IGOR NEIVA CAMARGO

9,2 (nove e dois)

Sistema de informação para representar a dinâmica de processos na manufatura Estudo de caso: fabricação de engrenagens

Trabalho de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Graduação em Engenharia Mecânica – Habilitação em Automação e Sistemas

Orientador:
Prof. Dr. José Reinaldo Silva

São Paulo
2002

Dedico este trabalho a meus pais, por todo o apoio a minha formação pessoal e desenvolvimento profissional.

AGRADECIMENTOS

Ao professor e orientador José Reinaldo Silva, por toda a dedicação prestada e pela orientação dada ao desenvolvimento deste trabalho.

Ao professor Marcos de Sales Guerra Tsuzuki, pela atenção especial e orientação prestada durante a realização dos meus primeiros projetos acadêmicos.

Aos demais professores da Escola Politécnica, e todos aqueles que de forma direta ou indireta contribuíram para minha formação acadêmica e a conseqüente realização deste trabalho.

RESUMO

Foi realizada uma revisão bibliográfica de conceitos gerais relacionados à administração da produção, dos processos para fabricação de engrenagens e da metodologia M*-Object para análise da organização da manufatura e seu sistema de informação. A metodologia M*-Object serviu como base para o desenvolvimento de um sistema de informação para o caso estudado: a fabricação de engrenagens. Este sistema está focado na descrição do estado da planta e na análise da dinâmica existente entre os processos da manufatura. O sistema foi desenvolvido em três fases: análise organizacional, projeto conceitual e implementação. Na implementação do sistema, foram introduzidas modificações nas redes PDN descritas pela metodologia, de forma a admitir lugares e arcos com peso diferente de um. O sistema desenvolvido é capaz de simular a dinâmica dos processos da manufatura entre o seu estado atual e um estado futuro, dentro das limitações impostas pelo modelo utilizado. Com os resultados da simulação, é possível se prever configurações futuras do estado da planta e os tempos necessários para o processamento de cada lote de peças.

ABSTRACT

It was performed a bibliographical review over the general concepts related to production management, gear production processes and the M*-Object methodology for manufacture organization and its information system analysis. The M*-Object methodology served as a base for the development of an information system for the studied case: gear manufacturing. This system is focused in the analysis of the dynamics manufacture process. The system design was developed in three phases: organizational analysis, conceptual design and implementation. In the implementation phase, some modifications were introduced in the PDN nets originally described by the methodology, in order to admit places and arches with weight. The developed system is capable to simulate the manufacture process dynamics, regarded the limitations imposed by the used model. Thus it is possible to foresee future manufacturing states and the respectively elapsed production times as a result of these simulations.

SUMÁRIO

LISTA DE FIGURAS

1. Introdução

O ciclo de vida de um produto se inicia em sua concepção, evoluindo através das etapas de seu projeto, produção e aperfeiçoamento. Este ciclo pode requerer um ambiente de manufatura integrado e flexível, capaz de se adaptar às constantes inovações. Neste ambiente de mudanças constantes, conhecer o estado atual da produção e prever estados futuros é uma atividade de difícil realização. Frente a essa necessidade, este trabalho desenvolve um sistema de informação para representar a dinâmica dos processos na manufatura.

Conhecendo-se a dinâmica dos processos da manufatura, é possível conhecer o seu estado atual. Por exemplo: a quantidade de itens já processada, a utilização dos recursos produtivos e a alocação destes recursos. Mais do que isso, um sistema simulando a dinâmica dos processos pode ser utilizado para prever tendências para estados futuros. Para isso, o sistema deve descrever de forma adequada a dinâmica e os fluxos de informação entre os processos, sem deixar de garantir a consistência das atividades representadas.

O sistema desenvolvido baseia-se, em parte, na metodologia M*-Object [1]. Essa metodologia analisa a organização da manufatura e do seu sistema de informação, apresentando também uma abordagem específica para o projeto do banco de dados que dará suporte ao sistema. A M*-Object faz uso de Redes de Dados e Processos (PDN – Process Data Network), que é uma integração entre redes-de-petri de alto nível com um modelo orientado a objeto.

A abordagem orientada a objeto suporta conceitos como modularidade, flexibilidade, reutilização, facilidade de extensão e uma estrutura hierárquica de classes, definida por relações de herança. Isso faz com que esse modelo seja apropriado para a descrição dos elementos envolvidos na manufatura, que passam então a ser analisados como objetos. A estrutura de classes descreve propriedades tanto estáticas como comportamentais do sistema. As redes-de-petri, por sua vez, servem para representar as propriedades dinâmicas dos processos envolvidos na manufatura.

O estudo de caso apresentado restringe a análise aos processos envolvidos a uma planta para fabricação de engrenagens automotivas. Essa simplificação, contudo, não priva os conceitos desenvolvidos durante o projeto de sua generalidade. A mesma metodologia poderia ser utilizada para descrever outros processos de fabricação ou montagem. Neste ponto, é importante a flexibilidade do modelo utilizado, que pode ser adaptado à estrutura de cada processo. E, mesmo em relação a um único processo, o sistema deve ser suficientemente flexível para acompanhar o ciclo de vida do produto.

2. Objetivos

O objetivo deste trabalho é o desenvolvimento de um sistema de informação capaz de representar a dinâmica dos processos em uma planta, de forma a fornecer o estado atual de cada objeto da manufatura – como peças e máquinas, por exemplo, e prever seus estados futuros. O objeto do projeto é uma planta simplificada para a fabricação de engrenagens, que constitui o estudo de caso. O sistema desenvolvido neste trabalho é composto pelos seguintes elementos:

- Projeto conceitual do sistema de informação, segundo a metodologia M*-Object, para a planta em análise. O modelo obtido descreve a dinâmica dos processos envolvidos na manufatura e o fluxo de informações associado.

- Um módulo de software capaz de simular a dinâmica dos processos na fábrica, interpretando as informações provenientes do chão-de-fábrica segundo o modelo PDN representado;

- Um repositório de dados capaz de representar o modelo PDN proposto e todas as demais informações relacionadas ao sistema.

O sistema desenvolvido deve ser capaz de fornecer a atual utilização da capacidade do chão-de-fábrica, o estágio de fabricação de todas as peças com pedidos e a previsão dos estados futuros de fabricação, tempos e recursos utilizados. A previsão dos estados futuros se baseia no estado atual e no modelo da dinâmica dos processos da planta, que é simulado pelo sistema.

3. Contexto do projeto

Os dados, informações e a própria dinâmica da empresa, assim como do seu ambiente de manufatura, precisam estar disponíveis e devidamente integrados. Portanto, é necessário que todos os sistemas compartilhem o mesmo conhecimento a respeito destes elementos. Isso só é possível perante a existência de uma infra-estrutura de integração e de um modelo que descreva a empresa.

A infra-estrutura de integração suporta, por meio de hardware e software, que um sistema A se comunique com outro B através de um fluxo de objetos de informação. O modelo da empresa é um referencial comum. Ele assegura que, quando um sistema A se refere a um conceito C, outro sistema B se referindo a C também possui o mesmo entendimento de C que o sistema A. Isto é, A e B compartilham o mesmo conhecimento sobre C [1].

Dentro do ambiente de manufatura CIM[1], diferentes aspectos da empresa podem ser modelados. Eles são:

- Produtos;
- Recursos;
- Informação;
- Organização e decisões;
- Recursos humanos;
- Processos de negócio.

Esse trabalho está focado na modelagem das informações da empresa, mais especificamente, informações sobre o ambiente da manufatura em termos da dinâmica dos processos nela envolvidos.

[1] CIM – Computer Integrated Manufacturing

As metodologias para projeto de sistemas de informação consistem em modelos usados para descrever o sistema sob análise (a empresa) e métodos para realizar essa descrição. As primeiras abordagens neste sentido enfatizaram a representação de estruturas estáticas, ou seja, os componentes do sistema e seus dados, como o modelo entidade-relacionamento. A coesão neste modelo é dada por restrições que garantem a integridade dos dados. Contudo, em ambientes de manufatura, a mera descrição dos dados não consegue representar de forma adequada os elementos dinâmicos do sistema.

Uma alternativa é a representação dos elementos dinâmicos, referidos como processos, em atividades estruturadas e regidas por eventos. As regras nesse caso são restrições que garantem a integridade da dinâmica do sistema, o que pode ser representado por redes-de-petri elementares, coloridas ou mesmo do tipo predicado/transição. Estas últimas conseguem, ainda, uma integração com o modelo entidade-relacionamento.

Contudo, esse modelo sozinho não é suficiente para representar os objetos envolvidos na manufatura, que requerem um maior grau de flexibilidade em sua representação. Essa característica é complementada incluindo-se o modelo orientado a objeto, que suporta os conceitos de modularidade, reutilização, facilidade de extensão, flexibilidade, herança de classes, polimorfismo e reusabilidade.

A metodologia M*-Object foi desenvolvida procurando-se integrar ambos os modelos: de redes de Petri de alto nível e orientação a objeto. Essa união busca também minimizar as fraquezas de ambas as abordagens, quando empregadas no desenvolvimento do sistema de informação dentro do paradigma CIM.

Este trabalho busca descrever a dinâmica dos processos manufatura seguindo o modelo de análise proposto pela metodologia M*-Object. A fase de implementação do sistema, contudo, conta com uma abordagem diferente da apresentada por esta metodologia.

4. Metodologia M* Object

4.1 Etapas

A M*-Object é uma metodologia para projeto de sistemas de informação desenvolvida para ambientes CIM, que compreende três fases:

- *Análise organizacional:* Esta fase está concentrada na análise funcional e organizacional do sistema real (a empresa, um sistema produtivo, um conjunto de atividades, uma planta etc.), em termos de um conjunto hierarquicamente estruturado de redes que descrevem o fluxo de objetos (informações, material) manipulados pela empresa e seu comportamento funcional (fluxo de controle).

  Essa etapa é basicamente um projeto top-down, com o objetivo de identificar os componentes do sistema e analisar o seu comportamento funcional em diferentes níveis de detalhamento. Ela está baseada nos seguintes princípios:

  - Decomposição hierárquica dos processos e atividades do sistema;
  - Descrição funcional das interações entre os componentes do sistema;
  - Especificação do comportamento intrínseco destes componentes

- *Projeto conceitual:* Nesta fase, a descrição funcional da empresa é utilizada para gerar a sua especificação conceitual, por meio de um "esquema conceitual". Esse esquema contém uma descrição unificada dos diferentes aspectos da realidade: elementos estáticos, dinâmicos e comportamentais.

- *Implementação:* A fase de implementação consiste em traduzir o modelo conceitual em uma descrição do sistema que possa ser diretamente aplicada através de um repositório de dados. Neste projeto, o banco de dados serve ainda como uma interface entre as informações vindas do ambiente da manufatura e o sistema em si.

4.2 Modelo PDN

4.2.1 Apresentação das redes de dados e processos

As Redes de Dados e Processos (PDN) são o modelo utilizado na fase conceitual da metodologia M*-Object. Essa etapa é essencialmente um projeto do tipo *bottom-up*, ou seja, inicia-se pela análise de componentes elementares e, através de níveis crescentes de abstração, são elaborados posteriormente os conceitos mais complexos. Essa ferramenta propicia uma forma de analisar o sistema de informação e seu projeto, dando também suporte ao desenvolvimento do banco de dados. A especificação obtida do processo por meio da PDN é um resultado detalhado, que pode ser usado para uma prototipagem do sistema, ou mesmo gerar um sistema executável através de um interpretador PDN.

Em PDN, o sistema é descrito por meio de seus estados, que são especificados, em qualquer momento, como um conjunto de objetos armazenados no banco de dados; e por meio das condições que regem os eventos aos quais o sistema responde. Estas condições especificam se os eventos estão ativos ou inativos.

O modelo pode ser apresentado introduzindo-se seus três elementos de linguagem componentes. Os dois primeiros se referem à representação de propriedades estáticas do sistema: a linguagem para modelagem de dados e a linguagem para representação e obtenção dos dados. O ultimo é a representação da dinâmica e do comportamento do sistema.

4.2.2 Representação de propriedades estáticas: modelo de dados

O modelo de dados fornece os seguintes mecanismos de abstração para o projeto:

1. *Objetos:* representam as entidades reais. Os objetos possuem estado, definido por um conjunto de atributos, e comportamento, definido por um conjunto de

métodos e as condições atribuídas a estes métodos (ativos ou inativos). Os objetos possuem um atributo identidade (oid: identificador do objeto); ou seja, isso faz com que eles sejam únicos no sistema e possam ter uma correspondência com objetos do mundo real.

2. *Classes:* Uma classe é um grupo de objetos que compartilham uma estrutura comum e possuem o mesmo comportamento. A classe define a estrutura e comportamento dos objetos que a instanciam.

3. *Atributos:* São os parâmetros que descrevem a estrutura de uma classe. Podem ser definidos por outras classes ou domínios específicos (por exemplo, *Strings*, inteiros etc.).

4. *Encapsulamento:* Os acessos aos objetos são realizados através de uma interface destes como o meio, representada pelos seus métodos. A implementação interna não deve afetar a interface (a complexidade interna passa a ser encapsulada).

O modelo de dados das PDNs possui embasamento em ambos os modelos: orientado a objeto e entidade-relacionamento. Uma espécie de atributo importante em uma classe são os *atributos de relacionamento,* que definem relações de cardinalidade entre a classe a qual pertencem e outra classe do sistema. Nesta classe correspondente, existirá também um atributo que define uma relação de cardinalidade inversa em relação à primeira.

Desta forma, sendo *e*, $e \in E$, um objeto da classe E; e sendo *f*, $f \in F$, um objeto da classe F, tem-se que: um atributo x de E definindo uma cardinalidade mínima de m_x e máxima de M_x em relação à classe F, implica que, para um objeto *e,* devem sempre estar relacionados um mínimo de m_x e um máximo de M_x objetos de F. O relacionamento entre as duas classes descrito pelo atributo x em E, e y em F, é representado por $\langle x(m_x, M_x), y(m_y, M_y) \rangle$.

Um ambiente genérico e simples para uma célula flexível de manufatura (MFC) é um bom exemplo de esquema de dados, conforme a Figura 4.1. Neste diagrama, os retângulos representam as classes, sendo os círculos seus atributos. Os atributos relacionais estão indicados ao lado dos losângulos, que indicam os relacionamentos entre as classes. As setas pontilhadas representam os atributos que são identificadores das classes.

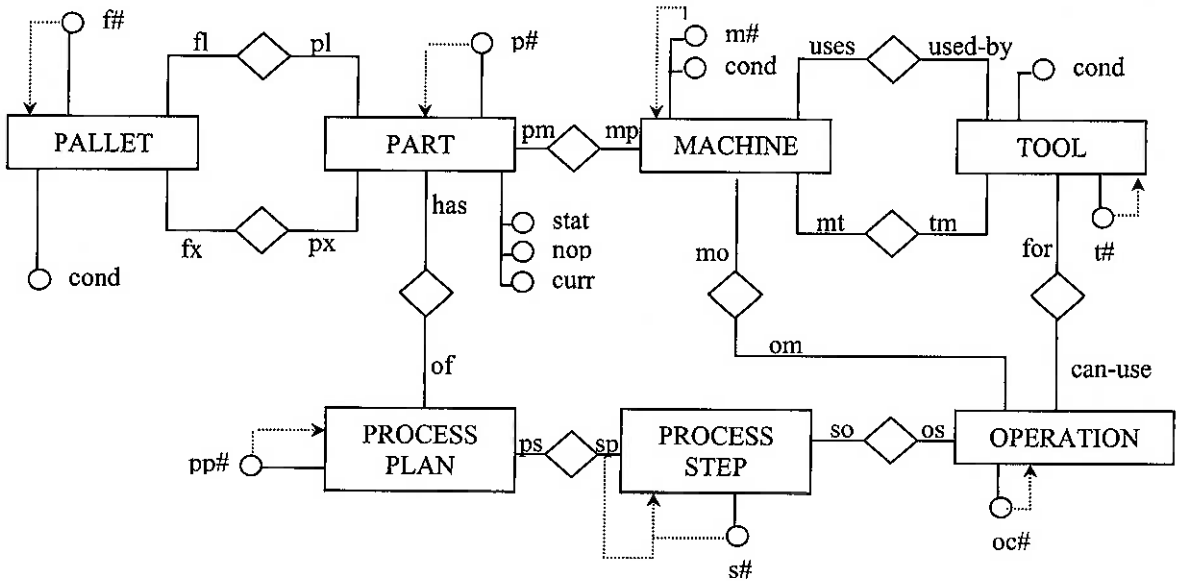


Figura 4.1 esquema de dados para o exemplo FMC

Há outros elementos gráficos para representação de herança e de outras características do modelo orientado a objeto. Para a estrutura dada anteriormente, um exemplo de definição de classe seria, para a classe Pallet:

```
class PALLET  identifier f#        -O oid da classe é f#
attributes
f#: string;                        -nome do atributo: tipo
cond:('available','in-use');       -tipo é uma lista de valores
fl(1,n): PART inverse of pl;       -o atributo fl expressa uma
                                   relação de cardinalidade de no
                                   mínimo 1 com a classe PART. O
                                   atributo relacional
                                   correspondente em PART é pl.

fx(0,1): PART inverse of px;
```

A manipulação dos Objetos ocorre por meio dos métodos definidos para as classes que eles pertencem. Há, alem disso, procedimentos especiais definidos para a inserção, alteração, exclusão e seleção de objetos em uma classe, respectivamente *Create*, *Update*, *Delete* e *Select*. Outros procedimentos, referentes à classe, são *Specialize* e *Generalize* para criar a partir de uma classe uma subclasse ou superclasse, característica do modelo orientado a objeto. Há também procedimentos para inserção e remoção de objetos em agrupamentos de objetos, *Add* e *Remove*.

Para o exemplo da MFC, o seguinte bloco de código exemplifica uma transação envolvendo manipulação de objetos:

```
use w|OPERATION,                    -Objeto w da classe OPERATION
    m|MACHINE, t|TOOL;...
exec                                -Início do bloco da transação
m.Select(m#='7');                   -Seleciona objeto com
                                    identificador m# igual a 7 em m
w.Select(oc='op12');                -Seleção por outro atributo
t.Create(t#:'59',                   -Cria uma nova ferramenta,
         cond:'Available,           com os atributos dados, e ainda
         for: w,                    atualizando o relaciomanto com a
         used-by: m);               classe m, dado pelo atributo
                                    used-by
m.Update(uses:+t);                  -insere t no no relacionamento
                                    com a m, dado pelo atrib. uses
w.Update(can-use:+t);               -idem
end
```

A linguagem de modelagem de dados em PDN é tanto uma forma de visualizar o relacionamento existente entre as diferentes classes de objetos, como também serve como código para interpretadores PDN capazes de gerar código para aplicações que comporão o sistema de informação do chão-de-fábrica e da organização como um todo. Ela comporta tanto o modelo entidade-relacionamento quanto a orientação a objeto.

4.2.3 Representação de propriedades estáticas: manipulação de dados

A) Views

A linguagem de manipulação de dados em PDN permite ao projetista do sistema de informação expressar os resultados das pesquisas (*queries*) no conjunto de objetos (futuramente implementado como um banco de dados) através de mecanismos especiais, chamados *views*. No modelo PDN, uma *view* é uma classe virtual, que corresponde no modelo entidade-relacionamento a um relacionamento "virtual".

A sintaxe de uma *view* é dada por:

```
view <<nome da view>>(<<lista de variáveis de comunicação>>)
use <<lista de variáveis>> where <<predicado>>
```

Na qual a lista de variáveis expressa os objetos que compõem a *view* e serão selecionados entre as classes definidas de acordo com a expressão dada pelo predicado definido após *where*. O predicado é um mecanismo de seleção para expressar um subconjunto de objetos contido no conjunto total das classes envolvidas. A seguinte *view*, baseada no exemplo FMC, demonstra o emprego da estrutura:

```
view PMT(p|PART, m|MACHINE, t|TOOL) where
     p.stat='working' and m.cond='working' and
     t.cond='working' and p<pm>m<mt>t
```

A *view* PMT especifica a parte que está sendo trabalhada na máquina m, usando-se a ferramenta t. A descrição de caminho *p<pm>m<mt>t* especifica justamente o relacionamento entre os objetos especificados por PMT através de seus atributos de relacionamento: *pm* entre a parte e a máquina e *mt* entre a máquina e a ferramenta.

B) Rotinas

Em PDN, as manipulações de dados ocorrem por meio de rotinas. As rotinas representam transições de estado dos objetos do sistema. Elas podem ser utilizadas para trocar mensagens com o ambiente externo. Além disso, definem ainda as transações mantidas com o banco de dados.

Estas transições de estado estão diretamente relacionadas às redes de processos (p-nets) apresentadas na próxima seção. Um exemplo de transição de estado é a rotina de montagem de uma peça no pallet, *Mount*:

```
message In(pr#:string)
view PR(p|PART) where p.stat='ready';
view FA(f|PALLET) where f.cond='available';
view PM(p|PART) where p.stat='mounted';
view FI(f|PALLET) where f.cond='in-use';
```

Essa primeira parte define as views e mensagens utilizadas pela rotina. As mensagens são, na representação p-net, os "lugares" que precedem e sucedem as transições de estado. A rotina então fica:

```
routine Mount
[accept In(pr#), PR(p), FA(f) return PM(p), FI(f)]
guard p.p#=pr# and p<pl>f
exec
p.Update(stat:'mounted', curr:1, px:f);
f.Update(cond:'in-use', fx: p)
end Mount
```

Os elementos de sintaxe presentes na declaração da rotina são:

- *Accept*: nesta cláusula são especificados as mensagens e os dados de entrada da rotina. As mensagens são eventos que precisão estar ativos para a execução da rotina. Os dados serão atualizados pela rotina.

- *Return*: uma mensagem de saída nesta cláusula descreve um evento que será ativado após a execução da rotina. As views declaradas são as apresentadas na saída da transição. Tanto mensagens como views nas cláusulas *accept* e *return* são os "lugares" nas p-nets.

- *Guard*: essa cláusula é utilizada para especificar condições à execução da rotina (condições necessárias à transição de estado) através de um predicado e descrição de caminho, envolvendo os dados de entrada e os dados locais.

- O bloco descrevendo as transações efetuadas na rotina é delimitado pelos termos **exec** e **end**.

No exemplo, as *views* PR e PM representam as partes antes e depois da transição de estados (execução da rotina). O mesmo em relação a FA e FI, para os pallets. Se a mensagem In é ativa, e a parte designada pode ser montada no pallet escolhido (condição verificada pela cláusula *guard*), então os objetos *parte* e *pallet* são "transferidos" para as views de saída. A representação desta transição na p-net de forma gráfica é mostrada pela Figura 4.2. Se a transição ocorre, a parte p pertencente ao *view* PR é transferida para a saída, no *view* PM. O mesmo ocorre com o *pallet* f. A mensagem In deve estar ativada para habilitar a transição. Da mesma forma, as entradas também devem satisfazer a condição dada pela cláusula *guard*.

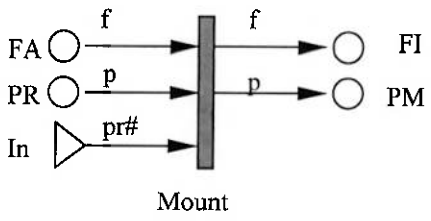


Figura 4.2 p-net representando a transição de estado (rotina) Mount.

4.2.4 Representação da dinâmica e do comportamento do sistema

Uma das propriedades do modelo PDN é a distinção entre as características dinâmicas e comportamentais dos objetos. A dinâmica é representada por meio da rede de processos, p-net; e o comportamento de uma classe é especificado através da rede de comportamento do objeto, ob-net.

A) Rede do processo

A p-net é uma representação do processo analisado, comportando sua estrutura dinâmica e as rotinas nele envolvidas. Voltando ao exemplo da célula de manufatura flexível, a p-net correspondente é dada pela Figura 4.3. A rede mostra o fluxo de partes, pallets, máquinas e ferramentas. As operações de montagem, preparação, trabalho, envio e desmontagem são dependentes da marcação presente na rede, assim

como dos objetos presentes em cada view (lugares da rede). Os procedimentos Mount, Work, Prep e Send são utilizados, respectivamente, para montar a parte no pallet, trabalhar a peça em uma máquina, preparar a operação e enviar a peça para uma nova operação.

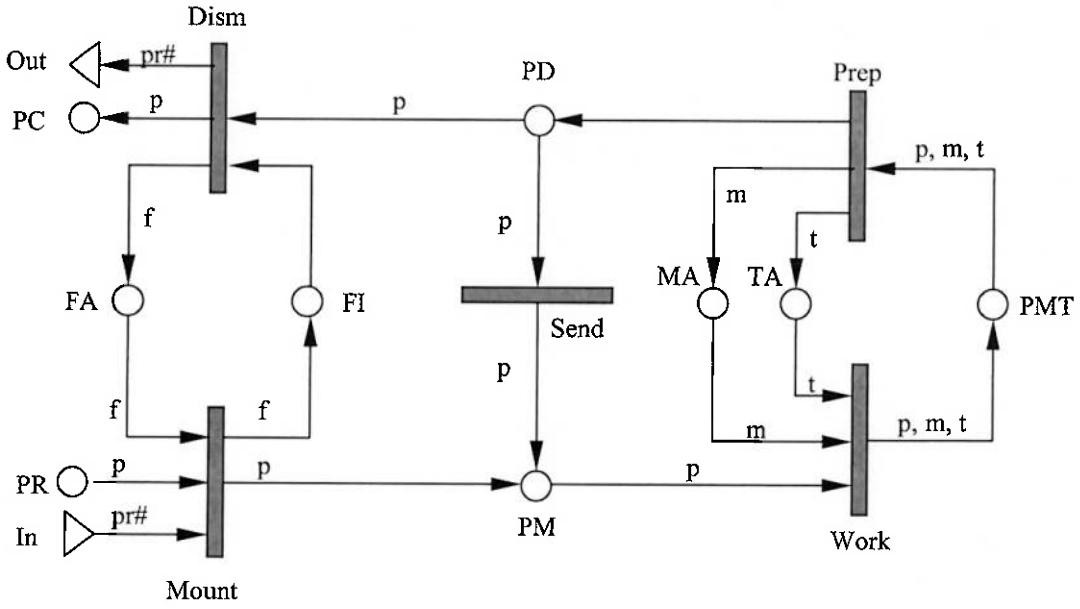


Figura 4.3 p-net correspondente ao exemplo da FMC

A p-net é composta pelos seguintes elementos:

- Uma rede expressando a estrutura de controle do sistema, envolvendo três conjuntos de elementos: lugares, rotinas (transições) e arcos direcionados – que conectam cada transição aos lugares de entrada e saída.
- O conjunto de lugares é subdividido entre mensagens e views, que especificam o conjunto de objetos do sistema.
- A marcação da p-net, que consiste em um conjunto de marcações dadas nas views, e de condições para as mensagens – ativas ou inativas.

- Um conjunto de regras de transição, que determinam a evolução do sistema. A marcação da p-net habilita uma transição T se todas as mensagens de entrada estiverem ativas, se todas as mensagens de saída estiverem inativas e, ainda, se existe um conjunto de objetos na entrada que satisfaz a regra substituição dada pela cláusula *guard* da rotina T. Ocorrendo a transição, os objetos processados presentes na entrada são removidos e realocados nos lugares da saída de T.

B) Rede de comportamento

O comportamento é uma propriedade inerente a cada classe, que estipula a forma pela qual seus objetos individuais podem evoluir, que pode ser diferente da evolução do processo (p-net). O comportamento estipula a ordem na qual os métodos podem ser chamados para cada objeto de uma classe.

A rede de comportamento do objeto (ob-net) de uma classe descreve os estados, métodos e todas as possíveis seqüências de chamada a estes métodos. As ob-nets são um caso particular de redes-de-petri, com as seguintes características:

- São constituídas de três conjuntos: um conjunto de *views*, representando os estados do objeto; um conjunto de métodos, que realizam a transição entre os estados dos objetos; e o último sendo um conjunto de arcos direcionados, ligando cada método às suas *views* de entrada e de saída.
- O conjunto de *views* pode ser interpretado como o estado do objeto.
- Os caminhos na rede determinam as seqüências possíveis de evocação dos métodos da classe. Um método só pode ser chamado por um objeto se este objeto pertence a suas *views* de entrada. Se a transição ocorre, isto é, o

método é chamado, então o objeto muda de estado – indo para a *view* de saída do método.

- Um método precisa ser chamado para que seja executado. A chamada de métodos é realizada dentro das rotinas das p-nets.

No exemplo da FMC, a ob-net para a classe PART é descrita textualmente por:

```
class PART ...
methods prm(f|PALLET);pmw(m|MACHINE);pwd;pdm;pdc
class PART implementation
  view PR(p|PART) where p.stat='ready';
  view PW(p|PART) where p.stat='working';
  view PC(p|PART) where p.stat='completed';
  view PM(p|PART) where p.stat='mounted';
  view PD(p|PART) where p.stat='worked';
  method prm(f|PALLET) accepts PR(p) return PM(p)
    exec p.Updadte(stat:'mounted', curr:1, px: f) end
  method pmw(m|MACHINE) accepts PM(p) return PW(p)
    exec p.Updadte(stat:'working', pm: m) end
  method pwd accepts PW(p) return PD(p)
    exec p.Updadte(stat:'worked', curr: curr+1, pm: null) end
  method pdm accepts PD(p) return PM(p)
    exec p.Updadte(stat:'mounted') end
  method pdm accepts PD(p) return PC(p)
    exec p.Updadte(stat:'completed', px: null) end
```

A representação gráfica desta rede é mais clara, permitindo identificar a evolução do objeto durante os processos. Na Figura 4.4, o método *prm* representa a montagem da peça no *pallet* e deve ser executado antes da peça ser trabalhada pela máquina (*pmw*). Após a peça ser trabalhada, o novo estado é obtido por *pwd*, ou seja, a parte adquire o estado de trabalhada. Desta etapa, ela pode ser encaminhada para uma nova operação (*pdm*) ou ser dada como acabada (*pdc*).

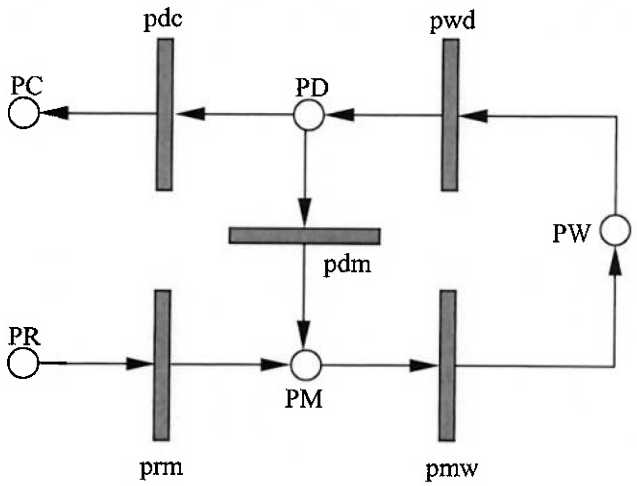


Figura 4.4 ob-net da classe PART.

4.2.5 Utilização da PDN no projeto

No projeto do sistema de informação proposto, foram utilizadas PDNs para descrever as características dinâmicas e de comportamento do sistema. A descrição textual das redes, contudo, não será utilizada neste trabalho. Essa descrição é útil em meio ao desenvolvimento utilizando-se ferramentas especiais, como programas interpretadores de PDN. Para o propósito apresentado, a descrição gráfica das redes é suficiente.

O sistema de informação desenvolvido implementa as redes de dados e processos, utilizando uma linguagem orientada a objeto: Java. Os elementos da rede, portanto, são diretamente implementados por meio de classes. Outro ponto importante: a representação dos objetos é primeiramente instanciada na memória a partir do banco de dados, para então poder ser utilizada, por exemplo, para realizar uma simulação. As *views*, portanto, não são diretamente atualizadas no banco de dados, mas sim na instância presente na memória alocada para o programa.

5. Análise Organizacional

5.1 Apresentação do sistema em análise

Esta etapa do projeto está focada na descrição do sistema em análise, nas suas características atuais e na definição de uma nova estrutura, que permita melhorar os processos envolvidos ou solucionar problemas existentes. Neste trabalho, o sistema em estudo é composto pelo conjunto de atividades existentes em uma planta de manufatura, pelos elementos envolvidos nestas atividades (peças trabalhadas, máquinas e outros recursos empregados) e pelos processos (seqüências de atividades).

O exemplo de manufatura escolhido para análise é uma fábrica genérica de engrenagens. Este estudo de caso envolve vários aspectos relevantes da manufatura, como: processos de usinagem; tratamento do material; controle de qualidade; processamento em lote; aglutinação e transporte em pallets; seqüências de processos envolvendo diferentes tipos de operações e máquinas. Cada um destes aspectos possui uma complexidade própria ao se relacionar com os demais, o que influi diretamente na dinâmica das atividades realizadas em cada processo.

5.2 Estudo de caso: fabricação de engrenagens

No caso em estudo, a engrenagens fabricadas fazem parte da caixa de transmissão de um veículo de passeio. Suas principais partes são:

- Dentado helicoidal: para transmissão do esforço mecânico proveniente do eixo para outra engrenagem;
- Cone de sincronização: utilizado durante o processo de sincronização para elevar a velocidade da engrenagem à do eixo, ou vice-versa, por meio do dispositivo sincronizador que acompanha a rotação do eixo;
- Chanfro do sistema de sincronização;

- Dentado de indexação: após a fase de sincronização, o indexador fixa a engrenagem ao dispositivo sincronizador, mantendo assim a engrenagem na mesma rotação que o eixo.

Na mudança de marcha, o dispositivo sincronizador é deslocado longitudinalmente no eixo, deixando uma engrenagem e seguindo para outra. Assim, a primeira engrenagem passa a rodar em falso no eixo. A segunda, após a sincronização e indexação por meio do dispositivo, passa a ser responsável pela a transmissão do torque entre o seu eixo e a engrenagem do segundo eixo. Ao mesmo tempo, as engrenagens correspondentes do segundo eixo também passam pelo mesmo processo de sincronização. Desta forma, realiza-se a mudança do par de engrenagens responsável pela transmissão de esforço mecânico entre os dois eixos. Com pares de engrenados envolvendo diferentes relações de transmissão, obtém-se então multiplicidade na escolha das relações de transmissão de velocidade e torque entre os eixos.

A fabricação das engrenagens descritas envolve os seguintes tipos de operações:

- "Blanking": é o processo de forjamento que resulta na peça em seu estado bruto (blank);
- Operações de remoção de material: hobbing, shaving, torneamento, brunimento (ou retificação) e lapidação;
- Operações de tratamento térmico do material (cementação, têmpera e revenido), lavagem;
- operações de controle de qualidade: metrologia, inspeção.

Considera-se, neste caso, a peça bruta já em seu estado forjado. O processo de fabricação pode ser dividido em três sub-processos:

1. Processo a verde: realizado no forjado em seu estado bruto. Compreende: o torneamento dos diâmetros interno e externo; o torneamento das faces; fresamento dos dentes de engrenamento do sistema de sincronização; hobbing para o corte dos dentes helicoidais. Esta fase pode incluir também o shaving, que substitui a retificação após o tratamento térmico. A inspeção das medidas da peça decide se a peça segue para as próximas etapas do processo, se vira refugo ou se deverá ser trabalhada novamente.

2. Tratamento térmico: as peças são lavadas por mergulho e seguem para o forno de cementação. A peça é então temperada e revenida, seguindo logo após para um estoque intermediário onde o material "descansa" (esfria).

3. Processo de retificação: para corrigir imperfeições geométricas causadas durante o processo de tratamento térmico e melhorar o acabamento superficial do material. Esta fase envolve verificações nas medidas e qualidade do material. Após as inspeções, a peça pode ser considerada dentro das especificações ou virar refugo.

Cada tipo de engrenagem produzida possui uma seqüência de operações e parâmetros própria. Será discutido um exemplo de seqüências de atividades no projeto conceitual do sistema.

5.3 Discussão: alguns problemas da manufatura

Uma preocupação constante na manufatura é encontrar formas de produzir levando-se em consideração as seguintes metas:

1. Diminuir custos;
2. Diminuir prazos de entrega;
3. Aumentar qualidade.

Encontrar um equilíbrio entre estes três fatores demanda bastante planejamento da produção e conhecimento de todos os processos envolvidos. Para a fabricação de uma peça, existem várias questões que devem ser consideradas na busca de uma alternativa ótima de produção. Por exemplo:

- Quais máquinas disponíveis no pátio devem ser empregadas na produção? Compensa investir na ampliação da linha para atender um novo cliente? Em quais pedidos empregar as máquinas mais rápidas e modernas?
- Que processos disponíveis devem ser utilizados? Compensa terceirizar alguma etapa da produção?
- Como utilizar melhor os recursos considerados "gargalos" da produção?
- Como melhorar a utilização do espaço da fábrica e diminuir a necessidade de movimentação interna de peças?
- Como otimizar a utilização do trabalho dos operadores e a distribuição destes entre os diversos turnos e células de produção?

Estando escolhida uma alternativa viável para a produção, continua ainda o esforço para melhorar a qualidade do produto e diminuir o tempo total de fabricação e os custos envolvidos. Além disso, há uma constante necessidade de reorganização da produção em virtude de fatores diversos, que podem ser previsíveis e imprevisíveis. Tanto a quantidade de refugo, quanto certos atrasos na produção podem ser previstos e tratados por meio de ferramentas estatísticas. Assim, são incorporadas na produção margens de segurança no tempo e na quantidade produzida. Estas duas variáveis estão inter-relacionadas: atrasos podem ser compensados por volumes maiores de produção, assim como a quantidade de peças refugadas também o é.

No modo de produção atual, várias operações são realizadas por inteiro para depois serem levantados dados sobre os resultados e o andamento da produção, como refugo e atrasos na fabricação. Assim, podem ocorrer discrepâncias que resultarão em atrasos na entrega ou mesmo insuficiência de matéria-prima para a produção. Para contornar esses problemas, são empregados métodos estatísticos para prever margens seguras de tempo de produção e estoques intermediários de peças antes e entre os vários processos produtivos.

Quanto maior o tempo previsto para a fabricação, levando-se em conta as margens de segurança de tempo, menor será a utilização da capacidade fabril no tempo. Ou seja, a produtividade é menor. A existência de estoques intermediários, por sua vez, implica em um maior volume de capital parado na produção, além de ocupar espaço valioso na fábrica e gerar gastos com logística [10].

Além de fatores previsíveis, como os descritos acima, existem também outros de difícil predição, que influenciam diretamente a produção. Por exemplo: atrasos nas entregas dos fornecedores e reparos longos em máquinas especiais.

Assim, pode-se elevar a produtividade e reduzir o custo da produção através de ações que:

1. Reduzam as margens de tempo necessárias para garantir o prazo de entrega dentro de limites aceitáveis;

2. Diminuam a quantidade de peças brutas em estoque e de peças semitrabalhadas em estoques intermediários;

3. Maximizem a utilização da capacidade fabril;

4. Reduzam o tempo de resposta entre a ocorrência de falhas no processo produtivo e as medidas para a correção destas.

Todas estas ações dependem da capacidade que a supervisão e a gerência de fábrica possuem em obter dados sobre a produção, dados que expressem o seu estado atual. Ou seja, o conhecimento do que acontece no chão-de-fábrica é o insumo mínimo necessário para a tomada de decisões sobre a produção.

A forma como os dados do chão-de-fábrica são coletados determina a rapidez com que estes dados estarão disponíveis para as diversas análises e tomadas de decisões que se fazem necessário. Contudo, a mera coleta de informações na fábrica não oferece, sozinha, o subsídio necessário para se compreender o atual estado da produção. É necessário relacionar os dados coletados no chão-de-fábrica com um modelo que expresse a dinâmica dos processos que estão sendo realizados. Dessa forma, também é possível fazer inferências em relação às tendências de mudança do estado atual.

Através do sistema de informação proposto, dados sobre o estado de cada processo são coletados e analisados sob o ponto de vista do modelo que representa a dinâmica destes processos. Uma vantagem do sistema de informação é poder fornecer, a qualquer momento, uma análise do estado da produção e das suas tendências, de forma imediata.

Há, todavia, um problema que dificulta a implementação de um sistema 100% instantâneo de coleta de dados do chão-de-fábrica: muitos processos não são monitorados de forma contínua. Isto dificulta a determinação do estado de realização de cada atividade prevista. Esse aspecto pode ser contornado aumentando a informatização e automação da planta, elevando-se assim o número de dados coletados durante a realização dos processos – por exemplo, sinais indicando o início e fim do processamento de cada peça ou conjunto de peças. Essa solução,

contudo, implica em investimentos e aumento de custos indiretos, que devem ser analisados antes de se tentar obter um sistema de coleta de dados 100% efetivo (o que pode, mesmo assim, não ser completamente possível).

5.4 Componentes relacionados ao processo de fabricação

Segundo a metodologia M*, podem ser definidos os seguintes componentes para a organização fabril em estudo:

1. **A Organização**: no caso em estudo, é a planta genérica para a fabricação de engrenagens.

2. **Ambiente da organização**: é um subconjunto da organização que possui o mesmo conjunto de funções e usuários. O ambiente compartilha a mesma visão do sistema de informação. Cada ambiente possui um conjunto de funções associadas, as denominadas funcionalidades do ambiente. O ambiente escolhido, no caso, é o responsável pela manufatura.

3. **Unidades organizacionais**: são um subconjunto do ambiente, o qual pode ainda ser dividido em centros de trabalho. Cada centro de trabalho é responsável pela realização de uma atividade. No caso em estudo, não se fará distinção entre centros de trabalho e unidades organizacionais. Será referido apenas o centro responsável pela realização de cada atividade.

Os conceitos de operação, agente e processo estão diretamente relacionados à organização fabril:

1. As operações são definidas como um conjunto de atividades elementares realizadas em um centro de trabalho, que podem ser realizadas sem interrupções e de forma independente de outras atividades. Alguns exemplos de operações são: realização completa do torneamento do diâmetro externo da engrenagem; tratamento térmico do blank usinado; inspeção de qualidade final.

2. O Agente é uma máquina ou um centro de trabalho responsável pela realização de um conjunto de atividades, ou seja, de uma operação.

3. Os processos são conjuntos coordenados de operações que desempenham um objetivo. No caso em estudo, o processo de fabricação integral de cada tipo diferente de peça constitui um processo. O conjunto de processos constitui a funcionalidade do ambiente da organização.

Claramente, existem diversas variantes de processos de fabricação e também uma grande variedade de tipos e formas de engrenagens produzidas. O escopo do trabalho, contudo, é analisar o estado da produção com base na dinâmica do conjunto de processos de fabricação, ou seja, da seqüência através da qual as várias operações – atividades – são realizadas.

Cada peça a ser produzida possui uma seqüência de atividades definida, assim como uma lista de máquinas (ou centros de produção) capazes de realizar cada uma das operações as quais estará submetida. Da mesma forma, cada máquina também está relacionada a um conjunto de tipos de peças e também um conjunto de processos dos quais estas podem fazer parte. Os processos, por sua vez, são específicos para cada tipo de peça. Contudo, podem ser desempenhados por diferentes centros de trabalho.

No sistema de informação desenvolvido, é denominado "estado da produção" o conjunto de estados de todos os objetos da manufatura (centros de trabalho, peças, pallets, etc.). Conhecer o estado atual da produção permite descrever o estágio no qual se encontra cada processo de fabricação, cada peça, cada agente. O estado da produção também descreve o que cada centro de trabalho processa no momento, as peças que aguardam serem processadas e as quantidades de estoques intermediários entre os processos.

6. Projeto conceitual

6.1 Objetivo do projeto conceitual

O projeto conceitual tem por objetivo descrever o sistema de informação e especificá-lo através do esquema conceitual. O esquema conceitual é um conjunto de p-nets, ob-nets, suas descrições, views, métodos, rotinas e outros elementos associados. Ele é a base para a implementação do sistema de informação, que constitui a última etapa do trabalho.

O comportamento de cada classe de objetos da manufatura será apresentado através de sua ob-net característica. Foram escolhidos os seguintes elementos para representar os objetos da manufatura: agente, peça e pallet.

O agente é o responsável pela transformação da peça. Pode ser uma máquina de usinagem, um forno, um operador ou mesmo uma célula inteira de produção. O conjunto com todos os agentes disponíveis representa a capacidade da fábrica. O pallet, por sua vez, é um dispositivo para agrupar peças. Este elemento foi inserido no contexto do projeto para incluir na análise a complexidade própria relacionada aos agrupamentos.

Esta fase também engloba a representação do processo de produção de um tipo de engrenagem através da sua p-net. Para a construção desta p-net, serão utilizados blocos de construção básicos, as operações. Agrupando-se os diferentes tipos de operações é possível construir p-nets para representar outros processos de fabricação, por exemplo, de outras engrenagens.

A descrição das ob-nets e das p-nets para o caso em estudo será utilizada como fundamento para a implementação do sistema.

6.2 Comportamento dos objetos: ob-nets

As redes próprias dos objetos, ob-nets, representam o comportamento dos objetos. A ob-net é o conjunto de estados e métodos de um objeto. Os estados e métodos das ob-nets não correspondem necessariamente aos estados e rotinas das p-nets. Os estados da ob-net estão relacionados ao objeto, enquanto que na p-net o estado se refere ao processo. Da mesma forma, o método modifica o estado de um objeto, enquanto que a rotina muda o estado do processo. A tabela a seguir dá uma visão geral dos estados e métodos associados a cada objeto.

Ob-net: peça		
Estados	EST_BR	Peça na sua forma bruta (forjado no estoque).
	AGD_MNT	A peça aguarda montagem no pallet.
	P_MNT	Peça montada no pallet aguarda o fim da montagem de outras peças no pallet.
	AGD_T	Aguarda início da atividade de transporte.
	T_PX_OP	Peça sendo transportada.
	P_AGD_OP	Aguarda início da próxima operação.
	AGD_DESMNT	Aguarda ser desmontada do pallet.
	OP	Peça sendo trabalhada por um agente.
	ESP	Peça em estado de espera programada.
	REF	Refugo
	RET	Destino da peça a ser re-trabalhada.
	AGD_MTRL	Aguardando metrologia.
	MTRL	Metrologia.
	PRONTA	Pronta para entrega.

Métodos	nova	Libera a peça bruta para montagem no pallet.
	mnt	Montagem da peça no pallet.
	fim_mnt	Fim da montagem da peça no pallet.
	desmnt	Desmontagem.
	transp_px_op	Transporte da peça para a próxima operação.
	chegada	A peça chega ao local da próxima operação.
	inic_op	Início da operação.
	fim_op	Fim da operação.
	esp_inic	Inicio de um período de espera programada.
	esp_fim	Fim do período de espera.
	inic_mtrl	Início da operação de metrologia / inspeção.
	aprovado	Aprovado na inspeção / metrologia.
	refugo	Peça fora das especificações, sem reaproveitamento.
	retrabalho	Peça desaprovada, contudo passível de re-trabalho.
Ob-net: pallet		
Est.	PL_DISP	Pallet disponível para montagem das peças.
	AGD_T	Aguarda para ser transportado, com peças.
	PL_AGD_DISP	O pallet aguarda tornar-se disponível para montagem.
	PL_AGD_MNT	Aguarda fim da montagem de peças.
	AGD_DESMNT	Aguarda para poder liberar as peças montadas.
	T_PX_OP	Sendo transportado, levando peças para próxima operação.
	AGD_ESP	Aguarda ser armazenado para o período de espera.
	ESP	Pallet espera o fim do período de espera.
	desmnt	Desmonta todas as peças do pallet.
	pl_disp	Disponibiliza o pallet para a montagem de peças.
	mnt	Montar uma peça no Pallet.
	n_cheio	O pallet ainda não está completamente cheio.
	fim_mnt	Fim da montagem de peças no pallet.
	esp_inic	Espera o início do período de espera.

	esp_fim	Fim do período de espera.
	transp_px_op	Pallet sendo transportado para a próxima operação.
	chegada	Pallet chega com as peças ao destino do transporte.
Ob-net: Agente		
Est.	AG_DISP	O Agente pode ser utilizado para realizar uma operação.
	AG_INDISP	O Agente já se encontra em uso.
Mét.	usar	Torna o Agende indisponível.
	liberar	Torna o Agente disponível.

As ob-nets correspondentes aos objetos da tabela anterior são melhor compreendidas em sua forma gráfica. As Figuras a seguir mostram as ob-nets correspondentes ao pallet, ao agente e à peça.

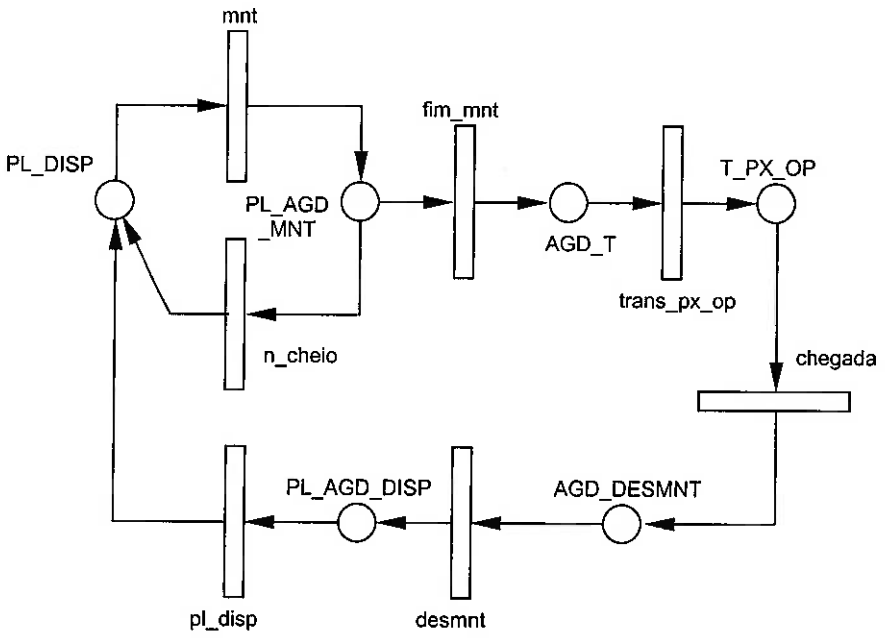


Figura 6.1 ob-net do pallet

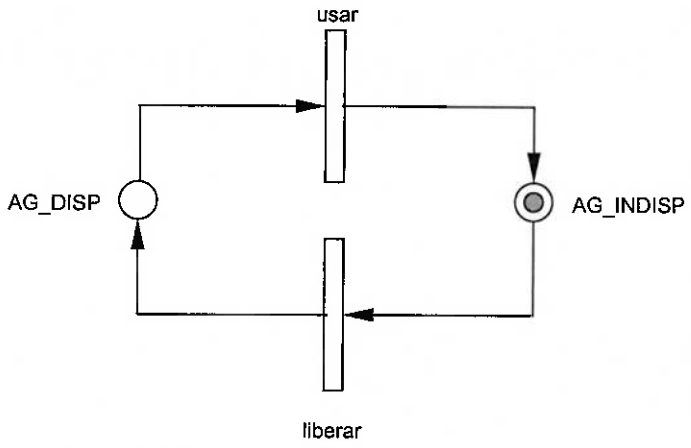


Figura 6.2 ob-net do agente

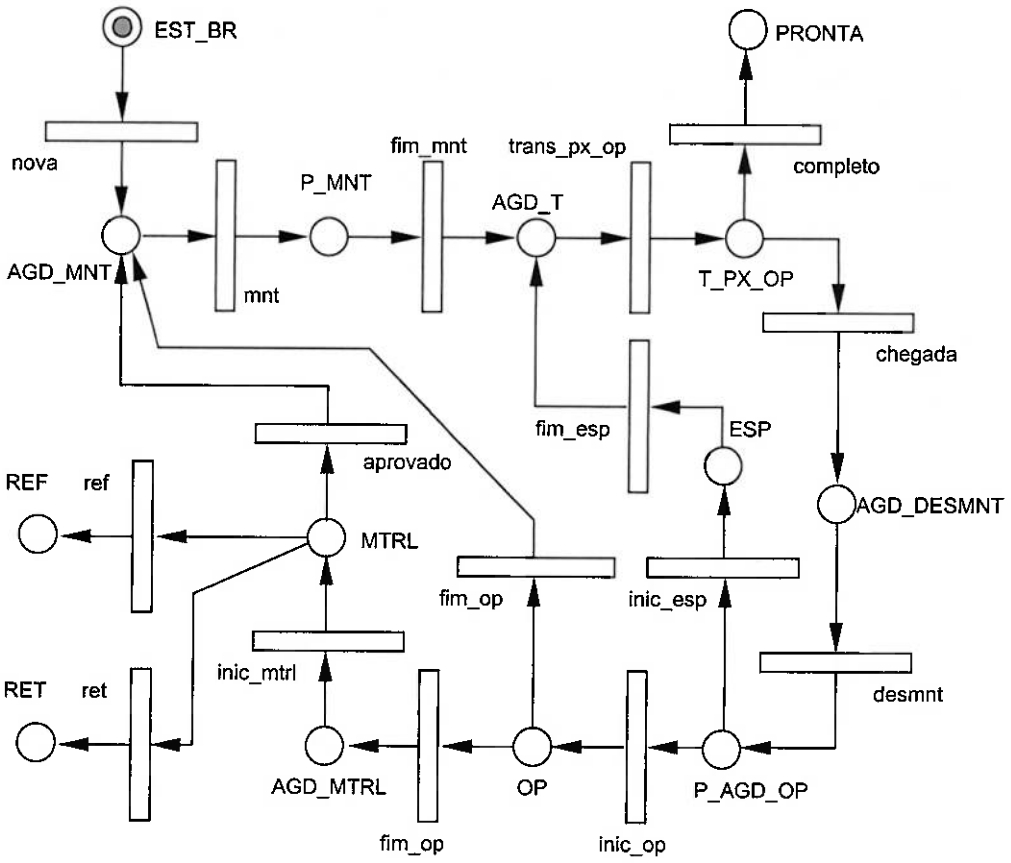


Figura 6.3 ob-net da peça

6.3 Dinâmica dos processos: p-nets

6.3.1 Blocos básicos: operações

Para o estudo de caso, será elaborada uma p-net para a fabricação de engrenagens de um tipo específico (Engrenagem Tipo "A"). Para engrenagens com processos semelhantes, o procedimento para construção da p-net aqui utilizado também é válido. Porém, cada p-net caracteriza um processo de fabricação específico.

A partir do estudo realizado sobre a organização da planta e seus componentes, foram identificadas seqüências de atividades que se repetem dentro dos processos. Estas atividades serão organizadas em blocos, denominados operações, que, posteriormente serão agrupados para formar a p-net do exemplo. As operações podem ser agrupadas para formar p-nets correspondentes a outros processos de fabricação

A nomenclatura dada aos lugares das redes de petri é específica de cada lugar da p-net do processo completo. Por exemplo, o bloco responsável pelo transporte de peças sem pallet possui um lugar denominado T_PX_OP, na descrição do bloco. Contudo, cada aparição do bloco na rede de processo possui um lugar T_PX_OP específico. Assim, na rede com o processo completo será utilizado um índice após a denominação do lugar, especificando qual *view* do processo está sendo tratada.

A mesma regra de nomenclatura também será utilizada para a designação das transições, mensagens e eventos associados a cada operação. A seguir, serão apresentadas as operações básicas para construção das p-nets.

Uma outra vantagem de se representar seqüências de atividades por meio de operações é a possibilidade de encapsular certas propriedades da rede que são próprias de cada bloco. Por exemplo, o tempo da duração de uma operação de montagem pode ser uma propriedade desta operação.

6.3.2 *Operação: Montagem de pallet*

O bloco de montagem recebe cada peça que aguarda a montagem (AGD_MNT) em quantidades unitárias (1.**p**) e as monta em pallets **pl** com capacidade *c*. Os pallets montados aguardam em um buffer (peças e pallets aguardando transporte: AGD_T) que os armazena até ser realizada a atividade de transporte para a próxima operação. A Figura 6.4 mostra a representação esquemática deste bloco. A Figura 6.5 mostra a p-net correspondente ao bloco de montagem.

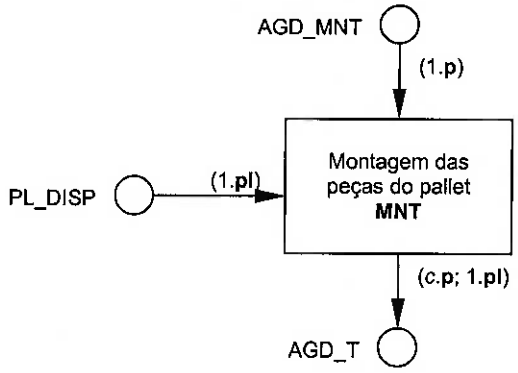


Figura 6.4 bloco para montagem de pallet

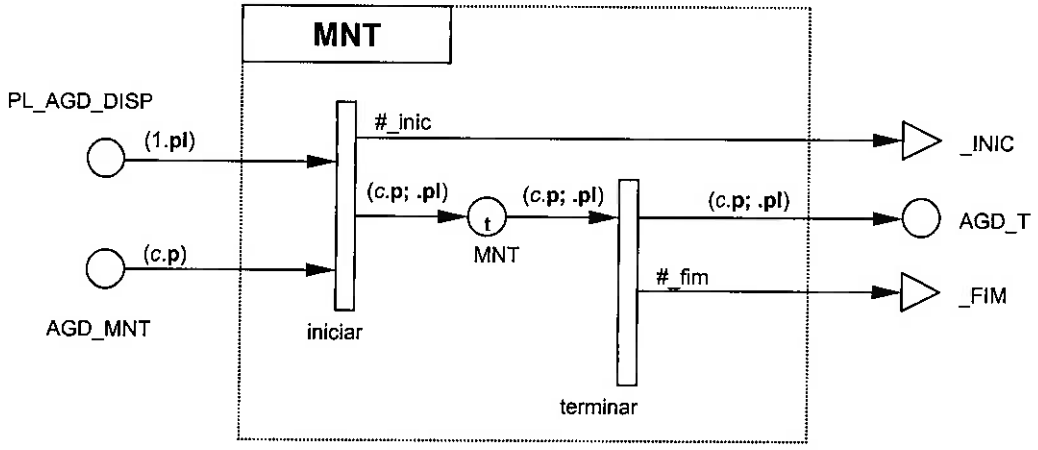


Figura 6.5 p-net para montagem de pallets

6.3.3 Operação: Desmontagem de pallet

Este bloco recebe um conjunto com pallet e peças, que será separado. O pallet é devolvido para o sistema. A transição *desmnt* leva as peças e o pallet do estado AGD_DESMNT (aguardando desmontagem) para os estados AGD_OP (peça aguardando operação) e PL_AGD_DISP (pallet aguardando disposição) respectivamente. O bloco representativo e a p-net correspondente são mostrados na Figura 6.6 e na Figura 6.7.

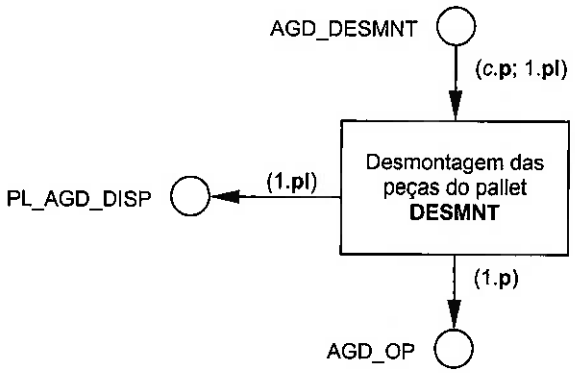


Figura 6.6 bloco para desmontagem de pallets

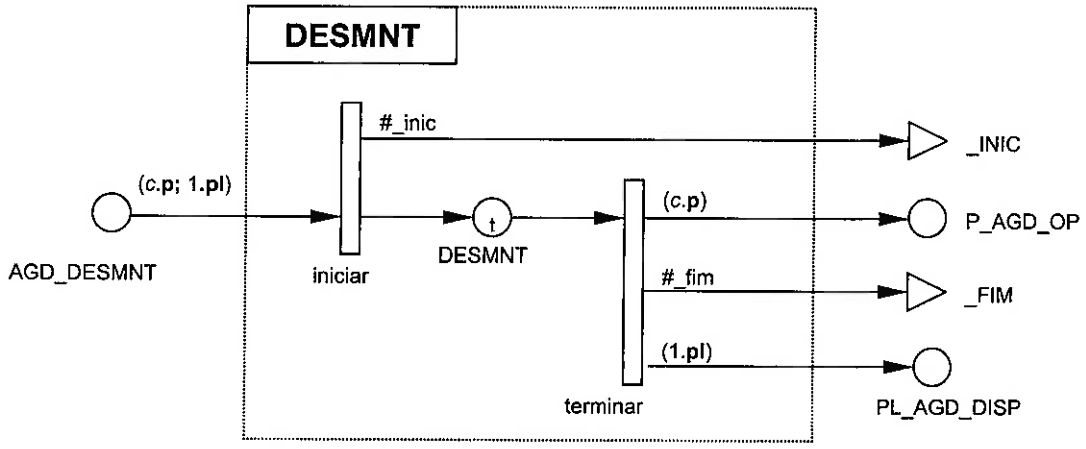


Figura 6.7 p-net para desmontagem de pallets

6.3.4 *Operação: Transporte de peças sem pallet*

Este bloco representa a modificação do estado da peça no que se refere à sua localização na planta. O bloco é responsável, portanto, por representar o transporte de um conjunto com um número *t* (capacidade do lugar T_PX_OP) de peças de um centro de trabalho para outro, sem que ela esteja montada em um pallet. O Bloco informa ao sistema o tempo gasto para transporte, indiretamente, por meio das mensagens de início e fim da atividade (*#t_inic* e *#t_fim*). A peça sai do estado AGD_T_OP (aguardando transporte para outra operação) para o estado T_PX_OP (em transporte para próxima operação), através da transição *trans_px_op*. A transição *chegada* indica o fim da operação de transporte. A peça muda para o estado P_AGD_OP (aguardando operação).

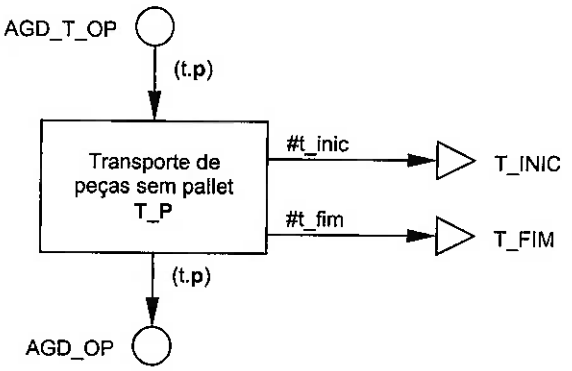


Figura 6.8 bloco para transporte de peças sem pallet

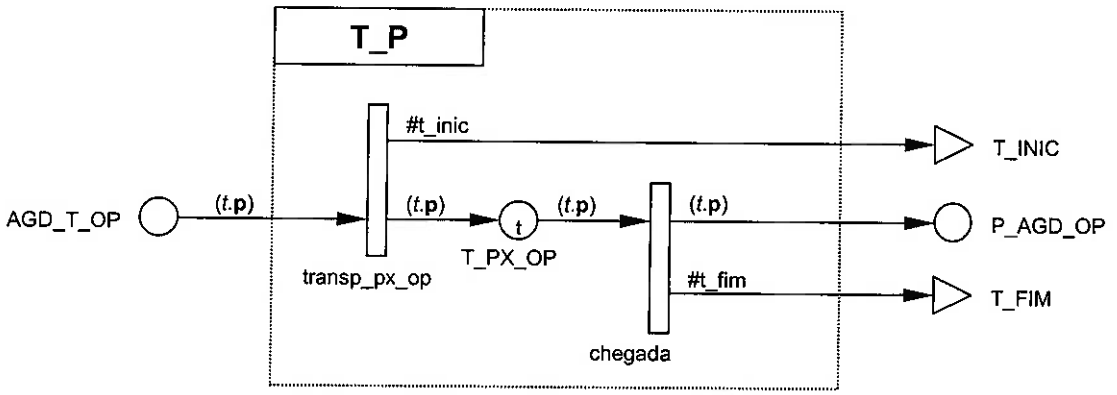


Figura 6.9 p-net para transporte de peças sem pallet

6.3.5 *Operação: Transporte de pallets com peças*

Este bloco é semelhante ao transporte de peças sem pallet. Ambos o pallet e a peça atravessam os estados AGD_T e T_PX_OP. Contudo, após a chegada no destino, o conjunto adquire o estado AGD_DESMNT (aguardando desmontagem), pois as engrenagens serão primeiramente separadas dos pallets para depois sofrerem a próxima operação. A quantidade de pallets transportada a cada evento de transporte é *tp* – que também é a capacidade do lugar T_PX_OP.

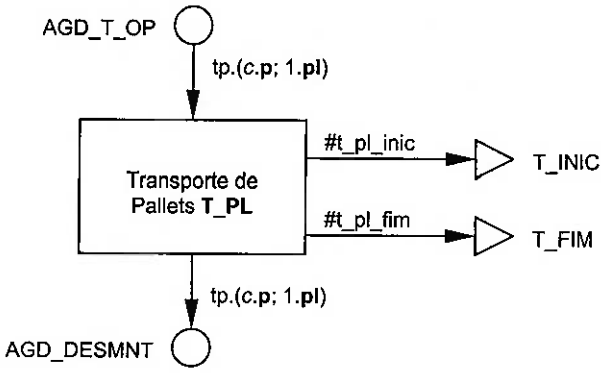


Figura 6.10 bloco para transporte de peças com pallet

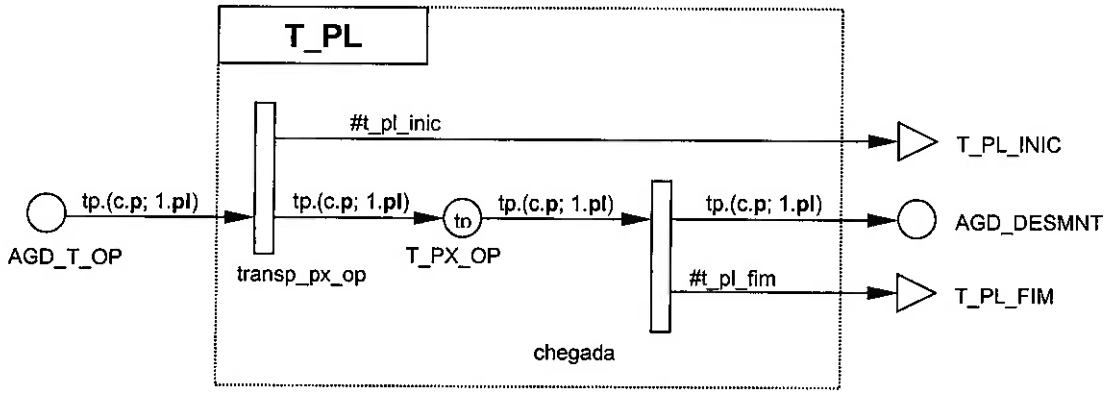


Figura 6.11 p-net para transporte de peças com pallet

6.3.6 *Operação: Processamento*

Para cada atividade da manufatura (usinagem, tratamento térmico, etc.) existe uma operação *processamento* correspondente. O peso *d* designa o número de peças processadas a cada ciclo de operação – por exemplo, a capacidade para peças em um dispositivo de fixação para usinagem. O conjunto de peças passa do estado AGD_OP para OP (em operação) e, em seguida, para AGD_(...), pode ser AGD_MNT ou AGD_T. Durante a operação, o agente passa juntamente com a peça do estado AG_DISP (disponível) para OP (em operação).

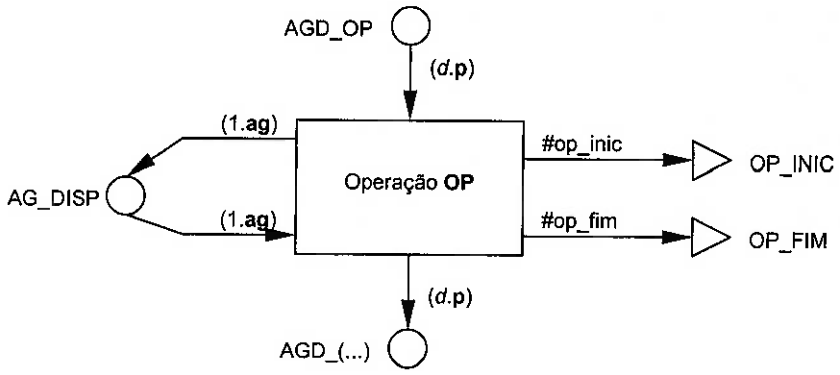


Figura 6.12 bloco Processamento

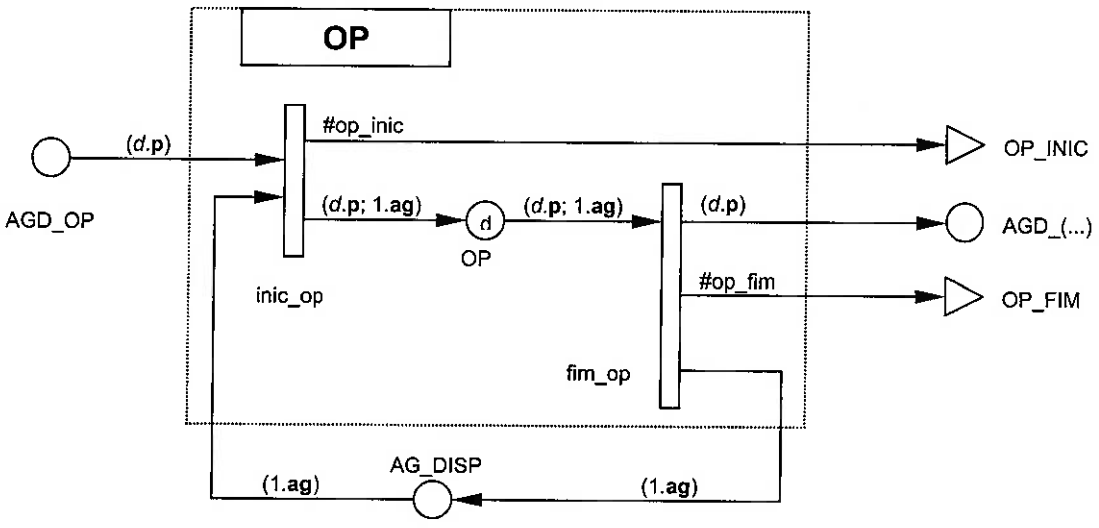


Figura 6.13 p-net da operação Processamento

6.3.7 *Operação: Descanso*

A operação de descanso, após o tratamento térmico, assemelha-se à operação de transporte, pois não utiliza um agente. Contudo, o descanso apresenta um tempo de execução não desprezível. O descanso ocorre com as peças dentro de pallets, no estado ESP (espera). O número de peças capazes de ocupar o lugar destinado para descanso é a capacidade de ESP, dada por *e*.

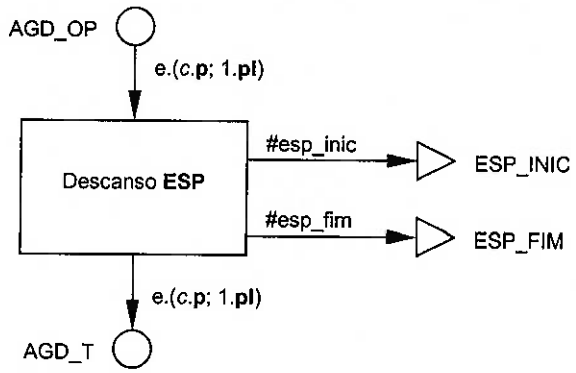


Figura 6.14 bloco Descanso

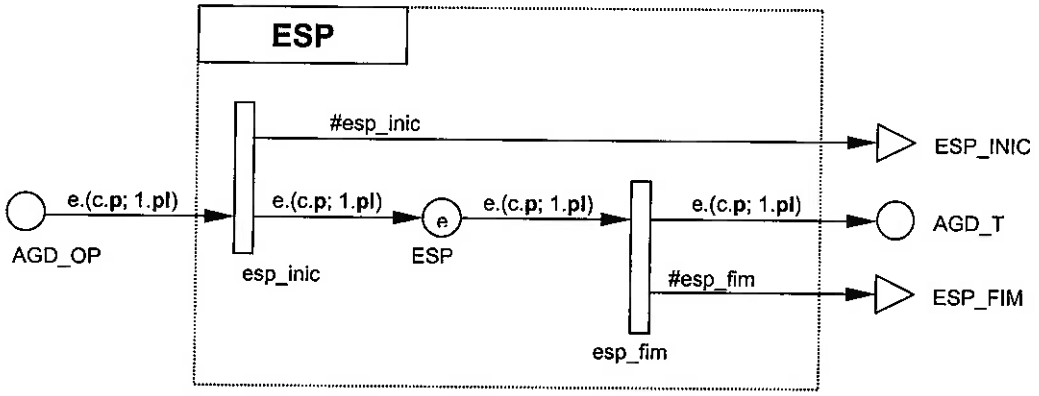


Figura 6.15 p-net da operação Descanso

6.3.8 Operação: Metrologia

Ao final da operação de metrologia, a peça será qualificada como aprovada (AGD_), refugo (REF) ou re-trabalho (RET). Uma peça destinada a re-trabalho será transportada para o processo anterior ou outro processo. Neste bloco, o agente passa do estado AG_DISP (disponível) para OP (em operação), junto à peça, como durante a realização de qualquer outra operação.

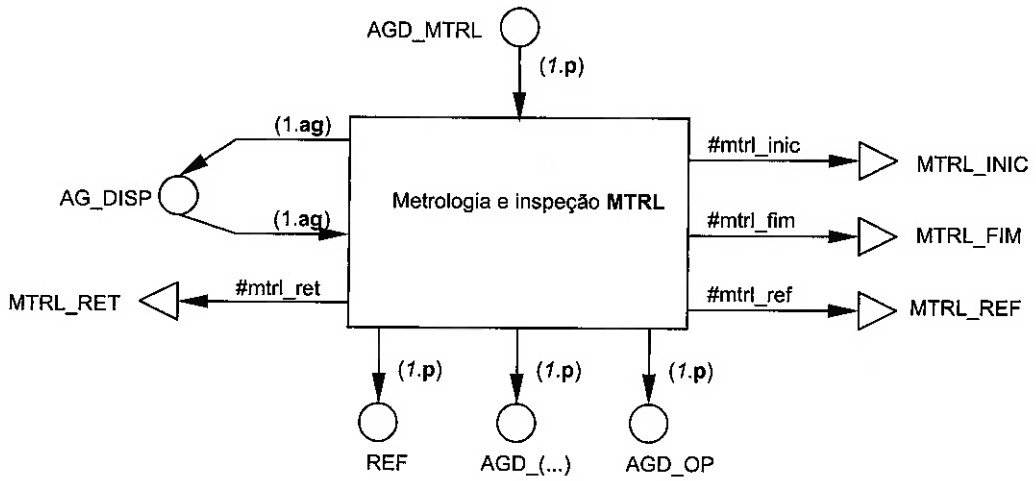


Figura 6.16 bloco Metrologia

A peça é recebida pela operação no estado AGD_MTRL. O agente processa a peça, ambos em estado OP. O fim da operação metrologia resulta em uma das transições: *aprovado*, *re_trabalho* ou *refugo*. A respectiva mensagem é gerada, indicando qual o novo estado da peça. A figura a seguir mostra a p-net para a operação de metrologia.

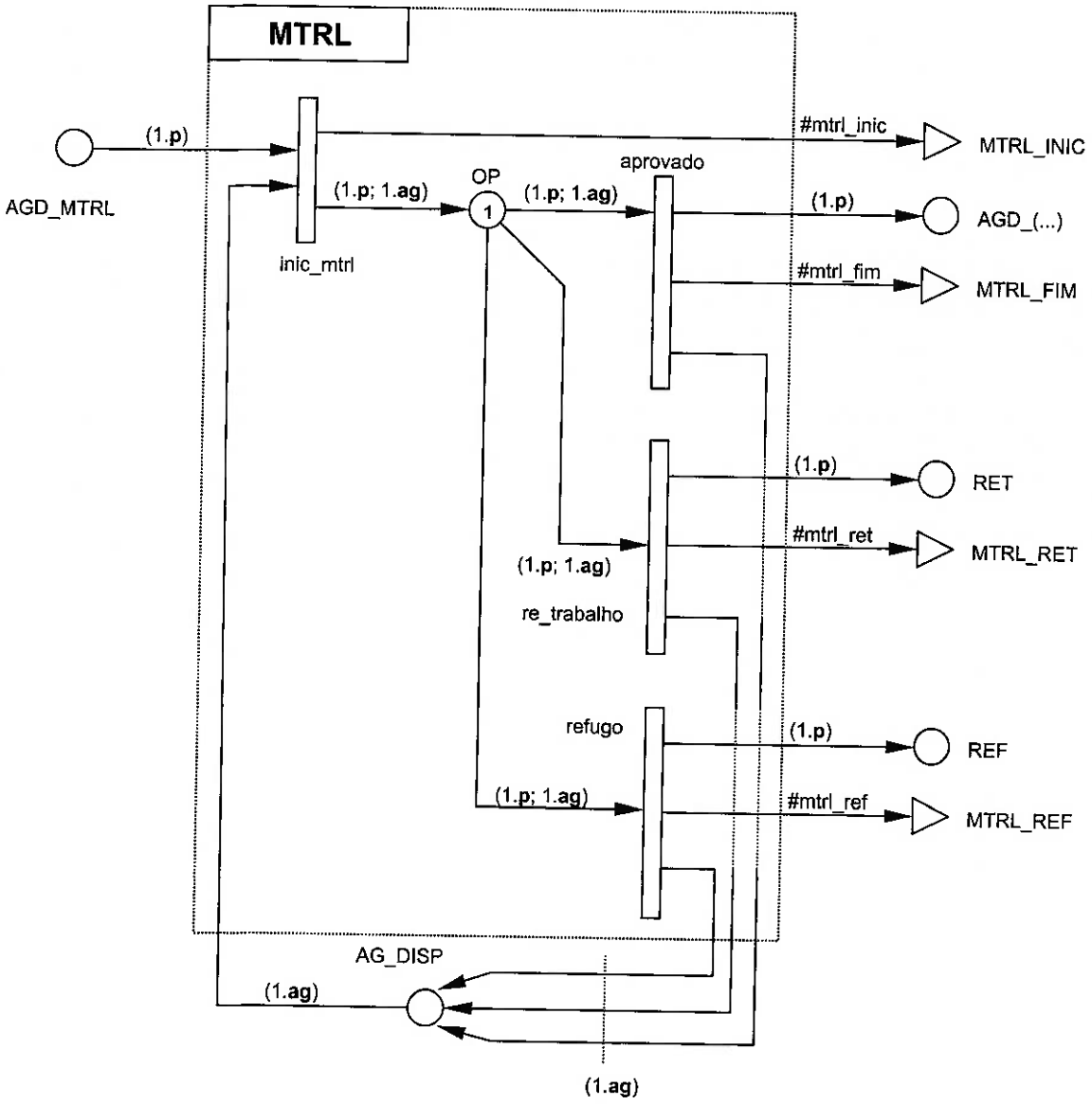


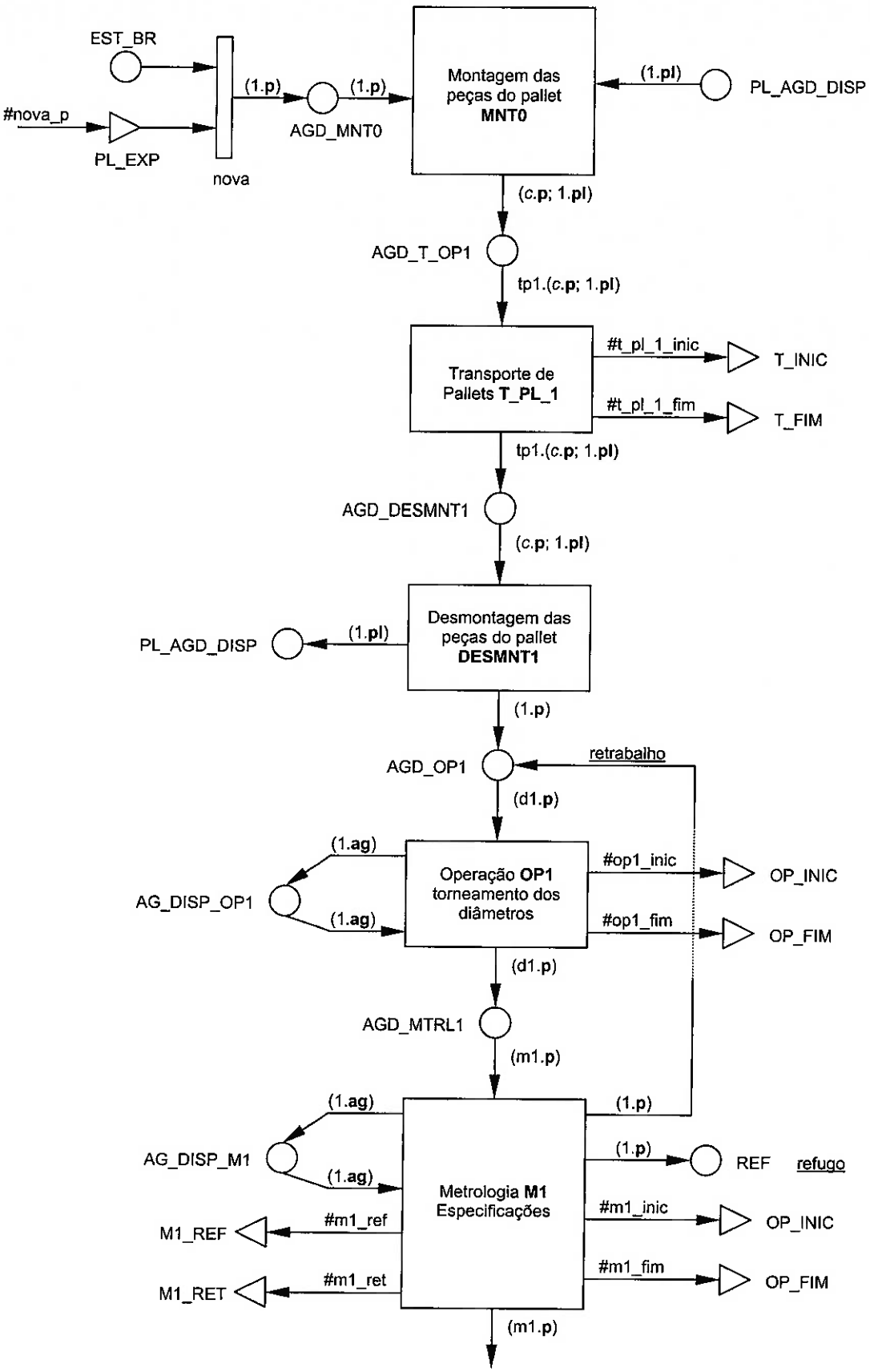
Figura 6.17 p-net do bloco Metrologia

6.3.9 p-net: fabricação de engrenagens do tipo "A"

A p-net a seguir utiliza os blocos descritos para representar o processo de fabricação de uma engrenagem do tipo "A", utilizado como exemplo. Para sua construção foram utilizados os blocos básicos definidos para os processos da planta.

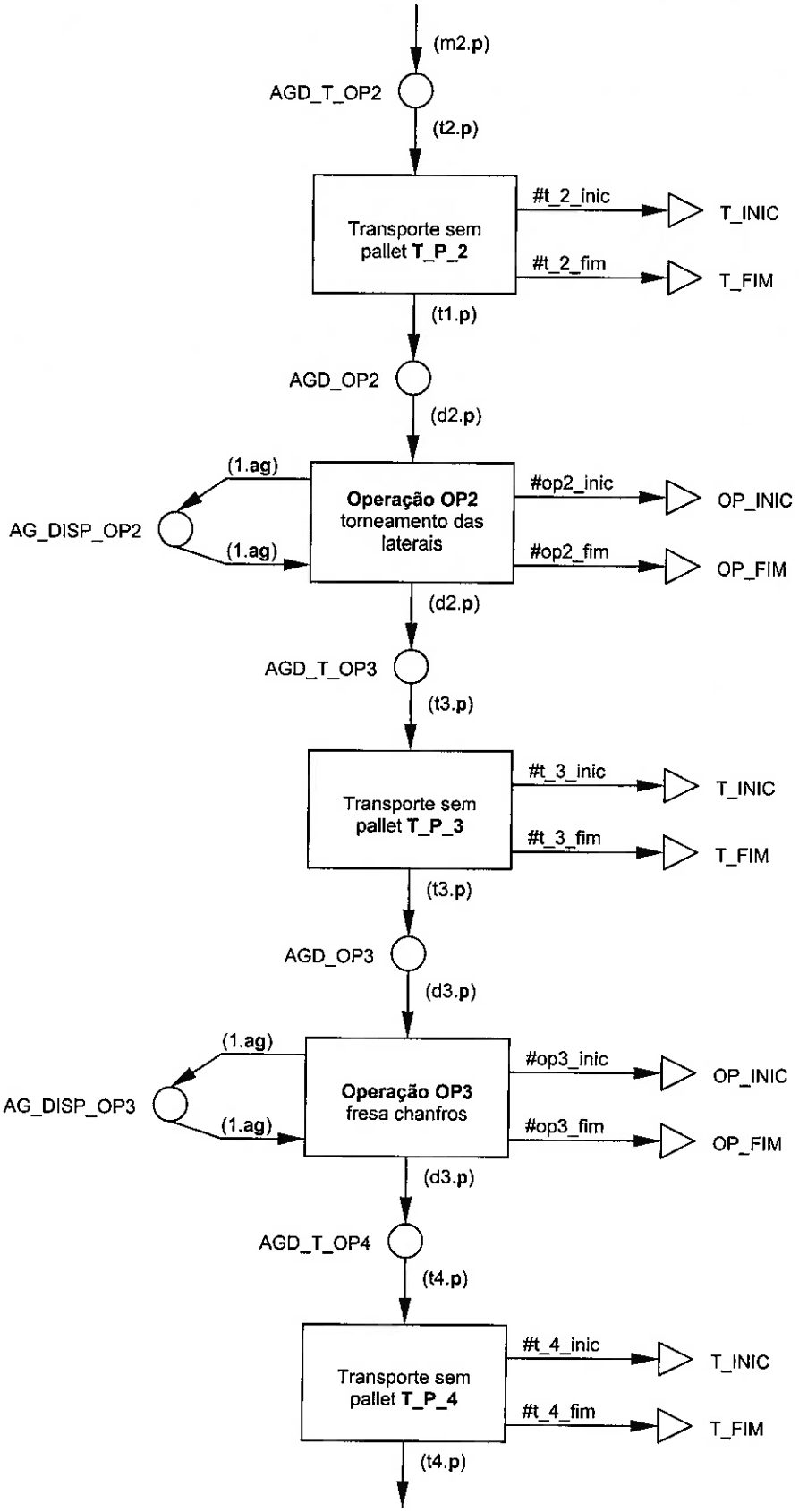
p-net - 1ª parte: processo a verde

(parte 1/8)



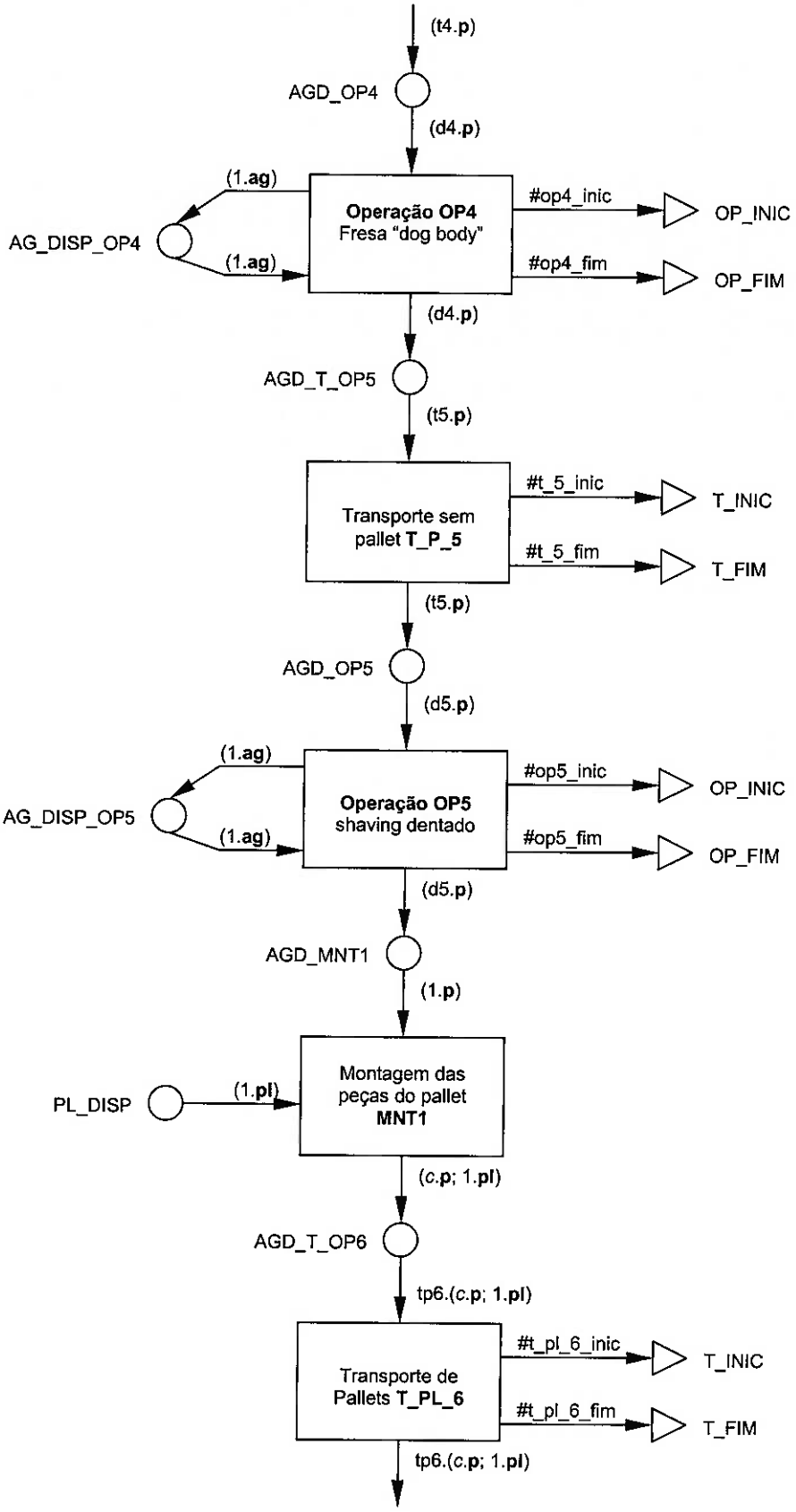
p-net - 1ª parte: processo a verde

(parte 2/8)



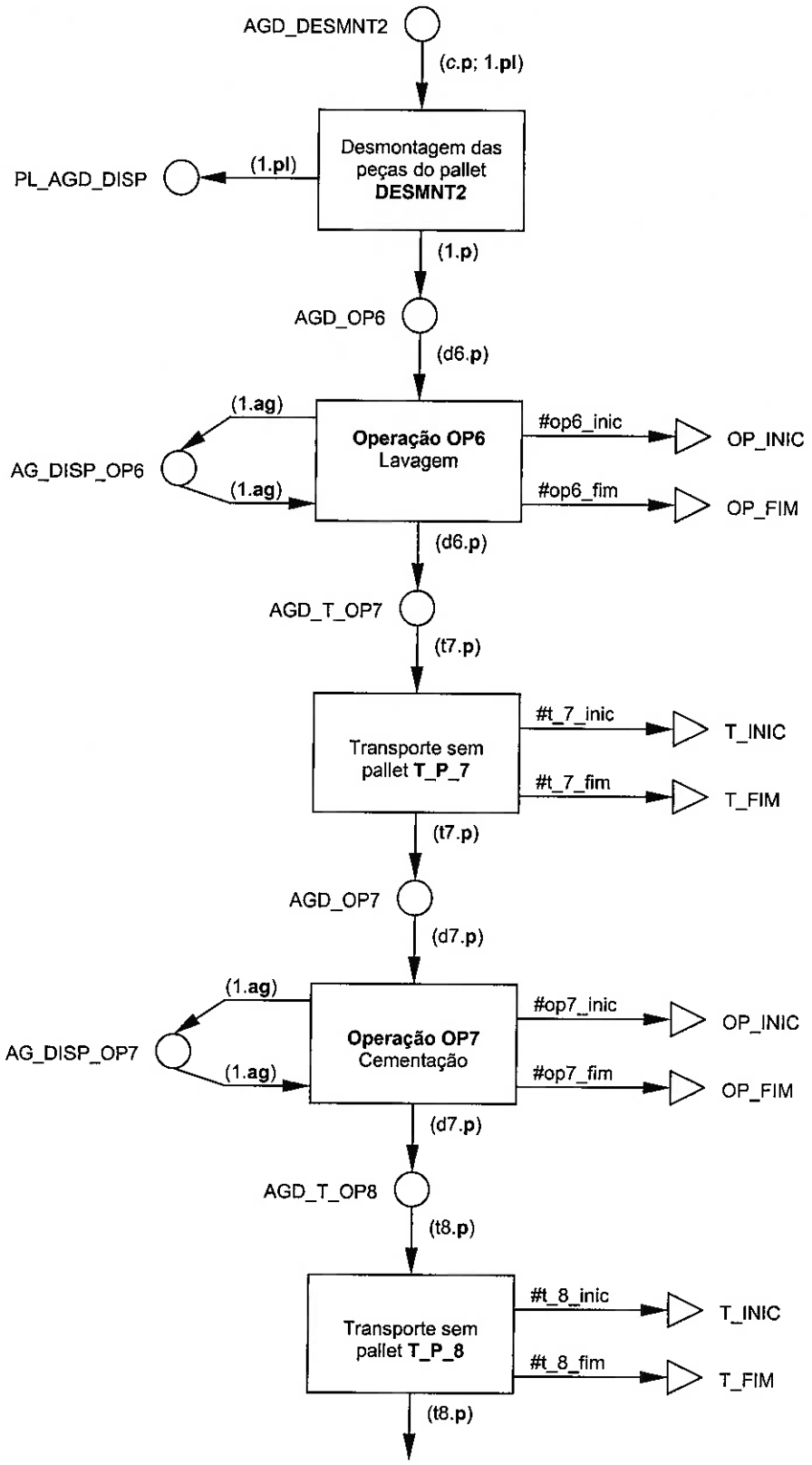
p-net - 1ª parte: processo a verde

(parte 3/8)



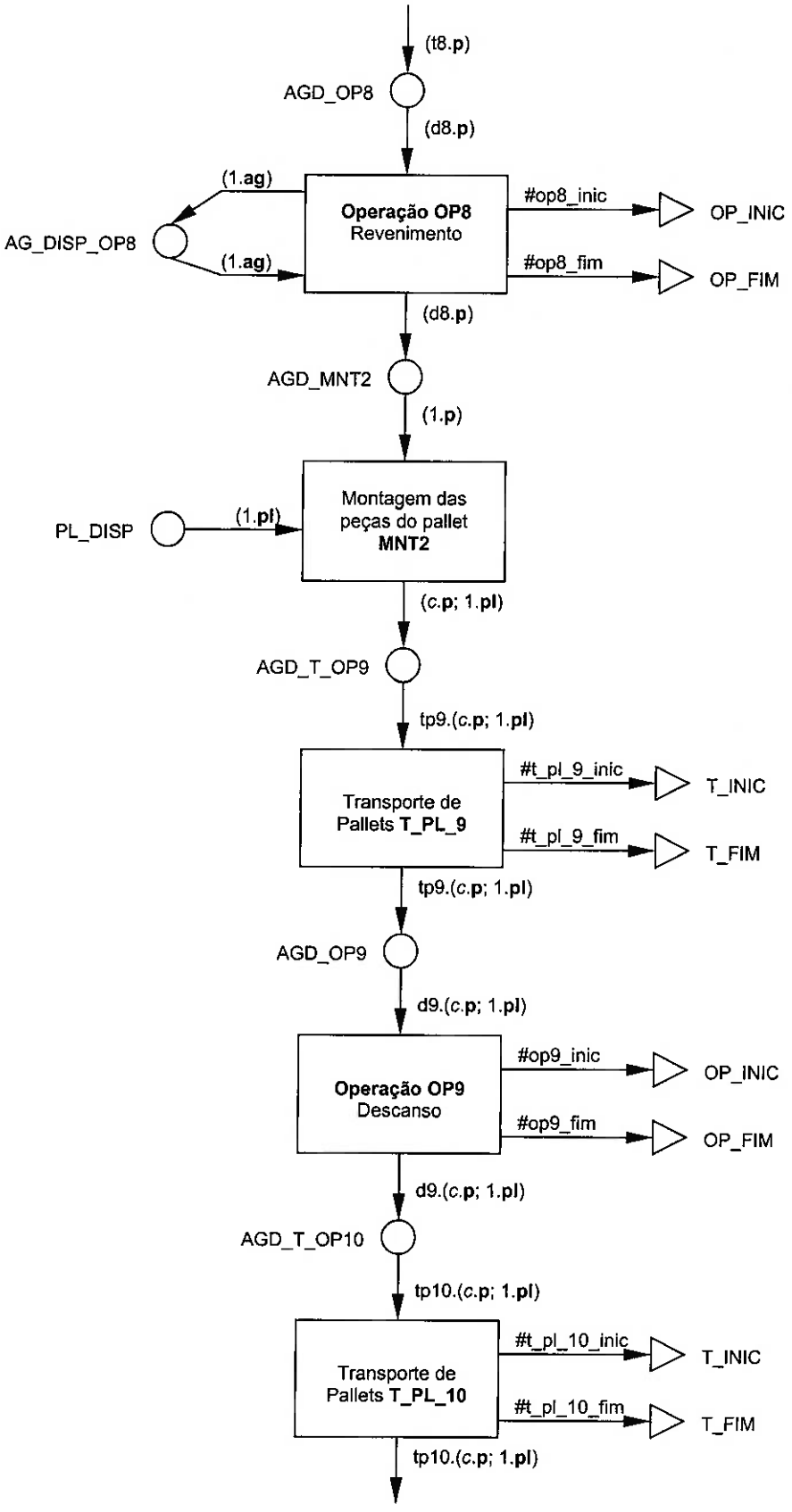
p-net - 2ª parte: tratamento térmico

(parte 4/8)



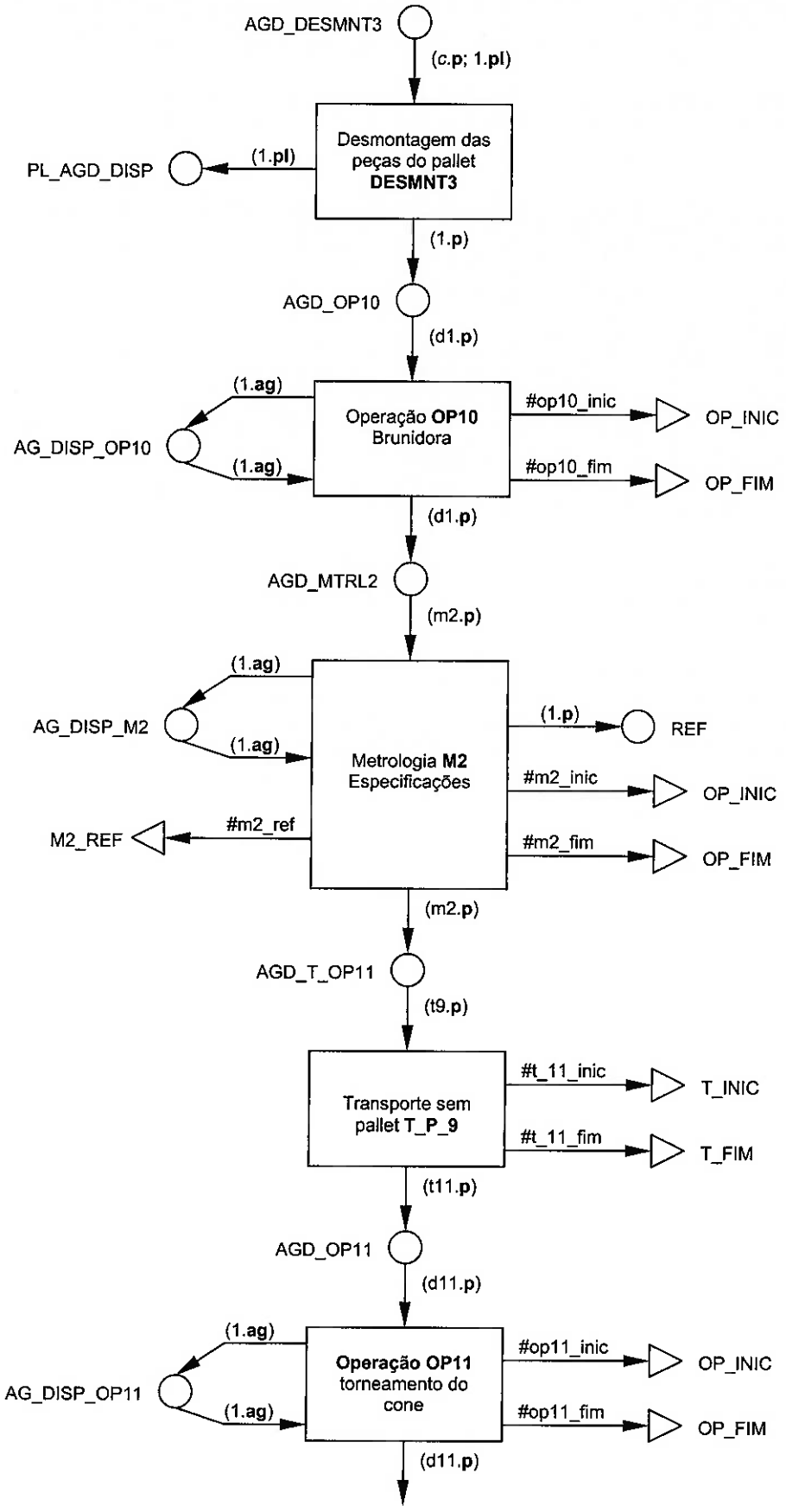
p-net - 2ª parte: tratamento térmico

(parte 5/8)



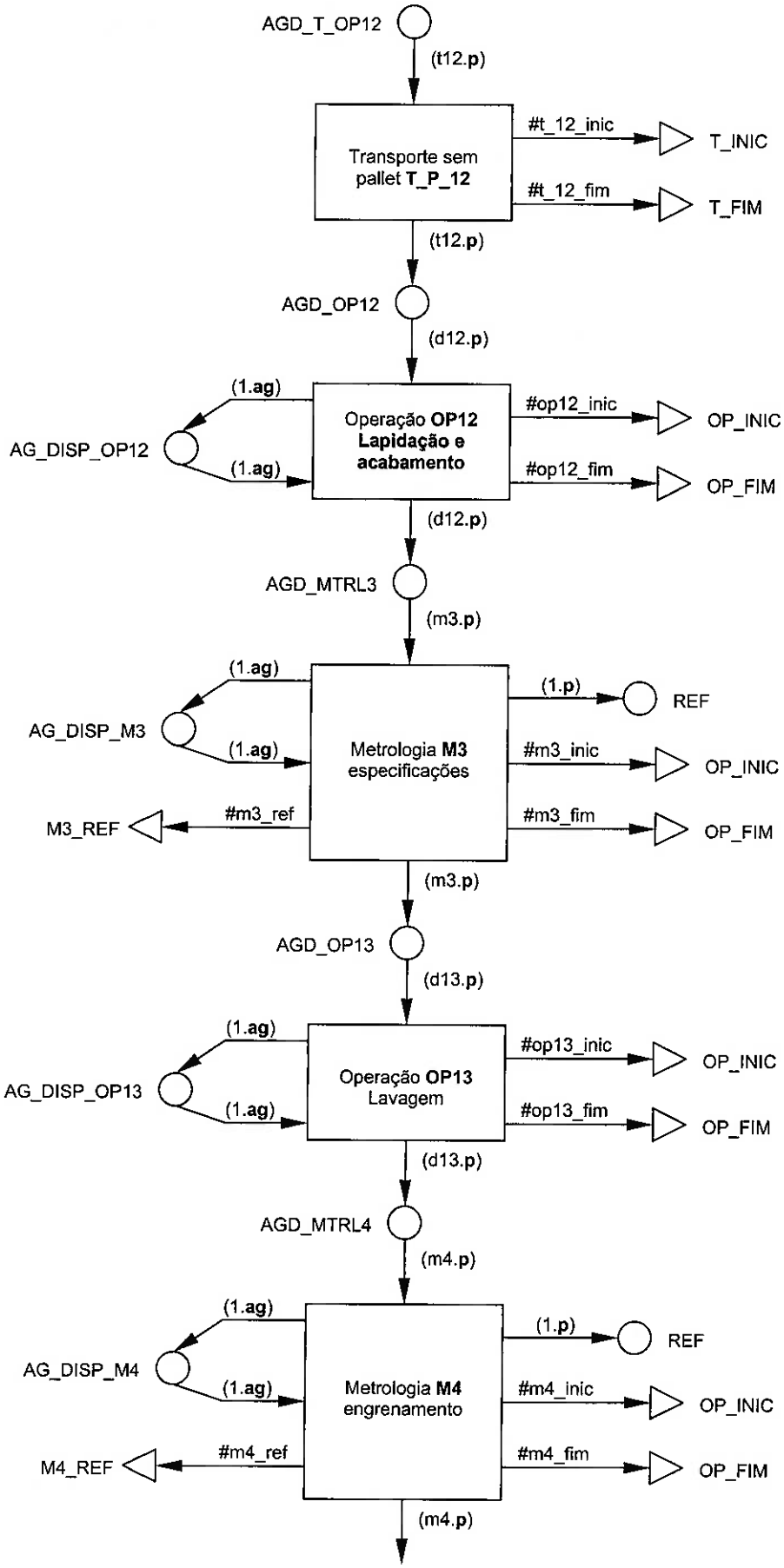
p-net - 3ª parte: usinagem final

(parte 6/8)



p-net - 3ª parte: usinagem final

(parte 7/8)



p-net - 3ª parte: usinagem final (parte 8/8)

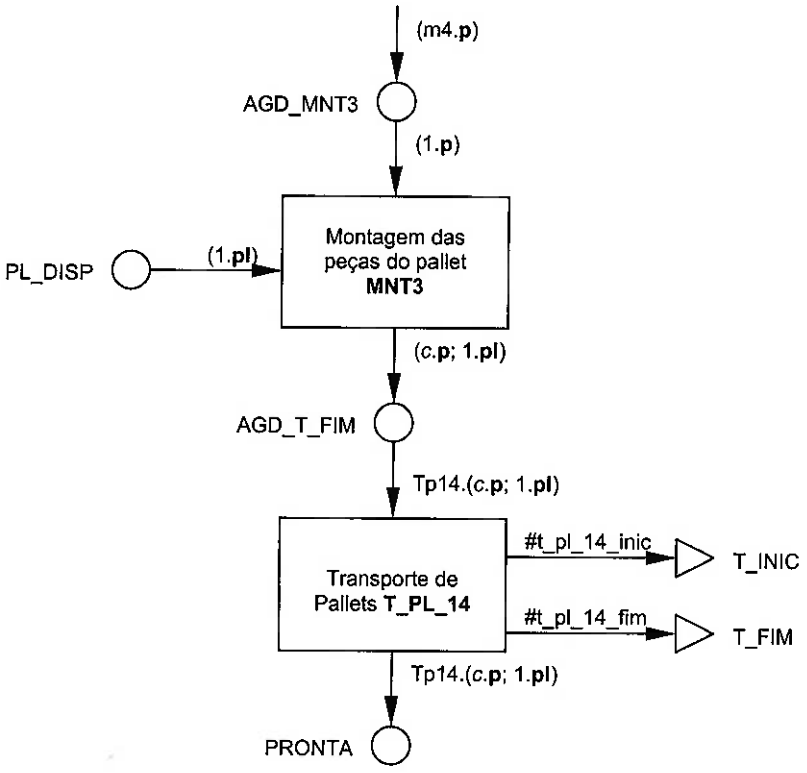


Figura 6.18 p-net do processo para a fabricação de engrenagens do tipo "A"

6.4 Resumo do projeto conceitual

Nesta etapa do projeto foram analisados dois conjuntos de elementos componentes do sistema: a rede, que simula a dinâmica dos processos no caso em estudo, e os elementos modificados através desta rede – agentes, peças e pallets. Estes últimos são os tokens destas redes, sendo que a mudança de estados da planta está associada à mudança de estado destes objetos.

7. Sistema de informação

7.1 Objetivos do sistema

A última etapa do projeto é propriamente a implementação do sistema de informação. Esta fase está fundamentada no que foi desenvolvido através do projeto conceitual. A seguir, serão descritas as funcionalidades de cada componente de um sistema de informação completo, que poderia ser utilizado em uma aplicação real:

- Simulador da Planta;
- Interface com a Planta;
- Identificador de Tendência;
- Gerador de Relatórios de Estado;
- Repositório de dados.

Este trabalho está focado nos elementos do sistema que são responsáveis pela representação da dinâmica dos processos na manufatura. Portanto, dos módulos acima, esse projeto engloba a implementação por meio de software dos seguintes componentes:

- Repositório de dados;
- Simulador da planta;
- Identificador de tendência;

A Figura 7.1 apresenta uma visão geral do sistema de informação completo e das respectivas trocas de dados entre seus componentes. Embora não seja objetivo deste projeto implementar o gerador de relatórios de estados e a interface com a planta, a funcionalidade destes componentes será descrita a seguir.

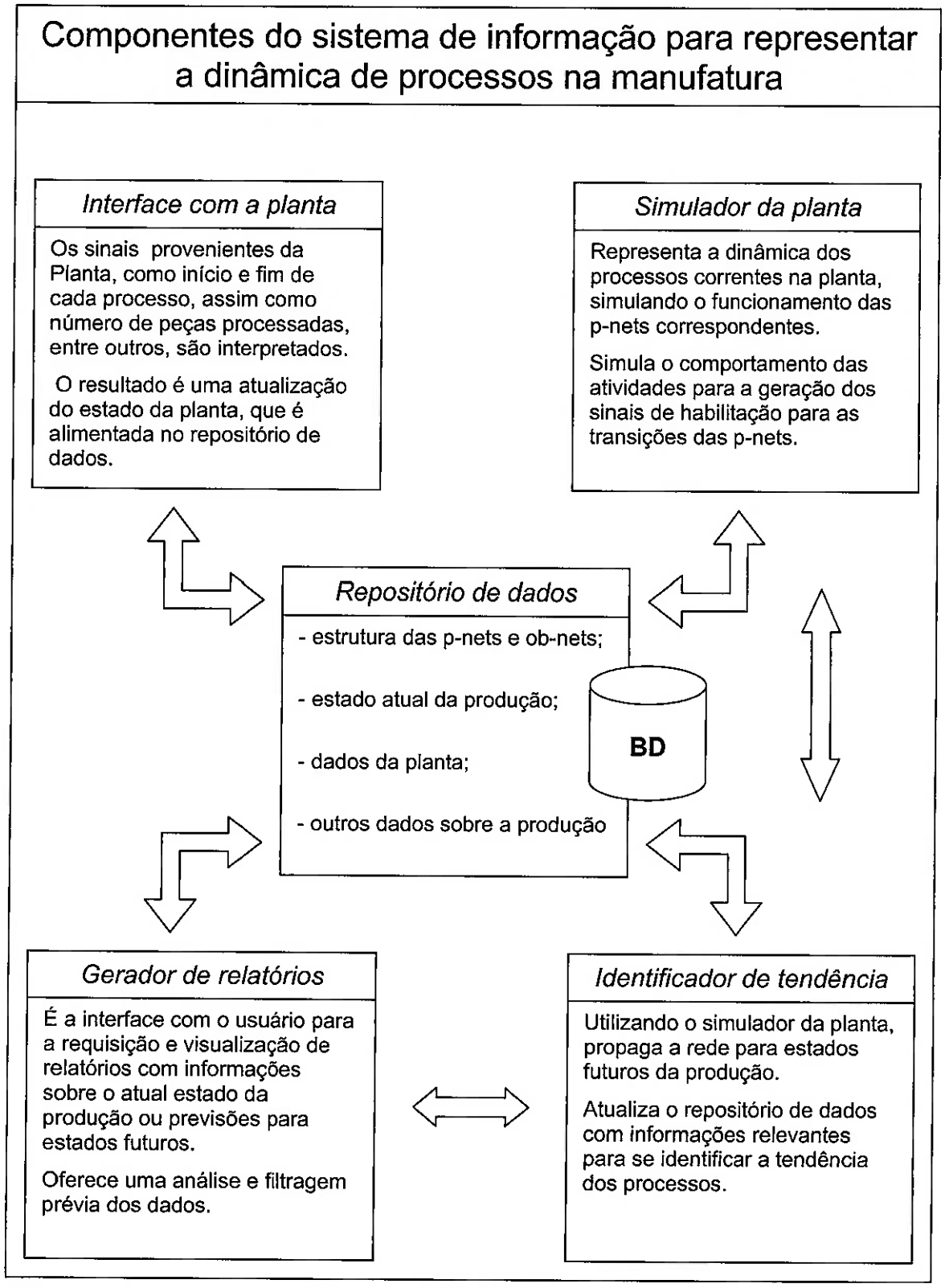


Figura 7.1 componentes do sistema de informação

7.2 Componentes do sistema

7.2.1 Interface com a planta

Esse componente tem a função de adquirir e interpretar os sinais provenientes do chão-de-fábrica, gerando assim uma atualização constante do estado da planta no repositório de dados. Este projeto não está focado na implementação de tal interface, mas é importante descrever a forma através da qual essa funcionalidade é utilizada.

Uma das condições necessárias para a simulação e representação da planta segundo o sistema proposto é que os sinais provenientes do chão-de-fábrica sejam suficientes para descrever e atualizar o conhecimento sobre o estado atual da planta. Ou seja, essas informações não transmitem o estado, mas sim informações sobre as mudanças neste estado. O papel do componente responsável pela interface com a planta é justamente interpretar essas informações e atualizar o repositório de dados com as decorrentes mudanças no estado da planta. A cada atualização dos sinais provenientes do chão de fábrica, a interface com a planta gera uma nova descrição para o estado de produção da fábrica.

Um exemplo, já anteriormente citado, é o conjunto de sinais informando o início e o fim de cada processo. Quando uma máquina termina de processar um lote de peças, essa informação é suficiente para que o componente de interface com a planta atualize no banco de dados os novos estados relacionados. As peças passam então para o estado seguinte, após o processamento. A máquina torna-se novamente disponível.

Desta forma, é importante ressaltar que a atualização dos estados só é possível conhecendo-se os estados anteriores. A interface com a planta utiliza-se tanto das informações em tempo real provenientes da planta, quanto das informações sobre estados anteriores disponíveis no repositório de dados.

7.2.2 Simulador da Planta

É o componente capaz de simular a dinâmica dos processos do sistema com base nas p-nets relacionadas à fabricação de cada tipo de engrenagem. A fase de implementação deste projeto consiste basicamente na criação da estrutura de classes para o software capaz de simular a planta.

A simulação é uma tarefa realizada à parte das demais. Ela é capaz de obter do repositório de dados tanto uma cópia do estado atual da planta, como também da própria estrutura das redes que representam a planta. A função básica da simulação é estabelecer uma relação entre estas redes, o estado da planta e o tempo. Assim, a simulação é capaz de interpretar e processar a rede até um período seguinte no tempo, gerando um novo estado, mais precisamente, o próximo estado.

Simulando-se estados sucessivos no tempo é possível obter os estados futuros da planta, segundo o modelo utilizado. Essa funcionalidade é utilizada de forma coordenada pelo identificador de tendência, outro componente do sistema de informações, para prever estados futuros e tempos necessários para executar uma determinada tarefa. Por exemplo, a fabricação completa de um lote de peças.

7.2.3 Identificador de tendência

Esse componente utiliza a funcionalidade oferecida pelo simulador da planta para simular as transições para estados futuros. Assim, é possível prever uma seqüência futura de estados que se assemelhe à realidade, dentro das limitações impostas pelo modelo que descreve a dinâmica dos processos envolvidos. Com esse resultado, é possível identificar uma tendência de modificação do estado da produção para um tempo futuro.

O identificador de tendências trabalha com uma cópia do repositório de dados correspondente ao estado da produção. Assim, durante a simulação, os dados originais são mantidos e continuam a ser atualizados normalmente, enquanto que a cópia é trabalhada na memória. É possível, portanto, extrair informações como: tempos decorridos para se terminar cada lote de peças; utilização da capacidade da fábrica no período; eventuais atrasos que poderão ocorrer, etc.

7.2.4 Gerador de Relatórios de Estado

Esse componente é a ferramenta utilizada pela gerência ou supervisão da fábrica para acessar informações sobre o atual estado da produção ou sobre previsões para estados futuros.

É também dele a responsabilidade de fornecer uma interpretação para o estado de produção sob os diversos pontos de vista úteis para a gerência, como:

- Utilização atual da capacidade fabril, previsão de utilização futura;
- Previsões de atrasos nos processos;
- Identificação imediata de anomalias na produção, como refugos acima do previsto ou atrasos nos processos;
- Visualização da alocação dos recursos da fábrica entre os diversos processos correntes;
- Verificação do estágio de processamento de cada lote, cada pedido, entre outros.

A base para seu funcionamento é a interação com o repositório de dados para a leitura do atual estado da produção e das tendências geradas pelo identificador de tendências. Esse software também interage com o identificador para requisitar as avaliações de tendência desejada, como por exemplo: qual o possível estado da produção em 2 horas; ou quanto tempo levará para se processar um lote específico, etc.

7.2.5 Repositório de dados

É o banco de dados do sistema, disponibilizado pelo aplicativo que o gerencia. O repositório de dados armazena todas as informações utilizadas pelo sistema de informação, sendo também o intermediário para a troca de dados entre o chão de fábrica e os outros componentes do sistema. Estão representados e armazenados no banco de dados:

1. Dados que caracterizam o atual estado da produção, segundo o modelo entidade-relacionamento proposto para o sistema;
2. A estrutura de todas as redes do sistema, tanto com suas regras de transição e comportamentos associados, como com as descrições dos caminhos entre os lugares e as transições.
3. Dados característicos da planta em análise, como: máquinas disponíveis; tempos de realização de cada processo, entre outros.

8. Implementação

8.1 Estrutura do sistema

Este capítulo descreve as classes implementadas para desenvolver o software correspondente aos componentes especificados no capítulo anterior. O anexo A trás a listagem das classes descritas. O anexo B apresenta a estrutura do repositório de dados através das tabelas existentes e seus relacionamentos.

O software foi desenvolvido utilizando-se a linguagem Java e um banco de dados genérico. Como o acesso à base de dados é feita via JDBC, qualquer banco de dados com suporte a java serve para os propósitos do sistema.

O programa foi testado, utilizando-se para isso um processo constituído por uma p-net formada por quatro operações: desmontagem, processamento, metrologia e montagem, na ordem dada. O estado inicial da planta utilizada para o teste foi a rede vazia, contento apenas 5 conjuntos no lugar inicial. Cada conjunto com um pallet e cinco peças.

O projeto do repositório de dados segue paralelamente a estrutura das classes desenvolvidas. Para cada entidade do modelo da planta, existe também uma classe que a representa no software e uma entidade no banco de dados. Além disso, uma estrutura de classes fundamentais dá o suporte necessário para o desenvolvimento de novas classes que possam representar outros elementos da manufatura, sejam eles novos processos, operações ou diferentes tipos de tokens.

Todas as classes desenvolvidas são utilizadas pela classe simulação. Como todas as funcionalidades das redes estão encapsuladas nelas próprias, a implementação de novas classes para realizar diferentes tipos de simulações, além do que foi proposto, é perfeitamente realizável sem grande esforço de programação. A classe tendência faz uso de um objeto simulação para realizar um teste de tendência na planta que estiver armazenada do repositório de dados.

8.2 Classes abstratas

8.2.1 Classes abstratas e interfaces

Toda a estrutura de dados utilizada na implementação do sistema está fundamentada em um conjunto de classes abstratas e interfaces que definem os tipos fundamentais de objetos do sistema.

Para evitar os problemas relacionados a heranças múltiplas, tipos comuns podem ser definidos por interfaces. A implementação de uma interface por uma classe permite, portanto, que esta herde as características de uma outra superclasse. Assim, tipos comuns podem ser implementados em classes que herdem diferentes superclasses.

As classes abstratas não podem ser instanciadas diretamente. Elas servem como modelo para a criação de novas classes que herdem suas características e, obrigatoriamente, implementem os métodos abstratos que definem tais superclasses. Ou seja, as classes abstratas também especificam um conjunto essencial de métodos, que devem ser implementados nas classes filhas, de acordo com suas próprias particularidades.

Os objetos reais do sistema precisam, portanto, ser instanciados a partir de classes concretas. A estrutura fundamental de classes abstratas e interfaces dá o subsídio necessário para que as classes concretas possam ser implementadas da forma devida, minimizando ainda o esforço de desenvolvimento. A Figura 8.1 mostra a convenção utilizada nos diagramas para representar classes e interfaces.

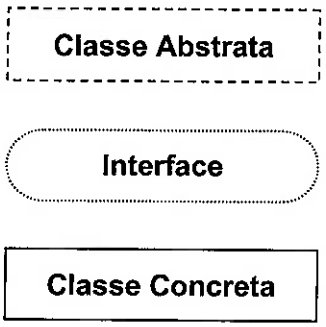


Figura 8.1 convenção utilizada para representar classes e interfaces

A estrutura fundamental de classes está representada na Figura 8.2. As setas apontam para as superclasses herdadas ou interfaces implementadas por cada classe.

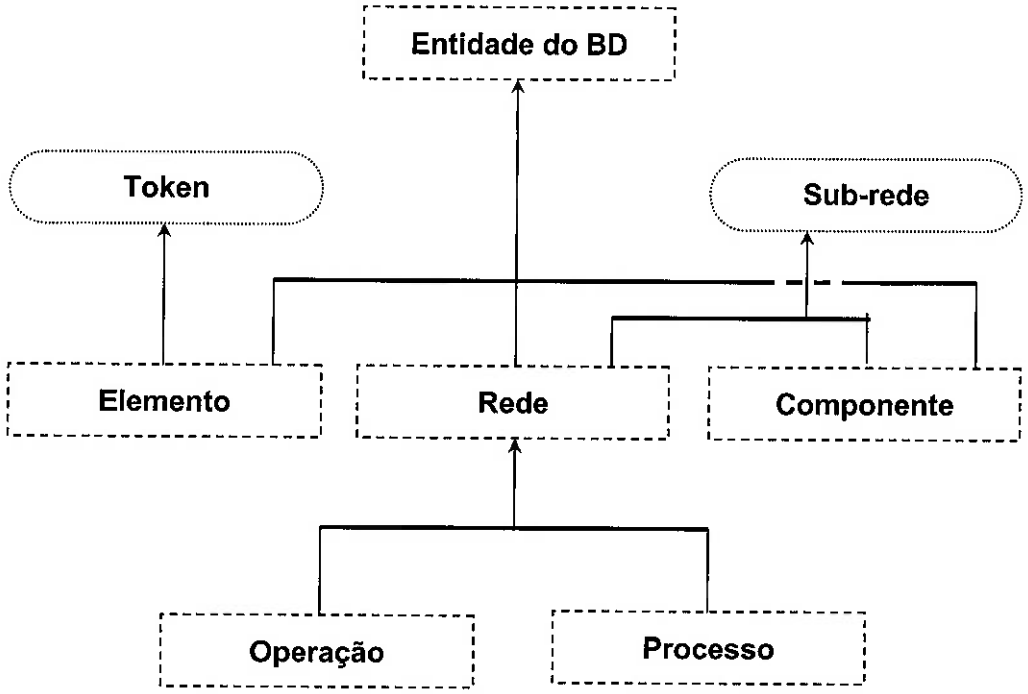


Figura 8.2 estrutura fundamental das classes do sistema

Um ponto importante é a distinção realizada entre as classes destinadas a representar os componentes das redes e as classes que representam os tokens da rede. Os tokens são entidades concretas da manufatura, sendo elas os agentes, peças e pallets. Todos os tokens herdam da classe Elemento. As demais classes são utilizadas para representar os componentes das redes ou até mesmo as próprias redes e sub-redes.

No banco de dados, entidades que herdam uma ou mais superclasses são representadas por múltiplas tabelas, cada uma contendo as propriedades de uma das classes implementadas – vide anexo B.

8.2.2 Classe InterfaceBD

Propriedades

conexão	*transação estabelecida com o BD apontado pelo objeto*
metaDados	*descrição do conteúdo dos resultados obtidos em uma busca*
resultado	*conteúdo do resultado de uma busca realizada no BD*
sentença	*query para a realização da busca, ou do último resultado*

Métodos

InterfaceBD	*inicializa o objeto, estabelecendo uma conexão com o BD*
finalize	*termina a conexão com o banco de dados, ao destruir o objeto*
iniciarConsulta	*inicia uma consulta e obtém o resultado para uma busca*
finalizarConsulta	*termina a consulta*
obter(...)	*utilizado dentro de uma consulta para obter um dado do resultado. O dado será buscado para a tabela especificada, no registro corrente do resultado.*
próximoRegistro	*dentro de uma consulta, muda o foco da busca para o próximo registro*

Esta classe, embora não representada na estrutura fundamental mostrada anteriormente, é uma classe acessória do sistema. Ela realiza a comunicação entre o programa e o repositório de dados. Cada classe do sistema, ao requisitar dados do banco de dados, realiza uma chamada a um objeto instanciado desta classe. Cada objeto da classe InterfaceBD representa uma transação com o banco de dados e precisa, portanto, ser eliminado após a realização da transação para que outros usuários possam utilizar o acesso ao banco.

Para realizar uma busca no banco de dados, a classe usuária de um objeto instanciado de InterfaceBD deve iniciar uma consulta com a query desejada. Estando a consulta iniciada, o resultado da busca pode ser utilizado para buscar, em cada registro, o conteúdo de um campo. Para isso, basta utilizar o método *obter* para o tipo desejado (*obterLong, obterInt, obterString, obterBoolean*) especificando-se o nome da coluna onde se encontra o campo. A busca no registro seguinte é realizada mudando-se o foco do método *obter* por meio do método *próximoRegistro*.

8.2.3 Classe EntidadeDoBD

Propriedades

ID	*Identificador único do objeto*

Métodos

EntidadeDoBD	*inicializa o objeto, atribuindo o identificador único à instância*
obterID	*retorna o número do ID*
obterReferências	*método abstrato - obtém em tempo de execução as referências internas por meio dos IDs de cada propriedade*
obterReferênciaDoID	*busca em uma lista de objetos a referência para um objeto que possua o ID especificado como argumento*
encontrarEntidade	*a partir de uma lista de tabelas, encontra a tabela contendo objetos da entidade desejada*
encontrarReferência	*busca em uma lista de tabelas, a referência para o objeto com ID especificado, dentro da tabela para a classe especificada*
obterEntidadeDoBD	*é uma função estática de classe, somente definida nas classes filhas. Obtém uma lista contendo todos os objetos da entidade correspondente a esta classe no banco de dados.*

Todos os objetos, cuja descrição se encontra no banco de dados, devem ser instanciados de classes que herdem a superclasse EntidadeDoBD. Essa classe define os métodos necessários para se estabelecer uma ligação entre o identificador do objeto no banco de dados, o ID, e as referências internas do programa. No banco de dados, o ID do objeto é a chave primária da tabela que comporta a sua entidade. No programa, as relações entre esse objeto e outros objetos é feita por meio de variáveis de referência. A referência é utilizada em tempo de execução para apontar um objeto na memória. O ID é persistente enquanto o objeto estiver presente no banco de dados.

Quando o programa realiza uma cópia do banco de dados para efeito de uma simulação, é obtida uma lista com as entidades presentes no banco (ver classe *Simulação*). Cada lista contém os objetos da entidade que esta especifica. Para obter a lista de objetos de uma entidade, o programa executa uma chamada ao método *obterEntidadeDoBD* para a classe correspondente à entidade. Esse método cria os objetos a partir do banco de dados e os insere na lista.

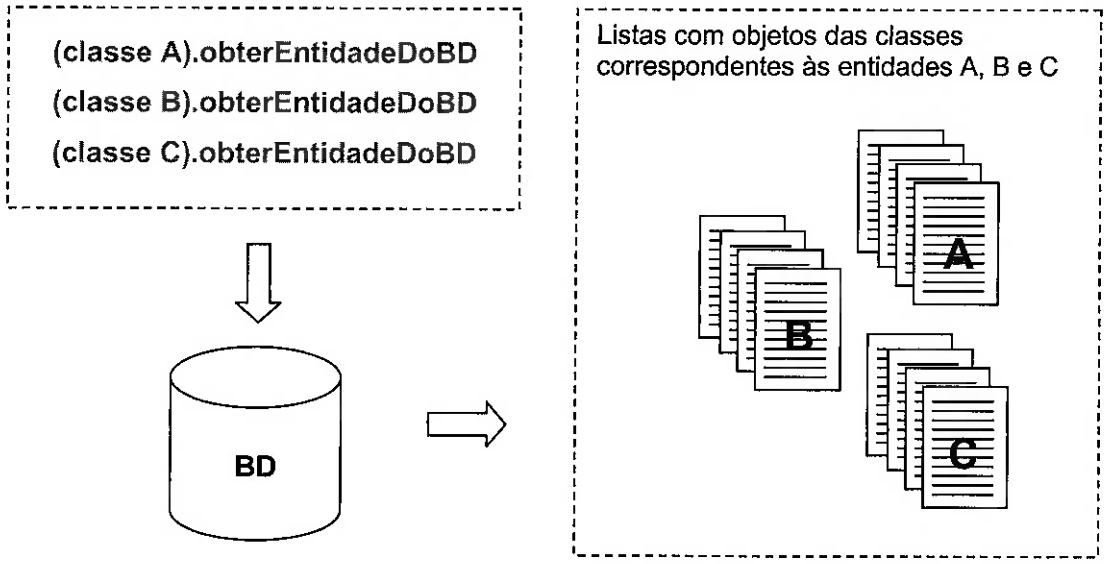


Figura 8.3 obtenção dos das listas de objetos do banco de dados

Os relacionamentos do banco são estabelecidos por meio de chaves e referências a estas chaves, os IDs. Estas referências com ID são números inteiros longos e precisam ser atualizadas no programa para as referências reais a objetos existentes em tempo de execução. Essa tarefa só é possível quando todas as listas de entidades estiverem prontas.

Para obter as referências reais em tempo de execução, a partir das referências por meio de IDs presentes em cada objeto, o programa deve chamar o método *obterReferências*. Uma lista de tabelas contendo todos os objetos obtidos do banco é passada como argumento ao método. Este busca entre as entidades presentes nesta lista todos os objetos os quais são referenciados pelo objeto solicitante. As referências reais são então procuradas nas listas de objetos correspondentes, utilizando-se para isso o ID de referência.

Para auxiliar a implementação do método *obterReferência*, esta classe disponibiliza métodos auxiliares *obterReferênciaDoID, encontrarReferência* e *encontrarEntidade* para realizar a busca de objetos nas listas. Os objetos são procurados segundo o ID de referência obtido do banco de dados e o nome da classe correspondente à entidade a qual o objeto pertence.

A Figura 8.4 mostra a obtenção das referências de um objeto da classe A. O objeto é instanciado a partir das informações do banco de dados, contendo apenas IDs das suas referências. Por meio dos métodos apropriados, o objeto encontra suas referências para objetos das classes B e C nas listas de objetos, por meio dos identificadores de B e C conhecidos.

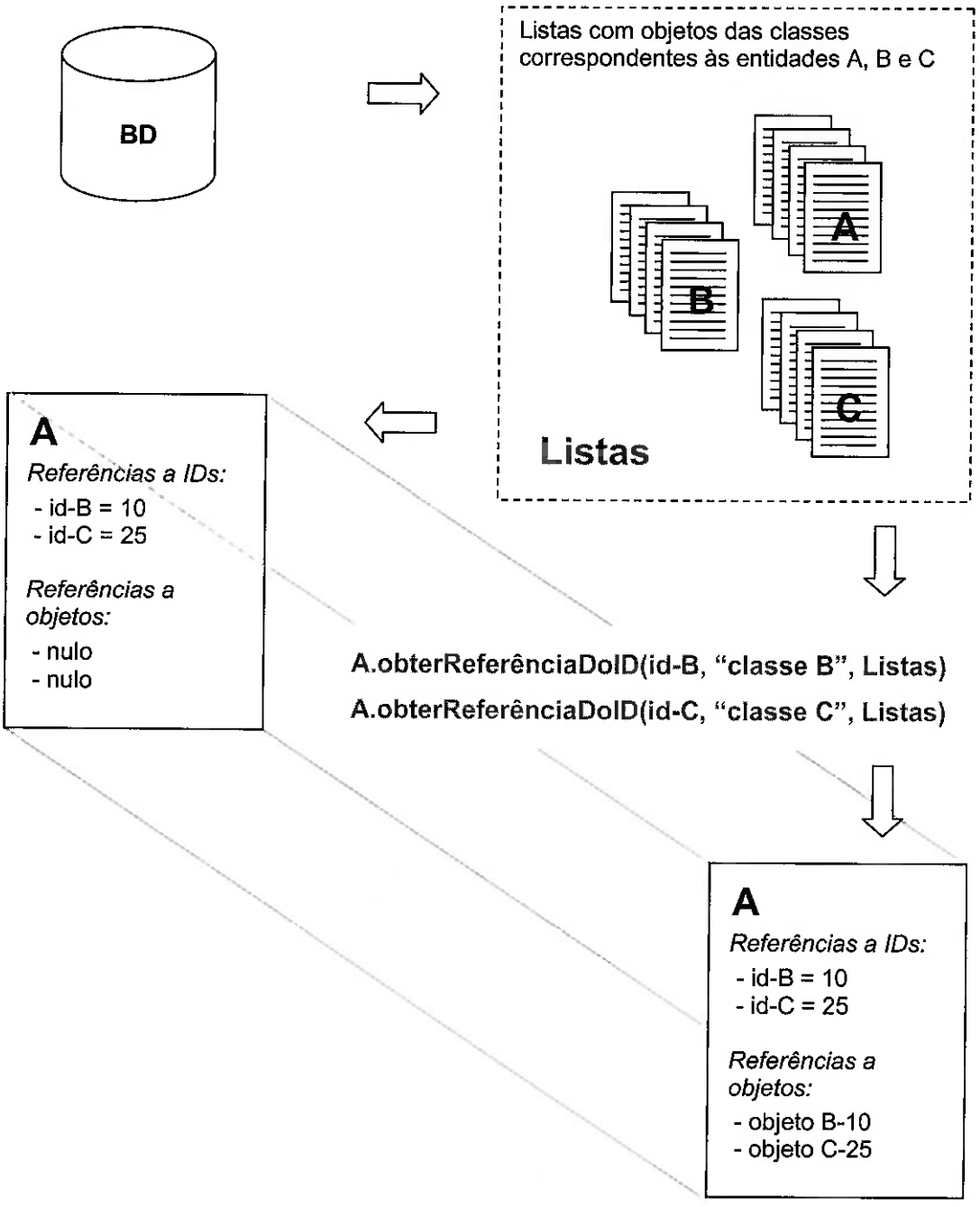


Figura 8.4 esquematização do método para obter referências

8.2.4 Classe Elemento

Superclasses:	*EntidadeDoBD*
Interfaces:	*Token*

Propriedades

estadoID	*identificador do Estado associado ao elemento*
superTokenID	*identificador do Token que contém o elemento, se diferente de nulo*
lugarID	*identificador do Lugar associado ao elemento*
tipoDeTokenID	*identificador do TipoDeToken associado ao elemento*
estado	*referência a um objeto que indica o estado do elemento*
lugar	*referência a um objeto Lugar, lugar ocupado pelo elemento na rede*
tipoDeToken	*referência a um objeto TipoDeToken, tipo de token que caracteriza o elemento*

Métodos

Elemento	*construtor da classe*
destruir	*anula o elemento para que este seja destruído e eliminado da lista de tokens que o comporta*
estadoAtual	*retorna o estado atual do elemento*
novoEstado	*atribui um novo estado ao elemento*
obterReferências	*obtém as referências em tempo de execução para os identificadores do estado, superToken, lugar e tipoDeToken*
obterSubTokens	*obtém todos os tokens pertencentes a este elemento, caso ele seja um token que comporta subtokens*
obterSubTokensDoTipo	*obtém apenas sub-tokens de um certo tipo*
obterSuperToken	*retorna o token que contém o elemento, caso exista*
tokenDoTipo	*verifica se o token é do tipo especificado no argumento*

Esta classe é a base para a herança de todas as classes referentes a objetos da manufatura: agentes, peças, pallets e conjuntos. A classe Elemento implementa a interface Token, ou seja, todos os objetos da manufatura são tokens presentes nas redes de dados e processos. A propriedade *tipoDeToken* informa à rede qual o tipo de token assumido pelo objeto. Outra forma de ligação entre o token e a rede são as propriedades *local* e *estado*. A primeira informa o local da rede no qual se localiza o

token no momento atual, enquanto que a propriedade *estado* informa o estado associado a esta posição. O conjunto de estados de todos os objetos da manufatura é o estado da planta.

Como um token pode conter outros tokens, por exemplo um *conjunto*, há métodos apropriados para se obter sub-tokens. Os tokens podem ainda ser filtrados por tipos. Assim, é possível identificar se o token é uma máquina ou uma peça, por exemplo.

A noção de token como conjunto foi introduzida para poder se agrupar, em apenas um token na rede, um conjunto contendo tokens de diferentes tipos. Por exemplo, uma máquina de usinagem – agente, com capacidade para 20 peças, pode ser agrupada com tais peças em um conjunto. Desta forma o token conjunto tem internamente 21 tokens: um agente e vinte peças. Essa simplificação evita inconsistências na atribuição de pesos aos arcos e lugares. O peso indica neste caso o número de conjuntos, e não o número de peças ou agentes – que pode, em casos especiais, sofrer variações.

Os elementos precisam, durante a evolução da rede, serem criados e destruídos. O método *destruir* marca o elemento com inválido. Assim, a rede que comporta o token pode eliminar este objeto de suas listas internas de tokens, liberando assim espaço na memória.

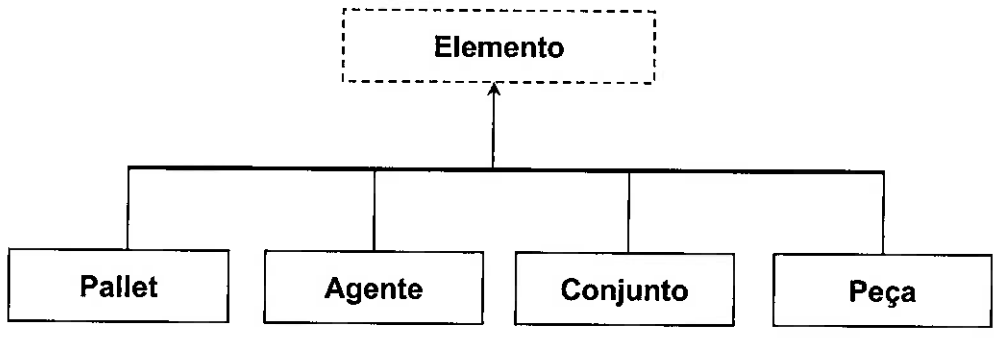


Figura 8.5 classes que herdam a classe Elemento

8.2.5 Interface Token

Métodos

destruir	*anula o token para que este seja destruído e eliminado da lista de tokens que o comporta*
estadoAtual	*retorna o estado atual do token*
novoEstado	*atribui um novo estado ao token*
obterSubTokens	*obtém todos os tokens pertencentes a este token, caso este seja um token que comporta subtokens*
obterSubTokensDoTipo	*obtém apenas sub-tokens de um certo tipo*
obterSuperToken	*retorna o token que contém este token, caso exista*
tokenDoTipo	*verifica se o token é do tipo especificado no argumento*

A interface Token define todos os métodos que uma classe do tipo Token deve implementar.

O método *destruir* deve assegurar que o token se tornará inválido na rede que o comporta, sendo em breve excluído no momento apropriado. O estado do token pode ser alterado ou avaliado, respectivamente através dos métodos *novoEstado* e *estadoAtual*. O tipo representado pelo token (peça, agente, pallet, conjunto, etc.) pode ser avaliado por meio do método *tokenDoTipo*.

Como um token pode ser um conjunto de tokens, os métodos *obterSubTokens* e *obterSubTokensDoTipo* prestam-se à obtenção dos tokens contidos internamente. O segundo filtra sua lista interna de tokens retornando apenas os tokens de um certo tipo.

Se o token pertence a um token superior, ou seja, está contido em um conjunto de tokens, o método *obterSuperToken* deve retornar uma referência para o token superior.

Como Token é uma interface, as classes que o implementam devem sobrescrever os métodos acima definidos e assegurar que estas funcionalidades sejam corretamente empenhadas.

8.2.6 Classe Componente

Superclasses:	*EntidadeDoBD*
Interfaces:	*SubRede*

Propriedades

redeID	*identificador da rede a qual este componente pertence*
posiçãoNaRede	*posição ocupada pelo componente na rede*

Métodos

Componente	*construtor da classe*
obterIDRedeSuperior	*retorna o identificador da rede que comporta este componente*
obterPosiçãoNaRedeSuperior	*retorna a posição ocupada pelo componente na rede*

A classe Componente é a superclasse para herança de todos os componentes elementares das redes de dados e processos: os lugares, arcos e transições.

Cada componente de uma rede é identificado por sua posição na rede. Assim, a rede pode obter o componente do banco de dados atribuí-lo ao lugar correto na rede. A rede que possui o componente é identificada pela propriedade *redeID*.

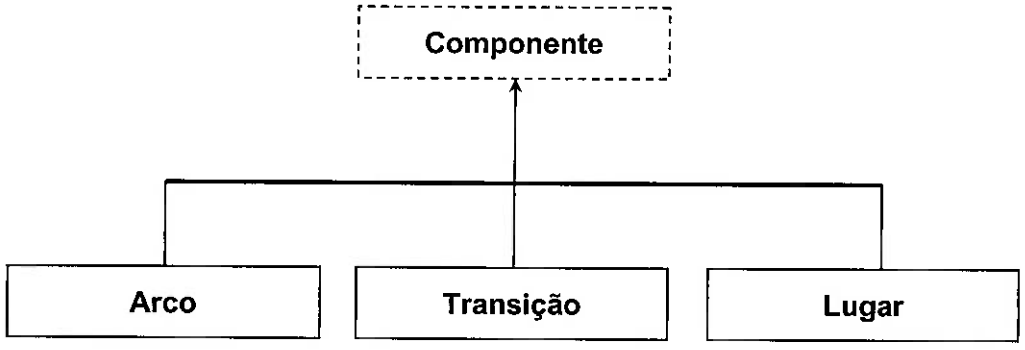


Figura 8.6 classes que herdam a classe Componente

8.2.7 Classe Rede

Superclasses:	*EntidadeDoBD*

Propriedades

listaDeComponentes	*lista com todos os componentes da rede, podendo estes serem arcos, transições, lugares, ou outros tipos que implementem a interface SubRede*

Métodos

Rede	*construtor da classe*
obterPróximaTransição	*retorna o tempo no qual ocorrerá a próxima transição contida nesta rede, caso o estado da rede permaneça inalterado*
processarRede	*processa a rede até o tempo final indicado no argumento*

A classe rede é a base para a herança de todas as classes as quais constituem redes de uma p-net. Tanto um processo inteiro, como uma operação ou um conjunto de operações, podem ser considerados como objetos do tipo Rede. As redes são formadas por uma lista de componentes. Estes podem ser tanto objetos do tipo Componente, como outras sub-redes. A classe Rede não define, porém, a posição dos objetos da lista *listaDeComponentes* internamente. Essa informação deve ser tratada nas classes filhas.

Duas funcionalidades próprias de um objeto do tipo rede são: a capacidade de verificar a próxima transição interna possível; e processar a rede até um tempo determinado como argumento. Os métodos *obterPróximaTransição* e *processarRede* desempenham estas funções. Estes métodos podem ser sobrescritos nas classes filhas, de forma a incluir as particularidades de cada classe na análise.

Assim, é possível definir, em um conjunto de redes, em qual delas deverá ocorrer a próxima transição. Basta comparar os resultados do método *obterPróximaTransição* para cada uma. Caso o resultado de uma ou mais redes seja o mesmo, a rede superior deve incluir um critério para decidir qual sub-rede ocorrerá primeiro. Escolhida a próxima rede, sua transição é realizada chamando-se o método *processarRede* para um tempo final igual ao calculado anteriormente pelo outro método.

8.2.8 Interface SubRede

Métodos

obterDescrição	*retorna um texto descrevendo a sub-rede*
obterPosiçãoNaRedeSuperior	*retorna a posição desta rede na rede superior*
obterIDRedeSuperior	*retorna o identificador da rede superior*

Esta interface define os métodos necessários para que uma classe que a implemente apresente as características desejadas de uma sub-rede. Uma sub-rede é um componente interno de outra rede (operação ou processo).

A sub-rede possui obrigatoriamente uma posição na rede superior que a comporta. Esta posição é obtida pelo método *obterPosiçãoNaRedeSuperior*. Uma referência à rede superior também deve estar presente. Esta pode ser obtida pelo método *obterIDRedeSuperior*.

8.2.9 Classe Processo

Superclasses:	*EntidadeDoBD ← Rede*

Propriedades

descrição	*descrição textual do processo instanciado*
loteID	*identificador do lote associado a este processo*
próximaOperação	*armazena uma referência a uma possível próxima operação, calculada por meio do método obterPróximaOperação, caso o estado das redes internas não se alterem*

Métodos

Processo	*construtor da classe*
obterEntidadeDoDB	*cria um uma lista com todos os objetos desta classe a partir dos registros contidos nas tabelas da entidade correspondente a esta classe*
obterPróximaTransição	*retorna o tempo no qual ocorrerá a próxima transição contida no processo, caso o estado da rede representada permaneça inalterado*
obterReferências	*obtém a referência em tempo de execução para o identificador do lote*
processarRede	*processa a rede até o tempo final indicado no argumento*

A classe processo é a superclasse herdada por todas as classes que representam um processo completo de manufatura. Tal processo pode ser definido como uma p-net composta por objetos do tipo Componente (arcos, transições, lugares) e operações. Assim, um objeto do tipo processo é uma rede p-net que comporta várias sub-redes.

Como qualquer outra rede, a classe Processo também implementa os métodos elementares das redes *obterPróximaTransição* e *processarRede*. Ao processar uma solicitação para se obter o tempo no qual ocorrerá a próxima transição, o objeto Processo armazena na propriedade *próximaOperação* uma referencia para a próxima sub-rede na qual ocorrerá a transição, caso o estado da rede não mude anteriormente.

A função *obterEntidadeDoBD* é um método estático da classe Processo, ou seja, não precisa ser chamada a partir de um objeto deste tipo. Esse método é responsável pela obtenção de todos os objetos do tipo Processo presentes no banco de dados.

8.2.10 Classe Operação

Superclasses:	*EntidadeDoBD ← Rede*
Interfaces:	*SubRede*

Propriedades

descrição	*descrição textual da operação instanciada*
redeID	*identificador da rede superior associada a esta operação (um processo)*
posiçãoNaRede	*posição ocupada na rede que comporta esta operação*
tempoOperação	*tempo necessário para a realização da operação*
tempoInício	*tempo de início da operação, caso ela tenha iniciado*

Métodos

Operação	*construtor da classe*
obterDescrição	*retorna a descrição textual da operação*
obterEntidadeDoBD	*cria um uma lista com todos os objetos desta classe a partir dos registros contidos nas tabelas da entidade correspondente a esta classe*
obterIDRedeSuperior	*retorna o identificador da rede superior*
obterPosiçãoNaRedeSuperior	*retorna a posição desta rede na rede superior*
obterPróximaTransição	*retorna o tempo no qual ocorrerá a próxima transição contida nesta operação, caso o estado da rede representada permaneça inalterado*
obterReferências	*obtém a referência em tempo de execução para o identificador da rede superior*
processarRede	*processa a rede até o tempo final indicado no argumento*
finalizarOperação	*método abstrato, define a execução da transição de estado relacionada ao término da operação*
iniciarOperação	*método abstrato, define a execução da transição de estado relacionada ao início da operação*
podeFinalizar	*método abstrato, avalia se a transição relacionada ao término da operação está habilitada*
podeIniciar	*método abstrato, avalia se a transição relacionada ao início da operação está habilitada*

Uma Operação é, por definição do modelo escolhido para a implementação, uma sub-rede de um objeto do tipo Processo. A classe Operação serve de base para a herança de todas as classes que desempenham o papel de uma sub-rede dentro de um

processo. A Figura 8.7 apresenta todas as classes herdadas de Operação para uma simulação completa do caso em estudo. Para os testes realizados, foram implementadas apenas as classes Montagem, Desmontagem, Processamento e Metrologia. As demais classes são variantes das primeiras.

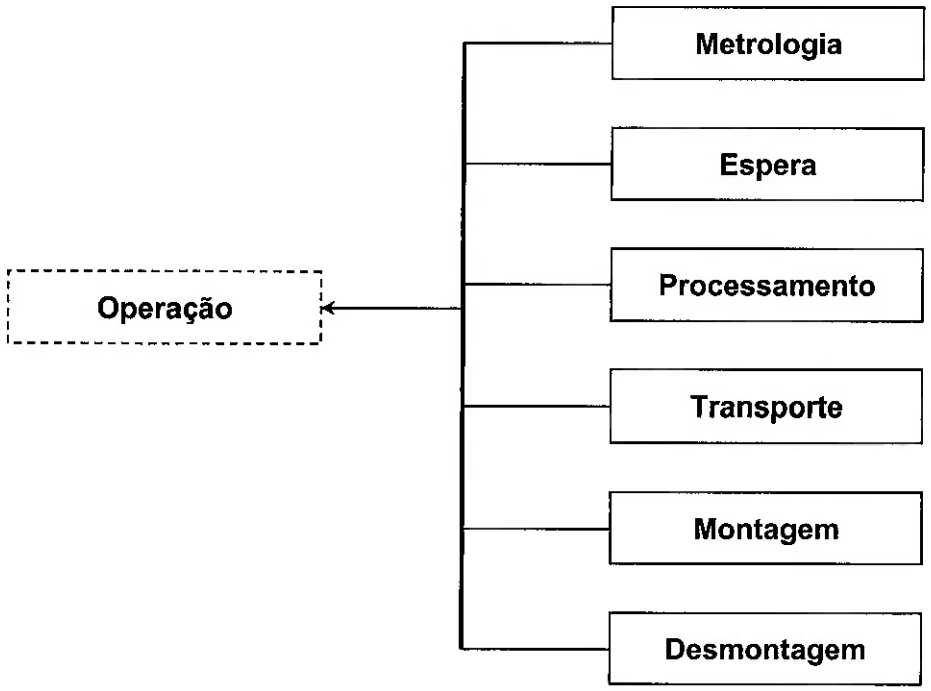


Figura 8.7 classes que herdam a classe Operação

Outros tipos de processos de manufatura podem ser representados pelo modelo desenvolvido no projeto. Para tanto, basta desenvolver novas classes do tipo Operação que encapsulem as particularidades envolvidas em cada etapa do processo simulado.

A classe Operação implementa a interface SubRede através dos métodos necessários. Os métodos elementares da classe Rede são sobrescritos, adicionando as características peculiares da operação ao algoritmo inicial.

Para facilitar a análise de cada etapa de um processo completo de manufatura, a classe Operação representa uma etapa unitária de um processo completo. Ou seja, a rede representada pela operação pode ser enxergada como uma sub-rede de processamento que apresenta início e fim. Para isso, a classe define um conjunto de

quatro métodos abstratos que devem obrigatoriamente ser sobrescritos nas classes filhas.

O método *podeIniciar* avalia se o estado atual da sub-rede apresenta as condições suficientes para que a operação que esta rede representa possa ser iniciada. Geralmente, a classe que herda a classe Operação vai possuir uma ou mais transições que estejam relacionadas ao início da operação. Caso alguma destas transições possa ocorrer, a condição retornada pelo método *podeIniciar* é verdadeira. Da mesma forma, o método *podeFinalizar*, também abstrato, verifica se a operação pode terminar.

As transições relacionadas ao início e ao fim de cada operação podem ser solicitadas através dos métodos *iniciarOperação* e *finalizarOperação*.

O tempo necessário para realizar a operação é o tempo decorrido entre o início da operação e o seu fim. O tempo do sistema é global, ou seja, ao ser iniciada a Operação o tempo atual do sistema é armazenado na propriedade *tempoInício*. Assim, o objeto do tipo Operação é capaz de identificar se a operação chegou ao final ou não, quando solicitado o método *finalizarOperação*. Essa mesma informação também é utilizada para o cálculo do método *obtérPróximaTransição*.

8.3 Classes concretas

8.3.1 Classes Peça, Pallet e Agente

Classe Peça

Superclasses:	*EntidadeDoBD ← Elemento*
Propriedades	
processoID	*identificador do processo associado a esta classe*
Métodos	
Peça	*construtor da classe*
obterEntidadeDoDB	*cria um uma lista com todos os objetos desta classe a partir dos registros contidos nas tabelas da entidade correspondente a esta classe*

Classe Pallet

Superclasses:	*EntidadeDoBD ← Elemento*
Propriedades	
capacidade	*capacidade do pallet, em número de peças*
Métodos	
Pallet	*construtor da classe*
obterEntidadeDoDB	*idem a peça*

Classe Agente

Superclasses:	*EntidadeDoBD ← Elemento*
Propriedades	
descrição	*descrição textual das características do agente*
Métodos	
Agente	*construtor da classe*
obterEntidadeDoDB	*idem a peça*

As classes Pallet, Peça e Agente herdam diretamente a classe Elemento. Estes objetos são os elementos reais da manufatura, representam entidades concretas.

A propriedade *processoID* da classe peça especifica que a peça precisa estar necessariamente relacionada a um processo de produção, ou seja, a uma p-net que descreva todas as etapas de fabricação desta peça.

Estas classes implementam também, indiretamente por meio da classe Elemento, a interface Token. Desta forma, todos os objetos envolvidos na manufatura são tokens. O estado de todos os tokens é o próprio estado da planta.

8.3.2 Classe Conjunto

Superclasses:	*EntidadeDoBD ← Elemento*

Propriedades

listaDeTokens	*lista com todos os tokens pertencentes ao conjunto*
ativo	*indica se o token está ativo ou não*

Métodos

Conjunto	*construtor da classe*
destruir	*anula o token para que este seja destruído e eliminado da lista de tokens que o comporta*
novoEstado	*atribui um novo estado ao token*
obterEntidadeDoBD	*cria um uma lista com todos os objetos desta classe a partir dos registros contidos nas tabelas da entidade correspondente a esta classe*
obterReferências	*obtém as referências em tempo de execução para os identificadores de todos os tokens contidos no conjunto*
obterSubTokens	*obtém todos os tokens pertencentes ao conjunto*
obterSubTokensDoTipo	*obtém apenas sub-tokens de um tipo específico*

A classe conjunto herda a classe Elemento e também implementa, por meio desta, a interface Token. Ao contrário das classes Peça, Pallet e Agente, a classe conjunto não representa um objeto real da manufatura, e sim um agrupamento de tokens. Os conjuntos são utilizados para representar tokens formados basicamente pela união entre agentes e peças ou pallets e peças.

Os métodos *obteSubToken* e *obterSubTokensDoTipo* já implementados na classe Elemento são sobrescritos nesta classe, de forma a suportar as funcionalidades relacionadas à existência de tokens internos ao objeto Conjunto.

8.3.3 Classe Arco

Superclasses:	*EntidadeDoBD ← Componente*

Propriedades

lugarID	*identificador do lugar associado ao arco*
tipoDeTokenID	*identificador do tipo de token associado ao arco*
transiçãoID	*identificador da transição associada ao arco*
deLugarParaTransição	*especifica a direção do arco: do lugar para uma transição ou vice-versa*
lugar	*referência para o lugar associado ao arco*
tipoDeToken	*referência para o tipo de token associado ao arco*
peso	*peso do arco, número de token transportados pelo arco – também é uma condição para a transição associada*
transição	*referência para a transição associada ao arco*

Métodos

Arco	*construtor da classe*
aceitaTransmitirTokens	*verifica se o lugar associado a este arco possui um número suficiente de tokens do tipo do arco para serem transmitidos, caso o lugar seja a orígem do arco. Caso contrário, verifica se o lugar possui espaço suficiente para receber o número de tokens igual ao peso do arco*
enviarTokens	*envia os tokens do argumento para o lugar de destino, se o lugar for destino*
obterDescrição	*obtém uma descrição textual dos objetos do arco*
obterEntidadeDoBD	*cria um uma lista com todos os objetos desta classe a partir dos registros contidos nas tabelas da entidade correspondente a esta classe*
obterReferências	*obtém as referências em tempo de execução para os identificadores do lugar, tipo de token e transição associados ao arco*
obterTipoDeToken	*retorna o tipo de token transmitido pelo arco*
obterTokens	*retorna uma lista com um número de tokens igual ao peso do arco, provenientes do lugar associado ao arco – caso o lugar seja a orígem do arco*

A classe Arco é um componente de rede. Um arco estabelece um relacionamento entre um lugar e uma transição na rede. A direção do arco indica se os tokens por ele

transmitidos são movimentados do lugar para a transição ou da transição para um lugar.

Um arco, que liga um lugar até uma transição, é capaz de transmitir tokens se o lugar possuir um número de tokens igual ao peso do arco. Ou seja, deve haver um número suficiente de tokens no lugar para serem transmitidos pelo arco. E, ainda, estes tokens devem necessariamente ser do mesmo tipo de token transmitido pelo arco. Se esta condição for satisfeita, o método *obterTokens* retira os tokens do lugar. A lista de tokens obtida é então transmitida para o destino correspondente.

Da mesma forma, se o arco liga a transição até o lugar, o lugar precisa ter espaço disponível suficiente para receber os tokens transmitidos pelo arco. Neste caso, o método *enviarTokens* é utilizado para enviar uma lista de tokens para o lugar especificado. Para os dois casos de orientação do arco, o método *aceita-TransmitirTokens* realiza esta verificação.

8.3.4 Classe Transição

Superclasses:	*EntidadeDoBD ← Componente*
Propriedades	
listaDeArcos	*lista com todos os arcos associados a esta transição*
Métodos	
Transição	*construtor da classe*
obterDescrição	*obtém descrição textual dos objetos desta classe*
obterEntidadeDoBD	*cria um uma lista com todos os objetos desta classe a partir dos registros contidos nas tabelas da entidade correspondente a esta classe*
obterReferências	*obtém as referências em tempo de execução para os identificadores de todos os arcos relacionados a esta transição*
verificarCondiçãoParaTransição	*verifica se as condições para a ocorrência desta transição estão satisfeitas, segundo o estado atual da rede*

O componente transição é objeto da rede que estabelece as características da mudança de estado relacionada à ocorrência de um evento. O objeto possui uma lista de arcos que define os lugares anteriores e posteriores à transição.

O método *verificarCondiçãoParaTransição* executa, para todos os arcos da transição, o método *aceitaTransmitirTokens*. Se todos os arcos aceitam a transferência de tokens, isso significa que a transição está habilitada.

A transição, contudo, só ocorre se estiver habilitada e ainda houver ocorrido um evento que a dispare. A ocorrência de tais eventos depende de características inerentes a cada processo da manufatura. Portanto, o disparo das transições é controlado pelas operações que as comportam. Assim, a lógica por trás do disparo de uma transição fica encapsulada na operação.

8.3.5 Classe Lugar

Superclasses:	*EntidadeDoBD ← Componente*

Propriedades

estadoID	*identificador do estado associado ao lugar*
tipoDeTokenID	*identificador do tipo de token associado ao lugar*
capacidade	*capacidade do lugar, em número de tokens*
listaDeTokens	*lista com todos os tokens presentes neste lugar*
estadoAssociado	*referência ao estado associado a os tokens presentes neste lugar*
tipoDeToken	*tipo de token que pode ocupar este lugar*

Métodos

Lugar	*construtor da classe*
adicionarTokens	*adiciona os tokens passados como argumento para a lista de tokens do lugar*
obterDescrição	*obtém uma descrição textual dos objetos relacionados ao lugar*
obterEntidadeDoBD	*cria um uma lista com todos os objetos desta classe a partir dos registros contidos nas tabelas da entidade correspondente a esta classe*
obterEstadoAssociado	*retorna uma referência ao estado associado aos tokens presentes neste lugar*
obterReferências	*obtém as referências em tempo de execução para os identificadores do estado e tipo de token associados a este lugar*
qualCapacidade	*retorna a capacidade total em número de tokens do lugar*
qualDisponibilidade	*retorna a disponibilidade restante da capacidade, tendo em vista o número de tokens já presentes*
removerTokens	*retorna uma lista com um número de tokens especificado como argumento*
atualizarEstadoTokens	*atualiza o estado dos tokes recém adicionados à lista de tokens para o estado associado ao lugar*

Um objeto lugar representa uma posição possível de ser ocupada pelos tokens na rede. A cada lugar está associado um estado diferente, que é atribuído ao token que ocupa aquele lugar. A disponibilidade do lugar informa o número de tokens que o lugar ainda pode abrigar na rede.

8.3.6 Classes herdadas de Operação

Superclasses:	*EntidadeDoBD ← Rede ← Operação*
Propriedades	
(arcos)	*referências aos arcos pertencentes à rede desta operação*
(transições)	*referências às transições pertencentes à rede desta operação*
(outras propriedades)	*propriedades específicas de cada tipo de operação, de acordo com as funcionalidades desempenhadas*
Métodos	
Construtor da classe	*classes desenvolvidadas: Processamento, Metrologia, Montagem e Desmontagem*
obterEntidadeDoBD	*cria um uma lista com todos os objetos desta classe a partir dos registros contidos nas tabelas da entidade correspondente a esta classe*
obterReferências	*obtém as referências em tempo de execução para os identificadores de todos os arcos, transições e outras propriedades relacionadas a esta operação*
finalizarOperação	*executa a transição de estado relacionada ao término da operação*
iniciarOperação	*executa a transição de estado relacionada ao início da operação*
podeFinalizar	*avalia se a transição relacionada ao término da operação está habilitada*
podeIniciar	*avalia se a transição relacionada ao início da operação está habilitada*

A estrutura de classe mostrada na tabela anterior é utilizada por todas as classes que herdam a superclasse Operação. As operações implementadas para testar o funcionamento do algoritmo foram: Desmontagem, Montagem, Processamento e Metrologia. As classes Transporte e Descanso não foram implementadas, mas são facilmente identificadas como variantes da operação processamento.

As classes que herdam Operação devem necessariamente implementar os métodos *finalizarOperação, iniciarOperação, podeFinalizar* e *podeIniciar*. Estes métodos são implementados tendo-se em vista as características próprias de cada operação.

Na operação Montagem, o início da operação resulta no agrupamento de um pallet e um número específico de peças em um token do tipo conjunto. Depois do tempo necessário para a realização da montagem, se a transição estiver habilitada, o fim da operação transmite o token conjunto contendo as peças montadas no pallet para a próxima operação. Na desmontagem, o processo é inverso: um token conjunto contendo peças e um pallet é desagrupado. O objeto conjunto restante é então marcado para eliminação.

A operação de processamento inicia-se com o agrupamento do token agente com as peças processadas. Ao final da operação, o conjunto é desagregado. O agente torna-se novamente disponível e as peças são passadas para a próxima operação.

A operação metrologia ocorre da mesma forma que o processamento. Mas, a transição final é escolhida entre três possibilidades: se a peça foi aprovada, se virou refugo, ou se precisa ser novamente trabalhada. As probabilidades ligadas a cada um destes eventos são características do objeto metrologia, cuja lógica decide qual dos três eventos disparar. As informações relativas as probabilidades também são armazenadas no repositório de dados, assim como as outras propriedades das operações.

Utilizando-se a estrutura fornecida pela superclasse Operação, é possível implementar uma grande variedade de tipos diferentes de operações da manufatura, incluindo em sua lógica interna o grau de complexidade desejado. Além disso, cada objeto Operação é totalmente flexível, pois os atributos da operação são também armazenados no banco de dados.

8.3.7 Classes herdadas de Processo

Superclasses:	*EntidadeDoBD ← Rede ← Processo*
Propriedades	
(propriedades)	*podem ser declaradas propriedades específicas para caracterizar particularidades inerentes a cada tipo de processo*
Métodos	
Construtor da classe	*Foi desnvolvida a classe Processo_A como um exemplo de classe Processo*
obterEntidadeDoBD	*cria um uma lista com todos os objetos desta classe a partir dos registros contidos nas tabelas da entidade correspondente a esta classe*
obterReferências	*obtém as referências em tempo de execução para os identificadores de todas as propriedades referidas por IDs*

A classe Processo_A herda diretamente a classe Processo para representar a p-net do processo de fabricação de uma engrenagem de um tipo específico, "A". Assim como a operação, as classes que herdem Processo têm grande flexibilidade para criar a lógica interna, de forma independente das demais redes existentes no programa. Novos atributos para a classe podem ser armazenados livremente no banco de dados.

No caso, a classe Processo_A, criada como exemplo, apenas estende a classe Processo sem realizar nenhum aperfeiçoamento à superclasse. É necessário herdar a classe, pois a classe Processo é abstrata por definição, devendo ser apenas utilizada como base para as classes de processo concretas.

A classe Processo contém sub-redes, geralmente operações – mas que podem perfeitamente ser outros processos. A posição de cada sub-rede na p-net do processo é determinada pelos próprios lugares das redes, que são únicos e servem como elo de ligação entre as diferentes operações ou processos internos.

8.3.8 Classe TipoDeToken

Superclasses:	*EntidadeDoBD*

Propriedades

descrição	*descrição textual do tipo de token representado por este objeto*

Métodos

TipoDeToken	*construtor da classe*
obterEntidadeDoBD	*cria um uma lista com todos os objetos desta classe a partir dos registros contidos nas tabelas da entidade correspondente a esta classe*
obterReferências	*sem ação nesta classe, não existem propriedades referenciadas por identificadores*
equals	*verifica se um objeto tipoDeToken expressa um tipo de token do mesmo tipo do definido por este objeto*

Essa classe tem a função essencial de caracterizar cada tipo possível de token presente nas redes existentes na simulação. Na simulação do exemplo, os tipos de token definidos são AGENTE, PEÇA, CONJUNTO e PALLET.

8.3.9 Classe Simulação

Propriedades

início	*tempo inicial da realização da simulação*
interfaceBD	*referência para um objeto do tipo InterfaceBD que realiza a interface entre a simulação e o bando de dados fonte*
listas	*lista ligada contendo todas as listas de componentes da simulação*
próximoProcesso	*processo onde ocorrerá a próxima transição, segundo o estado atual de toda a rede da planta simulada*
tempoAtual	*tempo atual estabelecido durante o decorrer da simulação*
listaDe (...): Agentes, Arcos, Conjuntos, Estados, Lugares, Operações, Pallets, Peças, Processos, TiposDeToken, Transições.	*cada lista comporta todos os objetos de uma entidade presente no banco de dados.*

Métodos

Simulação	*construtor da classe, cria uma cópia na memória da estrutura da rede e do estado atual da planta, ambos buscados no banco de dados*
imprimirListas	*imprime a descrição textual de todos os objetos obtidos do banco de dados e que estão armazenados nas listas internas da simulação*
simular ( )	*simula a rede até o um ponto no qual nenhuma transição mais seja possível*
simular (tempo)	*simula a rede até o momento especificado no argumento*
calcularPróximaTransição	*retorna o momento no qual ocorrerá a próxima transição no conjunto de redes da planta*
obterComponentesDasRedes	*obtém do banco de dados todos os componentes que definem as redes da planta*
obterEstadoDaPlanta	*obtém do banco de dados todos os objetos que definem o estado atual da planta, ou seja, os tokens e suas localizações*
obterReferências	*realiza a operação de busca de referência para identificadores de todos os objetos presentes nas listas da simulação*
prepararDados	*obtém os componentes das redes, o estado da planta e realiza então a busca de referências*

A classe Simulação é responsável pela execução da simulação da dinâmica dos processos da planta. Ao se criar um objeto Simulação, o construtor da classe já obtém do banco de dados, por meio do método *prepararDados,* todos os objetos da planta e das redes utilizadas para se modelar a planta.

O método *prepararDados* chama os métodos *obterComponentesDasRedes e obterEstadoDaPlanta.* Estes métodos executam, para cada tipo diferente de objeto, o método estático de classe *obterEntidadeDoBD.* Este método retorna uma lista contendo todos os objetos da classe utilizada que estão armazenados no repositório de dados. Estando todas as listas de objetos prontas, o próximo passo é realizar a busca de referências para todos os objetos. As referências do banco de dados são utilizadas pelos objetos para procurar as referências reais em tempo de execução.

O método interno *calcularPróximaTransição* busca, entre todas as redes internas da planta, em qual ocorrerá a próxima transição. Este método é utilizado pelo método principal da classe para realizar a simulação da planta: *simular.*

O método *simular* executa todas as transições da planta até o tempo especificado, partindo-se do estado inicial da planta obtido no banco de dados. Caso não seja especificado o tempo final da simulação, as transições ocorrem até não haver mais nenhuma transição habilitada.

Este método já imprime automaticamente para o usuário as mudanças de estado ocorridas após cada transição. Essa funcionalidade é opcional, tendo-se em vista que a principal utilidade da classe é obter estados futuros ou tempos de processamento. Esses resultados são utilizados nas análises de tendência de evolução da rede, para prever o andamento futuro dos processos da manufatura.

8.3.10 Correspondência entre a estrutura de classes e o banco de dados

A tabela a seguir mostra a correspondência entre a estrutura de classes do programa e as entidades do repositório de dados. Essa correspondência é essencial para o modelo escolhido para a análise, que explora tanto as funcionalidades da estrutura orientada a objeto, quanto do modelo entidade-relacionamento.

Classe	Entidades correspondentes
EntidadeDoBD	todas
Elemento	TBL_ELEMENTO
Agente	TBL_ELEMENTO, TBL_AGENTE
Conjunto	TBL_ELEMENTO, TBL_CONJUNTO
Pallet	TBL_ELEMENTO, TBL_PALLET
Peça	TBL_ELEMENTO, TBL_PEÇA
Componente	TBL_COMPONENTE
Arco	TBL_COMPONENTE, TBL_ARCO
Transição	TBL_COMPONENTE, TBL_TRANSIÇÃO
Lugar	TBL_COMPONENTE, TBL_LUGAR
Operação	TBL_OPERAÇÃO
Processamento	TBL_OPERAÇÃO, TBL_PROCESSAMENTO
Montagem	TBL_OPERAÇÃO, TBL_MONTAGEM _
Desmontagem	TBL_OPERAÇÃO, TBL_DESMONTAGEM
Metrologia	TBL_OPERAÇÃO, TBL_METROLOGIA
Processo	TBL_PROCESSO
Processo_A	TBL_PROCESSO, TBL_PROCESSO_A
Estado	TBL_ESTADO
TipoDeToken	TBL_TIPO_DE_TOKEN

Além das classes anteriormente mostradas, existe ainda a classe Estado e a classe Tendência. A primeira é utilizada apenas para descrever o estado associado a cada token, em um momento específico do processo. A classe tendência utiliza a classe Simulação para realizar um teste de tendência para a rede existente no banco de dados.

9. Resumo dos resultados

O caso escolhido para estudo, uma planta genérica para a fabricação de engrenagens, apresenta uma complexidade moderada, contudo suficiente para a aplicação da metodologia proposta, permitindo ainda a elaboração de um software simples, capaz de representar a dinâmica dos processos no sistema proposto.

O sistema desenvolvido é composto por: dados das redes de processos e dos objetos para o caso estudado; uma estrutura fundamental de classes que suporta o sistema desenvolvido; um repositório de dados que corresponde a estrutura de classes; e um módulo de software capaz de simular a dinâmica dos processos da manufatura, segundo o modelo proposto.

Os testes realizados (vide capítulo 8.1) mostraram o correto funcionamento do sistema. O modelo consegue representar de forma adequada a dinâmica do sistema tomado como exemplo. Há contudo uma dificuldade própria da complexidade do sistema ao se alimentar o banco de dados com os dados das redes e dos objetos presentes na planta. Para solucionar este problema, é necessário criar mais um módulo de software independente, capaz de criar as redes de processos segundo as especificações do usuário e então alimentá-las no repositório de dados.

O sistema desenvolvido mostrou-se capaz de fornecer a atual utilização da capacidade do chão de fábrica a cada instante, o estágio de fabricação de todas as peças com pedidos e a previsão dos estados futuros de fabricação, tempos e recursos utilizados.

Os resultados obtidos durante o desenvolvimento deste projeto mostraram que a coleta de dados do chão de fábrica, aliada ao modelo que expressa a dinâmica dos processos, é capaz de expressar não só o estado atual da produção, como também a tendência de mudança deste estado. A compreensão imediata do estado da produção e suas tendências são importantes subsídios para a tomada de decisões pela supervisão da planta e pela gerência da fábrica.

O presente trabalho mostra de forma realística, porém reduzida a um setor da planta, como podem ser utilizadas, na prática, técnicas modernas de modelagem e aplicação de sistemas de informação ao chão-de-fábrica. Esta tendência é nomeada Integração Vertical da Empresa. Na base desta integração, estão os sistemas de controle, fluxo de informações e materiais do chão-de-fábrica. Acima, estão distribuídos os diversos níveis de gestão.

Anexo A

Este anexo inclui as listagens do software desenvolvido, segundo a estrutura especificada na fase de implementação do sistema de informação proposto. O código foi escrito em linguagem Java. Todas as classes fazem parte de um único pacote, o *dinproc* ("Dinâmica de Processos"). Os acessos à base de dados são realizados via JDBC. Comentários sobre as propriedades e métodos de cada classe estão disponíveis no capítulo 8, sendo que as listagens aqui apresentadas apresentam apenas os comentários essenciais. As classes e interfaces componentes do pacote são:

1. Agente
2. Arco
3. Componente
4. Conjunto
5. Elemento
6. EntidadeDoBD
7. Estado
8. InterfaceBD
9. Lugar
10. SubRede
11. OP_Desmontagem
12. OP_Metrologia
13. OP_Montagem
14. OP_Processamento
15. Operação
16. Pallet
17. Peça
18. Processo
19. Processo_A
20. Rede
21. Simulação
22. Tendência
23. TipoDeToken
24. Token
25. Transição

A – 1 Classe Agente

```
package dinproc;

import java.util.*;

/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */

public class Agente extends Elemento {

   protected String descrição;

   public Agente() {
   }

   public Agente(long id, String descrição_, long tipo, long estado_,
                 long lugar_, long superToken_) {
      super(id, tipo, estado_, lugar_, superToken_);
      descrição = descrição_;
   }

   public String toString(){
      return "\n{[Agente] descrição: " + descrição + super.toString() + "}";
   }


   //|||||||||||||||||||||||||||| HERDADO DE ENTIDADE DO BD ||||||||||||||||||

   public static LinkedList obterEntidadeDoBD(InterfaceBD bd){
      LinkedList lista = new LinkedList();
      Agente agente;
      bd.iniciarConsulta("SELECT * FROM TBL_ELEMENTO INNER JOIN" +
                         " TBL_AGENTE ON TBL_ELEMENTO.ID = " +
                         "TBL_AGENTE.ID");
      while(bd.próximoRegistro()){
         agente = new Agente(bd.obterLong("ID"),
                             bd.obterString("descrição"),
                             bd.obterLong("tipoDoToken"),
                             bd.obterLong("estado"),
                             bd.obterLong("lugar"),
                             bd.obterLong("superToken"));
         lista.add(agente);
      }
      bd.finalizarConsulta();
      return lista;
   }
}
```

A – 2 Classe Arco

```
package dinproc;

import java.util.*;

/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */

public class Arco extends Componente {

   // propriedades
   private Lugar lugar;
   private Transição transição;
   private boolean deLugarParaTransição;
   private TipoDeToken tipoDeToken;
   private int peso;

   // Propriedades relacionadas ao BD:
   public long lugarID;
   public long transiçãoID;
   public long tipoDeTokenID;

   // Construtor de objeto vazio
   public Arco() {
   }

   public Arco(long id, long rede, long transição_, long tipo, long lugar_,
               boolean sentidoTransição, int pesoDoArco, int posição) {
      super(id, rede, posição);
      peso = pesoDoArco;
      transiçãoID = transição_;
      lugarID = lugar_;
      tipoDeTokenID = tipo;
      deLugarParaTransição = sentidoTransição;
      lugar = null;
      transição = null;
      tipoDeToken = null;
   }

   public boolean aceitaTransmitirTokens(){
      if(deLugarParaTransição){
         if(lugar.qualCapacidade()-lugar.qualDisponibilidade() >= peso)
         {
            return true;
         }
      }
      else {
         if(lugar.qualDisponibilidade() >= peso)
         {
            return true;
         }
      }
      return false;
   }
```

```
public TipoDeToken obterTipoDeToken(){
   return tipoDeToken;
}

public LinkedList obterTokens(){
   if(deLugarParaTransição){
      return lugar.removerTokens(peso);
   }
   return null;
}

public void enviarTokens(LinkedList lista){
   if(!deLugarParaTransição){
      lugar.adicionarTokens(lista);
   }
}

public void enviarTokens(Token token){
   if(!deLugarParaTransição){
      lugar.adicionarTokens(token);
   }
}

public String toString(){
   String string = "\n{[Arco] ";
   string += " tipoDoTokenID: " + tipoDeTokenID;
   string += ", transiçãoID: " + transiçãoID;
   string += ", lugarID: " + lugarID;
   string += ", peso: " + peso;
   string += ", sentido: L";
   if(deLugarParaTransição) string +="->T"; else string += "<-T";
   if(lugar != null && transição != null && tipoDeToken != null){
      string += "\n\tlugar: " + lugar.obterDescrição();
      string += "\n\ttransição: " + transição.obterDescrição();
      string += "\n\ttipoDoToken: " + tipoDeToken.descrição;
   }
   string += super.toString() + "}";
   return string;
}


//|||||||||||||||||||||||||||| IMPLEMENTADO DE OBJETO DE REDE ||||||||||||||

public String obterDescrição(){
      return  "(Arco ID: " + ID + ", posição na rede: " + posiçãoNaRede +
              ", tipo de Token: " + tipoDeToken.descrição + ", peso: " +
              peso + ")";
}


//|||||||||||||||||||||||||||| HERDADO DE ENTIDADE DO BD ||||||||||||||||||

public void obterReferências(LinkedList lista){
   Object ob;
   ob = obterReferênciaDoID(lista, "dinproc.Lugar", lugarID);
   if(ob instanceof Lugar) lugar = (Lugar)ob;
   else System.out.println("Sistema inconsistente: Arco sem lugar");
   ob = obterReferênciaDoID(lista, "dinproc.Transição", transiçãoID);
   if(ob instanceof Transição) transição = (Transição)ob;
   else System.out.println("Sistema inconsistente: Arco sem Transição");
   ob = obterReferênciaDoID(lista, "dinproc.TipoDeToken", tipoDeTokenID);
   if(ob instanceof TipoDeToken) tipoDeToken = (TipoDeToken)ob;
   else System.out.println("Sistema inconsistente: Arco sem TipoDeToken");
```

```
        super.obterReferências(lista);
    }

    public static LinkedList obterEntidadeDoBD(InterfaceBD bd){
        LinkedList lista = new LinkedList();
        Arco arco;
        bd.iniciarConsulta("SELECT * FROM TBL_COMPONENTE INNER JOIN" +
                           " TBL_ARCO ON TBL_COMPONENTE.ID = " +
                           "TBL_ARCO.ID");
        while(bd.próximoRegistro()){
            arco = new Arco(bd.obterLong("ID"),
                            bd.obterLong("rede"),
                            bd.obterLong("transição"),
                            bd.obterLong("tipoDeToken"),
                            bd.obterLong("lugar"),
                            bd.obterBoolean("deLugarParaTransição"),
                            (int)bd.obterLong("pesoDoArco"),
                            (int)bd.obterLong("posiçãoNaRede"));
            lista.add(arco);
        }
        bd.finalizarConsulta();
        return lista;
    }
}
```

A – 3 Classe Componente

```
package dinproc;

import java.util.*;

/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */

public abstract class Componente extends EntidadeDoBD implements SubRede {

   public long redeID; //não precisa ser referenciado
   public int posiçãoNaRede;

   public Componente(){
   }

   public Componente(long id, long rede, int posição) {
      super(id);
      redeID = rede;
      posiçãoNaRede = posição;
   }


   public String toString(){
      return "\n{[Componente] rede superior: " + redeID +
             ", posição na rede: "+
             posiçãoNaRede +
             super.toString() + "}";
   }



   //|||||||||||||||||||||||||||| IMPLEMENTADO DE OBJETO DE REDE ||||||||||||||

   public int obterPosiçãoNaRedeSuperior(){
      return posiçãoNaRede;
   }

   public long obterIDRedeSuperior(){
      return redeID;
   }



   //|||||||||||||||||||||||||||| HERDADO DE ENTIDADE DO BD ||||||||||||||||||

   public void obterReferências(LinkedList lista){
      //sem ação
   }
}
```

A – 4 Classe Conjunto

```
package dinproc;

import java.util.*;

/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */

public class Conjunto extends Elemento {

   private LinkedList listaDeTokens;
   public boolean ativo = true;

   public Conjunto() {
   }

   public Conjunto(long id, long tipo, long estado_, long lugar_,
                   long superToken_) {
      super(id, tipo, estado_, lugar_, superToken_);
      listaDeTokens = new LinkedList();
   }

   public Conjunto(long id, long tipo, long estado_, long lugar_,
                   long superToken_, LinkedList lista) {
      super(id, tipo, estado_, lugar_, superToken_);
      listaDeTokens = lista;
   }


   public Conjunto(long id, TipoDeToken tipo, LinkedList lista) {
      super(id, tipo);
      listaDeTokens = lista;
   }

   public String toString(){
      String string = "\n{[Conjunto] listaDeTokens, IDs: ";
      if(listaDeTokens != null){
         Iterator i = listaDeTokens.iterator();
         while(i.hasNext()){
            string += ((EntidadeDoBD)i.next()).obterID() + ", ";
         }
         if(listaDeTokens.size() == 0) string += "vazia";
      }
      else string += "vazia";
      string += super.toString() + "}";
      return string;
   }
```

```
//|||||||||||||||||||||||||||||| IMPLEMENTADO DE TOKEN ||||||||||||||||||||||||
public LinkedList obterSubTokensDoTipo(TipoDeToken tipo){
   LinkedList lista = new LinkedList();
   Iterator i = listaDeTokens.iterator();
   Object token;

   if(tipo.equals(tipoDoToken))
   {
      lista.add(this);
   }

   while(i.hasNext()){
      token = i.next();
      if(token instanceof Token){
         lista.addAll(  ((Token)token).obterSubTokensDoTipo(tipo));
      }
   }
   return lista;
}

public LinkedList obterSubTokens(){
   LinkedList lista = new LinkedList(listaDeTokens);
   return lista;
}

public void novoEstado(Estado estado_, Object resp){
   if(resp instanceof Lugar){
      lugar = (Lugar)resp;
   }
   estado = estado_;
   Iterator i = listaDeTokens.iterator();
   while(i.hasNext()){
      ((Token)i.next()).novoEstado(estado_,resp);
   }
}

public void destruir(){
   ativo = false;
}
```

```
//|||||||||||||||||||||||||||||| HERDADO DE ENTIDADE DO BD ||||||||||||||||||

public void obterReferências(LinkedList lista){
   LinkedList entidade;
   String sub[] = {"dinproc.Peça", "dinproc.Pallet",
                   "dinproc.Agente", "dinproc.Conjunto"};
   try{
      for(byte n = 0; n <4; n++){
         entidade = encontrarEntidade(lista, Class.forName(sub[n]));
         if(entidade != null){
            Object ob;
            Iterator i = entidade.iterator();
            while(i.hasNext()){
               ob = i.next();
               if(((Elemento)ob).superTokenID == ID)
                  listaDeTokens.add(ob);
            }
         }
      }
   }
```

```
        catch(ClassNotFoundException e){
            System.out.println("Classe não encontrada - em conjunto");
            System.exit(0);
        }
        super.obterReferências(lista);
    }

    public static LinkedList obterEntidadeDoBD(InterfaceBD bd){
        LinkedList lista = new LinkedList();
        Conjunto conjunto;
        bd.iniciarConsulta("SELECT * FROM TBL_ELEMENTO INNER JOIN" +
                           " TBL_CONJUNTO ON TBL_ELEMENTO.ID = " +
                           "TBL_CONJUNTO.ID");
        while(bd.próximoRegistro()){
            conjunto = new Conjunto(bd.obterLong("ID"),
                                    bd.obterLong("tipoDoToken"),
                                    bd.obterLong("estado"),
                                    bd.obterLong("lugar"),
                                    bd.obterLong("superToken"));
            lista.add(conjunto);
        }
        bd.finalizarConsulta();
        return lista;
    }
}
```

A – 5 Classe Elemento

```
package dinproc;

import java.util.*;

/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */

public abstract class Elemento extends EntidadeDoBD implements Token {

   protected TipoDeToken tipoDoToken;
   protected Estado estado;
   protected Lugar lugar;

   public long tipoDoTokenID;
   public long estadoID;
   public long lugarID;
   public long superTokenID;

   public Elemento() {
   }

   public Elemento(long id, long tipo, long estado_, long lugar_,
                   long superToken_) {
      super(id);
      tipoDoTokenID = tipo;
      estadoID = estado_;
      lugarID = lugar_;
      superTokenID = superToken_;
   }

   public Elemento(long id, TipoDeToken tipo, Estado estado_, Lugar lugar_,
                   long token) {
      super(id);
      tipoDoToken = tipo;
      tipoDoTokenID = tipo.obterID();
      estado = estado_;
      estadoID = estado_.obterID();
      lugar = lugar_;
      lugarID = lugar_.obterID();
      superTokenID = token;
   }

   public Elemento(long id, TipoDeToken tipo) {
      super(id);
      tipoDoToken = tipo;
      tipoDoTokenID = tipo.obterID();
   }

   public String toString(){
      String string = "\n{[Elemento] ";
      string += " tipoDoTokenID: " + tipoDoTokenID;
      string += ", estadoID: " + estadoID;
      string += ", lugarID: " + lugarID;
      string += ", superTokenID: " + superTokenID;
      if(lugar != null && estado != null && tipoDoToken != null){
```

```
            string += "\n\tlugar: " + lugar.obterDescrição();
            string += "\n\testado: " + estado.obterEstado();
            string += "\n\ttipoDoToken: " + tipoDoToken.descrição;
        }
        string +=  super.toString() + "}";
        return string;
    }

    //|||||||||||||||||||||||||||| HERDADO DE ENTIDADE DO BD ||||||||||||||||||

    public void obterReferências(LinkedList lista){
        Object ob;
        ob = obterReferênciaDoID(lista, "dinproc.TipoDeToken", tipoDoTokenID);
        if(ob instanceof TipoDeToken) tipoDoToken = (TipoDeToken)ob;
        else System.out.println("Sistema inconsistente: Elem. sem TipoDeToken");
        ob = obterReferênciaDoID(lista, "dinproc.Estado", estadoID);
        if(ob instanceof Estado) estado = (Estado)ob;
        else System.out.println("Sistema inconsistente: Elem. sem Estado");
        ob = obterReferênciaDoID(lista, "dinproc.Lugar", lugarID);
        if(ob instanceof Lugar) lugar = (Lugar)ob;
    }


    //|||||||||||||||||||||||||||| IMPLEMENTADO DE TOKEN ||||||||||||||||||||||

    public boolean tokenDoTipo(TipoDeToken tipo){
        return tipoDoToken.equals(tipo);
    }

    public LinkedList obterSubTokensDoTipo(TipoDeToken tipo){
        LinkedList lista = new LinkedList();
        if(tipoDoToken.equals(tipo)){
            lista.add(this);
        }
        return lista;
    }

    public LinkedList obterSubTokens(){
        return new LinkedList();
    }

    public long obterSuperToken(){
        return superTokenID;
    }

    public void novoEstado(Estado estado_, Object resp){
        if(resp instanceof Lugar)
            lugar = (Lugar)resp;
        estado = estado_;
    }

    public Estado estadoAtual(){
        return estado;
    }

    public void destruir(){
        //sem ação
    }
}
```

A – 6 Classe EntidadeDoBD

```
package dinproc;

import java.util.*;

/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */

public abstract class EntidadeDoBD {

   protected long ID;

   public EntidadeDoBD() {
   }

   public EntidadeDoBD(long id) {
      ID = id;
   }

   public abstract void obterReferências(LinkedList lista);

   public long obterID(){
      return ID;
   }

   public String toString(){
      return "\n{[EntidadeDoBD]" + " ID: " + ID + "}";
   }

   public static LinkedList encontrarEntidade(LinkedList listas, Class classe){
      LinkedList subLista = new LinkedList();
      Iterator sub = listas.iterator();
      while(sub.hasNext()){
         subLista = (LinkedList)sub.next();
         if(!subLista.isEmpty()){
            if(classe.isInstance(subLista.getFirst())){
               return subLista;
            }
         }
      }
      return new LinkedList();
   }

   public static Object encontrarReferência(LinkedList lista, long id){
      Object ob;
      Iterator i = lista.iterator();
      while(i.hasNext()){
         ob = i.next();
         if(((EntidadeDoBD)ob).obterID() == id){
            return ob;
         }
      }
      return new Object();
   }
```

```
    public static Object obterReferênciaDoID(LinkedList lista,
                                             String classe, long id){
        LinkedList entidade;
        Object ob;
        try{
            entidade = encontrarEntidade(lista, Class.forName(classe));
            if(entidade != null){
                return encontrarReferência(entidade, id);
            }
        }
        catch(ClassNotFoundException e){
            System.out.println("Classe não encontrada: [" + classe +"]");
        }
        return new Object();
    }
}
```

A – 7 Classe Estado

```
package dinproc;

import java.util.*;

/**
* Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */


public class Estado extends EntidadeDoBD {

   String estado;

   private static LinkedList listaDeEstados;


   public Estado() {
   }


   public Estado(long id, String estado_) {
       super(id);
       estado = estado_;
   }


   public static Estado obterRefEstado(String descrição_){
      if(listaDeEstados == null){
         System.out.println("Sem informação dinâmica da lista de Estados");
         System.exit(1);
      }
      Iterator i = listaDeEstados.iterator();
      Estado est;
      while(i.hasNext()){
         est =(Estado)i.next();
         if(est.obterEstado() == descrição_)
         {
            return est;
         }
      }
      est = new Estado(0, descrição_);
      listaDeEstados.add(est);
      return est;
   }


   public String obterEstado(){ return estado;}


   public String toString(){
      String string = "\n{[Estado] estado: " + estado;
      string +=  super.toString() + "}";
      return string;
   }
```

```
    //|||||||||||||||||||||||||||| HERDADO DE ENTIDADE DO BD ||||||||||||||||||
    public void obterReferências(LinkedList lista){
       //sem ação
    }

    public static LinkedList obterEntidadeDoBD(InterfaceBD bd){
       LinkedList lista = new LinkedList();
       Estado estado;
       bd.iniciarConsulta("SELECT * FROM TBL_ESTADO");
       while(bd.próximoRegistro()){
          estado = new Estado(bd.obterLong("ID"),
                              bd.obterString("descrição"));
          lista.add(estado);
       }
       bd.finalizarConsulta();
       listaDeEstados = lista;
       return lista;
    }
}
```

A – 8 Classe InterfaceBD

```
package dinproc;

import java.sql.*;


/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */


public class InterfaceBD {

   private Connection conecção;
   private ResultSet resultado;
   private Statement sentença;
   private ResultSetMetaData metaDados;


   public InterfaceBD() {
   }


   public InterfaceBD(String nomeDaPlanta) {
      String url = "jdbc:odbc:" + nomeDaPlanta; //default: PLANTA
      String usuário = "simulador";
      String senha = "simulador";

      // Carrega o driver para habilitar conexão ao banco de dados
      try {
         Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );

         conecção = DriverManager.getConnection(
            url, usuário, senha);
      }
      catch ( ClassNotFoundException cnfex ) {
         System.err.println(
            "Falha ao carregar JDBC/ODBC driver." );
         cnfex.printStackTrace();
         System.exit( 1 );  // fim do programa: erro
      }
      catch ( SQLException sqlex ) {
         System.err.println( "Não é possível se conectar ao BD." );
         sqlex.printStackTrace();
         System.exit( 1 );  // fim do programa: erro
      }
   }

   protected void finalize(){
      try{
         conecção.close();
      }
      catch ( SQLException sqlex ) {
         sqlex.printStackTrace();
         System.exit( 1 );  // fim do programa: erro
      }
   }
```

```
public void iniciarConsulta(String query){
   try {
      sentença = conecção.createStatement();
      resultado = sentença.executeQuery( query );
      metaDados = resultado.getMetaData();
   }
   catch ( SQLException sqlex ) {
      sqlex.printStackTrace();
   }
}


public void finalizarConsulta(){
   try {
      if(sentença != null)
         sentença.close();
   }
   catch ( SQLException sqlex ) {
      sqlex.printStackTrace();
      System.exit( 1 );  // fim do programa: erro
   }
   sentença = null;
   resultado = null;
   metaDados = null;
   System.gc(); //chama o coletor de lixo
}


public boolean próximoRegistro(){
   try{
      if(!resultado.next()){
         //fim dos registros ou não há registros
         return false;
      }
   }
   catch ( SQLException sqlex ) {
      sqlex.printStackTrace();
      System.exit( 1 );  // fim do programa: erro
   }
   return true;
}


public long obterLong(String nomeColuna){
   try{
         return resultado.getLong(nomeColuna);
   }
   catch ( SQLException sqlex ) {
      sqlex.printStackTrace();
      System.exit( 1 );  // fim do programa: erro
   }
   return 0;
}

public float obterFloat(String nomeColuna){
   try{
         return resultado.getFloat(nomeColuna);
   }
   catch ( SQLException sqlex ) {
      sqlex.printStackTrace();
      System.exit( 1 );  // fim do programa: erro
   }
   return 0;
}
```

```
    public boolean obterBoolean(String nomeColuna){
        try{
              return resultado.getBoolean(nomeColuna);
        }
        catch ( SQLException sqlex ) {
           sqlex.printStackTrace();
           System.exit( 1 );  // fim do programa: erro
        }
        return false;
    }


    public String obterString(String nomeColuna){
        try{
              return resultado.getString(nomeColuna);
        }
        catch ( SQLException sqlex ) {
           sqlex.printStackTrace();
           System.exit( 1 );  // fim do programa: erro
        }
        return "";
    }
}
```

A – 9 Classe Lugar

```
package dinproc;

import java.util.*;

/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */


public class Lugar extends Componente {

   // Capacidade (em número de tokens) associada ao lugar.
   int capacidade;

   // Tipo de Token associado ao lugar;
   private TipoDeToken tipoDeToken;
   public long tipoDeTokenID;

   // Elementos ocupando este lugar da rede:
   LinkedList listaDeTokens;
   public long estadoID;
   private Estado estadoAssociado;


   public Lugar() {
   }


   public Lugar(long id, long rede, int capacidadeDoLugar, long tipoDeToken_,
                int posição, long estado) {
      super(id, rede, posição);
      listaDeTokens = new LinkedList();
      tipoDeTokenID = tipoDeToken_;
      capacidade = capacidadeDoLugar;
      estadoID = estado;
   }


   public int qualCapacidade(){
      return capacidade;
   }


   public int qualDisponibilidade(){
      return capacidade - listaDeTokens.size();
   }


   public Estado obterEstadoAssociado(){
      return estadoAssociado;
   }

   public void adicionarTokens(LinkedList tokens){
      atualizarEstadoTokens(tokens);
      listaDeTokens.addAll(tokens);
   }
```

```
public void adicionarTokens(Token token){
   token.novoEstado(estadoAssociado, this);
   listaDeTokens.add(token);
}


public LinkedList removerTokens(int número){
   LinkedList lista = new LinkedList();
   while(!listaDeTokens.isEmpty() && número > 0){
      lista.add(listaDeTokens.removeFirst());
      número--;
   }
   return lista;
}


private void atualizarEstadoTokens(LinkedList lista){
   Iterator i = lista.iterator();
   Object ob;
   while(i.hasNext()){
      ob = i.next();
      if(ob instanceof Token){

         ((Token)ob).novoEstado(estadoAssociado, this);
      }
   }
}


public String toString(){

   String string = "\n[Lugar] ";
   string += " tipoDoTokenID: " + tipoDeTokenID;
   string += ", capacidade: " + capacidade;

   if(tipoDeToken != null){
      string += ", estado associado: " + estadoAssociado.obterEstado();
      string += ", tipoDoToken: " + tipoDeToken.descrição;
   }

   string += ", lista de tokens (IDs): ";
   if(listaDeTokens != null){
      Iterator i = listaDeTokens.iterator();
      while(i.hasNext()){
         string += ((EntidadeDoBD)i.next()).obterID() + ", ";
      }
      if(listaDeTokens.size() == 0) string += "vazia";
   }

   else string += "vazia";
   string += super.toString() + "}";
   return string;
}


public String obterStringListaDeTokens(){
   String string = "";
   int total = listaDeTokens.size();
   if(listaDeTokens != null){
      if(total <= 10){
         Iterator i = listaDeTokens.iterator();
         while(i.hasNext()){
            string += ((EntidadeDoBD)i.next()).obterID();
            total--;
```

```
            if(total > 0)
               string += ", ";
         }
         if(listaDeTokens.size() == 0) string += "  ";
      }
      else string += " Total de: " + total + " Tokens";
   }
   return string;
}


//|||||||||||||||||||||||||||| IMPLEMENTADO DE OBJETO DE REDE ||||||||||||||

public String obterDescrição(){
      return  "(Lugar ID: " + ID + ", posição na rede: " + posiçãoNaRede +
              ", estado associado: " + estadoAssociado.obterEstado() +
              ", tipo de Token: " + tipoDeToken.descrição +
              ", capacidade: " + listaDeTokens.size() + "/" + capacidade +
              " )";
}


//|||||||||||||||||||||||||||| HERDADO DE ENTIDADE DO BD ||||||||||||||||||

public void obterReferências(LinkedList lista){
   Object ob;

   super.obterReferências(lista);
   ob = obterReferênciaDoID(lista, "dinproc.TipoDeToken", tipoDeTokenID);
   if(ob instanceof TipoDeToken)
      tipoDeToken = (TipoDeToken)ob;
   else
      System.out.println("Sistema inconsistente: Lugar sem TipoDeToken");

   ob = obterReferênciaDoID(lista, "dinproc.Estado", estadoID);
   if(ob instanceof Estado)
      estadoAssociado= (Estado)ob;
   else
      System.out.println("Sistema inconsistente: Lugar sem Est. assoc.");

   LinkedList entidade;
   String sub[] = {"dinproc.Peça", "dinproc.Pallet",
                   "dinproc.Agente", "dinproc.Conjunto"};

   try{
      for(byte n = 0; n <4; n++){
         entidade = encontrarEntidade(lista, Class.forName(sub[n]));
         if(entidade != null){
            Iterator i = entidade.iterator();
            while(i.hasNext()){
               ob = i.next();
               if(((Elemento)ob).lugarID == ID &&
                  ((Elemento)ob).tipoDoTokenID == tipoDeTokenID)
                  listaDeTokens.add(ob);
            }
         }
      }
   }
   catch(ClassNotFoundException e){
      System.out.println("Classe não encontrada - em conjunto");
      System.exit(0);
   }
}
```

```
    public static LinkedList obterEntidadeDoBD(InterfaceBD bd){
        LinkedList lista = new LinkedList();
        Lugar lugar;
        bd.iniciarConsulta("SELECT * FROM TBL_COMPONENTE INNER JOIN" +
                           " TBL_LUGAR ON TBL_COMPONENTE.ID = " +
                           "TBL_LUGAR.ID");
        while(bd.próximoRegistro()){
            lugar = new Lugar(bd.obterLong("ID"),
                              bd.obterLong("rede"),
                              (int)bd.obterLong("capacidade"),
                              bd.obterLong("tipoDeToken"),
                              (int)bd.obterLong("posiçãoNaRede"),
                              bd.obterLong("estado"));
            lista.add(lugar);
        }
        bd.finalizarConsulta();
        return lista;
    }
}
```

A – 10 Interface SubRede

```
package dinproc;

/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */


public interface SubRede {

   public int obterPosiçãoNaRedeSuperior();

   public long obterIDRedeSuperior();

   public String obterDescrição();

}
```

A – 11 Classe OP_Desmontagem

```
package dinproc;

import java.util.*;

/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */


public class OP_Desmontagem extends Operação {

   // Componentes da sub-rede:
   Arco AGD_DESMNT__iniciar;
   Arco desmnt__PL_AGD_DISP;
   Arco desmnt__AGD_OP;
   Arco iniciar__DESMNT;
   Arco DESMNT__desmnt;
   Transição iniciar;
   Transição desmnt;


   public OP_Desmontagem() {
   }


   public OP_Desmontagem(long id, String descrição_, int posição, long redePai,
                         long tempoInício_, long tempoDaOperação_) {
      super(id, descrição_, posição, redePai, tempoInício_, tempoDaOperação_);
   }


   public String toString(){
      String string = "\n{[Desmontagem] ";
      string += "\n\tAGD_DESMNT__iniciar: ";
      if(AGD_DESMNT__iniciar != null)
         string += AGD_DESMNT__iniciar.obterDescrição();
      else string += "nulo";
      string += "\n\tdesmnt__PL_AGD_DISP: ";
      if(desmnt__PL_AGD_DISP != null)
         string += desmnt__PL_AGD_DISP.obterDescrição();
      else string += "nulo";
      string += "\n\tdesmnt__AGD_OP: ";
      if(desmnt__AGD_OP != null)
         string += desmnt__AGD_OP.obterDescrição();
      else string += "nulo";
      string += "\n\tiniciar__DESMNT: ";
      if(iniciar__DESMNT != null)
         string += iniciar__DESMNT.obterDescrição();
      else string += "nulo";
      string += "\n\tDESMNT__desmnt: ";
      if(DESMNT__desmnt != null)
         string += DESMNT__desmnt.obterDescrição();
      else string += "nulo";
      string += "\n\tiniciar: ";
      if(iniciar != null)
         string += iniciar.obterDescrição();
```

```
    else string += "nulo";
    string += "\n\tdesmnt: ";
    if(desmnt != null)
       string += desmnt.obterDescrição();
    else string += "nulo";
    string += super.toString() + "}";
    return string;
 }


 //|||||||||||||||||||||||||||| HERDADO DE ENTIDADE DO BD ||||||||||||||||||

 public void obterReferências(LinkedList lista){
    super.obterReferências(lista);
    Iterator i = listaDeComponentes.iterator();
    while(i.hasNext()){
       Object ob = i.next();
       int posição = ((SubRede)ob).obterPosiçãoNaRedeSuperior();
       switch(posição){
          case 1:     //"AGD_DESMNT__iniciar"
             AGD_DESMNT__iniciar = (Arco)ob;
             break;
          case 7:     //"desmnt__AGD_OP":
             desmnt__AGD_OP = (Arco)ob;
             break;
          case 2:     //"iniciar":
             iniciar = (Transição)ob;
             break;
          case 5:     //"desmnt":
             desmnt = (Transição)ob;
             break;
          case 3:     //"iniciar__DESMNT"
             iniciar__DESMNT = (Arco)ob;
             break;
          case 4:     //"DESMNT__desmnt"
             DESMNT__desmnt = (Arco)ob;
             break;
          case 6:    //"desmnt__PL_AGD_DISP"
             desmnt__PL_AGD_DISP = (Arco)ob;
             break;
       }
    }
 }


 public static LinkedList obterEntidadeDoBD(InterfaceBD bd){
    LinkedList lista = new LinkedList();
    OP_Desmontagem desmnt;
    bd.iniciarConsulta("SELECT * FROM TBL_OPERACAO INNER JOIN" +
                       " TBL_OP_DESMONTAGEM ON TBL_OPERACAO.ID = " +
                       "TBL_OP_DESMONTAGEM.ID");
    while(bd.próximoRegistro()){
       desmnt = new OP_Desmontagem(bd.obterLong("ID"),
                                   bd.obterString("descrição"),
                                   (int)bd.obterLong("posiçãoNaRede"),
                                    bd.obterLong("rede"),
                                    bd.obterLong("tempoInício"),
                                    bd.obterLong("tempoDaOperação"));
       lista.add(desmnt);
    }
    bd.finalizarConsulta();
    return lista;
 }
```

```
    //||||||||||||||||||||||||||||| HERDADOS DE OPERAÇÃO ||||||||||||||||||||||||||

    protected void iniciarOperação(){
       LinkedList tokenBuffer;
       if(iniciar.verificarCondiçãoParaTransição()){
          iniciar__DESMNT.enviarTokens(AGD_DESMNT__iniciar.obterTokens());
       }
    }


    protected void finalizarOperação(){
       if(desmnt.verificarCondiçãoParaTransição()){
          TipoDeToken tipoPeça = desmnt__AGD_OP.obterTipoDeToken();
          TipoDeToken tipoPallet = desmnt__PL_AGD_DISP.obterTipoDeToken();
          TipoDeToken tipoConjunto = iniciar__DESMNT.obterTipoDeToken();
          LinkedList tokens;
          tokens = DESMNT__desmnt.obterTokens();
          Iterator i = tokens.iterator();
          Object ob;
          while(i.hasNext()){
             ob = i.next();
             if(ob instanceof Token){
                desmnt__AGD_OP.enviarTokens(
                ((Token)ob).obterSubTokensDoTipo(tipoPeça));
                desmnt__PL_AGD_DISP.enviarTokens(
                ((Token)ob).obterSubTokensDoTipo(tipoPallet));
                ((Token)ob).destruir();
             }
          }
       }
    }


    protected boolean podeIniciar(){
       return iniciar.verificarCondiçãoParaTransição();
    }


    protected boolean podeFinalizar(){
       return desmnt.verificarCondiçãoParaTransição();
    }
}
```

A – 12 Classe OP_Metrologia

```
package dinproc;

import java.util.*;

/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */


public class OP_Metrologia extends Operação {

   // Componentes da sub-rede:
   Arco AGD_MTRL__iniciar;
   Arco iniciar__OP;
   Arco OP__aprovado;
   Arco OP__re_trabalho;
   Arco OP__refugo;
   Arco aprovado__AGD_;
   Arco aprovado__AGD_DISP;
   Arco re_trabalho__RET;
   Arco re_trabalho__AGD_DISP;
   Arco refugo__REF;
   Arco refugo__AGD_DISP;
   Arco AGD_DISP__iniciar;
   Transição iniciar;
   Transição aprovado;
   Transição re_trabalho;
   Transição refugo;

   //probabilidades: 0 a 100%
   float pRefugo;
   float pReTrabalho;

   //resultado da metrologia
   final int APROVADO = 0, RETRABALHO = 1, REFUGO = 2;
   int resultado;


   public OP_Metrologia() {
   }


   public OP_Metrologia(long id, String descrição_, int posição, long redePai,
                        long tempoInício_, long tempoDaOperação_, float pRefugo_,
                        float pReTrabalho_) {
      super(id, descrição_, posição, redePai, tempoInício_, tempoDaOperação_);
      pRefugo = pRefugo_;
      pReTrabalho = pReTrabalho_;
   }


   public String toString(){
      String string = "\n{[Metrologia] ";
      string += "\n\tAGD_MTRL__iniciar: ";
      if(AGD_MTRL__iniciar != null)
         string += AGD_MTRL__iniciar.obterDescrição();
```

```
        else string += "nulo";
        string += "\n\tiniciar__OP: ";
        if(iniciar__OP != null)
           string += iniciar__OP.obterDescrição();
        else string += "nulo";
        string += "\n\tOP__aprovado: ";
        if(OP__aprovado != null)
           string += OP__aprovado.obterDescrição();
        else string += "nulo";
        string += "\n\tOP__re_trabalho: ";
        if(OP__re_trabalho != null)
           string += OP__re_trabalho.obterDescrição();
        else string += "nulo";
        string += "\n\tOP__refugo: ";
        if(OP__refugo != null)
           string += OP__refugo.obterDescrição();
        else string += "nulo";
        string += "\n\taprovado__AGD_: ";
        if(aprovado__AGD_ != null)
           string += aprovado__AGD_.obterDescrição();
        else string += "nulo";
        string += "\n\taprovado__AGD_DISP: ";
        if(aprovado__AGD_DISP != null)
           string += aprovado__AGD_DISP.obterDescrição();
        else string += "nulo";
        string += "\n\tre_trabalho__RET: ";
        if(re_trabalho__RET != null)
           string += re_trabalho__RET.obterDescrição();
        else string += "nulo";
        string += "\n\tre_trabalho__AGD_DISP: ";
        if(re_trabalho__AGD_DISP != null)
           string += re_trabalho__AGD_DISP.obterDescrição();
        else string += "nulo";
              string += "\n\trefugo__REF: ";
        if(refugo__REF != null)
           string += refugo__REF.obterDescrição();
        else string += "nulo";
        string += "\n\trefugo__AGD_DISP: ";
        if(refugo__AGD_DISP != null)
           string += refugo__AGD_DISP.obterDescrição();
        else string += "nulo";
        string += "\n\tAGD_DISP__iniciar: ";
        if(AGD_DISP__iniciar != null)
           string += AGD_DISP__iniciar.obterDescrição();
        else string += "nulo";
        string += "\n\tiniciar: ";
        if(iniciar != null)
           string += iniciar.obterDescrição();
        else string += "nulo";
        string += "\n\taprovado: ";
        if(aprovado != null)
           string += aprovado.obterDescrição();
        else string += "nulo";
        string += "\n\tre_trabalho: ";
        if(re_trabalho != null)
           string += re_trabalho.obterDescrição();
        else string += "nulo";
        string += "\n\trefugo: ";
        if(refugo != null)
           string += refugo.obterDescrição();
        else string += "nulo";
        string += super.toString() + "}";
        return string;
    }
```

```
//|||||||||||||||||||||||||||| HERDADO DE ENTIDADE DO BD ||||||||||||||||||
public void obterReferências(LinkedList lista){
   super.obterReferências(lista);
   Iterator i = listaDeComponentes.iterator();
   while(i.hasNext()){
      Object ob = i.next();
      int posição = ((SubRede)ob).obterPosiçãoNaRedeSuperior();
      switch(posição){
         case 0:     //"AGD_MTRL__iniciar"
            AGD_MTRL__iniciar = (Arco)ob;
            break;
         case 1:     //"iniciar__OP":
            iniciar__OP = (Arco)ob;
            break;
         case 2:     //"OP__aprovado":
            OP__aprovado = (Arco)ob;
            break;
         case 3:     //"OP__re_trabalho":
            OP__re_trabalho = (Arco)ob;
            break;
         case 4:     //"OP__refugo":
            OP__refugo = (Arco)ob;
            break;
         case 5:     //"aprovado__AGD_":
            aprovado__AGD_ = (Arco)ob;
            break;
         case 6:     //"aprovado__AGD_DISP":
            aprovado__AGD_DISP = (Arco)ob;
            break;
         case 7:     //"re_trabalho__RET":
            re_trabalho__RET = (Arco)ob;
            break;
         case 8:     //"re_trabalho__AGD_DISP":
            re_trabalho__AGD_DISP = (Arco)ob;
            break;
         case 9:     //"refugo__REF":
            refugo__REF = (Arco)ob;
            break;
         case 10:     //"refugo__AGD_DISP":
            refugo__AGD_DISP = (Arco)ob;
            break;
         case 11:     //"AGD_DISP__iniciar":
            AGD_DISP__iniciar = (Arco)ob;
            break;
         case 12:     //"iniciar":
            iniciar = (Transição)ob;
            break;
         case 13:     //"aprovado":
            aprovado = (Transição)ob;
            break;
         case 14:     //"re_trabalho":
            re_trabalho = (Transição)ob;
            break;
         case 15:     //"refugo":
            refugo = (Transição)ob;
            break;
      }
   }
}

public static LinkedList obterEntidadeDoBD(InterfaceBD bd){
   LinkedList lista = new LinkedList();
   OP_Metrologia mtrl;
   bd.iniciarConsulta("SELECT * FROM TBL_OPERACAO INNER JOIN" +
                      " TBL_OP_METROLOGIA ON TBL_OPERACAO.ID = " +
```

```
                        "TBL_OP_METROLOGIA.ID");
    while(bd.próximoRegistro()){
        mtrl = new OP_Metrologia(bd.obterLong("ID"),
                                 bd.obterString("descrição"),
                                 (int)bd.obterLong("posiçãoNaRede"),
                                 bd.obterLong("rede"),
                                 bd.obterLong("tempoInício"),
                                 bd.obterLong("tempoDaOperação"),
                                 bd.obterFloat("pRetrabalho"),
                                 bd.obterFloat("pRefugo"));
        lista.add(mtrl);
    }
    bd.finalizarConsulta();
    return lista;
}


//|||||||||||||||||||||||||||| HERDADOS DE OPERAÇÃO ||||||||||||||||||||||

protected void iniciarOperação(){
    if(iniciar.verificarCondiçãoParaTransição()){
        //Identifica resultado da metrologia
        double randômico = Math.random();
        if(randômico < pRefugo)
            resultado = REFUGO;
        else if(pRefugo <= randômico && randômico < (pRefugo + pReTrabalho))
            resultado = RETRABALHO;
        else
            resultado = APROVADO;

        //Processa rede:
        TipoDeToken tipoPeça = AGD_MTRL__iniciar.obterTipoDeToken();
        TipoDeToken tipoConjunto = iniciar__OP.obterTipoDeToken();
        TipoDeToken tipoAgente = AGD_DISP__iniciar.obterTipoDeToken();

        LinkedList tokens;
        tokens = AGD_MTRL__iniciar.obterTokens(); //recebe peças para operação
        tokens.addAll(AGD_DISP__iniciar.obterTokens());//recebe agente
        Conjunto conj = new Conjunto(0, tipoConjunto, tokens);
        iniciar__OP.enviarTokens(conj);//envia tokens para OP
    }
}

protected void finalizarOperação(){
    if(podeFinalizar()){
        //Tipos:
        TipoDeToken tipoPeça = AGD_MTRL__iniciar.obterTipoDeToken();
        TipoDeToken tipoConjunto = iniciar__OP.obterTipoDeToken();
        TipoDeToken tipoAgente = AGD_DISP__iniciar.obterTipoDeToken();
        //itens
        LinkedList tokens, peças, agentes;
        agentes = new LinkedList();
        peças = new LinkedList();
        tokens = OP__aprovado.obterTokens();
        Iterator i = tokens.iterator();
        Object ob;
        while(i.hasNext()){
            ob = i.next();
            if(ob instanceof Token){
                peças.addAll(((Token)ob).obterSubTokensDoTipo(tipoPeça));
                agentes.addAll(((Token)ob).obterSubTokensDoTipo(tipoAgente));
                ((Token)ob).destruir();
            }
        }
```

```
            switch(resultado){
               case APROVADO:
                  aprovado__AGD_.enviarTokens(peças);
                  aprovado__AGD_DISP.enviarTokens(agentes);
                  break;
               case RETRABALHO:
                  re_trabalho__RET.enviarTokens(peças);
                  re_trabalho__AGD_DISP.enviarTokens(agentes);
                  break;
               case REFUGO:
                  refugo__REF.enviarTokens(peças);
                  refugo__AGD_DISP.enviarTokens(agentes);
                  break;
            }
         }
      }

      protected boolean podeIniciar(){
         return iniciar.verificarCondiçãoParaTransição();
      }

      protected boolean podeFinalizar(){
         boolean verificação = false;
         switch(resultado){
            case APROVADO:
               verificação = aprovado.verificarCondiçãoParaTransição();
               break;
            case RETRABALHO:
               verificação = re_trabalho.verificarCondiçãoParaTransição();
               break;
            case REFUGO:
               verificação = refugo.verificarCondiçãoParaTransição();
               break;
         }
         return verificação;
      }
   }
```

A – 13 Classe OP_Montagem

```
package dinproc;

import java.util.*;


/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */


public class OP_Montagem extends Operação {

   // Componentes da sub-rede:
   Arco AGD_MNT__iniciar;
   Arco PL_AGD_DISP__iniciar;
   Arco iniciar__MNT;
   Arco MNT__mnt;
   Arco mnt__AGD_;
   Transição iniciar;
   Transição mnt;


   public OP_Montagem() {
   }


   public OP_Montagem(long id, String descrição_, int posição, long redePai,
                      long tempoInício_, long tempoDaOperação_) {
      super(id, descrição_, posição, redePai, tempoInício_, tempoDaOperação_);
   }


   public String toString(){
      String string = "\n{[Montagem] ";
      string += "\n\tAGD_MNT__iniciar: ";
      if(AGD_MNT__iniciar != null)
         string += AGD_MNT__iniciar.obterDescrição();
      else string += "nulo";
      string += "\n\tPL_AGD_DISP__iniciar: ";
      if(PL_AGD_DISP__iniciar != null)
         string += PL_AGD_DISP__iniciar.obterDescrição();
      else string += "nulo";
      string += "\n\tiniciar__MNT: ";
      if(iniciar__MNT != null)
         string += iniciar__MNT.obterDescrição();
      else string += "nulo";
      string += "\n\tMNT__mnt: ";
      if(MNT__mnt != null)
         string += MNT__mnt.obterDescrição();
      else string += "nulo";
      string += "\n\tmnt__AGD_: ";
      if(mnt__AGD_ != null)
         string += mnt__AGD_.obterDescrição();
      else string += "nulo";
      string += "\n\tiniciar: ";
      if(iniciar != null)
```

```
            string += iniciar.obterDescrição();
        else string += "nulo";
        string += "\n\tmnt: ";
        if(mnt != null)
            string += mnt.obterDescrição();
        else string += "nulo";
        string += super.toString() + "}";
        return string;
    }

    //|||||||||||||||||||||||||||| HERDADO DE ENTIDADE DO BD ||||||||||||||||||

    public void obterReferências(LinkedList lista){
        super.obterReferências(lista);
        Iterator i = listaDeComponentes.iterator();
        while(i.hasNext()){
            Object ob = i.next();
            int posição = ((SubRede)ob).obterPosiçãoNaRedeSuperior();
            switch(posição){
                case 0:     //"AGD_MNT__iniciar"
                    AGD_MNT__iniciar = (Arco)ob;
                    break;
                case 1:     //"PL_AGD_DISP__iniciar":
                    PL_AGD_DISP__iniciar = (Arco)ob;
                    break;
                case 2:     //"iniciar__MNT":
                    iniciar__MNT = (Arco)ob;
                    break;
                case 3:     //"MNT__mnt":
                    MNT__mnt = (Arco)ob;
                    break;
                case 4:     //"mnt__AGD_":
                    mnt__AGD_ = (Arco)ob;
                    break;
                case 5:     //"iniciar":
                    iniciar = (Transição)ob;
                    break;
                case 6:     //"mnt":
                    mnt = (Transição)ob;
                    break;
            }
        }
    }

    public static LinkedList obterEntidadeDoBD(InterfaceBD bd){
        LinkedList lista = new LinkedList();
        OP_Montagem mnt;
        bd.iniciarConsulta("SELECT * FROM TBL_OPERACAO INNER JOIN" +
                           " TBL_OP_MONTAGEM ON TBL_OPERACAO.ID = " +
                           "TBL_OP_MONTAGEM.ID");
        while(bd.próximoRegistro()){
            mnt = new OP_Montagem(bd.obterLong("ID"),
                                  bd.obterString("descrição"),
                                  (int)bd.obterLong("posiçãoNaRede"),
                                  bd.obterLong("rede"),
                                  bd.obterLong("tempoInício"),
                                  bd.obterLong("tempoDaOperação"));
            lista.add(mnt);
        }
        bd.finalizarConsulta();
        return lista;
    }
```

```
    //||||||||||||||||||||||||||||| HERDADOS DE OPERAÇÃO |||||||||||||||||||||||
    protected void iniciarOperação(){
       if(iniciar.verificarCondiçãoParaTransição()){
          TipoDeToken tipoPeça = AGD_MNT__iniciar.obterTipoDeToken();
          TipoDeToken tipoPallet = PL_AGD_DISP__iniciar.obterTipoDeToken();
          TipoDeToken tipoConjunto= iniciar__MNT.obterTipoDeToken();

          LinkedList tokens;
          tokens = AGD_MNT__iniciar.obterTokens(); //recebe peças para montag.
          tokens.addAll(PL_AGD_DISP__iniciar.obterTokens());//recebe pallet
          Conjunto conj = new Conjunto(0, tipoConjunto, tokens);
          iniciar__MNT.enviarTokens(conj);//envia tokens para MNT
       }
    }


    protected void finalizarOperação(){
       if(mnt.verificarCondiçãoParaTransição()){
         mnt__AGD_.enviarTokens(MNT__mnt.obterTokens());
       }
    }


    protected boolean podeIniciar(){
       return iniciar.verificarCondiçãoParaTransição();
    }


    protected boolean podeFinalizar(){
       return mnt.verificarCondiçãoParaTransição();
    }

}
```

A – 14 Classe OP_Processamento

```
package dinproc;

import java.util.*;

/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */


public class OP_Processamento extends Operação {

   // Componentes da sub-rede:
   Arco AGD_OP__iniciar;
   Arco iniciar__OP;
   Arco OP__terminar;
   Arco terminar__AGD_;
   Arco terminar__AGD_DISP;
   Arco AGD_DISP__iniciar;
   Transição iniciar;
   Transição terminar;


   public OP_Processamento() {
   }

   public OP_Processamento(long id, String descrição_,
                           int posição, long redePai,
                           long tempoInício_, long tempoDaOperação_) {
      super(id, descrição_, posição, redePai, tempoInício_, tempoDaOperação_);
   }


   public String toString(){
      String string = "\n{[Processamento] ";
      string += "\n\tAGD_OP__iniciar: ";
      if(AGD_OP__iniciar != null)
         string += AGD_OP__iniciar.obterDescrição();
      else string += "nulo";
      string += "\n\tiniciar__OP: ";
      if(iniciar__OP != null)
         string += iniciar__OP.obterDescrição();
      else string += "nulo";
      string += "\n\tOP__terminar: ";
      if(OP__terminar != null)
         string += OP__terminar.obterDescrição();
      else string += "nulo";
      string += "\n\tterminar__AGD_: ";
      if(terminar__AGD_ != null)
         string += terminar__AGD_.obterDescrição();
      else string += "nulo";
      string += "\n\tterminar__AGD_DISP: ";
      if(terminar__AGD_DISP != null)
         string += terminar__AGD_DISP.obterDescrição();
      else string += "nulo";
      string += "\n\tAGD_DISP__iniciar: ";
      if(AGD_DISP__iniciar != null)
```

```
            string += AGD_DISP__iniciar.obterDescrição();
        else string += "nulo";
        string += "\n\tiniciar: ";
        if(iniciar != null)
            string += iniciar.obterDescrição();
        else string += "nulo";
        string += "\n\tterminar: ";
        if(terminar != null)
            string += terminar.obterDescrição();
        else string += "nulo";
        string += super.toString() + "}";
        return string;
    }


    //||||||||||||||||||||||||||| HERDADO DE ENTIDADE DO BD ||||||||||||||||||

    public void obterReferências(LinkedList lista){
        super.obterReferências(lista);
        Iterator i = listaDeComponentes.iterator();
        while(i.hasNext()){
            Object ob = i.next();
            int posição = ((SubRede)ob).obterPosiçãoNaRedeSuperior();
            switch(posição){
                case 0:     //"AGD_OP__iniciar"
                    AGD_OP__iniciar = (Arco)ob;
                    break;
                case 1:     //"iniciar__OP":
                    iniciar__OP = (Arco)ob;
                    break;
                case 2:     //"OP__terminar":
                    OP__terminar = (Arco)ob;
                    break;
                case 3:     //"terminar__AGD_":
                    terminar__AGD_ = (Arco)ob;
                    break;
                case 4:     //"terminar__AGD_DISP":
                    terminar__AGD_DISP = (Arco)ob;
                    break;
                case 5:     //"AGD_DISP__iniciar":
                    AGD_DISP__iniciar = (Arco)ob;
                    break;
                case 6:     //"iniciar":
                    iniciar = (Transição)ob;
                    break;
                case 7:     //"terminar":
                    terminar = (Transição)ob;
                    break;
            }
        }
    }


    public static LinkedList obterEntidadeDoBD(InterfaceBD bd){
        LinkedList lista = new LinkedList();
        OP_Processamento process;
        bd.iniciarConsulta("SELECT * FROM TBL_OPERACAO INNER JOIN" +
                           " TBL_OP_PROCESSAMENTO ON TBL_OPERACAO.ID = " +
                           "TBL_OP_PROCESSAMENTO.ID");
        while(bd.próximoRegistro()){
            process = new OP_Processamento(bd.obterLong("ID"),
                                           bd.obterString("descrição"),
                                           (int)bd.obterLong("posiçãoNaRede"),
                                           bd.obterLong("rede"),
```

```
                                        bd.obterLong("tempoInício"),
                                        bd.obterLong("tempoDaOperação"));
            lista.add(process);
        }
        bd.finalizarConsulta();
        return lista;
    }


    //||||||||||||||||||||||||||||| HERDADOS DE OPERAÇÃO |||||||||||||||||||||||

    protected void iniciarOperação(){
        if(iniciar.verificarCondiçãoParaTransição()){
            TipoDeToken tipoPeça = AGD_OP__iniciar.obterTipoDeToken();
            TipoDeToken tipoConjunto = iniciar__OP.obterTipoDeToken();
            TipoDeToken tipoAgente = AGD_DISP__iniciar.obterTipoDeToken();

            LinkedList tokens;
            tokens = AGD_OP__iniciar.obterTokens(); //recebe peças para operação
            tokens.addAll(AGD_DISP__iniciar.obterTokens());//recebe agente
            Conjunto conj = new Conjunto(0, tipoConjunto, tokens);
            iniciar__OP.enviarTokens(conj);//envia tokens para OP
        }
    }


    protected void finalizarOperação(){
        LinkedList tokenBuffer;
        if(terminar.verificarCondiçãoParaTransição()){
            TipoDeToken tipoPeça = AGD_OP__iniciar.obterTipoDeToken();
            TipoDeToken tipoConjunto = iniciar__OP.obterTipoDeToken();
            TipoDeToken tipoAgente = AGD_DISP__iniciar.obterTipoDeToken();

            LinkedList tokens;
            tokens = OP__terminar.obterTokens(); //recebe peças para operação
            Iterator i = tokens.iterator();
            Object ob;
            while(i.hasNext()){
                ob = i.next();
                if(ob instanceof Token){
                    terminar__AGD_.enviarTokens(
                    ((Token)ob).obterSubTokensDoTipo(tipoPeça));
                    terminar__AGD_DISP.enviarTokens(
                    ((Token)ob).obterSubTokensDoTipo(tipoAgente));
                    ((Token)ob).destruir();
                }
            }
        }
    }


    protected boolean podeIniciar(){
        return iniciar.verificarCondiçãoParaTransição();
    }

    protected boolean podeFinalizar(){
        return terminar.verificarCondiçãoParaTransição();
    }
}
```

A – 15 Classe Operação

```
package dinproc;

import java.util.*;

/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */


public abstract class Operação extends Rede implements SubRede {

   protected long tempoInício;
   private long tempoDaOperação;
   public String descrição;


   //rede superior:
   public long redeID; //não referenciada
   private int posiçãoNaRede;


   public Operação() {
   }


   public Operação(long id, String descrição_, int posição, long redePai,
                   long tempoInício_, long tempoDaOperação_) {
      super(id);
      redeID = redePai;
      tempoDaOperação = tempoDaOperação_;
      posiçãoNaRede = posição;
      descrição = descrição_;
      if(tempoInício_ < 0)
         tempoInício = Long.MIN_VALUE;
      else
         tempoInício = tempoInício_;
   }


   protected abstract void iniciarOperação();

   protected abstract void finalizarOperação();

   protected abstract boolean podeIniciar();

   protected abstract boolean podeFinalizar();


   public String toString(){
      String string = "\n[Operação] ";
      string += " redeID: " + redeID;
      string += ", descrição: " + descrição;
      string += ", posiçãoNaRede: " + posiçãoNaRede;
      string += ", tempoInício: " + tempoInício;
      string += ", tempoDaOperação: " + tempoDaOperação;
```

```
        string += super.toString() + "}";
        return string;
    }


    //|||||||||||||||||||||||||||| IMPLEMENTADO DE OBJETO DE REDE ||||||||||||||

    public int obterPosiçãoNaRedeSuperior(){
        return posiçãoNaRede;
    }


    public long obterIDRedeSuperior(){
        return redeID;
    }


    public String obterDescrição(){
          return  "(Operação ID: " + ID + ", posição na rede: " + posiçãoNaRede +
                  ", descrição: " + descrição + ", tempo início " + tempoInício
                  + ", tempo Operação: " + tempoDaOperação + ")";
    }


    //|||||||||||||||||||||||||||| HERDADO DE REDE ||||||||||||||||||||||||||||

    public final void processarRede(long fim, long tempoAtual){
        //se tempo > próximaTransição: deveria lançar excessão.
        if(fim == obterPróximaTransição(tempoAtual)){
            if(tempoInício >= 0){//operação já inicida
                finalizarOperação();
                tempoInício = Long.MIN_VALUE; //Não iniciada
            }
            else {//operação não iniciada
                iniciarOperação();
                tempoInício = tempoAtual;
            }
        }
    }


    public final long obterPróximaTransição(long tempoAtual){
        if(podeFinalizar() && tempoInício >= 0){
            return tempoInício + tempoDaOperação;
        }
        if(podeIniciar()){
            return tempoAtual;
        }
        return Long.MAX_VALUE;
    }


    //|||||||||||||||||||||||||||| HERDADO DE ENTIDADE DO BD ||||||||||||||||||

    public void obterReferências(LinkedList lista){
        LinkedList entidade;
        Object ob;
        String sub[] = {"dinproc.Arco", "dinproc.Transição",
                        "dinproc.Lugar"};
        try{
            for(byte n = 0; n <3; n++){
                entidade = encontrarEntidade(lista, Class.forName(sub[n]));
                if(entidade != null){
```

```
                Iterator i = entidade.iterator();
                while(i.hasNext()){
                   ob = i.next();
                   if(((Componente)ob).redeID == ID)
                      listaDeComponentes.add(ob);
                }
             }
          }
       }
       catch(ClassNotFoundException e){
          System.out.println("Classe não encontrada - em conjunto");
          System.exit(0);
       }
    }


    public static LinkedList obterEntidadeDoBD(InterfaceBD bd){
       LinkedList lista = new LinkedList();
       lista.addAll(OP_Desmontagem.obterEntidadeDoBD(bd));
       lista.addAll(OP_Processamento.obterEntidadeDoBD(bd));
       lista.addAll(OP_Montagem.obterEntidadeDoBD(bd));
       lista.addAll(OP_Metrologia.obterEntidadeDoBD(bd));
       return lista;
    }

}
```

A – 16 Classe Pallet

```
package dinproc;

import java.util.*;

/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */


public class Pallet extends Elemento {

   private int capacidade;

   public Pallet() {
   }


   public Pallet(long id, int capacidade_, long tipo, long estado_, long lugar_,
                 long superToken_) {
      super(id, tipo, estado_, lugar_, superToken_);
      capacidade = capacidade_;
   }


   public String toString(){
      return "\n{[Pallet] capacidade: " + capacidade + super.toString() + "}";
   }



   //|||||||||||||||||||||||||||| HERDADO DE ENTIDADE DO BD ||||||||||||||||||

   public static LinkedList obterEntidadeDoBD(InterfaceBD bd){
      LinkedList lista = new LinkedList();
      Pallet pallet;
      bd.iniciarConsulta("SELECT * FROM TBL_ELEMENTO INNER JOIN" +
                         " TBL_PALLET ON TBL_ELEMENTO.ID = " +
                         "TBL_PALLET.ID");
      while(bd.próximoRegistro()){
         pallet = new Pallet(bd.obterLong("ID"),
                             (int)bd.obterLong("capacidade"),
                             bd.obterLong("tipoDoToken"),
                             bd.obterLong("estado"),
                             bd.obterLong("lugar"),
                             bd.obterLong("superToken"));
         lista.add(pallet);
      }
      bd.finalizarConsulta();
      return lista;
   }

}
```

A – 17 Classe Peça

```
package dinproc;

import java.util.*;

/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */


public class Peça extends Elemento {

   public long processoID; //não referenciado

   public Peça() {
   }


   public Peça(long id, long processo, long tipo, long estado_, long lugar_,
               long superToken_) {
      super(id, tipo, estado_, lugar_, superToken_);
      processoID = processo;
   }


   public String toString(){
      return "\n{[Peça] processo: " + processoID + super.toString() + "}";
   }



   //||||||||||||||||||||||||||| HERDADO DE ENTIDADE DO BD ||||||||||||||||||

   public static LinkedList obterEntidadeDoBD(InterfaceBD bd){
      LinkedList lista = new LinkedList();
      Peça peça;
      bd.iniciarConsulta("SELECT * FROM TBL_ELEMENTO INNER JOIN" +
                         " TBL_PECA ON TBL_ELEMENTO.ID = " +
                         "TBL_PECA.ID");
      while(bd.próximoRegistro()){
         peça = new Peça(bd.obterLong("ID"),
                         bd.obterLong("processo"),
                         bd.obterLong("tipoDoToken"),
                         bd.obterLong("estado"),
                         bd.obterLong("lugar"),
                         bd.obterLong("superToken"));
         lista.add(peça);
      }
      bd.finalizarConsulta();
      return lista;
   }

}
```

A – 18 Classe Processo

```
package dinproc;

import java.util.*;

/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */

public abstract class Processo extends Rede {

   public long loteID; //não referenciado
   public String descrição;

   //resultado do cálculo da próxima transição.
   private Operação próximaOperação;


   public Processo() {
   }


   public Processo(long id, String descrição_, long lote) {
      super(id);
      loteID = lote;
      descrição = descrição_;
   }


   public long obterPróximaTransição(long tempoAtual){
      Iterator op = listaDeComponentes.iterator();
      long menor, esta;
      boolean primeira = true;
      Object ob;
      Operação operação;
      menor = Long.MAX_VALUE;
      próximaOperação = null;

      while(op.hasNext()){
         ob = op.next();
         if(ob instanceof Operação){
            operação = ((Operação)ob);
            esta = operação.obterPróximaTransição(tempoAtual);
            if(primeira){
               menor = esta;
               próximaOperação = operação;
            }
            else if(esta < menor){
               menor = esta;
               próximaOperação = operação;
            }
         }
         primeira = false;
      }
      return menor;
   }
```

```
    public void processarRede(long fim, long tempoAtual){
       if(próximaOperação != null){
          System.out.println("Processo: " + próximaOperação.obterID());
          próximaOperação.processarRede(fim, tempoAtual);
       }
    }


    public String toString(){
       return "\n{[Processo] descrição: " + descrição +
              ", lote: " + loteID + super.toString() + "}";
    }


    //||||||||||||||||||||||||||| HERDADO DE ENTIDADE DO BD ||||||||||||||||||

    public void obterReferências(LinkedList lista){
       LinkedList entidade;
       try{
          entidade = encontrarEntidade(lista, Class.forName("dinproc.Operação"));
          if(entidade != null){
             Object ob;
             Iterator i = entidade.iterator();
             while(i.hasNext()){
                ob = i.next();
                if(((SubRede)ob).obterIDRedeSuperior() == ID)
                   listaDeComponentes.add(ob);
             }
          }
       }
       catch(ClassNotFoundException e){
          System.out.println("Classe não encontrada");
          System.exit(0);
       }
       super.obterReferências(lista);
    }


    public static LinkedList obterEntidadeDoBD(InterfaceBD bd){
       LinkedList lista = new LinkedList();
       lista.addAll(Processo_A.obterEntidadeDoBD(bd));
       return lista;
    }

}
```

A – 19 Classe Processo_A

```
package dinproc;

import java.util.*;

/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */


public class Processo_A extends Processo {

   public Processo_A() {
   }


   public Processo_A(long id, String descrição_, long lote) {
      super(id, descrição_, lote);
   }


   public String toString(){
      return "\n{[Processo_A]" + super.toString() + "}";
   }


   //|||||||||||||||||||||||||||| HERDADO DE ENTIDADE DO BD ||||||||||||||||||

   public void obterReferências(LinkedList lista){
      super.obterReferências(lista);
   }


   public static LinkedList obterEntidadeDoBD(InterfaceBD bd){
      LinkedList lista = new LinkedList();
      Processo_A processo;
      bd.iniciarConsulta("SELECT * FROM TBL_PROCESSO INNER JOIN" +
                         " TBL_PROCESSO_A ON TBL_PROCESSO.ID = " +
                         "TBL_PROCESSO_A.ID");
      while(bd.próximoRegistro()){
         processo = new Processo_A(bd.obterLong("ID"),
                                   bd.obterString("descrição"),
                                   bd.obterLong("lote"));
         lista.add(processo);
      }
      bd.finalizarConsulta();
      return lista;
   }

}
```

A – 20 Classe Rede

```
package dinproc;

import java.util.*;

/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */

public abstract class Rede extends EntidadeDoBD {

   // Componentes da rede: Arcos, Lugares, Transições, outras redes, operações
   protected LinkedList listaDeComponentes;

   public Rede() {
   }

   public Rede(long id) {
      super(id);
      listaDeComponentes = new LinkedList();
   }


   public abstract long obterPróximaTransição(long tempoAtual);
   // Calcula o tempo que será transcorrido até a próxima transição.


   public abstract void processarRede(long fim, long tempoAtual);


   public String toString(){
      String string = "\n{[Rede] listaDeComponentes, IDs: ";
      if(listaDeComponentes != null){
         Iterator i = listaDeComponentes.iterator();
         while(i.hasNext()){
            string += ((EntidadeDoBD)i.next()).obterID() + ", ";
         }
         if(listaDeComponentes.size() == 0) string += "vazia";
      }
      else string += "vazia";
      string += super.toString() + "}";
      return string;
   }


   //||||||||||||||||||||||||||||| HERDADO DE ENTIDADE DO BD |||||||||||||||||||

   public void obterReferências(LinkedList lista){
      //sem ação
   }

}
```

A – 21 Classe Simulação

```
package dinproc;
import java.util.*;
import java.lang.Object;
import java.sql.*;

/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */
  //||||||||||||||||||||||||||||| ||||||||||||||||||||||||||||||||||||||||||||||

public class Simulação {

   // Rede que simula a planta completa:
   LinkedList listaDeLugares;
   LinkedList listaDeTransições;
   LinkedList listaDeArcos;
   LinkedList listaDeTiposDeToken;
   LinkedList listaDeEstados;
   LinkedList listaDeProcessos;
   LinkedList listaDeOperações;

   // Objetos da manufatura, presentes na planta (para esta simulação:)
   LinkedList listaDePeças;
   LinkedList listaDePallets;
   LinkedList listaDeAgentes;
   LinkedList listaDeConjuntos;

   // Propriedades internas
   protected LinkedList listas;
   protected InterfaceBD interfaceBD;
   protected Processo próximoProcesso;
   protected long tempoAtual;
   protected long INÍCIO = 0;

   public Simulação() {
   }

   public Simulação(InterfaceBD bd) {
      interfaceBD = bd;
      obterComponentesDasRedes();
      obterEstadoDaPlanta();
      prepararDados();
      obterReferências();
   }


   public String toString(){
      return "Simulação";
   }

   public void simular(){
      simular(Long.MAX_VALUE);
   }

   public void simular(long tempoFinal){
      long tempoPróximo = INÍCIO;
```

```
    boolean háPróximo = true;
    tempoAtual = INÍCIO;

    imprimirEstado();

    while(háPróximo){
       tempoPróximo = calcularPróximaTransição();
       if(tempoPróximo >= tempoFinal)
          return;
       System.out.println("Tempo do Próximo Processo: "+ tempoPróximo + "\n");
       if(tempoPróximo != Long.MAX_VALUE){
          if(próximoProcesso != null){
             próximoProcesso.processarRede(tempoPróximo, tempoAtual);
             imprimirEstado();
             tempoAtual = tempoPróximo;
          }
       }
       else
          háPróximo = false;
    }
 }


 private long calcularPróximaTransição(){
    // Tempo e Processo da próxima transição
    // Calcula o tempo para todos os processos
    Iterator op = listaDeProcessos.iterator();
    long menor, esta;
    boolean primeira = true;
    Object ob;
    Processo processo;
    menor = Long.MAX_VALUE;
    próximoProcesso = null;

    while(op.hasNext()){
       ob = op.next();
       if(ob instanceof Processo){
          processo = ((Processo)ob);
          esta = processo.obterPróximaTransição(tempoAtual);
          if(primeira){
             menor = esta;
             próximoProcesso = processo;
          }
          else if(esta < menor){
             menor = esta;
             próximoProcesso = processo;
          }
       }
       primeira = false;
    }
    return menor;
 }


 private void obterEstadoDaPlanta(){
    listaDePeças = Peça.obterEntidadeDoBD(interfaceBD);
    listaDePallets = Pallet.obterEntidadeDoBD(interfaceBD);
    listaDeAgentes = Agente.obterEntidadeDoBD(interfaceBD);
    listaDeConjuntos = Conjunto.obterEntidadeDoBD(interfaceBD);
 }

 private void obterComponentesDasRedes(){
    listaDeEstados      = Estado.obterEntidadeDoBD(interfaceBD);
    listaDeProcessos    = Processo.obterEntidadeDoBD(interfaceBD);
    listaDeLugares      = Lugar.obterEntidadeDoBD(interfaceBD);
```

```
        listaDeTransições    = Transição.obterEntidadeDoBD(interfaceBD);
        listaDeArcos         = Arco.obterEntidadeDoBD(interfaceBD);
        listaDeTiposDeToken = TipoDeToken.obterEntidadeDoBD(interfaceBD);
        listaDeOperações     = Operação.obterEntidadeDoBD(interfaceBD);
    }

    private void prepararDados(){
        listas = new LinkedList();
        listas.add(listaDePeças);
        listas.add(listaDePallets);
        listas.add(listaDeAgentes);
        listas.add(listaDeConjuntos);
        listas.add(listaDeEstados);
        listas.add(listaDeProcessos);
        listas.add(listaDeLugares);
        listas.add(listaDeTransições);
        listas.add(listaDeArcos);
        listas.add(listaDeTiposDeToken);
        listas.add(listaDeOperações);
    }

    private void obterReferências(){
        Iterator is = listas.iterator();
        Iterator i;
        while(is.hasNext()){
            i = ((LinkedList)is.next()).iterator();
            while(i.hasNext()){
                ((EntidadeDoBD)i.next()).obterReferências(listas);
            }
        }
    }

    public void imprimirListas(){
        Iterator is = listas.iterator();
        Iterator i;
        while(is.hasNext()){
            i = ((LinkedList)is.next()).iterator();
            while(i.hasNext()){
                System.out.println(i.next().toString());
            }
        }
    }

    private void imprimirEstado(){
        Iterator i = listaDeLugares.iterator();
        Object ob;
        while(i.hasNext()){
            ob = i.next();
            if(ob instanceof Lugar){
                System.out.println("(" + ((Lugar)ob).obterID() + ")\t ["
                    + ((Lugar)ob).obterEstadoAssociado().obterEstado()
                    + "]\tC: "
                    + ((Lugar)ob).qualCapacidade() + "\t\t-> {"
                    + ((Lugar)ob).obterStringListaDeTokens()
                    + "}"

                );
            }
        }
    }

}
```

A – 22 Classe Tendência

```
package dinproc;

import java.util.*;

import javax.swing.JOptionPane;


/**
 * Title:        DinProc - Simulador de dinamica de processos no chao de fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */

public class Tendência {

   public Tendência() {
   }


////////////////////////////////  MAIN para teste de tendência //////////


   public static void main(String[] args) {
      Tendência tendência = new Tendência();

      // Cria o objeto de interface com o banco de dados
      // Nome para acesso no banco de dados: PLANTA
      InterfaceBD bd = new InterfaceBD("PLANTA");

      // Cria o objeto de simulação a partir dos dados do BD
      Simulação simul = new Simulação(bd);

       // Simula a planta até um não haverem mais transições possíveis
      simul.simular();

      System.out.println("-----------------------------------------\n");

      // O objeto de simulação já imprime automaticamente os estados futuros
      // a cada mudança de estados ocorrida, assim como o tempo no qual
      // ocorreu cada transição.

      System.exit(0);  //Fim do programa
   }
}
```

A – 23 Classe TipoDeToken

```
package dinproc;

import java.util.*;


/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */


public class TipoDeToken extends EntidadeDoBD {


   public String descrição;


   public TipoDeToken() {
   }


   public TipoDeToken(String tipoVálido) {
      super(0);
      descrição = tipoVálido;
   }


   public boolean equals(Object ob){

      if(ob instanceof TipoDeToken){
         if(((TipoDeToken)ob).descrição.equals(descrição)){
            return true;
         }
      }
      return false;
   }


   public TipoDeToken(long id, String tipoVálido){
      super(id);
      descrição = tipoVálido;
   }



   public String toString(){
      return "\n{[TipoDeToken] descrição: " + descrição +
             super.toString() + "}";
   }
```

```
    //||||||||||||||||||||||||||||| HERDADO DE ENTIDADE DO BD ||||||||||||||||||

    public void obterReferências(LinkedList lista){
       //sem referências no BD
    }



    public static LinkedList obterEntidadeDoBD(InterfaceBD bd){
       LinkedList lista = new LinkedList();
       TipoDeToken tipo;
       bd.iniciarConsulta("SELECT * FROM TBL_TIPO_TOKEN");
       while(bd.próximoRegistro()){
          tipo = new TipoDeToken(bd.obterLong("ID_TipoDeToken"),
                                 bd.obterString("descrição"));
          lista.add(tipo);
       }
       bd.finalizarConsulta();
       return lista;
    }

}
```

A – 24 Interface Token

```
package dinproc;

import java.util.*;



/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */


public interface Token {

   public LinkedList obterSubTokens();

   public long obterSuperToken();

   public LinkedList obterSubTokensDoTipo(TipoDeToken tipo);

   public boolean tokenDoTipo(TipoDeToken tipo);

   public void novoEstado(Estado estado_, Object resp);

   public Estado estadoAtual();

   public void destruir();

}
```

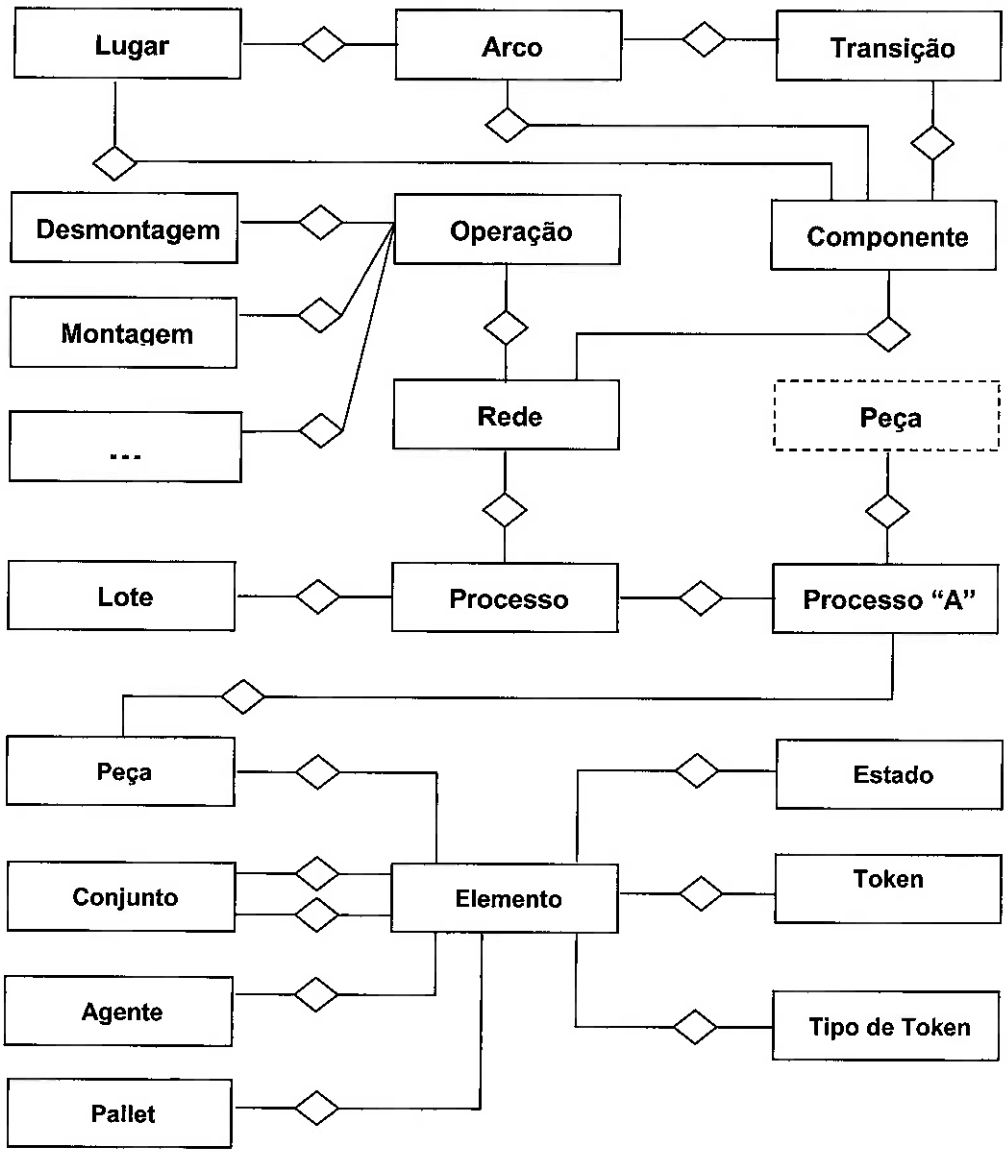
A – 25 Classe Transição

```
package dinproc;

import java.util.*;

/**
 * Title:        DinProc -Simulador da dinamica dos processos no chao-de-fabrica
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Escola Politecnica - USP
 * @author IGOR NEIVA CAMARGO
 * @version 1.0
 */


public class Transição extends Componente {

   private LinkedList listaDeArcos;


   public Transição() {
   }


   public Transição(long id, long rede, int posição) {
      super(id, rede, posição);
      listaDeArcos = new LinkedList();
   }


   public boolean verificarCondiçãoParaTransição(){
      Iterator i = listaDeArcos.iterator();
      boolean condiçãoOK = true;
      Arco arco;
      while(i.hasNext() && condiçãoOK == true){
         arco = (Arco)i.next();
         condiçãoOK = arco.aceitaTransmitirTokens();
      }
      return condiçãoOK;
   }


   public String toString(){
      String string = "\n{[Transiçao] lista de arcos (IDs): ";
      if(listaDeArcos != null){
         Iterator i = listaDeArcos.iterator();
         while(i.hasNext()){
            string += ((EntidadeDoBD)i.next()).obterID() + ", ";
         }
         if(listaDeArcos.size() == 0) string += "vazia";
      }
      else string += "vazia";
      string += super.toString() + "}";
      return string;
   }
```

```
//|||||||||||||||||||||||||||| IMPLEMENTADO DE OBJETO DE REDE ||||||||||||||

public String obterDescrição(){
      return  "(Transição ID: " + ID + ", posição na rede: " + posiçãoNaRede
              + ")";
}



//|||||||||||||||||||||||||||| HERDADO DE ENTIDADE DO BD ||||||||||||||||||

public void obterReferências(LinkedList lista){
   LinkedList entidade;
   try{
      entidade = encontrarEntidade(lista, Class.forName("dinproc.Arco"));
      if(entidade != null){
         Object ob;
         Iterator i = entidade.iterator();
         while(i.hasNext()){
            ob = i.next();
            if(((Arco)ob).transiçãoID == ID)
               listaDeArcos.add(ob);
         }
      }
   }
   catch(ClassNotFoundException e){
      System.out.println("Classe não encontrada");
      System.exit(0);
   }
   super.obterReferências(lista);
}



public static LinkedList obterEntidadeDoBD(InterfaceBD bd){
   LinkedList lista = new LinkedList();
   Transição transição;
   bd.iniciarConsulta("SELECT * FROM TBL_COMPONENTE INNER JOIN" +
                      " TBL_TRANSICAO ON TBL_COMPONENTE.ID = " +
                      "TBL_TRANSICAO.ID");
   while(bd.próximoRegistro()){
      transição = new Transição(bd.obterLong("ID"),
                        bd.obterLong("rede"),
                        (int)bd.obterLong("posiçãoNaRede"));
      lista.add(transição);
   }
   bd.finalizarConsulta();
   return lista;
}
}
```

Anexo B

Este anexo contém a estrutura das tabelas do repositório de dados. O acesso ao banco de dados foi implementado utilizando-se JDBC genérico. Portanto, qualquer banco de dados que suporte Java pode ser utilizado como fonte de dados. A figura a seguir mostra os relacionamentos existentes entre as tabelas.



TBL_ELEMENTO

Campo	Descrição / correspondência
ID	inteiro, chave primária, identificador do elemento
lugar	inteiro, referência ao lugar ocupado por este token
estadoDoToken	inteiro, referência a estado deste token
superToken	inteiro, referência ao token que contém este token
tipoDoToken	inteiro, referência ao tipo de token deste token

TBL_AGENTE

Campo	Descrição / correspondência
ID	inteiro, chave primária, identificador do agente – o mesmo existente para o elemento que define este agente
descrição	texto, descrição do agente

TBL_PEÇA

Campo	Descrição / correspondência
ID	inteiro, chave primária, identificador da peça– o mesmo existente para o elemento que define esta peça
processo	inteiro, referência ao processo para fabricação desta peça

TBL_PALLET

Campo	Descrição / correspondência
ID	inteiro, chave primária, identificador do pallet – o mesmo existente para o elemento que define este pallet
capacidade	inteiro, capacidade do pallet, em número de peças

TBL_CONJUNTO

Campo	Descrição / correspondência
ID	inteiro, chave primária, identificador do conjunto – o mesmo existente para o elemento que define este conjunto

TBL_COMPONENTE

Campo	Descrição / correspondência
ID	inteiro, chave primária, identificador do componente
rede	inteiro, referência à rede que contém este componente
posiçãoNaRede	inteiro, posição deste componente na rede

TBL_ARCO

Campo	Descrição / correspondência
ID	inteiro, chave primária, identificador do arco – o mesmo existente para o componente que define este arco
transição	inteiro, referência à transição relacionada a este arco
lugar	inteiro, referência à transição relacionada a este arco
pesoDoArco	inteiro, peso do arco
sentido	booleano, sentido do arco: transição ou lugar como orígem do arco
tipoDeToken	inteiro, referência ao tipo de token transportado pelo arco

TBL_TRANSIÇÃO

Campo	Descrição / correspondência
ID	inteiro, chave primária, identificador da transição – o mesmo existente para o componente que define esta transição

TBL_LUGAR

Campo	Descrição / correspondência
ID	inteiro, chave primária, identificador do lugar – o mesmo existente para o componente que define este lugar
capacidade	inteiro, capacidade do lugar, em número de tokens
estado	inteiro, referência ao estado associado a este lugar
tipoDeToken	inteiro, referência ao tipo de token armazenado pelo lugar

TBL_PROCESSO

Campo	Descrição / correspondência
ID	inteiro, chave primária, identificador do processo
lote	inteiro, referência ao lote associado a este processo
descrição	texto, descrição do processo

TBL_PROCESSO_A

Campo	Descrição / correspondência
ID	inteiro, chave primária, identificador de um processo do tipo "A" – o mesmo existente para o registro do processo relacionado

TBL_TIPO_DE_TOKEN

Campo	Descrição / correspondência
ID	inteiro, chave primária, identificador do tipo de token
descrição	texto, descrição do tipo de token

TBL_ESTADO

Campo	Descrição / correspondência
ID	inteiro, chave primária, identificador do estado
descrição	texto, descrição do estado

TBL_OPERAÇÃO

Campo	Descrição / correspondência
ID	inteiro, chave primária, identificador da Operação
descrição	texto, descrição da operação realizada
posiçãoNaRede	inteiro, posição ocupara pela operação na rede superior
tempoInício	inteiro, tempo quando iniciou a operação, se iniciou
tempoDaOperação	inteiro, duração da operação completa
rede	inteiro, referência à rede superior desta operação

TBL_OP_DESMONTAGEM

Campo	Descrição / correspondência
ID	inteiro, chave primária, identificador da operação – o mesmo existente para o registro da operação relacionada

TBL_OP_MONTAGEM

Campo	Descrição / correspondência
ID	inteiro, chave primária, identificador da operação – o mesmo existente para o registro da operação relacionada

TBL_OP_PROCESSAMENTO

Campo	Descrição / correspondência
ID	inteiro, chave primária, identificador da operação – o mesmo existente para o registro da operação relacionada

TBL_METROLOGIA

Campo	Descrição / correspondência
ID	inteiro, chave primária, identificador da operação – o mesmo existente para o registro da operação relacionada
pRefugo	ponto flutuante, probabilidade do resultado da metrologia ser "refugo", para o processo o qual comporta esta operação.
pRetrabalho	ponto flutuante, probabilidade do resultado da metrologia ser "re-trabalho", para o processo o qual comporta esta operação.

Referências Bibliográficas

[1] VERNADAT, F. **Enterprise Modeling and Integration, Principles and Applications**. Chapman & Hall, Primeira Edição, 1996.

[2] BERIO, G.; LEVA, A.; GIOLITO, P.; VERNADAT, F. **Object-oriented process development in the M*-Object methodology**. Journal of Intelligent Manufacturing (2000) 11, 113-125.

[3] BERIO, G.; LEVA, A.; GIOLITO, P.; VERNADAT, F. **Process and Data Nets: The Conceptual Model of the M*-Object Methodology**. IEEE Trans. on System, Man, and Cybernetics, Vol. 29, NO. 1 Feb. 1999.

[4] DATE, C. J. **Introdução a sistemas de bancos de dados**. 4.ed. Editora Campus, 1991.

[5] KAMINSKI, P. C. **Desenvolvendo produtos, planejamento, criatividade e qualidade**. São Paulo: LTC, 2000.

[6] HORIKAWA, O. **Fabricação auxiliada por computador (CAM).** (Notas de Aula) Escola Politécnica, USP, 2001.

[7] PEREIRA, A. L.; LIMA, T. B. **Informatização e Gerenciamento da Fabricação Mecânica usando a sistemografia. Estudo de caso: fabricação de engrenagens**. Trabalho de formatura apresentado à Escola Politécnica da Universidade de São Paulo, 2001.

[8] BERIO, G.; LEVA, A.; GIOLITO, P.; VERNADAT, F. **The M*-Object Methodology for Information System Design in CIM Enviroments**. IEEE Trans. on System, Man, and Cybernetics, Vol. 25, NO. 1 Feb. 1995.

[9] VERNADAT, F. **Manufacturing Systems Modelling, Specification and Analysis**. IFIP WG5.7 Working Conference on Evaluation of Production Management Methods, Porto Alegre, Brasil, Março 21-24, 1994.

[10] SLACK, N.; CHAMBERS, S.; HARLARD, C.; HARRISON, A.; JOHNSTON, R. **Administração da Produção.** Editora Atlas, Edição Compacta, 1999.