, as possíveis tomadas de decisão para cada conflito e as varáveis que serão consideradas para a tomada de decisão.

Situação de conflito	Escolha	Variáveis Consideradas
AGV no Estacionamento	Quando liberar AGVs.	Quantidade de produtos nos *buffers* de saída.
AGV no nó, ocupado, e com destino	Enviar AGV para área de *buffers* da máquina do nó. Enviar AGV para próximo nó. Enviar AGV para armazém. Enviar AGV para Estacionamento.	Destino do AGV. Nó atual.
AGV no nó, livre, e com destino	Enviar AGV para área de *buffers* da máquina do nó. Enviar AGV para o próximo nó. Enviar AGV para o armazém. Enviar AGV para o estacionamento.	Destino do AGV. Nó atual.
AGV no nó, livre, e sem destino	Escolher *Buffer* de saída para atender, escolher produto no *buffer* de saída para coletar, escolher *buffer* de entrada para depositar o produto escolhido.	Quantidade de produtos nos *buffers* de saída. Tipos de produtos nos *buffers* de saída. Número de nós. Distância a ser percorrida. Quantidade de produtos nos *buffers* de entrada.
AGV na entrada da área de *buffers*	Enviar AGV para carga ou descarga.	Ocupação de carga do AGV.
AGV no *buffer* de saída	Escolher o produto que será carregado no AGV.	Tipo de produto definido na ordem do AGV.

Nas subseções seguintes são apresentadas as abordagens adotadas para resolução de cada conflito do sistema, e como o sistema de controle trata cada uma das variáveis para tomar uma decisão e solucionar a situação de conflito.

5.2.4.1 AGV no estacionamento

Quando há AGVs no estacionamento o sistema de controle verifica a quantidade de produtos nos *buffers* de saída de todas as máquinas e do armazém. Se houver produto em algum destes *buffers* de saída o sistema de controle ativa o lugar de controle *C_Release_AGV*. Ativar um lugar de controle significa colocar uma marca neste lugar de controle para que a transição de guarda seja disparada.

5.2.4.2 AGV em nó, ocupado e com destino

Quando um AGV alcança um nó, e este AGV possui um destino e também porta um determinado produto o sistema de controle verifica o nó atual e o destino do AGV.

Caso o nó atual e o destino do AGV sejam os mesmos significa que o AGV está à frente da sua máquina de destino. O sistema de controle então ativa o lugar de controle *C_Load(i)* em que *i* representa o nó *N(i)*, bem como o número da máquina. Quando o lugar *C_Load(i)* for ativado a transição que leva o AGV para a área de *buffers* da máquina *M(i)* será disparada e o AGV então irá ocupar o lugar *Load(i)* que significa que este AGV está em frente à área de *buffers* da máquina. Na Figura 28 é apresentado um exemplo desta situação no módulo principal do modelo do FMS em rede de Petri.

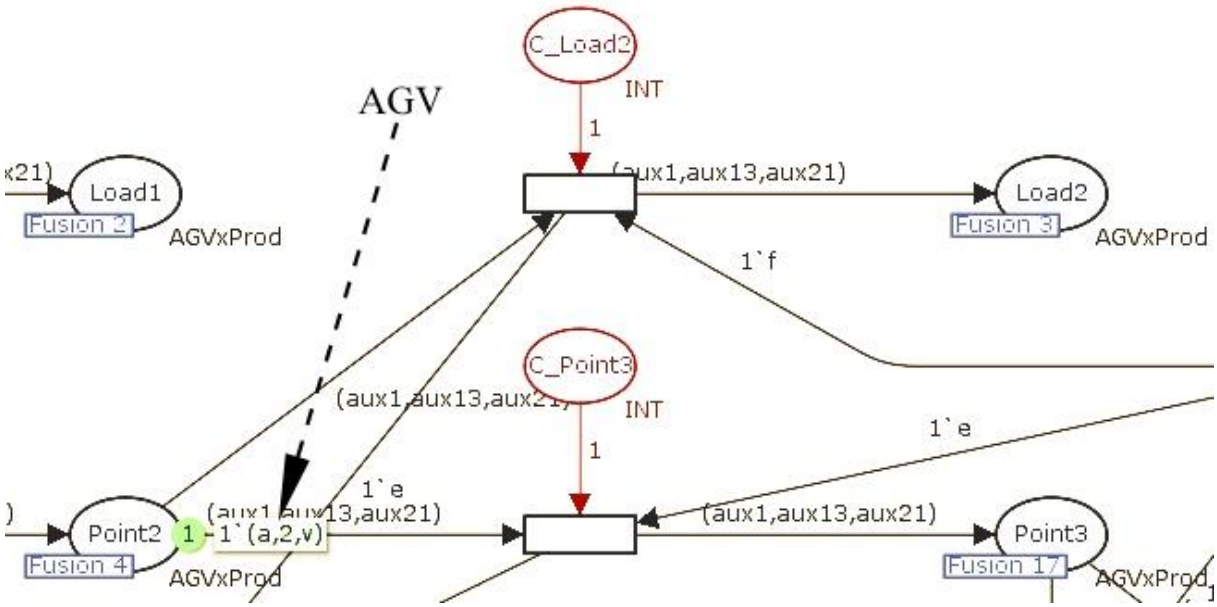


Figura 28 – AGV em nó, ocupado e com destino.

Como é mostrado na indicação de AGV na Figura 28, o AGV *a* está no nó *N2* representado pelo lugar *Point2* com um produto do tipo *v* em sua carga. O destino do AGV *a* é 2 como mostra a marca do lugar *Point2*. Isto significa que este AGV deverá se dirigir para a área de *buffers* da máquina *M2*. O lugar de controle *C_Load2* será ativado para que isso aconteça.

Caso o destino e o nó não sejam os mesmos este AGV deve seguir sua rota até encontrar o seu destino. Para isto acontecer o sistema de controle ativa o lugar de controle *C_Point(j)*, onde *j* é o número do próximo nó do caminho circular. Com o conflito resolvido o AGV segue para o próximo nó. Na Figura 29 é apresentado um exemplo em que ocorre esta situação.

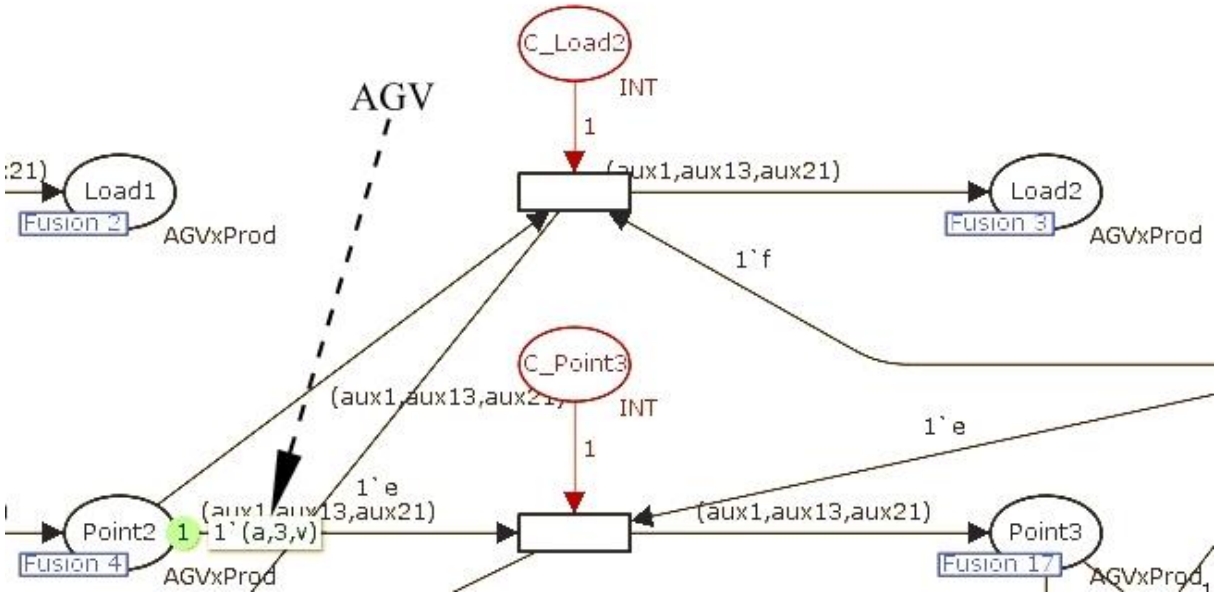


Figura 29 – AGV em nó com destino diferente do nó atual.

Como é mostrado na indicação de AGV na Figura 29, o AGV *a* está no nó *N2* representado pelo lugar *Point2* com um produto do tipo *v* em sua carga. O destino do AGV *a* é 3, como mostra a marca do lugar *Point2*. Isto significa que este AGV deverá seguir sua trajetória até se posicionar no nó *N3*. O lugar de controle *C_Point3* será ativado para que isso aconteça.

Há dois casos especiais em que o nó atual não é igual ao destino, mas o nó atual é o destino final do AGV. O primeiro deles é quando o destino do AGV é 100 e o nó atual é o nó *N1*. Neste caso, o sistema de controle ativa o lugar de controle *C_Load_Storage* para que o AGV se dirija ao armazém. Isto acontece devido ao fato de que, conforme o esquema físico do FMS, o armazém compartilha o caminho da entrada da área de *buffers* da máquina *M1*. O segundo caso especial acontece quando o destino do AGV é 101 e o nó atual é o nó *N5*. Neste caso o sistema ativa o lugar de controle *C_Park* para que o AGV se dirija ao estacionamento. A área de *buffers* da máquina *M5* compartilha o mesmo caminho que dá acesso ao estacionamento.

5.2.4.3 AGV em nó, livre e com destino

O sistema de controle ativa o lugar de controle *C_Point(j)* em que *j* é o número do próximo nó do caminho circular. O sistema tomará esta decisão em todos os nós que este AGV passar até que o nó seja igual ao destino do AGV.

5.2.4.4 AGV em nó, livre e sem destino

Neste conflito o sistema de controle verifica a quantidade de produtos em todos os *buffers* de saída do sistema. Para cada *buffer* de saída que tem quantidade produtos maior que zero, o sistema de controle verifica quais tipos de produtos há em neste *buffer* de saída. O sistema de controle então monta uma lista de *buffers* de saída que possui produtos e uma lista de tipos de produtos existentes em cada *buffer* de saída. Dado que cada tipo de produto em um determinado *buffer* de saída foi processado pela máquina da área de *buffers* a qual o *buffer* de saída faz parte, o sistema de controle conhece a máquina que processou o produto. Cruzando esta informação com a tabela de roteiros de produção o sistema de controle identifica qual o estágio de produção atual de cada produto nos *buffers* de saída e, portanto, qual será o próximo estágio de produção daquele produto. Também pela tabela de roteiros, o sistema de controle identifica quais máquinas podem processar o produto no próximo estágio de produção. Conhecendo as máquinas, o sistema identifica os *buffers* de entrada que podem receber cada produto.

Neste momento o sistema monta uma tabela com todas as possibilidades de escolhas de pares de *buffers* de saída e *buffers* de entrada e tipos de produto que pode ser transportado entre cada par de *buffer*.

A partir desta tabela de pares de *buffers*, o sistema de controle calcula a distância entre cada par de *buffer* de saída e *buffer* de entrada. A esta distância soma-se a distância entre a posição atual do AGV e o *buffer* de saída. A distância total é adicionada na tabela de pares de *buffers* de saída e entrada para cada par.

De modo análogo, o sistema calcula o número de nós entre os *buffers* e soma este valor com o número de nós entre o nó atual do AGV até o *buffer* de saída.

O sistema de controle registra a informação do número total de nós na tabela.

Feito isso, o sistema possui os valores de todas as variáveis que serão consideradas para a resolução deste conflito. Cada registro na tabela representa uma possibilidade de ordem para o AGV.

O sistema de controle emprega prioridade para cada uma destas ordens.

Para calcular a prioridade de cada ordem foi desenvolvido um sistema *Fuzzy*.

O sistema *Fuzzy* desenvolvido tem quatro variáveis de entrada a saber: quantidade de peças no *buffer* de saída; número de nós; distância; quantidade de peças no *buffer* de entrada.

A variável de quantidade de produtos no *buffer* de entrada tem três conjuntos *Fuzzy*: vazio, meia capacidade, e cheio, e seu conjunto universo pode variar de 0 a 100, considerando que o *buffer* vazio tem 0 produto e um *buffer* com lotação máxima tem 100 produtos.

A variável de número de nós tem três conjuntos *Fuzzy*: baixo, médio e alto. O valor escalar máximo para a variável número de nós é 6 conforme o arranjo físico do FMS modelado.

A variável de distância tem três conjuntos *Fuzzy*: próximo, meia distância, e longe. A maior distância que um AGV pode percorrer no sistema é de 100 metros.

A variável de quantidade de produtos em *buffer* de entrada tem três conjuntos *Fuzzy*: vazio, meia capacidade, e cheio. A variação escalar desta variável se dá entre 0 e 100.

A variável de saída representa a prioridade que o sistema de controle deve considerar para uma determinada ordem na tabela de ordens. Os conjuntos *Fuzzy* desta variável são: baixa, média, e alta. Seu valor escalar varia de 0 a 10.

A base de regras *Fuzzy* foi construída considerando as quatro variáveis de entrada e a variável de saída. No total, 81 regras formam a base de regras. Na Tabela 5 é apresentada uma amostra da base de regras.

Tabela 5 – Amostra da base de regras do sistema *Fuzzy*.

Nome	Regra
R1	SE *buffer de saída* é vazio E *número de nós* é baixo E *distância* é próximo e *buffer de saída* é vazio ENTÃO *prioridade* é média
R2	SE *buffer de saída* é meia capacidade E *número de nós* é baixo E *distância* é próximo e *buffer de saída* é vazio ENTÃO *prioridade* é alta
R3	SE *buffer de saída* é vazio E *número de nós* é baixo E *distância* é próximo e *buffer de saída* é vazio ENTÃO *prioridade* é alta
R4	SE *buffer de saída* é vazio E *número de nós* é médio E *distância* é próximo e *buffer de saída* é vazio ENTÃO *prioridade* é média
R5	SE *buffer de saída* é vazio E *número de nós* é alto E *distância* é próximo e *buffer de saída* é vazio ENTÃO *prioridade* é média
R6	SE *buffer de saída* é vazio E *número de nós* é baixo E *distância* é meia distância e *buffer de saída* é vazio ENTÃO *prioridade* média
...	...
R75	SE *buffer de saída* é cheio E *número de nós* é alto E *distância* é longe e *buffer de saída* é cheio ENTÃO *prioridade* baixa
R76	SE *buffer de saída* é cheio E *número de nós* é alto E *distância* é longe e *buffer de saída* é meia capacidade ENTÃO *prioridade* baixa
R77	SE *buffer de saída* é cheio E *número de nós* é alto E *distância* é longe e *buffer de saída* é vazio ENTÃO *prioridade* média
R78	SE *buffer de saída* é cheio E *número de nós* é alto E *distância* é meia distância e *buffer de saída* é vazio ENTÃO *prioridade* média
R79	SE *buffer de saída* é cheio E *número de nós* é alto E *distância* é próximo e *buffer de saída* é vazio ENTÃO *prioridade* média
R80	SE *buffer de saída* é cheio E *número de nós* é médio E *distância* é próximo e *buffer de saída* é vazio ENTÃO *prioridade* alta

R81	SE *buffer de saída* é cheio E *número de nós* é baixo E *distância* é próximo e *buffer de saída* é vazio ENTÃO *prioridade* alta

As variáveis são ponderadas pelo sistema *Fuzzy* de a fim de alcançar os seguintes objetivos:

- Reduzir o número de máquinas que param de trabalhar por não ter produtos no *buffer* de entrada;
- Reduzir o número de máquinas que param de trabalhar por não ter espaço no *buffer* de saída para depositar os produtos processados;
- Reduzir o a distâncias percorridas pelos AGVs;
- Reduzir o número de nós percorridos pelos AGVs.

Após a execução do sistema *Fuzzy* o sistema de controle registra na tabela de ordens a prioridade para cada ordem.

O sistema de controle em seguida escolhe a ordem com maior prioridade e descarta a tabela de ordens. A ordem escolhida usada em outra tabela *AGVxOrdem* que registra o *buffer* de saída para qual o AGV deve se dirigir, o produto que o AGV irá coletar naquele *buffer* de saída, e para qual *buffer* de entrada o AGV deverá se dirigir após carregar o produto.

A informação de destino do AGV é imediatamente alterada para o novo destino na marca do AGV ainda enquanto está no nó atual.

O produto que está na ordem *AGVxOrdem* é marcado para não entrar na ordem de outro AGV e, para que apenas aquele o AGV que possui a ordem de transporta-lo possa fazê-lo.

5.2.4.5 AGV na entrada da área de *buffers*

Quando um AGV se posiciona na área no lugar *AGV_Path(i)* o sistema de controle verifica se o AGV possui um produto em carga. Em caso positivo o sistema de controle ativa o lugar de controle *C_ProdIn(i)* dando condição para o AGV seguir para o *buffer* de entrada da máquina *M(i)* depositar o produto que está em sua carga para que este seja processado pela máquina.

Caso contrário, o sistema de controle ativa o lugar de controle *C_ProdOut(i)*. Assim o AGV segue para o *buffer* de saída da máquina para carregar-se com algum produto e transporta-lo.

5.2.4.6 AGV no *buffer* de saída

Nesta situação de conflito o sistema de controle consulta a tabela *AGVxOrdem* na qual estão definidas todas as ordens para todos os AGVs. O sistema de controle identifica o AGV pelo nome e busca na tabela a sua última ordem.

Na ordem está registrado o produto que aquele AGV deve transportar a partir do *buffer* de saída em que está.

O sistema de controle identifica o produto e ativa o lugar de controle que permite que este produto saia do *buffer* de saída e seja carregado pelo AGV.

O destino do AGV é alterado para o *buffer* de entrada da ordem, para o qual o produto deverá ser transportado. O AGV segue para o caminho de transporte.

5.2.5 Desenvolvimento do sistema de controle

Com o modelo finalizado e todas as estratégias de resolução de conflitos definidas, deu-se inicio ao desenvolvimento do sistema de controle.

O desenvolvimento do sistema de controle foi dividido em três etapas, a saber:

- Desenvolvimento da interface de comunicação com o modelo;
- Desenvolvimento do sistema *Fuzzy*;
- Desenvolvimento do software controlador.

A seguir são detalhadas cada uma das etapas.

5.2.5.1 Desenvolvimento da interface de comunicação com o modelo

O modelo, que foi construído usando-se a ferramenta CPN Tools, não tem seus conflitos resolvidos no próprio modelo. Como o sistema de controle é externo, foi necessário construir uma interface de comunicação entre o sistema de controle desenvolvido e a ferramenta CPN Tools.

O projeto CPN Tools disponibiliza a biblioteca de software ACCESS/CPN. Esta biblioteca de software possui uma coleção de classes escritas em Java que permite a interação de outros sistemas com o CPN Tools. A ACCESS/CPN é uma biblioteca de código aberto. Por meio do ACCESS/CPN é possível ler o estado de todos os elementos, de uma rede de Petri escrita no CPN Tools.

Com base no ACCESS/CPN foi feito um Web Service Java chamado *PlaceWatcher* que disponibiliza três serviços, a saber:

- *getMark*
    - Serviço que recebe como entrada o nome de um lugar existente na rede de Petri do modelo e retorna as marcas do lugar no momento da chamada;
- *setMark*
    - A partir de duas variáveis de entrada que são: nome de um lugar existente na rede de Petri do modelo e a marca que o lugar deverá receber. O serviço altera a marca do lugar que passa a conter as marcas passadas como parâmetro;

- *execute*
    - Este serviço não recebe nenhum parâmetro de entrada. Sua função é executar um passo de simulação no simulador do CPN Tools. Quando este serviço é chamado todas as transições habilitadas do modelo são disparadas e a rede assume o estado pós execução das transições habilitadas.

O serviço foi escrito em Java usando-se da ferramenta Eclipse e foi executado em um servidor Apache Tomcat usando a engine de *Web Services* Axis2.

5.2.5.2 Desenvolvimento do sistema *Fuzzy*

O sistema *Fuzzy* foi desenvolvido por meio da ferramenta *Fuzzy* do *software* Matlab.

Primeiramente foi criado um projeto *Fuzzy* e em seguida foram definidas as variáveis de entrada e saída para o sistema *Fuzzy*. As definições das variáveis são apresentadas a seguir.

5.2.5.2.1 Variável quantidade de produtos em buffer de saída

- Nome: *quantidade-de-produtos-em-buffer-saída*;
- Tipo: entrada;
- Conjunto universo: {0, 1, 2, ..., 100};
- Conjuntos *Fuzzy*: vazio, meia capacidade, cheio;
- Tipo das funções dos conjuntos *Fuzzy*: triangular.

5.2.5.2.2 Variável número de nós

- Nome: *numero-de-nos*;

- Tipo: entrada;
- Conjunto universo: {1, 2, 3, 4, 5, 6};
- Conjuntos *Fuzzy*: baixo, médio, alto;
- Tipo das funções dos conjuntos *Fuzzy*: triangular.

5.2.5.2.3 Variável distância

- Nome: *distancia*;
- Tipo: entrada;
- Conjunto universo: [0 100];
- Conjuntos *Fuzzy*: próximo, meia distância, longe;
- Tipo das funções dos conjuntos *Fuzzy*: triangular;

5.2.5.2.4 Variável quantidade de produtos em buffer de saída

- Nome: *quantidade-de-produtos-buffer-entrada*;
- Tipo: entrada;
- Conjunto universo: [0 100];
- Tipo das funções dos conjuntos *Fuzzy*: triangular;

5.2.5.2.5 Variável prioridade

- Nome: *prioridade*;
- Tipo: saída;
- Conjunto universo: [0 10];
- Tipo das funções dos conjuntos *Fuzzy*: triangular;

Definidas as variáveis de entrada e saída foram definidas as regras que formam a base de regras do sistema *Fuzzy*. Na Tabela 5 é apresentada uma amostra da base de regras.

Em seguida foi construída uma função Matlab chamada *PrioridadeDeOrdem* que recebe as quatro variáveis de entrada do sistema *Fuzzy* como parâmetro e retorna a saída do sistema *Fuzzy*. Esta função aciona o sistema *Fuzzy* e retorna um valor entre 0 e 10 que representa a prioridade da ordem.

Essa função foi então compilada pela ferramenta *deploy* do Matlab. A partir disto foi gerado um componente de *software* do tipo *.NET Framework Assembly*.

5.2.5.3 Desenvolvimento do sistema controlador

O sistema controlador foi escrito em linguagem C# por meio da ferramenta Visual Studio. O paradigma de programação usado foi o de programação orientada a objetos.

Foi criada uma solução com 7 projetos: *AGVDispatch*, *AGVLoadControl*, *ControlInterface*, *Decision*, *ProductControl*, e *TearFMS*.

Cada um destes projetos deu origem a um componente de software em formato *Assembly*. Cada *assembly* tem um papel definido.

A função do *assembly AGVDispatch* é dar ordens aos AGVs, isto é, definir qual *buffer* de saída o AGV irá atender, qual produto será coletado neste *buffer* de saída, e para qual *buffer* de entrada o AGV deve se dirigir para descarregar este produto. Este *assembly* também é responsável por controlar a liberação dos AGVs que estão no estacionamento e são escalados para atender algum *buffer*.

O papel do *assembly AGVLoadControl* é resolver a situação de conflito em que um AGV acessa a área de *buffers* de uma máquina. Ele decide se o AGV deve carregar ou descarregar produto. Tomada a decisão este componente realiza o controle ativando o lugar adequado na rede de Petri do modelo.

O *assembly ControlInterface* é responsável pelas interações com o CPN Tools. Este *assembly* é cliente dos serviços do Web Service *PlaceWatcher* e intermedia todas as ações de controle realizadas pelos outros módulos *assembly*.

O papel do *assembly Decision* é fazer a interface do sistema controlador com o sistema *Fuzzy*. Todas as operações de resolução de conflito, nas quais as variáveis do FMS são consideras, são processadas por este *assembly*.

O papel do *assembly ProductControl* é controlar a liberação de produtos que serão carregados pelos AGVs de acordo com a ordem para cada AGV que esteja em frente a um *buffer* de saída. Quando um AGV estiver em um *buffer* de saída este *assembly* analisa a ordem do AGV verifica qual é o produto e ativa o lugar de controle que libera um produto que tenha o mesmo tipo que aquele produto da ordem do AGV.

Todos os módulos *assembly* possuem o método *Controller*, com exceção dos *assembly TearFMS* e *Decision*. O sistema controlador executa o método *Controller* de cada *assembly* para realizar o controle da rede de Petri do modelo em CPN Tools. Os *assembly TearFMS* e *Decision*, não possuem o método *Controller* por serem módulos *assembly* que prestam serviços para os outros módulos.

O papel do *assembly TearFMS* é representar todos os elementos físicos do modelo como nós, AGVs, pontos de controle, entradas de *buffers*, *buffers*, estacionamento e máquinas. Este *assembly* abstrai de forma orientada a objetos todos os recursos do FMS para que a programação do controle destes seja dada de uma forma que tenha alta legibilidade. Ele também permite que qualquer alteração na configuração do modelo seja feita de maneira transparente para o sistema de controle em sua totalidade. Isto diminui os esforços para a configuração de simulações, pois reduz os esforços de programação exigidos nos outros módulos *assembly* do sistema controlador.

5.3 Simulação e validação do modelo

A validação do modelo foi realizada por meio da execução de planos de produção no modelo. Todos os planos de produção foram executados e terminados sem que o sistema entrasse em estado de *deadlock*. A Tabela 6 apresenta os planos de produção de acordo com os roteiros de produção definidos na Tabela 3.

Tabela 6 – Planos de Produção.

Plano de Produção	Quantidades de produtos X	Quantidade de produtos U	Quantidade de produtos V
1	30	30	30
2	50	50	0
3	50	0	50
4	0	50	50
5	10	10	10

O sistema de controle resolveu todos os conflitos previstos no modelo. Todos os planos de produção foram completados com sucesso.

O tempo de resposta do sistema controlador e do sistema *Fuzzy* é relativo ao número de possibilidades de escolha a serem escolhidas pelo sistema de controle em uma determinada situação de conflito, e ao sistema computacional no qual o sistema de controle está sendo executado.

O modelo e o sistema de controle foram executados em um sistema computacional com as seguintes configurações:

- Processador Intel® Core™2 Duo com 2.20GHz;
- Memória RAM de 4.00 GB.

O número de possibilidades de resolução para uma situação de conflito pode ser calculado pela fórmula:

$$\sum_{i}^{i=j} x(i) + u(i) + v(i)$$

Onde:

- *i* é o *buffer* de saída da máquina *M(i);*
- *j* é o número de *buffers* de saída que possuem produtos;
- *x(i)* é o número de alternativas de roteiros para o produto do tipo x a partir do *buffer* de saída da máquina *M(i);*
- *u(i)* é o número de alternativas de roteiros para o produto do tipo *u* a partir do *buffer* de saída da máquina *M(i);*
- *v(i)* é o número de alternativas de roteiros para o produto do tipo *v* a partir do *buffer* de saída da máquina *M(i);*

Na Tabela 7 é apresentada uma amostra de tempos de resposta obtidos durante a execução do Plano de produção 1.

Tabela 7 – Tempos de resposta do sistema de controle para conflitos do Plano de Produção 1.

***Buffers* de saída com produto**	**Tipos de Produtos nos *Buffers* de saída**	**Possibilidades de escolha**	**Tempo de resposta em segundos**
1	*x, u, v*	6	0.39
2	*x, u, v*	10	0.57
3	*x, u, v*	11	0.57
4	*x, u, v*	15	0.62
5	*x, u, v*	15	0.62
5	*x, u, v*	57	1.32

4	*x, u, v*	36	1.19

Os tempos de resposta para a resolução dos conflitos se apresentaram de maneira satisfatória. Com os tempos apresentados o sistema de controle deste trabalho pode ser considerado como um sistema de tempo real, sendo adequado para um FMS.

Capítulo 6

CONCLUSÃO

O desafio deste trabalho foi evoluir nas questões de modelagem e controle de FMSs.

Várias abordagens para estas questões foram estudadas a fim de conhecer o que tinha sido feito a respeito do projeto e desenvolvimento de sistemas do tipo FMS, para que a proposta feita neste trabalho fosse elaborada de forma coerente com as pesquisas até então realizadas. Deste modo foi possível apresentar um avanço nos estudos sobre modelagem e controle de FMSs.

Entendeu-se que uma maior representatividade dos sistemas FMS poderia ser alcançada em relação ao trabalho de (ARAÚJO, 2006).

Informações do FMS como número de produtos em *buffer*, número de nós e distância do transporte, são importantes para a medição de taxas de desempenho do sistema e para a realização de uma sintonia do próprio sistema para que este alcance os seus objetivos de produção. O trabalho de Araújo propôs um método de modelagem de sistemas FMS que garante modelos livres de *deadlocks* e que permite a simulação de planos de produção distintos. Para a realização do controle na estratégia de (ARAÚJO, 2006) não são consideradas informações do FMS em tempo de execução. Esta foi uma questão na qual o trabalho presente evoluiu.

Foi proposta no trabalho presente a inclusão destas informações no tratamento realizado pelo controle do sistema.

Para tanto, foram realizadas alterações no método de modelagem de (ARAÚJO, 2006), e um sistema de controle foi desenvolvido.

As alterações realizadas no método de modelagem foram no sentido de colocar o modelo em estado de *deadlock* sempre que uma situação de conflito fosse alcançada. Com o modelo em rede de Petri sem condições de continuar a execução do sistema, o sistema de controle toma ciência da situação de conflito e inicia um processo de tomada de decisão.

Este processo de decisão leva em consideração as seguintes variáveis do sistema: quantidade de produtos em buffer, distância a ser percorrida pelos AGVs durante o transporte de produtos, o número de nós ou pontos de entroncamento que são ocupados pelos AGVs durante o transporte.

Para ponderar estas variáveis foi desenvolvido um sistema *Fuzzy*. O sistema *Fuzzy* retorna em tempo real todas as informações que o sistema de controle necessita para realizar a tomada de decisão para solucionar o conflito em questão.

Tomada a decisão, o sistema de controle, por meio de uma interface de comunicação, resgata o modelo do estado de *deadlock* o que permite que o processo de produção continue de acordo com a tomada de decisão.

A interface de comunicação do software do sistema de controle com o modelo foi desenvolvida com base na abordagem de software como serviço. Esta abordagem permite que outros sistemas de outras plataformas possam se comunicar o modelo, haja vista que os serviços são acessados por meio de protocolos padrões da indústria, os quais são implementados pela maioria das plataformas. Isto estabelece uma margem facilitadora para que trabalhos futuros sejam realizados em prol da evolução dos estudos a respeito de controle de FMS, pois não impõe desafios relacionados a tecnologias de desenvolvimento de software, o que significa que quaisquer linguagens de programação, por exemplo, podem ser usadas para dar continuidade a este trabalho.

Teve-se como resultado um sistema de controle para o modelo de um FMS hipotético modelado em redes de Petri que realiza o controle deste modelo considerando informações do sistema FMS em tempo real, ponderando variáveis do próprio FMS.

O avanço em relação ao trabalho de (ARAÚJO, 2006) foi o aumento da representatividade do modelo ainda com a garantia de que o sistema seja livre de deadlocks; um método que facilita a alteração dos planos de produção e inclusão de novos elementos no modelo de FMS; um modelo que permite que um sistema de controle trabalhe em função de diminuir o *makespan* e a ociosidade dos recursos do modelo do FMS.

6.1 Trabalhos futuros

Muito ainda pode ser feito para avançar em relação a modelagem e controle de sistemas do tipo FMS. A própria indústria evolui no sentido de demandar cada vez mais tecnologia para solucionar os problemas que surgem com as novas necessidades apresentadas pelos sistemas de produção.

Com base neste trabalho, diversas inclusões e extensões podem ser desenvolvidas. Algumas destas são listadas a seguir:

- Inclusão de outras informações ou variáveis do sistema, que podem ir além do chão de fábrica, na resolução de conflitos pelo sistema de controle.

- A extensão do sistema de controle para uma plataforma de software capaz de fazer medição de taxas de desempenho como *makespan* e ociosidade de recursos do sistema.

- Aprendizagem de máquina e otimização para a sintonia do sistema *Fuzzy*.

- Considerar o sistema de controle para realizar ensaios com outros planos de produção em outros modelos.

REFERÊNCIAS BIBLIOGRÁFICAS

AGUIRRE, L. A. **Enciclopédia de Automática Controle e Automação**. São Paulo: Editora Blucker, v. 1, 2007.

ARAÚJO, G. R. **Um Método para Modelagem de Controle de AMS usando Redes de Petri Virtuais**. 2006. 91 p. Dissertação (Mestrado em Computação) - Pograma de Pós-Graduação em Ciência da Computação, UFSCar. São Carlos. 2011.

BANNAT, A. et al. Artificial Cognition in Production Systems. **Automation Science and Engineering**, v. 8, p. 148-174, Janeiro 2011. ISSN 10.1109/TASE.2010.2053534.

BILGE, Ü.; FıRAT, M.; ALBEY, E. Aparametric fuzzy logic approach to dynamic part routing under full routing flexibility. **Computers & Industrial Engineering**, v. 55, p. 15–33, August 2008. ISSN 0360-8352.

BUYURGAN, N.; SAYGIN, C. An integrated control framework for flexible manufacturing systems. **International Journal of Advanced Manufacturing Technology**, v. 27, 2006. ISSN 1248–1259.

CAPRIHAN, R.; KUMAR, R.; STECKE, K. E. A fuzzy dispatching strategy for due-date scheduling of FMSs with information delays. **International Journal of Flexible Manufacturing Systems**, v. 18, p. 29-53, March 2006. ISSN 0920-6299.

CHAN, F. T. S.; CHAN, H. K.; KAZEROONI, A. Real time fuzzy scheduling rules in FMS. **Journal of Intelligent Manufacturing**, v. 14, n. 3, p. 341-350, June 2003. ISSN 0956-5515.

CHANG, A. Y. On the measurement of routing flexibility: A multiple attribute approach. **International Journal of Production Economics**, 2007. 122–136.

CHENGEN, W.; JIANYING, Z.; ZHONGXIN, W. An expert system for FMS control. **Intelligent Systems Engineering**, v. 2, n. 4, p. 223-230, Winter 1993.

CHOI, K.-H.; LEE, S.-H. Hybrid shop floor control system for Computer Integrated Manufacturing (CIM). **Journal of Mechanical Science and Technology**, 01 May 2001. 544-544.

DELGADILLO, G. M.; LLANO, S. P. **Scheduling Application using Petri Nets:** A Case Study: Intergráficas S.A. International Conference on Production Research. [S.l.]: [s.n.]. 2007.

GOLMAKANI, H. R.; MILLS, J. K.; BENHABIB, B. **Deadlock-free optimal routing in flexible manufacturing cells via supervisory control theory**. IEEE International Conference on Systems, Man and Cybernetics. [S.l.]: [s.n.]. 2003. p. 3390-3395.

GOMES, L. **On conflict resolution in Petri nets models through model structuring and composition**. Industrial Informatics, 2005. INDIN '05. 2005 3rd IEEE International Conference on. [S.l.]: IEEE. 2005. p. 489 - 494.

GROOVER, M. P. **Automation, production systems, and computer-integrated manufacturing**. [S.l.]: Prentice Hall, 2007.

KUO, C.-H.; HUANG, H.-P. Failure modeling and process monitoring for flexible manufacturing systems using colored timed Petri nets. **Robotics and Automation, IEEE Transactions on**, June 2000. 301 - 312.

LEDUC, R. J.; LAWFORD, M.; DAI, P. Hierarchical interface-based supervisory control of a flexible manufacturing system. **Control Systems Technology**, v. 14, n. 4, p. 654- 668, July 2006. ISSN 10.1109/TCST.2006.876635.

LEE, D. Y.; DICESARE, F. Scheduling flexible manufacturing systems using Petri nets and heuristic search. **Robotics and Automation, IEEE Transactions on**, April 1994. 123 - 132.

LEE, K. K. Fuzzy rule generation for adaptive scheduling in a dynamic manufacturing environment. **Applied Soft Computing**, September 2008. 1295-1304.

LEITAO, P.; RESTIVO, F. J. Implementation of a Holonic Control System in a Flexible Manufacturing System. **Systems, Man, and Cybernetics, Part C: Applications and Reviews**, v. 38, n. 5, p. 699-709, September 2008. ISSN 10.1109/TSMCC.2008.923881.

LEME, R. A. S. **Controles na producao**. 2. ed. São Paulo: LIVRARIA PIONEIRA, 1974.

MORANDIN, O. . J.; KATO, E. R. R. **Virtual Petri nets as a modular modeling method for planning and control tasks of FMS**. Systems, Man and Cybernetics, 2003. IEEE International Conference on. [S.l.]: IEEE. 2003. p. 1521 - 1527 vol.2.

MORANDIN, O.; ARAUJO, R. G.; KATO, E. R. R. **A Modeling Strategy of Shop-Floor Interlocking and Controlling using Virtual Petri Nets and Production Simulation**. Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on. [S.l.]: IEEE. 2006. p. 4084.

NOUREDDINE, M.; MARTINEAU, P. Modelling and Analysis Flexible Manufacturing Systems. **Information Technology Journal**, 2005. 233-238.

ODREY, N. C.; MEJÍA, G. **An augmented Petri Net approach for error recovery in manufacturing systems control**. Robotics and Computer-Integrated Manufacturing. [S.l.]: [s.n.]. 2005. p. 346–354.

OGATA, K. **Engenharia de controle moderno**. 2. ed. Rio de Janeiro: Prentice-Hall do Brasil, 1982.

OUELHADJ, D.; PETROVIC, S. A survey of dynamic scheduling in manufacturing systems. **Journal of Scheduling**, 2009. 417–431.

SGAVIOLI, M. **Modelagem de sistemas de manufatura usando Redes de Petri Coloridas Fuzzy focando a solução de conflitos**. 2010. 96 p. Dissertação (Mestrado em Computação) - Programa de Pós-Graduação em Ciência da Computação, UFSCar, São Carlos, 2010.

SHEN, W. Distributed manufacturing scheduling using intelligent agents. **Intelligent Systems, IEEE**, v. 17, n. 1, p. 88- 94, Jan/Feb 2002. ISSN 1541-1672.

SHNITS, B.; SINREICH, D. Controlling flexible manufacturing systems based on a dynamic selection of the appropriate operational criteria and scheduling policy. **International Journal of Flexible Manufacturing Systems**, v. 18, p. 1-27, 2006.

SINREICH, D.; SHNITS, B. A Robust FMS Control Architecture with an Embedded Adaptive Scheduling Mechanism. **Journal of Manufacturing Systems**, 2006. Vol. 25/No. 4.

SLACK, N.; CHAMBERS, S.; JOHNSTON, R. **Administracao da producao. [Operation management]**. São Paulo: Atlas, 1999.

SRINOI, P.; SHAYAN, E.; GHOTB, F. Scheduling of Flexible Manufacturing Systems Using Fuzzy Logic. **Profiles in Industrial Research: Knowledge and Innovation**, p. 94-102, 2002.

TSINARAKIS, G. J.; TSOURVELOUDIS, N. C.; VALAVANIS, K. P. **Petri Net Modeling of Routing and Operation Flexibility in Production Systems**. Mediterranean Conference on Control and Automation. [S.l.]: [s.n.]. 2005. p. 27-29.

TUNCEL, G. A. Heuristic Rule-Based Approach for Dynamic Scheduling of Flexible Manufacturing Systems Multiprocessor Scheduling: Theory and Applications. **Itech Education and Publishing**, Vienna, Austria, December 2007. 436.

TUNG, L.-F.; LIN, L.; NAGI, R. Multiple-Objective Scheduling for the Hierarchical Control of Flexible Manufacturing Systems. **International Journal of Flexible Manufacturing Systems** , 01 October 1999. 379-409.

TURGAY, S. Agent-based FMS control. **Robotics and Computer-Integrated Manufacturing**, v. 25, p. 470-480, April 2009. ISSN 0736-5845.

TÜYSÜZ, F.; KAHRAMAN, C. **Modeling a flexible manufacturing cell using stochastic Petri nets with fuzzy parameters**. Expert Systems with Applications. [S.l.]: [s.n.]. 2010. p. 3910–3920.

VIEIRA, G. E.; HERRMANN, J. W.; LIN, E. Rescheduling manufacturing systems: a framework of strategies, policies, and methods. **Journal Of Scheduling**, 2003. 35-58.

VOLLMANN, T. et al. **Sistemas de planejamento & controle da produção para o gerenciamento da cadeia de suprimentos. [Manufacturing planning and control systems for supply chain management]**. 5. ed. Porto Alegre: Bookman, 2008.

WU, N.; ZHOU, M.; HU, G. **On Petri Net Modeling of Automated Manufacturing Systems**. Networking, Sensing and Control, 2007 IEEE International Conference on. London: IEEE. 2007. p. 228 - 233.

XIANG, W.; LEE, H. P. Antcolonyintelligence in multi-agent dynamicmanufacturingscheduling. **Engineering Applications of Artificial Intelligence**, v. 21, p. 73–85, February 2008. ISSN 0952-1976.

ZHANG, H.; JIANG, Z.; GUO, C. **Simulation Based Real-time Scheduling Method for Dispatching and Rework Control of Semiconductor Manufacturing System**. IEEE International Conference on Systems, Man and Cybernetics. [S.l.]: IEEE. 2007. p. 2901-2905.

ZHOU, M.; VENKATESH, K. **Modeling, Simulation, and Control of Flexible Manufacturing Systems:** A Petri Net Approach. [S.l.]: World Scientific Publishing Company , 1999.

Apêndice A

REDES DE PETRI

As redes de Petri são uma ferramenta matemática de modelagem aplicável a muitos sistemas. São usadas para a descrição e estudos de sistemas assíncronos, concorrentes, distribuídos, paralelos, não determinísticos e/ou estocásticos.

Devido as suas representações gráficas, as redes de Petri podem ser usadas para apoiar a leitura visual de sistemas complexos.

Marcas podem ser usadas nas redes de Petri para simular as atividades dinâmicas e concorrentes dos sistemas.

Enquanto ferramenta matemática, as redes de Petri permitem a elaboração de equações para governar o comportamento dos sistemas.

As redes de Petri são usadas em simulações e estudos, bem como em sistemas de controle.

Para facilitar a modelagem dos variados tipos de sistemas que as redes de Petri podem representar, muitas extensões de redes de Petri foram propostas a fim de abordar com mais propriedade as características inerentes a cada tipo de sistemas. Dentre as extensões de rede de Petri propostas estão: redes de Petri Colorida; redes de Petri Virtuais; e, Redes de Petri Colorida *Fuzzy*, entre outras.

Basicamente as redes de Petri se agrupam em duas grandes classes: Ordinárias e Não-Ordinárias (de alto nível). As redes ordinárias se caracterizam pelo tipo de suas marcas que são do tipo inteiro e não negativo, enquanto que as de alto nível possuem tipos de marcas particulares. As redes ordinárias se subdividem em:

- Redes binárias: é a rede mais elementar dentre todas. Essa rede só permite até uma marca em cada lugar, e todos os arcos possuem valor unitário;
- Redes lugar-transição: é o tipo de rede que permite que mais de uma marca em um lugar os valores de seus arcos não são unitários.

As redes de alto nível são caracterizadas pelos tipos de suas marcas, que não são mais elementos do tipo inteiro positivo. Esse tipo de rede permite a individualização de uma marca (pertencente a um grupo) em um mesmo lugar. Essa individualização pode ser realizada por meio de vários artifícios, como por exemplo, cor da marca. Redes não-ordinárias aumentam o poder de representação de um modelo. Entretanto, elas permitem uma clareza e um maior nível de abstração ao modelo.

Definição 1: As redes de Petri do tipo lugar-transição são formalmente definidas pela tupla, $PN = (P, T, I, O)$ onde:

- $P = \{p_1, p_2, ..., p_m\}$ é um conjunto finito de lugares;
- $T = \{t_1, t_2, ..., t_3\}$ é um conjunto finito de transições;
- $I \subseteq (P \text{ x } T)$ é o conjunto dos arcos de entrada nas transições em T;
- $O \subseteq (T \text{ x } P)$ é um conjunto de arcos de entrada nos lugares em P.

Uma estrutura de rede de Petri $N = (P, T, I, O)$ sem nenhuma marcação inicial específica é dada por N. Uma rede de Petri com uma marcação inicial é denotada por (N, M_0), onde:

- M_0 é a marcação inicial da rede.

$$M : P \rightarrow \mathbb{N}$$

$M(p)$ é o número de marcas contidas no lugar p.

O Comportamento de muitos sistemas pode ser descrito em termos de estados e das mudanças destes estados. Para simular o comportamento dinâmico de um sistema, o estado ou marca em uma rede de Petri é alterado de acordo com a seguinte regra de transição:

1. Uma transição *t* está habilitada se cada lugar de entrada *p* em *t* estiver marcado com no mínimo uma marca.
2. Uma transição só pode ser disparada se esta estiver habilitada.
3. Quando uma transição *t* é disparada são removidas as marcas de cada lugar *p* de entrada em *t*, e adicionadas marcas em cada lugar de saída p da transição *t*.

Graficamente os lugares *p* são representados por elipses ou círculos enquanto as transições *t* são representadas por linhas ou retângulos. Já os arcos são setas direcionadas de lugares *p* para transições *t*, ou de transições *t* para lugares *p*. Na Figura 30 é apresentada uma rede de Petri.

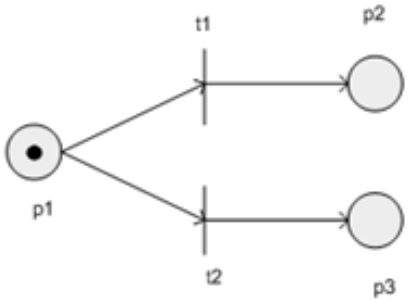


Figura 30 – Lugares transições e arcos de uma rede de Petri.

A rede de Petri *N* representada graficamente na Figura 30 é definida por $N = (\{p_1, p_2, p_3\}, \{t_1, t_2\}, \{(p_1 \times t_1), (p_1 \times t_2)\}, \{(t_1 \times p_2), (t_2 \times p_3)\}, \{1, 0, 0\})$. É uma rede de Petri com 3 lugares p_1, p_2, p_3, 2 transições t_1, t_2, e inicialmente marcada com 1 marca no lugar p_1.

Algumas extensões das redes de Petri permitem que as marcas sejam de tipos abstratos ou de alto nível. Isto significa que é possível representar o estado de objetos de um determinado sistema nas marcas da rede de Petri. Uma das extensões que tem essa característica é a rede de Petri Colorida.

Definição 2: As redes de Petri Coloridas são definidas formalmente pela tupla $N = (P, T, A, C, E, G)$ onde:

- P é um conjunto finito de lugares;
- T é um conjunto de transições, com $P \cap T = \varnothing$;
- $A \subseteq P \times T \cup T \times P$ é um conjunto de arcos. Para $t \in T$, é definido $A_t = ((P \times \{t\}) \cup (\{t\} \times P)) \cap A$;
- $C : P \rightarrow$ TIPO associa cor (tipos) aos lugares;
- $E : A \rightarrow$ EXP é o conjunto de inscrições de arcos que satisfazem tipo($E(a)$) = bag($C(p)$), para todo $p \in P$ e $a \in A_p$;
- $G : T \rightarrow$ {verdadeiro, falso} é uma função de guarda satisfazendo var($G(t)$) $\subseteq \cup_{a \in At}$ var($E(a)$);

Qualquer função $m : P \rightarrow$ EXP que associa cada p em P a uma expressão livre de variáveis e que satisfaz tipo(e) = bag($C(p)$) é chamada de marca. Uma rede de Petri Colorida (CPN) é um par (N, m) onde N é uma CPN e m é uma marcação inicial.

Na Figura 31 é apresentado um modelo em CPN.

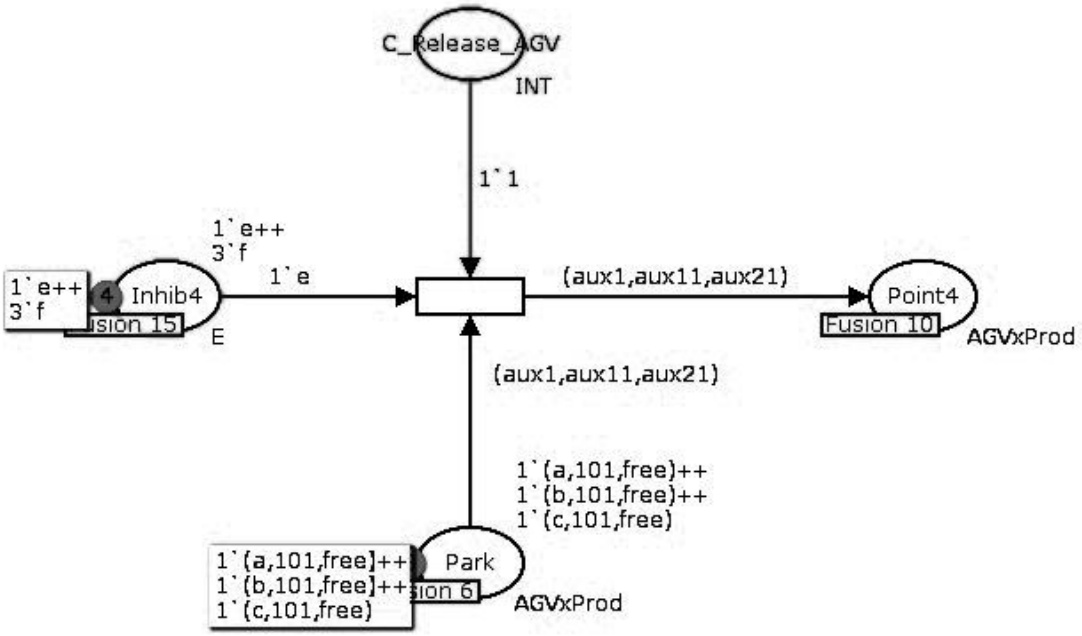


Figura 31 – Exemplo de CPN.

No exemplo mostrado na Figura 31 o lugar *Park* possui 3 marcas. As marcas que estão em *Park* são do tipo *AGVxProd* definido por *AGVxProd*=(*N*, *D*, *P*), onde:

- *N* é uma cadeia de caracteres que representa o nome do AGV.
- *D* é um número em ℕ que representa o destino do AGV.
- *P* em {*x*, *u*, *v*, *free*} representa o tipo de produto que o AGV está transportando.

Dessa forma é possível abstrair um sistema de produção em uma rede de Petri Colorida que tem um alto nível de representação. Seria possível usar uma rede de Petri ordinária para representar o mesmo sistema. Entretanto o nível de abstração seria mais baixo.