Faculdade de Engenharia da Universidade do Porto
Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Ano Lectivo 2008/2009

Sistemas de Informação Industriais

Aplicação de Controlo de uma Célula Flexível de Fabrico

Relatório do Projecto

Alunos:

010503034 Filipe David Maia Ferreira

950503230 Vasco Alexandre Martins Neves Fernandes

Índice

1. Abordagem do Problema

O projecto consistiu no desenvolvimento de uma aplicação de controlo para o "kit staudinger", onde o utilizador pode seleccionar o número e tipo de peças que quer que a linha produza e entregue no armazém. Como só existe um kit, foi criado um simulador, numa versão simplificada da realidade, que será o alvo dos testes desenvolvidos ao longo do projecto.

Numa primeira abordagem ao problema foi feita uma divisão deste, numa perspectiva de "dividir para reinar". Foram pensadas as seguintes classes:

Classes Base:

- "Tapete de Interacção com Armazém";
- "Tapete Linear";
- "Tapete Rotativo";
- "Maquina".

Classes de nível II:

- "Plate de Interacção com Armazém";
- "Conjunto de Tapetes Superiores da Estação Série";
- "Conjunto de Tapetes inferiores Estação Série";
- "Conjunto de Tapetes superiores Estação Paralela";
- "Conjunto de Tapetes inferiores Estação Paralela";
- "Estação de Trabalho Série";
- "Estação de Trabalho Paralela".

Classe principal:

- "ShoopFloor".

Quanto às classes base, é mais ou menos óbvia a sua razão, uma vez que são os constituintes 'atómicos' do sistema.

Em relação às classes de nível II, a divisão do sistema de transporte em conjuntos de tapetes permite encapsular vários tapetes lineares e rotativos fornecendo os serviços de recepção e envio das peças, simplificando assim o controlo ao nível superior. Além disso, com esta divisão, será possível, no futuro, acrescentar módulos do tipo "Plate Ms", "Plate Mp" ou "Plate A", sem nenhum inconveniente, uma vez que as classes foram pensadas tendo em vista essa modularidade.

As classes "estação serie" e "estação paralela" englobam o conjunto de máquinas e tapetes que as constituem e têm em vista o encapsulamento dos serviços de recepção, produção e posterior entrega de peças ao sistema de transporte.

A classe "Plate de interacção com armazém" pretende englobar os dois tapetes de interacção, fornecendo os serviços de retirada e inserção de peças de determinado tipo no armazém.

Nesta fase convém deixar claro que, tendo em vista a exequibilidade do controlador, no ambiente académico e nos prazos estipulados, algumas simplificações foram introduzidas, nomeadamente a retirada de peças do armazém apenas no tapete superior e inserção no inferior (sendo aliás os serviços que o próprio simulador disponibiliza); a entrada de peças nas estações de trabalho é, consequentemente, feita apenas do lado superior e saída do lado inferior. Para além disso, nesta primeira abordagem, não foram consideradas optimizações ao nível do escalonamento.

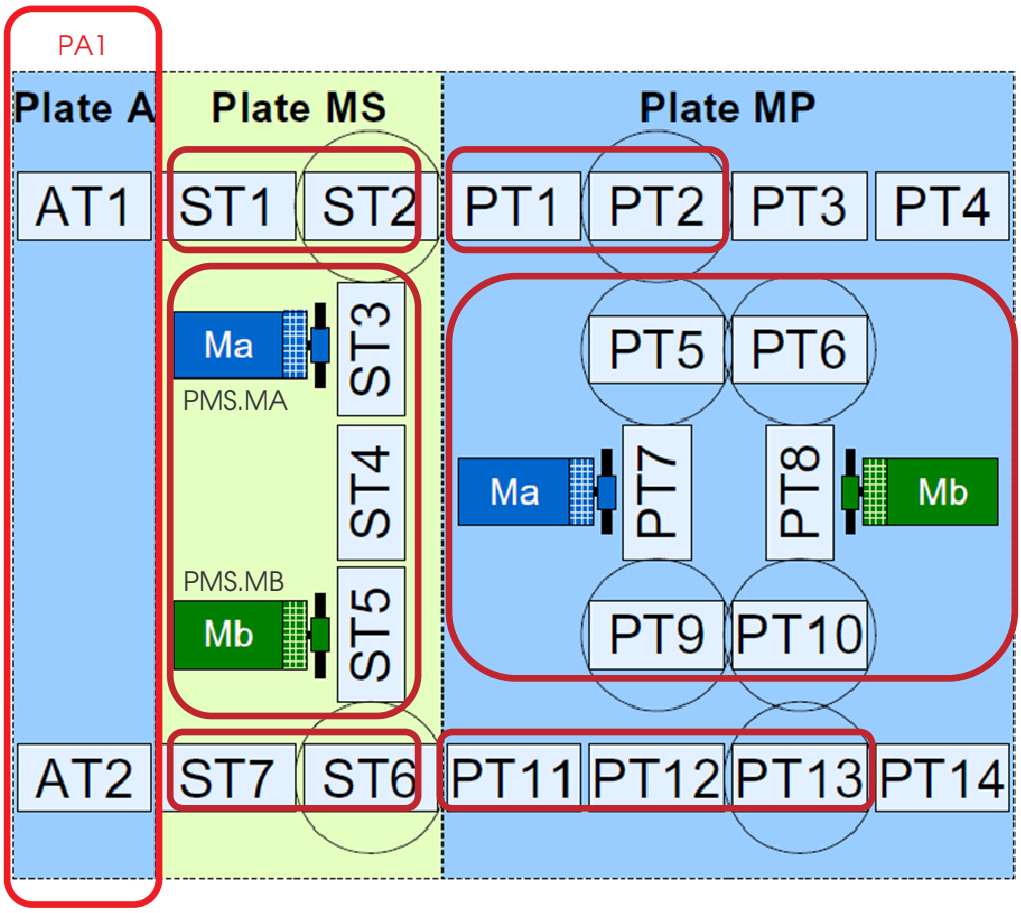


Figura 1 - Divisão topológica do ShopFloor

2. Modelo "UML" do sistema

Partindo da abordagem inicial do problema, e recorrendo à linguagem de modelação "UML", apresentada nas aulas teóricas, foram produzidos o diagrama de classes e alguns diagramas de sequência baseados em casos de uso, tendo-se revelado, estes últimos, muito úteis na especificação dos serviços que seriam necessários associar às classes de forma a tornar possível a implementação do sistema de controlo.

A figura seguinte mostra o diagrama de classes "UML".

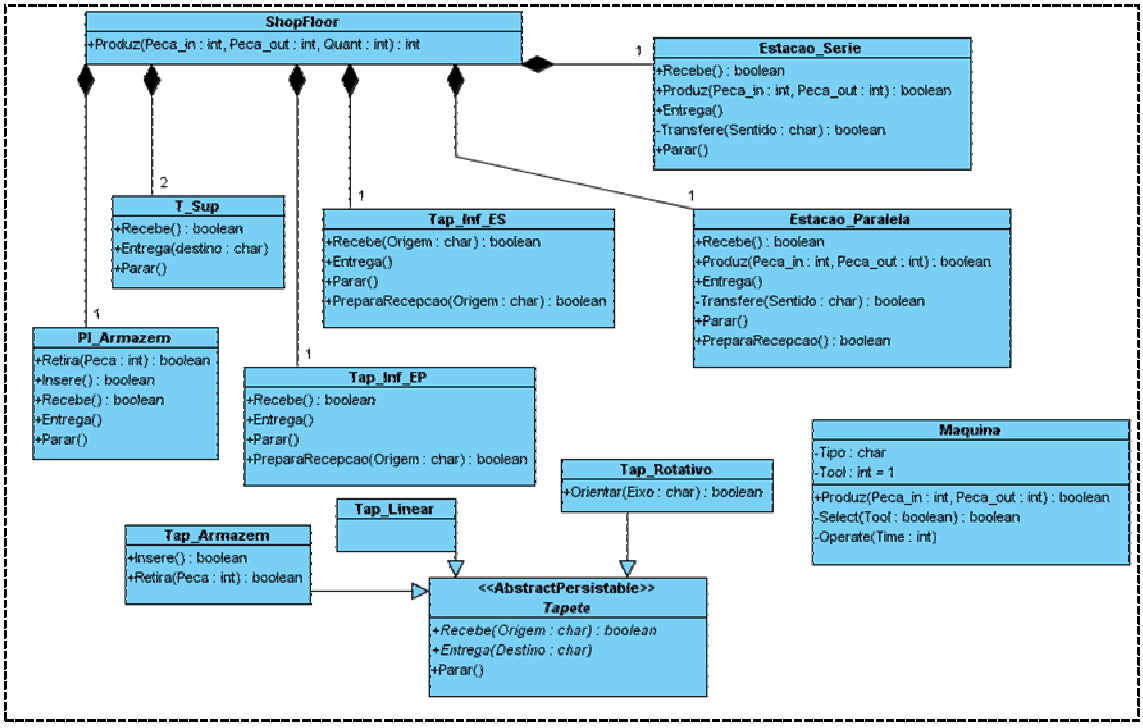


Figura 2 - Diagrama de classes Inicial

Numa discussão intermédia, fomos alertados pelo docente da complexidade que a implementação das classes de mais baixo nível poderia introduzir, e foram levantadas dúvidas quanto aos benefícios reais da sua programação. De facto, do ponto de vista de uma possível optimização, considerar os elementos de forma atómica parecia uma estratégia aconselhável, mas pensando que os tempos de processamento das estações de trabalho serão os mais limitadores de todo o processo, não adiantará muito que as classes de transporte estejam muito optimizadas, quando as estações de trabalho vão acabar por impor tempos de espera a estas.

Para alguns casos de uso, nomeadamente, produção na estação de trabalho série e maquinação P1->P6 na estação série, foram elaborados os diagramas

de sequência, de forma a auxiliar a validação dos serviços associados às classes. As figuras seguintes mostram os diagramas de sequência para os dois casos de uso.

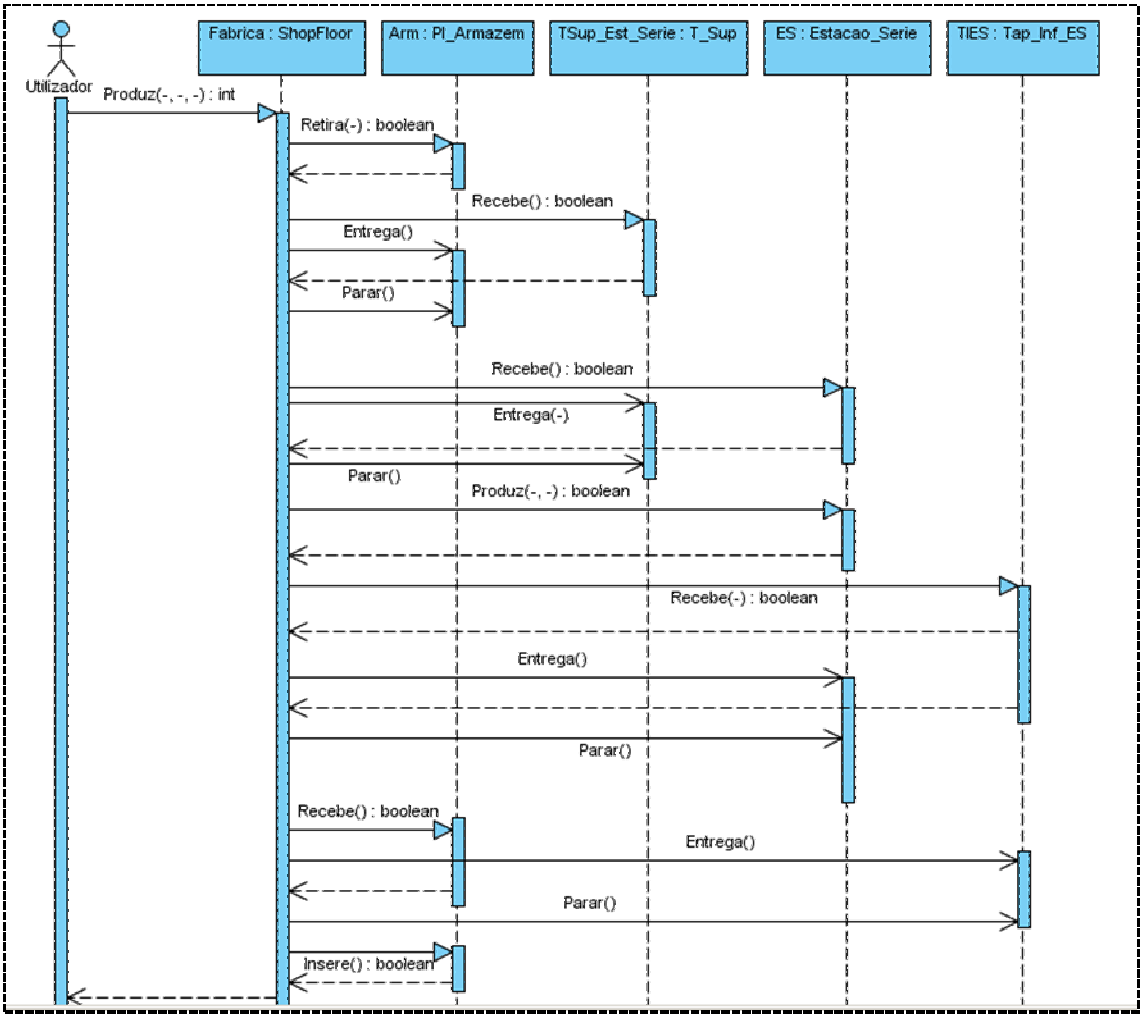


Figura 3- Diagrama de sequência para o caso "produção na estação série"

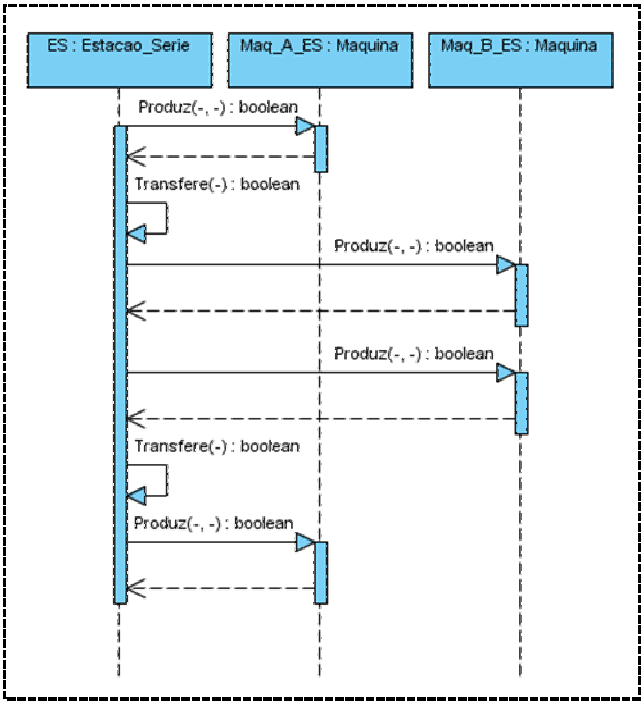


Figura 4 - Diagrama de sequência para o caso "Maquinação P1->P6 na estação série"

3. Implementação do sistema

A implementação do sistema de controlo foi feita usando uma IDE que implementa as linguagens da norma IEC 61131-3, o ICS Triplex IsaGRAF.

Transportar um modelo em UML para essas linguagens acarreta algumas particularidades. Não é propriamente uma linguagem orientada a objectos, apesar das possibilidades conferidas pelo conceito dos *Function Block Types* e *Function Block Instances*. Não existe o conceito de métodos de forma explícita, facto que acaba por se ultrapassar imaginando uma entrada de um Function Block (que implemente uma classe) para designar o método que pretendemos evocar. Os retornos, caso aplicável (ou seja, perante chamadas síncronas), são conseguidos através de outputs dos Function Blocks, e o bloqueio do processamento da classe que faz a chamada, através da sua própria máquina de estados, que numa etapa invoca o Function Block e na transição seguinte testa a saída respectiva.

3.1. Implementação das Máquinas de Ferramenta

A classe Máquina "MAQ" não contempla os tapetes respectivos, sendo que se foca na própria máquina em si (troca de ferramenta, rotação da ferramenta e tempo de rotação).

O serviço prestado pela classe consiste em produzir uma peça de um determinado tipo, passando como parâmetro os tipos de peças de entrada e de saída. Daí a necessidade de definir as variáveis de entrada **(INT)PEÇA_IN**, **(INT)PEÇA_OUT** e **(STRING)OPERAÇÃO**, sendo que esta ultima dá a ordem de produção à maquina, quando toma o valor **'PRD'**. Para assinalar o fim de produção, foi criada uma variável de saída **"FIM_PRD",** que vai a 1 sempre que se acaba de produzir a peça pedida. Existe um segundo método assíncrono, para confirmação, apenas para assegurar que a máquina fique pronta para nova maquinação por ordem da estação onde está inserida, e que não execute nova maquinação intempestivamente.

Como existem dois tipos de máquinas, existia a hipótese de criar dois *Function Block Type diferentes*, um para a máquina tipo A e outro para a máquina tipo B. Contudo, apenas foi criado um e acrescentada uma entrada **"Tipo"** à interface do *Function Block*, de forma a poder definir o tipo de máquina que se pretende.

De forma a memorizar a ferramenta montada, foi criada uma variável local denominada de **"TOOL"** e uma outra para definir a ferramenta que se pretende montar **"SET_TOOL"** (tool setpoint).

O controlo das máquinas está dividido em dois ramos, um que controla a escolha da ferramenta, trata-se de um controlo cíclico que testa continuamente se a "tool" montada é igual ao setpoint (set_tool). O outro ramo controla a maquinação das peças conforme o estado das variáveis "peça_in" e "peça_out" e os tempos necessários em cada combinação. Existem 3 combinações de maquinações possíveis para cada tipo de máquina, daí os 3 sub-ramos.

De seguida é apresentado o diagrama de estados da classe máquina, note-se que apenas está desenhado o ramo de controlo referente ao tipo de máquina A, sendo que o ramo para o tipo B é exactamente o mesmo, alterando apenas a combinação de peças produzidas, os tempos e ferramentas envolvidas.

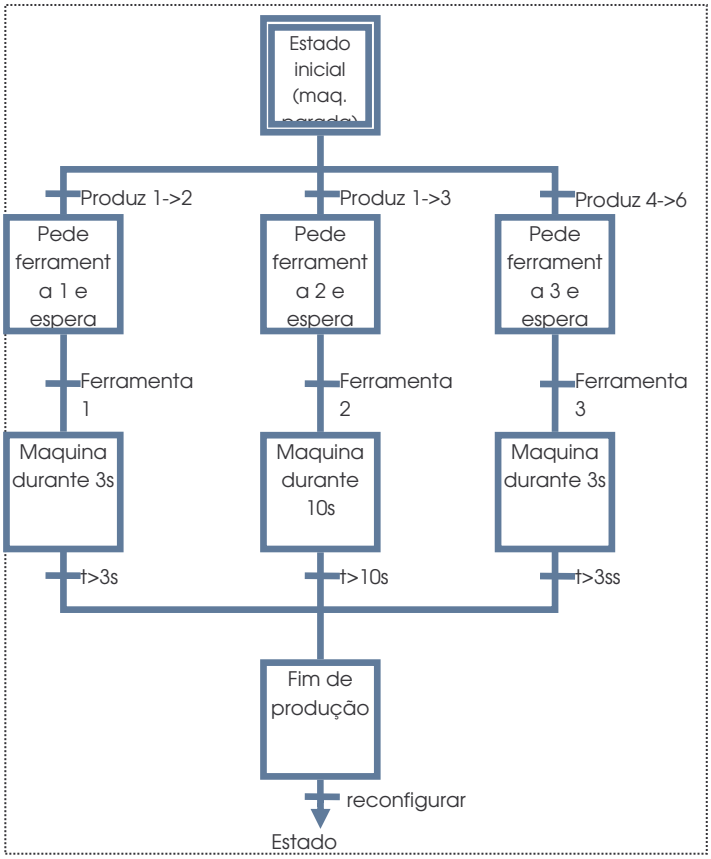


Figura 5 - Diagrama de estados para o controlo das maquinações

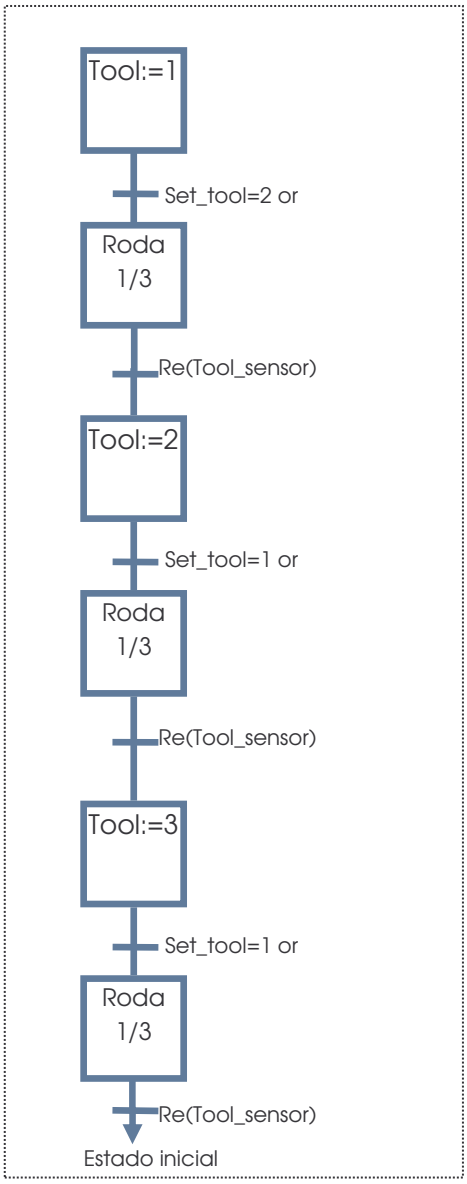


Figura 6 - Diagrama de estados para controlo da ferramenta

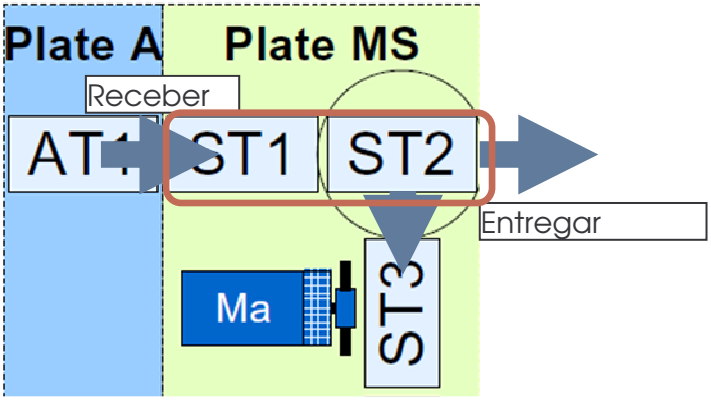
3.2. Lógica de Implementação das classes ao nível do transporte

Durante a implementação do sistema, concluiu-se que os serviços das classes base "tapete linear" e "tapete rotativo" se limitavam a fazer actuar uma ou no máximo duas saídas, pelo que se optou por não implementar estas classes, sendo que as classes de nível superior operam directamente as entradas e saídas dos seus componentes.

Considera-se nível de transporte o conjunto de classes que englobam os tapetes superiores e inferiores de ambas as "Plates"("TAP_SUP_S"; "TAP_INF_S"; "TAP_SUP_P" e "TAP_INF_P") . Todas estas classes seguem um algoritmo de

controlo similar, que consiste em preparar o sistema para a recepção (nos casos em que a recepção seja feita em tapete rotativo), receber a peça, preparar o envio e finalmente enviar até que seja feito um pedido de paragem. Devido à existência de apenas um sensor de presença de peça no centro do tapete, uma dada instância do sistema de transporte não tem consciência de quando finalizou o envio de uma peça, sendo que o programa principal "ShoopFloor" terá que enviar o comando de paragem a essa instância.

De seguida é apresentada um pouco em detalhe a implementação de uma das classes de transporte (tapetes superiores da estação serie) - **"TAP_SUP_SERIE".**



Neste caso, a classe recebe sempre a peça de "OESTE", podendo entregá-la a "ESTE" ou a "SUL". Para receber a peça, não é necessária uma preparação prévia, contudo para o envio da peça, o tapete rotativo alinha-se sempre com o primeiro, recebe a peça, e depois é tomada a decisão de este se reorientar ou não consoante o destino pretendido. Os serviços fornecidos por esta classe serão: recebe() que retorna o sinal booleano "fim_rec" ; envia(destino) e, por fim, 'stop', para poder parar os motores quando a estação seguinte detecta a chegada da peça.

As seguintes variáveis foram criadas para este Function Block:

VAR_INPUT
Operacao: string(3) {'REC';'ENV';'STP'};
Destino: string(1){'E';'S'};
END_VAR;

VAR_OUTPUT
FIM_REC:BOOL
END_VAR;

A figura seguinte mostra o diagrama de estados pensado para esta classe.

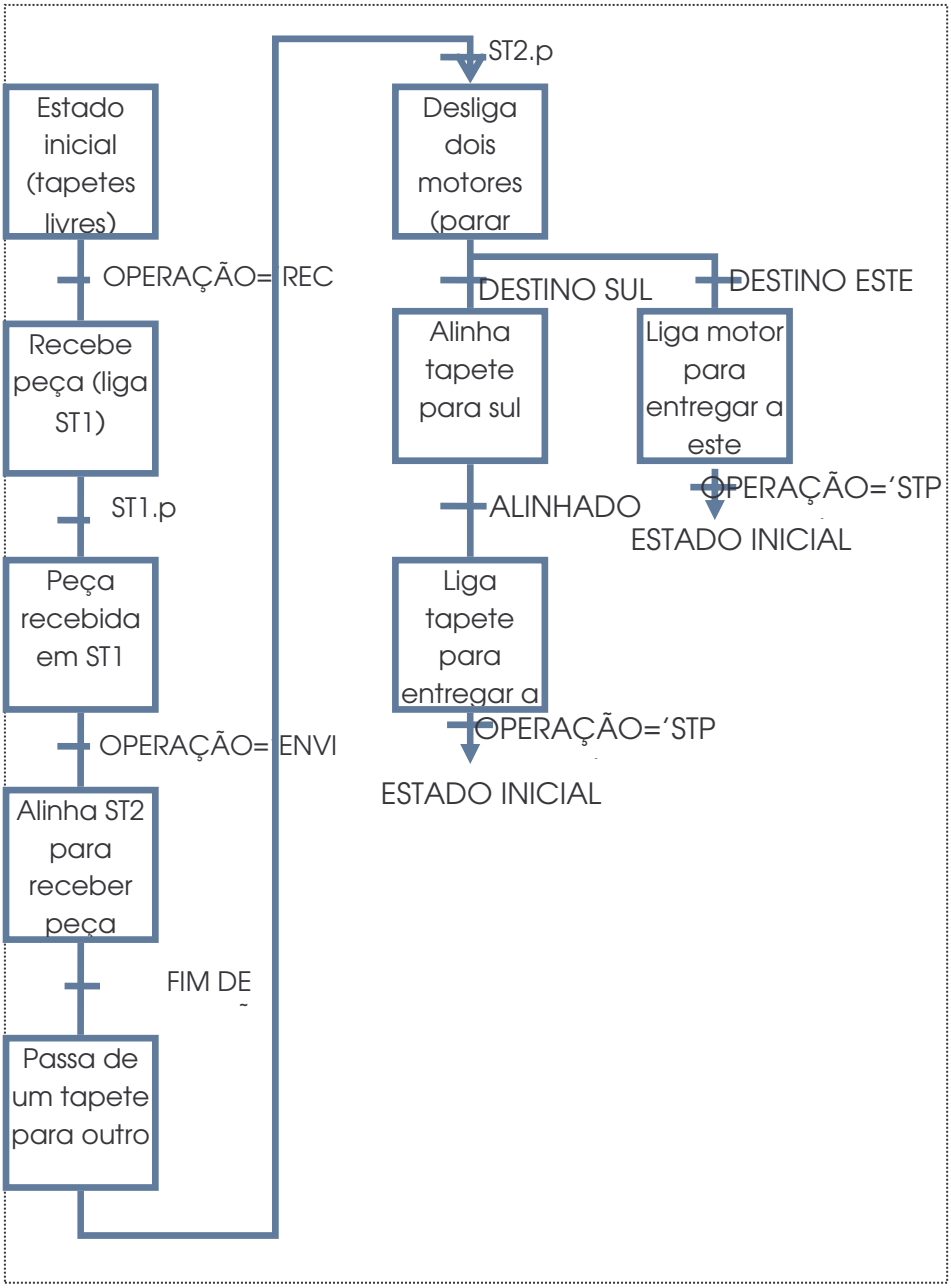


Figura 7 - Diagrama de estados da classe Tap_Sup_Serie

As restantes três classes do nível do sistema de transporte seguem, com algumas variações, o mesmo modelo de funcionamento e possuem os mesmos serviços, pelo que não são aqui apresentadas, contudo podem ser consultadas no código da aplicação que pode ser encontrado em http://paginas.fe.up.pt/~ee01034/files/Projecto_SIIN_v7.rar (projecto Isagraf).

Salienta-se, desde já, o facto de as classes que modelam as estações de trabalho também disponibilizarem os serviços de recepção (a paralela de preparação, pois recebe num tapete rotativo), de envio e de paragem, seguindo a mesma lógica aqui descrita para as classes de apenas transporte.

3.3. Lógica de produção de Peças e considerações sobre implementação das classes "ESTAÇÃO SÉRIE" E "ESTAÇÃO PARALELA"

O diagrama da figura 8 esclarece a lógica que impera nas estações de trabalho, no que diz respeito ao serviço que estas proporcionam à classe superior: Produz (P_in, P_out).

De facto, existem 4 peças possíveis como matérias-primas {P1,P2,P3,P4}, e 5 peças possíveis como produto final {P2,P3,P4,P5,P6}. Apesar disso, as hipóteses de maquinações para obter um determinado produto final a partir de um outro produto inicial reduzem-se às representadas no diagrama abaixo, pelo que foi suficiente implementar uma estrutura de decisão que contemplasse aquelas hipóteses.

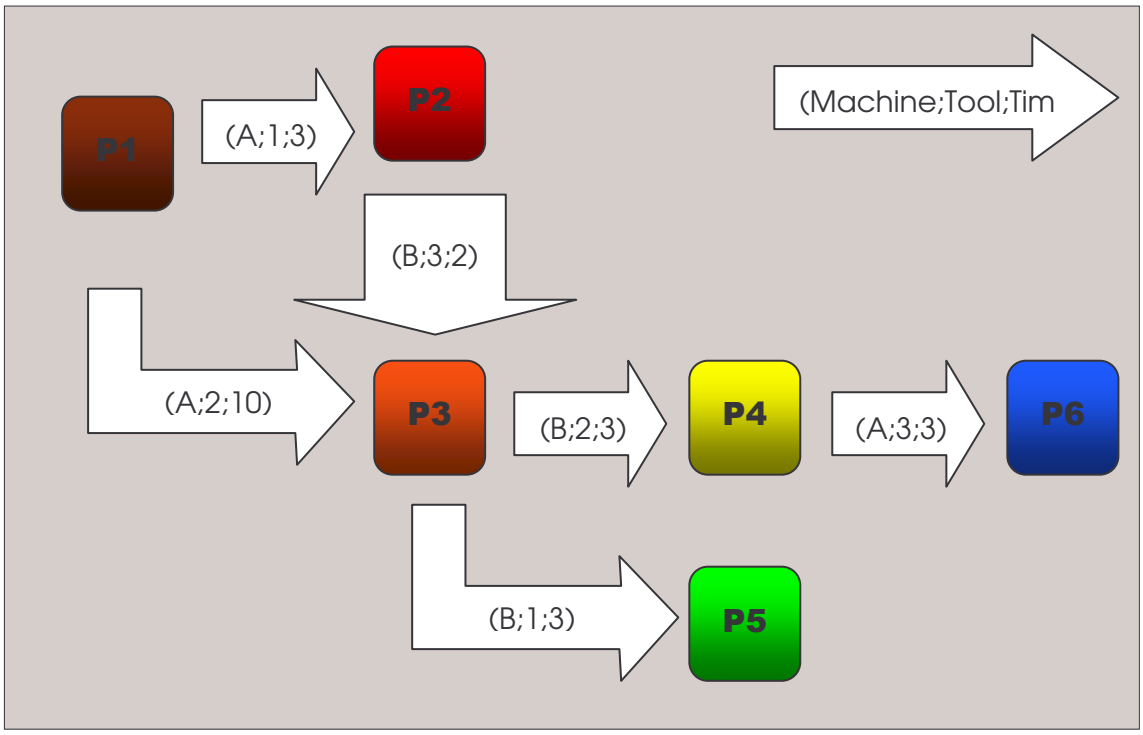


Figura 8 - Diagrama de transformação das peças

As estações recorrem assim às suas instâncias da classe "MAQ", bem como a métodos privados para transferências das peças entre máquinas. No caso da estação série, os métodos não foram propriamente implementados, pois dependem apenas de uma etapa no SFC (ligar 3 tapetes), e uma transição (sensor do tapete adequado). Já na estação paralela, uma vez que a transferência envolve alguns procedimentos mais complexos, os métodos foram criados recorrendo a SFC *childs*, cuja utilização é feita chamando numa etapa com o qualificador **S**et, e quando o SFC Child indicar o fim da sua

acção (através de uma saída booleana), invoca-se de novo com o qualificador **R**eset (essencialmente para que em futura evocação, o SFC Child comece a sua acção pela etapa inicial, e perca a memória da etapa corrente, algo característico destas POU´s entre invocações).

Essa é a razão pela qual surgem 4 SFC Childs associados ao Function Block "Estação Paralela", concretamente: ***TransfAB*** (feito pelos tapetes abaixo das máquinas), ***TransfBA*** (feito pelos tapetes acima das máquinas), e ainda ***IniciaPRDemB***, ***TerminaPRDemA***, métodos que, como implicam preparação de tapetes e transferência entre eles, se agruparam numa macro acção.

De referir apenas que estes métodos (privados da classe Estação Paralela), foram criados, tendo em vista o seu uso sincronizado, pensando numa potencial melhoria do seu funcionamento, podendo executar operações sobre duas peças, efectivamente em paralelo.

3.4. Lógica Global do Escalonamento – Rede de Petri

A classe "Shopfloor" enquadra-se num típico problema de Escalonamento de um Sistema de Manufactura. Usamos uma Rede de Petri para modelar o seu funcionamento com alguns objectivos em mente:

- Compreender os momentos em que os recursos envolvidos deviam ser requisitados, bem como quando podem ser libertados;
- Realçar os pontos-chave da aplicação, concretamente: o processo de decisão perante a ocupação ou não das estações de trabalho, e ainda o uso concorrente do recurso crítico ao qual chamamos Tapete Inferior da Estação Série (TIES);
- Usar uma ferramenta que tem um modelo matemático associado, que conhece diversas aplicações que suportam os seus métodos de análise, e que com a sua extensão temporal, poderia ser útil numa fase de testes de cenários, para um trabalho de optimização subsequente.

O transporte da Rede para a aplicação de controlo envolveu um esforço mínimo, pois o mapeamento é quase imediato. Associamos a cada recurso uma etapa no SFC do ShopFloor, cujo testemunho é devidamente consumido através de um AND lógico antecedendo uma transição, e libertado de forma análoga.

Mais tarde, em fase de testes, essa solução revelou-se inadequada para o recurso crítico referido anteriormente. À partida, seria de prever a não ocorrência de uma *race-condition*, pelo facto da aplicação não estar distribuída. Contudo, durante a fase de teste, surgiu uma sequência de produção que fez com que os dois ramos que dependem desse recurso

fossem activados. A resolução passou pela substituição da etapa por uma variável booleana, simulando um comportamento similar (passa a 0 ao sair dos estados onde deve ser consumido o recurso, e a 1 à entrada dos estados seguintes às transições onde deve ser libertado).

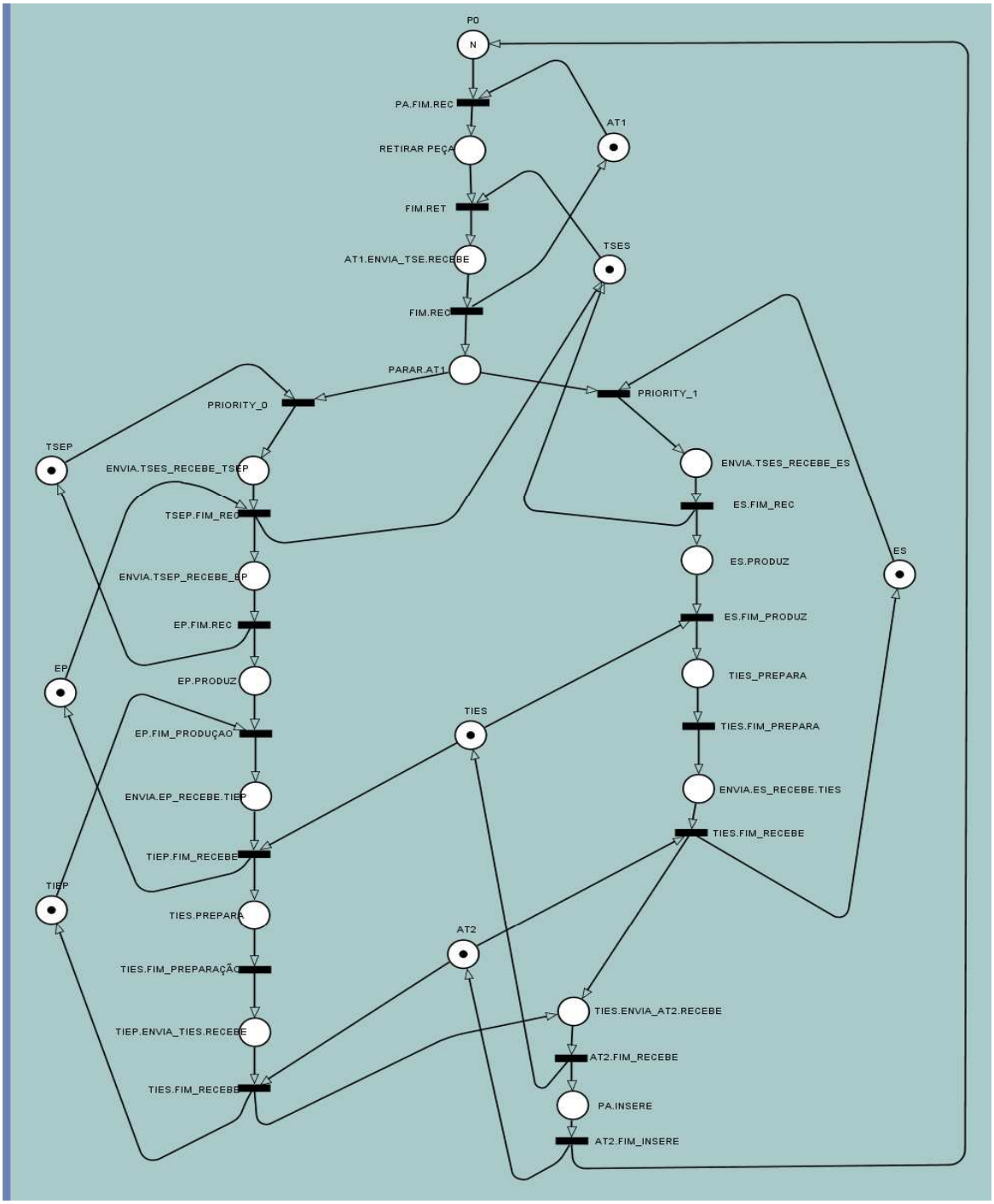


Figura 9 - Modelo do ShopFloor em Rede de Petri

4. Implementação colaborativa (versões)

Foi necessário criar uma dinâmica de edição partilhada do algoritmo de controlo, para que os elementos do grupo fossem desenvolvendo partes da aplicação em paralelo, e que a sua integração fosse simples e bem sucedida.

Desse facto, surgiram um conjunto de versões do desenvolvimento, que a seguir descrevemos:

V1 – Primeiras Interacções com o simulador. Testes dos serviços de retirada e inserção de peças no armazém;

V2 – Implementação do diagrama de sequência da Figura 3, ainda sem produção;

V3 – Implementação de um diagrama de sequência análogo ao anterior, desta vez via Estação Paralela, igualmente sem produção;

V4 – Criação do Escalonamento Global, ao nível do ShopFloor, deixando ainda de lado a produção, simulando a carga temporal desta com atrasos em substituição dos serviços Produz(p_in, p_out);

V5 – Integração da classe Máquina, Estação Série e Estação Paralela. Alteração do Shopfloor, podendo na etapa inicial definir os parâmetros Peca_in, Peca_out e N_Pecas; Shopfloor cumpre nesta fase apenas 1 pedido com sucesso;

V6 – Primeira tentativa de implementação de pedidos sequenciados. Ou seja, a partir do momento em que o shopfloor termina um ciclo de produção, ser possível desencadear um novo ciclo de produção. Tentativa instável, comportamento não conforme, possivelmente pelo uso de variáveis de Input e Output, que necessitam de estar mapeadas no módulo "I/O wiring" do IsaGRAF, e suspeitamos que acarretem consequências ao nível de atrasos na comunicação com o simulador.

V7 – Segunda tentativa de cumprir o desejado em V6, mas desta vez usando uma lista de variáveis internas do Shopfloor, adicionadas a uma Spy List, para interacção com o utilizador. Versão estável, testada com sucesso.

5. Considerações sobre Possíveis Optimizações

Ao longo da fase de modelação, bem como da implementação, foi feito, como já foi referido, um conjunto de simplificações que possibilitassem programar uma aplicação de controlo que desempenhasse o pretendido, devidamente testada e livre de funcionamentos anormais. Foram deixadas para segundo plano, um conjunto de optimizações que seria viável fazer com pequenas alterações no funcionamento de algumas das classes.

Incluindo aqui a aprendizagem extra que a própria apresentação formal com o docente possibilitou, referimos ainda algumas melhorias que podem ser feitas tendo em vista uma minimização global dos tempos de execução do serviço prestado pela classe global: Produz(Peça_in, Peça_out, N_peças).

Nesse sentido apontamos as seguintes:

- Tornar o método **Select_Tool** público

  Ao possibilitarmos à Estação de Trabalho, que tem desde logo consciência das máquinas por onde deve passar uma peça que lhe é entregue para a transformar numa outra, poderíamos fazer com que a classe MAQ providenciasse uma selecção da ferramenta, que a Estação usasse de forma prévia (enquanto as peças se dirigem para as máquinas). A desvantagem desta solução é alguma perda de encapsulamento, pois a Estação passa a ter a responsabilidade de saber que ferramentas devem ser usadas numa transformação simples (até então responsabilidade das máquinas).

- Rever o processo de decisão na escolha patente na Rede de Petri.

  Para um determinado serviço global, o ShopFloor mantém, através de dois contadores, o número de peças que retirou do armazém e o número de peças que armazenou. Usando esta informação, e usando a experiência do mau comportamento da nossa aplicação para um baixo número de peças, poderíamos conjugar a informação do número de peças que faltam para completar o serviço com a informação sobre a ocupação dos recursos.

- Melhorar a disponibilização dos serviços das Estações de Trabalho

  De facto, as estações de trabalho são as classes que claramente impõem mais limitações do ponto de vista temporal. A optimização destas poderia resultar em ganhos globais muito interessantes,

principalmente para a execução de grandes números de peças. Por isso, faria sentido elaborar mais a máquina de estados de cada uma das estações para voltar a disponibilizar alguns serviços à classe ShopFloor, que estão inibidos sem necessidade. Um exemplo disso é o da Estação Série, que quando transfere uma peça para a máquina B, poderia iniciar a recepção de nova peça, desde que a produção em curso não implique um retorno à máquina A.

6. Conclusões

Este projecto, dado o seu grau médio em termos de complexidade (pelo menos em relação a outros que nos foram colocados até agora ao longo do curso), potenciou a aprendizagem de uma metodologia para abordar problemas genéricos de engenharia de software e não só. A capacidade de abstracção, o uso de uma linguagem de modelação para descrever o funcionamento da aplicação (independente dos pormenores de implementação), e o posterior uso das linguagens da norma IEC 61131-3 foram treinadas.

Em cada item dos referidos, retiramos um conjunto de ensinamentos, dos quais destacamos:

- Na capacidade de abstracção:

  i. Perceber as vantagens da divisão do problema em sub-problemas menores, com a noção que essa subdivisão tem um limite razoável (nem sempre perceptível) onde a partir do qual o esforço ultrapassa os ganhos;
  ii. Perceber que normalmente não existe uma única solução para um dado problema;
  iii. Identificar entre os sub-problemas comportamentos semelhantes e agrupa-los.

- No uso do UML:

  i. Entender as vantagens da modelação como fase antecedente da implementação;
  ii. Compreender o potencial de uma linguagem de modelação como meio eficaz de comunicação entre as fases de concepção e implementação.

- Na implementação:

  i. Ser capaz de fazer a escolha da linguagem adequada à lógica a implementar;
  ii. Desenvolver código reutilizável, seja pela aplicação em causa, ou por outras aplicações;
  iii. Saber como implementar a hierarquia subjacente na modelação, com as particularidades definidas na norma.