UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM

CIÊNCIA DA COMPUTAÇÃO

"Um Framework para Construção de Sistemas Inteligentes de Seqüenciamento da Produção"

ALUNO: Rafael Rabêlo Silva
ORIENTADOR: Prof. Dr. Orides Morandin Junior

São Carlos
Outubro/2009

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM

CIÊNCIA DA COMPUTAÇÃO

"Um Framework para Construção de Sistemas Inteligentes de Seqüenciamento da Produção"

RAFAEL RABÊLO SILVA

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.

ORIENTADOR: **Prof. Dr. Orides Morandin Junior**

São Carlos
Outubro/2009

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

S586fc

Silva, Rafael Rabêlo.
Um framework para construção de sistemas inteligentes de seqüenciamento da produção / Rafael Rabêlo Silva. -- São Carlos : UFSCar, 2011.
74 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2009.

1. Ciência da computação. 2. Sequenciamento da produção. 3. Simulação de sistemas industriais. 4. Sistemas inteligentes. I. Título.

CDD: 004 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Um Framework para Construção de Sistemas Inteligentes de Seqüenciamento da Produção"

RAFAEL RABÊLO SILVA

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 Júnior
(Orientador - DC/UFSCar)

Prof. Dr. Edilson Reis Rodrigues Kato
(DC/UFSCar)

Profa. Dra. Jandira Guenka Palma
(UEL)

São Carlos
Novembro/2009

Dedicatória

A minha noiva, Cleonice

A minha mãe, Vágna

e A meu pai, David.

Agradecimentos

À Cleonice por toda a paciência e apoio.

A meus pais, David e Vágna, por sempre acreditarem em mim.

Ao meu orientador, Prof. Dr. Orides Morandin Jr, pela grande colaboração para meu crescimento acadêmico, por toda a amizade, honestidade, companheirismo e preocupação. Jamais medir esforços em me orientar na superação de quaisquer obstáculos.

Aos meus colegas que dividem ou dividiram moradia comigo, Abimael "Abimas", Afonso, Gabriel "Gagá", Rodrigo Murta, por toda a força, os conselhos e o bom convívio.

Aos meus amigos Hiromiti e Carol por toda a ajuda no transporte, aconchego.

Ao meu amigo André Di Thommazo, pela motivação e sábios conselhos.

À minha irmã, Patrícia, pela paciência em momentos difíceis.

Aos colegas da Legião de Maria, por todas as orações e apoio espiritual.

Aos amigos Raphael "P3" e Fabiana Assao, por me incentivarem a ingressar no programa.

Aos amigos Marcos "Soulboy", Bruno "BTG" e Murilo por todo apoio e companheirismo nos estudos das disciplinas.

A todos os professores do Departamento dos quais fui aluno no mestrado, sempre dispostos a tirar dúvidas do conteúdo e empenhados na boa formação dos alunos.

A todos os professores e profissionais da minha Graduação na UFSCar.

A todos aqueles que de algum modo contribuíram para a conclusão desse trabalho. Muito obrigado pela paciência e companheirismo durante o mestrado.

Sobretudo, agradeço a Deus.

Resumo

Silva, R. R.; *Um Framework para Construção de Sistemas Inteligentes de Seqüenciamento da Produção*. Dissertação de Mestrado. Universidade Federal de São Carlos – Departamento de Computação. São Carlos, 2009.

No Tear (Laboratório de Pesquisa e Desenvolvimento de Tecnologia e Estratégias de Automação) vários trabalhos vêm sendo desenvolvidos na temática de seqüenciamento e programação da produção. Em um dos focos, o de seqüenciamento de produtos na entrada de um sistema produtivo, tem-se investigado o uso de simulação em cooperação com técnicas de Inteligência Artificial buscando-se uma aplicação como solução para condições de re-seqüenciamento ou de seqüenciamento reativo da entrada do sistema produtivo.

A proposta deste trabalho é modelar um *framework* para um sistema de auxílio ao seqüenciamento na entrada do sistema produtivo. O *framework* deve servir como guia na construção e customização do sistema.

Palavras-chave: Seqüenciamento da Produção. Simulação de Sistemas Industriais. Sistemas Inteligentes.

Abstract

Silva, R. R.; *A Framework to Intelligent Systems Construction of Production Sequencing*. São Carlos, 2009. Master's Degree Dissertation. Federal University of São Carlos – Computer Science Department.

There are many researches being developed in the Production Sequencing and Production Scheduling by the Tear team. In one of the researches focus, it has been investigated the use of simulation in cooperation with Artificial Intelligence to obtain a computer system as the solution of a reactive system in the entrance of production system.

The main goal is to have a framework model to an intelligent system to help the products sequencing in the entrance of production systems. The framework should be a guide in the construction and customization of the intelligent system to be constructed.

Keywords: Production Sequencing. Industrial Systems Simulation. Intelligent Systems.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

ACS	Avaliador de Cenários Nebulosos.
ANCSP	Analisador Nebuloso de Cenários para o Seqüenciamento da Produção.
FMS	Sistema Flexível de Manufatura (do inglês *Flexible Manufacture System*).
MVC	Do inglês *Model-View-Controller*.
SADSP	Sistema de Apoio à Decisão para o Seqüenciamento da Produção.
SISP	Sistema Inteligente para o Seqüenciamento da Produção.
SPPS	Sistema de Planejamento da Produção Baseado em Simulação.
UML	Linguagem de Modelagem Unificada (do inglês *Unified Modeling Language*).

Capítulo 1 - Introdução

No mercado global atual, as mais bem sucedidas indústrias são aquelas que apresentam produtos com menor preço, maior qualidade e menor prazo para entrega, sem comprometer seus lucros, necessários para investimento em pesquisa e desenvolvimento de novos produtos. Para atingir os seus objetivos, as indústrias dão cada vez maior importância ao planejamento da produção e à modernização do sistema de manufatura, utilizando o desenvolvimento de processos de fabricação mais eficazes e de flexibilização da produção (Fernandes, 2004).

A adoção de Sistemas Flexíveis de Manufatura permite que um mesmo sistema de produção produza diversos produtos com o mínimo de intervenção manual. O uso de FMSs aumenta a flexibilização da produção, aumentando a produtividade (Silva, 2005). Mas acarreta na existência de vários tipos de decisões a serem tomadas (Smith & Stecke, 1996). A de maior interesse para o trabalho é a de como ocorrerá o seqüenciamento dos produtos na entrada de um sistema produtivo, podendo o sistema produtivo ser representado por um FMS.

O Grupo Tear desenvolve trabalhos para auxílio no seqüenciamento de entrada de um sistema produtivo. A motivação do trabalho aqui apresentado vem de ter os trabalhos previamente desenvolvidos nessa área integrados num único sistema, fornecendo um guia para obter esse cenário. Considerar uma estrutura que permita os trabalhos do Grupo Tear integrados num único sistema, assim como possibilidade e integração de novos trabalhos é vantajoso. Guia a arquitetura básica da definição de futuros trabalhos para rápida inclusão num sistema único. Agiliza eventuais manutenções e/ou customizações de trabalhos previamente desenvolvidos e mesmo do sistema como um todo. Direciona a migração do sistema para uso comercial.

O objetivo principal do trabalho é modelar um *framework* para um sistema computacional de auxílio ao seqüenciamento na entrada de um sistema produtivo. O *framework* deve ser construído para servir como guia na construção e customização do sistema. Portanto, deve-se preocupar em modelar uma estrutura que oriente customizações do sistema.

O *framework* deverá ser voltado para sistemas que auxiliem na definição do seqüenciamento de entrada de um sistema de produção através de simulação ou através de técnica de filtragem de seqüências para uso no sistema produtivo. A reutilização de código deverá ser também focada, definindo-se independência entre componentes de interface gráfica com componentes lógicos e componentes de armazenamento de dados.

1.1 Estrutura do Trabalho

O foco do trabalho é apresentar um *framework* para sistemas computacionais que auxiliem no seqüenciamento de entrada de um sistema produtivo.

Apresenta-se o capítulo 2 com revisões bibliográficas de conceitos e trabalhos pesquisados de importância para a proposta.

O capítulo 3 aprofunda em trabalhos do Grupo Tear que serviram de maior motivação para a necessidade do *framework* e que estabelecem seu contexto, o contexto da proposta.

O capítulo 4 mostra como foram definidas as estratégias de modelagem para a definição da proposta. Conceitos usados na modelagem *framework* são apresentados.

A proposta do trabalho é discutida em detalhes no capítulo 5. Diagramas de classes, pacotes, são apresentados. Discutem-se também apontamentos de como estender o *framework* para criação de um sistema inteligente de auxílio no seqüenciamento de entrada do sistema produtivo.

As conclusões sobre a proposta apresentada e apontamentos para futuros trabalhos são discutidos no capítulo 6.

Capítulo 2 – Revisões Bibliográficas

Este capítulo apresenta uma visão geral de conceitos considerados e trabalhos pesquisados.

2.1 Framework

Foote & Johnson (1988) definiram *framework* como um projeto abstrato (em alto nível) orientado a objetos para um tipo de aplicação, que consiste de uma classe abstrata para cada importante componente do tipo de aplicação contemplado. O artigo também aponta que *frameworks* são úteis não somente para o reuso do código principal de uma aplicação, mas também para descrever abstratamente como seriam os componentes da biblioteca da aplicação. A habilidade de permitir a extensão de componentes de bibliotecas é um dos pontos fortes do conceito de *framework*.

Um *framework* geralmente faz o papel de comandar a seqüência de execução de uma aplicação. O *framework* geralmente tem a função de ditar como a aplicação deve se comportar, através do *design* de suas classes (Foote & Johnson, 1988). Um *framework* é um padrão de arquitetura que fornece um *template* extensível para aplicações dentro de um domínio (Booch *et. al.*, 2005).

Gamma *et. al.* (2000) afirmam que o *framework* dita a arquitetura de uma aplicação. O *framework* é um conjunto de classes cooperantes que constroem um projeto reutilizável para uma determinada categoria de software. Ele define a estrutura geral do projeto, as responsabilidades-chave das classes de objetos, como estas colaboram, e o fluxo de controle.

É reforçado por Gamma *et. al.* (2000) que os *frameworks* sempre têm um particular domínio de aplicação, podendo, portanto, um *framework* para editor gráfico ser usado para construção de parte do software de simulação de uma fábrica, mas não podendo confundi-lo como um *framework* para simulação.

Gamma *et. al.* (2000) apontam que *frameworks* enfatizam a reutilização de projetos. Ao utilizar um *framework*, o desenvolvedor reutiliza o corpo principal definido e escreve o código que o corpo chama. O desenvolvedor terá que escrever operações com nomes e convenções de chamada já especificadas, reduzindo, portanto, as decisões de projeto a serem tomadas. No entanto um *framework* inclui subclasses concretas que o desenvolvedor pode usar diretamente, sem necessidade de reescrita e/ou adaptação.

Os *frameworks* estão se tornando cada vez mais comuns e importantes, de acordo com Gamma *et. al.* (2000). Através deles sistemas orientados a objetos conseguem maior reutilização, provendo a maior parte do projeto e do código de uma aplicação.

2.2 Seqüenciamento da Produção

O seqüenciamento da produção é responsável por determinar a ordem em que os produtos serão executados no sistema produtivo (Slack *et. al.*, 1997).

Para estabelecer o seqüenciamento dos produtos, é necessário escolher uma regra de prioridade. Segundo Tubino (2000), regras de prioridade são heurísticas utilizadas para selecionar, a partir de informações sobre os produtos e/ou sobre o estado do sistema produtivo, qual dos produtos terá prioridade de processamento. Ao aplicar uma regra de prioridade, os produtos são classificados e uma seqüência é gerada. Carvalho (2003), Fernandes (2004) e Silva (2005) mostram exemplos de regras de prioridade encontrados na literatura.

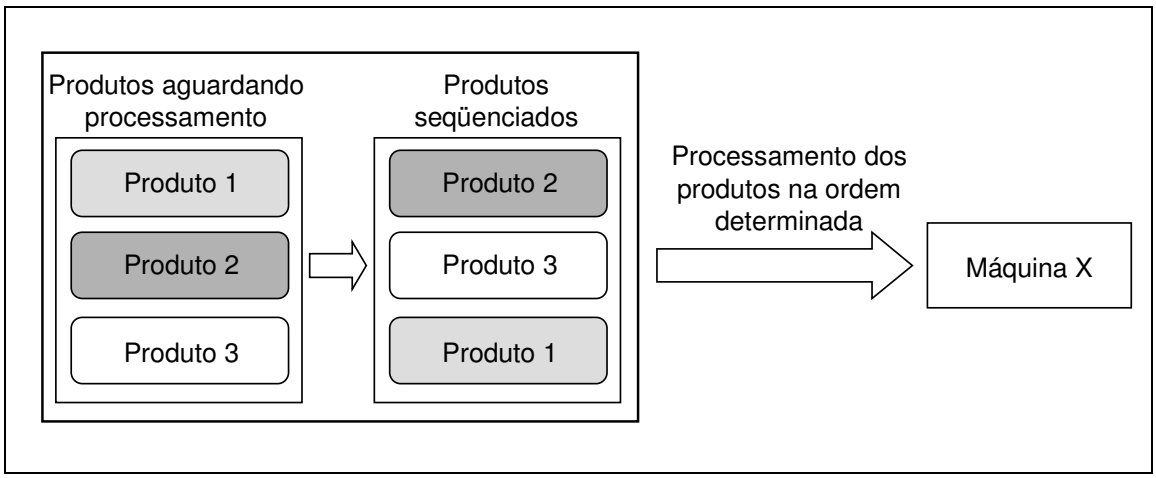


Figura 2.1: Seqüenciamento feito em cada máquina

O seqüenciamento da produção pode ser dividido, no entanto, em duas abordagens. O seqüenciamento pode ser realizado em cada máquina do sistema produtivo, como mostrado na figura 2.1. Outra forma, como mostrado na figura 2.2, é realizando o seqüenciamento na entrada do sistema produtivo.

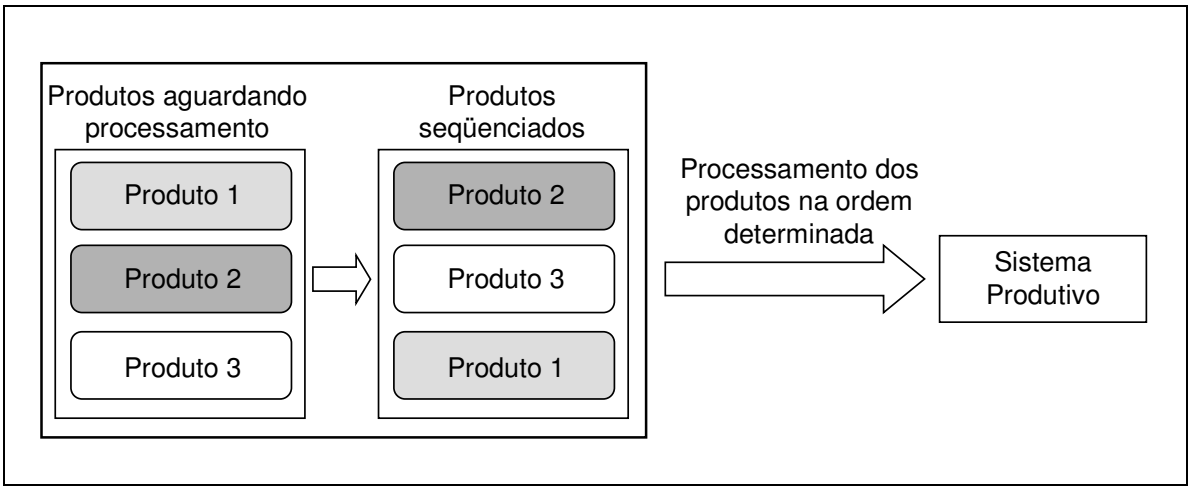


Figura 2.2: Seqüenciamento feito na entrada do sistema produtivo

2.3 Trabalhos na Área de Seqüenciamento de Entrada do Sistema Produtivo

Foram pesquisados trabalhos realizados na área de pesquisa de seqüenciamento da entrada do sistema produtivo.

Alguns trabalhos têm realizado uso de algoritmos genéticos como forma de definição do seqüenciamento da entrada da produção. Algoritmos genéticos lidam com busca randômica em grandes espaços, trabalhando com o conceito de população para, através do princípio da sobrevivência do mais apto ao meio, descobrir uma quase ótima solução após uma série de iterações de evolução da população.

Boqin *et. al.* (2005) implementaram uma versão melhorada de algoritmo genético para seqüenciamento da produção. Foram propostas melhorias em aspectos do algoritmo genético. A versão implementada foi testada ao aplicá-la a cenários também usados em outros conhecidos algoritmos genéticos. Os resultados mostraram melhor desempenho ns critérios desejados para o trabalho proposto.

Kampk & Kochel (2006) atacaram o problema do seqüenciamento e da otimização do tamanho do lote num FMS através da proposta de um sistema que combina simulação com uso de algoritmo genético. São estabelecidos dois módulos, o simulador e o de otimização, sendo o de otimização o responsável pela entrada e análise da saída do simulador e o que faz processamento de algoritmo genético. Os resultados de comparação com resultados obtidos usando-se somente algoritmo genético com os da combinação de algoritmo genético com simulação mostraram que a abordagem do trabalho é interessante, e, de fato,

atende a proposta. No entanto, mais testes e experimentos foram apontados como necessários para se tirar melhores conclusões.

O uso de lógica *fuzzy* foi constatado como aplicado para ajudar na solução do seqüenciamento de produção.

O trabalho de Fernandes (2004) apresenta um sistema que, com o uso de técnica *fuzzy*, promove a redução de cenários a serem usados por um módulo simulador. Desenvolvido dentro do grupo Tear, o trabalho apresentou resultados satisfatórios ao verificar que as seqüências por ele apontadas a serem simuladas estavam entre as que apresentaram melhor resultados, após simulação de todas possíveis no cenário de teste.

Também dentro do grupo Tear, Silva (2005) criou um sistema também usando técnica *fuzzy*. O sistema avalia, dando notas, às seqüências simuladas. Após o uso de um módulo de simulação, o sistema ajuda na tomada da decisão de escolha da seqüência de produtos a usar num sistema. A avaliação de desempenho do sistema, positiva, aconteceu ao se avaliar todas as seqüências simuladas quanto aos critérios adotados pelo trabalho, e, em seguida, comparar com os resultados obtidos pelo sistema implementado.

Foram também encontrados trabalhos apontando desenvolvimento de algoritmos específicos ou que melhorassem outros trabalhos existentes, assim como trabalhos dedicados a linhas de produção particulares.

Dentro do grupo Tear, Carvalho (2003) propôs um sistema para redução do número de seqüências de entradas a serem simuladas. O sistema consistia de um algoritmo, contemplando critérios específicos de desempenho. A medição do desempenho do sistema deu-se ao ter um modelo de simulação, com seqüências específicas, que foram simuladas, avaliadas, e comparadas com as apontadas na redução promovida pelo sistema implementado.

Também dentro do grupo Tear, Roman (2006) propôs um sistema que integrasse o sistema redutor de cenários apontado por Carvalho (2003), um módulo simulador, e o sistema avaliador de cenários simulados apontado por Silva (2005). Testes realizados com o sistema mostraram que ele conseguia de fato ter num só sistema os módulos apontados.

Chen & Shukla (2001) produziram um sistema inteligente de apoio à decisão para escolha da entrada de matéria-prima num FMS. Uma rede neural foi construída para se obter a seqüência de entrada de matérias-primas num FMS através de treinamento por resultados de simulação do modelo de FMS no software Automod. O sistema discutido apresenta banco de dados e leva em conta também dados passados. Houve preocupação no menor tempo de resposta possível para que seja de fato um sistema de tempo real. São levados

em conta os dados correntes do FMS para o uso da rede neural. Comparações com resultados de simulações para mesmas seqüências e uso das mesmas num FMS real mostraram resultados favoráveis para as combinações indicadas pelo sistema.

Morton *et. al.* (2002) propuseram o uso de simulação para auxiliar na análise do planejamento do seqüenciamento de produtos. O seqüenciamento é aplicado num ambiente onde cada produto (fios de metal) possui um carretel específico a ser empacotado e máquinas específicas para ser processado. A simulação é usada para todos cenários possíveis, possibilitando escolher a que apresenta resultados mais satisfatórios de acordo com os critérios desejados. A simulação também apresenta a utilidade de apresentar o resultado do que aconteceria se acontecesse certa seqüência ou certos eventos ou certas alterações no modelo proposto.

Chan *et. al.* (2003) desenvolveram um modelo de simulação de FMS para minimizar o tempo médio de fluxo, o atraso médio e o adiantamento médio. No modelo desenvolvido nesse trabalho, a regra de despacho é dinamicamente alterada. Para medição do desempenho do modelo, há comparações de seus resultados com os melhores obtidos ao se usar um único critério para despacho (regra de produção estática). As comparações feitas indicam vantagem de desempenho para o trabalho desenvolvido.

Dar-El *et. al.* (2006), com o objetivo de maximizar o *throughput* e minimizar o tempo de processamento dos produtos, propuseram uma heurística própria para o seqüenciamento na entrada da produção, validando seu resultado ao confrontar as medidas que se desejou trabalhar entre o trabalho deles e o de outros autores. Os resultados indicaram vantagens para a heurística adotada no trabalho.

Bard *et. al.* (2008) trabalharam numa heurística para resolver o problema do seqüenciamento de entrada num sistema de manufatura de eletrônicos com alto volume de produção, processando os produtos conforme demanda. O sistema era composto por várias linhas de produção, onde cada linha processa um limitado número de famílias de produtos (armazenadas numa espécie de *buffer* para entrada numa linha, sendo que cada família, na verdade, uma placa para introdução de componentes). A entrada no *buffer* de uma linha implica em tempo de *setup* (armazenamento das placas, retirada da linha de componentes que eram destinados para os da família que cedeu lugar no *buffer*, inserção na linha de componentes para a família que entrou no *buffer*). Bard *et. al.* focaram em reduzir os gastos com *setup*. Conseguiram verificar bons resultados do trabalho ao comparar a aplicação da heurística trabalhada por eles com a aplicação da heurística previamente aplicada na fábrica em estudo. A comparação foi feita usando-se dados reais.

Rahimi & Tavakkoli (2006) desenvolveram um algoritmo para resolver o problema do seqüenciamento numa linha de produção para vários tipos de produto. Simultaneamente, o algoritmo desenvolvido busca minimizar o trabalho gasto na produção, a variação no total de produtos gerados e o custo com *setup* de máquinas. O algoritmo mostrou-se mais eficiente nos objetivos que buscava ao se comparar o resultado da aplicação dele com os resultados de aplicação de um software que resolvia o mesmo problema. A comparação ocorreu aplicando-se tanto o software como o algoritmo a dados gerados por distribuições uniformes. Jolai *et. al.* (2007) também desenvolveram algoritmo para resolver o mesmo problema, para o mesmo tipo de linha de produção. No entanto, verificou-se maior eficiência do algoritmo do trabalho de Jolai *et. al.* (2007) ao se comparar o resultado da aplicação dele com os resultados de aplicação de outros três conhecidos algoritmos genéticos pelos autores.

Maravelias & Prasad (2008) partiram para construção de um algoritmo que busca resolver ao mesmo tempo selecionar como ficam os lotes numa linha de produção, em quais máquinas ficam presentes, e a seqüência dos mesmos, apresentando possibilidades de otimização quanto ao *makespan*, redução de atraso e custo da produção. Testes foram conduzidos pelo algoritmo construído ao compará-lo com outro algoritmo existente. O algoritmo mostrou-se mais eficiente nos critérios que buscou otimizar.

He, Smith (2007) propuseram um algoritmo para auxiliar no problema do seqüenciamento dos produtos na entrada de um FMS. O algoritmo é proposto para ser usado de forma reativa para o re-seqüenciamento de entrada, buscando balancear a carga de uso das máquinas do FMS. Na definição do algoritmo são usadas funções matemáticas discretas. Uma análise teórica dos autores, considerando matemática discreta, chega à conclusão de que o algoritmo proposto é eficiente para balancear o uso das máquinas do FMS. De acordo com a literatura referida no artigo, a melhora no balanço de uso das máquinas do FMS infere em maior produtividade. Portanto, os autores chegaram à conclusão de que o algoritmo proposto incrementa a produtividade de FMSs.

Kim *et. al.* (2001) investigaram a resolução do problema do seqüenciamento dos produtos na entrada de um FMS usando heurísticas para duas fases da solução do problema. A proposta deles é dividir o problema em dois subproblemas: o agrupamento de uma quantidade de produtos para formar grupos de diferentes tipos de produtos e o seqüenciamento dos grupos de produtos. Para o primeiro subproblema é necessário definir a quantidade para cada tipo de produto a integrar um grupo de produtos. Para o segundo subproblema é necessário definir a seqüência dos produtos dentro do grupo. Assim sendo, os autores consideram o uso de algoritmos dedicados para solução do primeiro subproblema. E

algoritmos dedicados para solução do segundo subproblema. O trabalho conclui, através de simulações com geração de dados randômicas e comparações com resultados sem uso da abordagem com duas fases para solução, que decompor o problema em dois subproblemas para resolução apresentou melhor resultado para minimização do *makespan*.

Smith & Stecke (1996) propõem integrar a solução do problema da pré-determinação da quantidade de produtos a formarem os lotes a serem seqüenciados com o problema da ordem da seqüencia dos lotes na entrada do sistema produtivo. Os autores fazem uso de técnicas de simulação para realizar os experimentos, confrontando os resultados apresentados quando usadas regras de decisão proposta no trabalho e quando não usadas as mesmas. Os autores propõem regras de prioridade que buscam aumentar a taxa de utilização das máquinas e que são aplicadas periodicamente ao sistema produtivo, dado que máquinas podem ficar indisponíveis. Os resultados mostrados no trabalho apontam melhor desempenho do FMS (em relação ao uso das máquinas) quando também considerado o tamanho dos lotes dos produtos na entrada do sistema produtivo.

Capítulo 3 - Trabalhos do Grupo Tear

O grupo Tear trabalha no contexto de auxiliar o usuário na tomada de decisão para escolha de uma seqüência a usar na entrada do sistema produtivo de forma reativa.

Partindo do projeto SPPS – Sistema de Planejamento da Produção Baseado em Simulação (Kato *et. al.*, 2000), que realiza tomada de decisão reativa em uma fábrica a partir de dados recebidos da mesma, e da modelagem de um FMS (Morandin, 1999), foram desenvolvidos quatro trabalhos de mestrado dentro do grupo Tear, motivando a proposta do trabalho aqui discutido. Esses quatro trabalhos são discutidos em maiores detalhes nas subseções do corrente capítulo.

Considerando um contexto em que se tem um sistema produtivo, e que, para determinar a melhor seqüência de entrada de produtos nesse sistema é usado um módulo de simulação, o trabalho "Um Modelo de Seqüenciamento da Produção para um Sistema de Apoio à Decisão" (Carvalho, 2003) propõe um sistema que trabalha como um filtro de seqüências. O sistema proposto, agindo de forma reativa, através de algoritmo interno, seleciona, dentre todas as seqüências de entrada disponíveis, as que ele julga potencialmente terem maior chance de ser a melhor a ser escolhida. No contexto abordado pelo trabalho, o usuário teria então uma relação reduzida de seqüências para simulação após usar o sistema proposto. Maiores detalhes do trabalho de Carvalho (2003) são discutidos na seção 3.1.

No mesmo contexto que o trabalho de Carvalho (2003), o trabalho "Um Método de Análise de Cenários para Seqüenciamento da Produção Usando Lógica Nebulosa" (Silva, 2005) propõe um sistema reativo que também trabalha como um filtro de seqüências. A diferença para o trabalho de Carvalho (2003), é que Silva (2005) propõe a seleção das seqüências de entrada disponíveis através do uso de Lógica Nebulosa. Maiores detalhes do trabalho de Silva (2005) são discutidos na seção 3.2.

Abordando novamente o mesmo conceito dos trabalhos de Carvalho (2003) e de Silva (2005), o trabalho "Um Avaliador de Cenários Simulados para Re-seqüenciamento da Produção em Sistemas Automatizados de Manufatura usando lógica nebulosa" (Fernandes, 2004) propõe um sistema que trabalha como analisador de seqüências simuladas. O sistema proposto, através do uso de Lógica Nebulosa, auxilia um usuário a escolher qual a melhor seqüência a usar na entrada de um sistema produtivo do tipo FMS de forma reativa. A escolha do usuário deverá acontecer após simulação de um conjunto de seqüências, de acordo com o contexto considerado. O sistema atribui, para cada seqüência simulada, uma nota. Maiores detalhes do trabalho de Fernandes (2004) são discutidos na seção 3.3.

O trabalho "Um Sistema Inteligente para o Seqüenciamento da Produção com o Apoio de Simulação" (Roman, 2006) propõe um sistema que integra as funcionalidades

apresentadas pelos trabalhos de Carvalho (2003) e de Silva (2005), juntamente com um núcleo de simulação considerado nos contextos dos trabalhos de Carvalho (2003) e de Silva (2005). Maiores detalhes do trabalho de Roman (2006) são discutidos na seção 3.4.

Todos os trabalhos discutidos nessa seção consideram, portanto, a abordagem do problema do seqüenciamento como sendo a do seqüenciamento de produtos na entrada do sistema produtivo, podendo esse ser um FMS. É verificado e apontado pelos trabalhos discutidos nessa seção que medidas da execução de um modelo FMS simulado variam de acordo com a seqüência de produtos adotada na entrada. É mostrada essa abordagem do seqüenciamento na figura 2.2.

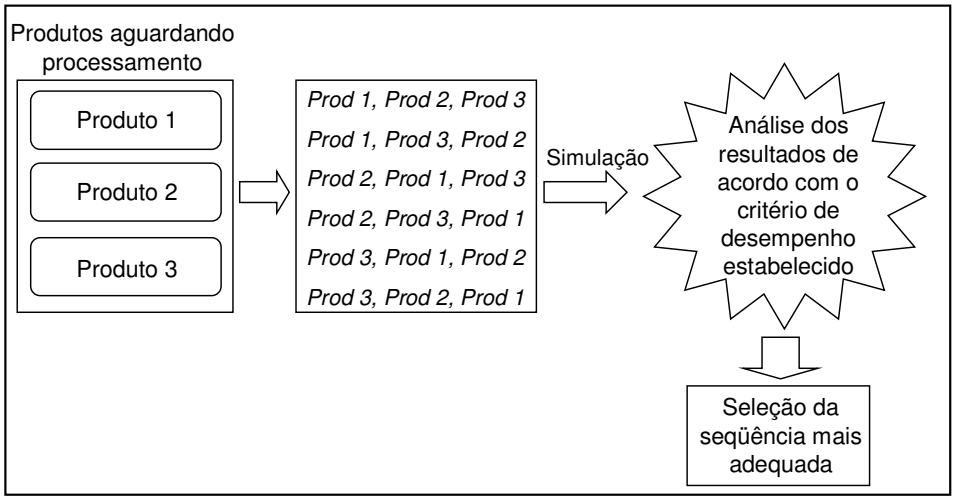


Figura 3.1: Uso de Simulação para seleção de seqüência mais adequada

Pelo contexto considerado, ao ocorrer algum evento não programado num modelo FMS real, é necessário confirmar qual a melhor seqüência de entrada do sistema a usar. Essa verificação se dá por simulação. Logo, existe, portanto, o problema da explosão combinatória. Num caso em que se tem 5 diferentes tipos de produtos para entrar no sistema produtivo, existem, portanto, 120 seqüências diferentes disponíveis. O responsável por decidir uma seqüência a usar após o evento não programado teria, portanto, que aguardar a simulação de 120 diferentes seqüências. É ilustrado na figura 3.1 o cenário da identificação da seqüência mais adequada através da técnica de simulação.

3.1 Sistema de Apoio à Decisão para o Seqüenciamento da Produção

O trabalho "Um Modelo de Seqüenciamento da Produção para um Sistema de Apoio à Decisão" (Carvalho, 2003) propõe um Sistema de Apoio à Decisão para o Seqüenciamento da Produção (cujo nome é abreviado para SADSP). O sistema proposto objetiva minimizar o número de seqüências de produtos disponíveis para a entrada de simulação de um modelo FMS, representando um Sistema Produtivo. Essa seção discute maiores detalhes do SADSP.

O SADSP tem como propósito auxiliar o responsável por decidir uma seqüência a usar na entrada do sistema produtivo após ocorrer evento não programado. O SADSP é usado para reduzir o número de seqüências que deverão ser simuladas, como mostra a figura 3.2. Um algoritmo interno do SADSP, desenvolvido também no trabalho de Carvalho (2003), seleciona um conjunto reduzido, de tamanho a ser determinado pelo usuário, das seqüências mais prováveis de serem verificadas como a melhor através da simulação. Menos seqüências são submetidas ao processo de simulação, e, assim, o usuário tem a duração da simulação otimizada.

Ao considerar o SADSP como um sistema de apoio à decisão, Carvalho (2003) considera que um sistema de apoio à decisão é um sistema computacional que provê informações em um dado domínio de aplicação, através de modelos de decisão e de banco de dados, de forma a auxiliar os tomadores de decisão a solucionarem problemas de natureza semi-estruturada e não-estruturada. Desse modo, o SADSP pode ser visto como um sistema de apoio à decisão auxiliando o usuário que necessita escolher uma seqüência de entrada no sistema quando ocorre evento não programado, e tendo que fazê-lo a partir de simulação.

Adaptando de *apud* Binder (1999), Carvalho (2003) considera como componentes de um sistema de apoio à decisão, o banco de dados, o banco de modelos e a interface gráfica com o usuário. O banco de dados é uma coleção de dados inter-relacionados, organizados de forma a atender as necessidades da organização (*apud* Turban & Aronson, 2001). Para que dados armazenados no banco de dados sejam transformados em informações relevantes à tomada de decisão, é necessário existir ao menos um modelo disponível no banco de modelos (*apud* Watson *et. al.*, 1997). Modelos permitem que um grande número de alternativas de solução de um determinado problema sejam analisadas. A interface gráfica é o componente responsável pela interação do usuário com o sistema.

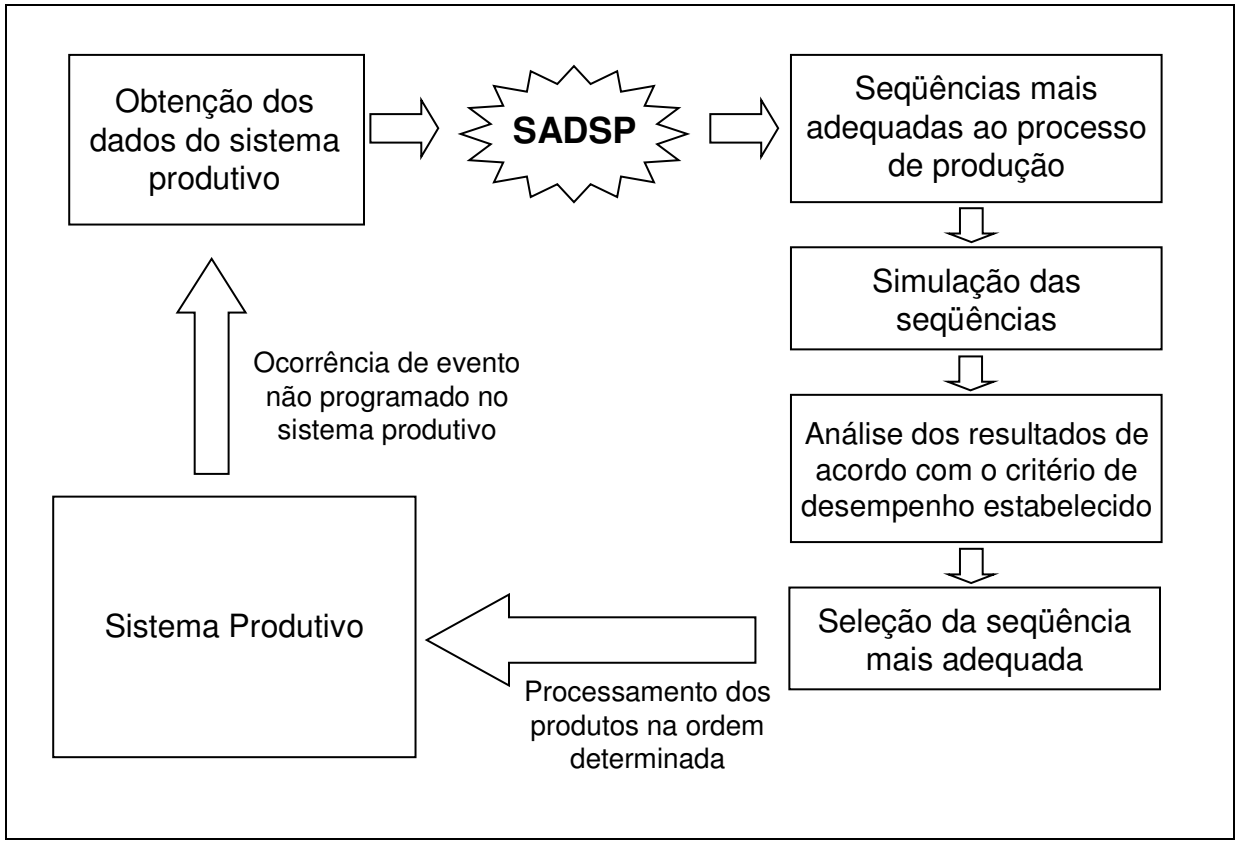


Figura 3.2: Uso do SADSP para redução do número de seqüências para simular

Carvalho (2003) identifica os componentes de um sistema de apoio à decisão do SADSP. O componente banco de dados é o que fornece as informações referentes ao sistema produtivo em que a aplicação auxilia o usuário. O componente banco de modelos é o algoritmo que irá reduzir o número de seqüências disponíveis para simulação, mediante consulta do banco de dados. O componente da interface gráfica é o a interface gráfica com o usuário que o SADSP possui.

O algoritmo do SADSP para redução do número de seqüências de entrada do sistema produtivo consiste em, a partir do número máximo de seqüências a serem simuladas informadas pelo usuário (número máximo de seqüências apontadas pelo SADSP), eliminar, primeiramente, seqüências que não respeitem a data de entrega dos produtos. Se ainda assim estiver provendo mais seqüências que o número máximo informado, procede-se à seleção de seqüências que gerem maior lucratividade. O critério final, para reduzir ainda mais, se necessário, o número de seqüências, é selecionar seqüências com menor tempo de produção.

Carvalho (2003) analisou o desempenho do SADSP comparando-se seqüências indicadas pelo SADSP com um conjunto de seqüências classificadas a partir do resultado obtido de simulações executadas neste conjunto de seqüências de entrada do sistema

produtivo. A análise mostrou que o SADSP lista pelo menos uma das seqüências mais adequadas ao sistema de produção.

3.2 Analisador Nebuloso de Cenários para o Seqüenciamento da Produção

O trabalho "Um Método de Análise de Cenários para Seqüenciamento da Produção Usando Lógica Nebulosa" (Silva, 2005) propõe e implementa um sistema nomeado como Analisador Nebuloso de Cenários para o Seqüenciamento da Produção (cujo nome é abreviado para ANCSP). O sistema proposto objetiva minimizar o número de seqüências de produtos disponíveis para a entrada de simulação de um modelo FMS. Essa seção discute maiores detalhes do ANCSP.

O ANCSP trata o problema da explosão combinatória do seqüenciamento, podendo ser usado para o re-seqüenciamento da produção por levar em conta eventos não programados. O ANCSP busca reduzir o número de seqüências a serem simuladas, como mostra a figura 3.3. O ANCSP seleciona um conjunto reduzido, de tamanho a ser determinado pelo usuário, das seqüências mais prováveis de serem verificadas, através de simulação, como a melhor em relação, principalmente, ao cumprimento de prazo de entrega. Menos seqüências são submetidas ao processo de simulação, e, assim, o usuário tem a duração da simulação otimizada. O usuário também é requisitado a indicar, pelo ANCSP, antes de sua execução, informações sobre eventos não programados, indicando máquinas quebradas e/ou matérias-primas que estejam em falta. Desse modo, o ANCSP consegue identificar produtos inviabilizados de serem produzidos e, portanto, seqüências não aplicáveis, mostrando, assim, sua importância ao re-seqüenciamento da entrada do sistema produtivo.

O ANCSP identifica, primeiramente, a partir da entrada das informações do usuário sobre máquinas quebradas e matérias-primas que estejam em falta, produtos que não podem ser produzidos, diminuindo, assim, o número de seqüências existentes para a entrada do sistema produtivo. A identificação é possível porque o ANCSP, segundo Silva (2005), realiza consulta à base de dados disponível dentro do ANCSP para verificar se uma máquina quebrada impede a produção de um ou mais produtos, ou se uma matéria-prima faltante implica também impede a produção de um ou mais produtos.

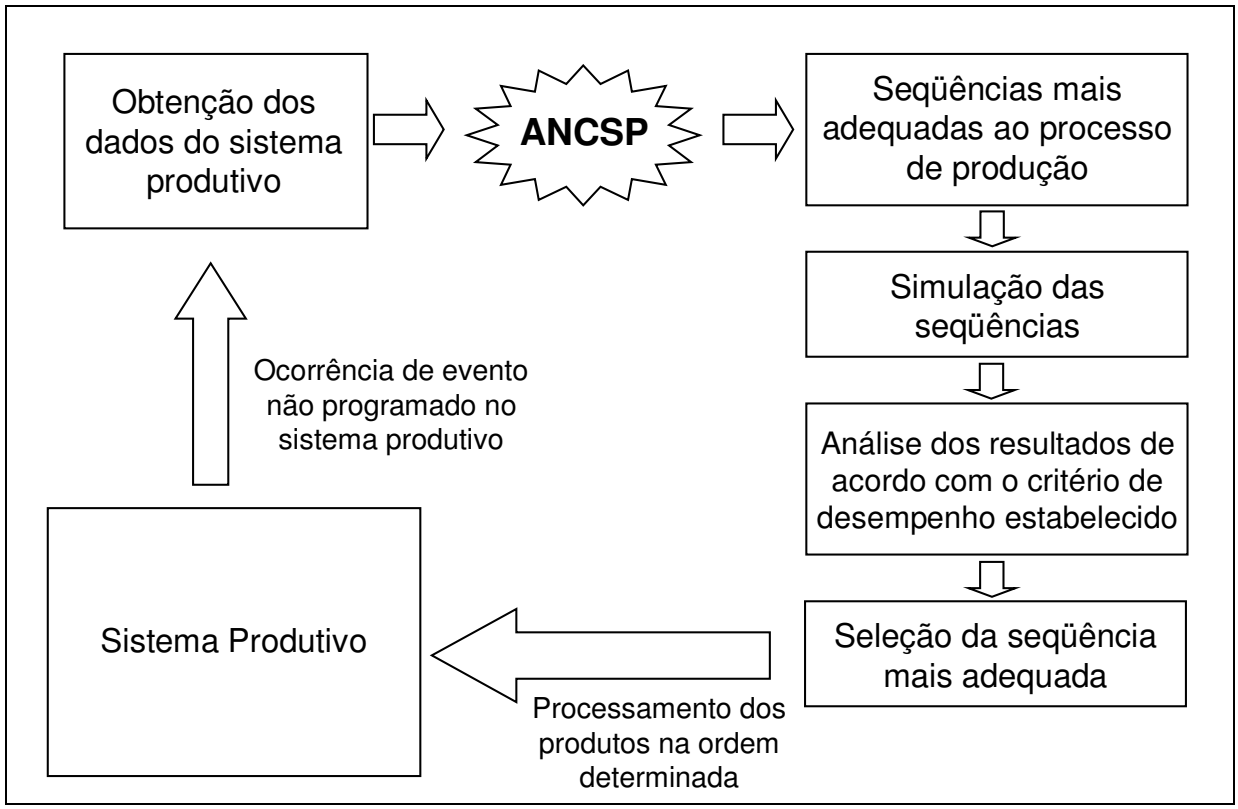
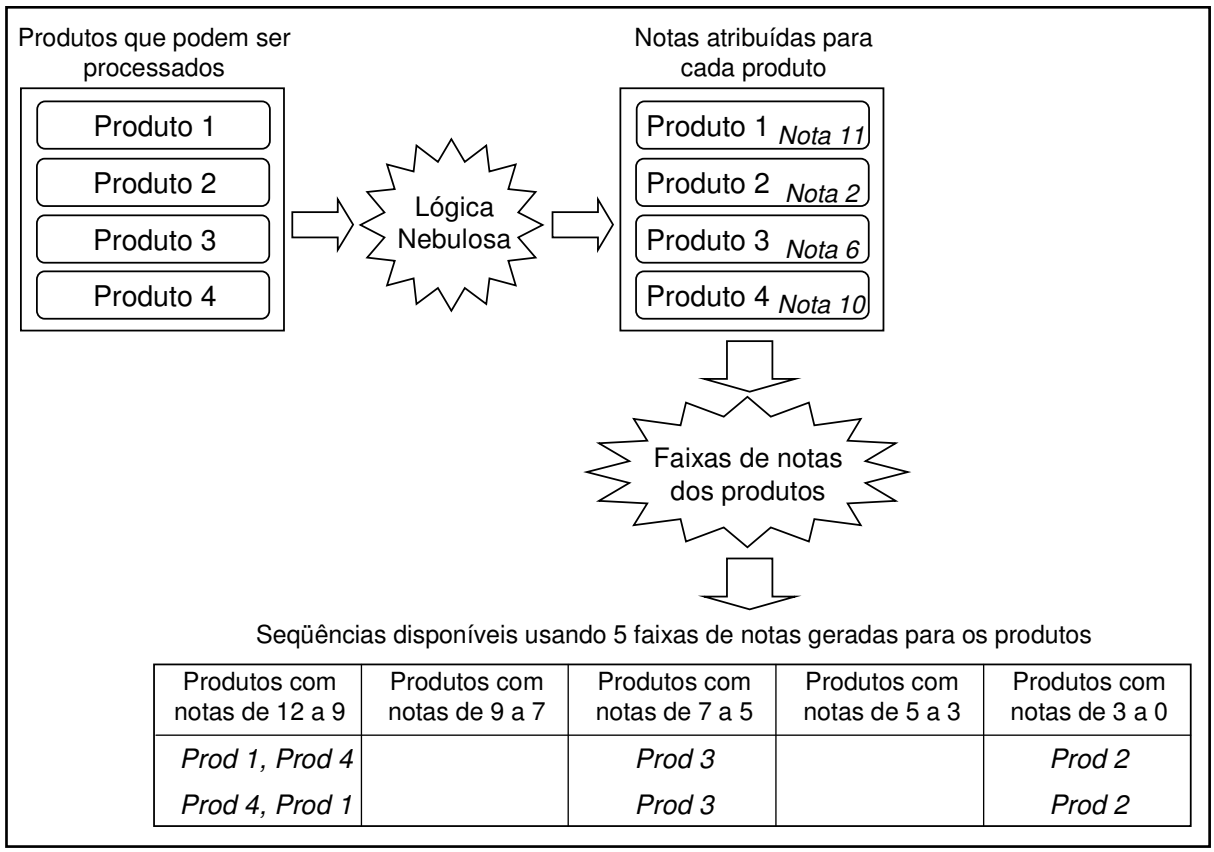


Figura 3.3: Uso do ANCSP para redução do número de seqüências a simular

Após a identificação de quais produtos o sistema produtivo é de fato capaz de produzir existem, então, duas etapas de funcionamento do ANCSP. Essas duas etapas são mostradas na figura 3.4. É usada lógica nebulosa considerando valores atualizados em relação a prazo de entrega dos produtos envolvidos no seqüenciamento, tempo de processamento total dos produtos, e variável lingüística de avaliação do roteiro de produção para os produtos. Para a lógica nebulosa, é aplicado o Método de Inferência de Mandani, com conversão nebuloso-escalar através do método Centro de Gravidade, fornecendo uma nota para cada produto identificado como possível de ser usado (sem impedimento por conta de máquina quebrada ou matéria-prima em falta). Os produtos são, então, agrupados em faixas de valores de suas respectivas notas, gerando-se seqüências que respeitem, inicialmente, cinco faixas das notas estabelecidas (podendo essas faixas ser aumentadas, até obter-se um número de seqüências dentro do limite estabelecido pelo usuário).

Silva (2005) realizou testes com o ANCSP, de modo a simular todas as seqüências de entrada de um FMS modelado no software de simulação Automod e verificar que o ANCSP apontava seqüências que estivessem entre as melhores de todos os conjuntos possíveis, em relação ao critério data de entrega. Os testes indicaram de fato que o ANCSP conseguia identificar seqüências que estavam sempre entre as melhores (senão a melhor) para uso na entrada do FMS considerado no trabalho.



Produtos com notas de 12 a 9	Produtos com notas de 9 a 7	Produtos com notas de 7 a 5	Produtos com notas de 5 a 3	Produtos com notas de 3 a 0
Prod 1, Prod 4		*Prod 3*		*Prod 2*
Prod 4, Prod 1		*Prod 3*		*Prod 2*

Figura 3.4: Aplicação de lógica nebulosa e faixas de notas para os produtos pelo ANCSP

3.3 Avaliador de Cenários Nebulosos

O trabalho "Um Avaliador de Cenários Simulados para Re-seqüenciamento da Produção em Sistemas Automatizados de Manufatura usando lógica nebulosa" (Fernandes, 2004) implementa um Avaliador de Cenários Nebulosos (título do sistema implementado pelo trabalho e abreviado para ACS). O objetivo do trabalho é auxiliar o usuário na escolha de qual seqüência usar na entrada de um sistema produtivo após simulação de um conjunto de seqüências.

O ACS busca, na ordem apresentada, priorizar, de acordo com Fernandes (2004), o cumprimento do prazo de entrega dos produtos e redução do custo de produção. A hierarquia de prioridades do ACS foi elaborada com base em Slack *et. al.* (1999). As prioridades do ACS é que justificaram a forma como a lógica nebulosa é aplicada, resultando em notas (conceitos) para cada seqüência de entrada simulada.

O funcionamento do ACS consiste em, ao ser iniciado, coletar dados dos usuários sobre o sistema produtivo, gerando todas as seqüências possíveis de produtos a serem usados na entrada do sistema produtivo. Em seguida, ocorrem simulações do sistema produtivo no Automod para todas as seqüências que podem ser usadas na entrada. Terminada a simulação, o ACS coleta dados das simulações ocorridas, usa lógica nebulosa, produzindo notas numéricas para cada seqüência. A lógica nebulosa usada considera a conversão de escalar para nebuloso pelo Método de Mandani, e a conversão de nebuloso para escalar pelo Método Centro de Gravidade. O funcionamento do ACS é mostrado na figura 3.5.

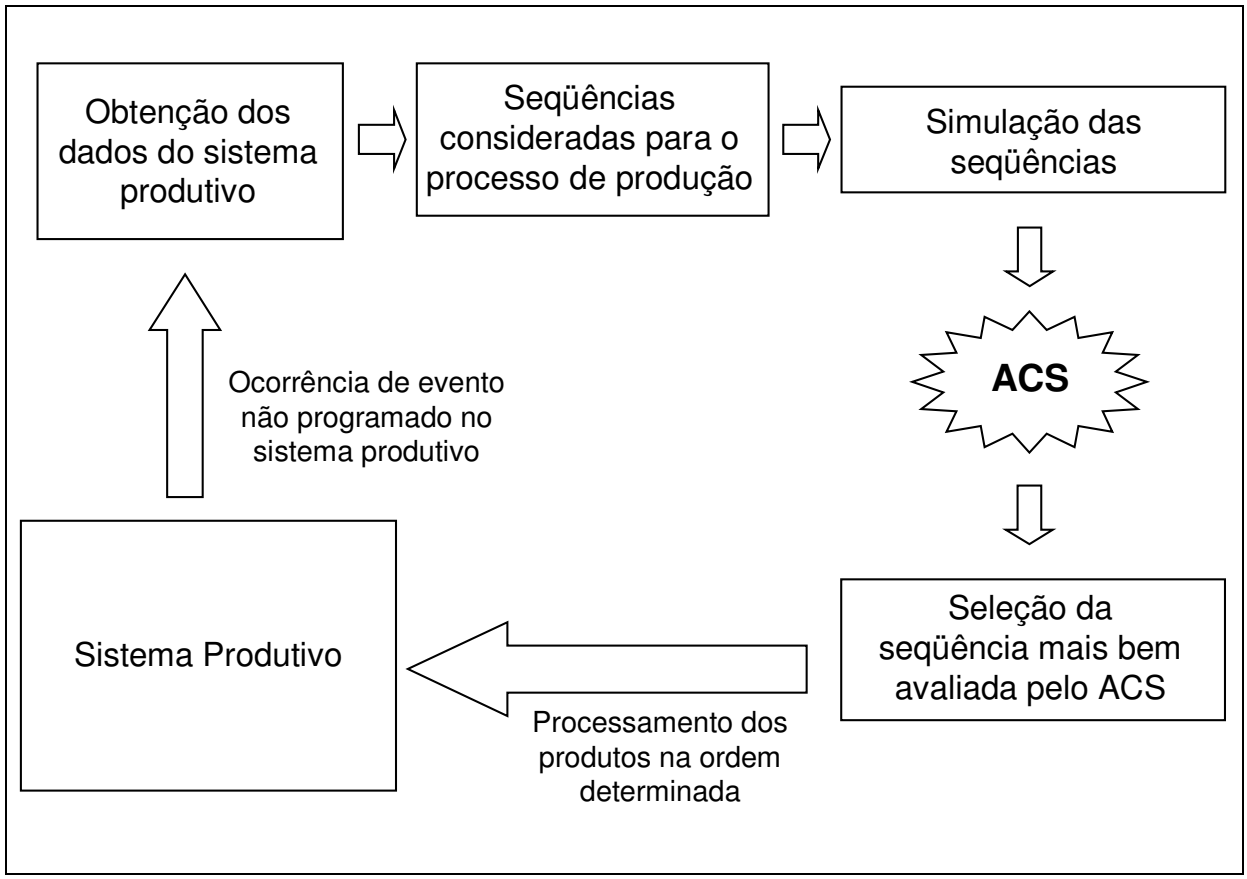


Figura 3.5: Funcionamento do ACS

Fernandes (2004) aponta que testes foram efetuados e indicaram que o ACS apresenta uma avaliação robusta e com boa precisão. Quanto maior a gama de produtos, maior a aplicabilidade do ACS. Isso devido à dificuldade de um usuário, necessitando definir melhor seqüência de entrada num sistema produtivo dentre um conjunto de simuladas, ter de avaliar uma quantidade grande de dados aferidos na simulação para cada seqüência simulada.

3.4 Sistema Inteligente para o Seqüenciamento da Produção com o Apoio de Simulação

O trabalho "Um Sistema Inteligente para o Seqüenciamento da Produção com o Apoio de Simulação" (Roman, 2006) propõe e implementa um sistema com o objetivo de auxiliar o re-seqüenciamento da entrada de produtos num sistema produtivo através do uso de simulação e dos sistemas desenvolvidos pelos trabalhos desenvolvidos por Carvalho (2003) e Fernandes (2004), o SADSP e o ACS. O sistema proposto e implementado por Roman (2006) leva o mesmo nome de seu trabalho, e é conhecido também pela sigla SISP.

O sistema proposto por Roman (2006) é um sistema reconstruído que agregou novas funcionalidades e aperfeiçoou modelos já desenvolvidos pelo grupo de pesquisa Tear em ferramentas diferentes. Ele integra os sistemas SADSP e ACS num único sistema, que melhor auxilie um usuário na tarefa de decidir, de forma reativa, a melhor seqüência de entrada num FMS.

O funcionamento do SISP é ilustrado de forma resumida na figura 3.6. Tendo um conjunto disponível de seqüências de entradas, o SADSP escolhe as que ele considera mais prováveis de ser a melhor quanto a metas estabelecidas, tendo, assim, um número de seqüências reduzidas. São simuladas as seqüências apontadas pelo SADSP. O resultado da simulação é analisado pelo ACS, que produz uma nota para cada seqüência simulada.

O SISP, no entanto, possibilita, por configuração, não usar o SADSP para filtrar a quantidade de seqüências a serem simuladas. Também, por configuração, pode-se desabilitar o uso do ACS para avaliação das seqüências simuladas. Dessa maneira, o sistema oferece certa flexibilidade de execução ao usuário. Ressaltando que em qualquer dessas configurações, o próprio SISP que dispara a simulação das seqüências de entrada.

Testes foram realizados, e, segundo Roman (2006), o SISP comportou-se de forma adequada. Foram apresentados, com detalhes dos resultados, dois casos de uso, habilitando e desabilitando o uso do SADSP.

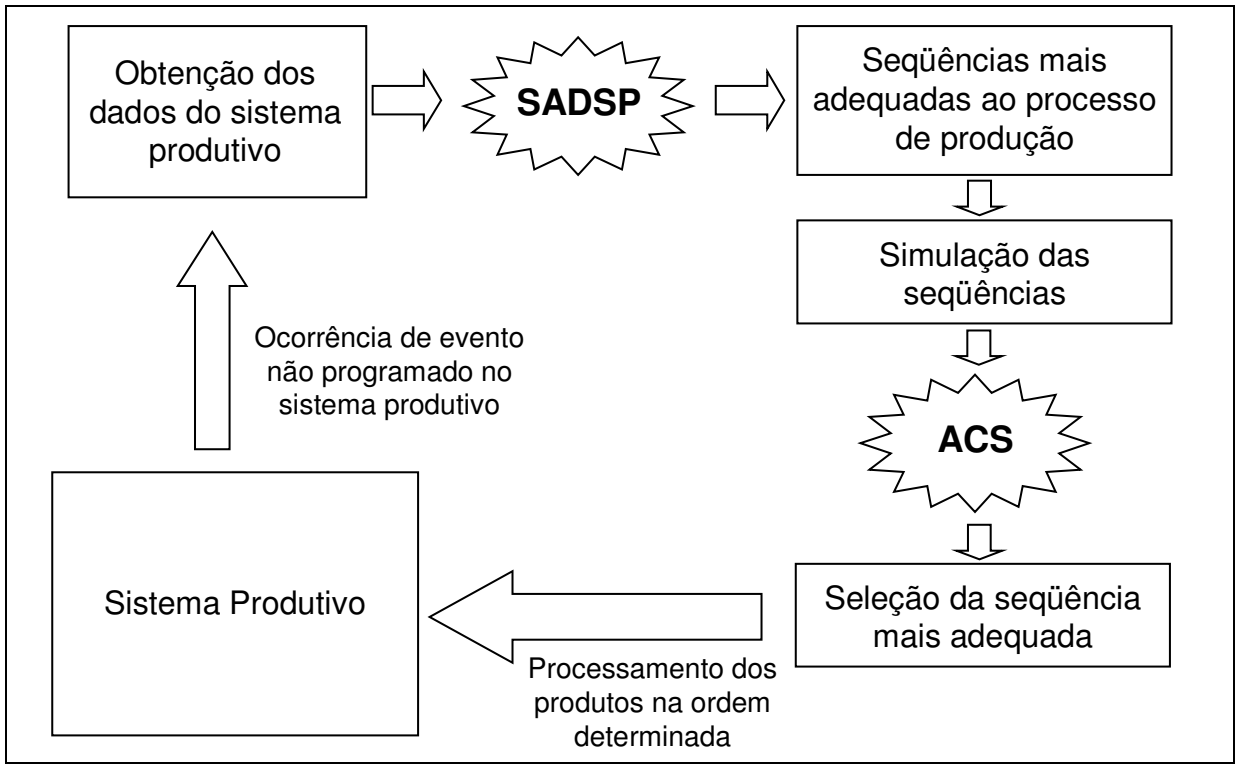


Figura 3.6: Funcionamento do SISP

Capítulo 4 - Estratégias de Modelagem

O capítulo 4 apresenta os sistemas concebidos pelos trabalhos de Carvalho (2003), Silva (2005) e Fernandes (2004). São trabalhos desenvolvidos dentro do grupo Tear, focados na ajuda da solução do seqüenciamento dos produtos na entrada de um FMS. Eles foram concebidos isoladamente, sem intenção de integração em um único sistema. Eles não apresentaram uma mesma metodologia de desenvolvimento, uma mesma linguagem, semelhantes arquiteturas, que possibilitassem agregá-los muito facilmente num único sistema sem necessidade de re-escrita de todo o código de um ou mais sistemas.

Roman (2006) propôs a unificação dos sistemas especificados por Carvalho (2003) e Fernandes (2004) em um único sistema. No entanto não houve um guia para como expandir e/ou customizar o sistema com inclusão de trabalhos semelhantes aos de Carvalho (2003) e Fernandes (2004), ou mesmo possibilitar outras opções de uso de simulação.

A elaboração da proposta do trabalho deve levar em conta estratégias que possibilitem, numa reconstrução, agrupar os trabalhos de Carvalho (2003), Silva (2005) e Fernandes (2004) num único sistema, aperfeiçoar a proposta de Roman (2006) para inclusão de futuros trabalhos com semelhante idéia.

4.1 Linguagem de Modelagem Unificada

Roman (2006) apontou o uso da UML (Linguagem de Modelagem Unificada, do inglês *Unified Modeling Language*), especificando o Sistema Inteligente para o Seqüenciamento da Produção (SISP) através de seu uso. O autor cita que a UML é usada para documentar projetos de software e também para especificar, visualizar e construir artefatos de sistemas. Segundo Booch *et. al.* (2005), a UML é uma linguagem-padrão para a elaboração da estrutura de projetos de software, permitindo a compreensão de um sistema.

Roman (2006), ao especificar um sistema com o uso de UML, apresentou também os diferentes diagramas existentes para compreendê-lo. Os diagramas são desenhados para permitir a visualização de um sistema sob diferentes perspectivas (Booch *et. al.*, 2005). O uso da UML, principalmente de seu Diagrama de Classes (mostrado no trabalho de Roman, 2006) faz-se importante devido toda a grande ajuda para compreensão de um sistema, de um *framework.*

Um exemplo de Diagrama de Classes é mostrado na figura 4.1, sendo este baseado em diagrama mostrado em Roman (2006). Nesse diagrama é possível ver o uso de classes, operações, atributos e associações. São definidas nele as classes Produto,

MateriaPrima, e ProdMat. É mostrada uma associação da classe Produto com a classe MateriaPrima, indicando que uma instância de Produto pode estar associada com 1 ou mais instâncias de MateriaPrima (através da notação *1..** colocada ao lado da classe MateriaPrima) e que uma instância de MateriaPrima pode estar associada a 1 ou mais instâncias de Produto (através da notação *1..** colocada ao lado da classe Produto). A classe ProdMat é desenhada como uma classe de associação, a ser sempre associada a um par existente entre as classes Produto e MateriaPrima.

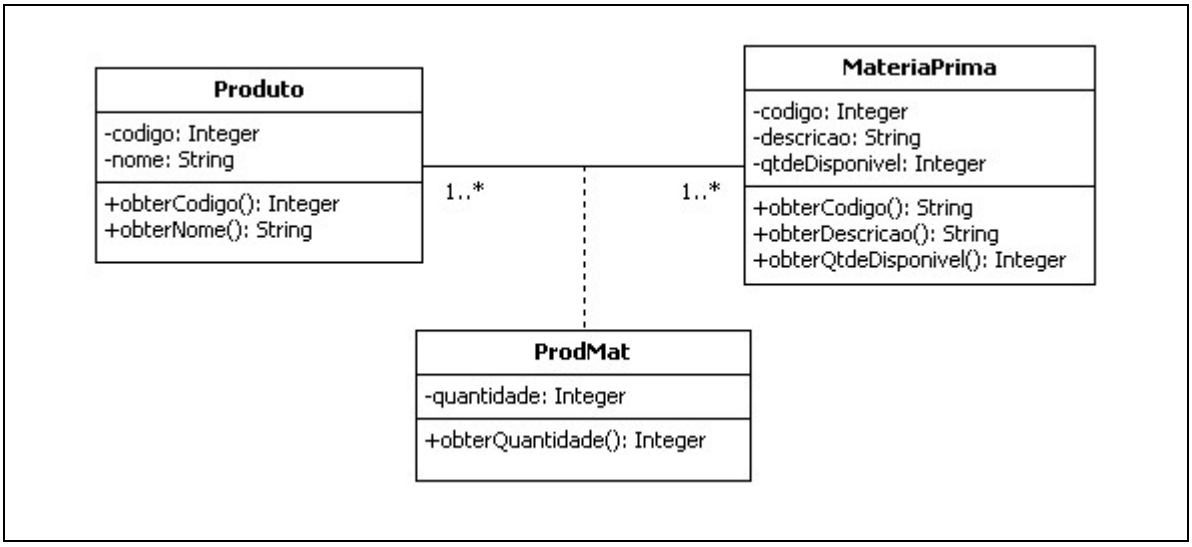


Figura 4.1: Exemplo de um Diagrama de Classes

4.2 Orientação a Objetos

Segundo Deitel & Deitel (2006), a modelagem orientada a objetos descreve uma aplicação em termos similares àqueles que as pessoas usam para descrever os objetos do mundo real. Objetos de certa classe, tais como os da classe de veículos, por exemplo, têm as mesmas características.

O uso de Orientação a Objetos foi usado por Roman (2006), que descreveu classes para compor o pacote banco de dados.

Booch *et. al.* (2005) afirma que a visão contemporânea no desenvolvimento de software adota uma perspectiva orientada a objetos. O principal bloco de construção de todos os sistemas de software é o objeto ou a classe. Um objeto é alguma coisa geralmente estruturada a partir do vocabulário do espaço do problema ou do espaço da solução. Uma classe é a descrição de um conjunto de objetos comuns. Todos os objetos têm uma identidade, um estado e um comportamento.

Ainda segundo Booch *et. al.* (2005), tem sido provado o valor do uso de Orientação a Objetos para a construção de sistemas em todos os tipos de domínios de problemas, abrangendo todos os graus de tamanho e de complexidade. Muitas linguagens, ferramentas e sistemas contemporâneos, de alguma forma, são orientados a objetos. Implica-se no fortalecimento da visão de mundo em termos de objetos.

A própria definição de *framework* envolve o uso de Orientação a Objetos, de acordo com a obra de Gamma *et. al.* (2000). Por buscar a modelagem de um *framework*, fica mais evidente ainda o uso de Orientação a Objetos no corrente trabalho.

4.3 Model-View-Controller

Os trabalhos de Carvalho (2003), Fernandes (2004) e Silva (2005) inferem a idéia de uma arquitetura em que se tem uma camada responsável pelo tratamento da interface gráfica com o usuário, outra responsável pela inteligência da solução a ser entregue e uma terceira responsável por gerenciar os dados necessários para funcionamento do sistema. Essa abordagem de arquitetura é conhecida como *Model-View-Controller*, ou, simplesmente, MVC (Krasner & Pope, 1988).

É ilustrada a abordagem da arquitetura MVC na figura 4.2, mostrando a interação entre suas principais camadas, ou elementos. A camada *Model* fica responsável por representar os dados persistidos em uma aplicação, sendo acessada tanto pela *Controller*, como pela *View*. A camada *View* é a interface gráfica mostrada ao usuário, que acessa tanto a *Controller*, como a *View*. A camada *Controller* é a que contém a lógica do sistema, sua inteligência, acessando apenas a *Model.*

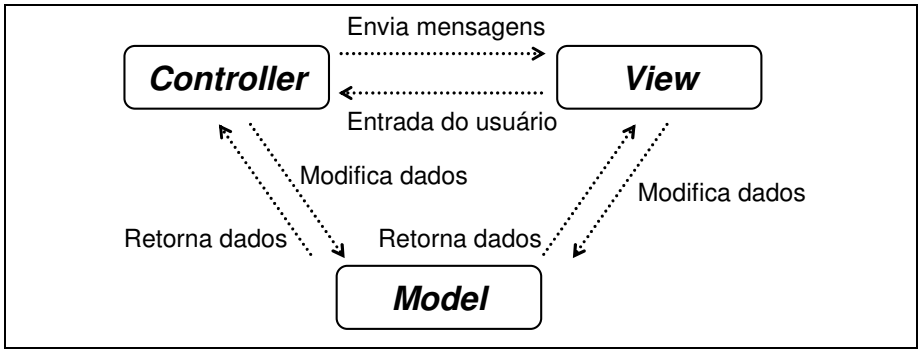


Figura 4.2: As camadas do MVC (Krasner & Pope, 1988)

A proposta do trabalho tem como foco modelar o *framework* para as camadas *Model* e *Controller*, auxiliando assim a construção personalizada da camada *View*, ao se usar as classes modeladas das camadas de que faz uso, conforme foi mostrado na figura 4.2.

4.4 Padrões de Projeto

Um padrão de projeto nomeia, abstrai e identifica os aspectos-chave de uma estrutura de projeto comum para torná-la útil para a criação de um projeto orientado a objetos reutilizável. O padrão de projeto identifica as classes e instâncias participantes, seus papéis, colaborações e a distribuição de responsabilidades (Gamma *et. al.*, 2000).

Um padrão de projeto é uma solução comum para um problema básico em um determinado contexto. O uso de padrões de projeto ajudam a visualizar, especificar, construir e documentar os artefatos de um sistema complexo de software (Booch *et. al.*, 2005).

É considerado o uso de Padrões de Projeto para a proposta. Existem padrões identificados para uso na modelagem do *framework* a que o trabalho se propõe. Segundo Booch *et. al.* (2005), um *framework* pode ser entendido como um tipo de microarquitetura abrangendo um conjunto de padrões de projeto que trabalham juntos para resolver um problema básico de um domínio comum.

A combinação do uso de dois padrões de projetos conhecidos como *Factory Method* e *Singleton* é usada para a modelagem do *framework*.

4.4.1 Factory Method

O *Factory Method* é um padrão de projeto com intenção de definir uma interface para criar um objeto, deixando as subclasses decidirem que classe instanciar. O *Factory Method* possibilita criar objetos dentro de uma classe com um método fábrica. Essa abordagem é mais flexível do que criar um objeto diretamente (Gamma *et. al.*, 2000).

A estrutura do padrão *Factory Method* é ilustrada na figura 4.3. Ela tem como participantes:

- *Product*: define a interface de objetos que *factoryMethod* cria.
- *ConcreteProduct*: implementa a interface de *Product*.
- *Creator*: declara *factoryMethod* (método fábrica), retornando um objeto do tipo *Product*.

- *ConcreteCreator*: redefine *factoryMethod* para retornar a uma instância de um *ConcreteProduct*.

Um trecho de pseudocódigo é também mostrado na figura 4.3, ilustrando como seria o código para obter-se uma instância de *ConcreteProduct*.

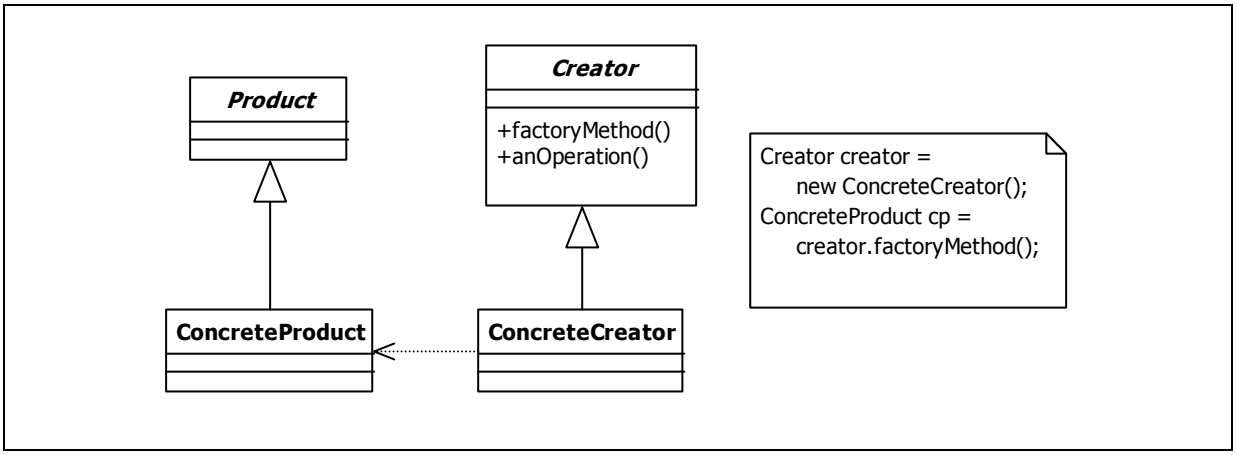


Figura 4.3: Estrutura do Padrão *Factory Method*

4.4.2 Singleton

O padrão *Singleton* é usado para garantir que uma classe tenha somente uma instância e fornecer um ponto global de acesso para a mesma (Gamma *et. al.*, 2000).

A estrutura do padrão *Singleton* é mostrada na figura 4.4. Ela tem como participante uma classe *Singleton*, que define um método estático *getInstance* e mantém um atributo estático *instance*. O método *getInstance* fica responsável por permitir aos clientes acessarem uma única instância de *Singleton*.

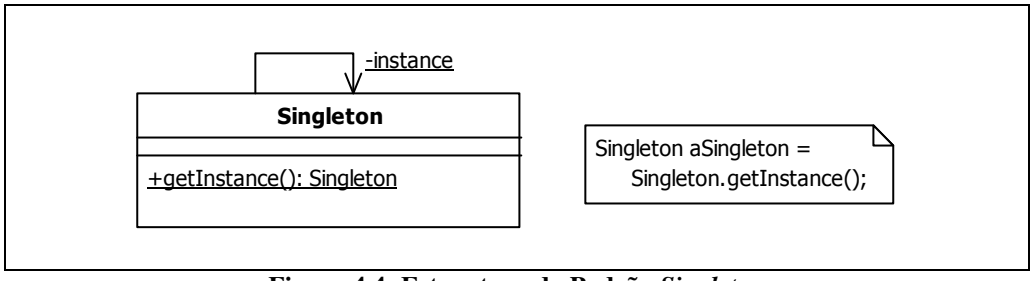


Figura 4.4: Esturutura do Padrão *Singleton*

4.4.3 Combinação de *Singleton* com *Factory Method*

O *Factory Method* pode ser variado para que o *FactoryMethod* da classe *Creator* seja parametrizado, *Creator* não seja uma classe abstrata. A combinação com *Singleton* se dá ao considerar a classe *Creator* como sendo uma *Singleton*. O resultado obtido é ilustrado na figura 4.5.

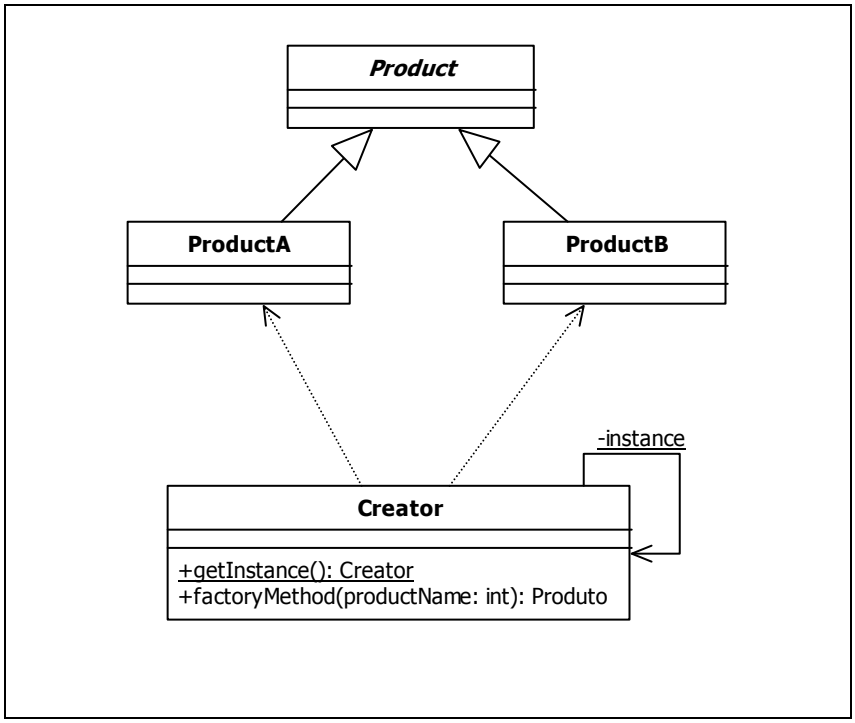


Figura 4.5: Estrutura da Combinação de *Singleton* com *Factory Method*

Capítulo 5 – Proposta do Trabalho

O capítulo apresenta a modelagem de um *framework* para construção de um sistema inteligente que auxilie o seqüenciamento de produtos na entrada de um sistema produtivo, proposta do trabalho. O sistema de auxílio é visto como uma integração dos sistemas propostos pelos trabalhos discutidos nas seções 3.1, 3.2 e 3.3 do capítulo 3 e faz uso das estratégias discutidas no capítulo 4.

O contexto de execução do sistema inteligente para auxílio no seqüenciamento da produção a que o *framework* a ser proposto se dispõe como guia é mostrado na figura 5.1. Identificam-se na figura 5.1 3 elementos que podem ser entendidos como módulos principais atuando no sistema: Filtro, Simulador e Analisador. O Filtro realiza a redução do número de seqüências para a simulação. O Simulador se encarrega da simulação das seqüências no sistema produtivo nele modelado. O Analisador avalia as seqüências simuladas, atribuindo notas a elas.

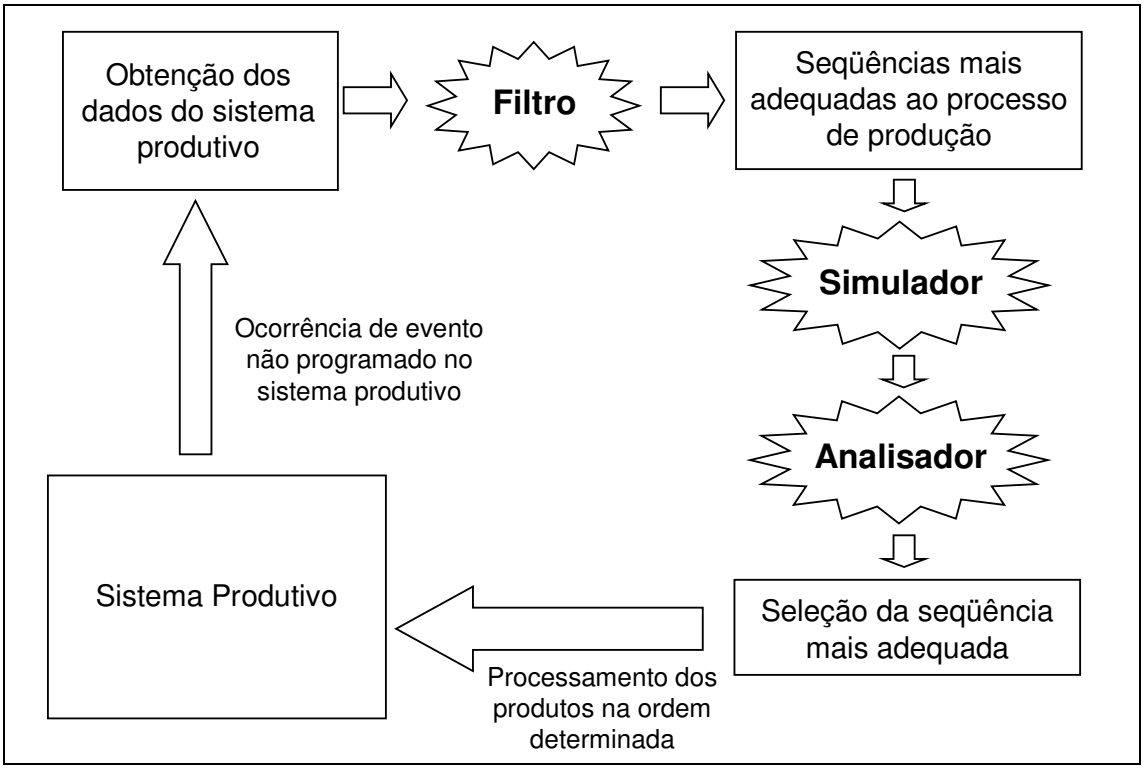


Figura 5.1: Contexto de execução de um sistema para seqüenciamento de entrada de um FMS

O *framework* a ser proposto é calcado no conceito de arquitetura MVC. No entanto, as modelagens a serem apresentadas são somente para as camadas *Model* e *Controller*. A camada *View* deve fazer uso das classes modeladas na *Model* e na *Controller*.

Para deixá-la flexível, no que concerne à sua construção, não são estipuladas classes a serem usadas na construção da interação com o usuário de fato.

Ao se analisar o SADSP, o ANCSP e o ACS, mostrados no capítulo 3, é possível vislumbrar a combinação deles num único sistema (como o SISP), leva-nos a reconhecer a existência de módulos para compor a camada *Controller* do sistema e todos eles dependendo da camada *Model*.

Uma visão inicial, com diagrama de pacotes, é mostrada. Desse modo, é oferecida uma visão abstrata e rápida sobre a modelagem do *framework*, mostrando seus principais pacotes e a que camadas pertencem, na arquitetura MVC.

A camada *Model* é discutida em seguida. Discute-se como é feito o gerenciamento dos dados a serem usados pela interface gráfica e pelos módulos existentes para disponibilizar a lógica do sistema. Diagramas de classes são apresentados e discutidos, bem como comentários gerais sobre armazenamento de dados pelo sistema.

Os pacotes compondo a camada *Controller* são discutidos na seqüência. São apresentadas as classes e comentários de como se orientar na construção de um módulo dedicado à simulação de um sistema produtivo, como mostrado pelo elemento chamado simulador na figura 5.1. Apresentam-se modelagem de classes e comentários sobre como agregar trabalhos do tipo filtro de redução de seqüências para entrada de um FMS (ou simplesmente filtro, como mostrado na figura 5.1). Para encerrar a exposição da *Controller*, são apresentadas modelagens de classes e comentários sobre agregação de trabalhos do tipo análise de seqüências simuladas, elementos do tipo chamado analisador na figura 5.1.

Uma rápida exposição de como montar a camada *View* fazendo-se uso do *framework* é também oferecida.

Diagramas de classes apresentados nesse capítulo não refletem fielmente o uso de alguma linguagem de programação. São mostrados diagramas que usam tipos padrão da UML construídos com uso da ferramenta case *StarUML*. Por exemplo, um atributo *data*, de uma classe, que poderia ser do tipo *DateTime* na linguagem *C#*, é apresentado nos diagramas como do tipo *String*.

Detalhamentos aprofundados sobre conceitos de UML podem ser encontrados nas obras de Roman (2006) e de Booch *et. al.* (2005). Padrões de projeto usados podem ser aprofundados na obra de Gamma *et. al.* (2000) e foram apresentados brevemente em seções do capítulo 4.

5.1 Visão Geral

O *framework* é composto pelos pacotes mostrados na figura 5.2. São pacotes a conter classes a serem estendidas para construção de um sistema para auxílio do seqüenciamento de entrada de um sistema produtivo.

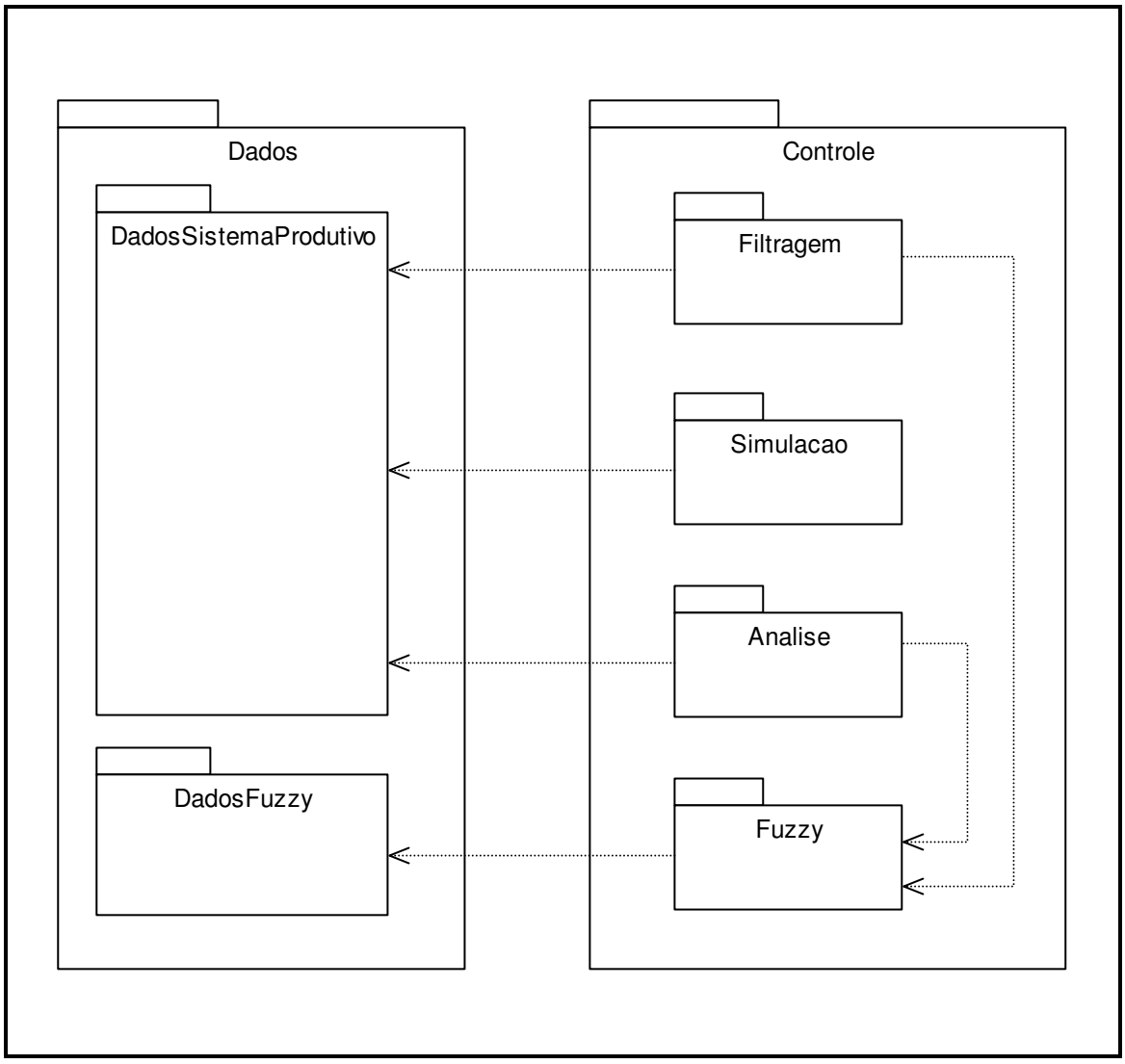


Figura 5.2: Diagrama de Pacotes do *framework*

O pacote *Dados* representa a camada *Model*. Outros dois pacotes são englobados por ele: *DadosSistemaProdutivo* e *DadosFuzzy*. O pacote *Dados* e os pacotes contidos dentro deles são responsáveis pelo gerenciamento de dados de uma aplicação que estende o *framework* proposto.

O pacote *Controle* representa a camada *Controller*. *Controle* contém os pacotes *Filtragem*, *Simulacao*, *Analise* e *Fuzzy*, cada um contendo classes destinadas a ser responsáveis pela lógica do sistema.

5.2 Gerenciamento dos Dados

Ao analisar o SADSP, o ANCSP e o ACS, verifica-se a presença de algum modelo de banco de dados, destinado a armazenar os dados requeridos para o auxílio pretendido ao seqüenciamento de entrada da produção.

No entanto, não existe uma uniformidade entre eles no que se refere ao modo de armazenamento de dados (uso de banco de dados, ou somente arquivos). Existe sim a idéia de um módulo destinado a armazenar/recuperar dados de domínio do sistema.

A proposta do *framework* para contemplar uma camada que gerencie os dados é apontar o uso de uma camada que funcione como interface com um banco de dados e/ou com algum gerenciador para armazenamento em arquivos. Essa camada deverá se destinar a gerenciar quaisquer dados que venham necessitar ser persistidos e/ou recuperados pelo sistema construído usando o *framework*.

A camada a representar o gerenciamento dos dados deve ser entendida como a representada pela camada *Model* na abordagem MVC. Um sistema construído a partir do *framework* mostrado no trabalho deve considerar a existência dessa camada e utilizar persistência/recuperação de dados através dela. Ela não deve ser entendida como unicamente persistindo dados num banco de dados. Alguns dados de configuração de um sistema final calcado no *framework* exposto no trabalho poderiam ser armazenados em arquivos XML e recuperados/modificados através dessa camada, por exemplo.

O pacote *Dados* representa a camada *Model*. Dentro do pacote *Dados*, especificam-se outros dois pacotes com classes que representam os dados principais a serem usados por um sistema a que o *framework* se dispõe, o pacote *DadosSistemaProdutivo* e o pacote *DadosFuzzy*.

5.2.1 Classes para Dados do Sistema Produtivo

Um sistema voltado para tratar o problema do seqüenciamento na entrada de um sistema produtivo necessita ter armazenado dados relacionados a ele, tais, como,

máquinas disponíveis, produtos existentes, roteiros de fabricação dos produtos, seqüências existentes. Uma coleção de dados é necessária.

Os trabalhos apresentados do grupo Tear apresentados no capítulo 3 não foram concebidos pensando em um único domínio no banco de dados, tanto semanticamente como em termos de definição de meio físico de armazenamento (arquivos ou bancos de dados). O SISP, no entanto, apresenta a modelagem de um banco de dados que suporte um sistema contendo o SADSP e o ACS.

Impor que quaisquer outros "módulos" a serem inclusos no sistema (seja eles do tipo simulador, filtro ou analisador de seqüências) estejam aptos a usarem um mesmo domínio de dados (banco de dados, por exemplo), sem necessidade de customização do mesmo, parece um tanto quanto arriscado na manutenção de um sistema a longo prazo.

Um novo trabalho, digamos, um novo tipo de filtro redutor de cenários, poderia propor, junto com a solução por ele apresentada, um novo domínio de dados de um sistema produtivo, como evolução do estágio atual de conhecimento do mesmo, ou mesmo, a inclusão de simplesmente uma nova tabela que seja uma modificação parcial de uma já existente.

A alternativa encontrada é orientar à construção de um pacote mínimo de classes que represente dados essenciais envolvidos num sistema produtivo para uma aplicação que agregue trabalhos como os discutidos do grupo Tear no capítulo 3. A alternativa encontrada é mostrada na figura 5.3, através do pacote *DadosSistemaProdutivo* e suas classes.

Num sistema usando o *framework* apresentado em que se agregue um novo trabalho com novos dados requeridos em relação ao ambiente de sistema produtivo que venham a conflitar com a modelagem de dados usada (estendida ou não do pacote mostrado na figura 5.3), ficam duas opções para o desenvolvedor: a primeira é criar um novo pacote a ser usado pelo trabalho agregado (seja ele do tipo filtro, simulador ou analisador); a segunda é adaptar o pacote com as classes de dados do sistema produtivo existente, devidamente refletindo as mudanças nos trabalhos agregados no sistema existente. A primeira opção fica como a mais viável, por ser mais flexível e não oferecer riscos de efeitos colaterais para trabalhos agregados e, eventualmente, em perfeito funcionamento. Mais interessante ainda, no entanto, é, ao desenvolver um sistema, considerar refinar da melhor maneira possível o pacote apresentado na figura 5.3, pois, abrangendo um domínio maior de informação e mais completo dificultaria a chegada de um momento em que se teria que tomar uma decisão com relação a remodelar a base de dados ou apenas estendê-la.

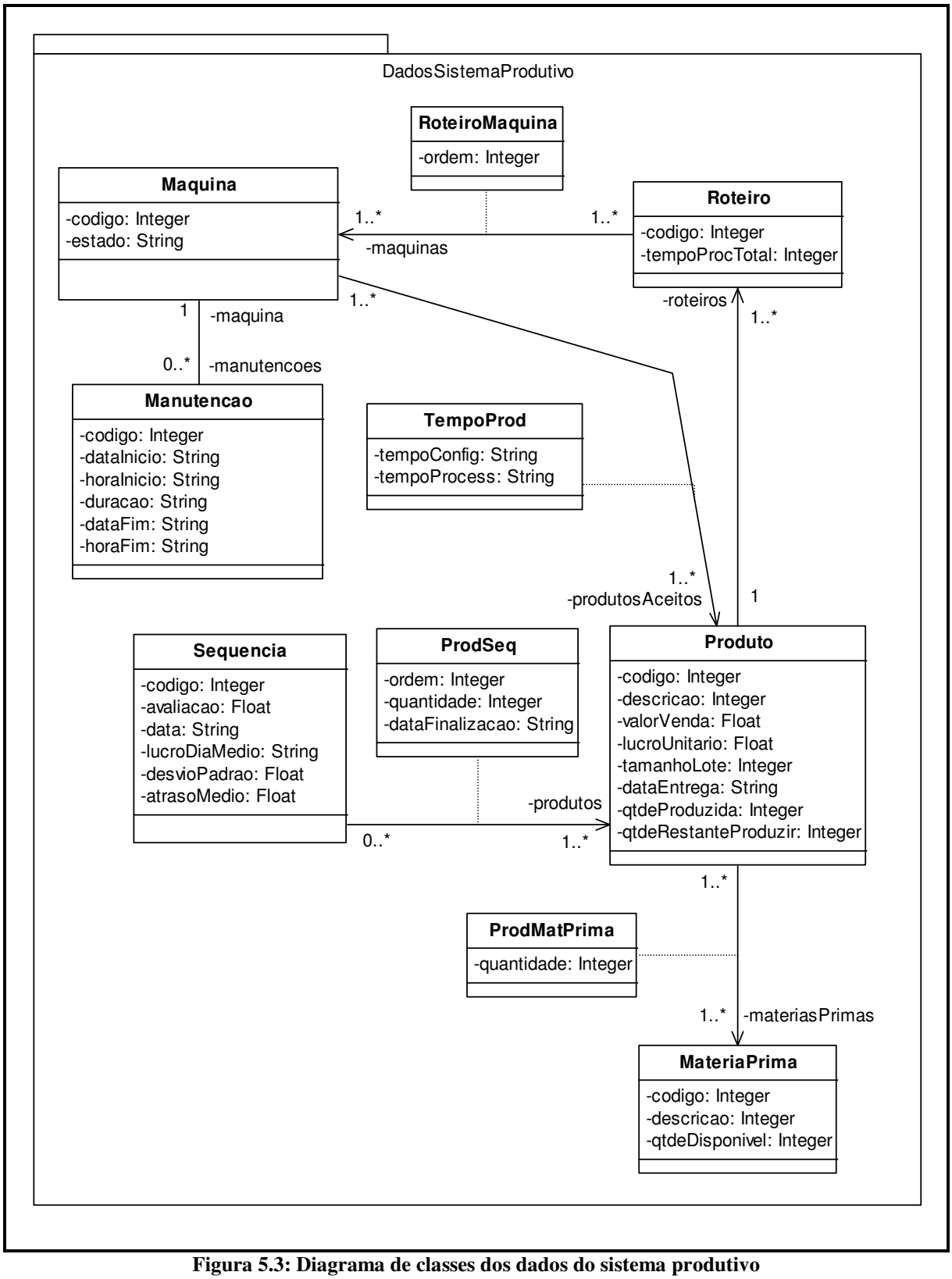


Figura 5.3: Diagrama de classes dos dados do sistema produtivo

O pacote de classes mostrado na figura 5.3 poderia expressar fielmente o banco de dados de um sistema que usa a proposta do trabalho. Ele é muito semelhante ao banco e

dados proposto por Roman (2006) para o SISP. No entanto, as classes devem ser entendidas como reflexo da modelagem usada na persistência dos dados, sendo a persistência feita através um arquivo XML ou um de um banco de dados, ou mesmo a mistura de uso de arquivos texto com arquivos XML e banco de dados. A decisão de como persistir os dados, de que banco de dados o sistema deve usar (e se usar), deve caber ao projeto do sistema que estende o *framework*.

No diagrama da figura 5.3 omitiram-se os métodos para acesso de leitura e escrita dos valores dos atributos das classes para facilitar a leitura. Todos os atributos possuem visibilidade privada, necessitando, portanto, cada um, de método de leitura e de método de escrita (ambos com visibilidade pública). As classes do pacote *DadosSistemaProdutivo* também não ficam encarregadas de saberem se instanciarem ou persistirem dados. Essa responsabilidade é discutida adiante. Segue-se, nessa seção, uma breve definição do significado de cada classe (e seus atributos) mostrada na figura 5.3. A tabela 5.1 mostra definição do significado das classes. A tabela 5.2 mostra definição do significado dos atributos das classes.

Tabela 5.1: Descrição das classes do pacote *DadosSistemaProdutivo*

Classe	***Descrição***
Maquina	Máquina existente no sistema produtivo
Manutencao	Atividade de manutenção de uma máquina
Produto	Produto processado pelo sistema produtivo
Roteiro	Um roteiro possível a ser seguido por um produto para seu processamento, indicando quais máquinas necessárias e em que ordem
RoteiroMaquina	Classe de associação definindo a ordem de uma máquina em um roteiro
TempoProd	Classe de associação definindo o tempo de configuração e o tempo de processamento de um produto em uma máquina
Sequencia	Seqüência de produtos podendo conter dados sobre uma eventual avaliação recebida com base em sua simulação no sistema produtivo
ProdSeq	Classe de associação definindo a ordem em que um produto se encontra numa seqüência e o tamanho do lote do produto existente na ordem dita

MateriaPrima	Matérias-primas para fabricação de produtos
ProdMatPrima	Classe de associação definindo a quantidade de matéria-prima para um produto

Tabela 5.2: Descrição dos atributos das classes do pacote *DadosFms*

Classe	***Atributo***	***Descrição***
Maquina	codigo	Discriminador de máquinas
	estado	Estado atual da máquina (“ociosa”, “parada”, “manutenção”, etc)
	manutencoes	Conjunto de manutenções associadas a uma máquina
Manutencao	codigo	Discriminador de manutenções
	dataInicio	Data prevista ou de real início da manutenção
	horaInicio	Hora prevista ou de real início da manutenção
	duracao	Tempo de duração da manutenção
	dataFim	Data prevista ou de real finalização da manutenção
	horaFim	Hora prevista ou de real finalização da manutenção
	maquina	Máquina associada à manutenção
Produto	codigo	Discriminador de um produto
	descricao	Descrição do produto
	valorVenda	Valor de venda de um produto
	lucroUnitario	Lucro unitário de um produto
	tamanhoLote	Quantidade de produtos em um lote do produto
	dataEntrega	Data de entrega prevista para um produto
	qtdeProduzida	Quantidade total do produto a ser produzida
	qtdeRestanteProduzir	Quantidade restante de um produto a ser produzida
	roteiros	Coleção de roteiros existentes para fabricação do produto
	materiasPrimas	Coleção de matérias-primas necessárias para fabricação do produto
Roteiro	codigo	Discriminador de um roteiro
	tempoProcTotal	Tempo de processamento total que um produto gasta seguindo-o
	maquinas	Coleção de máquinas presentes no roteiro

RoteiroMaquina	ordem	Ordem de uma máquina num roteiro
TempoProd	tempoConfig	Tempo de configuração de uma máquina para manufatura de um produto
	tempoProcess	Tempo de processamento de cada produto numa máquina
Sequencia	codigo	Discriminaor de uma seqüência
	avaliacao	Nota atribuída à seqüência após simulação e avaliação de um analisador
	data	Data na qual a seqüência foi gerada
	lucroDiaMedio	Lucro médio ao dia obtido ao se usar a seqüência.
	desvioPadrao	Desvio padrão de utilização das máquinas
	atrasoMedio	Atraso médio dos produtos da seqüência
	produtos	Coleção de produtos pertencentes à seqüência
ProdSeq	ordem	Ordem em que um produto está numa seqüência
	quantidade	Quantidade de um produto na seqüência
	dataFinalizacao	Data de finalização prevista pelo processo de simulação
MateriaPrima	codigo	Discriminador de matéria-prima
	descricao	Descrição da matéria-prima
	qtdeDisponivel	Quantidade disponível de matéria-prima
ProdMatPrima	quantidade	Quantidade de matéria-prima de um produto

5.2.2 Classes para Dados de um Sistema *Fuzzy*

O *framework* modelado no trabalho foi motivado por trabalhos do Grupo Tear, entre eles os que produziram o ANCSP e o ACS. Em ambos os sistemas isolados é possível notar que existe um tipo de camada, módulo, responsável apenas pela lógica *Fuzzy*. Em ambos os trabalhos assume-se o uso de apenas uma base de regras Mamdani, para cada um deles.

Considere um cenário em que se agregue num único sistema o ACS e algum outro futuro trabalho (sendo ele do tipo filtro ou analisador) que venha a usar a mesma base de regras apresentada pelo ACS. Uma maneira que facilite esse compartilhamento de dados é isolando a parte da lógica *Fuzzy* relacionada à definição das regras e conjuntos.

Outra vantagem ao se isolar a definição de regras e conjuntos é facilitar a modificação de um conjunto, função de pertinência. Necessitar-se-ia apenas modificar dados desejados.

A proposta é considerar que os dados de uma lógica *Fuzzy* sejam armazenados em meio físico e recuperados para as classes modeladas na figura 5.4, do pacote *DadosFuzzy*.

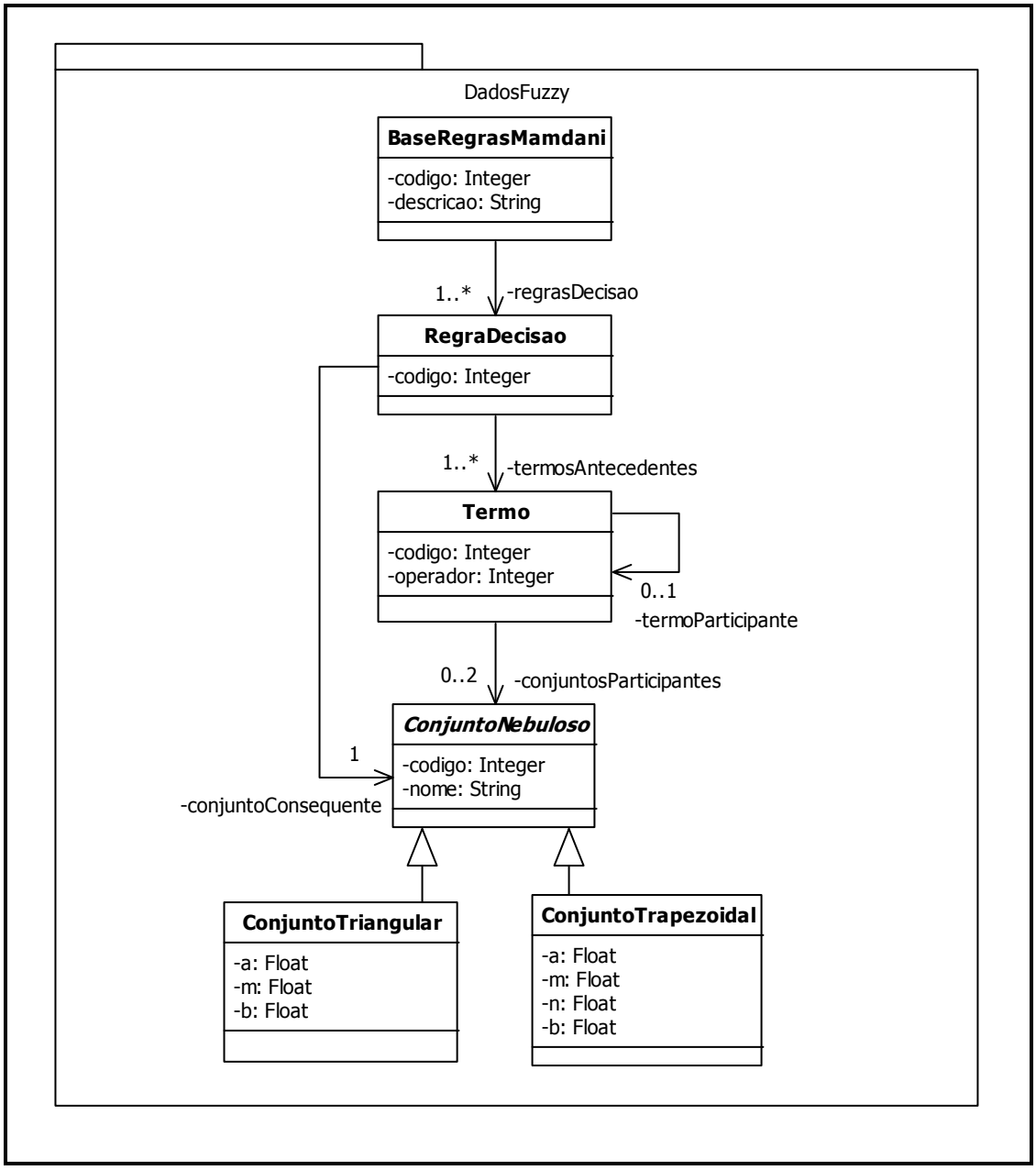


Figura 5.4: Diagrama de classes de um sistema *Fuzzy*

O pacote *DadosFuzzy* contempla apenas o básico das informações das regras *Fuzzy* encontrados no ANCSP e no ACS, podendo ser devidamente estendido enquanto usando o *framework* para construção de um sistema de auxílio ao seqüenciamento de entrada

de um sistema produtivo. São considerados os conjuntos nebulosos do tipo trapezoidal e triangular, mostrados, respectivamente, na figura 5.5 e figura 5.6. Apenas persistem-se informações relacionadas aos parâmetros de suas funções de pertinência.

$$A(x) = \begin{cases} 0 & \text{se } x < a \\ (x - a)/(m - a) & \text{se } x \in [a, m] \\ 1 & \text{se } x \in [m, n] \\ (b - x)/(b - n) & \text{se } x \in [n, b] \\ 0 & \text{se } x > b \end{cases}$$

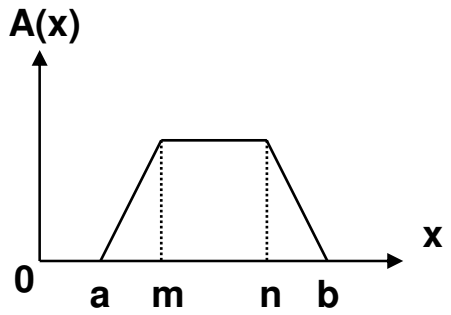


Figura 5.5: Conjunto Nebuloso Trapezoidal

$$A(x) = \begin{cases} 0 & \text{se } x < a \\ (x - a)/(m - a) & \text{se } x \in [a, m] \\ (b - x)/(b - m) & \text{se } x \in [m, b] \\ 0 & \text{se } x > b \end{cases}$$

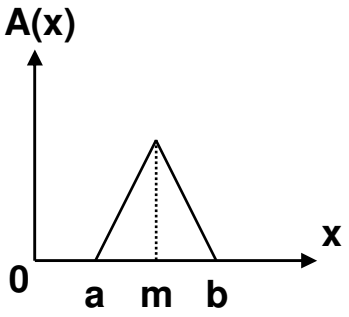


Figura 5.6: Conjunto Nebuloso Triangular

Segue-se, nessa seção, a descrição do significado das classes (tabela 5.3) e dos atributos das classes (tabela 5.4).

Tabela 5.3: Classes do pacote *DadosFuzzy*

Classe	***Descrição***
BaseRegrasMamdani	Base de regras para uso em inferência Mamdani
ConjuntoNebuloso	Classe abstrata representando um conjunto nebuloso
ConjuntoTrapezoidal	Um conjunto nebuloso trapezoidal
ConjuntoTriangular	Um conjunto nebuloso triangular
RegraDecisao	Uma regra de decisão
Termo	Um termo é o agrupamento de dois elementos através de um operador lógico (*and*, *or*), podendo um elemento ser um conjunto nebuloso ou um termo

Tabela 5.4: Descrição dos atributos das classes do pacote *DadosFuzzy*

Classe	***Atributo***	***Descrição***
BaseRegrasMamdani	codigo	Discriminador de base de regras Mamdani
	descricao	Descrição do conjunto de regras Mamdani
	regrasDecisao	Conjunto de regras de decisão
ConjuntoNebuloso	codigo	Discriminador de conjunto nebuloso
	nome	Nome do conjunto nebuloso
ConjuntoTrapezoidal	a	Variável *a* definida na figura 5.5
	m	Variável *m* definida na figura 5.5
	n	Variável *n* definida na figura 5.5
	b	Variável *b* definida na figura 5.5
ConjuntoTriangular	a	Variável *a* definida na figura 5.6
	m	Variável *m* definida na figura 5.6
	b	Variável *b* definida na figura 5.6
RegraDecisao	codigo	Discriminador de regra de decisão
	conjuntoConsequente	Conjunto nebuloso que é o conseqüente na regra de decisão
	termosAntecedentes	Conjunto de termos definindo relações lógicas entre os conjuntos nebulosos antecedentes
Termo	codigo	Discriminador do termo

	operador	Operador lógico ligando dois elementos
	termoParticipante	Um termo (no máximo) participando para compor um termo. No caso de haver 2 elementos participantes do tipo conjunto nebuloso, não deverá haver um valor para termoParticipante.
	conjuntosParticipantes	Conjunto(s) participando da formação de um termo.

5.2.3 Classes para Gerenciamento de Dados

As classes representando dados dos pacotes *DadosSistemaProdutivo* e *DadosFuzzy* não possuem a capacidade de persistirem ou recuperarem informações do sistema de armazenamento de dados do sistema computacional. Essa é a idéia, ter essas classes como entidades que representem os dados que são armazenados ou deverão ser armazenadas.

A mesma idéia deverá também ser aplicada para classes que venham a representar algum dado de configuração do sistema. Por exemplo: uma classe que indique os nomes dos filtros, simuladores e analisadores que o sistema deverá mostrar ao usuário. Uma classe como a exemplificada não terá inteligência para persistir, recuperar dados.

É proposta a existência um pacote *Dados*, englobando todos os pacotes semelhantes aos mostrados em 5.2.1 e 5.2.2, assim como também classes dedicadas a recuperar e/ou persistir os dados do sistema. Na figura 5.7 está a modelagem do pacote *Dados*.

Classes que venham a gerenciar um domínio de dados devem ser derivadas da *GerenciaDados*, mostrada na figura 5.7. *GerenciaDados* é uma classe abstrata definindo um método para recuperação dos dados que ela toma conta, e um outro método para persistência dos dados. Eles são, respectivamente, *recuperarTodosDados* e *salvarTodosDados*.

No diagrama da figura 5.7 pode-se ver a classe *GerenciaDadosSistemaProdutivo*. Subclasse de *GerenciaDados*, ela é responsável por gerenciar os dados do sistema produtivo, tendo todos eles como atributos. O acesso a eles ou inclusão deles para armazenamento em meio não-volátil deve ser feito através de *GerenciaDados*. Os métodos de acesso a eles não são mostrados no diagrama, apenas métodos básicos para exclusão e inclusão.

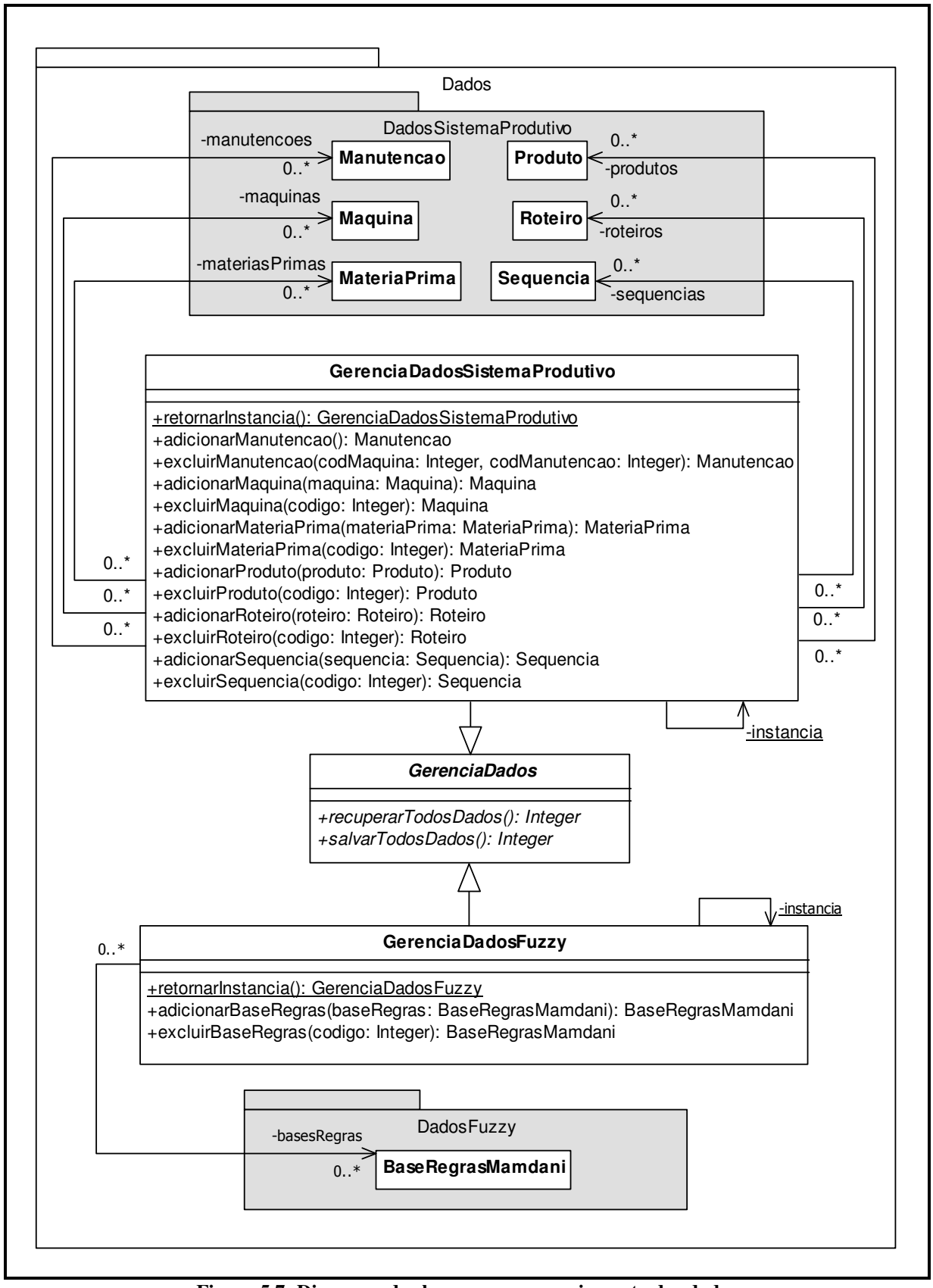


Figura 5.7: Diagrama de classes para gerenciamento dos dados

É proposto que o sistema tenha apenas uma instância de *GerenciaDadosSistemaProdutivo*. O uso do padrão de projeto *Singleton* é empregado para tal.

Para se obter uma instância de *GerenciaDadosSistemaProdutivo*, deve-se fazer uma chamada ao método estático *retornarInstancia*, que assegura retornar uma mesma instância para todos os clientes que venham chamá-lo.

No diagrama da figura 5.7 pode-se notar também a classe *GerenciaDadosFuzzy*, responsável por gerenciar os dados relacionados a lógica *Fuzzy*. *GerenciaDadosFuzzy* possui como atributo uma coleção de instâncias da classe *BaseRegrasMamdani* (pacote *DadosFuzzy*), através da qual navega pelas regras e conjuntos envolvidos. A classe é responsável por recuperar e persistir instâncias de *BaseRegrasMamdani*. Métodos para acesso ao atributo *baseRegras* não são mostrados no diagrama da figura 5.7.

Também é proposto que o sistema tenha apenas uma instância de *GerenciaDadosFuzzy*, usando-se para isso o padrão de projeto *Singleton*. Uma instância de *GerenciaDadosFuzzy* pode ser obtida por uma chamada ao método estático *retornarInstancia*, de forma semelhante ao que acontece à classe *GerenciaDados*.

No geral recomenda-se que classes derivadas de *GerenciaDados* também implementem o padrão de projeto *Singleton*, como as subclasses de *GerenciaDados* mostradas no diagrama da figura 5.7.

5.3 Filtro de Seqüências para Simulação

A seção 3.1 do capítulo 3 apresentou o SADSP. A seção 3.2 do capítulo 3 apresentou o ANCSP. Cada um desses trabalhos mostra um redutor do número de seqüências de entrada num sistema produtivo para posterior simulação, após a qual objetiva-se encontrar a melhor seqüência possível para entrada no sistema produtivo simulado.

O SADSP e o ANCSP demonstram o conceito de filtro de seqüências de entrada de um sistema produtivo a serem usadas em simulação. No entanto, esses mesmos trabalhos poderiam ser considerados usados isoladamente do uso de simulação. Um sistema inteligente para seqüenciamento da entrada de um sistema produtivo poderia ter várias opções de filtros e de simulação, mas oferecendo a opção do usuário descartar o uso de simulação e assumir como resultado de ajuda na decisão de que seqüência adotar o gerado por um filtro de seqüências.

Roman (2006) aponta a possibilidade do sistema proposto por ele não usar simulação, ficando restrito ao filtro de seqüências. Entretanto não é apontado que poderia haver várias opções de diferentes tipos de filtros, para uso pelo usuário.

O *framework* proposto procura dar essa flexibilidade a um sistema construído a partir dele, de poderem existir diferentes tipos de filtros no sistema. Considera-se que os filtros ficam definidos internamente a um pacote chamado *Filtragem*. O pacote *Filtragem* é considerado como sendo parte da camada *Controller*, conforme a visão MVC. Conhecimentos de persistência e recuperação de dados do meio físico ficam delegados para classes do pacote *Dados*. Na figura 5.8 são mostradas as principais classes que fazem parte do pacote *Filtragem* e são apontadas pelo *framework* proposto. Ao usar o *framework* para construção de um sistema, classes concretas e derivadas das classes apresentadas no pacote *Filtragem* representam filtros implementados.

As classes mostradas no pacote *Filtragem* são discutidas na seção 5.3.1. O pacote *Fuzzy*, também mostrado na figura 5.8 é discutido na seção 5.3.2.

5.3.1 Pacote *Filtragem*

O pacote *Filtragem*, como proposto pelo *framework* e mostrado na figura 5.8, é composto pelas classes *CriadorDeFiltro*, *Filtro* e *FiltroFuzzy*. A escolha da estrutura apresentada deve-se por buscar uma superclasse que representassem os filtros de seqüências e uma forma fácil de se ter um filtro específico criado, não necessitando conhecer exatamente a subclasse envolvida definindo-o.

A classe *CriadorDeFiltro* possui o comportamento da combinação dos padrões de projeto *Singleton* e *FactoryMethod*. Ela é responsável por prover instâncias de classes concretas que representem um filtro (comportamento de um *FactoryMethod*).

O cliente que usa a classe *CriadorDeFiltro* necessita apenas informar um número que deve estar associado a um filtro para obter a instância da subclasse específica ao filtro desejado, através de chamada ao método *criarFiltro*.

Apenas uma instância de *CriadorDeFiltros* é necessária no sistema (comportamento de um *Singleton*). Uma instância de *CriadorDeFiltros* pode ser obtida por uma chamada ao seu método *retornarInstancia*.

A classe abstrata *Filtro* representa um filtro de seqüências para entrada de um sistema produtivo. É definido o método abstrato *filtrarSequencias*, que estabelece uma

assinatura comum de método a ser implementado pelas classes concretas descendentes de *Filtro* para realizar a operação de redução de seqüências na entrada do sistema produtivo.

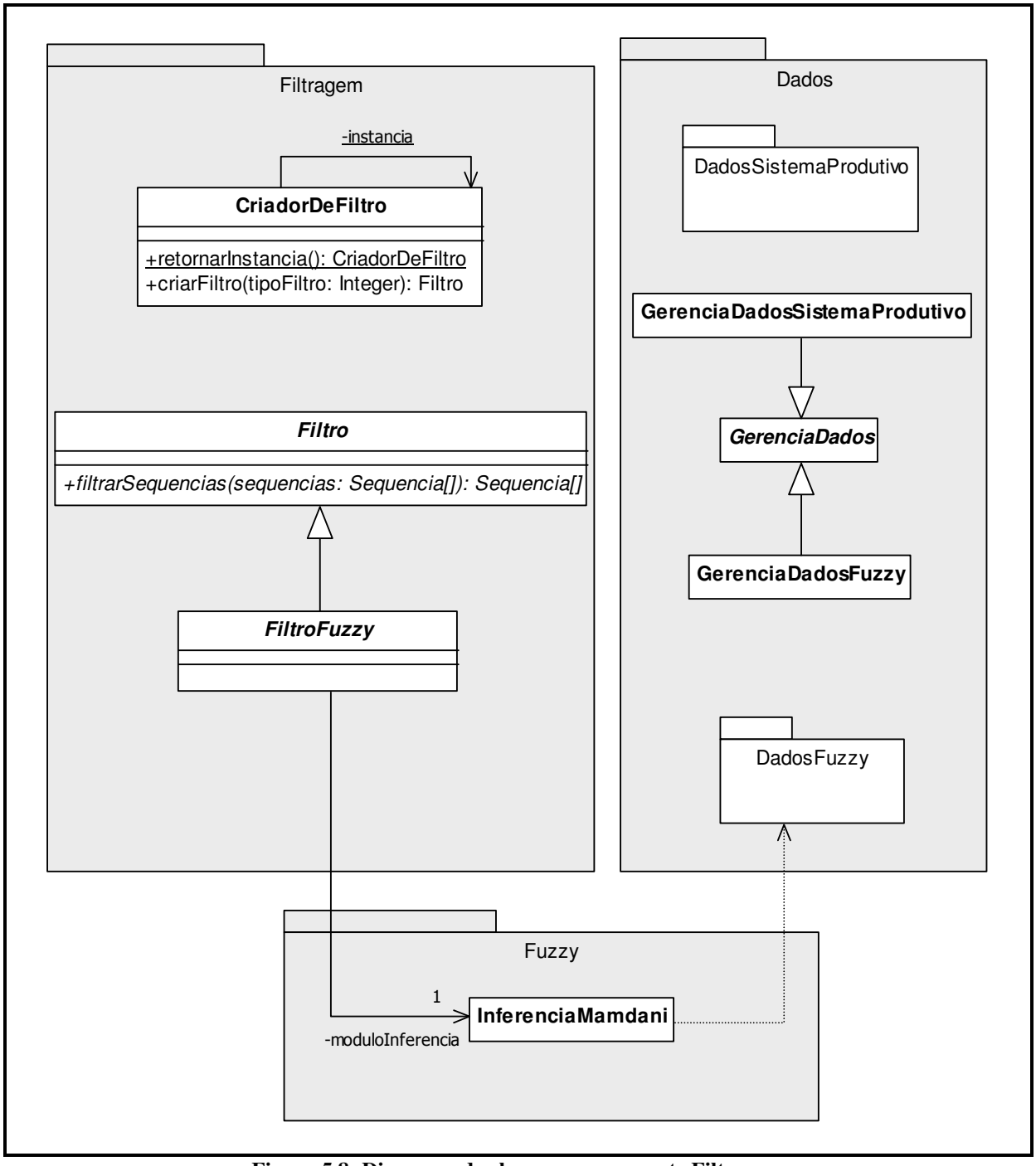


Figura 5.8: Diagrama de classes para o pacote Filtragem

Motivado pelo trabalho de Silva (2005), definiu-se uma subclasse abstrata de *Filtro*, a *FiltroFuzzy*. Silva (2005) faz uso de lógica *Fuzzy* para reduzir o número de seqüências da entrada de um FMS a usar na simulação. O método de inferência de *Mamdani* é

usado. Silva (2005) também traz discussão aprofundada sobre lógica *Fuzzy* e método de inferência *Mamdani*.

FiltroFuzzy deve ser usada como superclasse de filtros que usem de algum modo lógica *Fuzzy* para redução das seqüências. *FiltroFuzzy* possui o atributo *moduloInferencia*, instância da classe *InferenciaMamdani*, do pacote *Fuzzy*.

5.3.2 Pacote *Fuzzy*

O uso de lógica *Fuzzy* aparece em dois trabalhos que serviram como motivação ao *framework* proposto. Decidiu-se por considerar um pacote, também nomeado como *Fuzzy*, como parte da camada *Controller* quando analisando um sistema pela abordagem de arquitetura MVC. O pacote também faz parte do *framework*, tendo dependência com o pacote *DadosFuzzy*.

O pacote *Fuzzy* e suas classes são mostrados na figura 5.9. Apesar de apresentado dentro da seção dedicada ao pacote *Filtragem*, o pacote *Fuzzy* não deve ser considerado como de uso exclusivo de filtros. Pode ser usado por outros pacotes envolvidos na camada *Controller* do sistema (na visão MVC) ou mesmo os apresentados pelo *framework*.

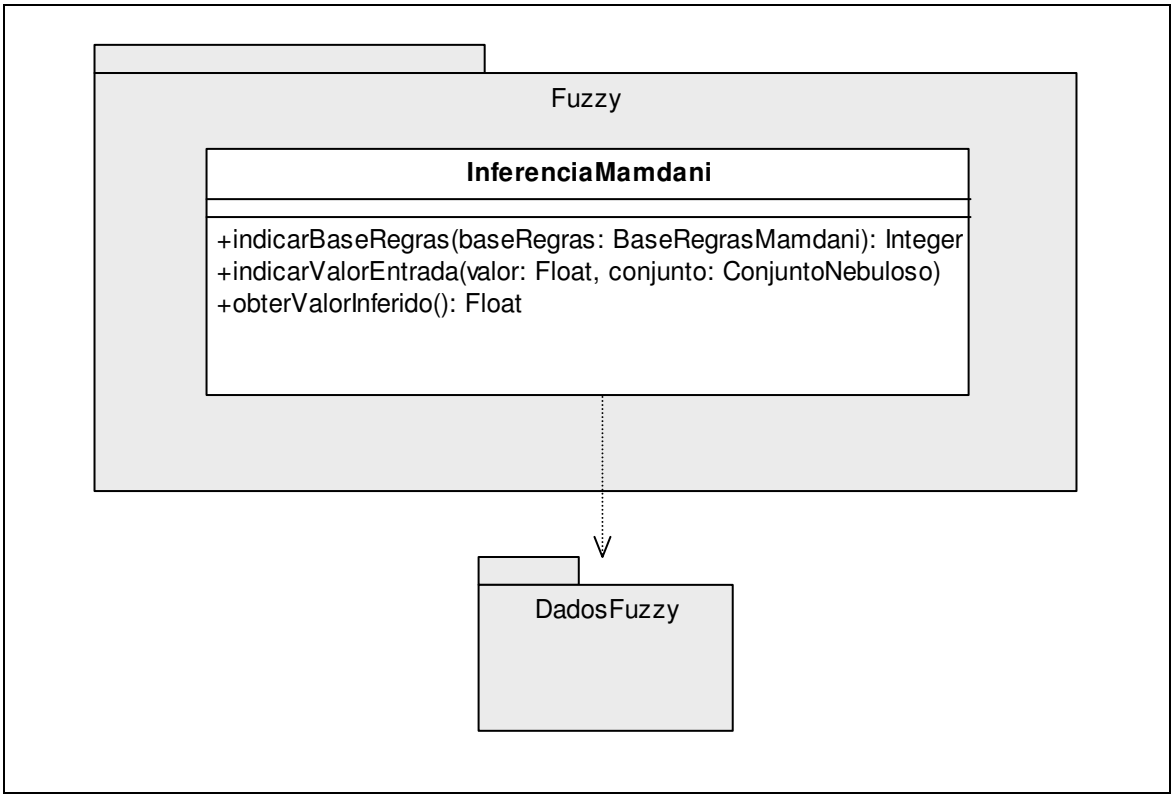


Figura 5.9: Pacote *Fuzzy*

A classe *InferenciaMamdani* é responsável por todos cálculos envolvidos na operação de inferência. O método *indicarBaseRegras* permite ao cliente usuário dela indicar uma base de regras existente. O método *indicarValorEntrada* deve ser chamado para indicar o valor escalar de uma variável para um conjunto nebuloso de entrada na base de regras escolhida. O método *obterValorInferido* deverá ser chamado, após definição da base de regras e indicação dos valores escalares de entrada, para a inferência, obtendo um valor escalar.

A forma como a inferência se dará, se por código numa *dll* a ser chamada pelo sistema ou somente por escrita de código do próprio sistema no corpo dos métodos de *InferenciaMamdani* fica a critério do desenvolvedor do sistema. A vantagem do uso da classe *InferenciaMamdani* para cálculos *fuzzy* é permitir que se abstraia como e onde os cálculos são realizados, facilitando manutenção de código de um sistema.

5.4 Interface com Simulador

Para os trabalhos do grupo Tear discutidos no capítulo 3, a simulação é realizada por um modelo de FMS implementado no software *Automod*. Interações com o *Automod* são realizadas diretamente pelos módulos definidos nos sistemas construídos por esses trabalhos. Não existe uma interface de acesso que fica responsável por iniciar a simulação, informar os dados requeridos para iniciá-la e extrair as informações disponibilizadas após a simulação.

Ao ter uma interface que abstraia como de fato acontece a interação com o simulador, tem-se a vantagem de impactar menos um sistema dependente do simulador num cenário envolvendo mudanças no simulador. Num cenário hipotético, como esse, ter-se-ia apenas que dar manutenção no código da interface com o simulador.

Ao desenvolver e eventualmente customizar um sistema para seqüenciamento da entrada de uma fábrica, o cliente pode requerer opções de simuladores ou mesmo exigir a troca de um simulador por outro. Sem uma interface abstraindo o contato com o simulador tem-se um desenvolvimento mais custoso, requer maiores modificações no sistema como um todo. Com uma interface, facilita-se, pois todos módulos usuários do simulador desconhecem qual simulador eles estão lidando.

Para realizar a interface com o Simulador, o *framework* contém o pacote *Simulacao*, apresentado na figura 5.10. O pacote *Simulacao* deve ser entendido como parte da camada *Controller*, numa visão de arquitetura MVC.

5.4.1 Pacote *Simulacao*

O pacote *Simulacao*, mostrado na figura 5.10, é composto pelas classes *CriadorDeSimulador* e *Simulador*. A escolha da estrutura apresentada deveu-se a buscar uma superclasse que representasse interface com simulador agregado a um sistema de seqüenciamento de entrada de uma fábrica. Também se buscou uma forma fácil de se ter uma interface com simulador específica criada, não necessitando conhecer exatamente a subclasse envolvida defininda.

A classe *CriadorDeSimulador* possui o comportamento da combinação dos padrões de projeto *Singleton* e *FactoryMethod*. Esse tipo de combinação foi apresentado na seção 4.4.3. Ela é responsável por prover instâncias de classes concretas que representem um filtro (comportamento de um *FactoryMethod*).

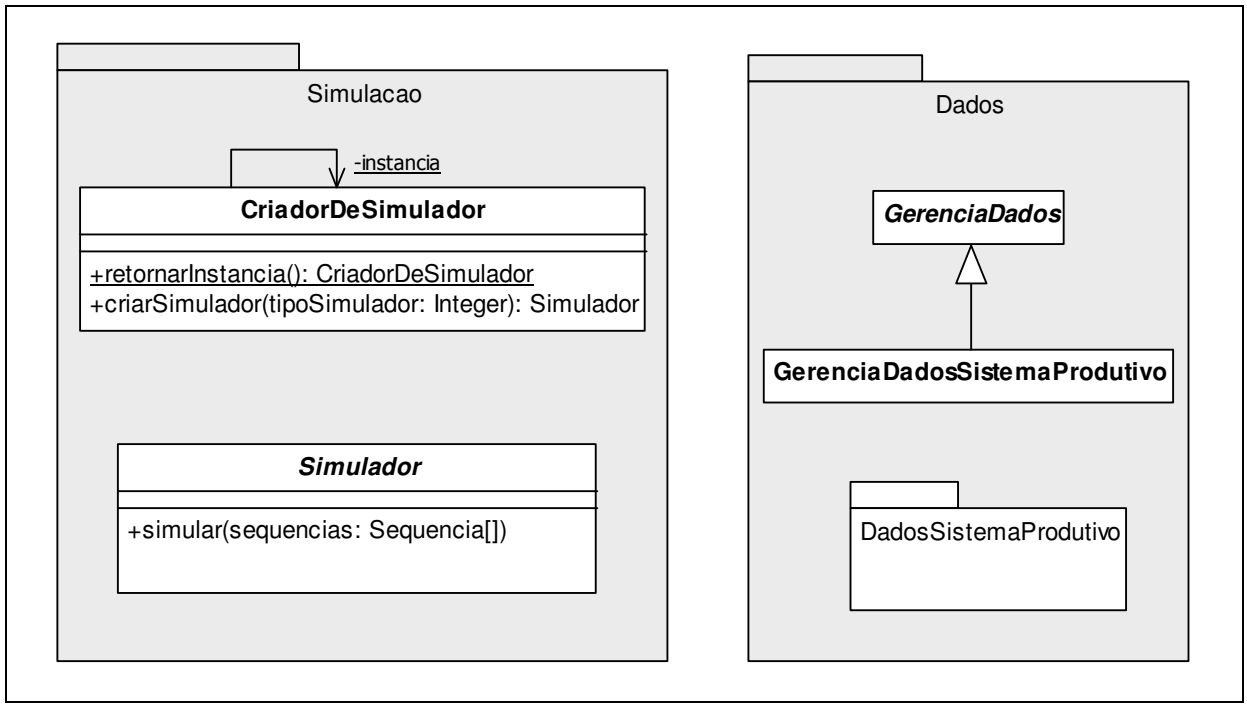


Figura 5.10: Pacote *Simulacao*

O cliente que usa a classe *CriadorDeSimulador* necessita apenas informar um número que deve estar associado a um filtro para obter a instância da subclasse específica ao filtro desejado, através de chamada ao método *CriarFiltro*.

Apenas uma instância de *CriadorDeSimulador* é necessária no sistema (comportamento de um *Singleton*). Uma instância de *CriadorDeSimulador* pode ser obtida por uma chamada ao seu método *retornarInstancia*.

A classe abstrata *Simulador* representa uma interface com simulador de um FMS. É definido o método abstrato *simular*, que estabelece uma assinatura comum de método a ser implementado pelas classes concretas descendentes de *Simulador* para realizar a operação de simulação de seqüências indicadas.

Uma classe concreta descendente de *Simulador* deve seguir o comportamento de persistir/atualizar dados da simulação do FMS através do pacote *Dados*. A coleção de seqüências recebidas como parâmetro no método *simular* (da classe Simulador) é um conjunto de instâncias da classe *Sequencia*, do pacote *DadosSistemaProdutivo*.

5.5 Analisador de Seqüências Simuladas

A seção 3.3 do capítulo 3 apresentou o ACS. É um trabalho que mostra um analisador de seqüências de entrada num sistema produtivo após o uso de simulação. A execução do sistema proposto por Fernandes (2004) atribui notas às seqüências simuladas.

Trabalhos da natureza do ACS podem ser implementados futuramente e agregados a um sistema inteligente de seqüenciamento de entrada da produção. Um sistema inteligente para seqüenciamento da entrada de um sistema produtivo poderia ter várias opções de analisadores de seqüências simuladas.

O *framework* proposto procura dar essa flexibilidade a um sistema construído a partir dele, de poderem existir diferentes tipos de analisadores no sistema. Considera-se que os analisadores ficam definidos internamente a um pacote chamado *Analise*. O pacote *Analise* é considerado como sendo parte da camada *Controller*, conforme a visão MVC. Conhecimentos de persistência e recuperação de dados do meio físico ficam delegados para classes do pacote *Dados*. Na figura 5.11 são mostradas as principais classes que fazem parte do pacote *Analise* e são apontadas pelo *framework* proposto. Ao usar o *framework* para construção de um sistema, classes concretas e derivadas das classes apresentadas no pacote *Analise* representam analisadores implementados.

5.5.1 Pacote *Analise*

O pacote *Analise*, como proposto pelo *framework* e mostrado na figura 5.11, é composto pelas classes *CriadorDeAnalisador*, *Analisador* e *AnalisadorFuzzy*. A escolha da

estrutura apresentada deve-se por buscar uma superclasse que representasse os analisadores de seqüências e uma forma fácil de se ter um analisador específico criado, não necessitando conhecer exatamente a subclasse envolvida definindo-o.

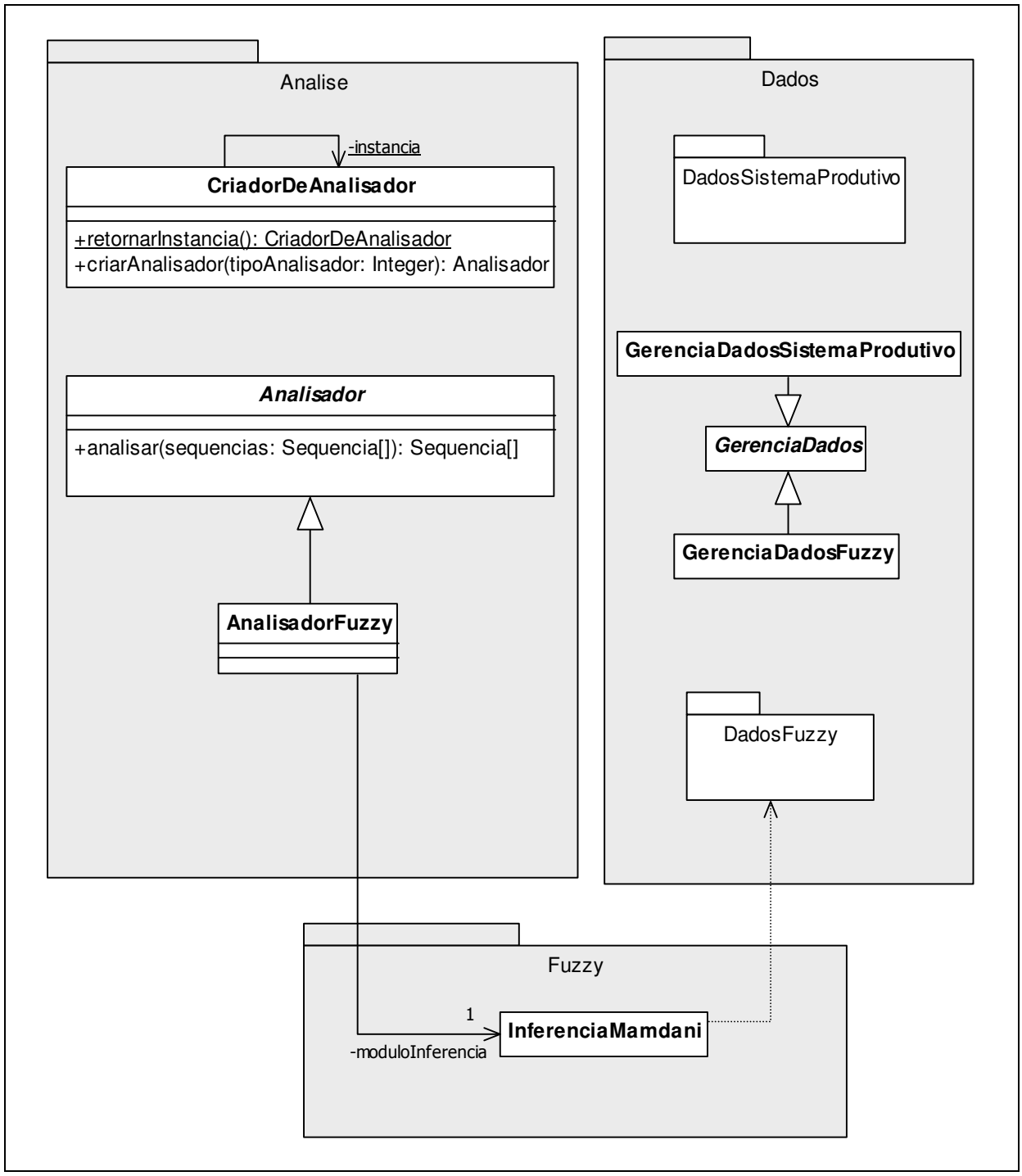


Figura 5.11: Diagrama de Classes para o Pacote *Analise*

A classe *CriadorDeAnalisador* possui o comportamento da combinação dos padrões de projeto *Singleton* e *FactoryMethod*. Esse tipo de combinação foi apresentado na seção 4.4.3. Ela é responsável por prover instâncias de classes concretas que representem um analisador (comportamento de um *FactoryMethod*).

O cliente que usa a classe *CriadorDeAnalisador* necessita apenas informar um número que deve estar associado a um analisador para obter a instância da subclasse específica ao filtro desejado, através de chamada ao método *criarAnalisador*.

Apenas uma instância de *CriadorDeAnalisador* é necessária no sistema (comportamento de um *Singleton*). Uma instância de *CriadorDeAnalisador* pode ser obtida por uma chamada ao seu método *retornarInstancia*.

A classe abstrata *Analisador* representa um analisador de seqüências simuladas para entrada de um FMS. É definido o método abstrato *analisar*, que estabelece uma assinatura comum de método a ser implementado pelas classes concretas descendentes de *Analisador* para realizar a operação de redução de seqüências na entrada do sistema produtivo.

Motivado pelo trabalho de Fernandes (2004), definiu-se uma subclasse abstrata de *Filtro*, a *AnalisadorFuzzy*. Fernandes (2004) faz uso de lógica *Fuzzy* para avaliar as seqüências simuladas. O método de inferência de *Mamdani* é usado. Fernandes (2004) também traz discussão aprofundada sobre lógica *Fuzzy* e método de inferência *Mamdani*.

AnalisadorFuzzy deve ser usada como superclasse de analisadores que usem de algum modo lógica *Fuzzy* para avaliação das seqüências. *AnalisadorFuzzy* possui o atributo *moduloInferencia*, instância da classe *InferenciaMamdani*, do pacote *Fuzzy*.

5.6 Considerações Finais

O capítulo apresenta, no decorrer das suas seções, diagramas de classes que compõem um *framework* para um sistema inteligente de seqüenciamento de entrada de um sistema produtivo, conforme o contexto de execução apresentado na figura 5.1.

O *framework* apresentado serve como guia para construção, tentando trazer o máximo de flexibilidade possível para sua construção e a melhor organização possível de pacotes de classes, agrupando-as conforme seus significados, papéis.

O *framework* preenche as camadas *Controller* e *Model* de um sistema com arquitetura MVC. A camada *Controller* possui módulos representados pelos pacotes *Filtragem*, *Fuzzy*, *Simulacao* e *Analise*. Não é mostrada comunicação direta entre os módulos

Filtragem, *Simulacao* e *Analise*, objetivando deixar um sistema construído a partir do *framework* flexível para definição (dinâmica ou não) de que módulos usar, podendo mesmo ter mais de um tipo de filtro disponível, ou simulador ou analisador.

As chamadas para exercício de filtragem, simulação ou análise podem partir diretamente de uma camada *View* implementada para um sistema desenvolvido a partir do *framework*. Todo o controle, lógica, inteligência do sistema está na camada *Controller*.

Embora não especificada uma camada *View* no *framework*, recomenda-se o uso do pacote *Dados* do mesmo (estendendo-o devidamente) para armazenar configurações gerais do sistema, tais como, por exemplo, preferências de uso de tipos de filtros do usuário.

Um exemplo de pequeno pseudocódigo da camada *View* de um sistema computacional construído a partir do *framework* apresentado é mostrado na figura 5.12. O objetivo da figura é mostrar como usar o *framework*, sem se preocupar com transcrição total de um código real do sistema. O trecho de pseudocódigo mostrado na figura 5.12 é o básico necessário para realizar todos os passos mostrados na figura 5.1.

```
// 1 -> obter seqüências indicadas pelo filtro do tipo A (fictício)

// Uso de filtro do tipo A (supondo subclasse construída)
// Supondo constante inteira FILTRO_A

CriadorDeFiltro criaFiltro = CriadorDeFiltro.retornarInstancia();
Filtro filtroDoTipoA = criaFiltro.criarFiltro(FILTRO_A);

// Sequencia é classe existente do pacote Dados, do framework
Sequencia[] sequenciasFiltradasPorA =
    filtroDoTipoA.filtrarSequenciasPossiveis(todasSequenciasPossiveis);


// 2 -> simular seqüências filtradas pelo Automod (ficticio)

// Uso de simulador do tipo Automod (supondo subclasse construída)
// Supondo constante inteira SIMULADOR_AUTOMOD

CriadordeSimulador criaSim = CriadorDeSimulador.retornarInstancia();
Simulador automod = criaSim.criarSimulador(SIMULADOR_AUTOMOD);

Sequencia[] sequenciasSimulacao = sequenciasFiltradasPorA;
automod.simular(sequenciasSimulacao);


// 3 -> realizar análise seqüências simuladas pelo analisador Alfa (fictício)

// Uso de analisador do tipo Alfa (supondo subclasse construída)
// Supondo constante inteira ANALISADOR_ALFA

CriadorDeAnalisador criaAnalisador = CriadorDeAnalisador.retornarInstancia();
Analisador anlisadorAlfa = criaAnalisador.criarAnalisador(ANALISADOR_ALFA);

Sequencia seqAnalisadas = analisadorAlfa.analisar(sequenciasSimulacao);
```

Figura 5.12: Pseudocódigo da camada *View* fazendo uso do *framework* proposto

Capítulo 6 - Conclusão

O trabalho propõe a criação de um *framework* para um sistema inteligente de auxílio no seqüenciamento dos produtos na entrada de um sistema produtivo. O sistema imaginado estava no contexto da combinação de trabalhos realizados pelo Grupo Tear para auxílio do problema de seqüenciamento de entrada de um FMS.

Um estudo foi levantado para compreensão do problema, pesquisando-se trabalhos na área de seqüenciamento de produtos na entrada de sistemas produtivos. Um aprofundamento nos trabalhos do Grupo Tear que motivaram o trabalho foi realizado e apontado no capítulo 3.

Os trabalhos que motivaram a obra apontam conceitos que podem se fundir num único sistema. Esses conceitos, com uso de estratégias de modelagem discutidas no capítulo 4, foram usados para a criação do *framework*.

O *framework* foi modelado no capítulo 5. Apontamentos e diagramas de classes foram escritos como guia para construção de um sistema inteligente de auxílio no seqüenciamento dos produtos na entrada de um sistema produtivo. Houve grande preocupação em como facilitar futuras customizações requeridas por um sistema construído a partir dele. Inclusões de novos trabalhos do grupo Tear no contexto apresentado pelo *framework*, assim como construção de um sistema unificando trabalhos discutidos no capítulo 3, são possíveis.

A criação do *framework* apresentou a modelagem das classes a compô-lo e serem estendidas para implementação do sistema. Conforme definição do conceito de *framework* feita por Gamma *et. al.* (2000), apresentou-se uma modelagem que definisse a estrutura geral do sistema, as responsabilidades-chave das classes de objetos, como estas colaboram.

Ao considerar uma arquitetura do tipo MVC, o *framework* proposto indica classes e pacotes compondo as camadas *Model* e *Controller*. A camada *View*, ao ser construída no sistema inteligente, deve fazer uso de classes das camadas *Model* e *Controller*.

O *framework* não foi aplicado na construção de um sistema. O uso dele na construção de um sistema poderia trazer considerável evolução do mesmo. No processo de implementação de um sistema poder-se-ia encontrar outros elementos a integrar o *framework*. Por outro lado, por ter o resultado do estudo em cima de como é o contexto e atuação para um sistema inteligente de seqüenciamento de entrada de produtos num sistema produtivo, o *framework* apresentado pelo trabalho mostra-se de grande importância como guia de construção de um sistema assim.

Um trabalho futuro é construir um sistema a partir do *framework* modelado, totalmente configurável, no contexto de seqüenciamento de entrada de FMS discutido na proposta do trabalho.

O estudo de novos trabalhos que se insiram nos conceitos de filtros, simuladores e analisadores, mostrados no capítulo 3, para extensão do *framework* é um futuro trabalho a existir como apontamento do aqui apresentado.

Referências

Ammi, I.; Benbouzid-Sitayeb, F.; Varnier, C.; Zerhouni, N.; *Applying Ant Colony Optimization for the Joint Production and Preventive Maintenance Scheduling Problem in the Flowshop Sequencing Problem*, 3rd International Conference on Information and Communication Technologies: From Theory to Applications, ICTTA, 2008.

Au, K.C.; Chan, F. T. S.; Chan, P.L.Y.; *A Decision Support System for Production Scheduling in an Ion Plating Cell*, Expert Systems with Applications, v. 30, n. 4, p 727-738, maio 2006.

Bae, J. W.; Kim, K. H.; Lee, Hyun Yong; Song, J. Y.; *Distributed Scheduling and Shop Floor Control Method*, Computers & Industrial Engineering, v. 31, n. 3-4, p. 583-586, dezembro 2007.

Bard, J. F.; Monkman, S. K.; Morrice, D. J.; *A Production Scheduling Heuristic for an Electronics Manufacturer with Sequence-Dependent Setup Costs*, European Journal of Operational Research, v. 187, n. 3, p 1100-1114, junho 2008.

Barnes, S.; Cheeseman, M. J.; Hesketh, G. B.; Swann, P.; *Adaptive Manufacturing Scheduling: a Flexible and Configurable Agent-Based Prototype*, Production Planning and Control, v. 16, n. 5, p 479-487, julho 2005.

Binder, F. V.; Sistemas de Apoio à Decisão. Editora Érica Ltda, 1999.

Booch, G.; Jacobson, I.; Rumbaugh, J.; UML Guia do Usuário, Rio de Janeiro: Elsevier, 2005. Tradução de: Fábio Freitas da Silva e Cristina de Amorim Machado.

Boqin, F.; Haichang, G.; Li, Z.; *An Improved Genetic Algorithm for Flow Shop Sequencing*, Proceedings of 2005 International Conference on Neural Networks and Brain Proceedings, v. 1, 2005.

Boschi, E.; Rossi, A.; *A Hybrid Heuristic To Solve the Parallel Machines Job-Shop Scheduling Problem*, Advances in Engineering Software, v. 40, n. 2, p 118-127, fevereiro 2009.

Buyurgan, N.; Saygin, Can; *An Integrated Control Framework for Flexible Manufacturing Systems*, International Journal of Advanced Manufacturing Technology, v. 27, n. 11-12, p 1248-1259, fevereiro 2006.

Caprihan, R.; Gusikhin, O.; Stecke, K. E.; *Least In-Sequence Probability Heuristic for Mixed-Volume Production Lines*, International Journal of Production Research, v. 46, p 647-673, fevereiro 2008.

Carvalho, V. O.; *Um Modelo de Seqüenciamento da Produção para um Sistema de Apoio à Decisão*, Dissertação de Mestrado, DC/UFSCar, São Carlos, 2003.

Chan, F. T. S.; Chan, H. K.; Ip, R. W. L.; Lau, H. C. W.; *Analysis of Dynamic Dispatching Rules for a Flexible Manufacturing System*, Journal of Materials Processing Technology, 138, p 325–331, 2003.

Charpentier, P.; Thomas, A.; *Reducing Simulation Models for Scheduling Manufacturing Facilities*, European Journal of Operational Research, v. 161, n. 1, p 111-125, fevereiro 2005.

Chen, F. F.; Shukla, C. S.; *An Intelligent Decision Support System for Part Launching in a Flexible Manufacturing System*, International Journal of Advanced Manufacturing Technology, v. 18, n. 6, p 422-433, 2001.

Cho, S.; Prabhu, V. V.; *Distributed Adaptive Control of Production Scheduling and Machine Capacity*, Journal of Manufacturing Systems, v. 26, n. 2, p 65-74, abril 2007.

Dar-El, E. M.; Greco, M. P.; Sarin, S. C.; *Sequencing and Loading of Products on a Flowline*, European Journal of Operational Research, v. 168, n. 3, p 905-921, fevereiro 2006.

Deitel, H. M.; Deitel, P. J.; Java How to Program, 7a ed. New Jersey: Prentice Hall, 2006.

Fernandes, M. C.; *Um Avaliador de Cenários Simulados para Re-escalonamento da Produção em Sistemas Automatizados de Manufatura Usando Lógica Nebulosa*, Dissertação de Mestrado, DC/UFSCar, São Carlos, 2004.

Foote, B.; Johnson, R. E.; *Designing Reusable Classes*, Journal of Object-Oriented Programming, v. 1, n. 2, p 22-35, junho/julho 1988.

Gaither, N.; Frazier, G.; Administração da Produção e Operações. Editora Pioneira, 2001.

Gamma, E.; Helm, R.; Johnson R.; Vlissides, J.; *Padrões de Projeto*, Porto Alegre: Bookman, 2000. Tradução de: Luiz A. Meirelles Salgado.

He, Y.; Smith, M. L.; *A dynamic heuristic-based algorithm to part input sequencing in flexible manufacturing systems for mass customization capability*, International Journal of Flexible Manufacturing Systems, v. 19, n. 4, p 392-409, dezembro 2007.

Herrmann, S.; Veit, M.; *Model-View-Controller and Object Teams: A Perfect Match of Paradigms*, 2nd International Conference on Aspect-Oriented Software Development, p. 140-149, 2003.

Jia, H. Z.; Nee, A. Y. C.; Fuh, J. Y. H.; Zhang, Y. F.; *A Modified Genetic Algorithm for Distributed Scheduling Problems*, Journal of Intelligent Manufacturing, 14, p 351-362, 2003.

Jolai, F.; Rabbani, M.; Rahimi-Vahed, A.R.; Tavakkoli-Moghaddam, R.; Torabi, S.A.; *A Multi-Objective Scatter Search for a Mixed-Model Assembly Line Sequencing Problem*, Advanced Engineering Informatics, v. 21, n. 1, p 85-99, janeiro 2007.

Kampf, M.; Kochel, P.; *Simulation-Based Sequencing and Lot Size Optimisation for a Production-And-Inventory System With Multiple Items*, International Journal of Production Economics, v. 104, n. 1, p 191-200, novembro 2006.

Kato, E. R. R.; Morandin Jr., O.; Politano, P. R.; Camargo, H. A.; Zampronio, J.; *Production Planning System Based on Simulation: Finding the Best Plant Stocking Policy*, Proceedings of the IV INDUSCON – IEEE – Industry Applications Society, p 435-440, novembro 2000.

Kim, Y.D.; Lee, D.H.; Yoon, C.M.; *Two-Stage Heuristic Algorithms for Part Input Sequencing in Flexible Manufacturing Systems*, European Journal of Operational Research, v. 133, n. 3, p 624-634, setembro 2001.

Khoo, L. P.; Lee, S.G.; Yin, X.F.; *A Prototype Genetic Algorithm-Enhanced Multi-Objective Scheduler for Manufacturing Systems*, International Journal of Advanced Manufacturing Technology, v. 16, n. 2, p 131-138, 2000.

Krasner, G. E.; Pope, Stephen T.; *A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80*, Journal of Object-Oriented Programming, v. 1, n. 3, p 26-49, agosto/setembro 1988.

Kunnathur, A. S.; Sundararaghavan, P. S.; Sampath, S.; *Dynamic Rescheduling Using a Simulation-Based Expert System*, Journal of Manufacturing Technology Management, 15(2), p 199-212, 2004.

López-Ortega, O.*; Java Fuzzy Kit (JFK): A shell to build fuzzy inference systems according to the generalized principle of extension*, Expert Systems with Applications, v. 34, n. 1, p. 796-804, janeiro 2008.

Maravelias, C. T.; Prasad, P.; *Batch Selection, Assignment and Sequencing in Multi-Stage Multi-Product Processes, Computers and Chemical Engineering*, v. 32, n. 6, p 1114-1127, junho 2008.

Marvel, J. H.; Schaub, M. A.; Weckman, G.; *Validating the Capacity Planning Process and Flowline Product Sequencing Through Simulation Analysis*, Proceedings of the 2005 Winter Simulation Conference, v. 2005, p 2112-2118, 2005.

Morandin Jr., O.*; Metodologia de Modelagem de Sistemas Automatizados de Manufatura, Utilizando Rede de Petri Virtual*, Tese de Doutorado, Universidade de São Paulo – Escola de Engenharia de São Carlos, São Carlos, 1999.

Moreira, D. A.; *Administração da Produção e Operações*, 2 ed. São Paulo: Cengage Learning, 2008.

Morton, Y. T.; Pizza, G. A.; Troy, D. A.; *An Approach to Develop Component-Based Control Software for Flexible Manufacturing Systems*, Proceedings of the American Control Conference, v. 6, p 4708-4713, 2002.

Mukhopadhyay, S.K.; Saha, J.; Tiwari, M.K.; *Heuristic Solution Approaches for Combined-Job Sequencing and Machine Loading Problem in Flexible Manufacturing Systems*, International Journal of Advanced Manufacturing Technology, v. 31, n. 7-8, p 716-730, janeiro 2007.

Namballa, R. K.; Yalcin, A.; *An Object-Oriented Simulation Framework for Real-Time Control of Automated Flexible Manufacturing Systems*, Computers and Industrial Engineering, v. 48, n. 1, p 111-127, janeiro 2005.

Rahimi-Vahed, A. R.; Tavakkoli-Moghaddam, R.; *A Memetic Algorithm for Multi-Criteria Sequencing Problem for a Mixed-Model Assembly Line in a JIT Production System*, 2006 IEEE Congress on Evolutionary Computation, p 2993-2998, 2006.

Rau, H.; Yin, Y.-L.; *Dynamic Selection of Sequencing Rules for a Class-Based Unit-Load Automated Storage and Retrieval System*, International Journal of Advanced Manufacturing Technology, v. 29, n. 11-12, p 1259-1266, agosto 2006.

Roman, E. S.; *Um Sistema Inteligente para o Seqüenciamento da Produção com o Apoio de Simulação*, Dissertação de Mestrado, DC/UFSCar, São Carlos, 2006.

Silva, A. R.; *Um Método de Análise de Cenários para Seqüenciamento da Produção Usando Lógica Nebulosa*, Dissertação de Mestrado, DC/UFSCar, São Carlos, 2005.

Slack, N. *et. al.*; *Administração da produção*. São Paulo: Atlas, 1997. Tradução de: Ailton Bomfim Brandão.

Smith, T.M.; Stecke, K.E.; *On the robustness of using balanced part mix ratios to determine cyclic part input sequences into flexible flow systems*, International Journal of Production Research, v. 34, n. 10, p 2925-2941, 1996.

Tubino, D. F.; *Manual de planejamento e controle da produção*, 2 ed. São Paulo: Atlas, 2000.

Turban, E.; Aronson, J. E.; Decision Support Systems and Intelligent Systems. Prentice Hall, 2001.

Watson, H. J.; Houdeshel, G.; Rainer Jr., R. K.; Building Executive Information Systems and Other Decision Support Applications. John Wiley & Sons, 1997.