Faculdade de Engenharia da Universidade do Porto

Padrões de Desenho em Soluções Programadas de Automação Flexível

Miguel Ribeiro Pereira

Dissertação no âmbito do Mestrado Integrado em Engenharia Mecânica
Opção de Automação
Secção de Automação, Instrumentação e Controlo

Orientador: Prof. Dr. António Pessoa de Magalhães

Setembro de 2011

"Sou uma parte de tudo aquilo que encontrei no meu caminho"

Acrescentado a Alfred Tennyson

…na vida profissional, somos todos os empregos que temos e todos os problemas que resolvemos

Agradecimentos

Gostaria de deixar os meus sinceros agradecimentos ao meu orientador Prof. António Pessoa de Magalhães, por conseguir que cada resposta sua fosse o nascer de outra pergunta, mostrando-me dessa forma o trajecto a percorrer. Sem essa ajuda não seria possível encontrar as respostas que me levariam ao caminho certo.

A nível profissional agradeço a todos os colaboradores de Europac Kraft Viana, em especial ao Eng.º Miguel Soares, pelo partilhar de sabedoria do mundo da automação e ao Eng.º Luís Teixeira pelo fulcral incentivo inicial que me fez voltar ao mundo académico, e por me dar a liberdade de conciliar da melhor forma o meu trabalho com o estudo.

A nível pessoal deixo aqui o agradecimento à minha família, sempre presente e pronta a dar todo o apoio necessário a cada momento, não só na minha dissertação, como em toda a minha vida. Por último agradeço à Patrícia Santos pela sobeja paciência denotada e por, de alguma forma, me acompanhar neste trajecto.

Resumo

Os padrões de desenho – *design patterns* – são uma forma de resolver problemas comuns, de forma metódica, integrada, testada e fiável. Diversas áreas da informática utilizam já padrões de desenho de forma sistemática. A informática industrial dá os seus primeiros passos nesta matéria, mas já com resultados promissores.

A aplicação de padrões de desenho a soluções de automação industrial, garante uma flexibilidade e modularidade aos sistemas, que permite seguir o paradigma actual da produção industrial - customização em massa.

Adicionalmente, a crescente complexidade e abrangência dos sistemas de automação torna-os difíceis de compreender, especificar, desenvolver, testar, modificar e integrar. Para minimizar estas dificuldades, têm emergido diversas normas, metodologias de especificação e ferramentas de apoio ao desenvolvimento. O principal objectivo deste trabalho é o de tentar contribuir para essa tendência ao procurar formas de desenvolvimento disciplinado, metódico e racional de *software* para sistemas flexíveis de automação tendo por base padrões de desenho.

O presente trabalho começa por enquadrar a investigação nas normas e conceitos existentes. Seguidamente são caracterizados os vários tipos de sistemas de produção industrial: discretos, contínuos e por partidas. A investigação experimental consiste na busca de padrões de desenho para dois sistemas típicos na indústria: Sorting e Batching. Para isso são consideradas diversas soluções de controlo que procurem garantir a máxima flexibilidade e modularidade a cada um dos sistemas. A plataforma de teste escolhida é o software ITS PLC.

O estudo apresentado comprova a importância da flexibilidade na automação e apresenta formas de a alcançar. Os resultados obtidos podem, e devem, ser extrapolados para outros tipos de sistemas industriais.

Abstract

Design patterns can be an approach to solve common problems, in a methodic, integrated, tested and trustful way. Several areas of computing, very frequently use design patterns. Although industrial computing has just begun to consider this matter, the results are very promising.

The application of design patterns in industrial automation, guarantees flexibility and modularity to the systems, following the current paradigm of industrial production – mass customization.

Additionally, the increasing complexity and extensiveness of automation systems makes it difficult to understand, develop, specify, test, modify and integrate them. To minimize these difficulties, there have been created standards, specification methodologies and tools that help the development of automation solutions. The main goal of this work is, based on design patterns, try to look for ways of disciplined, methodic and rational software development for flexible systems of automation.

This present work begins contextualizing the investigation in the standards and existing concepts. Afterwards, the various industrial production systems - discrete, continuous and batch - will be defined. The experimental investigation consists in searching design patterns for two different systems ordinarily used in industry: Sorting and Batching. For that, several solutions of control that guarantee maximum flexibility and modularity on each system, has been considered. The chosen test platform is the ITS PLC software.

This study proves the high value of flexibility in automation and presents ways of achieving it. The results can, and should be, expanded for other industrial systems.

Breve Currículo do Autor

Após concluir a licenciatura em Engenharia Mecânica – ramo de Automação Industrial - na Faculdade de Engenharia da Universidade do Porto em 2006/07, juntamente com três colegas de curso, fundei no Porto a empresa Real Games cujo objectivo era desenvolver um software inovador para a formação e treino da programação de PLCs. Esse desenvolvimento teve origem num projecto extra-curricular desenvolvido durante o último ano do curso, com a supervisão do orientador da presente dissertação. Desde 2006 até 2009, para além da co-gerência da sociedade, tive a meu cargo o desenvolvimento dos sistemas que compõem o software – ITS PLC (Interactive Training System for PLC) – e a programação de autómatos de forma a testar a funcionalidade dos sistemas. Em Dezembro de 2007 o software foi lançado comercialmente e desde então foquei-me na promoção, marketing, angariação de distribuidores e venda do software. A aceitação do produto foi bastante positiva e a experiência de empreendedorismo foi a todos os níveis gratificante e constantemente motivadora. Somente a sensação de se ver algo crescer desde o nada, exclusivamente através do nossos conhecimentos e esforço interno serviria para qualificar esta experiência profissional como única e inesquecível.

Não obstante, decidi que necessitava de ter um contacto mais próximo com a engenharia industrial e em Maio de 2009 integrei o núcleo de projectos de engenharia da empresa ibérica, líder de produção de papel de cartão, Europac Kraft Viana. Desde então e até à presente data, tenho vindo a desempenhar funções de desenvolvimento de projectos de melhoria produtiva, e o contacto com o dia-a-dia de um ambiente fabril de grandes dimensões tem sido um enorme e agradável desafio. Alguns dos trabalhos mais relevantes que tive a meu cargo foram a supervisão da montagem mecânica de uma nova central de cogeração eléctrica, colaborando activamente com fabricantes de referência como a Rolls-Royce, MAN Turbo ou UMAG, e o planeamento e coordenação de um significativo projecto de melhoria da parte mais fundamental da fábrica - a Máquina do Papel.

Recentemente decidi aceitar o desafio de emigrar para a Noruega, onde exercerei funções de projecto hidráulico de equipamentos vocacionados para a indústria petrolífera, para a multi-nacional National Oilwell Varco.

Índice

Índice de tabelas

Índice de figuras

1. Introdução

Quando um aluno termina a sua vida académica e inicia a sua carreira profissional, existe um sem-número de factores que podem condicionar o seu caminho. Sendo a Engenharia Mecânica, a meu ver, uma base de conhecimento e não uma profissão *per se*, existem muitos caminhos que um jovem engenheiro pode tomar, mas mais importante do que rapidamente encontrar o caminho, é aprender o mais possível durante o percurso. Com maior ou menor esforço, um bom profissional alcançará o trabalho pretendido na sua plena especificidade, mas existe tanto conhecimento a assimilar em qualquer área, que é redutor pensar-se que só quando for alcançada a total realização profissional é que um engenheiro deve explanar toda a sua capacidade e empenho.

No meu trajecto profissional tenho encontrado desafios distintos, desde o desenvolvimento de software de sistemas virtuais para o ensino e formação de automação industrial até à supervisão da montagem mecânica de uma central de cogeração, passando pela programação de autómatos e pelo projecto mecânico de soluções de melhoria produtiva num ambiente industrial muito específico, como é a produção de pasta e papel. Em cada uma das etapas tem-me sido possível aprender e potenciar novas valências e humildemente constatar a vastidão do mundo da engenharia. Não obstante, a paixão e a curiosidade orientada para o mundo da automação tem sido uma constante.

Mantendo o gosto pelos sistemas virtuais, que ajudei a desenvolver e que permitem hoje em dia que inúmeros alunos entendam um pouco melhor como implementar soluções de automação, deparo-me com um ambiente fabril profícuo em sistemas reais que ilustram tudo o que vem nos livros, e em que se aplica tudo do que se fala ao longo do curso.

O espaço físico de uma unidade fabril de grande dimensão é então propício para o lançar de questões várias sobre como nasce uma solução de automação. Acredito que a melhor solução é sempre aquela que permite a aplicação a mais do que um problema, e que pensar além da dificuldade que temos à nossa frente é uma boa forma de nos precavermos de situações de que não tenhamos saída.

Pela minha experiência, o *know-how* sobre a aplicação de soluções programadas de automação flexível na indústria, encontra-se maioritariamente nos grandes fornecedores de automação (Siemens, Honeywell, Rockwell Automation, Metso, etc.) e nas empresas que oferecem serviços de integração, que normalmente são compostas por antigos colaboradores dos grandes fornecedores de automação. A indústria limita-se a comprar o serviço e pouco acesso tem ao que está por detrás da interface do utilizador. Muitas vezes têm acesso ao código, mas o entender a arquitectura da solução e o porquê do escolher da solução A ao invés da B, é algo que não se infere facilmente.

A necessidade de saber, é algo que excede a necessidade de resolver um problema em concreto, e movido por esta minha vontade de encontrar respostas globais tendo por base problemas específicos, decidi voltar à minha nobre faculdade e aumentar os meus conhecimentos, podendo assim, quem sabe, descobrir pelo caminho outras perguntas que me façam ir mais além na minha vida profissional.

Quando encontramos um sistema que precisa ser controlado automaticamente, quais os passos que se devem tomar? Devemos procurar algum sistema semelhante e adaptar a solução? A cada implementação deve-se criar algo novo ou devemo-nos sempre basear nas "boas práticas" existentes? As normas existem para serem usadas ou apenas para servirem de *tábula rasa*? Até que ponto é possível traçar paralelismos entre diferentes sistemas? A programação orientada por objectos tem o seu lugar na automação? Devemos procurar soluções dedicadas ou flexíveis?

Estas são algumas perguntas conceptuais que estão na origem da escolha do tema para esta dissertação.

1.1. Conjectura e Objectivos

Desde o início da utilização massiva dos Controladores Lógicos Programáveis (PLC), que a flexibilidade de um sistema controlável tem vindo a aumentar. O que era *hard-wired* passou a ser gerido por software. O que era feito em software de linguagem própria passou a ser feito em software de linguagem normalizada. O que era centralizado passou a ser distribuído.

Com a soma destas vantagens é possível olhar-se para um sistema de automação de forma diferente. As capacidades de tecnologia computacional permitem criar soluções complexas num abrir e fechar de olhos. O aparecimento da norma IEC 61131-3 (IEC, 2003)[1], tornou essa programação totalmente universal e definiu um conjunto de linguagens que permitiu aos engenheiros desenvolverem uma solução de automação antes sequer de se pensar no hardware a utilizar. A automação distribuída é actualmente considerada uma escolha avançada que aumenta a eficiência e oferece redundância aos sistemas industriais (Dai & Vyatkin, 2010). A norma IEC 61131-3 tem algumas limitações no que diz respeito à aplicação a sistemas distribuídos, pelo que em 2005 foi criada a norma IEC 61499 (IEC, 2005) com o intuito de complementar a norma existente e responder a requisitos específicos do controlo distribuído e flexível (Dai & Vyatkin 2, 2010).

A flexibilidade da produção torna-se cada vez mais um factor chave para o sucesso de qualquer indústria. O equipamento físico de muitas unidades de produção industrial tende a ser facilmente reconfigurado, pelo que as soluções de automação têm que acompanhar essa tendência (Brandl, 2007). A criação de padrões de desenho de soluções de automação tem por objectivo facilitar a reconfiguração de sistemas sem a necessidade de reprogramação completa. Um dos objectivos de fundo da presente dissertação é mostrar que a automação não pode ser vista como o gargalo das soluções industriais flexíveis.

O conceito de padrões de desenho é amplamente usado na informática para resolver problemas recorrentes de programação. A aplicação desse conceito em soluções programadas de automação,

[1] A primeira versão da norma foi na realidade apresentada em 1993, com o nome IEC 1131. A versão actual, e a que é referenciada ao longo da dissertação, é a segunda versão e com a nova designação de IEC 61131. A parte 3 da norma refere-se às linguagens e metodologias de programação.

entronca na ideia de criar soluções gerais e eficazes que possam ser aplicadas a sistemas flexíveis ou a sistemas semelhantes entre si (mas não iguais).

A programação orientada por objectos, implementada por linguagens como o C++, C# ou Java tem como conceito mais básico a utilização de classes que definem um dado objecto genérico e a consequente instanciação dessa classe que define um objecto em particular, com todas as características já definidas pela classe.

A aplicação deste conceito à programação de PLCs não é visto como algo natural, mas olhando para a linguagem de blocos funcionais (*Function Blocks*) implementada pela IEC 61131-3 ou pelos bem específicos blocos funcionais da IEC 61499, nota-se uma clara convergência de ideias que é legítima explorar.

Esta dissertação conjectura que a aplicação de padrões de desenho ao desenvolvimento de soluções para sistemas de automação, garante à solução a máxima flexibilidade e modularidade.

Os objectivos concretos desta dissertação são os seguintes,

- Estudar a metodologia de criação de soluções flexíveis de automação;
- Criação de padrões de desenho e aplicação a sistemas típicos e representativos da indústria, procurando sempre chegar a um nível elevado de flexibilidade e modularidade;
- Utilizar como *testbed* às diversas experiências, dois sistemas distintos e relevantes da indústria - Sorting e Batching - disponíveis através de um software de sistemas virtuais de treino – ITS PLC;
- Análise de um sistema industrial real, sendo esse mais um teste à aplicação dos padrões de desenho encontrados.

Encontrar um tema unificante em todas as ideias apresentadas é algo que não se alcança de ânimo leve e deseja-se portanto que tal se revele como uma importante valia do estudo realizado.

1.2. Organização da Dissertação

A dissertação está organizada em 7 capítulos e 4 anexos. No capítulo 2 procura-se a definição clara do problema que justifica o tema da tese, apresentando o conceito geral de padrões de desenho e identificando o escopo das soluções de automação que irão ser investigadas. É ainda apresentado o estado da arte, com a devida revisão bibliográfica. O capítulo 3 entrega-se aos fundamentos teóricos – programação orientada por objectos, padrões de desenho na automação industrial, blocos funcionais segundo a norma IEC 61499 e sua comparação com os blocos funcionais segundo a norma IEC 61131-3. É também abordada a forma de que se parte destes fundamentos para se chegar às soluções flexíveis pretendidas. No capítulo 4 são caracterizados e distinguidos os vários tipos de processos industriais e seleccionados os sistemas virtuais que servirão de *testbed* à dissertação. Os capítulos 5 e 6 expõem toda a parte experimental deste trabalho e são analisadas e comparadas todas as abordagens tomadas

para o sistema Sorting e Batching respectivamente. No capítulo 7 são tecidas as conclusões finais e são propostos trabalhos futuros que possam tirar proveito desta dissertação.

Em anexo apresenta-se informação suplementar para a compreensão e apreciação do presente trabalho, nomeadamente os algoritmos das várias abordagens das soluções de automação e informação adicional sobre os sistemas virtuais utilizados.

2. Definição do Problema

A criação de soluções programadas de automação é um grande foco do ensino da automação industrial. Ensinam-se linguagens, abordagens e normas. Descrevem-se aplicações comuns e tecnologias de controlo. O desenvolvimento de padrões de desenho em soluções programadas de automação flexível procura juntar todas estas vertentes e elaborar um *template* aplicável à especificação e elaboração de soluções configuráveis, modulares, flexíveis, distribuíveis, suficientemente abrangentes e de fácil implementação.

Um padrão de desenho não é um desenho, é sim um template para resolver problemas complexos que se pode aplicar a situações diferentes, mas relacionadas entre si (Brandl, 2007).

2.1. Justificação e Formalização do Problema

O actual mercado global e as suas necessidades de consumo em constante alteração, provam ser um grande desafio às indústrias produtoras. O paradigma actual é a "customização" em massa ao invés da tradicional produção em massa. A necessidade de fabricar produtos diferentes ou de formas distintas com os mesmos equipamentos é cada vez mais premente. Especialistas de diferentes tipos de indústrias identificaram a tecnologia de sistemas de produção distribuída e reconfigurável como uma das tecnologias chave da próxima geração de sistemas de automação (Sunder, Zoitl, & Dutzler, 2006).

Aliado à vocação dinâmica da produção, a paragem de uma fábrica de laboração contínua traz custos elevadíssimos e indesejáveis. A tendência é minorar os tempos de paragem e fazer com que cada inversão fabril seja pouco sentida ao nível da quebra produtiva.

A experiência pessoal do autor sugere que o engenheiro de automação não consegue dominar por completo a crescente complexidade de uma solução global de automação. A estrutura comum dos programas de PLC não permite a implementação fácil de novas funções e a reconfiguração dinâmica e flexível. Para combater os problemas apresentados, são necessárias novas abordagens e arquitecturas de software.

Elaborar uma solução flexível implica a abstracção da mesma em relação aos equipamentos controlados. Só dessa forma é possível garantir que o mesmo código possa ser usado em diversas aplicações, sem necessidade de revisão completa da solução. Existe porém um limite para a abstracção de uma solução, a partir da qual o trabalho de configuração inicial da solução torna-se tão ou mais complexo do que alterar de raiz a própria solução

De forma a ilustrar tal problemática, é apresentada a Figura 1. Notar que a assumpção da razão entre a flexibilidade e a complexidade de configuração ser linear é meramente teórica e não é baseada em dados quantitativos.

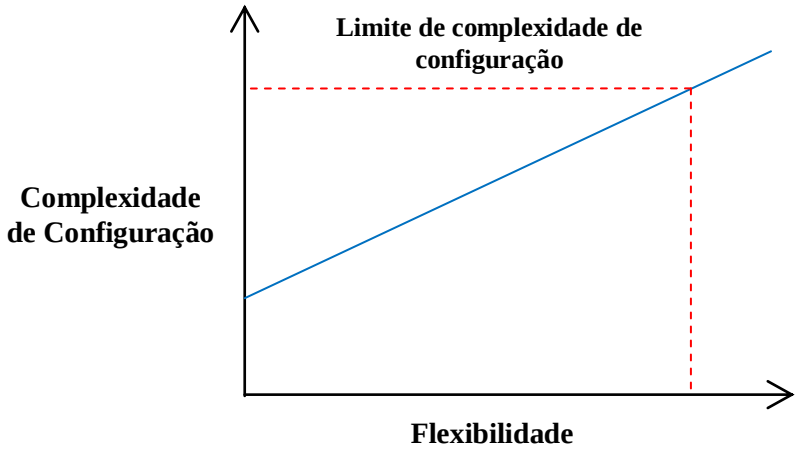


Figura 1 – Complexidade de configuração *vs* flexibilidade

O problema da criação de padrões de desenho em soluções de automação flexível pode ser sintetizado nos seguintes desafios de certa forma sequenciais:

- Analisar o sistema e distinguir claramente quais os equipamentos a controlar;
- Subdividir o sistema em módulos que possam ser susceptíveis de reconfiguração ou de diferentes aplicações noutros sistemas e que suportem soluções de controlo distribuído;
- Elaborar uma solução que permita o controlo de cada equipamento, a interligação entre eles e o funcionamento do sistema como um todo, tendo em conta as ordens de produção;
- Procurar outra configuração do sistema ou outros tipos de sistemas semelhantes que possam tirar partido da mesma solução;
- Alterar a solução de forma a abranger algum caso não previsto inicialmente;
- Repetir as últimas três etapas até se chegar ao compromisso ideal entre a flexibilidade e a complexidade de configuração.

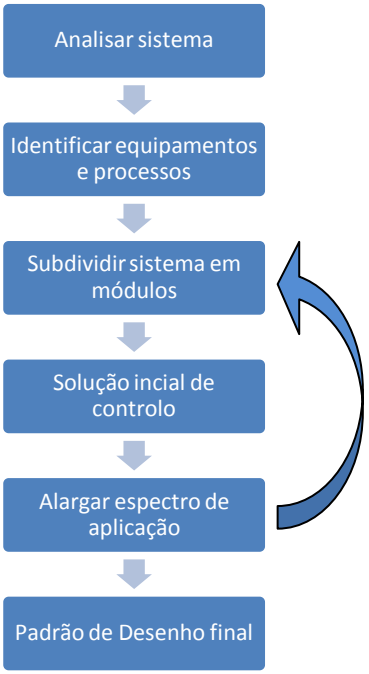


Figura 2 – Etapas da procura de soluções flexíveis de automação

Este conjunto de etapas ilustrados na Figura 2, revela que parte deste processo acaba por ser iterativo, no sentido em que apenas com a criação, experimentação e modificação de uma dada solução e a integração em diversos sistemas, é possível alcançar e atestar a sua flexibilidade e chegar ao padrão de desenho desejado.

De forma a testar estes conceitos, é necessário encontrar sistemas industriais que sejam relevantes na indústria generalizada e com esses sistemas procurar a melhor abordagem para a criação de soluções de controlo flexíveis.

A investigação desenvolvida procura as melhores práticas para cada tipo de sistema e debate-se sobre a definição do limite de abstracção da solução. A linguagem de blocos funcionais contemplada na norma IEC 61131-3 ou os blocos funcionais descritos na norma IEC 61499 tornam esta abstracção passível de ser executada.

Seguindo um princípio da programação orientada por objectos, cada equipamento ou processo (ou conjunto deles) a controlar pode ser definido por um bloco funcional - Figura 3. Como se definem cada um destes blocos, quão abrangentes devem ser, como se ligam entre si e de que forma deve ser feita a sua integração horizontal e vertical são questões que necessitam de atenta análise e experimentação.

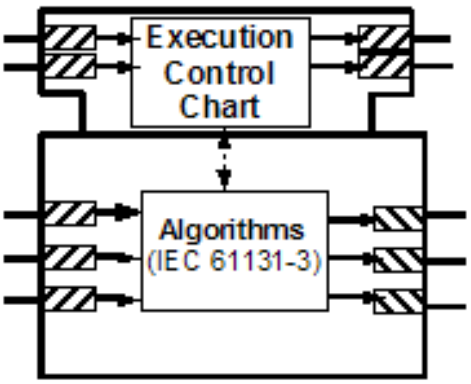


Figura 3 – Bloco funcional segundo a norma IEC 61499 (IEC, 2005)

A adopção da IEC 61499 a nível industrial é ainda escassa, principalmente devido à falta de ferramentas, métodos e modelos de implementação. Por isso a indústria

não terá ainda certamente considerado as vantagens da norma e do conceito fundamental de bloco funcional baseado em eventos.

Antevendo-se que não será provável encontrar uma solução unívoca para cada sistema em teste, procura-se demonstrar com as diversas experimentações o que se deve ou não fazer na busca de soluções flexíveis e passíveis de serem distribuídas.

2.2. Estado da Arte

Actualmente os **sistemas de controlo distribuídos** (DCS) - Figura 4 - permitem que as soluções de controlo sejam um conjunto de diversos controladores espalhados pela fábrica agregados de forma inteligível.

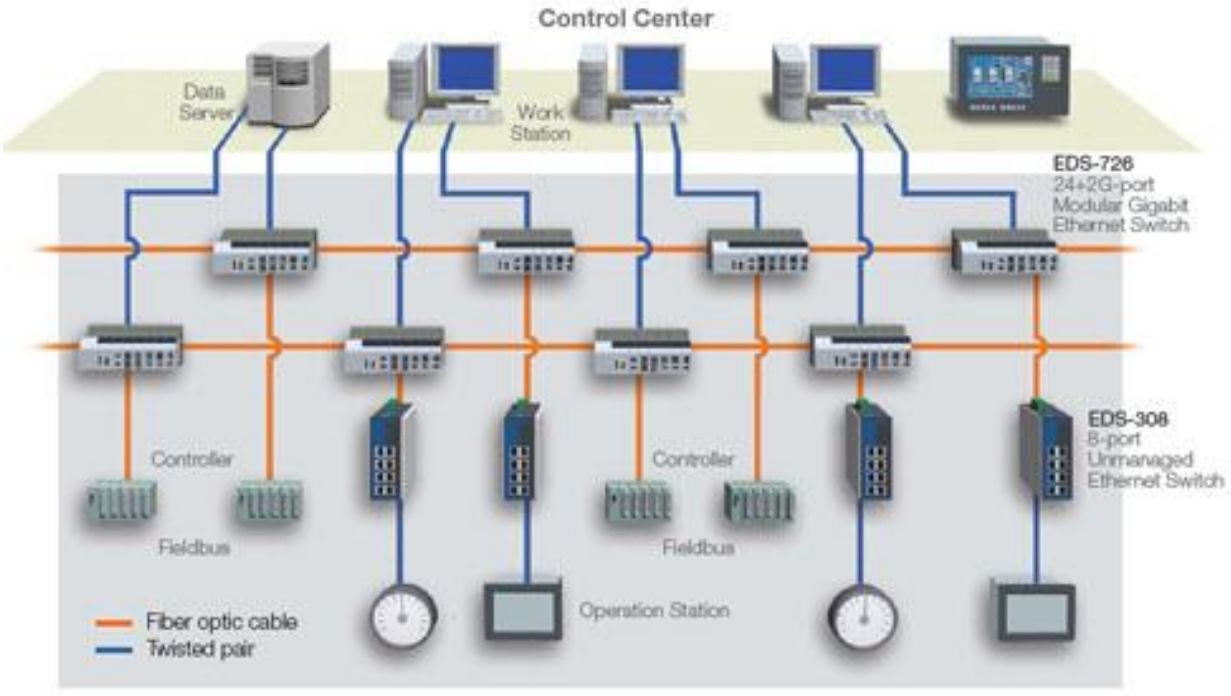


Figura 4 – Ilustração de um sistema de controlo distribuído (http://www.moxa.com)

A filosofia de controlo distribuído vaticina que cada recurso de produção tenha o seu próprio controlador. O dispositivo de controlo cumpre a sua tarefa e comunica com o sistema de controlo central, que por sua vez comunica com outros dispositivos de forma a cumprir o objectivo global.

O *know-how* por detrás de um DCS está quase sempre do lado dos fornecedores destes sistemas, pelo que a reconfiguração e adaptação do sistema a novos desafios produtivos passa sempre por intervenção ou comissionamento do fornecedor.

Na busca de novas soluções distribuídas, certos tipos de processos, tais como sistemas de *sorting*, foram alvo de experiencias em que cada transportador é controlado por um dispositivo embebido com comunicação sem-fios entre dispositivos. Este tipo de abordagem multi-agente foi aplicada com

sucesso, tendo como factor limitador o elevado volume de informação comunicada entre agentes e consequente ineficiência (Hayslip, Sastry, & Gerhardt, 2006).

Os **padrões de desenho** apareceram primeiramente na informática em 1987 devido à necessidade de resolver problemas de programação de forma metódica e *standard*. Os padrões de desenho podem ser vistos como uma estratégia que define como atacar determinado problema, não exclusivamente na programação.

A aplicação de um padrão de desenho numa solução de programação pressupõe apenas que o programador entenda claramente o problema e que conheça o padrão a aplicar. A eficiência de trabalho e a garantia de que um problema conhecido será resolvido de uma forma rápida, inequívoca, assente em paradigmas estabelecidos e sem possibilidade de erros são as claras mais-valias da aplicação de padrões de desenho.

Em 1995 foi elaborada a norma ANSI/ISA-88.01 (a IEC 61512-1 em 1997 adaptou para a Europa esta norma americana) que aborda a criação de padrões de desenho com vista ao controlo de processos flexíveis por partidas (batching). Entretanto apareceram aplicações da norma em processos discretos e contínuos. A adopção desta norma permitiu, a um vasto tipo de indústrias, cortes substanciais nas despesas de projecto e de operação (Brandl, 2007).

Os padrões de desenho S88 e NS88 expandem a ANSI/ISA-88.01 e demonstram métodos para a implementar. Estes padrões apresentam as bases de uma solução de controlo flexível e contemplam características fundamentais tais como: separação receita / equipamento, várias camadas de controlo (básico, procedural e coordenador), modularização e células de processamento (Figura 5).

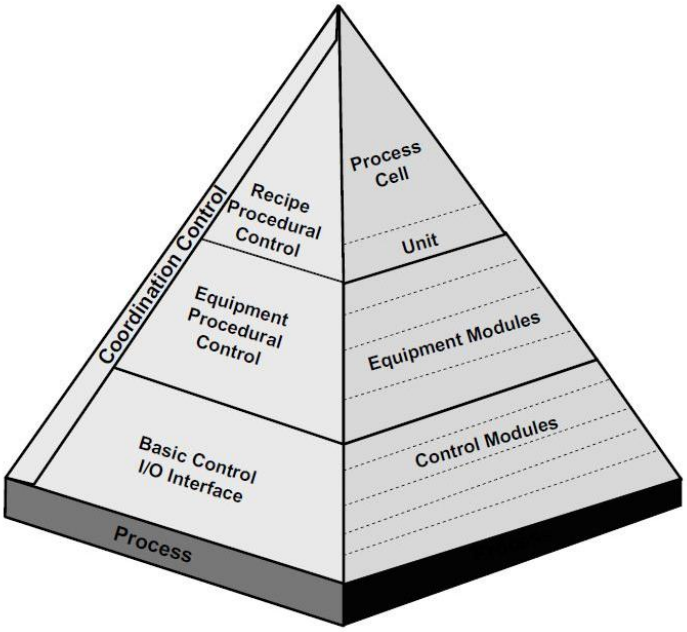


Figura 5 – Hierarquia de controlo e de equipamentos segundo a norma ISA-88.01 e o padrão S88 (Brandl, 2007)

Ao nível da automação e controlo por PLC, a norma **IEC 61131** define um conjunto de regras para a utilização e programação de PLCs. A parte 3 da norma define o modelo de software e as linguagens que podem ser usadas - diagrama de contactos, texto estruturado, lista de instruções, gráfico de funções sequenciais, e diagrama de blocos funcionais. Presentemente, qualquer fabricante de autómatos tende a seguir esta norma e implementa a maioria das linguagens definidas ou cria a sua própria variante com base nas mesmas. Do ponto de vista do engenheiro de automação, a uniformização de modelos de software e de linguagens trazida

pela IEC 61131-3 torna mais fácil a criação de aplicações, independentemente do fabricante do controlador.

A norma **IEC 61499** propôs-se melhorar a portabilidade, reconfigurabilidade e interoptabilidade da norma IEC 61131-3 (ver Figura 6). A portabilidade é dada pela facilidade de utilização de bibliotecas de software ou blocos de controlo. A reconfigurabilidade é dada pela agilidade com que se consegue alterar uma solução de controlo. A interoperabilidade caracteriza a possibilidade de ligação e comunicação entre equipamentos diversos.

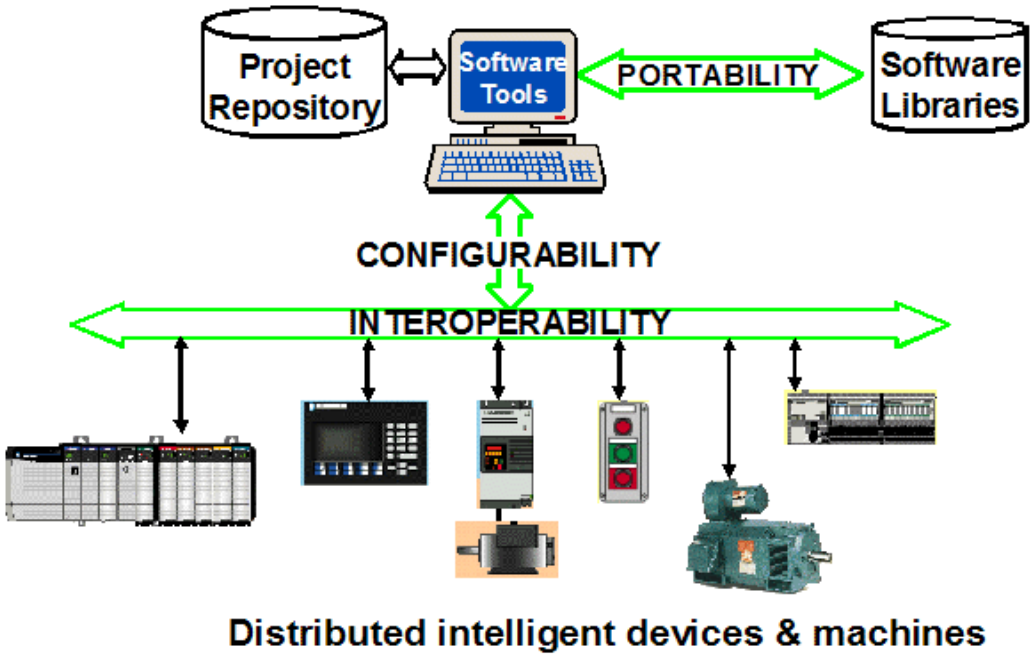


Figura 6 – Requisitos cumpridos por uma solução de controlo segundo a IEC 61499 *(http://knol.google.com/k/iec-61499)*

O elemento fundamental da IEC 61499 é o **bloco funcional** que permite encapsular código de diferentes linguagens e executá-lo ao ser disparado o evento correspondente (Figura 7).

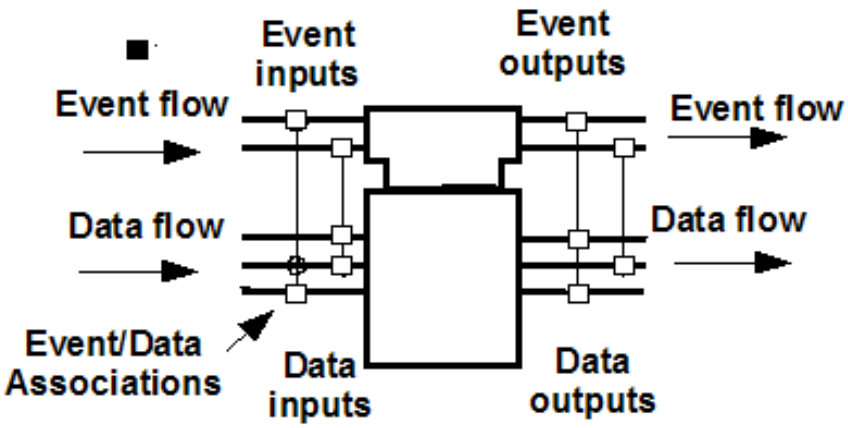


Figura 7 – Bloco funcional segundo a IEC 61499 e o seu fluxo de eventos e informação (IEC, 2005)

A norma IEC 61131-3 nasceu da necessidade de reconciliar múltiplas implementações de linguagens de controlo por PLC já usadas. A IEC 61499 foi proposta e desenvolvida antes da utilização comum dos blocos funcionais "*event-based*" e dos programas de controlo distribuído. Surge então um problema complicado: não se desenvolvem ferramentas para um *standard* até que esse seja de utilização franca, mas ao mesmo tempo não existe utilização massiva antes que sejam disponibilizadas as ferramentas correctas.

Actualmente existe um conjunto limitado de ferramentas de software para o desenvolvimento de soluções com blocos funcionais segundo a IEC 61499. As mais relevantes são as seguintes:

- FBDK – Function Block Development Kit

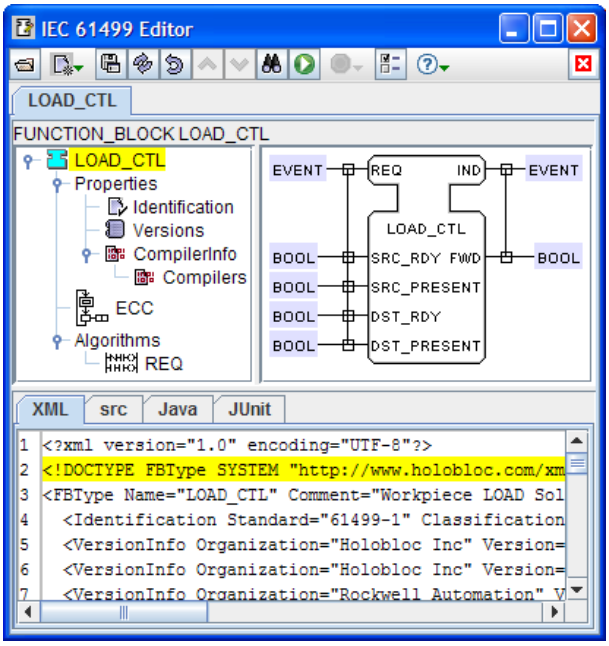


Figura 8 – Exemplo de bloco funcional desenvolvido no FBDK *(http://www.holobloc.com)*

- IsaGraf

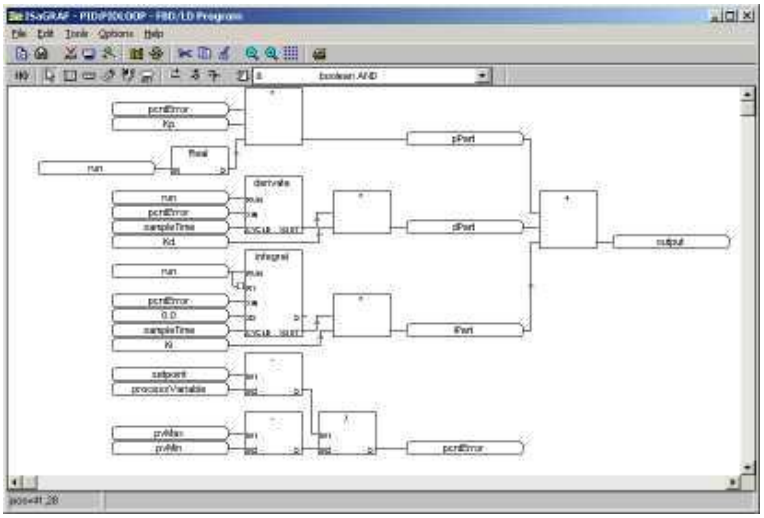


Figura 9 – Exemplo de diagrama de blocos desenvolvido no IsaGraf *(http://www.isagraf.com)*

- FBench

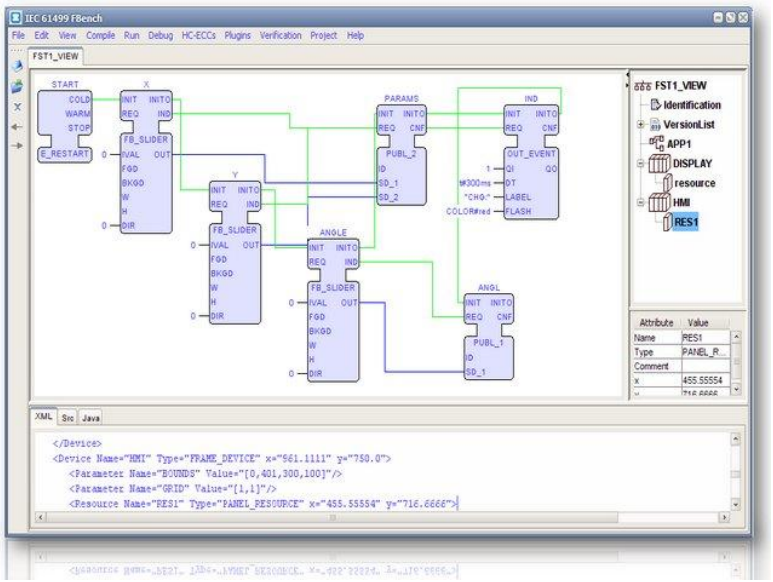


Figura 10 – Exemplo de diagrama de blocos desenvolvido no FBench *(http://www.ece.auckland.ac.nz/~vyatkin/fbench/)*

Mais detalhes sobre estes softwares podem ser encontrados nas páginas Web indicadas.

Resumidamente, nenhuma destas soluções é ainda robusta, totalmente funcional e com aplicabilidade prática em ambiente industrial.

No que diz respeito à estrutura de uma solução de automação, tradicionalmente esta é inconsistente com a estrutura física do sistema. Uma das causas é o facto das ferramentas de programação de PLC pouco encorajarem a correcta estruturação (ao contrário de qualquer ferramenta de programação de linguagem informática de alto nível, orientada por objectos por exemplo).

Em 2000 foi proposta a utilização da norma IEC 61499 para a modelação e implementação de **sistemas de controlo multi-agente holónicos** (Fletcher, Garcia-Herreros, Christensen, Deen, & Mittmann, 2000). Um sistema de controlo multi-agente é uma colecção de agentes individuais, em que cada um denota autonomia no que diz respeito às suas acções e percepções. A aplicação global é a soma de cada aplicação autónoma. Cada agente encapsula o seu funcionamento e torna a solução modular. Na abordagem holónica, cada agente visível pode ser constituído por n sub-agentes (Schillo & Fischer).

Existe uma grande diversidade de tentativas de melhorar a eficiência da engenharia e reengenharia de sistemas de automação. Linguagens abstractas tais como o UML – Unified Modelling Language podem servir para a descrição genérica de sistemas, mas a sua implementação tem demasiadas especificidades para ser de uso geral. A arquitectura pensada segundo a IEC 61499 parece oferecer o melhor rácio entre abstracção e facilidade de implementação.

Diversos projectos de investigação, académicos e industriais, procuram solucionar desafios específicos do projecto e implementação de soluções de controlo flexível e distribuído, e todos eles definem os blocos funcionais como o elemento básico da solução. Um inquérito, mostra que o

denominador comum das investigações nesta área é a abordagem da **programação orientada por objectos aplicada a blocos funcionais** (Frey & Hussain, 2006). Esta permite, de forma natura,l a reutilização de partes do código de uma solução em diferentes aplicações. O reaproveitar de fragmentos de código ou de conceitos garante celeridade na resolução de problemas habituais para os programadores

Por muito útil que seja, a reutilização de software é um problema nada trivial e depende não só dos meios disponibilizados pelas linguagens e protocolos, mas também da própria estrutura da aplicação. Surge então um ponto onde praticamente todas as investigações coincidem. **A estrutura da solução de controlo deve estar organizada da mesma forma que o sistema físico a controlar**. A Figura 11 ilustra um exemplo prático em que isso é evidente.

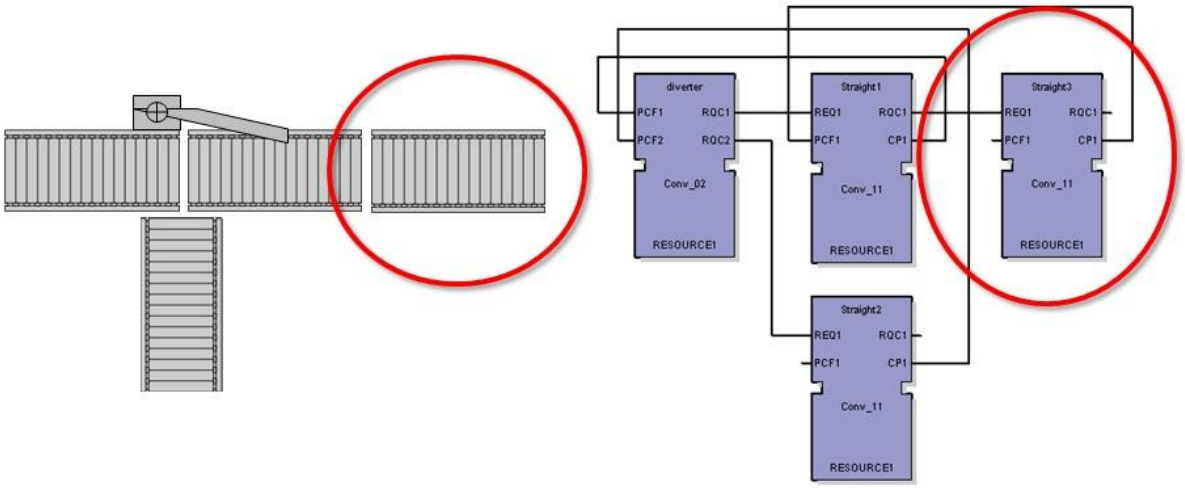


Figura 11 – Exemplo de solução de controlo seguindo a disposição física do sistema

Tal como nas linguagens de alto nível, na automação industrial pode-se considerar uma estrutura de blocos de software que replique a estrutura física da instalação a controlar (Dai & Vyatkin 2, 2010). Esta semelhança entre a estrutura de software é o sistema real faz com que seja natural a sua configuração e reconfiguração. Em 2006 foi proposta uma arquitectura de software de automação que implementa esta abordagem. A norma IEC 61499 foi escolhida para modelar esta arquitectura, tirando partido dos seus blocos funcionais. Existem diversas experiências funcionais que demonstram a viabilidade desta solução (Sunder, Zoitl, & Dutzler, 2006).

Com base nas conclusões e opiniões reunidas nas diversas publicações referenciadas, a Tabela 1 apresenta uma comparação entre soluções de automação de arquitectura tradicional e de arquitectura distribuída e flexível.

Tabela 1 – Soluções convencionais *vs* soluções distribuídas e flexíveis

	Solução Convencional	**Solução Distribuída e Flexível**
Eficiência de engenharia	• Baixa • Reutilização de código limitada faz com que o trabalho de engenharia em qualquer alteração do sistema seja moroso • Grandes esforços para pequenas alterações	• Alta • Vasta reutilização de código, facilita o trabalho de engenharia
Flexibilidade	• Reconfiguração possível mas dispendiosa	• Adaptação rápida • Alterar uma parte não implica alterar o todo
Escalabilidade	• Limitada	• Podem-se adicionar ou remover componentes facilmente
Robustez	• Considerável • Se falha um controlador ou um bloco, falha todo o sistema	• Cada falha ou problema fica confinado ao próprio componente
Manutenção	• A falta de estruturação de software torna qualquer manutenção uma tarefa complexa	• O encapsulamento de componentes e a definição clara de interfaces permitem a fácil substituição de componentes para manutenção

As vantagens da solução flexível são claras, sendo notoriamente legítimo o esforço de investigação denotado nesta área.

2.3.Linhas de Investigação

As linhas de investigação e desenvolvimento a adoptar, prendem-se com a criação, experimentação e análise de soluções flexíveis de controlo por PLC de dois sistemas típicos na indústria – Sorting e Batching.

Usando um PLC que adopte minimamente a norma IEC 61131-3, é possível obter-se uma aproximação do que pode ser uma implementação fidedigna da norma IEC 61499 e seus blocos funcionais *event-based*. Desta forma, convertendo os eventos em variáveis booleanas, em cada ciclo do controlador, cada bloco funcional verifica se os seus *event inputs* são verdadeiros. Se o forem, é

então processado o código interno e actualizados os *outputs*. Se nenhum *event input* for activado, passa-se para o próximo bloco funcional. Adicionalmente, as variáveis usadas serão todas locais, para não se quebrar o encapsulamento.

Esta aproximação tem a desvantagem que cada bloco funcional irá verificar os seus *event inputs* a cada ciclo, o que implica tempo de execução mesmo que nenhum evento decorra. Esta nuance faz com que a solução se afaste do verdadeiro espírito da IEC 61499, mas as conclusões tiradas, com a excepção da constatação da eficiência de código, são válidas e relevantes.

O controlador escolhido é o CoDeSys v2.3 da 3S Software (Figura 12). O CoDeSys é um Soft PLC que implementa todas as linguagens previstas na IEC 61131-3 e que permite a interligação ao sistema a controlar através de OPC.

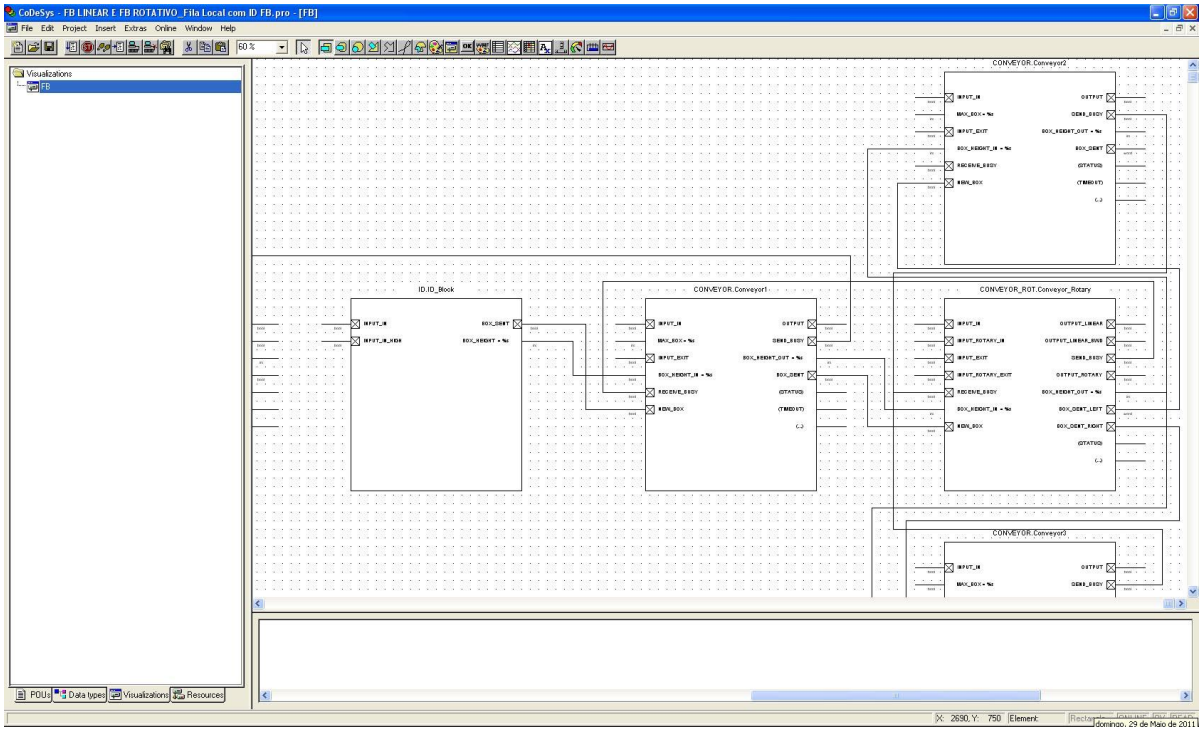


Figura 12 – Exemplo de diagrama de blocos desenvolvido no CoDeSys

Na impossibilidade de usar sistemas reais para o desenvolvimento das soluções de controlo, foi escolhido um software que emula sistemas virtuais passíveis de serem controlados por PLC. Este software denominado ITS PLC (Figura 13), da empresa portuguesa de software Real Games Lda, é presentemente a melhor solução para o treino da programação de PLC e os seus sistemas servirão de *sandbox* para toda a investigação e experimentação.

Figura 13 – ITS PLC – Interactive Training System for PLC (http://www.realgames.pt)

O ITS PLC recorre a gráficos 3D em tempo real, física, som e total interactividade para criar um ambiente validador de soluções de automação. Mais informações sobre o software são dadas em anexo e toda a informação necessária encontra-se disponível online em www.realgames.pt.

Figura 14 – Logótipo da fundação OPC

A interligação dos sistemas virtuais com o controlador CoDeSys é feita através de OPC Data Access V2 (OPC Foundation). Notar que foi usada uma versão especial do ITS PLC que permite precisamente a ligação por OPC, ao invés da ligação cablada a um autómato (ver Figura 13).

Os sistemas virtuais escolhidos de entre os cinco presentes no ITS PLC, são o Sorting e o Batching (Figura 15). Mais adiante nesta dissertação será explicado o funcionamento de cada sistema.

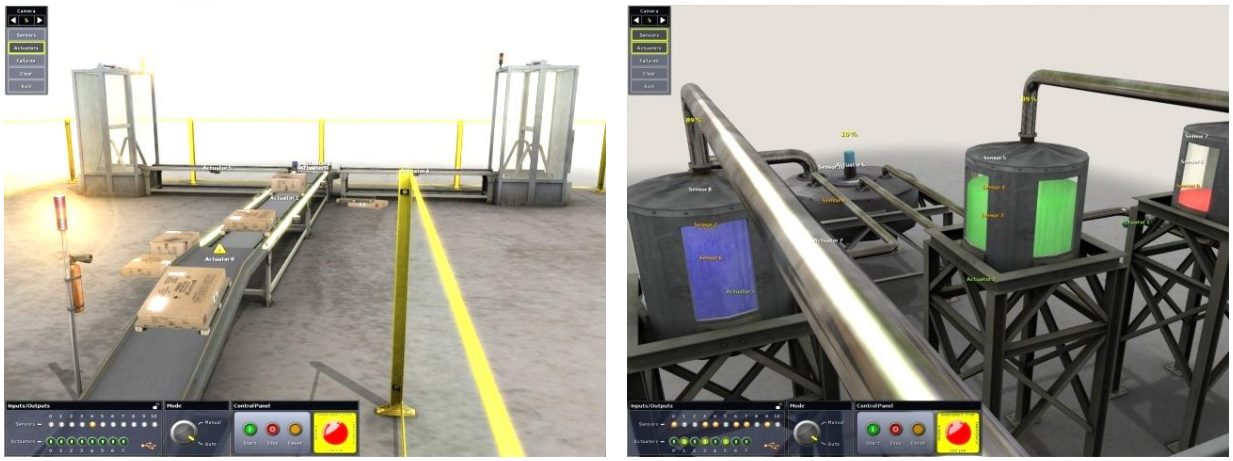


Figura 15 – Sistemas escolhidos para a investigação - Sorting e Batching

As ideias e conclusões tiradas da experimentação de soluções de automação nos sistemas virtuais serão então aplicadas, de forma teórica, a um sistema real existente na fábrica de pasta e papel Europac Kraft Viana, em Viana do Castelo (Figura 16). Mais detalhes sobre o sistema serão dados no capítulo 5.6.

Figura 16 – Transporte de bobines de papel kraft. Cortesia da Europac Kraft Viana

Em síntese, constata-se que muitos sistemas de produção industrial estão fisicamente dispersos em áreas vastas demais para serem controlados por um único programa a correr num único controlador. Com um DCS, tradicionalmente, o método usado é o de decompor o sistema em subsistemas mais restritos, programar cada um deles separadamente e depois criar código que congregue os vários subsistemas como um todo. Recentemente surgiram desenvolvimentos tecnológicos e conceptuais que ajudaram na implementação de sistemas de controlo distribuído verdadeiramente flexível.

Existem diversas experimentações e investigações no âmbito da aplicação da IEC 61499 ao controlo de sistemas distribuídos e flexíveis. Genericamente as conclusões encontradas são:

- A modularidade garante a flexibilidade pretendida;
- A eficiência de manuseamento de informação ainda não é a melhor devido à quantidade de informação necessária de comunicar entre blocos funcionais;
- Há uma melhoria clara da qualidade do software e uma redução do esforço de desenvolvimento de sistemas de controlo distribuído (Zoitl & Vyatkin, 2009);

- Presentemente não existem soluções comerciais de software que implementem de forma completa e funcional a norma. Tal não significa contudo o fracasso da aplicação da norma.

A criação de soluções flexíveis de controlo por PLC é um tema de relevância industrial e a notória inexistência de referências práticas da sua elaboração, faz com que as experimentações presentes nesta dissertação sejam uma mais-valia para qualquer apaixonado da automação e uma sólida base de trabalho para a sua implementação em diversos sistemas de produção industrial.

Por não existir actualmente software que implemente totalmente a norma IEC 61499, esta dissertação vai apenas abordar levemente a sua aplicação efectiva e será feita uma aproximação da sua utilização através da IEC 61131-3.

Sistemas de automação de controlo distribuído, padrões de desenho, programação orientada por objectos, IEC 61131-3, IEC 61499, controlo flexível e reconfigurável são as palavras-chave do estado da arte em foco nesta dissertação.

De um modo geral esta dissertação segue o principio de que, mais importante do que retirar conclusões unívocas, é estudar o que está por trás de cada elemento de uma solução de automação.

Os sistemas virtuais presentes no software de simulação para o treino e formação da programação de PLCs – ITS PLC Professional Edition, servirão de *testbed* para as ideias enunciadas ao longo deste trabalho e consequente criação de padrões de desenho. Com base no ambiente industrial em que estou inserido e na relevância geral de cada tipo de sistema no mundo industrial, os *case studies* abordados nesta dissertação são exemplos proeminentes dos tipos de sistemas de produção discretos e por partidas – Sorting e Batching.

3. Fundamentos Teóricos

Os fundamentos teóricos relevantes para o âmbito da presente dissertação são os seguintes: programação orientada por objectos, padrões de desenho, a norma IEC 61499, com especial atenção aos seus blocos funcionais e sua comparação com os blocos funcionais da IEC 61131.

Este capítulo não pretende ser exaustivo, mas apenas estabelecer bases teóricas que provem ser importantes para a consecução da investigação e o seu entendimento.

3.1. Programação Orientada por Objectos

A programação orientada por objectos é uma abordagem de programação baseada numa hierarquia de classes e objectos bem definidos.

Uma **classe** é uma estrutura que define a informação e os métodos de trabalhar essa mesma informação.

Um **objecto** é uma **instância** de uma classe ou uma cópia executável. Uma classe pode ser instanciada as vezes necessárias e a informação de cada objecto é **encapsulada**.

Os objectos implementam **métodos** que consistem numa sequência de acções de forma a assimilar os *inputs* existentes e produzir um ou mais resultados (Oracle Sun Developer Network, 2011).

A Figura 17 ilustra estes conceitos e relações entre si.

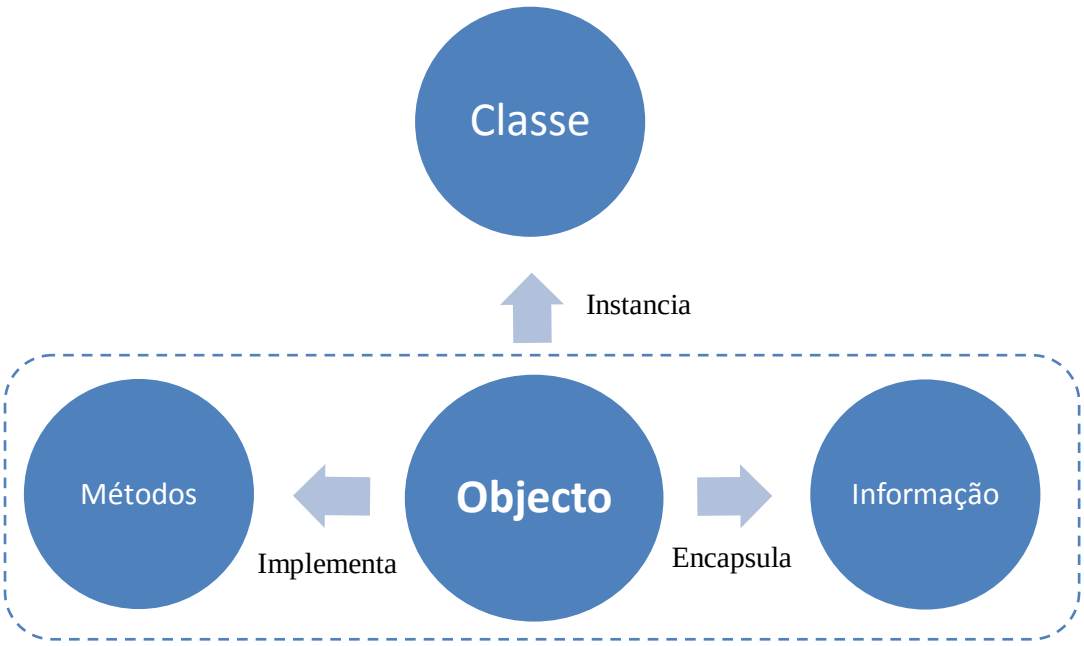


Figura 17 – Princípios fundamentais da programação orientada por objectos

Existem mais conceitos que definem a programação orientada por objectos, mas que não serão abordados neste trabalho – herança, polimorfismo, abstracção, etc.

3.2. Padrões de Desenho na Automação

A aplicação actual de padrões de desenho na automação ainda não é particularmente notória, existe contudo um esforço considerável de normalização nesse sentido, que assenta nas seguintes normas e padrões.

A norma ANSI/ISA-88.01 (ou a equivalente IEC 61512-1) e os padrões de desenho sobre ela desenvolvidos S88 e NS88 apresentam as especificidades que uma solução de automação deve ter, de forma a ser flexível.

Separação Receita / Equipamento

O ponto de partida dos padrões de desenho S88 e NS88 é existir uma separação clara entre a definição do produto e a definição do funcionamento do equipamento.

A definição de produto pode ser vista como uma receita que especifica o que é necessário produzir, quando, e com que quantidade. O equipamento consiste em todo o conjunto funcional que permite a produção de um dado produto. O equipamento recebe as instruções da receita, interpreta-as e dedica-se à produção.

A receita assume a existência do equipamento e especifica como o usar, mas não especifica como é feita cada acção que o equipamento executa. São as capacidades do equipamento que definem como as acções são executadas (Figura 18 e 19).

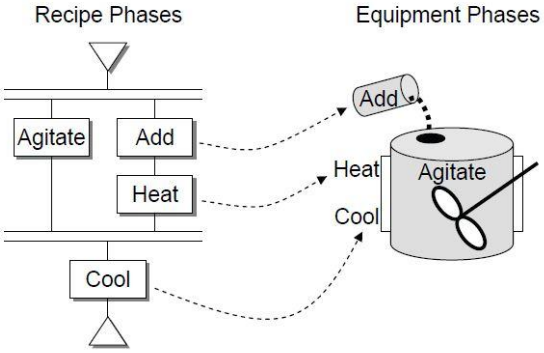


Figura 18 – Exemplo de separação entre receita e equipamento (Brandl, 2007)

Pegando no exemplo que será abordado no capítulo 6 – sistema por partidas de produção de tintas - a receita será a definição da cor de tinta a produzir e de todas as variáveis que podem ser alteradas numa partida. O equipamento é o sistema em si, cada válvula, doseador e misturador.

O acto de criação de uma receita de produção deve ser suficientemente simples para não necessitar dos serviços de um engenheiro de automação. Dessa forma a receita deverá seguir uma estrutura bem definida e de fácil entendimento. A mesma receita pode ser executada em sistemas distintos, desde que implementem os mesmos equipamentos.

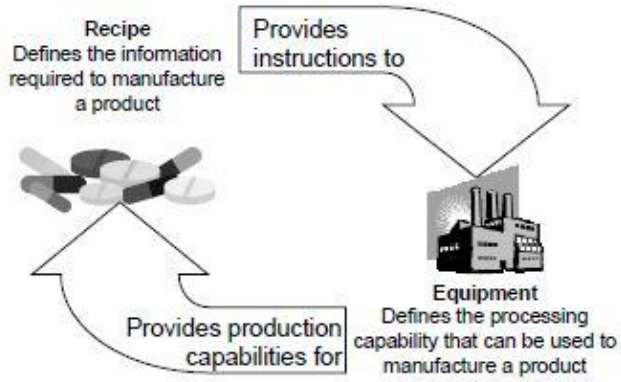


Figura 19 – Forma de separação entre receita e equipamento (Brandl, 2007)

Existem diversos tipos de receitas – geral, local, mestre e de controlo. Para as aplicações desenvolvidas neste trabalho, a distinção entre elas não é relevante. Importa, no entanto, reter o conceito global de receita, e a noção de que abstraindo o equipamento em relação à receita é possível produzir diversos produtos, relacionados entre si, num só equipamento ou conjunto de equipamentos, sem necessidade de trabalhos morosos de reconfiguração.

Tipos de Controlo

De forma a executar as ordens de produção e funcionamento, um sistema necessita de vários tipos de controlo: coordenação, procedural e básico (Figura 20). As funções e características de cada tipo são as seguintes:

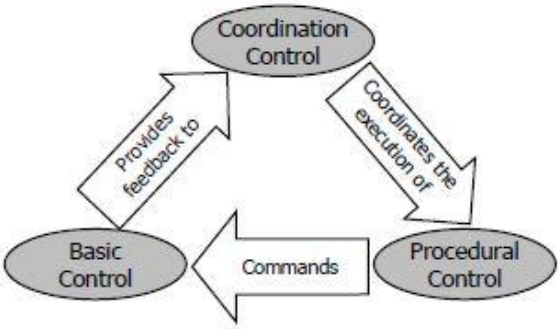


Figura 20 – Os três tipos de controlo no padrão de desenho S88 (Brandl, 2007)

- Controlo de Coordenação
    - Coordena a execução do controlo procedural;
    - Reserva, atribui e liberta recursos;
    - Existem soluções comerciais específicas para este nível de controlo (*recipe-execution systems*).
- Controlo Procedural
    - Comanda o controlo básico;
    - Interpreta as receitas e converte-as em ordens de produção;
    - Dirige as acções e dá-lhes uma sequência de forma a se obter o produto desejado
- Controlo Básico
    - Executa a lógica de controlo ao nível dos equipamentos e processos;
    - Normalmente implementado usando uma das em linguagens contempladas na norma IEC 61131-3.

Os tipos de controlo abordados com mais detalhe neste trabalho serão o controlo procedural e básico. Contudo, devido à sua separação não ser clara nas abordagens previstas, é de menor importância o estudo mais aprofundado destes fundamentos teóricos.

Equipamentos

Os equipamentos podem ser vistos segundo uma hierarquia, que deve ser seguida de forma a manter a flexibilidade. As definições presentes na hierarquia representada, baseiam-se na norma ISA-95. A Figura 21 mostra a hierarquia de módulos e células (face direita) e a hierarquia de controlo relacionada (face esquerda).

Note-se a relação entre células de processo, unidades, módulos de equipamento e módulos de controlo com os vários tipos de controlo e o carácter transversal do controlo de coordenação.

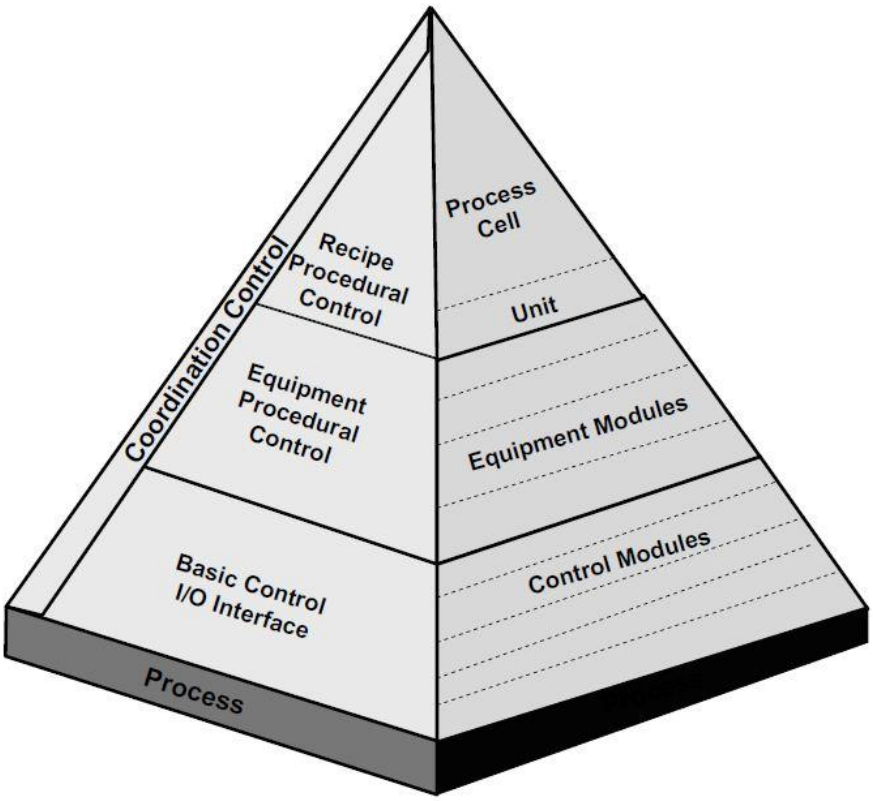


Figura 21 – Hierarquia de controlo e de equipamentos segundo o padrão S88 (Brandl, 2007)

De seguida são apresentados em traços gerais, as definições e regras dos padrões de desenho para cada nível hierárquico de equipamento definido. Na Figura 22 é também possível entender melhor como os vários níveis são integrados.

Célula de Processo (Process Cell)

- Contém todo o equipamento físico e capacidades de controlo para produzir completamente um ou vários produtos;
- As fronteiras de uma célula de processo são normalmente coincidentes com os pontos do sistema onde o produto perde a sua identidade. i.e. Pontos de separação ou sorting, pontos de convergência ou mistura;
- Gere e coordena múltiplas ordens de produção e atribui no máximo uma ordem por cada unidade;

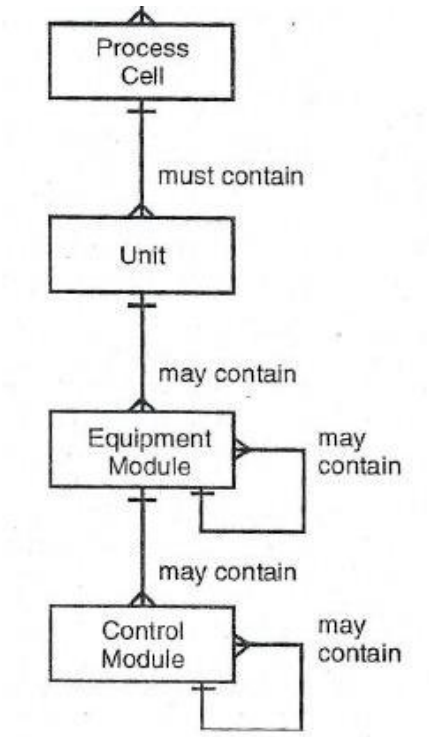


Figura 22 – Hierarquia de equipamentos segundo o padrão S88 (Brandl, 2007)

- Incorpora controlo de coordenação que é usado pelas unidades para alocar e libertar recursos partilhados;
- Possui controlo procedural para interpretar a receita;
- De que forma a unidade executa o controlo do equipamento não é visível para a célula de processo;
- Origina um registo de todas as acções executadas na produção.

Nos sistemas estudados nos capítulos futuros, a célula de processo pode ser vista como o conjunto de todos os equipamentos e das suas capacidades de controlo.

Unidade (Unit)

- Contém todo o equipamento físico e capacidades de controlo para executar uma só acção relevante na produção de um produto;
- Opera num único produto de cada vez;
- Opera de forma relativamente independente, com a excepção do momento em que transfere material de ou para outra unidade;
- Pode operar o equipamento a montante ou a jusante;
- É composta por um ou mais módulos de equipamento dedicados, ou partilhados;
- Possui controlo de coordenação para a utilização de módulos de equipamento ou outros recursos partilhados;
- Módulos de equipamento e módulos de controlo podem ser partilhados dentro de uma unidade ou através de diversas unidades dentro de uma célula de processo;
- A forma que um módulo de equipamento executa o controlo do equipamento não é exposta à unidade. Apenas os resultados, estados e modos são expostos;
- Pode ser uma instância de uma *Unit Class*.

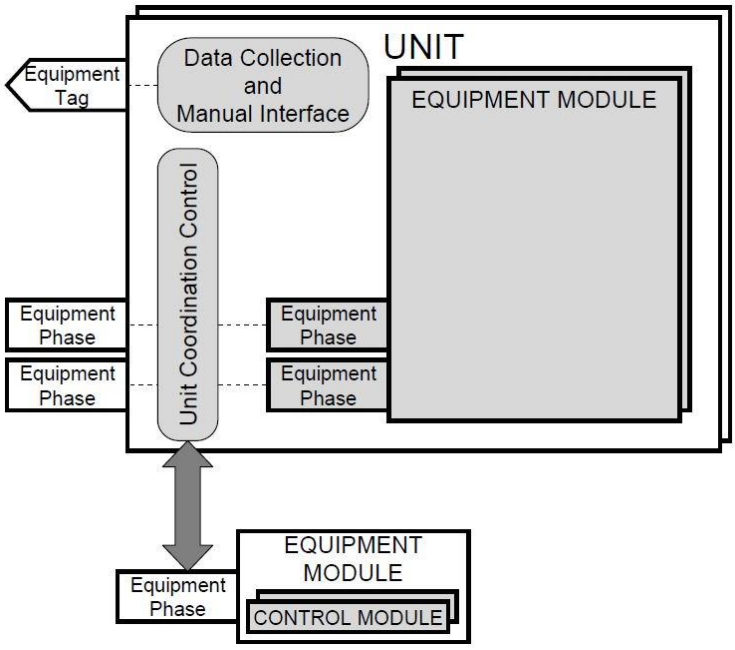


Figura 23 – Definição de unidade segundo o padrão de desenho S88 (Brandl, 2007)

Nos sistemas abordados, uma unidade pode ser interpretada como o conjunto de um dado equipamento físico com as suas próprias capacidades de controlo.

Módulo de Equipamento

- Executa a lógica procedural necessária ao funcionamento de um equipamento;
- Pode implementar fases de equipamento que são o encapsulamento de uma acção específica de um equipamento;
- É constituído por zero ou mais módulos de controlo;
- Pode ser exclusivo para uma só unidade ou ser partilhado por várias unidades;
- A receita não tem acesso à forma com que o módulo de equipamento executa acções de controlo;
- O controlo procedural é executado através de sequência de etapas de controlo;
- Acções de controlo são os comandos dados a outros módulos de equipamentos ou aos módulos de controlo;
- Suporta modo de operação manual ou automático.

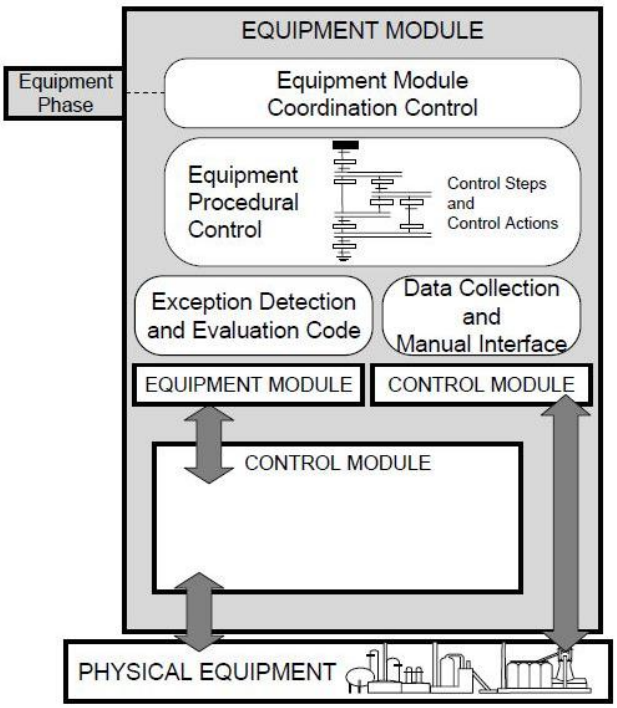


Figura 24 – Definição de módulo de equipamento segundo o padrão de desenho S88 (Brandl, 2007)

Nos sistemas abordados, um módulo de equipamento em conjunto com um módulo de controlo será a constituição fundamental dos blocos funcionais desenvolvidos.

A Figura 24 ilustra um exemplo de um sistema de batching em que são representados os vários módulos de equipamento – cadeias de controlo (sensores e válvula), misturador e válvula de descarga.

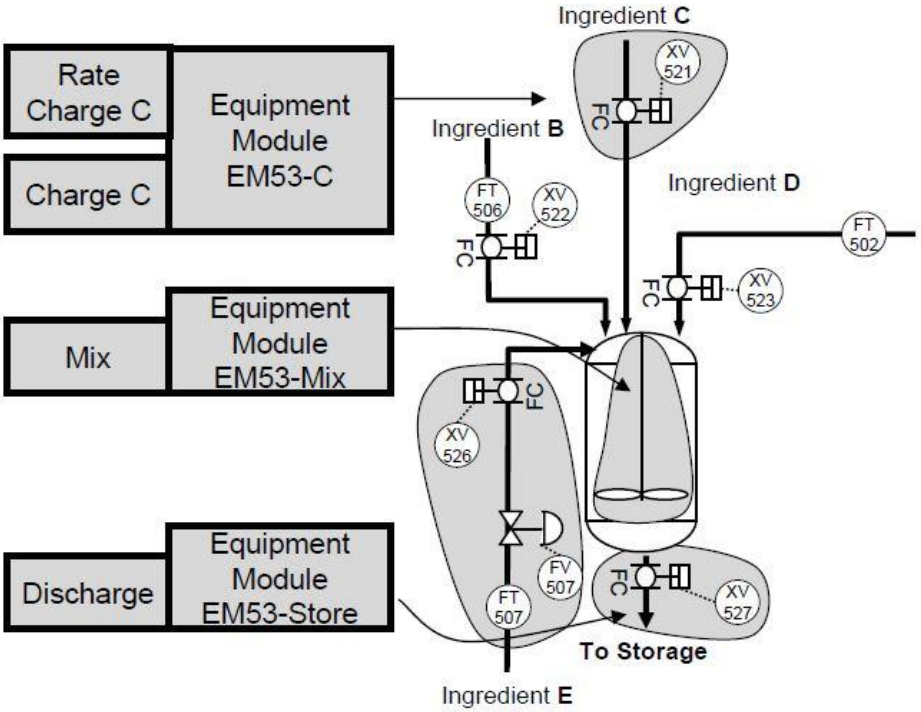


Figura 25 – Exemplo de módulos de equipamento distribuídos num sistema (Brandl, 2007)

Módulo de Controlo

- Executa uma ou mais funções básicas de controlo;
- Faz a interface final com o sistema físico a controlar – "liga" os sensores e actuadores ao sistema de controlo;
- O estado de um módulo de controlo é o próprio estado do equipamento físico controlado;
- Recebe ordens de controlo e de estado;
- Expõe ao módulo de equipamento o seu modo e estado;
- Pode conter outros módulos de controlo;
- Pode ser usado exclusivamente por um módulo de equipamento ou pode ser partilhado.
- Se for partilhado tem que conter lógica de controlo de coordenação, para gerir múltiplos pedidos concorrentes;
- Suporta modo de operação automático e manual.

O Figura 26 demonstra o mesmo sistema já apresentado, mas detalhando os vários módulos de controlo. De notar que existem módulos de controlo compostos por outros módulos de controlo.

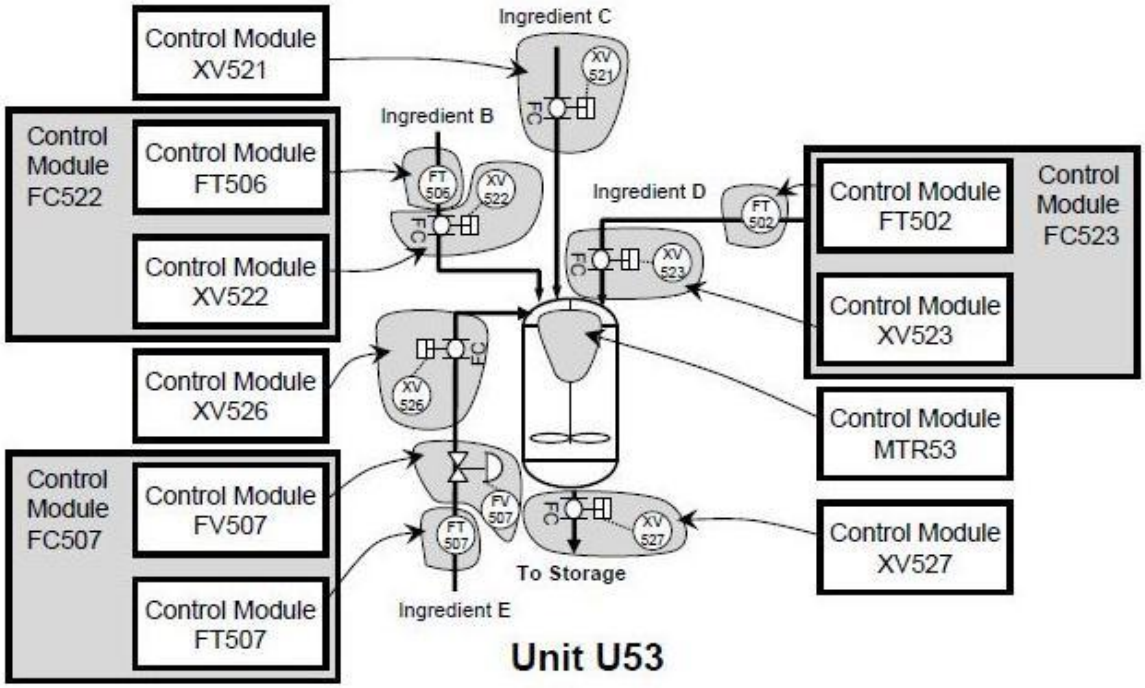


Figura 26 – Exemplo de módulos de controlo aplicados a um sistema (Brandl, 2007)

Aparentemente pode existir uma grande semelhança dos módulos de equipamento com os módulos de controlo, contudo convém lembrar que estes últimos apenas incorporam o controlo de mais baixo nível do equipamento.

Transferência de Material

Um dos pontos críticos num sistema estruturado segundo os padrões de desenho apresentados, é o local onde se dá a transferência de material entre equipamentos. A unidade a montante deve estar pronta a enviar o material e a unidade a jusante deve estar pronta para o receber. A transferência só pode ocorrer quando existir acordo de ambos os lados. Há duas abordagens a este problema.

Para o caso em que existem transferências um-para-um, muitos-para-um ou um-para-muitos aplica-se um padrão simples de transferência (Figura 27). Esse padrão consiste em cada equipamento implementar uma fase de equipamento *transfer-in* em cada unidade a jusante e uma fase de equipamento *transfer-out* em cada unidade a montante. Os equipamentos físicos (válvulas, bombas, motores, actuadores, etc.) usados na transferência podem ser associados à unidade a montante ou a jusante. Em todo o caso esta atribuição deve ser consistente em toda a célula de processo.

O sincronismo da transferência pode ser executado pelo controlo de coordenação ou através de comunicação directa entre módulos de equipamento. Este sincronismo pode ser feito com mensagens do tipo Ready_to_Send e Ready_to_Receive. Se existir comunicação directa entre módulos de equipamentos, é preferível aplicar este padrão apenas a transferências um-para-um, e aplicar o padrão seguidamente descrito a todos os outros tipos de transferência.

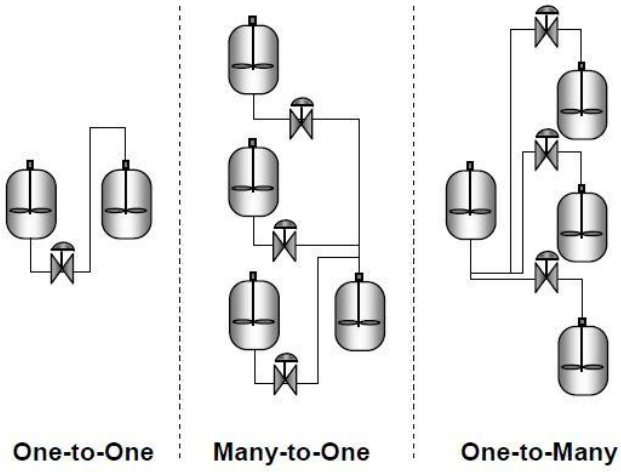


Figura 27 – Tipos de relação entre unidades de equipamento (Brandl, 2007)

Para o caso em que existem transferências muitos-para-muitos (ou para quando se quiser coordenar a transferência com comunicação entre equipamentos) aplica-se um padrão de colector de transferência (*transfer header*) (Figura 28). Esse padrão consiste em criar um módulo de equipamento partilhado para o colector que gere as transferências entre as unidades. Este módulo não é visível para a receita mas é usado pelas fases de equipamento *transfer-in* e *transfer-out*.

O colector não está alocado a nenhuma unidade em específico, mas presta serviços a todas que estão ligadas a si.

O colector deve gerir os pedidos para receber e enviar material e então determinar se há um caminho disponível, alocar esse caminho e dar o sinal às unidades respectivas a montante e a jusante. São necessários sinais de Ready_to_Send e Ready_to_Receive para o sincronismo, cada unidade ligada ao colector deve ainda dar a resposta Clear_to_Send.

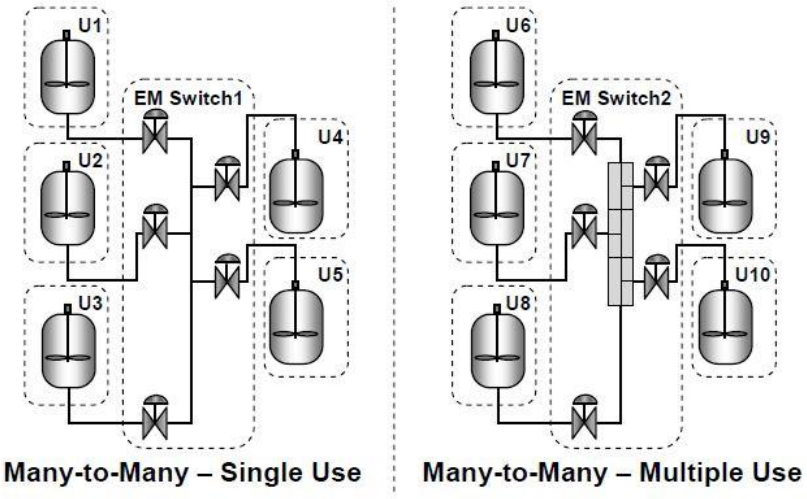


Figura 28 – Tipos de relação entre unidades de equipamento (Brandl, 2007)

As fases de equipamento implementadas devem sempre conter um parâmetro que define o produto a ser transferido, de forma a prevenir a existência de erros e também para permitir a identificação do produto a todo o momento, sem necessidade de assunções ou inferências.

De um modo objectivo, ao depararmo-nos com um sistema, existe a clara dificuldade em definir e distinguir o que é uma unidade ou um módulo de equipamento ou de controlo. Devemo-nos lembrar que se um suposto equipamento necessitar de uma receita para funcionar, então é uma unidade. Caso contrário o próprio equipamento faz parte da unidade. Os módulos de equipamento são as entidades hierarquicamente mais baixas em que a receita pode actuar. O controlo de baixo nível que está intrínseco ao equipamento e não ao processo como um todo, deve estar no módulo de controlo.

3.3.IEC 61499 – Blocos Funcionais

A norma IEC 61499, publicada em 2005, define uma arquitectura para sistemas de controlo e de automação distribuídos e flexíveis, e prevê a utilização de blocos funcionais que podem ser interligados e distribuídos por múltiplos controladores. A sua estrutura é a seguinte:

- Parte 1 – *Architecture*
- Parte 2 – *Software Tool Requirements*
- Parte 3 – *Application Guidelines*
- Parte 4 – *Guidelines for Compliance Profiles*

A primeira parte da norma é a mais relevante para um engenheiro que faz automação, as restantes partes dizem essencialmente respeito aos fabricantes de equipamentos. A arquitectura definida na IEC 61499-1 permite o desenvolvimento de módulos de software reutilizáveis e sua aplicação em sistemas distribuídos que cumpram os requisitos de portabilidade, interoptabilidade e reconfigurabilidade.

O elemento fundamental é o bloco funcional. Pensando em linguagem orientada por objectos, um bloco funcional é uma classe que define o comportamento de algo. Pode ser um equipamento, um actuador, um processo completo, um sub-processo, uma cadeia de medição, etc.

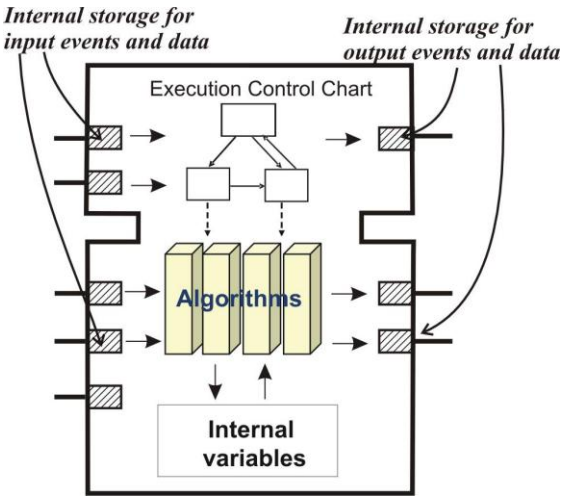


Figura 29 – Visão geral do bloco funcional da IEC 61499 *(http://www.fb61499.com)*

A Figura 29 representa um aspecto possível de um bloco funcional e algumas das suas características.

O bloco funcional mais básico é constituído por:

- ***Event Inputs / Outputs***

  Eventos que são usados para sincronizar o funcionamento dos blocos na aplicação e a execução dos algoritmos dentro do próprio bloco.

  Esta característica faz com que os blocos sejam "*event driven*", ou seja só são activados quando o evento correspondente é disparado, ao contrário da execução cíclica habitual dos PLCs.

  A figura abaixo representa de forma simples a diferença entre uma variável de evento (a vermelho) e uma variável booleana (a azul).

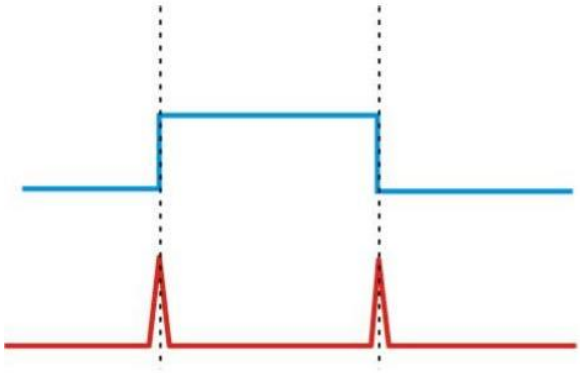


Figura 30 – Diferença entre variáveis booleanas e *events (http://www.fb61499.com)*

- ***Data Inputs / Outputs***

  Variáveis de interface com o exterior do bloco. Podem ser variáveis de qualquer tipo (bool, int, word, etc.). São as variáveis que transportam verdadeiramente a informação relevante entre blocos.

- ***Execution Control Chart (ECC)***

  Máquina de estados interna que despoleta e gere os algoritmos do bloco com base nos eventos recebidos e activa os eventos de saída.

  No caso de blocos funcionais compostos (que contenham outros blocos funcionais), não existe ECC nos blocos hierarquicamente superiores.

  A Figura 31 consiste num exemplo de um ECC. Visualmente parece-se com uma máquina de estados e no fundo o seu funcionamento é muito similar.

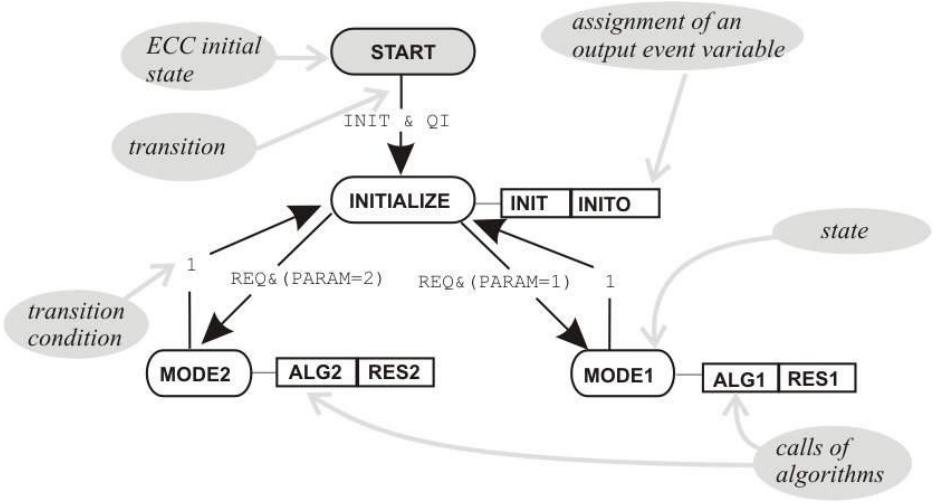


Figura 31 – Exemplo de ECC *(http://www.fb61499.com)*

- **Algoritmo Interno**

  O algoritmo interno só é visível para o próprio bloco e pode ser programado em qualquer linguagem contemplada na IEC 61131-3 ou numa linguagem de alto nível tal como Java. O algoritmo pode ou não ser único

Face ao exposto, o que convém reter é a distinção entre *event inputs/outputs* e *data inputs/outputs.* A Figura 32 indica como estas variáveis podem estar representadas num bloco.

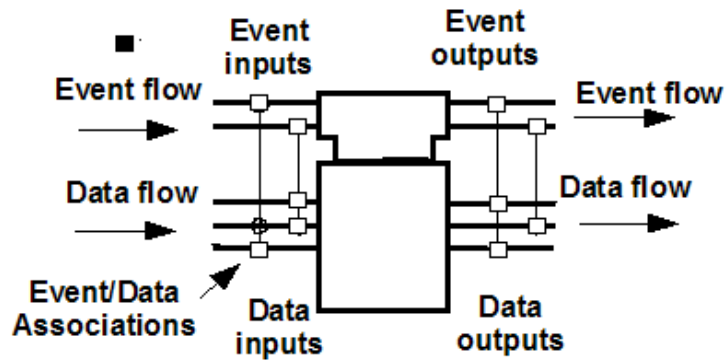


Figura 32 – Bloco funcional e seu fluxo de eventos e informação (IEC, 2005)

Existe um tipo específico de blocos funcionais, os chamados *Service Interface Function Blocks*, que representam de uma forma própria a interface dos blocos com o hardware, interface de utilizador e serviços de comunicação. Estes blocos não irão ser abordados nesta investigação.

Criando instâncias de um bloco funcional é então possível criar aplicações de controlo distribuído modular e flexível. Cada instância só tem acesso às suas próprias variáveis e à informação trocada com outros blocos através de *data inputs / outputs* e *event inputs / outputs*.

Ao juntar blocos funcionais obtém-se uma rede em que o fluxo de informação define a execução dos blocos. Cada bloco funcional de uma dada aplicação pode ser distribuído por diferentes dispositivos ou recursos.

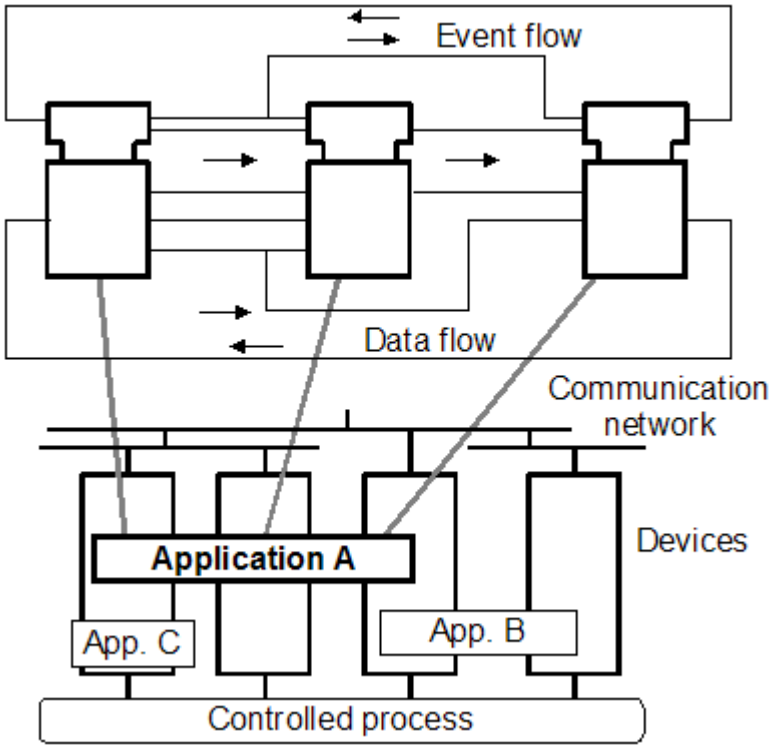


Figura 33 – Aplicações de controlo distribuídas por vários dispositivos (IEC, 2005)

Como se pode observar no exemplo da Figura 33, a aplicação A encontra-se distribuída por vários dispositivos (controladores) e a aplicação em si é constituída por uma rede de blocos funcionais.

A norma também prevê que uma mesma aplicação possa ser distribuída por vários recursos dentro de um dispositivo (várias *tasks* no mesmo PLC, por exemplo) (Figura 34).

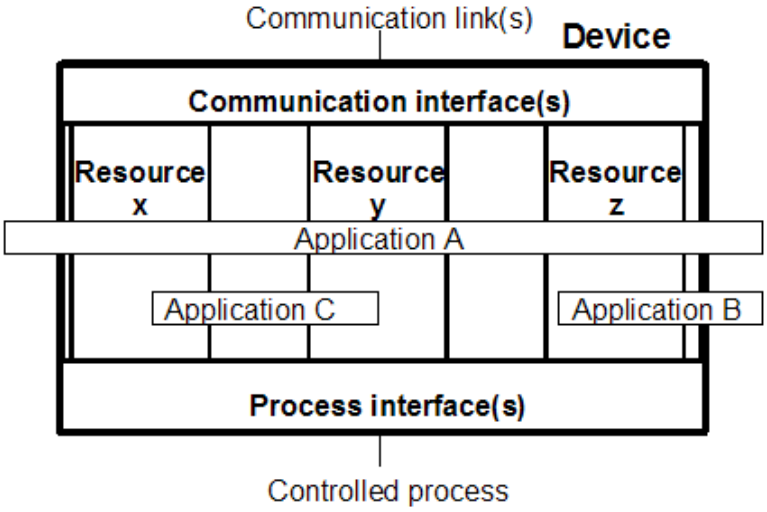


Figura 34 – Distribuição de aplicações por vários recursos do mesmo dispositivo (IEC, 2005)

A troca de informação *event-driven* entre blocos funcionais dá-se da seguinte forma:

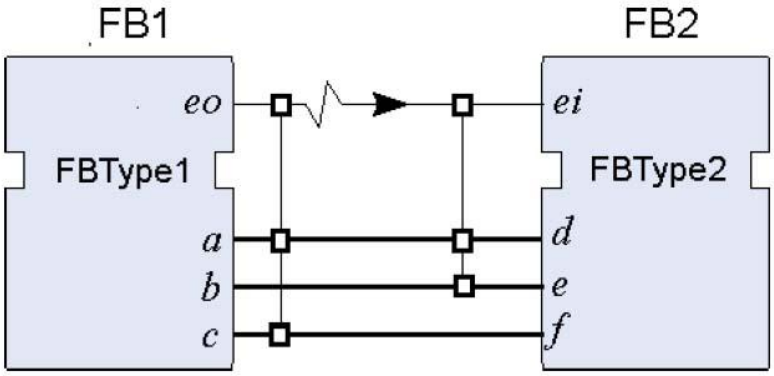


Figura 35 – Troca de informação entre blocos funcionais *(http://www.fb61499.com)*

Os quadrados indicados junto às variáveis de entrada e saída, aludem à forma com que os *data inputs/outputs* estão integrados com os eventos.

O evento eO do FB1 (*function block* 1) está ligado ao evento eI do FB2. Assim que o FB1 emite o evento eO, dispara a execução do FB2. Os valores dos *data inputs* d e e apenas serão actualizados imediatamente antes que a execução do FB2 tenha início, pois estão associados ao event input eI. Apenas o *data input* f será actualizado durante a execução do FB1

Resumindo:

- Eventos despoletam a execução dos blocos funcionais.
- *Data inputs* e *outputs* podem ser transferidos simultaneamente com o evento, ou a todo o instante. Esta nuance prende-se com a necessidade que um bloco pode ter de receber ou transmitir informação de forma contínua, e não apenas associada a um evento.

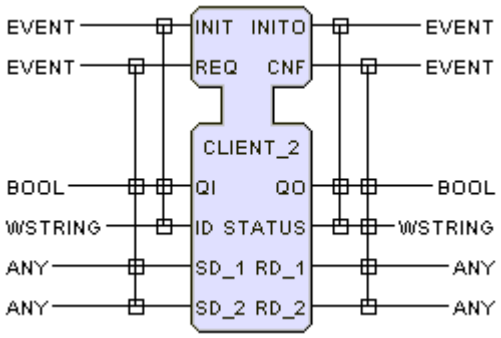


Figura 36 – Exemplo de bloco funcional *(http://www.fb61499.com)*

A sequência de execução de um bloco funcional pode ser descrita com o seguinte exemplo:

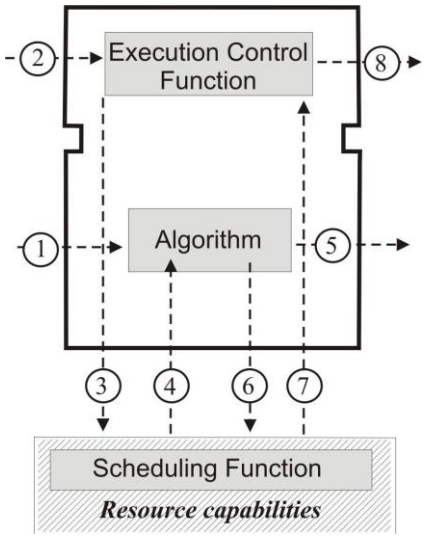


Figura 37 – Sequência de execução de um bloco funcional *(http://www.fb61499.com)*

1. Os *data inputs* associados ao *event input* são disponibilizados;
2. O event input ocorre e o ECC é então despoletado;
3. Atendendo ao ECC em questão, é dada a ordem de execução do algoritmo;
4. O algoritmo é iniciado;
5. O algoritmo termina e estabelece os valores dos *data outputs* associados aos *event outputs;*
6. O ECC é notificado que o algoritmo terminou;
7. É pedido ao ECC que dê sequência à execução;
8. O ECC dá ordem de transmissão dos *event inputs* que tornarão possível a comunicação dos *data outputs*

Estas etapas são coordenadas pelo próprio bloco funcional e não são visíveis para o utilizador.

3.4. Blocos Funcionais IEC 61131-3 *vs* IEC 61499

Entre outras linguagens de programação, a norma IEC 61131-3 apresentada na versão inicial em 1993, descreve também a utilização de blocos funcionais que se definem como uma unidade de organização que, quando executada, produz um ou mais resultados. Instâncias podem ser criadas de um dado bloco funcional e a sua informação é de certo modo encapsulada, apenas sendo praticamente possível aceder às variáveis definidas como *inputs* ou *outputs*.

Os blocos funcionais da IEC 61131-3 são aparentemente em tudo muito semelhantes ao previsto pela IEC 61499, contudo algumas diferenças relevantes surgem. A Figura 38 apresenta os dois tipos de blocos lado a lado para comparação.

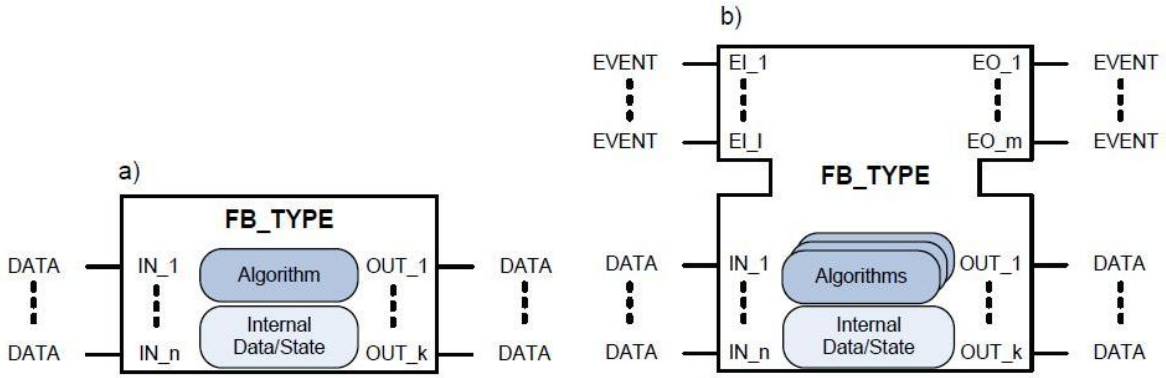


Figura 38 – Representação de bloco funcional segundo a norma IEC 61131 (a) e a norma IEC 61499 (b) *(http://www.fb61499.com)*

Os blocos funcionais segundo a IEC 61131-3 diferem da IEC 61449 nos seguintes aspectos:

- Cada bloco apenas pode conter um algoritmo e não pode conter outros blocos;
- Autorizam a utilização de variáveis globais. Esta característica quebra o encapsulamento e leva a situações em que não se verifica a modularidade e reutilização desejada para uma solução flexível e faz com que uma parte da solução fique dependente de outra;
- O encapsulamento não é total, pois é possível através de um *access path* aceder externamente às variáveis internas;
- Não é possível controlar a ordem de execução dos blocos, pois são executados segundo estão ligados entre si de acordo com as regras da norma, regras essas que deixam alguma margem para interpretação, podendo dessa forma a mesma aplicação comportar-se de forma diferente em controladores diferentes;
- Não existe o conceito de *event inputs / outputs;*
- A performance de um programa de PLC segundo a 61131-3 é normalmente ditado pelo tempo de ciclo (*scan time*). Esse tempo é a soma de todas as instruções acrescido ao tempo para ler as entradas e actualizar as saídas. Na 61499 não existe o conceito de *scan time*. Os algoritmos são executados apenas quando são activados por eventos e o programa executa-os de forma contínua, seguindo a rede de blocos, até que não haja mais blocos para activar.

Em súmula, estas diferenças são consideráveis mas não fazem com que o objectivo desta dissertação seja gorado. Como se verá, a implementação de padrões de desenho através de blocos funcionais segundo a IEC 61131-3 possui paralelismos suficientes para que possam ser tiradas conclusões válidas e generalizantes.

4. Sistemas de Produção Industrial

O objectivo de um sistema de produção industrial é a conversão de matéria-prima em produtos acabados. Essa conversão envolve operações mecânicas ou químicas e é aqui que a automação industrial é imprescindível no panorama actual.

Cada tipo de produto, pressupõe um modelo específico de sistema de produção industrial que garante a máxima eficiência e qualidade de fabrico.

Os sistemas de produção industrial podem ser divididos em três grandes grupos – Discretos, Contínuos e por Partidas – e podem ser caracterizados genericamente da seguinte forma:

1. **Discretos**
    - Produção de quantidades específicas de materiais discretos;
    - O material a produzir é uma entidade bem identificada que se move de A para B, sofrendo diversos processos, transformações ou triagens;
    - Cada processo, transformação ou triagem depende da anterior para garantir a sequência estabelecida e acrescenta valor ou diferenciação ao produto;
    - O transporte de materiais entre processos é uma parte integrante e fulcral do sistema de produção;
    - Podem ou não existir filas de espera para os materiais à espera de um dado processo, transformação ou triagem;
    - Exemplos: Fabrico de componentes automóveis, produção de componentes eléctricos, processos de embalamento, etc.

Figura 39 – Linha de produção de peças metálicas para o ramo automóvel *(http://www.automation.com)*

Figura 40 – Conveyor de rolos transportando caixas de cartão *(http://www.wh1.com)*

Os sistemas discretos possuem uma parte inalienável que é o transporte de mercadorias. O transporte pode dar-se antes e após (e até durante) qualquer processamento ou trabalho realizado sobre o material. Quando o material a transportar é uniforme no que diz respeito à sua forma e tamanho, o equipamento tradicional de transporte é o ***conveyor*.** Existem muitos tipos de conveyors: de rolos, de correntes, de tela, gravíticos, rotativos, de sem-fim, etc. Todos eles seguem os mesmos princípios e objectivos:

- Transportar eficaz e eficientemente material desde o seu ponto de partida até ao ponto de chegada;
- Nos pontos de partida e chegada, o material é trocado com outro conveyor, outro equipamento ou com um operador manual;
- O material pode ou não sofrer alterações, acções, processamentos ou identificações no decorrer do transporte;
- Além do material é frequente transportar a informação relativa ao mesmo.

2. Contínuos

- Produção de material de forma contínua. (muitas vezes 24/24h);
- Não existe uma quantidade discreta que possa ser claramente identificável no material produzido;
- Os materiais produzidos são normalmente medidos em massa, volume, ou comprimento;
- Todos os processos, transformações ou triagens são executados no material como um todo, sem clara delimitação do seu início e fim;
- Não existe rigidez no conceito de transporte de material, pois este está sempre em movimento, independentemente do processo;
- Existe uma fase de arranque e paragem da instalação em que se assume que o material produzido não é aproveitável. Com a excepção dessa situação transitória, não há interrupções nem tempos de espera na produção;
- Com a mesma instalação é possível a todo o momento variar o material produzido, actuando sobre as variáveis de processo - a flexibilidade é, de certo modo, inerente ao próprio sistema;

- Exemplos: Refinaria, produção de electricidade, produção de pasta de papel, produção de papel, etc.

Figura 41 – Vista panorâmica de uma refinaria petrolífera *(http://www.elpower.net)*

O seguinte tipo de sistema de produção industrial – por partidas - é considerado por alguns autores, um sub-tipo dos sistemas discretos. Contudo, por possuir também características dos sistemas contínuos, justifica-se, a meu ver, que seja um tipo distinto de processo.

3. **Por partidas (*Batch*)**
   - Produção cíclica de material;
   - Produção de material homogéneo em quantidades finitas em cada ciclo;
   - Cada partida de material é produzida submetendo quantidades específicas de matérias-primas a uma dada sequência de processos, transformações ou misturas;
   - Uma nova partida de material só pode ser produzida após a partida anterior ter sido concluída – não se adiciona nem remove material durante o processamento;
   - Num dado equipamento só se produz uma partida de cada vez;
   - O produto obtido numa partida pode ser final ou pode ser o material base para uma nova partida;
   - Num sistema por partidas complexo, o material final passa por uma sequência de processamentos, pelo que o sistema deve ser visto de forma holónica como um conjunto de pequenos sistemas por partidas autónomos. A separação entre subsistemas é por isso vital para o garantir da flexibilidade da solução de automação;
   - O controlo é iminentemente sequencial e abrange características do controlo de sistemas discretos e contínuos;
   - Normalmente existem filas de espera para o material antes de cada processo de batch;

- Há uma necessidade clara de transporte das partidas entre processos – através de bombagem, diferenças de pressão ou gravidade.
- Com a mesma instalação é possível obterem-se diferentes produtos, variando as variáveis de processo;
- Estas variáveis de processo podem e são normalmente definidas numa receita, que define as regras de fabrico e o comportamento dos equipamentos;
- Exemplos: Produção de medicamentos, fabrico de bebidas, produção de tintas, etc.

Figura 42 – Sistema de batching de granulados para a indústria farmacêutica *(http://www.weighfill.com)*

Num outro nível de abstracção, os sistemas de produção podem ainda ser classificados quanto ao seu funcionamento como **sistemas *pull*** ou **sistemas *push***.

A diferença fundamental entre sistemas pull e push é a forma como é despoletada a produção de material.

- Num sistema push as ordens de produção são programadas a um nível superior – planeamento de produção - e a informação de produção é transportada no sentido do fluxo de material – para jusante.
- Num sistema pull as ordens são autorizadas a partir de um pedido feito a jusante, ou seja a informação não segue o sentido do fluxo de material.
- Os sistemas push podem ser vistos como sistemas *make-to-order*. Já os sistemas pull podem ser vistos como sistemas *make-to-stock*.

A Figura 43 ilustra este princípio. A verde mostra-se o fluxo da informação de produção num sistema push, e a vermelho num sistema pull.

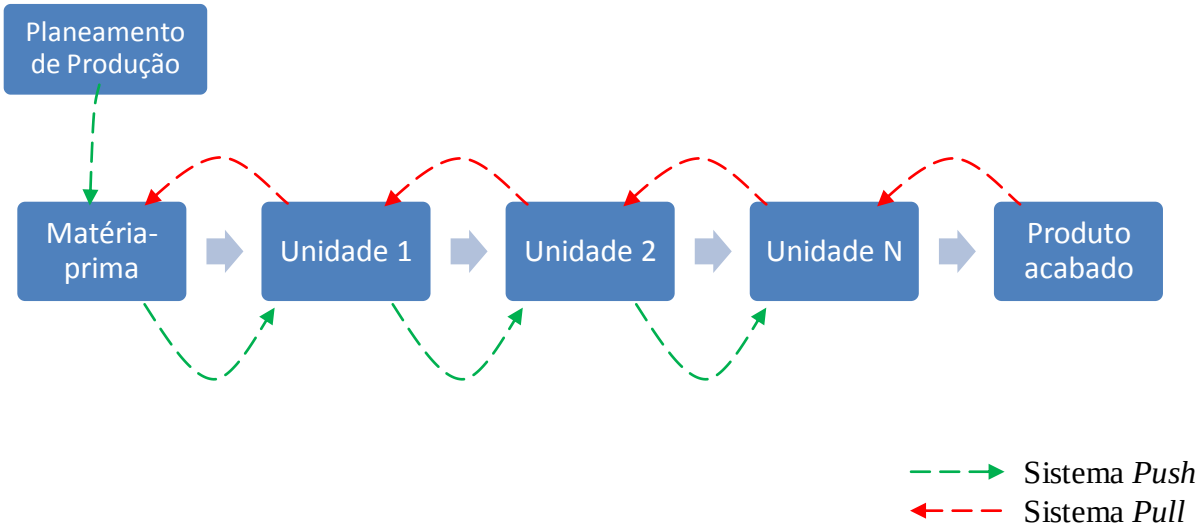


Figura 43 – Definição de sistemas push e pull

Se este conceito for transposto para o nível mais baixo da coordenação entre unidades de produção, a comunicação entre dois equipamentos pode ser vista também como push ou pull. Como exemplo, imaginando-se dois conveyors em série, num sistema push o primeiro conveyor fornece material a jusante assim que tenha material para fornecer, num sistema pull o primeiro conveyor fornece material assim que tenha um pedido ou autorização do conveyor a jusante para o fazer.

Ficam assim caracterizados a vários níveis os diversos tipos de processos industriais. O controlo por PLC para cada tipo de sistema de produção possui um tipo próprio de solução de automação geralmente aplicada. Adaptar essas soluções ao princípio de flexibilidade e modularidade tem implicações ao nível da sua estruturação.

A primeira tarefa ao buscar uma solução de automação flexível, passa por definir uma subdivisão do sistema em módulos, interligáveis e substituíveis. Nos sistemas discretos e por partidas isso é relativamente fácil de atingir. Nos sistemas contínuos o facto de tal não ser trivial dificulta grandemente a tarefa de flexibilização da solução de controlo. Adicionalmente, um sistema contínuo, por definição, possui desde logo alguma flexibilidade, pelo que esta dissertação foca-se em sistemas discretos – *Sorting* de caixas – e por partidas – *Batching* de tintas.

5. Investigação Experimental – Sistema Sorting

Este capítulo, mais do que uma descrição exaustiva de tudo o que foi investigado e experimentado, pretende ser o partilhar do fio de pensamento, descobertas e inferências que me levaram às conclusões finais deste trabalho.

A investigação realizada, incidiu primeiramente num sistema típico de sorting presente no software de sistemas virtuais ITS PLC. Foram experimentadas várias abordagens na busca de uma ou mais soluções que provem ser flexíveis, podendo ser facilmente adaptadas a outros sistemas iguais no seu fundamento mas diferentes na sua configuração.

Seguindo as várias etapas para a criação de um padrão de desenho no âmbito do controlo flexível apresentadas no capítulo 2.1, os primeiros passos são então a análise do sistema e o identificar de equipamentos e processos.

5.1. Análise do Sistema e Identificação de Equipamentos e Processos

Um sistema de sorting genérico é basicamente um sistema que transporta e separa material bem definido sob uma dada ordem. O sistema Sorting de separação de caixas do ITS PLC é composto por 5 conveyors (Figura 44).

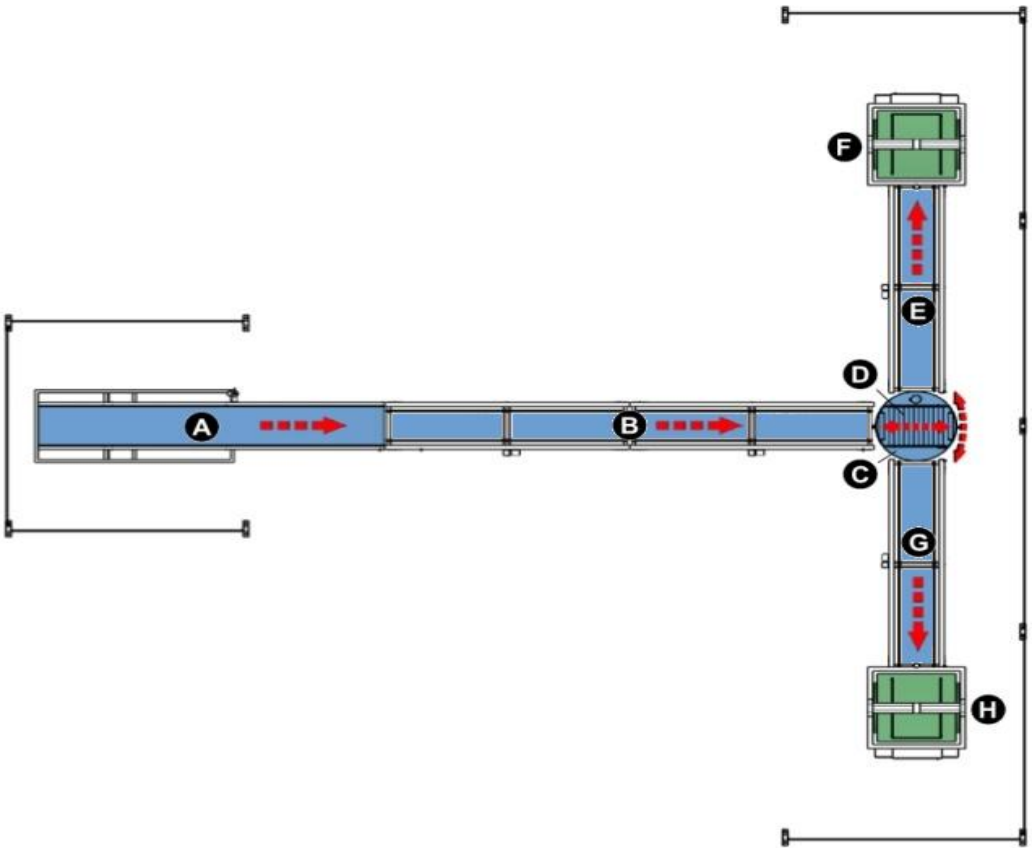


Figura 44 – Planta do sistema Sorting e sua constituição *(http://www.realgames.pt)*

A – Conveyor alimentador

B – Conveyor transportador

C – Conveyor rotativo (também tem movimento linear D para carregar e descarregar caixas)

E, G – Conveyors transportadores de saída

Notar que os elevadores F e H são automáticos e não são controlados pelo PLC.

Figura 45 – Caixas transportadas pelo sistema

As caixas de duas alturas diferentes, como ilustrado na Figura 45, são transportadas de A até F ou H, sendo medidas em altura (sensores 1 e 2 – Figura 46) ao entrarem no conveyor B e separadas, mediante uma lógica definida pelo controlador, no conveyor rotativo C – Figura 47

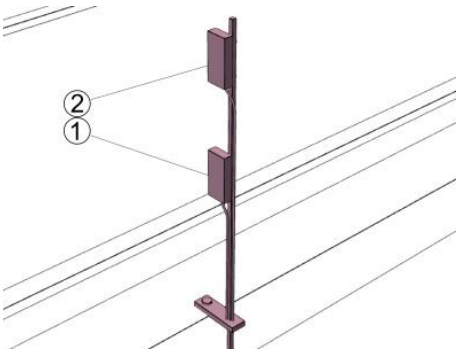


Figura 46 – Sensores do tapete B

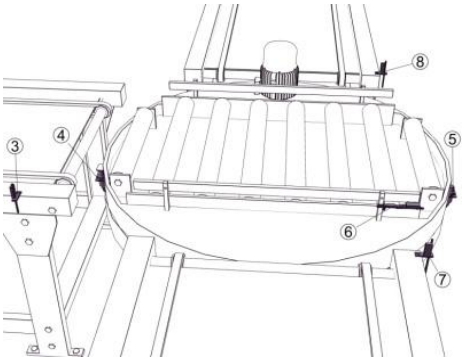


Figura 47 – Mesa rotativa

Cada conveyor, possui grosso modo um sensor à entrada e outro à saída, que detecta a respectiva entrada e saída de caixas. As duas excepções são o conveyor alimentador que não possui sensor de entrada – caixas são alimentadas automaticamente com uma cadência pré-definida pelo software, e o conveyor rotativo que só possui sensor de saída (sensor 6) quando a caixa é encaminhada para um dos lados e não possui sensor de entrada.

O conveyor linear B não possui propriamente um sensor de entrada, mas os sensores que fazem a detecção da altura das caixas podem executar essa função, ou então pode ser usado o sensor de saída do conveyor alimentador precedente. Mais à frente será analisada qual a melhor opção.

Cada conveyor efectua o seu trabalho de transportar a caixa desde a sua entrada até à saída. O conveyor B adiciona a isso o facto de efectuar a verificação de altura das caixas. O conveyor rotativo efectua a verdadeira acção de sorting.

Para uma descrição mais detalhada do sistema, pode ser consultada a informação em anexo ou visitado o site www.realgames.pt. A secção da galeria de vídeos é especialmente elucidativa.

Generalizando, todo e qualquer sistema de sorting pode ser visto como um sistema transportador de material desde um ou mais pontos de alimentação de material até um ou mais pontos de recepção, passando obrigatoriamente por um ou mais pontos de aferição da qualidade ou tipo de material e por um ou mais pontos de separação, seguindo uma dada ordem. Esta análise generalizante provar-se-á

relevante aquando da etapa de alargamento do espectro da solução na busca de um padrão de desenho flexível.

Importa referir que as seguintes três etapas da busca de uma solução flexível são as primeiras ocorrências do processo iterativo que é esta experimentação. Para quem lê, tal como a mim provou ser, será mais fácil entender o desfecho desta busca se compreender e sentir os problemas do caminho até ele, e é minha convicção que, não raras vezes, nesse caminho levantam-se mais questões valiosas do que com a conclusão final.

5.2. Subdivisão Inicial do Sistema em Módulos

Ao passar-se à fase seguinte – subdivisão do sistema em módulos - entra-se na primeira etapa que envolve esforço de programação e estruturação. Esta etapa revelar-se-á iterativa, na medida em que não é imediatamente aparente a forma óptima de subdividir um sistema em módulos.

Logo à partida, e uma vez que todo o sistema é constituído apenas por conveyors de vários tipos, surge a ideia de que para alcançar uma solução flexível é necessário encontrar até que ponto se pode abstrair um conveyor. O módulo fundamental para este sistema é o conveyor.

Olhando para o sistema, pode-se em tese enumerar 3 hipóteses iniciais de subdivisão. A Tabela 2 apresenta as várias hipóteses iniciais idealizadas. Cada FB (*Function Block*) diferente é uma classe definida de forma específica e que pode ser instanciada as vezes que forem necessárias.

Tabela 2 – Hipóteses iniciais de subdivisão do sistema tendo o conta os vários conveyors

Conveyor	**Hipóteses Iniciais de Subdivisão do Sistema**		
	1	2	3
A – Alimentador (0)	FB1	FB1	FB1
B – Transportador (1)	FB2	FB1	FB1
C – Rotativo (Rotary)	FB3	FB2	FB1
E – Transportador de Saída (2)	FB2	FB1	FB1
G – Transportador de Saída (3)	FB2	FB1	FB1

Entre parêntesis está indicado como cada conveyor será chamado no código do programa.

As hipóteses apresentadas (de 1 até 3) seguem, à partida, uma ordem crescente de flexibilidade. Importa encontrar qual o limite a partir do qual essa flexibilidade traz problemas de dificuldade de configuração inicial da solução de controlo, ou não promove a reutilização eficiente de código.

Sendo esta etapa iterativa, para efeito de justificação de raciocínio e após a análise das soluções iniciais de controlo, será revisitada a subdivisão, em módulos que eventualmente se revelem mais flexíveis.

5.3. Solução Inicial de Controlo

Tendo por base os fundamentos teóricos apontados, a abstracção de um conveyor nos seus vários níveis é conseguida com a aplicação de blocos funcionais que encapsulam o seu funcionamento e apenas comunicam ao exterior a informação relevante ao seu correcto funcionamento quando pensando no sistema como um todo. Um bloco funcional que encapsule o funcionamento do conveyor deve conter toda a lógica e informação necessária para que um conveyor execute a sua função fundamental – transportar caixas desde a sua entrada até à sua saída.

Um bloco funcional segundo a norma IEC 61131-3 é constituído por variáveis de entrada (VAR_INPUT), variáveis de saída (VAR_OUTPUT) e variáveis internas (VAR).

O exemplo seguinte ilustra a forma de declarar as variáveis de entrada e saída, usando a comum linguagem de texto estruturado (também contemplada na IEC 61131-3).

```
VAR_INPUT
(*Inputs Físicos*)
    INPUT_IN: BOOL; (*Sensor de Entrada*)
    INPUT_EXIT:BOOL; (*Sensor de Saída*)
END_VAR
VAR_OUTPUT
(*Outputs Físicos*)
    OUTPUT: BOOL; (*Comando do motor do conveyor*)
END_VAR
```

Dentro do bloco funcional em si, está contido o código que garante o seu funcionamento. Mais à frente este código será analisado e serão debatidas várias abordagens e particularidades.

Um bloco funcional só se torna activo se devidamente instanciado. Os exemplos seguintes indicam a forma de invocar e configurar uma instância - Conveyor1 - do bloco funcional CONVEYOR.

```
PROGRAM Sorting
VAR
    Conveyor1 : CONVEYOR;
END_VAR
```

```
(*Conveyor Linear*)
Conveyor1 (INPUT_IN := In_0);
Conveyor1 (INPUT_EXIT := In_3);
Out_1 := Conveyor1.OUTPUT;
```

O bloco funcional ilustrado neste exemplo permite o controlo de um conveyor isoladamente, sem ligação a nenhum outro equipamento controlado. Este conveyor possui um sensor de entrada (INPUT_IN) e um de saída (INPUT_EXIT) e é accionado através de uma única variável binária (OUTPUT). As suas variáveis internas (indicadas no GRAFCET da Figura 48) controlam a evolução sequencial dos estados do conveyor. Este bloco presume que apenas uma caixa é transportada de cada vez.

Partindo deste bloco muito básico, é possível começar a estruturar uma solução de controlo para o sistema, mas antes de mais, importa primeiramente abordar que tipo de integração horizontal e vertical podem ter os conveyors.

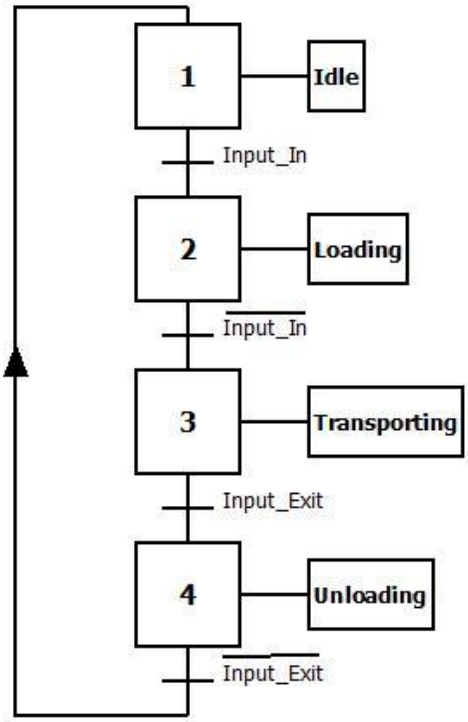


Figura 48 – GRAFCET genérico de um conveyor

Integração Horizontal de Conveyors

Com a integração horizontal de conveyors define-se a forma de interligação entre os transportadores e que informação é necessária trocar para a correcta implementação da solução de controlo.

Com base nos fundamentos teóricos e nos bons princípios do ramo, o mínimo que dois conveyors necessitam de comunicar entre si, de forma a funcionar de formar segura e eficiente, é uma variável booleana que indique o seu estado a todo o momento. O bloco funcional de um conveyor tem então que possuir uma saída que comunique ao conveyor a montante quando está ocupado e não pode receber mais caixas (SEND_BUSY). Da mesma forma, deve possuir uma entrada que permita receber esta informação e integra-la no seu código encapsulado de controlo (RECEIVE_BUSY).

```
SEND_BUSY:BOOL;(*Variável que indica ao conveyor antecedente que não se encontra em posição de receber mais caixas*)
RECEIVE_BUSY:BOOL; (*Variável de entrada que indica que o próximo conveyor não se encontra em posição de receber mais caixas*)
```

Além das variáveis físicas de entrada e saída (INPUT_IN, INPUT_EXIT e OUTPUT), existem mais duas variáveis booleanas que são apenas memórias no controlador. Convém realçar que estas variáveis são todas locais, pelo que uma instância de um conveyor só tem acesso às variáveis de entrada ou saída de outra se assim forem chamadas no programa. De seguida é apresentada a forma mais simples de o fazer.

```
Conveyor0 (RECEIVE_BUSY:= Conveyor1.SEND_BUSY);
```

O conveyor 0 passa a ter a variável de entrada RECEIVE_BUSY associada a todo o momento à variável de saída do conveyor 1 SEND_BUSY.

Cada instância só publica as variáveis que são definidas como entradas ou saídas do bloco. Esta integração horizontal é então fundamental para que o sincronismo entre etapas de diferentes blocos funcionais seja possível. Mais informações trocadas revelar-se-ão indispensáveis em futuras iterações.

Integração Vertical de Conveyors

Com a integração vertical de conveyors define-se o seu comportamento no sistema como um todo. Define-se também as características particulares de cada conveyor, ou seja o que distingue cada instância para além da atribuição de entradas e saídas físicas.

A variável de entrada mais relevante é um inteiro MAX_BOX que defina o máximo número de caixas presentes simultaneamente num dado conveyor, variável essa que despoleta o envio da *flag* SEND_BUSY.

```
MAX_BOX:INT; (*Máximo de caixas presentes simultaneamente no conveyor*)
```

O seu valor pode ser definido no programa principal (neste caso toma o valor 2) ou pode ser definido dinamicamente através de um HMI ou um SCADA, por exemplo.

```
Conveyor1 (MAX_BOX:= 2);
```

Provavelmente será necessário criar uma variável de entrada que diga ao conveyor rotativo para onde deve descarregar cada caixa, mas para simplificação esta será para já omitida.

A variável MAX_BOX pode e deve ser vista como a primeira abordagem à receita que dita o funcionamento do sistema. A relevância desta variável explica-se, por exemplo, pela eventualidade de um conveyor necessitar de limitar a quantidade de caixas a transportar por questões de potência dos motores de accionamento. Este exemplo particular serve para relembrar que só com soluções gerais para problemas concretos se conseguirá uma abordagem verdadeiramente flexível.

Definição de Abordagens

Baseado na minha experiência, antes de partir para o controlo de todo o sistema é um bom princípio controlar apenas uma parte do mesmo, para assim a aplicabilidade da subdivisão ser mais óbvia e para serem analisadas criteriosamente as importantes relações horizontais e verticais.

A partir daí serão feitas abordagens crescentemente abrangentes, na esperança de cada uma trazer para cima da mesa novas questões e problemas que obriguem repensar um ou outro ponto do código elaborado.

A Tabela 3 e a Figura 49 resumem as abordagens seguidas:

Tabela 3 – Resumo das várias abordagens e hipóteses de subdivisão para o controlo do sistema Sorting

Objectivo	Hipóteses de subdivisão aplicáveis
5.3.1 – Controlar conveyor alimentador e o conveyor B	1 e 3
5.3.2 – Controlar conveyor alimentador, conveyor B e o conveyor rotativo, efectuando o devido encaminhamento das caixas apenas para um dos lados	1, 2 e 3
5.3.3 – Controlar conveyor alimentador, conveyor B, conveyor rotativo e o conveyor E	1, 2 e 3
5.3.4 – Controlar todo o sistema	1, 2 e 3

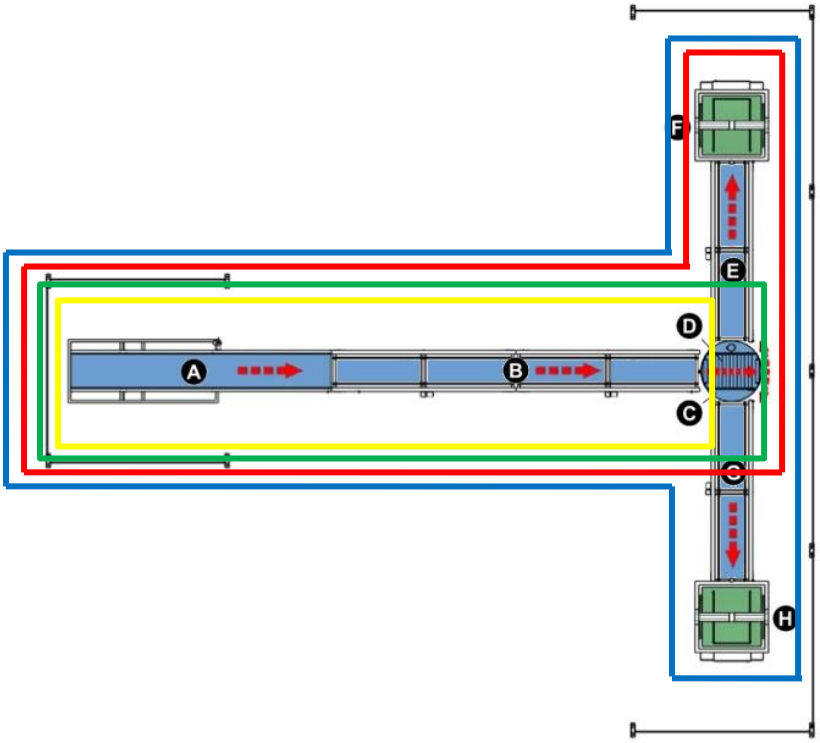


Figura 49 – Ilustração dos limites das abordagens para o controlo do sistema

Com a descrição de cada abordagem, serão tecidas algumas considerações sobre características marcantes, pontos positivos e negativos. Isso permitirá balizar progressivamente as soluções até se chegar à solução final.

5.3.1. Controlo do Conveyor Alimentador e do Conveyor B

Esquecendo o restante sistema, foi estruturada e implementada uma solução que permite o controlo dos dois primeiros conveyors do sistema. No que respeita à subdivisão em blocos funcionais, foram testadas as hipóteses 1 e 3, ou seja, usar um bloco funcional diferente para cada conveyor ou um bloco funcional igual para as duas instâncias.

A primeira hipótese cedo se revelou desnecessária, pois um conveyor alimentador não difere fundamentalmente de um outro qualquer conveyor com as mesmas propriedades físicas. A única diferença reside, neste caso, na inexistência de sensor de entrada, ou seja presume-se que a montante do conveyor alimentador não existe nenhum equipamento passível de ser controlado. As caixas são introduzidas de forma constante no sistema e com uma cadência definida além controlo.

Por questões de entendimento e organização, este exemplo será apresentado no corpo do texto da dissertação, sendo que os restantes exemplos serão maioritariamente expostos em anexo.

As variáveis definidas para os conveyors nesta abordagem são as seguintes.

```
FUNCTION_BLOCK CONVEYOR
VAR_INPUT
(*Inputs Físicos*)
    INPUT_IN: BOOL; (*Sensor de Entrada*)
    INPUT_IN_HIGH:BOOL; (*Sensor de Entrada - Caixa Alta*)
    INPUT_EXIT:BOOL; (*Sensor de Saída*)

(*Inputs Software*)
    MAX_BOX:INT; (*Máximo de caixas presentes simultaneamente no conveyor*)
    RECEIVE_BUSY:BOOL; (*Variável de entrada que indica que o próximo conveyor
                         não se encontra em posição de receber mais caixas*)
END_VAR
VAR_OUTPUT
(*Outputs Físicos*)
    OUTPUT: BOOL; (*Comando do motor do conveyor*)

(*Outputs Software*)
    SEND_BUSY:BOOL;(*Variável que indica ao conveyor antecedente que não se
                    encontra em posição de receber mais caixas*)
END_VAR
VAR
(*Triggers*)
    R_In    : R_TRIG; (*Rising trigger do sensor de entrada*)
    F_In    : F_TRIG; (*Falling trigger do sensor de saída*)
    R_Out : R_TRIG; (*Rising trigger do sensor de saída*)
    F_Out  : F_TRIG;(*Falling trigger do sensor de saída*)
    Counter :CTUD; (*Contador do Max_Box*)

(*Estados Internos*)
    Idle:BOOL:=TRUE; (*Parado e pronto a iniciar marcha*)
    Loading:BOOL :=FALSE; (*A carregar uma caixa*)
    Transporting:BOOL:=FALSE; (*Atransportar uma caixa*)
    Unloading:BOOL:=FALSE; (*A descarregar uma caixa*)
END_VAR
```

Os seguintes diagramas de estados explicam a lógica implementada dentro do bloco.

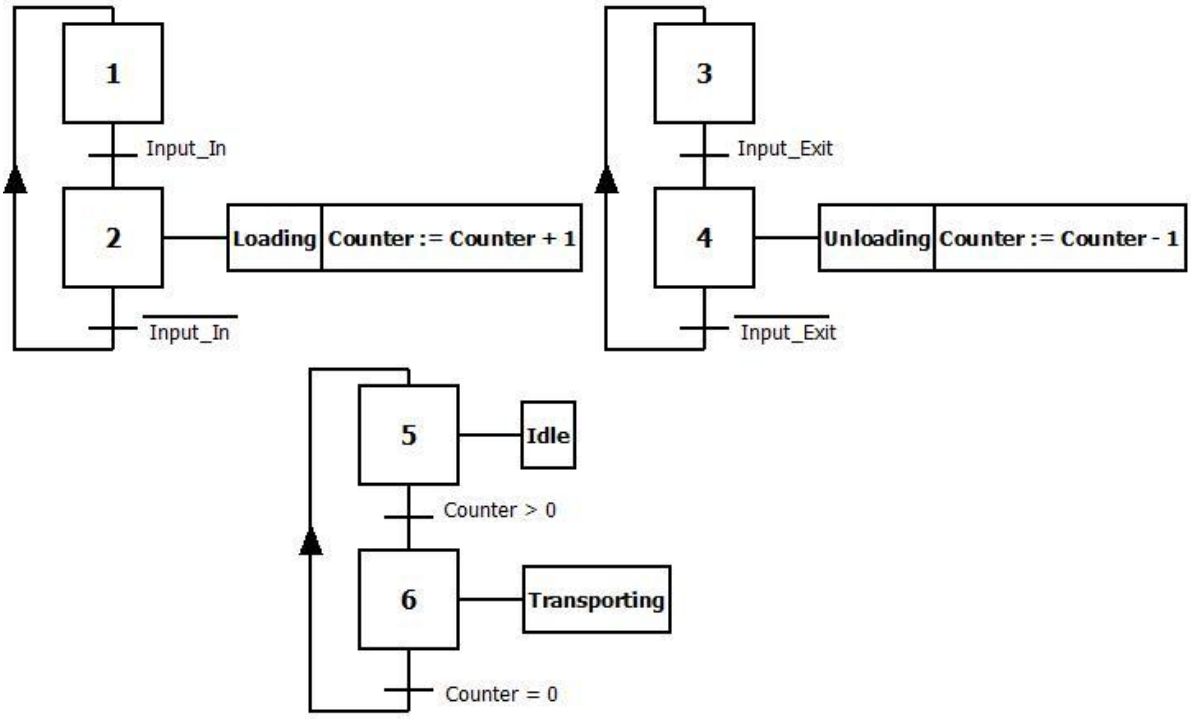


Figura 50 – GRAFCET genérico de conveyor com lógica de funcionamento contínuo

O código é apresentado de seguida.

```
(* Inicialização de Edge Detections e Counters*)

R_In (CLK := INPUT_IN);
F_Out (CLK := INPUT_EXIT);
F_In (CLK := INPUT_IN);
R_Out(CLK := INPUT_EXIT);

(*Contador de Max_Box*)

Counter(CU := F_In.Q, CD:=F_Out.Q, Reset := In_14, PV := MAX_BOX);

(*Implementação de Max_Box*)

IF Counter.QU THEN
    SEND_BUSY := TRUE;
END_IF;

IF NOT Counter.QU THEN
    SEND_BUSY :=FALSE;
END_IF;

(*Controlo de Estados*)

IF Idle AND INPUT_IN THEN
    Idle:=FALSE;
    Loading:=TRUE;
END_IF;

IF Loading AND F_In.Q THEN
    Loading:=FALSE;
    Transporting := TRUE;
END_IF;

IF Transporting AND R_Out.Q THEN
    Transporting := FALSE;
    Unloading :=TRUE;
END_IF;

IF Unloading AND F_Out.Q THEN
    Unloading := FALSE;
    IF NOT  Counter.QU THEN
        Idle:=TRUE;
    ELSE
        Transporting := TRUE;
    END_IF;
END_IF;

(*Afectação das saídas*)

OUTPUT:=((Loading OR Transporting OR Unloading) AND NOT ((RECEIVE_BUSY AND INPUT_EXIT)));
```

Com esta abordagem aplicaram-se as ideias de integração já apresentadas e desenvolveu-se uma solução com base sólida na solução apresentada em (Magalhães, 2010). As restantes abordagens foram baseadas igualmente nesta referência.

O bloco funcional resultante pode-se representar da seguinte forma:

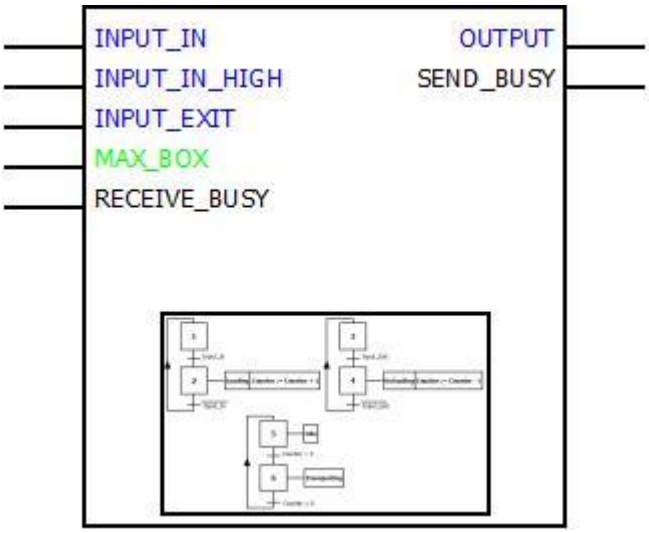


Figura 51 – Bloco funcional de conveyor linear

Foi usada a seguinte convenção de representação. A azul estão representadas as variáveis associadas a componentes físicos (sensores e actuadores). A verde representa-se as variáveis de configuração e/ou integração vertical. A negro indicam-se as variáveis de integração horizontal, ou seja as que serão necessárias interligar com outros blocos para o correcto funcionamento do sistema. O GRAFCET serve para relembrar a lógica sequencial que o bloco implementa. Todos os blocos representados no decorrer desta dissertação seguirão este princípio gráfico.

A partir da classe CONVEYOR as instâncias para cada um dos dois conveyors são criadas da seguinte forma:

```
(*Conveyor Linear de Entrada*)
Conveyor0 (INPUT_IN := TRUE);
Conveyor0 (INPUT_EXIT := In_0);
Conveyor0 (RECEIVE_BUSY:= Conveyor1.SEND_BUSY);
Out_0 := Conveyor0.OUTPUT;
Conveyor0 (MAX_BOX:= 2);

(*Conveyor Linear de Identificação de Caixas*)
Conveyor1 (INPUT_IN := In_0);
Conveyor1 (INPUT_IN_HIGH:= In_2);
Conveyor1 (INPUT_EXIT := In_3);
Conveyor1 (RECEIVE_BUSY:=ConveyorRotary.SEND_BUSY);
Out_1 := Conveyor1.OUTPUT;
Conveyor1 (MAX_BOX:= 2);
```

Cada um dos *tags* dos sensores e actuadores foram atribuídos tendo em conta a tabela de pontos de I/O do Sorting presente em anexo.

Reparar que na falta de existir um sensor de entrada no Conveyor0, o seu INPUT_IN é sempre mantido ligado. A variável de entrada INPUT_IN do Conveyor1 foi configurada como sendo o sensor de saída do conveyor alimentador, isto deve-se a condicionantes físicas do sistema e porque mais adiante será necessário abstrair os sensores 1 e 2 para uma aplicação específica.

Visualizando os blocos funcionais de forma gráfica é mais fácil entender a sua ligação e configuração.

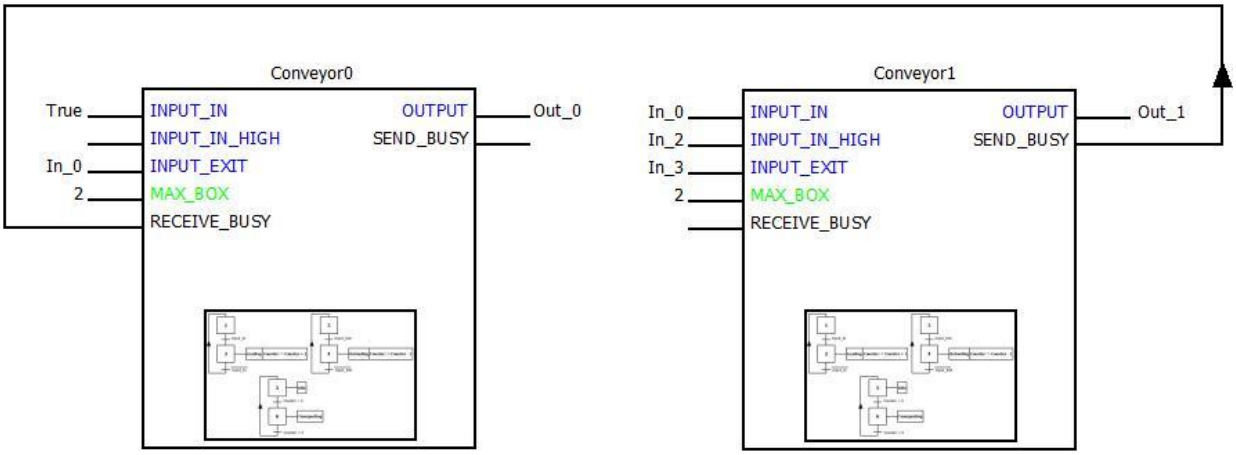


Figura 52 – Diagrama de blocos da solução de controlo dos dois primeiros conveyors

Como se pode observar, a única ligação entre os dois conveyors é a relação de disponibilidade do conveyor a jusante receber caixas do conveyor a montante. Internamente no bloco, esta disponibilidade é gerida por um contador que é incrementado ou decrementado sempre que entra ou sai uma caixa respectivamente, e pelo mascarar da variável de saída OUTPUT.

De notar que o segundo conveyor possui um sensor que permite detectar a altura da caixa, não sendo ainda usado nesta abordagem (se bem que o bloco funcional já prevê essa possibilidade).

Com esta primeira abordagem começa-se a entender o valor da reutilização de código que o encapsulamento em blocos funcionais permite.

5.3.2. Controlo do Conveyor Alimentador, Conveyor B e o Conveyor Rotativo (sem separação)

Nesta abordagem, adiciona-se à anterior o controlo do conveyor rotativo, se bem que ainda sem separação por alturas, mas sim apenas o encaminhamento de todas as caixas para um dos conveyors de saída (sem controlar ainda este).

Pelas mesmas razões da abordagem anterior, a hipótese 1 de subdivisão em módulos foi descartada. Foram consideradas então as hipóteses 2 e 3 – blocos funcionais diferentes para os conveyors lineares e rotativos ou blocos funcionais idênticos para todos os conveyors.

Hipótese 2

Pretende-se controlar três conveyors diferentes com dois tipos de blocos funcionais.

Os conveyors lineares podem ser controlados por instâncias de um bloco funcional idêntico ao apresentado em 5.3.1.

O conveyor rotativo apresenta uma clara diferença em relação aos restantes conveyors no sistema, por possuir mais do que um movimento – linear e rotativo - e em ambos os sentidos, pelo que a hipótese de existir um bloco funcional específico para o conveyor rotativo justifica-se.

A sua determinação não apresenta problemas de maior, com base no bloco desenvolvido para o conveyor linear, é apenas necessário definir os seus *inputs* e *outputs* específicos (INPUT_ROTARY_IN, INPUT_ROTARY_EXIT e OUTPUT_ROTARY). A sequenciação de etapas é também muito semelhante ao conveyor linear e a integração horizontal (SEND_BUSY e RECEIVE_BUSY) com outros conveyors é idêntica.

No caso do conveyor rotativo deste sistema, o conceito de MAX_BOX não se justifica, pois o conveyor só permite acomodar uma caixa de cada vez. Este informa o conveyor precedente que está ocupado sempre que entra em funcionamento. Em todo o caso, a contagem das caixas podia ser feita da mesma forma que na abordagem anterior.

A lógica desenvolvida é baseada no seguinte GRAFCET.

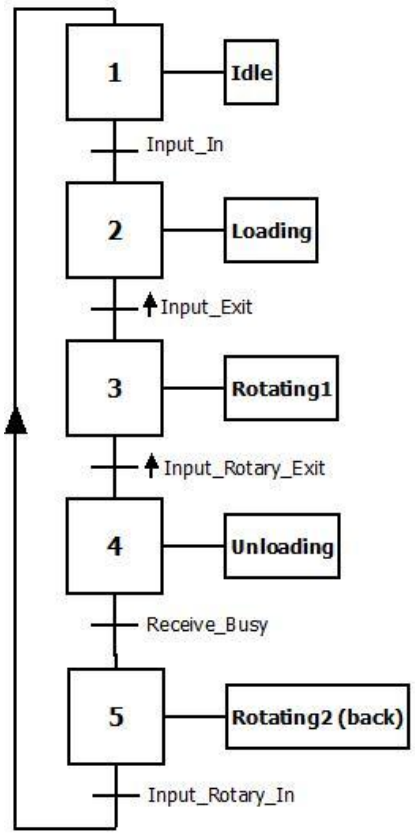


Figura 53 – GRAFCET do conveyor rotativo

```
(* Inicialização de Edge Detections e Counters*)

R_In (CLK := INPUT_IN);
F_Out (CLK := INPUT_EXIT);
F_In (CLK := INPUT_IN);
R_Out(CLK := INPUT_EXIT);
R_In_Rotary(CLK := INPUT_ROTARY_IN);
R_Out_Rotary(CLK := INPUT_ROTARY_EXIT);

(*Controlo de Estados*)

IF Idle AND INPUT_IN THEN
    Idle:=FALSE;
    Loading:=TRUE;
END_IF;

IF Loading AND R_Out.Q  THEN
    Loading:=FALSE;
    Rotating1 := TRUE;
END_IF;

IF Rotating1 AND R_Out_Rotary.Q THEN
    Rotating1 :=FALSE;
    Unloading:=TRUE;
END_IF;

IF Unloading AND RECEIVE_BUSY THEN
    Unloading := FALSE;
    Rotating2:=TRUE;
END_IF;

IF Rotating2 AND INPUT_ROTARY_IN THEN
    Rotating2:=FALSE;
    Idle:=TRUE;
END_IF;

(*Afectação das saídas*)

SEND_BUSY:= Rotating1 OR Rotating2 OR Unloading;
OUTPUT_LINEAR:=((Loading OR Unloading) AND NOT (RECEIVE_BUSY AND INPUT_EXIT));
OUTPUT_ROTARY:=(Rotating1 OR Unloading) AND NOT RECEIVE_BUSY;
```

De notar que a saída do estado Unloading dá-se quando o RECEIVE_BUSY é activado. Isto deve-se ao facto de o sensor de saída INPUT_EXIT (In_6) não cumprir a sua função devidamente e revelar que origina problemas no descarregar das caixas. Sendo assim presume-se que o descarregar da caixa termina quando o conveyor seguinte (ainda não controlado nesta abordagem) indica que está ocupado, sendo que tal acontece quando atinge o valor máximo de caixas a transportar.

Esta particularidade não se prende com qualquer cedência na busca de uma solução flexível, mas apenas com um problema de concepção do próprio sistema em análise.

Em anexo pode ser consultado o código de configuração dos blocos funcionais. Subsequentemente apresenta-se o bloco introduzido nesta abordagem.

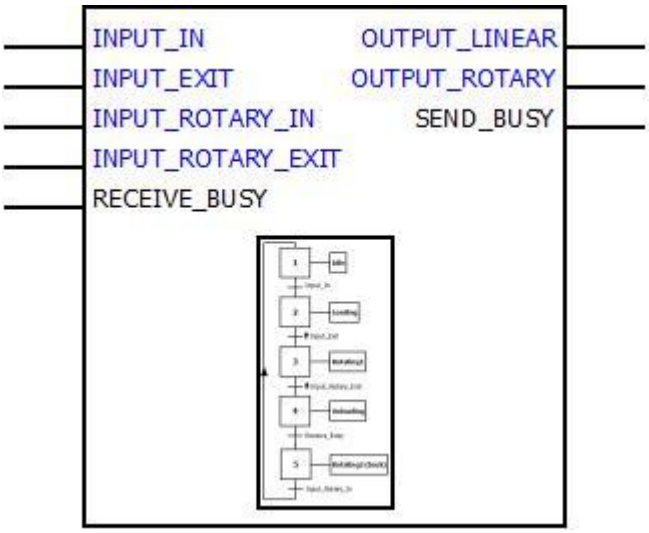


Figura 54 – Bloco funcional do conveyor rotativo

A imagem seguinte demonstra os vários blocos funcionais devidamente configurados e ligados entre si.

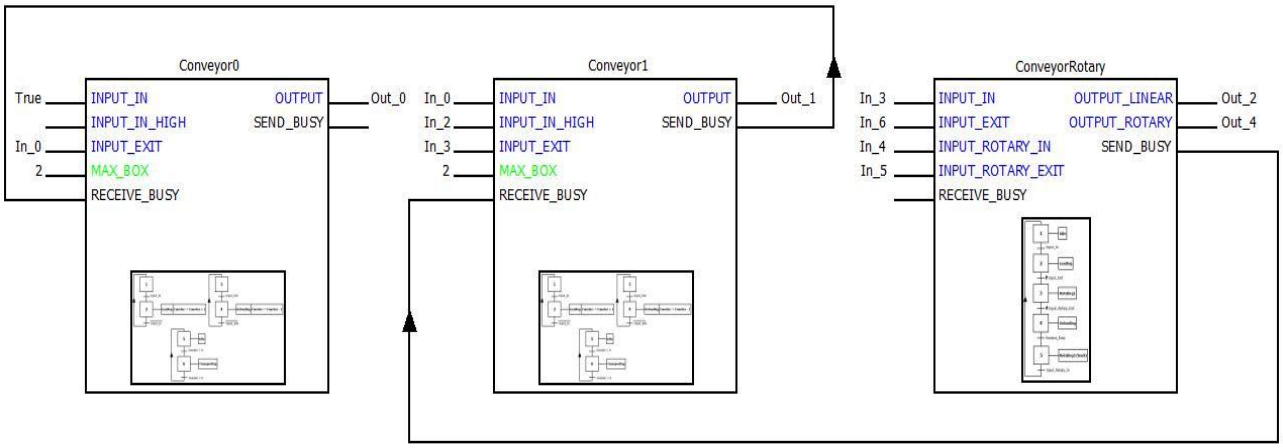


Figura 55 – Diagrama de blocos da solução de controlo dos três primeiros conveyors

Hipótese 3

Pretende-se controlar os três conveyors diferentes desta abordagem com um único bloco funcional.

Devido à diferença fundamental entre um conveyor rotativo e um linear, a solução encontrada para que código relativamente diferente coabite dentro de um bloco funcional, é existir uma variável de configuração que informe o bloco funcional se cada instância criada é um conveyor rotativo ou não.

O código que sequencia as etapas de um conveyor difere no caso em que ele é definido como sendo rotativo (IS_ROTARY) ou não, como se pode ver pelo excerto seguinte.

```
IF Loading AND F_In.Q AND NOT IS_ROTARY THEN
    Loading:=FALSE;
    Transporting := TRUE;
END_IF;

IF Loading AND R_Out.Q AND IS_ROTARY THEN
    Loading:=FALSE;
    Rotating1 := TRUE;
END_IF;
```

Este facto manifestamente introduz alguma repetição de linhas de código, nomeadamente na afectação de variáveis de estado. Outro aspecto importante é que cada bloco funcional acaba por possuir algumas instruções que só serão utilizadas por conveyors rotativos, isso faz com que a eficiência do programa não seja a melhor (se bem que para sistemas simples como o estudado, este problema é desprezável).

Seguidamente representa-se o bloco funcional único alcançado.

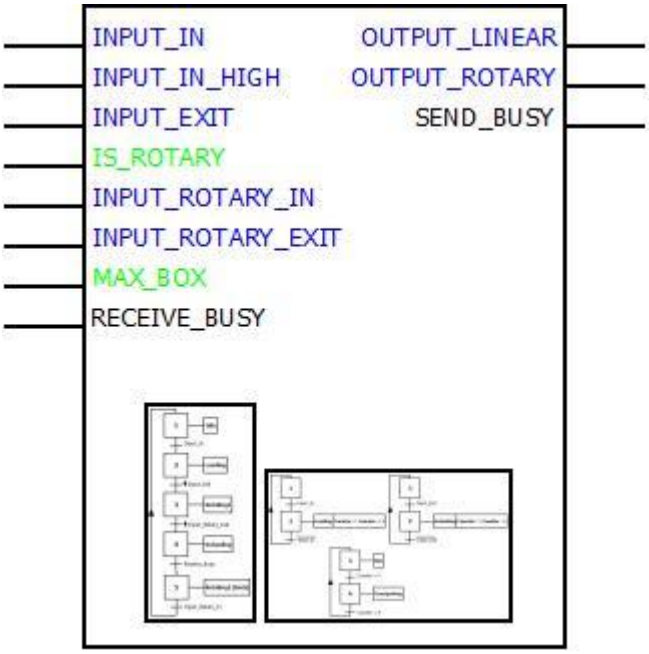


Figura 56 – Bloco funcional único para controlo de conveyors lineares e rotativos

Estão representados dois GRAFCETs internos, mas na realidade só existe um que conjuga as duas lógicas.

A próxima figura demonstra os vários blocos funcionais configurados e ligados entre si.

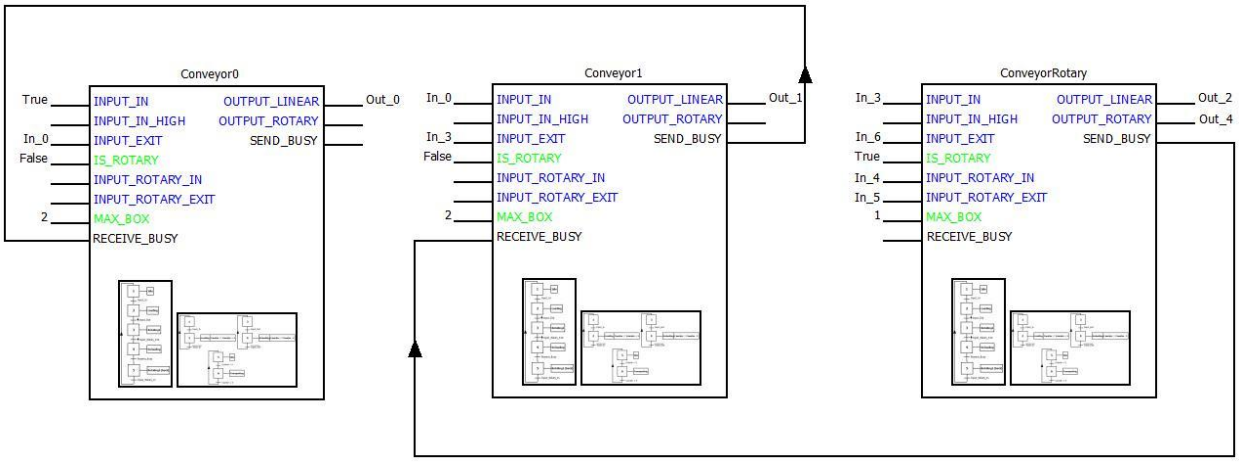


Figura 57 – Diagrama de blocos da solução de controlo para os três primeiros conveyors, com base em instâncias de um único bloco funcional

Uma rápida análise ao diagrama apresentado permite constatar que a configuração de cada bloco funcional necessita de considerável atenção, pois existem diversas variáveis apenas aplicáveis no conveyor rotativo. Tal facto introduz uma característica pouco desejável numa solução flexível – propensão a erros de configuração.

5.3.3. Controlo do Conveyor Alimentador, Conveyor B, Conveyor Rotativo (sem separação) e Conveyor E

Após a abordagem 5.3.2, controlar mais um conveyor linear resume-se a criar mais uma instância de um bloco funcional. O funcionamento do conveyor de saída E é em tudo semelhante ao conveyor B. com a diferença que na sua configuração a variável MAX_BOX deve ser igual a 1 (devido às dimensões do conveyor e à limitação demonstrada na abordagem anterior (o SEND_BUSY do Conveyor E servirá de condição para o término da operação Unloading do conveyor rotativo).

Existem neste momento 4 conveyors diferentes, controlados por instâncias de apenas um (hipótese 3) ou dois blocos funcionais (hipótese 2).

Como exemplo é apresentado o conjunto de blocos funcionais da hipótese 2.

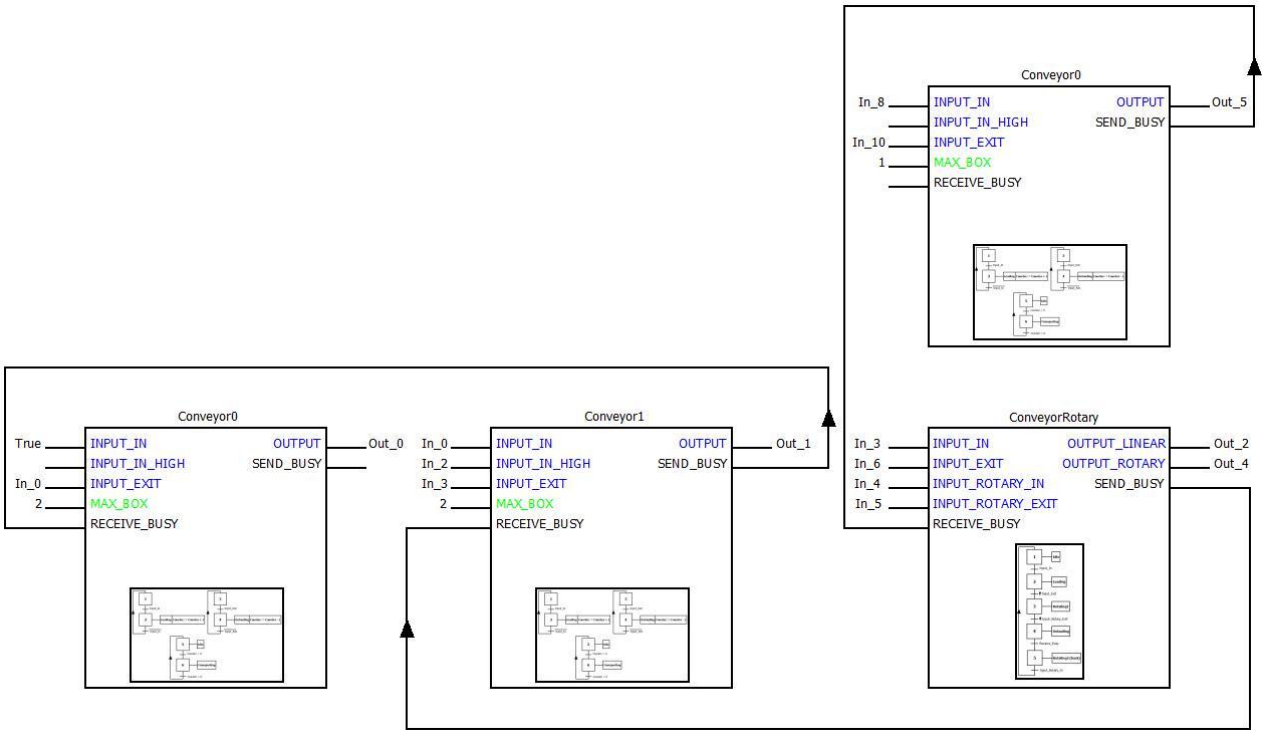


Figura 58 – Diagrama de blocos para a solução de controlo de 4 conveyors (hipótese 2)

5.3.4. Controlo de Todo o Sistema

Com esta abordagem, a investigação chega ao ponto de controlar todo o sistema de sorting. Todos os conveyors serão controlados e todas as funcionalidades implementadas. A novidade em relação às abordagens anteriores é a necessidade de efectuar a triagem das caixas mediante as suas alturas.

Foram testadas as hipóteses 2 e 3 e tendo em conta a experiência obtida nas abordagens anteriores, a que à partida dará melhores resultados é a hipótese 2 – dois tipos diferentes de conveyors.

A investigação prossegue então com a análise da hipótese 2 para o controlo de todo o sistema.

Qualquer sistema de sorting, possui sempre um local onde é feita a análise dos diferentes materiais transportados, e um ponto onde é efectuada a sua concreta separação.

No sistema em questão, a identificação é feita no início do conveyor B pelos sensores 1 e 2 dispostos verticalmente. A separação é dada pela escolha do sentido de descarga do conveyor rotativo (actuador 2 ou 3).

Importa analisar cada um destes processos em separado.

Identificação de Caixas

O bloco funcional de um conveyor linear tem que passar a incluir lógica de identificação de caixas. Uma solução encontrada passa por conter esse pedaço de código num ciclo IF, usando uma

variável booleana IS_QUEUE que define, aquando da configuração inicial, se a instância de um dado bloco funcional irá ou não implementar esta função de medição e criação de uma lista com as alturas aferidas. A lógica de guardar a informação de altura das caixas com a utilização de duas variáveis do tipo WORD – COUNT e QUEUE – foi directamente retirada do livro (Magalhães, 2010).

```
(*Detecção de Alturas*)

IF IS_QUEUE THEN
    IF F_In.Q THEN
        COUNT := ROL (COUNT,1);
        QUEUE := SHL (QUEUE,1);       (* Desloca QUEUE para a esquerda introduzindo um 0 *)
        IF INPUT_IN_HIGH THEN
        QUEUE := QUEUE OR WORD#1; (* Modifica para 1 se a palete que entrou é alta *)
        END_IF;
    END_IF;
END_IF;
```

Devido à utilização de blocos funcionais que encapsulam o código, a informação de um conveyor só está disponível para outro conveyor se existir uma ligação de variáveis de entrada e saída entre si. A lista de alturas tem que ser comunicada ao conveyor rotativo. Como abordagem inicial as variáveis COUNT e QUEUE serão definidas como variáveis globais para atalhar este problema de comunicação.

Separação de Caixas

O conveyor rotativo deve receber a informação do tipo de caixa (lendo as variáveis globais definidas pelo conveyor de identificação) e deve actualizar essa informação. Sempre que separa uma caixa é necessário que a informação da mesma seja retirada da lista. Sendo as variáveis em questão globais, a implementação desta solução é feita com o auxílio de dois estados internos Choose_Side e Divert.

```
Choose_Side:BOOL:=FALSE; (*Estado de transição até que seja definida para onde vai a caixa*)
Divert:BOOL:=FALSE; (*Variável que definie se a caixa vai para a direita ou esquerda*)
                    (*Pode-se generalizar como uma situação de divert*)
```

O conveyor rotativo ao chegar ao ponto de decidir para onde deve encaminhar as caixas, despoleta o algoritmo de leitura e actualização da lista.

De seguida é apresentada a parte mais relevante do código de separação das caixas e a afectação das saídas. A restante lógica interna na generalidade não se altera em relação ao já apresentado.

```
IF Rotating1 AND R_Out_Rotary.Q THEN
    Rotating1 :=FALSE;
    Choose_Side :=TRUE;
    Unloading:=TRUE;
END_IF;

IF R_Choose_Side.Q THEN
    IF (COUNT AND QUEUE) <> WORD#0 THEN  (* Se a palete que sai é alta ... *)
        Divert := TRUE;    (*... será descarregada para a esquerda *)
    ELSE
        Divert := FALSE;   (* senão, para a direita *)
    END_IF;
    COUNT := ROR(COUNT,1);          (* O contador só agora pode ser decrementado *)
END_IF;
```

```
(*Afectação das saídas*)

SEND_BUSY:= Rotating1 OR Rotating2 OR Unloading;
OUTPUT_LINEAR:=((Loading OR Unloading) AND NOT ((RECEIVE_BUSY AND INPUT_EXIT) OR Divert));
OUTPUT_LINEAR_BWD:= Unloading AND Divert;
OUTPUT_ROTARY:=(Rotating1 OR Unloading) AND NOT RECEIVE_BUSY;
```

De notar que a necessidade da existência do estado Choose_Side muito semelhante ao estado de Unloading deve-se a um problema de ordem de execução de código. Se a variável que despoleta o algoritmo de Divert fosse o próprio estado Unloading, o conveyor descarregava sempre para o mesmo lado, uma vez que esse mesmo Unloading, juntamente com a variável Divert, comanda os movimentos lineares de descarga. É necessário existir um estado que garantidamente seja actualizado imediatamente antes do Unloading, para que, quando forem verificadas as condições de afectação das saídas, a variável Divert estar já devidamente actualizada.

Esta solução obriga a que a definição de como se separam as caixas esteja *hard coded* no bloco funcional do conveyor rotativo. Desta forma, o princípio de separação receita / equipamento não é seguido. Contudo tal seria trivialmente resolvido com a criação de input booleano SIDE no bloco funcional, que se podia definir na configuração inicial ou ser alterado em tempo-real com um *Recipe Manager*, HMI ou SCADA. Neste sistema, como só existem dois caminhos possíveis, essa variável podia então inverter o sentido de descarga definido por defeito no bloco funcional, simplesmente alterando a variável Divert que afecta as saídas da seguinte forma.

```
Divert1 := Divert AND SIDE

(*Afectação das saídas*)

SEND_BUSY:= Rotating1 OR Rotating2 OR Unloading;
OUTPUT_LINEAR:=((Loading OR Unloading) AND NOT ((RECEIVE_BUSY AND INPUT_EXIT) OR Divert1));
OUTPUT_LINEAR_BWD:= Unloading AND Divert1;
OUTPUT_ROTARY:=(Rotating1 OR Unloading) AND NOT RECEIVE_BUSY;
```

Em todo o caso, por questões de simplificação da solução, esta alteração não será implementada.

Os blocos encontrados são então os seguintes:

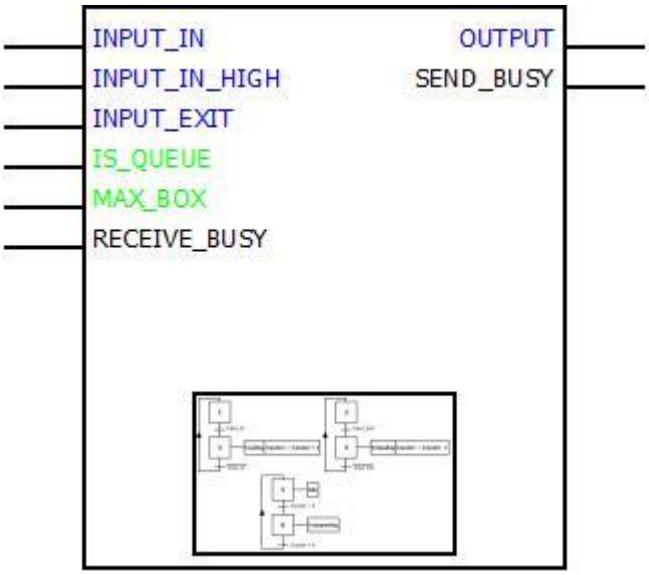


Figura 59 – Bloco funcional do conveyor linear

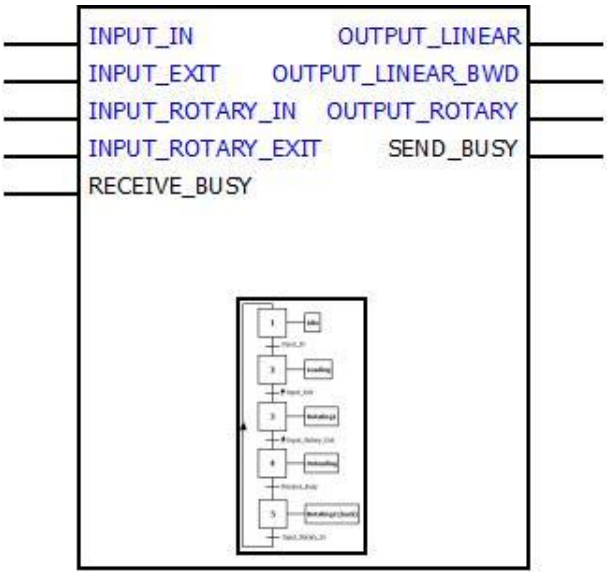


Figura 60 – Bloco funcional do conveyor rotativo

Relativamente à configuração inicial da solução, adicionalmente às abordagens anteriores é necessário apenas definir qual o conveyor linear que possui capacidades de identificação.

```
Conveyor1 (IS_QUEUE := TRUE);
```

É também necessário que os dois conveyors a jusante do rotativo comuniquem o seu estado de indisponibilidade da seguinte forma.

```
ConveyorRotary (RECEIVE_BUSY:= (Conveyor2.SEND_BUSY OR Conveyor3.SEND_BUSY));
```

A seguinte figura apresenta todo o sistema de conveyors com os seus respectivos blocos funcionais configurados e ligados entre si.

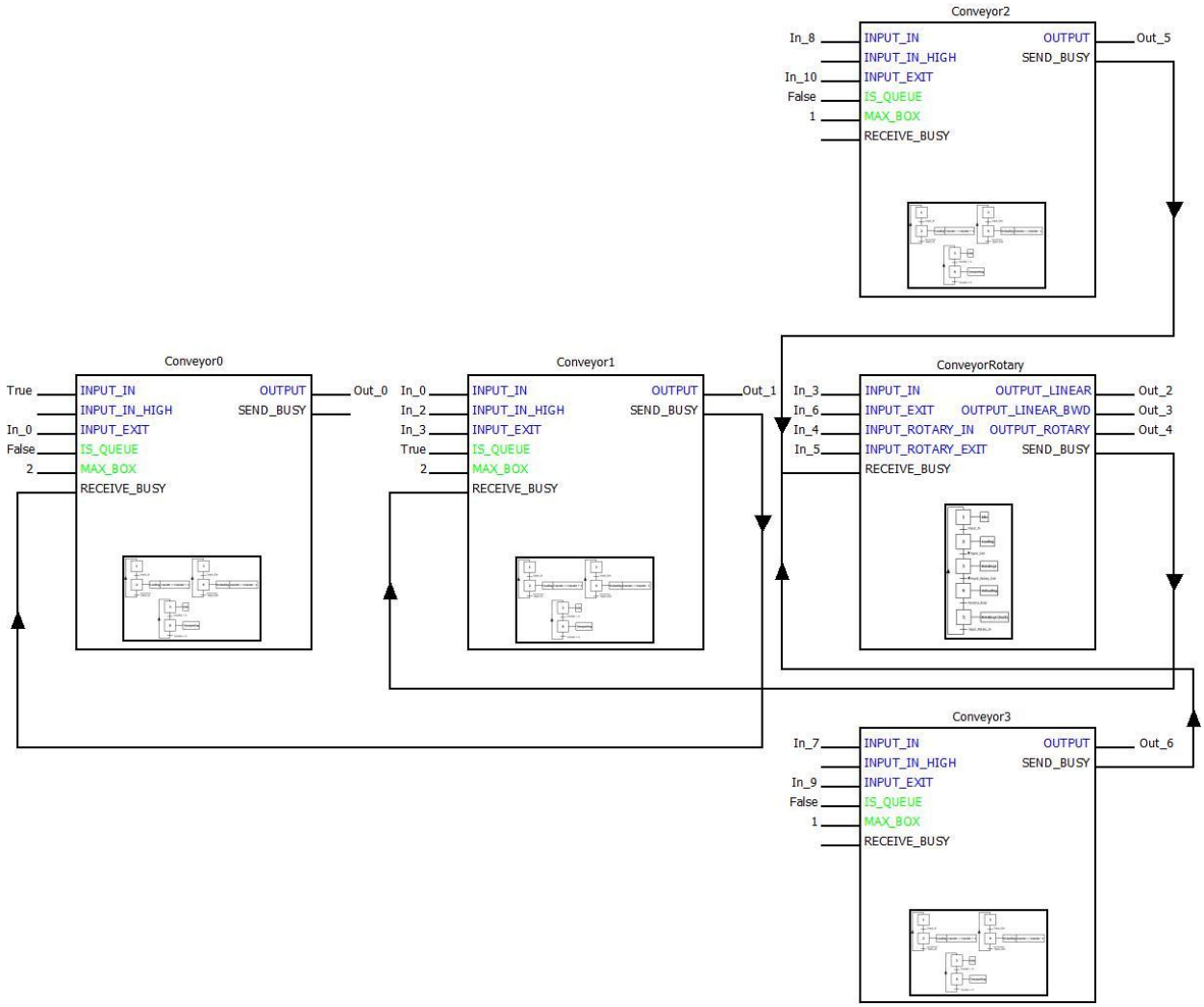


Figura 61 – Diagrama de blocos da solução de controlo para todo o sistema

Com a análise da Figura 61, é notório o alcançar do objectivo da estruturação da solução de controlo seguir a disposição física dos processos e equipamentos. Este é um dos passos fulcrais para a consecução de uma solução de automação flexível e passível de ser distribuída por vários controladores. Contudo para a solução ser suficientemente flexível é necessário pensar-se em que outros sistemas uma dada solução pode ser aplicada. Por outro lado, pensando no objectivo de distribuir o controlo, tal não é possível com a utilização de variáveis globais.

Com a clara noção de que o caminho tem sido profícuo em descobertas sobre como se deve ou não construir uma solução de automação flexível, é necessário eliminar as más práticas deparadas e expandir a solução, na busca do tal limite ideal de flexibilidade.

5.4.Expansão da Aplicação da Solução

A solução descrita em 5.3.4 segue muitos dos princípios teóricos que deram a base a esta dissertação – encapsulamento, blocos funcionais, separação receita / equipamento, padrões de desenho, etc. Contudo é necessário pensar-se noutros sistemas de sorting e na aplicação desta solução de forma a atestar a sua verdadeira flexibilidade e modularidade.

Até ao momento a hipótese 2 - um bloco funcional que permite ser aplicado a qualquer tipo de conveyor linear e um bloco funcional que permite ser aplicado a conveyors rotativos - assegura-se como a solução mais válida e a que permite atingir a melhor relação entre flexibilidade e simplicidade de configuração.

Praticamente qualquer sistema, composto por estes dois tipos de conveyors, pode ser modularmente controlado através da aplicação e configuração de instâncias dos blocos funcionais desenvolvidos. Conveyors que tenham accionamento diferente podem ser controlados tendo por base esta solução, alterando apenas as saídas do bloco. Até equipamentos distintos, tais como um simples elevador, pode ser visto também como um conveyor linear, na medida em que o seu movimento dá-se entre dois ou mais pontos bem definidos. É tudo uma questão de alguma adaptação funcional sem necessidade de reestruturação fundamental.

Porém, após análise e reflexão, os seguintes problemas podem ser levantados:

- A utilização de variáveis globais (lista de alturas das caixas) vai contra o princípio de controlo distribuído segundo a IEC 61499;
- Se um sistema possuir mais do que um ponto de separação (várias mesas rotativas) passa a ser necessário existir mais do que uma lista de alturas, mesmo que só haja um ponto de medição de alturas;
- Se um dado conveyor possuir mais do que um ponto de medição a solução não se aplica;
- Se existir um conveyor entre o conveyor de medição e o rotativo a solução não é valida;
- Cada conveyor não detém informação alguma sobre que tipo de caixas está a transportar. A informação é criada aquando da medição de alturas e utilizada no ponto de separação e nada mais. Isto não é conceptualmente um bom princípio. O conceito de *traceability* não é respeitado.

Com base nos problemas e falhas descobertas é forçoso voltar à primeira etapa iterativa deste capítulo.

Subdivisão do Sistema em Módulos.

Todos os problemas encontrados prendem-se de alguma forma com o processo de medição de alturas e com a forma como essa informação é comunicada e gerida. A flexibilidade almejada justifica que seja elaborada uma solução com mais um bloco funcional distinto, com o simples objectivo de efectuar a medição de alturas e comunicar aos conveyors essa informação. Dessa forma a lógica de medição de alturas torna-se externa ao próprio conveyor e esse ponto de medição pode-se situar em qualquer parte do sistema, podendo até existir mais do que um.

Tabela 4 – Hipótese abrangente de subdivisão do sistema em módulos de equipamento

Equipamento	Hipótese Final de Subdivisão do Sistema
Alimentador (0)	FB1
Transportador (1)	FB1
Identificador (ID)	FB3
Rotativo (Rotary)	FB2
Transportador de Saída (2)	FB1
Transportador de Saída (3)	FB1

Com esta alteração à estrutura da solução e com a obrigação de eliminar a utilização de variáveis globais, os conveyors carecem de receber e comunicar a informação das caixas que transportam, pelo que se deve rever algum do seu código.

Em particular, o processo de transmitir correctamente informação entre conveyors é de extrema importância e por vezes de difícil solução, pelo que foi feito um esforço de encontrar uma resolução que seja ao mesmo tempo elegante e eficaz.

5.5. Solução Final

Com base nas alterações idealizadas e após um longo período de experiências, a solução final alcançada é composta por instâncias de três blocos funcionais – Conveyor Linear, Conveyor Rotativo e Identificador.

Importa analisar cada bloco e suas particularidades. Começando pelo que é novo em relação às abordagens iniciais.

Identificador

O bloco funcional que permite às caixas serem identificadas, pode ser visto como o ponto de entrada de informação no sistema, e este ponto pode não coincidir com o ponto de entrada de caixas. A partir da criação da informação, esta deve ser seguramente transmitida ao equipamento em que se encontra posicionado o sistema de identificação (neste caso o Conveyor1).

```
FUNCTION_BLOCK ID
VAR_INPUT
    INPUT_IN: BOOL; (*Sensor detecçao Caixa*)
    INPUT_IN_HIGH:BOOL; (*Sensor detecçao Caixa Alta*)
END_VAR
VAR_OUTPUT
    BOX_HEIGHT:INT:=0; (*Altura da caixa*)
    BOX_SENT:BOOL:=FALSE; (*Evento de caixa identificada*)
END_VAR
VAR
    R_In    : R_TRIG; (*Rising  trigger do sensor de entrada*)
    F_In:F_TRIG; (*Falling trigger do sensor de caixa alta de entrada*)
    F_Identifying :F_TRIG; (*Falling trigger da flag Identifying*)

    Idle:BOOL :=TRUE; (*Estado idle*)
    Identifying :BOOL:= FALSE; (*Estado de caixa a ser identificada*)
END_VAR
```

As variáveis de entrada (INPUT_IN e INPUT_IN_HIGH) resumem-se aos sensores que permitem a detecção de alturas.

As variáveis de saída (BOX_HEIGHT e BOX_SENT) são toda a informação que é necessária transmitir ao conveyor que possui o identificador.

O identificador possui dois estados: em espera – Idle – e a identificar – Identifying.

Ao ocorrer a transição descendente do estado Identifying, a variável de saída BOX_SENT é enviada juntamente com a informação actualizada da altura da caixa em BOX_HEIGHT. Esta variável é um inteiro e pode tomar os seguintes valores:

Tabela 5 – Definição da variável BOX_HEIGHT do bloco funcional Identificador

BOX_HEIGHT	Tipo de Caixa
-	Indiferente
1	Baixa

2	Alta

Se existirem caixas com mais que duas alturas diferentes, é necessário considerar essa alteração no bloco. No caso de sistemas com processos diferentes de identificação de materiais é necessário refazer a lógica que está contida no ciclo IF Identifying. Em todo o caso a estrutura do bloco pode manter-se a mesma.

```
R_In (CLK := INPUT_IN);
F_In (CLK:=  INPUT_IN);
F_Identifying (CLK:= Identifying);

IF Idle AND R_In.Q THEN
    Idle:=FALSE;
    BOX_HEIGHT := 1;
    Identifying:=TRUE;
END_IF;

IF Identifying THEN
    IF INPUT_IN_HIGH THEN
    BOX_HEIGHT := 2;
    END_IF;
END_IF;

IF Identifying AND F_In.Q THEN
    Identifying := FALSE;
    Idle :=TRUE;
END_IF;

BOX_SENT := F_Identifying.Q;
```

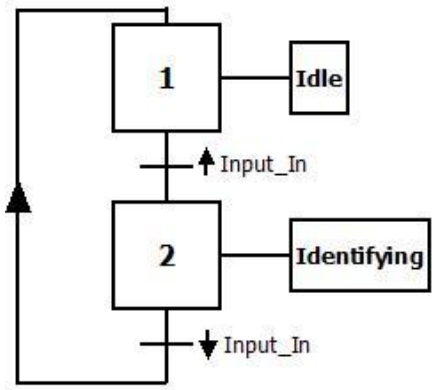


Figura 62 – GRAFCET que descreve a lógica do bloco funcional Identificador

A representação gráfica do bloco funcional Identificador é a que se segue.

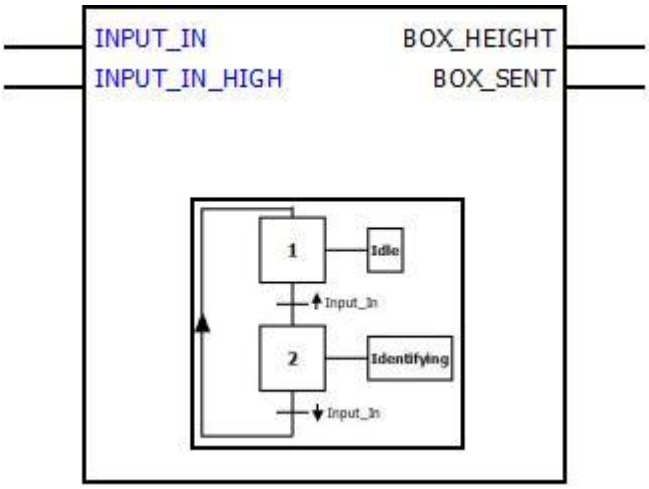


Figura 63 – Bloco funcional de identificação da altura das caixas

Conveyor Linear

O bloco de conveyor linear é semelhante ao desenvolvido nas abordagens iniciais. Acresce-se a capacidade de qualquer conveyor transportar informação além de transportar as próprias caixas. Em relação às variáveis de entrada, além dos inputs físicos (INPUT_IN e INPUT_EXIT) existem os inputs de configuração MAX_BOX e RECEIVE_BUSY, que já foram apresentados anteriormente, e as novas variáveis NEW_BOX e BOX_HEIGHT_IN.

```
FUNCTION_BLOCK CONVEYOR
VAR_INPUT
(*Inputs Físicos*)
    INPUT_IN: BOOL; (*Sensor de Entrada*)
    INPUT_EXIT:BOOL; (*Sensor de Saída*)

(*Inputs Software*)
    MAX_BOX:INT; (*Máximo de caixas presentes simultaneamente no conveyor*)
    RECEIVE_BUSY:BOOL; (*Variável de entrada que indica que o próximo conveyor
                         não se encontra em posição de receber mais caixas*)
    NEW_BOX:BOOL;(*Evento que indica que uma nova caixa entrou no conveyor*)
    BOX_HEIGHT_IN:INT:=0;(*Altura da caixa que entrou*)
END_VAR
```

A variável NEW_BOX é um booleano que deve ser ligado ao BOX_SENT do conveyor precedente ou de um bloco identificador. Esta variável faz com a fila interna do conveyor seja actualizada com a informação de uma nova caixa. A informação é lida de BOX_HEIGHT_IN que deve estar ligada ao BOX_HEIGHT de um conveyor precedente ou bloco identificador.

As variáveis de saída são o comando do movimento do conveyor (OUTPUT) e as variáveis de coordenação SEND_BUSY, BOX_HEIGHT_OUT e BOX_SENT.

```
VAR_OUTPUT
(*Outputs Físicos*)
    OUTPUT: BOOL; (*Comando do motor do conveyor*)

(*Outputs Software*)
    SEND_BUSY:BOOL;(*Variável que indica ao conveyor antecedente que não
                     se encontra em posição de receber mais caixas*)
    BOX_HEIGHT_OUT:INT;(*Altura da caixa que sai*)
    BOX_SENT:BOOL:=FALSE;(*Evento de caixa enviada para o conveyor seguinte*)
END_VAR
```

O booleano SEND_BUSY já foi apresentado em abordagens anteriores. O inteiro BOX_HEIGHT_OUT é a informação da altura da caixa que acaba de sair do conveyor. Esta é transmitida aquando da activação do evento booleano BOX_SENT.

As variáveis internas do bloco são apresentadas de seguida.

```
VAR
(*Triggers*)
    R_In    : R_TRIG; (*Rising trigger do sensor de entrada*)
    F_In    : F_TRIG; (*Falling trigger do sensor de saída*)
    R_Out : R_TRIG; (*Rising trigger do sensor de saída*)
    F_Out  : F_TRIG;(*Falling trigger do sensor de saída*)
    Counter :CTUD; (*Contador do Max_Box*)
    R_New_Box :R_TRIG;(*Rising trigger do evento New_Box*)
    F_Unloading:F_TRIG; (*Falling trigger do estado Unloading*)
    Pulse_Box_Sent :TP; (*Timer Pulse que garante que a informação de caixa
                         enviada é correctamente recebida pelo conveyor seguinte*)

(*Estados Internos*)
    Idle:BOOL:=TRUE; (*Parado e pronto a iniciar marcha*)
    Loading:BOOL :=FALSE; (*A carregar uma caixa*)
    Transporting:BOOL:=FALSE; (*A transportar uma caixa*)
    Unloading:BOOL:=FALSE; (*A descarregar uma caixa*)

(*Lista de caixas e suas alturas*)
    Count:WORD:=16#8000; (*Contador caixas*)
    Queue:WORD; (*Fila caixas*)
    Is_Queue: BOOL; (*Flag que faz com que a queue / count sejam actualizadas
                     à saida do conveyor.*)
END_VAR
```

Importa atentar à lista de alturas das caixas Count e Queue, e na variável Is_Queue que na execução do código informa que a fila deve ser actualizada ao enviar uma caixa para o conveyor seguinte. Isto permite que, se um dado conveyor não receber informação das caixas que entram, esta também não será tentada transmitir aquando da saída das caixas.

O código interno do bloco encontra-se detalhado em anexo. Apresenta-se uma porção de código relevante.

```
(*Gestão da Informação de Alturas*)

IF R_New_Box.Q THEN
    Is_Queue:=TRUE;
    Count := ROL (Count,1);
    Queue := SHL (Queue,1);
    IF BOX_HEIGHT_IN = 2 THEN
        Queue := Queue OR WORD#1;
    END_IF;
END_IF;

IF Is_Queue AND F_Out.Q THEN
        IF (Count AND Queue) <> WORD#0 THEN
        BOX_HEIGHT_OUT := 2;
        ELSE
        BOX_HEIGHT_OUT := 1;
        END_IF;
        Count := ROR(Count,1);
        Is_Queue:=FALSE;
END_IF;
```

O conveyor ao receber informação do conveyor a montante ou do identificador, actualiza a sua lista interna. Imediatamente após dar-se a saída de uma caixa (F_Out.Q), e estando o conveyor informado da altura da caixa, este actualiza a sua fila interna e transmite a altura da caixa para o próximo conveyor.

De seguida a representação gráfica do bloco funcional Conveyor Linear

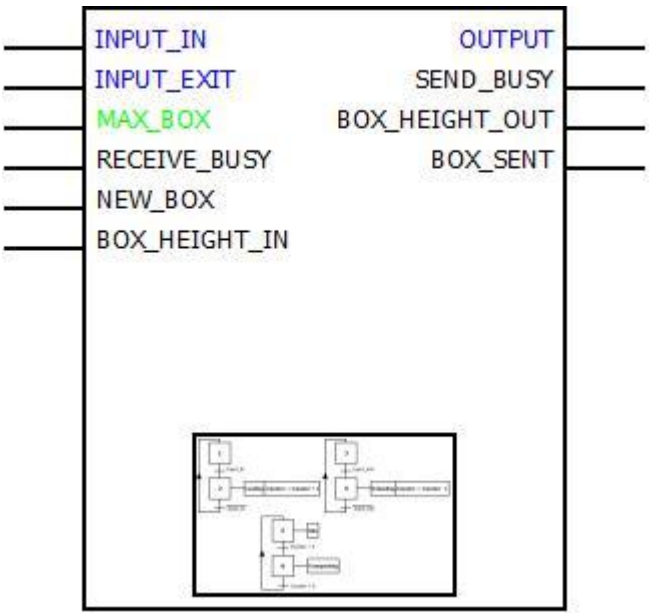


Figura 64 – Bloco funcional de conveyor linear

Conveyor Rotativo

O bloco funcional de um conveyor rotativo tem por base o bloco de conveyor linear, com as suas particularidades físicas de funcionamento e com o acréscimo de código que lhe permite efectuar a correcta separação de caixas com base na informação que recebe sobre essas.

As suas variáveis de entrada físicas são INPUT_IN, INPUT_EXIT, INPUT_ROTARY_IN e INPUT_ROTARY_EXIT. As variáveis de software são iguais às do conveyor linear, com a excepção que não existe a definição de MAX_BOX, já que neste caso o conveyor rotativo só pode mesmo transportar uma caixa de cada vez.

```
FUNCTION_BLOCK CONVEYOR_ROT
VAR_INPUT
(*Inputs Físicos*)
    INPUT_IN: BOOL; (*Sensor de Entrada*)
    INPUT_EXIT:BOOL; (*Sensor de Saída*)
    INPUT_ROTARY_IN:BOOL; (*Sensor de tapete rotativo na posição de carga*)
    INPUT_ROTARY_EXIT:BOOL;(*Sensor de tapete rotativo na posição de descarga*)

(*Inputs Software*)
    RECEIVE_BUSY:BOOL; (*Variável de entrada que indica que o próximo conveyor
                         não se encontra em posição de receber mais caixas*)
    NEW_BOX:BOOL; (*Evento que indica que uma nova caixa entrou no conveyor*)
    BOX_HEIGHT_IN:INT:=0; (*Altura da caixa que entrou*)
END_VAR
```

As variáveis de saída partilham também semelhanças com o conveyor linear, existindo ademais uma distinção no evento BOX_SENT (LEFT ou RIGHT) que permite que os conveyors a jusante saibam qual o tipo de caixa que vão receber (BOX_HEIGHT_OUT).

```
VAR_OUTPUT
(*Outputs Físicos*)
    OUTPUT_LINEAR: BOOL; (*Comando do motor do accionamento linear para a frente*)
    OUTPUT_LINEAR_BWD:BOOL;(*Comando do motor do accionamento linear para trás*)
    OUTPUT_ROTARY:BOOL; (*Comando do motor do accionamento rortativo*)

(*Outputs Software*)
    SEND_BUSY:BOOL; (*Variável que indica ao conveyor antecedente que não
                      se encontra em posição de receber mais caixas*)
    BOX_SENT_LEFT:BOOL; (*Evento de caixa enviada para para a esquerda*)
    BOX_SENT_RIGHT:BOOL; (*Evento de caixa enviada para para a direita*)
    BOX_HEIGHT_OUT:INT;(*Altura da caixa que sai*)
END_VAR
```

Pode surgir a pergunta sobre qual o interesse de informar os conveyors de saída qual a caixa que estão a transportar já que só existem dois tipos de caixas. A resposta é: se pensarmos num sistema em que existam mais tipos de caixas, é necessário fazer o sorting em vários conveyors rotativos, sendo preciso, portanto, passar a informação da altura das caixas até ao último processo de separação.

Como exemplo apresenta-se a porção de código de afectação das saídas do bloco. Convém reparar que a informação da caixa que sai do conveyor é a mesma que entrou, sem haver alterações intermédias. Por não poderem existir variáveis de entrada e saída com o mesmo nome, justifica-se existirem duas variáveis que tomarão, em todo o instante, valores iguais mas possuem nomes diferentes.

```
(*Afectação das saídas*)

SEND_BUSY:= Rotating1 OR Rotating2 OR Unloading;
OUTPUT_LINEAR:=(Loading OR (Unloading AND BOX_HEIGHT_IN=1));
OUTPUT_LINEAR_BWD:= Unloading AND BOX_HEIGHT_IN = 2;
OUTPUT_ROTARY:=(Rotating1 OR Unloading) AND NOT RECEIVE_BUSY;
BOX_SENT_LEFT:= BOX_HEIGHT_IN = 1 AND Pulse_Box_Sent.Q;
BOX_SENT_RIGHT:= BOX_HEIGHT_IN = 2 AND Pulse_Box_Sent.Q;
BOX_HEIGHT_OUT := BOX_HEIGHT_IN;
```

O restante código deste bloco pode ser consultado em anexo e a representação gráfica do bloco é a seguinte.

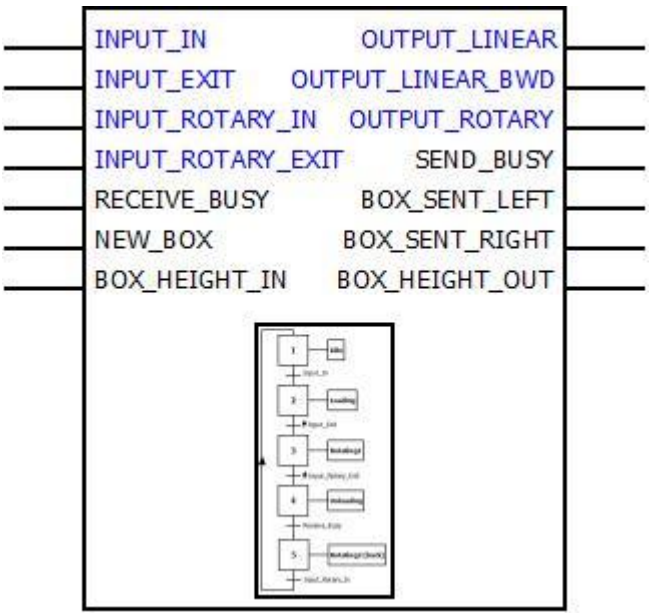


Figura 65 – Bloco funcional de conveyor rotativo

Diagrama de Blocos Funcionais

Ao serem instanciados, configurados e interligados, os blocos funcionais apresentados conseguem garantir o controlo modular e flexível do sistema de Sorting em análise.

O que pode parecer um razoavelmente complexo emaranhado de blocos e ligações, não é mais que o replicar da estrutura física do próprio sistema.

O diagrama de blocos da solução final é o seguinte:

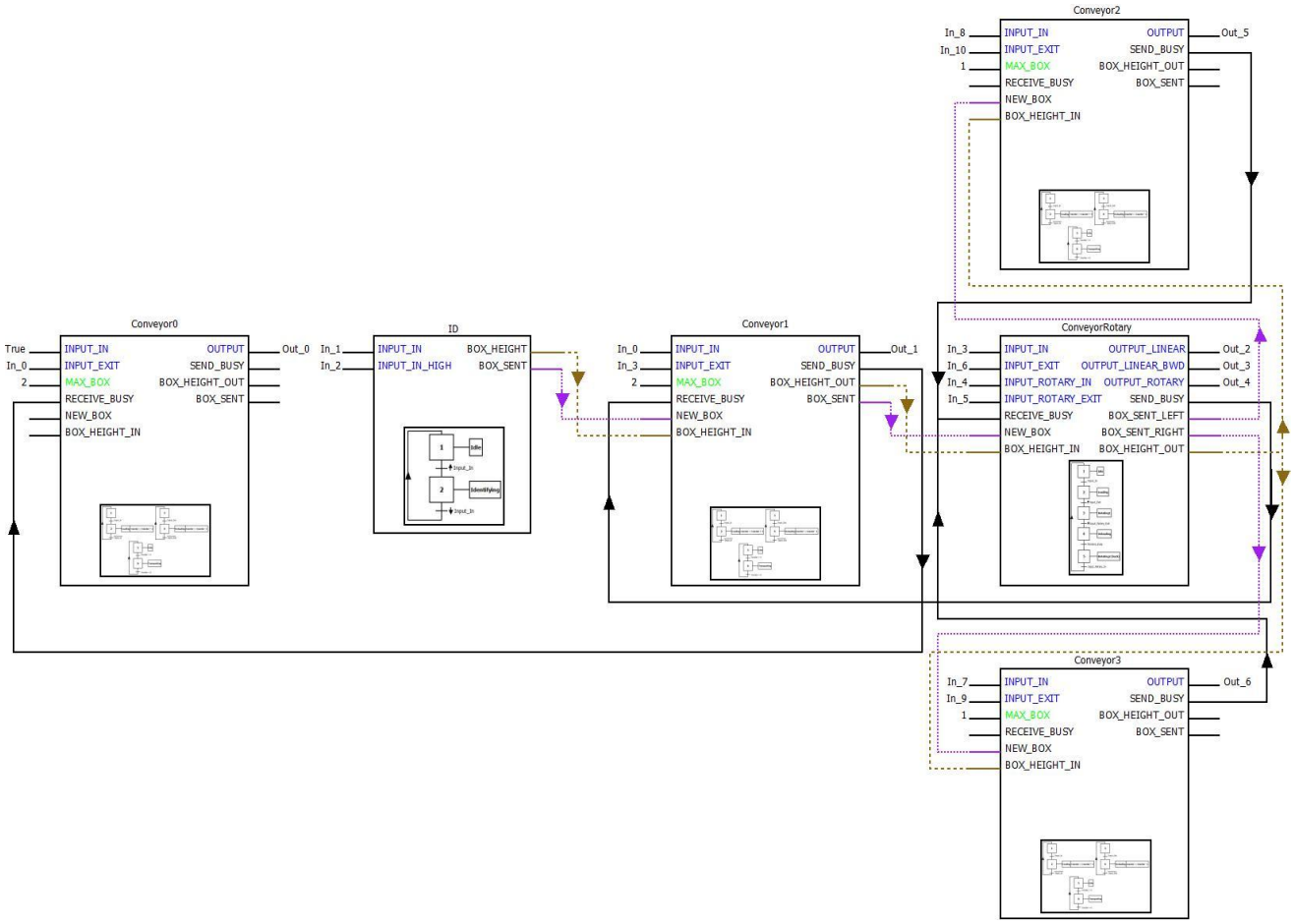


Figura 66 – Diagrama de blocos da solução final para o controlo do sistema Sorting

Em anexo pode-se consultar este diagrama com mais detalhe.

É claro o replicar da estrutura física do sistema na estrutura do software. O que não é tão claro, mas vital referir, é que o facto de existir um ponto de criação de informação e todos os equipamentos poderem transmitir essa informação faz com que existam duas camadas da solução de controlo:

- **Camada Física** – Toda a sensorização, accionamento e sequenciação de acções que fazem com que os equipamentos executem as suas funções;
- **Camada de Informação** – Toda a informação das caixas que são transportadas e sua consequente gestão e comunicação.

Ao relembrar-se os fundamentos abordados no capítulo teórico sobre padrões de desenho, a camada física pode ser interpretada como o controlo procedural e básico, e a camada de informação o controlo de coordenação.

Estas duas camadas estão essencialmente separadas entre si, pois mesmo que um conveyor não saiba que tipo de caixa tem na sua entrada, ele vai executar na mesma a sua função – transportar a caixa até à saída. Vendo por outro prisma, as camadas estão relacionadas na medida em que cada caixa e sua informação estão bem localizadas em cada conveyor.

Pensando numa questão em particular, será necessário que cada conveyor possua um sensor de entrada ou basta que o conveyor precedente informe que chegou uma caixa? Devido à separação entre camadas é importante que exista o sensor físico para que uma camada nunca dependa da outra mas apenas se complementem e ajudem na consecução do objectivo do sistema.

Esta relação prova ser chave para uma solução flexível passível de ser aplicada a sistemas distintos.

Comparativo de Soluções

É de clara utilidade efectuar-se uma comparação da solução final com as hipóteses testadas pelo caminho (Tabela 6). A métrica qualitativa usada é a seguinte:

- **Flexibilidade / Modularidade** – Mede a aplicabilidade da solução a sistemas distintos, tendo em conta o aspecto modular da sua estrutura.
- **Facilidade de Configuração** – Lembrando o gráfico da Figura 1, a facilidade de configuração inicial é importante, pois é necessário chegar-se a um compromisso deste valor com a flexibilidade. Por muito flexível que uma solução seja, se exigir um esforço exagerado de configuração e compreensão da sua aplicação, perdem-se as vantagens de um bom padrão de desenho.
- **Reutilização de Software** – Mede a quantidade de código que se reutiliza. Quanto maior a reutilização de código e menos se repetir o mesmo código em partes diferentes da solução, mais sólida e eficiente esta é.

Tabela 6 – Comparativo qualitativo das várias hipóteses de controlo do sistema Sorting

Hipótese	Flexibilidade / Modularidade	Facilidade de Configuração	Reutilização Software
1	***	****	****
2	***	***	***
3	**	*	**
Solução final	*****	****	***

A solução final é a que apresenta melhor compromisso nas duas fundamentais características flexibilidade / facilidade de configuração. A reutilização de software é elevada, apenas pecando na repetição de algumas etapas do funcionamento dos conveyors lineares e rotativos.

A métrica apresentada, não obstante de ser indicativa da verdadeira aplicabilidade da solução e suas vantagens, é de cariz subjectivo. De forma a tornar esta análise mais tangível, pode-se traçar uma relação destas variáveis com o que mais interessa a nível industrial – clara vantagem económica.

Imagine-se que o sistema de sorting estudado necessita uma vez por ano de sofrer alguma alteração de forma a acomodar as necessidades produtivas. A alteração implicaria adicionar novos tapetes e mudar o tipo de caixas a separar. Numa situação destas, as características apresentadas na Tabela 6 seriam realmente postas à prova, e a flexibilidade / modularidade e a facilidade de configuração seriam na realidade sinónimos de dinheiro poupado. A poupança justifica-se pela rapidez com que a alteração seria efectuada e pelo eliminar de imponderáveis de programação que pudessem condicionar o arranque da nova configuração da instalação.

Sobre a reutilização de software pode-se debater se existe correlação com algo económico, mas sendo que a reutilização de software minimiza a propensão a erros, um sistema de produção sem interrupções por mau funcionamento é um sistema mais lucrativo.

Até ao momento provou-se que várias abordagens, hipóteses, soluções encontradas e muitos conceitos adquiridos foram aplicados da melhor forma, todavia, relembrando uma considerável parte da fundamentação teórica desta dissertação, quem lê pode e deve ocorrer-lhe a seguinte questão:

E então a norma IEC 61499?

Como anunciado, o controlador usado – CoDeSys – não segue a norma IEC 61499, pelo que as soluções desenvolvidas tiverem de ser pensadas de forma respeitar a norma em si mesmo, independente desta estar aplicada ou não.

A linguagem de programação escolhida – Blocos Funcionais segundo a norma IEC 61131-3 – aproxima-se desde logo da linguagem de estruturação de soluções de automação eleita pela IEC 61499. Analisando as diferenças já apresentadas no capítulo 3.4 e olhando para a solução encontrada, prova-se que há uma clara aproximação da norma IEC 61149 pelos seguintes motivos:

- A solução final não utiliza variáveis globais nem *access paths* para aceder a variáveis internas;
- Pode-se argumentar que os blocos funcionais usados só possuem um nível (não existem blocos contidos dentro de blocos), mas para o sistema em questão tal não se revelou necessário;
- A solução encontrada não é efectivamente *event-driven* segundo a definição da IEC 61499, todavia existe uma semelhança clara entre conceitos:

- *Event inputs / outputs* – As variáveis booleanas (sem correspondência física) de entrada e saída dos blocos funcionais (NEW_BOX por exemplo) podem ser vistas como os eventos que despoletam acções dentro dos blocos e que inibem e desinibem a leitura das restantes variáveis comunicadas;
- *Data inputs / outputs* – Essas mesmas variáveis comunicadas (BOX_HEIGHT por exemplo) são uma analogia dos *data inputs / outputs* da norma 61499;
- ECC / Algoritmo interno – O algoritmo interno possui uma dada quantidade de estados e com a sua execução é feito o controlo de cada equipamento.

- A distribuição da aplicação de controlo por vários recursos de um dispositivo, como previsto na Figura 34, foi simulado colocando cada bloco funcional a correr numa *task* independente do controlador. O funcionamento obtido foi idêntico ao caso em que todo o código se encontra a ser executado numa só *task*.

No que se prende com a ausência do *scan time* na norma IEC 61499, não há forma de replicar esse conceito na presente abordagem. Ou seja, a cada ciclo do PLC, qualquer evento deve ser verificado se ocorre ou não. Daqui se conclui que a nível de eficiência de código pouco se pode aferir com esta abordagem. De qualquer forma a grande ambição de busca desta solução é a nível estrutural e não de implementação *per se*.

Ilustra-se de seguida como seria o bloco funcional Conveyor segundo a implementação pura da norma IEC 61499.

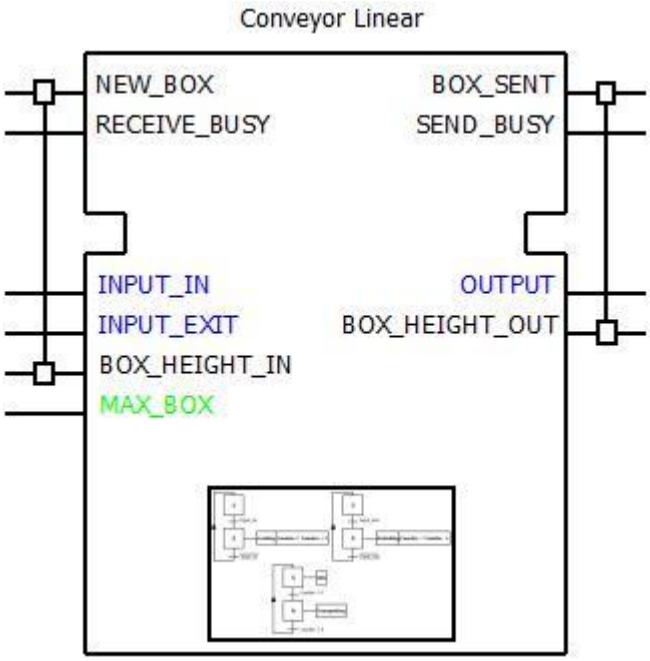


Figura 67 – Bloco funcional segundo a norma IEC 61449 para um conveyor linear

Esta representação revela-se de acordo com as bases teóricas expostas e com as investigações desenvolvidas na área. Notar que NEW_BOX, RECEIVE_BUSY, BOX_SENT e SEND_BUSY são definidas como *events*, e as restantes variáveis como *data*. Salienta-se ainda o sincronismo entre as variáveis NEW_BOX e BOX_HEIGHT_IN e entre BOX_SENT e BOX_HEIGHT_OUT (assinalado com um quadrado). Este sincronismo garante que o código seja apenas executado na altura devida, usufruindo da leitura no momento certo da variável associada.

Os restantes blocos seguiriam o mesmo princípio. Pelo que não são representados.

Concluindo, os blocos funcionais encontrados são, por si mesmo, os padrões de desenho que a dissertação se propôs encontrar, e a sua implementação segundo a 61499 é possível e traria as vantagens de interoperabilidade, reconfigurabilidade e portabilidade que a norma acarreta.

5.6. Sistema Real

Após tamanha experimentação em ambiente virtual, como corolário escolheu-se a aplicação teórica das abordagens testadas a um sistema bem real e de vital importância no ambiente industrial da Europac Kraft Viana, em que me encontro a desenvolver funções de projecto e manutenção.

O sistema é um RHS – Roll Handling System - Linha de Acabamentos de Bobines de Papel - e consiste num sistema de encaminhamento e processamento de bobines de papel kraft até o seu destino no armazém geral. Na sua essência não é um sistema de sorting, pois o seu objectivo principal não é a separação mas em todo o caso a semelhança tecnológica, funcional e estrutural justifica a sua análise.

Análise do Sistema

O sistema é composto por dez conveyors lineares específicos para o transporte de bobines de papel, um conveyor rotativo, uma queda gravítica e um descedor de bobines. Ao longo do percurso são executadas diversas acções sobre as bobines: pesagem, impressão de etiquetas, impressão de informação e cintagem.

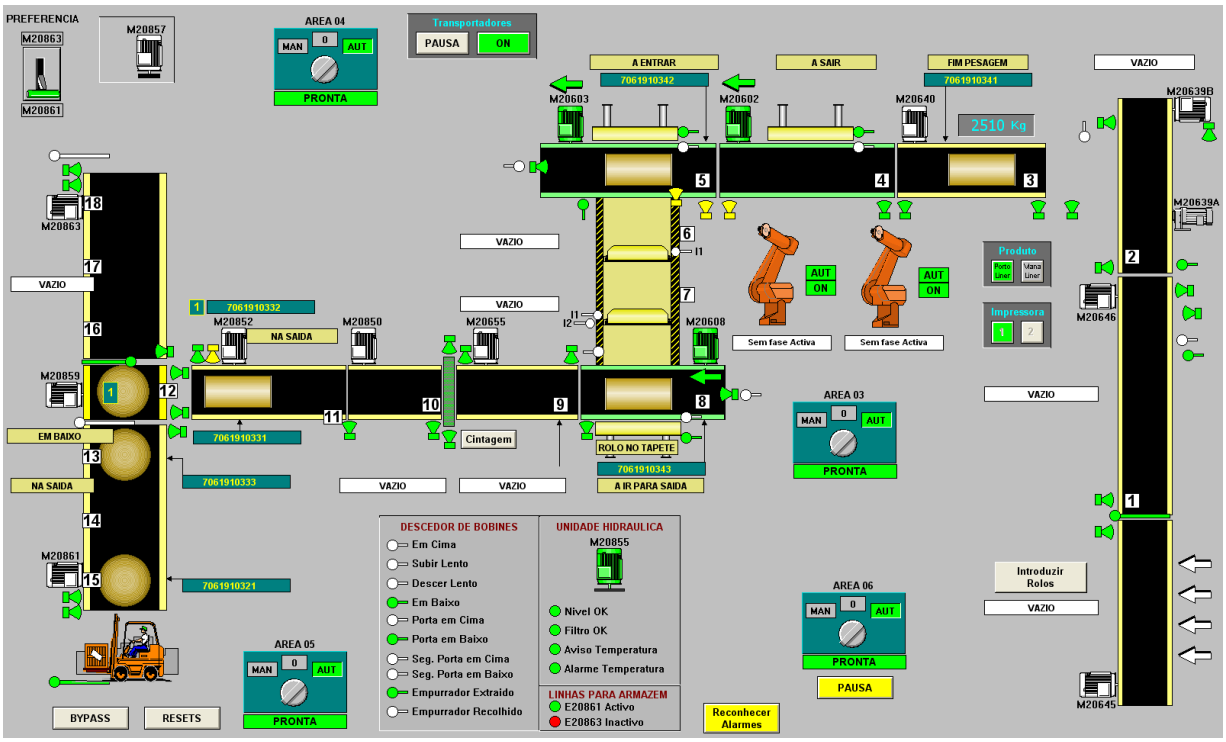


Figura 68 – Sistema de supervisão e controlo da linha de acabamentos de bobines de papel *(cortesia da Europac Kraft Viana)*

Segue-se a descrição do sistema um pouco mais em detalhe, usando os números indicados na Figura 68:

- A entrada de bobines no sistema dá-se pelo primeiro tapete (no canto inferior direito), sendo que este é comandado manualmente. As bobines chegam a este tapete após serem bobinadas na bobinadora (não representada).
- 1 – Conveyor que transporta bobines até ao conveyor rotativo 2 e que também possui movimento de elevação. O sensor fotoeléctrico de entrada, associado ao encoder do motor, é usado para aferir o comprimento efectivo da bobine – Figura 69.

Figura 69 – Conveyor elevador de bobines *(cortesia da Europac Kraft Viana)*

- 2 – Conveyor rotativo que roda as bobines para serem alimentadas ao conveyor 3 – Figura 70.

Figura 70 – Conveyor rotativo de bobines *(cortesia da Europac Kraft Viana)*

- 3 – Conveyor linear que efectua a pesagem das bobines individuais – Figura 71.

Figura 71 – Conveyor de pesagem de bobines *(cortesia da Europac Kraft Viana)*

- 4 – Conveyor linear em que são impressas, através de um robot, algumas informações directamente na bobine e onde é também colada por outro robot uma etiqueta com toda a informação necessária ao despacho da bobine para o cliente – Figura 72.

Figura 72 – Conveyor linear de impressão de bobines *(cortesia da Europac Kraft Viana)*

- 5 – Conveyor linear que transporta a bobine até ao ponto de queda gravítica até 8 – Figura 73.

Figura 73 – Conveyor linear de bobines *(cortesia da Europac Kraft Viana)*

- 6, 7 – Queda gravítica conferida por cilindro pneumático que desequilibra a bobine e que tira partido da forma cilíndrica da bobine, para a rodar até ao conveyor 8 paralelo ao 5 – Figura 74.

Figura 74 – Queda gravítica de bobines *(cortesia da Europac Kraft Viana)*

- 8 – Conveyor linear que recebe a bobine da descida gravítica, equilibra-a e encaminha-a para o conveyor seguinte - Figura 75.

Figura 75 – Conveyor de recepção de bobines *(cortesia da Europac Kraft Viana)*

- 9 – Conveyor linear onde é feita a cintagem da bobine - Figura 76.

Figura 76 – Conveyor de cintagem de bobines *(cortesia da Europac Kraft Viana)*

- 10, 11 – Conveyors lineares que funcionam como *buffers* até as bobines serem descidas até ao piso do armazém - Figura 77.

Figura 77 – Conveyor *buffer* de bobines *(cortesia da Europac Kraft Viana)*

- 12 – Descedor hidráulico que combina dois movimentos lineares por conveyor com movimento de elevação por cilindro hidráulico – Figura 78.

Figura 78 – Conveyor descedor de bobines *(cortesia da Europac Kraft Viana)*

- 13, 16 – Conveyors lineares de saída de bobines para armazém – Figura 79.

Figura 79 – Conveyors de descarga final de bobines *(cortesia da Europac Kraft Viana)*

- A partir daqui o material é transportado por empilhadores operados manualmente.

Aparentemente trata-se de um sistema razoavelmente complexo. Não obstante, toda a linha é controlada através de um único controlador e possui um sistema de supervisão sobre ele. Num nível mais elevado, esta linha é parte de um sistema integrado de gestão (Optivision) altamente dedicado à indústria papeleira, da empresa Honeywell. A ligação é feita através de um SFIS – Shop Floor Interface System – que serve de intermediário e tradutor de informação entre o PLC no chão-de-fábrica e o sistema de gestão.

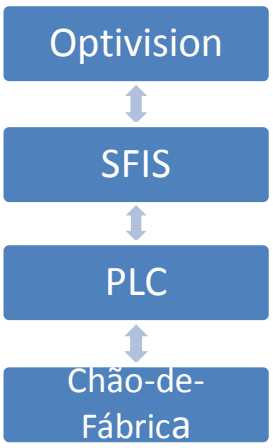


Figura 80 – Estruturação dos sistemas de gestão e controlo

No que diz respeito ao funcionamento do sistema, as bobines são posicionadas em pontos específicos dos conveyors para serem sujeitas aos diversos procedimentos da linha de acabamentos. Estes procedimentos podem ser vistos genericamente como uma acção executada sobre a bobine e não é relevante neste âmbito explicar os detalhes dos procedimentos em si. Todos os conveyors com a excepção dos conveyors 10 e 11 apenas permitem o transporte simultâneo de uma bobine.

Uma particularidade interessante é que as bobines não são posicionadas nos pontos de processamento com recurso a sensores indutivos ou barreiras fotoeléctricas, pois as bobines podem ter tamanhos diversos. O posicionamento é então garantido com a leitura de *encoders* existentes nos accionamentos eléctricos dos conveyors, e através de um algoritmo que calcula o número de impulsos necessários para uma bobine com um comprimento X ser posicionada num ponto Y.

A solução de automação actualmente existente é de certo modo modular. Cada conveyor é um bloco que contém o código necessário para este executar a sua função de transporte e posicionamento, mas a camada de informação não está encapsulada no bloco mas sim centralizada numa base de dados no controlador.

Cada bobine que entra na linha de acabamentos possui um grande número de variáveis que importa transmitir aos robots que imprimem informação e ao operador que, com um empilhador, no fim da linha transporta a bobine para um dado ponto do armazém.

A informação que entra no sistema juntamente com cada bobine é a seguinte: número de série da bobine, gramagem, peso, largura, diâmetro, tolerâncias dimensionais, ordem cliente, destino no armazém, quantidade de etiquetas, número de cintas, entre outras de menor relevância. Adicionalmente, após a medição de largura e pesagem é acrescentada a esta informação os valores reais destas grandezas. Cada bobine guarda a sua informação numa posição de uma fila de memória e esta é associada a cada etapa da linha de acabamentos e por consequência é associada a cada conveyor.

A Figura 81 mostra um exemplo de uma posição da fila, com os dados relativos a uma dada bobine.

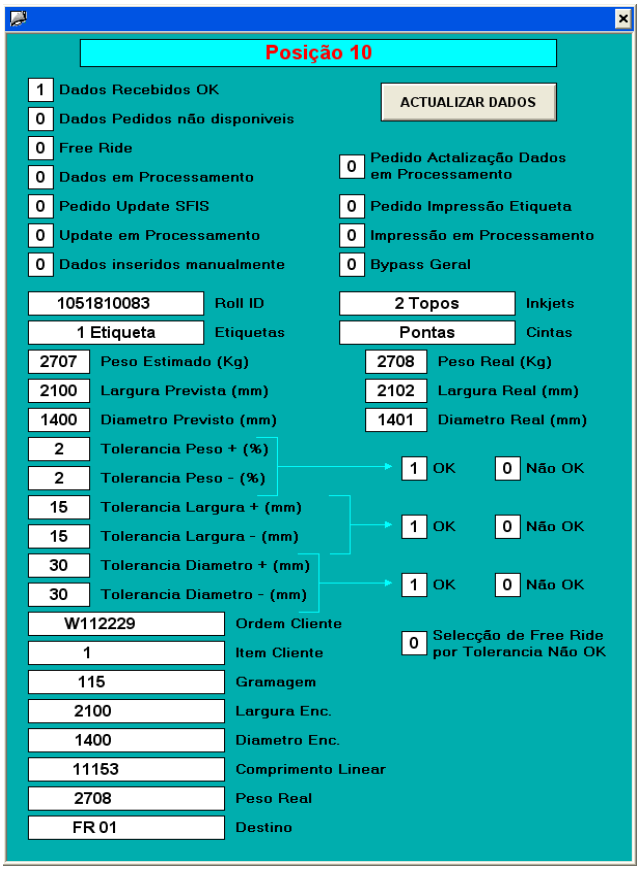


Figura 81 – Página do sistema de gestão de dados das bobines de papel

É notório que existe uma clara separação entre o controlo do movimento das bobines e a gestão da informação sobre elas. Uma parte

é o controlo de baixo nível que lê sensores, executa algoritmos e actualiza estados de motores e actuadores. Outra parte é a gestão dos pacotes de informação que definem cada bobine. As tarefas de actualização e comunicação da informação estão completamente abstraídas do funcionamento físico do sistema, independentemente disso o controlo centralizado sabe sempre onde se encontra cada bobine no sistema (com a assunção de correcto funcionamento).

Com a solução de controlo actual, se for substituído um conveyor por outro, o sistema comporta-se da mesma forma, pois do ponto de vista do controlador, o conveyor é um bloco de código bem definido que apenas se limita a executar a sua função – transportar bobines e posiciona-las por vezes em pontos específicos do seu curso. Porém, devido à informação das bobines ser centralizada, se for necessário alterar a disposição do sistema é necessária uma reconfiguração total da lógica. Ou seja a estrutura de software (nas duas camadas – física e de informação) não segue a disposição física dos equipamentos. A informação centralizada é clara inimiga da flexibilidade.

De modo objectivo, o sistema actual não segue por completo os padrões de desenho que esta dissertação explora. Com o fim de procurar uma possível solução realmente flexível seguem-se os princípios apresentados ao longo de todo o capítulo 5.

Subdivisão do sistema em módulos

Com base no que foi compreendido com a experimentação e após análise e entendimento do sistema em questão, a subdivisão ideal em módulos pode ser rapidamente encontrada.

- FB1 - Bloco funcional que encapsula o funcionamento de um conveyor que transporta uma bobine apenas;
- FB2 - Bloco funcional que encapsula o funcionamento de um conveyor que pode transportar *n* bobines;
- FB3 - Bloco funcional que encapsula o funcionamento do descedor de bobines.

Esta subdivisão bastante abrangente explica-se porque praticamente todos os conveyors executam a mesma função de transporte e posicionamento. Mesmo que a acção desenvolvida após a bobine estar posicionada seja diferente nos vários conveyors, estes apenas necessitam de dar a ordem para que essa acção ocorra e nada mais.

A razão fundamental para a distinção entre os dois primeiros blocos funcionais é a necessidade de no segundo bloco criar-se um *buffer* interno com as *n* caixas que possam entrar num conveyor. O facto de nos conveyors que transportam só uma bobine existir um processamento algures no seu trajecto também ajuda a justificar a subdivisão.

O descedor de bobines justifica-se ser um bloco funcional distinto, devido à sua especificidade de funcionamento e pelo agregar de movimentos diferentes inerentes ao transporte e não a outro tipo de processamento.

Cabe referir que o único ponto de criação de informação na base de dados é o sistema precedente que descarrega as bobines para a linha de acabamentos. Desta forma, não faz sentido pensar-se em blocos funcionais identificadores específicos como usado no Sorting.

Solução de Controlo

Devido à óbvia impossibilidade de implementar no terreno as soluções alcançadas, este capítulo limita-se à busca teórica da estrutura da solução de controlo que permita máxima flexibilidade ao sistema. A solução actualmente existente, já sumariamente descrita, serve de ponto de partida.

Os problemas encontrados devem-se especialmente ao não encapsulamento da informação das bobines nos conveyors. Ou seja, o maior foco de não flexibilidade é a centralização da informação. A aplicação dos padrões de desenho adaptados a este sistema particular procuram resolver esse problema.

Uma solução encontrada consiste em aplicar o mesmo conceito de passagem de informação entre conveyors que foi usada no Sorting. A diferença traduz-se no facto de em vez de se passar a informação num inteiro BOX_HEIGHT, passa-se um conjunto de variáveis ou até um agregado de toda informação de uma posição da base de dados numa simples *Word* – POSITION.

Cada bloco funcional seria constituído da seguinte forma:

1. **FB1 – Conveyor (uma bobine)**
    - Variáveis de entrada
        - ENCODER – leitura do *encoder* associado à rotação do motor de accionamento do tapete
        - INPUT_IN - Sensor fotoeléctrico de entrada
        - INPUT_EXIT - Sensor fotoeléctrico de saída
        - POSITION_IN – Entrada de toda a informação da bobine vinda da entrada do sistema ou do conveyor precedente
        - POSITION_UPDATE – Informação que o sistema de medição de largura e pesagem envia para que o conveyor actualize a informação correspondente
        - RECEIVE_BUSY – Entrada que indica que o próximo conveyor não está em condições de receber bobines.

- Variáveis de saída
  - OUTPUT – Accionamento do motor eléctrico que faz movimentar o conveyor.
  - PROCESS – Ordem para que seja executado o processamento (elevação, pesagem, rotação, impressão, etiquetagem, cintagem, etc.) após o correcto posicionamento da bobine
  - POSITION_OUT – Envio de toda a informação da bobine para o próximo conveyor
  - SEND_BUSY – Saída que indica ao conveyor precedente, que não se encontra em condições de receber bobines.

- Algoritmo interno
  - Implementa uma sequência de estados internos inspirados nos usados no Sorting, neste caso – Idle, Loading, Positioning, Ready, Processing, Transporting e Unloading
  - Implementa funções que permitam a leitura do encoder e o consequente posicionamento das bobines
  - A ordem de processamento é despoletada pelo associar da variável PROCESS ao estado Ready

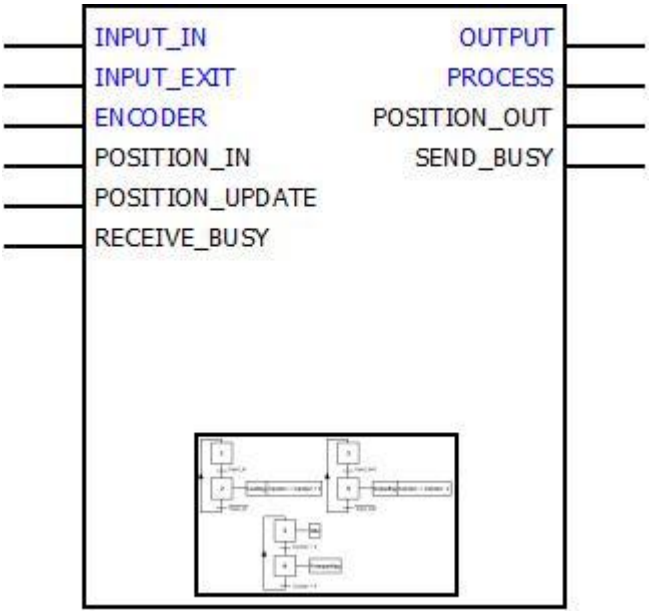


Figura 82 – Bloco funcional de conveyors de transporte e processamento de uma só bobine

2. **FB2 – Conveyor (*n* bobines)**
    - Em tudo semelhante ao FB1, com a alteração de que possui variáveis internas para acomodar mais do que uma posição da base de dados de bobines – implementa um *buffer* do tipo FIFO
    - Não necessita da variável de entrada POSITION_UPDATE nem da variável de saída PROCESS, pois nestes conveyors não são executadas acções para além do seu transporte.

À excepção da ausência das variáveis POSITION_UPDATE e PROCESS, a sua representação é muito similar à Figura 82, pois as restantes diferenças são apenas no código interno.

3. **FB3 – Descedor de bobines**
    - Variáveis de entrada
        - ENCODER – leitura do *encoder* associado à rotação do motor de accionamento do tapete
        - INPUT_IN - Sensor fotoeléctrico de entrada
        - INPUT_EXIT - Sensor fotoeléctrico de saída
        - INPUT_HIGH – Fim de curso superior do cilindro hidráulico que efectua o movimento de elevação
        - INPUT_LOW – Fim de curso inferior do cilindro hidráulico que efectua o movimento de elevação
        - POSITION_IN – Entrada de toda a informação da bobine
        - RECEIVE_BUSY – Entrada que indica que o próximo conveyor não está em condições de receber bobines.
    - Variáveis de saída
        - OUTPUT_LINEAR1 – Accionamento do motor eléctrico do primeiro movimento linear
        - OUTPUT_DOWN – Accionamento da electroválvula de comando do cilindro hidráulico para que este desça
        - OUTPUT_UP - Accionamento da electroválvula de comando do cilindro hidráulico para que este suba
        - OUTPUT_LINEAR2 – Accionamento do motor eléctrico do segundo movimento linear (após descida, descarregar bobine para os conveyors de saída)
        - OUTPUT_DOOR – Ordem de fecho pneumático de porta de segurança entre o descedor e o conveyor precedente (deveriam ser definidas

variáveis de entrada para os fins de curso deste actuador pneumático, contudo por simplificação tal não foi feito)

- POSITION_OUT – Saída de toda a informação da bobine
- SEND_BUSY – Saída que indica ao conveyor precedente, que não se encontra em condições de receber bobines.

- Algoritmo interno
  - Semelhante aos outros blocos
  - Encadeamento das várias etapas internas – Idle, Loading, Lowering, Unloading e Rising.

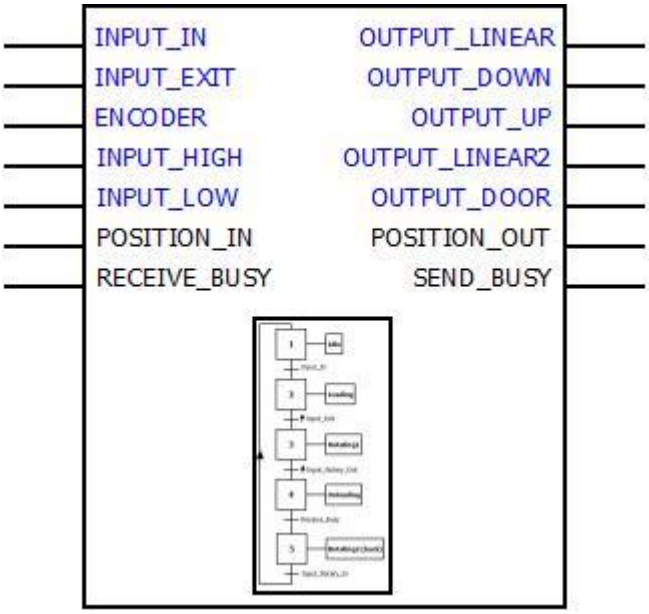


Figura 83 – Bloco funcional do descedor de bobines

Com a interligação destes blocos (através de POSITION_OUT a POSITION_IN) e subsequente estruturação, mimetizando a disposição física dos vários conveyors, é possível em tese alcançar-se uma solução com um bom compromisso entre flexibilidade e facilidade de configuração.

O processamento externo ao conveyor pode ser o processo de elevação, rotação, pesagem, impressão, desequilibro, equilíbrio e cintagem. Se fosse necessário criar um bloco funcional diferente para cada conveyor que sofre um destes processos a configuração poderia ser mais simples mas a reutilização de código não era eficiente e ficaríamos com uma biblioteca demasiado grande de blocos e de aplicação restrita.

Convém mencionar que num sistema tão específico e dedicado quanto este, o conceito de flexibilidade não é o factor chave de uma solução de controlo, pois devido à forte integração com o Optivision e o SFIS, qualquer alteração ao nível mais baixo de controlo implica uma considerável reestruturação superior. Porém a validade conceptual de aplicação de uma solução é universal, o que confere validade à abordagem consumada.

Para legitimação final dos padrões de desenho propostos para este sistema, pode-se imaginar o seguinte cenário: o sistema de transporte e acabamento de bobines passará a ser alimentado por bobines oriundas de duas bobinadoras a montante. A segunda bobinadora estaria posicionada do lado oposto do conveyor rotativo. Seriam então necessários mais dois conveyors para espelhar o trajecto original das bobines - Figura 84. Esses dois conveyors, seguindo os padrões enunciados, seriam integrados na solução de controlo simplesmente com a introdução de duas instâncias do FB1 devidamente configuradas. Porém convém realçar que nem tudo se revelaria trivial, pois seria necessária uma considerável alteração no conveyor rotativo. Este passaria a movimentar-se linearmente em dois sentidos e teria que estar preparado para receber a informação do sistema de supervisão sobre a escolha da bobine a processar. Em todo o caso, e após reunir opiniões diversas, a rapidez da implementação da alteração da solução de controlo seguindo os padrões de desenho, seria presumivelmente na ordem das duas vezes mais célere em relação à solução actual.

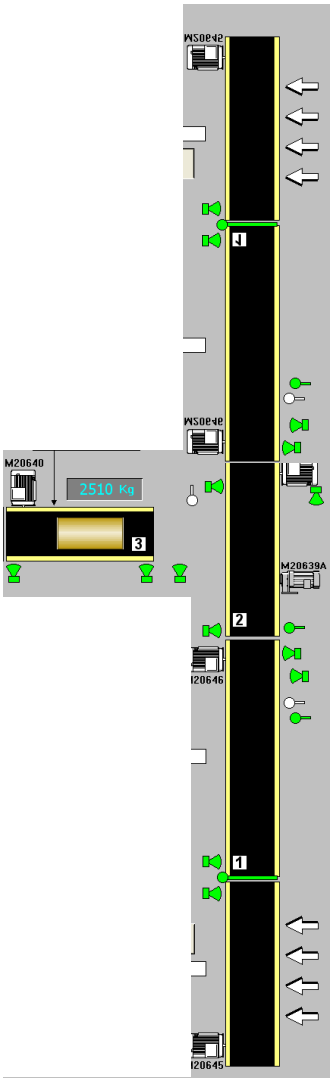


Figura 84 – Cenário hipotético de alteração do layout de conveyors

Traduzindo o tempo ganho para a unidade que qualquer indústria mais preza – o Euro - chegam-se aos seguintes valores.

Uma inversão fabril deste nível é sempre realizada numa paragem da instalação programada anualmente, e convém relembrar que sendo a Europac Kraft Viana uma fábrica de laboração contínua, qualquer hora de produção é valiosa. Seguindo os valores de 2010, em que a produção anual foi de 360.000 toneladas de papel, por hora são produzidas sensivelmente 41 toneladas de papel. O preço médio do papel kraft em 2010 foi cerca de 400€/ton, o que indica que por cada hora de funcionamento da máquina de papel são facturados 16.400€ (a produção é feita com base em encomendas e não para stock).

Com a solução de controlo actual, assumindo prontidão da montagem mecânica e eléctrica dos conveyors e respectivos sensores e a devida alteração prévia do software em modo offline, o tempo de comissionamento estima-se que seja 48 horas de trabalho. Com a aplicação dos padrões de desenho enunciados, este tempo poderia decrescer para as 24h de trabalho. O que faria com que a produção reiniciasse 24 horas antes do originalmente previsto. Este avanço traduzir-se-ia em 393.600€ de facturação adicional de produção (não considerando sequer a melhoria produtiva que a alteração concebida traria desde logo).

Posto isto, é fácil entender a aplicabilidade de uma solução flexível e facilmente reconfigurável, que garanta melhores prazos de execução de alterações no processo produtivo.

6. Investigação Experimental – Sistema Batching

A investigação experimental prossegue adoptando as pretensões do capítulo anterior, desta feita para um sistema bastante diferente, um sistema de mistura de tintas do tipo batching.

Algum do esforço elucidativo das várias etapas até chegar-se à solução desejada já foi feito no capítulo anterior, pelo que as abordagens apresentadas neste capítulo pretendem ser mais directas, partindo do princípio de algum conhecimento já adquirido.

O sistema escolhido está igualmente presente no software de sistemas virtuais de treino ITS PLC e foram conduzidas abordagens variadas para alcançar uma solução flexível de controlo de sistemas de batching genéricos.

6.1. Análise do Sistema e Identificação de Equipamentos e Processos

Um sistema de batching genérico é caracterizado pela mistura e processamento de uma dada quantidade de produtos miscíveis. O sistema Batching de tintas do ITS PLC é constituído da seguinte forma (Figura 85).

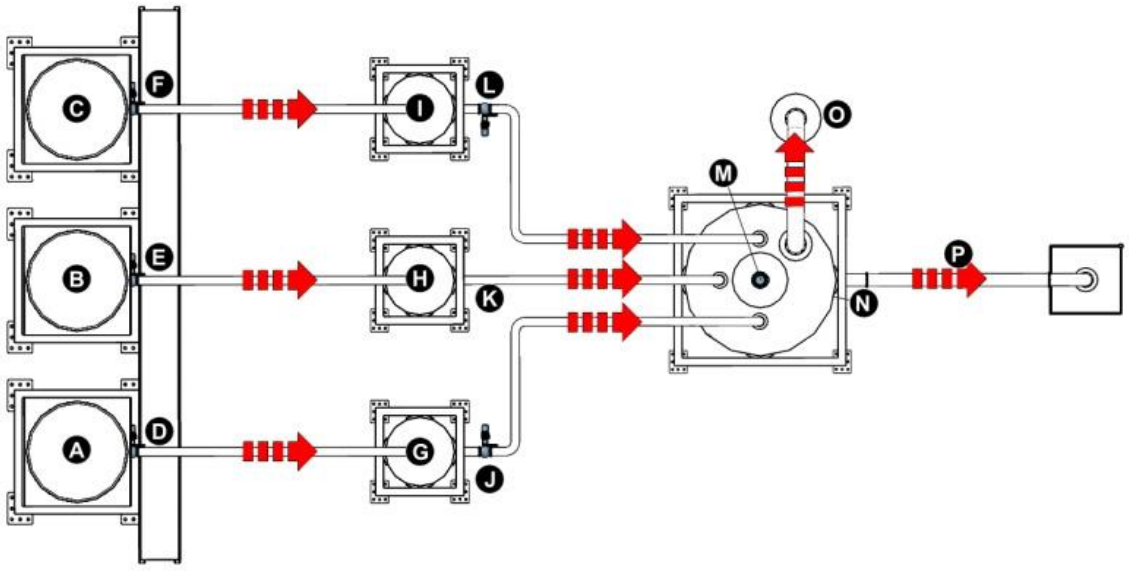


Figura 85 – Planta do sistema Batching do ITS PLC *(http://www.realgames.pt)*

A, B, C – Reservatórios de tinta vermelha, verde e azul respectivamente

D, E, F – Válvulas que controlam a saída de tinta de cada reservatório

G, H, I – Doseadores de tinta vermelha, verde e azul respectivamente

J, K, L – Válvulas que controlam a saída de tinta de cada doseador

M – Tanque de mistura de tintas

N – Válvula de saída de tinta do tanque de mistura

Os pontos O e P são meramente uma saída de transbordo de tinta e um visor transparente que permite ver a cor da tinta expedida respectivamente.

O objectivo global do sistema é a produção de diversas cores de tintas através da mistura das três cores primárias (vermelho, verde e azul). Para isso há que dosear cada tinta em quantidades bem definidas para depois serem correctamente misturadas. As quantidades de dosagem são medidas através de 3 sensores em cada doseador (vazio, meia dose e dose completa). Assume-se que os reservatórios de tinta possuem sempre tinta pronta a ser doseada.

Sugere-se que para uma descrição mais detalhada do sistema, seja consultada a informação em anexo ou visitado o site www.realgames.pt.

Figura 86 – Tanque doseador do sistema Batching

Numa primeira análise distinguem-se claramente dois processos vitais e que estão encadeados na sequência de processamento - doseamento e mistura.

A nível de accionamento, este resume-se ao comando de 7 válvulas On/Off – uma na saída de cada reservatório (3), doseador (3) e tanque de mistura (1) – e ao comando do motor do misturador de tinta.

6.2. Subdivisão Inicial do Sistema em Módulos

Os módulos que resultam da subdivisão deste sistema podem ser interpretados como as diferentes acções que são efectuadas nas tintas, e só existem na realidade duas acções básicas – dosear e misturar. Tudo que é necessário efectuar em cada uma dessas acções deve estar encapsulado no bloco funcional do equipamento – encher, processar e esvaziar.

A Figura 87 ilustra de forma simplificada o sistema. A parte representada a azul repete-se três vezes (uma para cada tinta). A parte representada a verde é única e está interligada a cada bloco a montante. Fica claro que estas divisões podem dar origem a dois blocos funcionais. Pode-se ainda tentar criar um bloco funcional único que possa ser instanciado para qualquer equipamento.

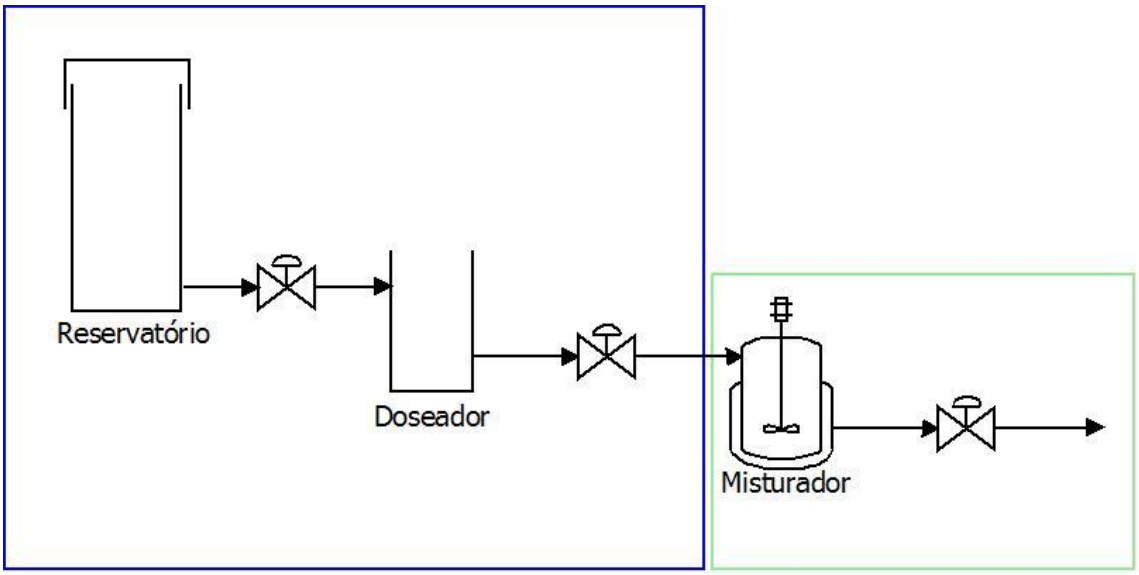


Figura 87 – Hipótese 1 para a subdivisão inicial do sistema

A Tabela 7 resume as hipóteses iniciais consideradas para a subdivisão do sistema em módulos.

Tabela 7 – Hipóteses iniciais de subdivisão do sistema, tendo em conta os dois equipamentos fundamentais

Equipamento	**Hipóteses Iniciais de Subdivisão do Sistema**	
	1	2
Doseador (Dosing)	FB1	FB1
Misturador (Mixing)	FB1	FB2

A questão das fronteiras de cada módulo é de extrema importância num sistema de batching. Cada equipamento (seja doseador ou misturador) possui uma válvula a montante (enchimento) e a jusante (descarga). Estas válvulas deverão ficar encapsuladas em que bloco? Ou poderão até ser blocos separados? Onde acaba o bloco do doseador e começa o bloco do misturador? São tudo questões que a solução procurada tenta responder.

6.3. Solução Inicial de Controlo

Dada a relativa simplicidade do sistema, não existe a necessidade de aplicar a solução de controlo de forma faseada. É possível logo em primeira abordagem testar-se o controlo do sistema como um todo, tendo por base as hipóteses apresentadas.

Antes de se descobrir qual a melhor subdivisão e definir qual a estrutura da solução, importa observar como devem ser implementados os blocos funcionais e como devem ser delimitados.

Doseador

Tomando por exemplo o equipamento doseador da hipótese 2, este pode ter as seguintes quatro opções de definição de limites (Figura 88).

A – Engloba a válvula de enchimento e a de esvaziamento

B – Engloba apenas a válvula de esvaziamento

C – Engloba apenas a válvula de enchimento

D – Apenas o doseador

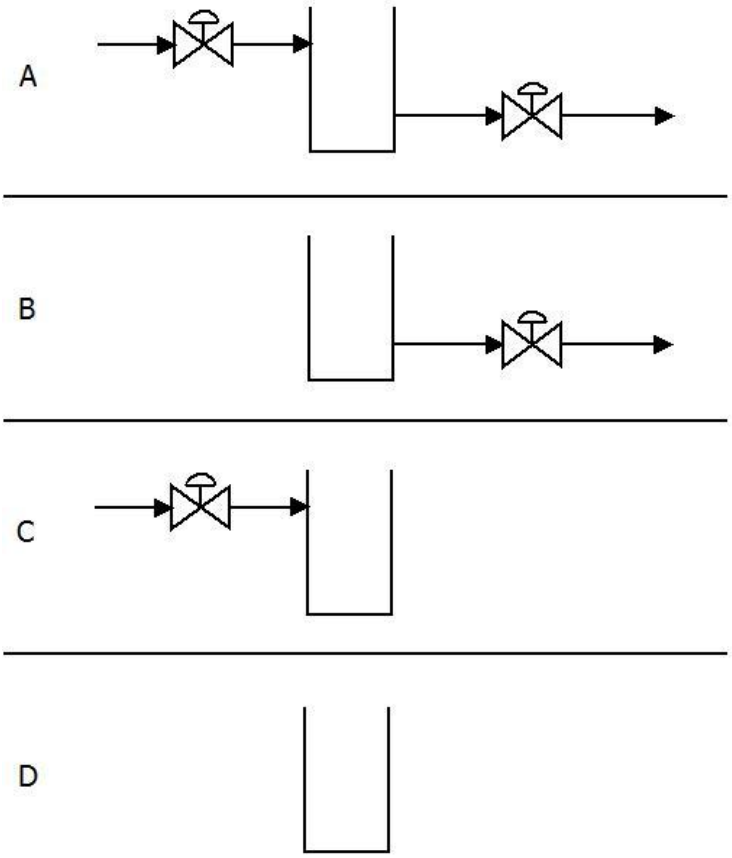


Figura 88 – Limites possíveis do equipamento doseador

Para melhor entender qual das hipóteses de limitação é mais interessante, convém entender-se o princípio por detrás da lógica de um doseador.

A lógica de um doseador genérico resume-se na mais simples das formas às seguintes três etapas.

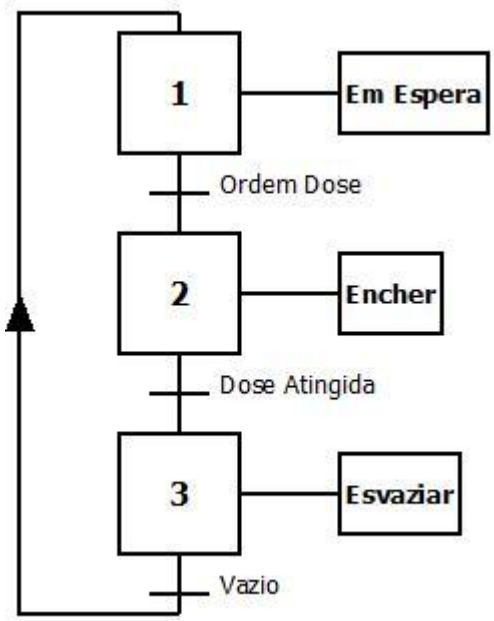


Figura 89 – GRAFCET base para um doseador genérico

Com base nisto, é acertado encapsular as acções de enchimento e esvaziamento no próprio bloco Doseador em si, pois as acções de controlo principais do ponto de vista do doseador são na realidade abrir e fechar as válvulas, a montante e a jusante (Figura 90).

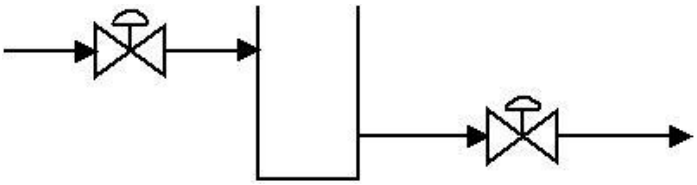


Figura 90 – Limite escolhido para o equipamento doseador

Misturador

Os limites do bloco Misturador podem ser vistos da mesma forma que os do doseador.

A – Engloba a válvula de enchimento e a de esvaziamento

B – Engloba apenas a válvula de esvaziamento

C – Engloba apenas a válvula de enchimento

D – Apenas o misturador

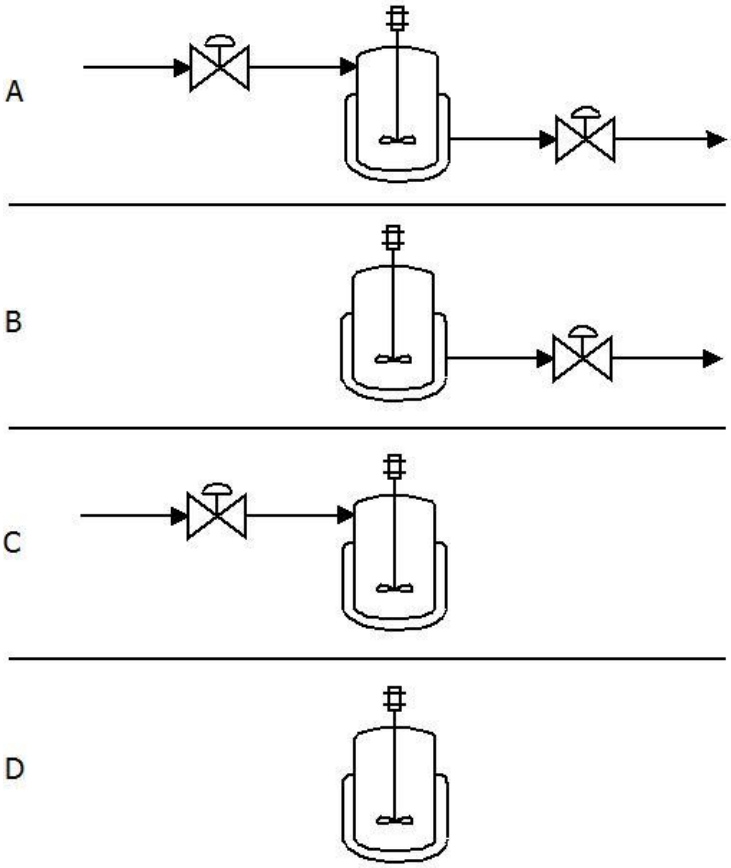


Figura 91 – Limites possíveis para o equipamento misturador

A lógica de um misturador pode-se considerar semelhante a um doseador, com o acréscimo de um estado de processamento (mistura) entre o encher e o esvaziar.

À partida, a solução A seria novamente a mais acertada, contudo se pensarmos que o misturador está encadeado com um ou mais doseadores, chega-se à conclusão que desta forma a válvula de enchimento acaba por ser a válvula de saída de cada doseador.

Para evitar complicações na configuração das instâncias dos blocos, é melhor opção que o misturador apenas controle a válvula de saída.

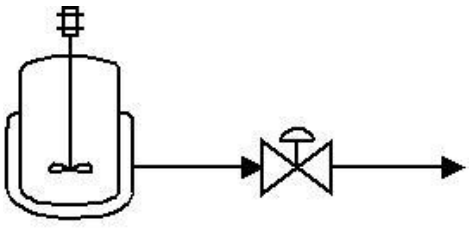


Figura 92 – Limite escolhido para o equipamento misturador

Esta opção revela-se fulcral pois limita a hipótese inicial de subdivisão número 1, dado que se só existe um bloco para todos os equipamentos, seria necessário que o bloco tivesse variáveis de

configuração que permitissem definir se o equipamento tem válvulas a montante, a jusante ou em ambos os lados. Esta decisão pode suscitar discussão, visto a semelhança conceptual entre um doseador e um misturador, contudo com o pensamento de alcançar soluções de simples implementação e configuração, a hipótese escolhida é a número 2 – um bloco doseador e um bloco misturador

Integração Horizontal

Com base na subdivisão escolhida e nas delimitações seleccionadas para os blocos funcionais, como primeira abordagem, a integração horizontal entre os vários blocos resume-se à definição das seguintes variáveis e consequente ligação.

```
CNF_COMPLETE:BOOL; (*Confirmação de término do doseamento*)

REQ_MIX:BOOL; (*Pedido de mistura*)
```

```
Mixing1 (REQ_MIX:= DosingRed.CNF_COMPLETE AND DosingGreen.CNF_COMPLETE AND DosingBlue.CNF_COMPLETE);
```

Só é necessário ligar o misturador aos doseadores. O misturador inicia a sua acção de mistura assim que todos os doseadores informem que completaram a sua função – dosear e esvaziar. Por outras palavras, a variável de entrada REQ_MIX é associada à variável de saída dos doseadores CNF_COMPLETE.

Mesmo que uma dada mistura não inclua determinada cor de tinta, como se verá mais adiante, a variável CNF_COMPLETE está associada ao estado inactivo do doseador, pelo que se o doseador estiver inactivo significa que já completou a sua dosagem ou que não recebeu sequer ordem de produção de tinta.

Isto significa que o sistema comporta-se como um sistema push, na medida em que o pedido de mistura de tinta é dado dos doseadores para o misturador.

Integração Vertical

A integração vertical neste caso é algo essencial e indissociável do funcionamento do sistema. No sistema de sorting anterior, o objectivo do sistema era intrínseco ao funcionamento (transportar caixas e separa-las), neste sistema o objectivo é fabricar tintas, mas alterando o seu funcionamento podem ser fabricadas uma grande quantidade de tintas, pelo que há necessidade de ordens de produção bem definidas.

Os blocos funcionais necessitam de receber as ordens através de variáveis de entrada e precisam de informar o seu estado, ou pelo menos o seu término de actividade, através de variáveis de saída.

As variáveis de entrada necessárias definir no bloco doseador, de forma a se efectuar devidamente a integração vertical, são as seguintes.

```
REQ_DOSE:BOOL; (*Pedido para que seja processada a Dose*)
REQ_DELIVERY:BOOL; (*Pedido para que seja entregue a dose*)
DOSE:BOOL; (*Definição de quantidade -> 1 DOSE = TRUE | 1/2 DOSE = FALSE*)
```

O booleano de entrada REQ_DOSE é a ordem de comando para que seja processada uma dada quantidade de tinta. Essa quantidade é definida com o booleano de entrada DOSE. Como os doseadores em questão apenas podem possuir duas quantidades de tintas (uma dose ou meia dose), uma variável booleana é suficiente. A variável REQ_DELIVERY efectua o pedido de entrega da dose para o misturador. Nesta primeira abordagem a ordem REQ_DELIVERY é dada externamente pelo utilizador, contudo pode-se também pensar numa solução em que essa ordem seja dada automaticamente pelo funcionamento do sistema (ou seja através de integração horizontal com o misturador).

O bloco doseador nesta abordagem inicial não possui qualquer variável de saída para integração vertical.

As variáveis de entrada a nível de integração vertical no bloco misturador são a definição do tempo de mistura e o pedido de entrega de tinta. O funcionamento do bloco misturador revelará que a entrega de tinta dá-se automaticamente após o tempo de mistura definido pelo inteiro MIX_TIME. Adicionalmente, a variável REQ_DELIVERY permite que a todo o momento o tanque misturador seja esvaziado.

```
REQ_DELIVERY: BOOL;(*Pedido de entrega, independentemente do tempo de mistura*)
MIX_TIME:INT; (*Tempo de mistura*)
```

O bloco misturador, sendo sempre o bloco final do sistema, deve informar através da variável de saída CNF_COMPLETE, que terminou a sua produção.

```
CNF_COMPLETE:BOOL; (*Confirmação de término da mistura*)
```

Note-se que esta variável também poderia ser usada horizontalmente para encadear dois misturadores.

Com a integração definida entre blocos e verticalmente, importa de seguida definir com detalhe como são compostos os blocos e quais os algoritmos internos que permitem que a sua aplicação responda às necessidades do sistema. A lógica interna implementada baseia-se nas soluções presentes no livro (Magalhães, 2010).

Blocos Funcionais

A declaração de variáveis do bloco doseador é a seguinte. As variáveis de entrada são os vários sensores de nível (LOW_LEVEL, MID_LEVEL e HIGH_LEVEL) e as já referidas variáveis de ligação REQ_DOSE, REQ_DELIVERY e DOSE.

As variáveis de saída são o comando das válvulas a montante e a jusante do doseador (UPSTREAM_VALVE e DOWNSTREAM_VALVE) e a variável de ligação ao misturador CNF_COMPLETE. Todas estas variáveis são booleanas.

No que respeita a variáveis internas, além da definição de um *trigger* necessário ao arranque do equipamento, existem os estados internos Inactivo, Enche, Cheio e Esvazia.

A variável interna Dose_1 é uma cópia da variável de entrada Dose, para garantir que durante um processo de doseamento, caso surja uma nova ordem de produção, esta não comprometa a produção em curso.

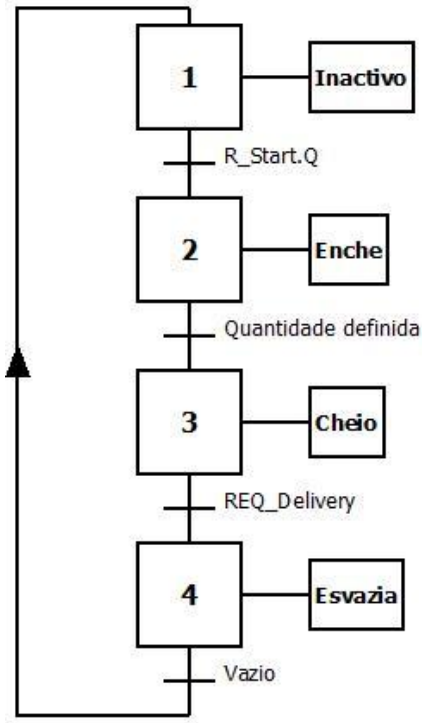


Figura 93 – GRAFCET da lógica interna do bloco doseador

O algoritmo interno implementa o controlo sequencial de estados representado na Figura 93, e pode ser representado pelo seguinte bloco funcional.

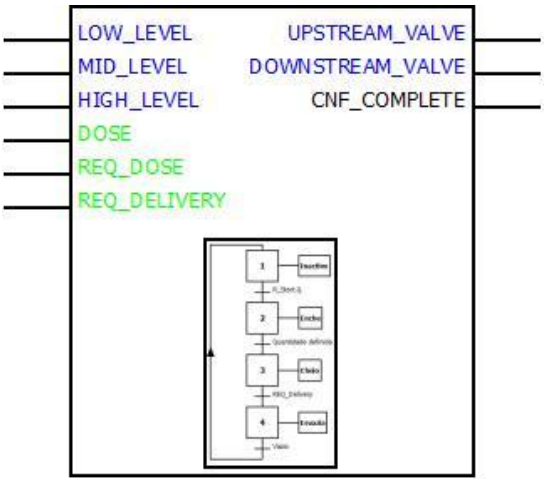


Figura 94 – Bloco funcional do doseador

```
FUNCTION_BLOCK DOSING
VAR_INPUT
(*Variáveis Físicas*)
    LOW_LEVEL:BOOL; (*Sensor de nível baixo*)
    MID_LEVEL:BOOL; (*Sensor de nível médio*)
    HIGH_LEVEL:BOOL; (*Sensor de nível alto*)

(*Variáveis Software*)
    REQ_DOSE:BOOL; (*Pedido para que seja processada a Dose*)
    REQ_DELIVERY:BOOL; (*Pedido para que seja entregue a dose*)
    DOSE:BOOL; (*Definição de quantidade -> 1 DOSE = TRUE | 1/2 DOSE = FALSE*)
END_VAR
VAR_OUTPUT
(*Variáveis Físicas*)
    UPSTREAM_VALVE:BOOL; (*Comando da válvula a montante*)
    DOWNSTREAM_VALVE:BOOL; (*Comando da válvula a jusante*)

(*Variáveis Software*)
    CNF_COMPLETE:BOOL; (*Confirmação de término do doseamento*)
END_VAR
VAR
(*Triggers*)
    R_Start: R_TRIG; (* Transição ascendente do Start*)

(*Estados Internos*)
    Inactivo  : BOOL := TRUE;  (*Estado de doseador Idle*)
    Enche  : BOOL := FALSE; (* Estado de doseador a encher*)
    Cheio :BOOL :=FALSE; (*Estado de dosesador cheio*)
    Esvazia : BOOL := FALSE; (* Estado de doseador a esvaziar*)

(*Varíaveis Auxiliares*)
    Dose_1 : BOOL; (* Cópia da variável Dose, para garantir que mesmo que seja dada uma ordem
                      durante o processamento de outra, esta não é comprometida *)
END_VAR
```

```
(*Inicialização de Edge Detections*)
R_Start (CLK :=REQ_DOSE);

(*Definição de quantidade*)
IF R_Start.Q THEN
Dose_1 := Dose;
END_IF;

(*Controlo de Estados*)
IF Inactivo AND R_Start.Q THEN (* Quando é requisitada tinta *)
    Inactivo := FALSE;
    Enche := TRUE;
END_IF;

IF Enche AND (HIGH_LEVEL OR MID_LEVEL AND NOT Dose_1) THEN
(* Quando a quantidade desejada de tinta é atingida *)
    Enche := FALSE;
    Cheio := TRUE;
END_IF;

IF Cheio AND REQ_DELIVERY THEN (* Quando há autorização para descarregar *)
    Cheio:= FALSE;
    Esvazia := TRUE;
END_IF;

IF Esvazia AND NOT LOW_LEVEL THEN (* Quando fica vazio *)
    Esvazia := FALSE;
    Inactivo := TRUE;
END_IF;

(* Afectação das Saídas *)
UPSTREAM_VALVE := Enche;
DOWNSTREAM_VALVE := Esvazia;
CNF_COMPLETE := Inactivo;
```

A sequência de estados é facilmente compreensível e a afectação de saídas é feita directamente através das variáveis de estado, com especial atenção para a afectação da saída CNF_COMPLETE, pois esta é uma cópia do estado Inactivo. Isto significa que para uma dada produção de tinta, o doseador informa o misturador que terminou a sua produção ou que a tinta a produzir não necessita de uma determinada cor (estado Inactivo é o estado por defeito do doseador).

Em relação ao bloco misturador, a sua declaração de variáveis é a seguinte. As variáveis de entrada são os seus sensores de nível (LOW_LEVEL e HIGH_LEVEL) e as suas variáveis de integração vertical e com outros blocos (REQ_MIX, REQ_DELIVERY e MIX_TIME).

As suas variáveis de saída são o comando da válvula a jusante (DOWNSTREAM_VALVE) e o comando do accionamento do misturador (MIXER). Existe a variável de integração CNF_COMPLETE que informa quando a função do misturador chega ao seu fim.

As variáveis internas do bloco são as habituais variáveis de estado (Inactivo, Enche, Mistura e Descarga), o *trigger* de início de funcionamento e o temporizador que gere o tempo de mistura. Este temporizador é configurado através da variável de entrada MIX_TIME_1, que é uma cópia interna da variável de entrada MIX_TIME. Os motivos desta cópia são os mesmos que foram apresentados para justificar a existência da variável Dose_1 no bloco doseador.

O bloco funcional implementa o algoritmo interno do misturador que segue o seguinte diagrama de estados.

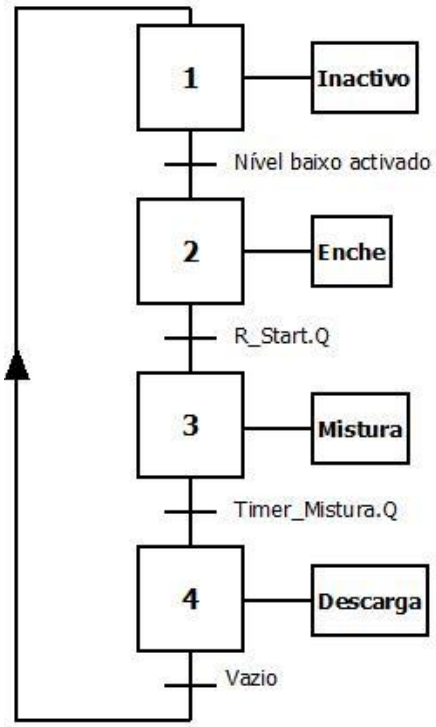


Figura 96 – GRAFCET da lógica interna do bloco misturador

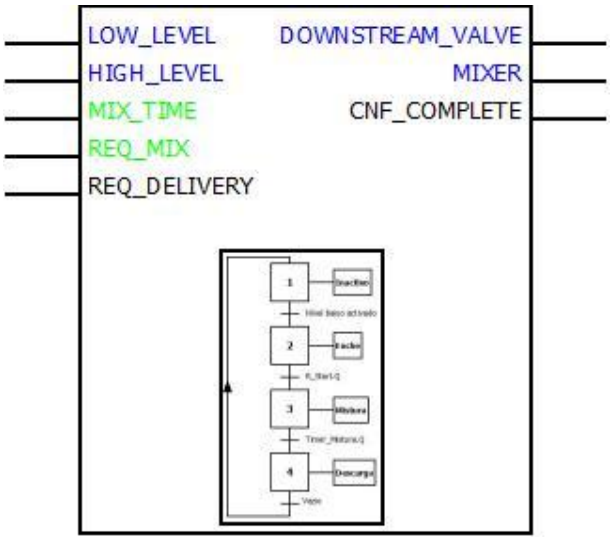


Figura 95 – Bloco funcional do bloco misturador

```
FUNCTION_BLOCK MIXING
VAR_INPUT
(*Variáveis Físicas*)
    LOW_LEVEL:BOOL; (*Sensor de nível baixo*)
    HIGH_LEVEL:BOOL; (*Sensor de nível alto*)

(*Variáveis Software*)
    REQ_MIX:BOOL; (*Pedido de mistura*)
    REQ_DELIVERY: BOOL;(*Pedido de entrega, independentemente do tempo de mistura*)
    MIX_TIME:INT; (*Tempo de mistura*)
END_VAR
VAR_OUTPUT
(*Variáveis Físicas*)
    DOWNSTREAM_VALVE:BOOL; (*Comando da válvula a jusante*)
    MIXER:BOOL; (*Comando do accionamento do misturador*)

(*Variáveis Software*)
    CNF_COMPLETE:BOOL; (*Confirmação de término da mistura*)
END_VAR
VAR
(*Triggers*)
    R_Start: R_TRIG; (* Transição ascendente do Start*)

(*Timers*)
    Timer_mistura : TON; (* Temporizador On-delay *)
```

```
(*Estados Internos*)
    Inactivo  : BOOL := TRUE; (* Estado de misturador idle*)
    Enche  : BOOL := FALSE; (* Estado de misturador a encher*)
    Mistura : BOOL := FALSE; (* Estado de misturador a misturar*)
    Descarga : BOOL := FALSE; (* Estado de misturador a descarregar*)

(*Variaveis Auxiliares*)
    Mix_Time_1: INT;(* Cópia da variável Mix_Time, para garantir que mesmo que seja dada uma ordem
                       durante o processamento de outra, esta não é comprometida *)
END_VAR
```

Relembrando a Figura 92, esta ilustra bem que o processo de enchimento do misturador depende da abertura da válvula a jusante, e esta não se encontra definida neste bloco mas sim no bloco do doseador. Sendo assim a transição do estado Inactivo para o estado Enche é dada simplesmente pela activação do sensor de nível inferior do tanque misturador. Quando entrar qualquer quantidade de tinta, o estado do misturador altera-se para Enche. O que de seguida despoleta a acção de mistura é o *trigger* R_Start, sendo este por sua vez actuado pela variável de entrada REQ_MIX. Como já foi explicado aquando da elucidação da integração horizontal, o REQ_MIX é interligado às variáveis dos três doseadores que informam que acabaram de despejar a tinta por completo.

```
(*Inicialização de Edge Detections*)
R_Start (CLK :=REQ_MIX);

(*Definição do tempo de mistura*)
IF R_Start.Q THEN
Mix_Time_1 := MIX_TIME;
END_IF;

(*Controlo de Estados*)
IF Inactivo AND LOW_LEVEL THEN (* Quando começa a receber tinta *)
    Inactivo := FALSE;
    Enche := TRUE;
END_IF;

IF Enche AND R_Start.Q THEN (* Quando recebe informação que foi entregue toda a tinta *)
    Enche := FALSE;
    Mistura := TRUE;
END_IF;

IF Mistura AND Timer_mistura.Q THEN (* Ao fim do tempo de mistura *)
    Mistura := FALSE;
    Descarga := TRUE;
END_IF;

IF Descarga AND NOT LOW_LEVEL THEN (* Quando vazio *)
    Descarga := FALSE;
    Inactivo := TRUE;
END_IF;

(*Afectação das Saídas*)
DOWNSTREAM_VALVE:=Descarga OR REQ_DELIVERY;
MIXER:=Mistura;
CNF_COMPLETE := Inactivo;

(*Temporizador com base no tempo de mistura definido*)
Timer_mistura(IN := Mistura, PT := INT_TO_TIME (Mix_Time_1));
```

Agora que foram apresentados os dois blocos funcionais que servem de *bulding blocks* para a primeira abordagem da solução de controlo, de forma a controlar o sistema como um todo é então necessário instanciar os dois blocos, com vista a ter cada equipamento controlado por um bloco funcional devidamente configurado.

```
PROGRAM Main
VAR
    DosingRed:DOSING;
    DosingGreen:DOSING;
    DosingBlue:DOSING;
    Mixing1:MIXING;
END_VAR
```

A configuração consiste em associar os respectivos sensores e actuadores dos equipamentos com os blocos funcionais e configurar as integrações horizontais e verticais.

```
(*Doseador Vermelho*)
DosingRed (REQ_DOSE:= In_12); (*Simulação de pedido de doseamento de tinta*)
DosingRed (REQ_DELIVERY := In_14); (*Simulação de pedido de entrega de tinta doseada*)
DosingRed (LOW_LEVEL:= In_0);
DosingRed (MID_LEVEL :=In_1);
DosingRed (HIGH_LEVEL:=In_2);
DosingRED (DOSE := TRUE); (*Simuação de pedido de dose completa*)
Out_0 := DosingRed.UPSTREAM_VALVE;
Out_1 := DosingRed.DOWNSTREAM_VALVE;

(*Doseador Verde*)
DosingGreen (REQ_DOSE:= In_12); (*Simulação de pedido de doseamento de tinta*)
DosingGreen (REQ_DELIVERY := In_14); (*Simulação de pedido de entrega de tinta*)
DosingGreen (LOW_LEVEL:= In_3);
DosingGreen (MID_LEVEL :=In_4);
DosingGreen (HIGH_LEVEL:=In_5);
DosingGreen (DOSE := FALSE); (*Simuação de pedido de meia dose*)
Out_2 := DosingGreen.UPSTREAM_VALVE;
Out_3 := DosingGreen.DOWNSTREAM_VALVE;

(*Doseador Azul*)
DosingBlue (REQ_DOSE:= In_12); (*Simulação de pedido de doseamento de tinta*)
DosingBlue (REQ_DELIVERY := In_14); (*Simulação de pedido de entrega de tinta*)
DosingBlue (LOW_LEVEL:= In_6);
DosingBlue (MID_LEVEL :=In_7);
DosingBlue (HIGH_LEVEL:=In_8);
DosingBlue (DOSE := TRUE); (*Simuação de pedido de dose completa*)
Out_4 := DosingBlue.UPSTREAM_VALVE;
Out_5 := DosingBlue.DOWNSTREAM_VALVE;

(*Misturador*)
Mixing1 (REQ_MIX:= DosingRed.CNF_COMPLETE AND DosingGreen.CNF_COMPLETE AND DosingBlue.CNF_COMPLETE);
Mixing1 (MIX_TIME := 5500);
Mixing1 (LOW_LEVEL:= In_9);
Mixing1 (HIGH_LEVEL := In_10);
Out_6 := Mixing1.MIXER;
Out_7 := Mixing1.DOWNSTREAM_VALVE;
```

Notar que as variáveis de entrada dos doseadores REQ_DOSE e REQ_DELIVERY são controladas manualmente através do accionamento de botoneiras existentes no ITS PLC (entradas In_12 e In_14). A presente abordagem necessita que a cor da tinta a produzir seja configurada

directamente no código de configuração através da definição da variável DOSE em cada doseador. A integração vertical que permite que a cor a produzir seja facilmente definida, será alcançada numa próxima abordagem. A variável de entrada do misturador MIX_TIME é também definida através de configuração inicial por código (neste caso 5500ms).

Os blocos funcionais ligados entre si para controlar o sistema apresentam a seguinte forma.

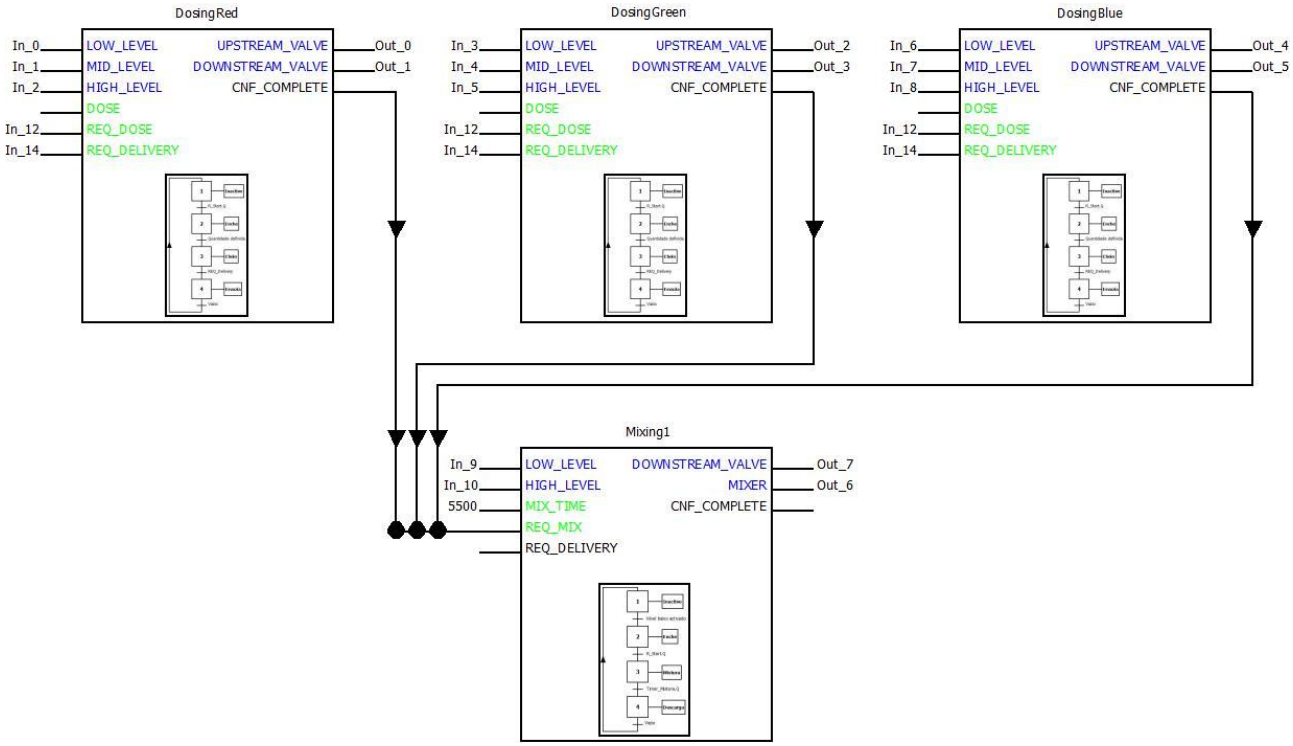


Figura 97 – Diagrama de blocos da solução de controlo para todo o sistema Batching

Desta forma controla-se o sistema específico de batching analisado. A solução de controlo constrói-se com a instanciação de dois blocos funcionais fundamentais e da ligação entre eles.

Procurar o alargar de aplicação da solução é o próximo passo desta investigação, procurando para isso tirar partido da integração vertical, ainda pouco explorada, e da consequente facilidade de configuração.

6.4.Expansão da Aplicação da Solução

Um importante aspecto de um sistema de batching é o transporte do material (fluído ou não) entre os vários equipamentos ou processos. O material no sistema em análise (tinta líquida) é transportado por gravidade através da abertura e fecho de válvulas antes e depois dos equipamentos. A maior parte dos sistemas de batching industriais possuem válvulas (manuais ou automáticas) apenas para seccionamento de tubagens e o fluído é movido através de bombagem. Normalmente o funcionamento das bombas está condicionado com os encravamentos das válvulas na compressão e na admissão.

Sendo assim, a ordem de inicio de bombagem é sempre acompanhada pela ordem de abertura de válvulas. Aplicando este conceito na solução de automação encontrada, não existe alteração alguma, pois as variáveis DOWNSTREAM_VALVE e UPSTREAM_VALVE podem ser usadas para comandar o arranque de bombas em vez de comandarem simplesmente a abertura de válvulas.

Em resumo, o método de transporte de material entre equipamentos não é factor limitativo da solução apresentada.

A definição do nível da tinta em cada doseador foi definida por um booleano (1 Dose – TRUE, ½ Dose – FALSE). Isto limita a aplicação da solução a qualquer outro tipo de doseador. Sendo assim é preferível que o nível seja definido com um inteiro. Desta forma alterações futuras do código para implementar doseadores com mais níveis serão simples de promover.

Na abordagem inicial definiu-se o bloco doseador com válvula a jusante e a montante e o misturador apenas com válvula a jusante. Pensando num sistema genérico, esta abordagem não só não permite que se possam ligar facilmente dois doseadores em série, devido à coexistência redundante de válvulas, como também não possibilita que um doseador possua mais do que uma válvula de entrada (se bem que esta última particularidade é algo que na realidade também raramente irá acontecer, um doseador normalmente doseia apenas um produto).

Uma abordagem alternativa seria cada equipamento possuir no seu bloco apenas a válvula a montante ou a jusante, dessa forma a interligação entre equipamentos iguais torna-se possível. O diagrama seguinte ilustra a alternativa da válvula a montante.

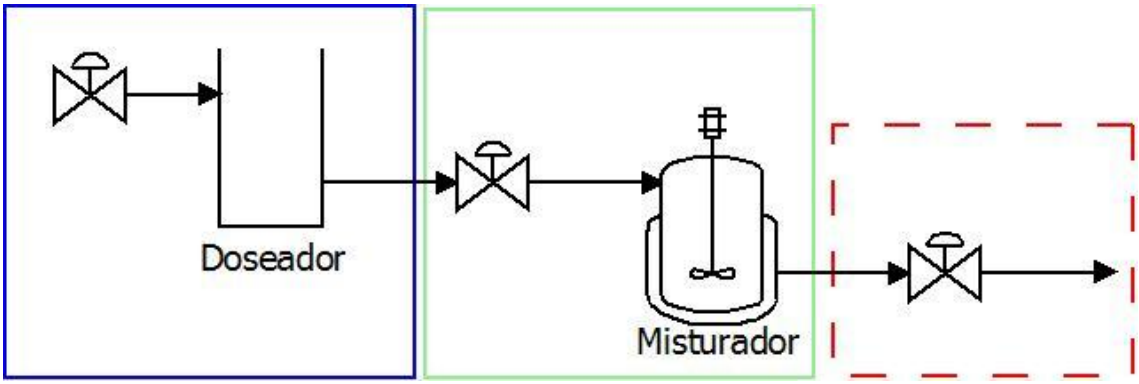


Figura 98 – Alternativa para a subdivisão de equipamentos

Surge contudo o problema do ponto específico de entrada ou saída do sistema obrigar a que seja criado um bloco próprio, pois em algum ponto irá faltar uma válvula de entrada ou saída. A necessidade de criação de um bloco para uma válvula faz relembrar o conceito de módulos partilhados de equipamento. Pode-se congeminar uma outra solução em que cada válvula seja um módulo partilhado pelos equipamentos a montante e a jusante.

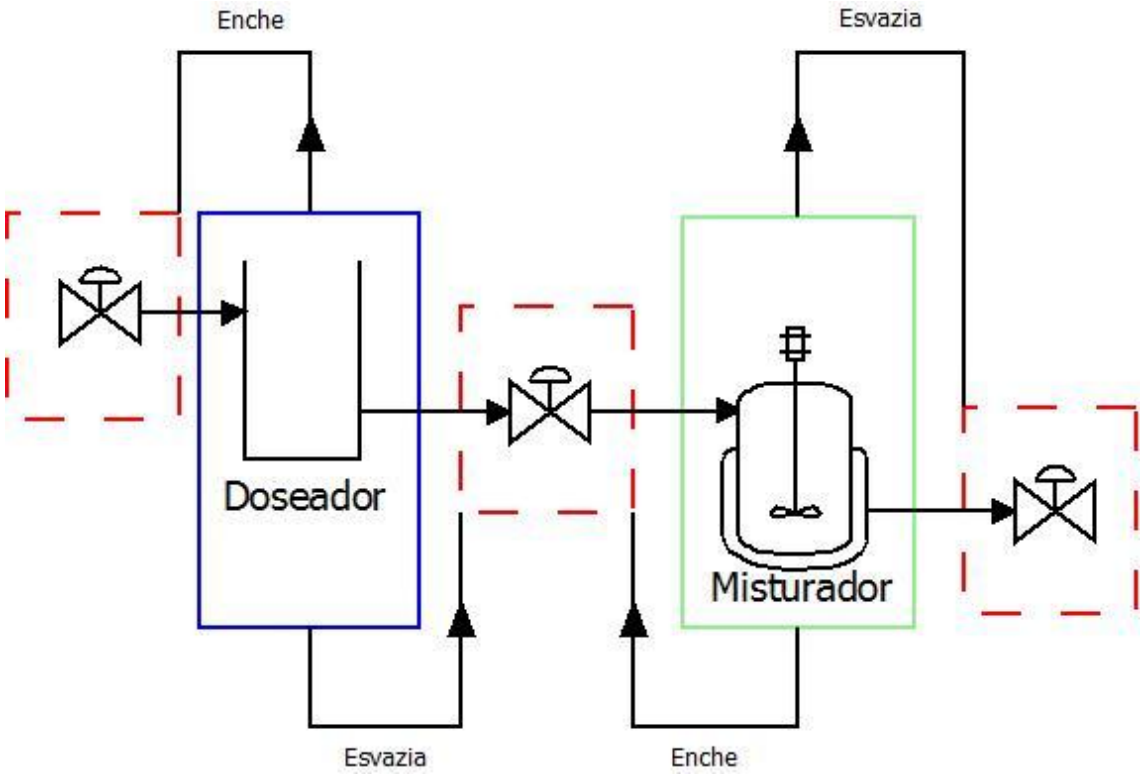


Figura 99 – Alternativa de módulos partilhados de equipamento, para a subdivisão do sistema

A vermelho são representados os módulos partilhados, que seriam implementados através de blocos funcionais que se podem ligar aos blocos dos equipamentos principais do sistema (doseador e misturador). Cada equipamento solicita então um "serviço" ao bloco da válvula que tem de gerir os pedidos de serviço a montante e a jusante. O serviço prestado aos equipamentos acaba por ser o mesmo (esvaziar um equipamento a montante encadeado noutro, é a mesma coisa que encher o equipamento a jusante), mas é necessário efectuar a gestão de disponibilidades dos equipamentos (como visto na integração horizontal na solução inicial) (Figura 100). Esta comunicação de disponibilidades engloba características de sistemas tipo pull e tipo push, na medida em que as ordens de pedido de esvaziamento são do tipo push – o equipamento pede à válvula a jusante que abra. Por sua vez, as ordens de pedido de enchimento são do tipo pull – o equipamento informa a válvula a montante que está disponível para receber tinta.

Figura 100 – GRAFCET base de um possível bloco partilhado que encapsula uma válvula

Num sistema de batching genérico, aplicando esta solução, permitiria que cada equipamento pudesse ter *n* válvulas e não existiriam problemas de ligações entre equipamentos iguais ou diferentes. Qualquer *layout* de equipamentos poderia ser controlado através da conjugação dos blocos de equipamentos e dos blocos partilhados para as válvulas. A grande desvantagem seria o aumento exponencial de número de instâncias de blocos funcionais criados para cada sistema. No exemplo

deste capítulo em vez de 4 instâncias de 2 blocos funcionais, existiriam 11 instâncias de 3 blocos funcionais.

Posto isto, a opção tomada é conservar a solução inicial com motivações de simplicidade de implementação, mesmo abdicando da solução de máxima flexibilidade. Fica o registo de uma interessante abordagem a merecer atenção em trabalhos futuros.

Ainda que mantendo a filosofia de subdivisão de equipamentos da solução inicial, de forma a aplicar um dos fundamentos dos padrões de desenho – separação receita / equipamento – é necessário procurar uma melhor forma para a definição, exterior aos blocos dos equipamentos, das seguintes variáveis:

- Cor a produzir
- Quantidade de doses a produzir
- Tempo de mistura

Na abordagem inicial duas destas variáveis já existem mas encontram-se contidas nos próprios blocos de equipamento. Apenas a variável "quantidade de doses a produzir" é um conceito novo que é importante para fins de funcionamento contínuo do sistema.

A melhor forma de garantir a separação clara das variáveis que definem uma partida, ao mesmo tempo que se fomenta a integração vertical, é encapsular a lógica da receita e gestão da produção num bloco funcional.

6.5. Solução Final

Com base na solução inicial e na necessidade de alargar o espectro de aplicações, foi alcançada uma solução de controlo mais flexível para o sistema de batching proposto, composta por três blocos funcionais – Receita, Doseador e Misturador.

Importa escrutinar devidamente cada bloco funcional.

Receita

O bloco funcional Receita, que no fundo engloba a informação de como produzir uma determinada tinta e a própria lógica de gestão da produção, permite o encapsulamento das seguintes variáveis:

- Variáveis de Entrada
    - REQ – Booleano que dá a ordem de produção de tinta(s)

- COMPLETE – Booleano que serve de *feedback* do sistema, no que diz respeito ao término de um ciclo de produção de tinta
- QUANTITY – Inteiro que define quantos ciclos de produção de tinta serão processados
- COLOR – Inteiro que indica qual a cor a ser produzida. Cada número indica uma cor que se encontra *hard coded* dentro do bloco.

- Variáveis de Saída
  - DOSE – Booleano que informa os doseadores que devem iniciar uma partida
  - RED_LEVEL, GREEN_LEVEL, BLUE_LEVEL – Inteiro que define a quantidade de tinta para a produção da cor final especificada
  - MIX_TIME – Inteiro que define o tempo de mistura
  - QUANTITY_LEFT – Inteiro que informa o sistema de quantos ciclos restam até o fim da produção. A contagem dos ciclos é feita dentro do bloco Receita, por isso é necessário que esta informação seja transmitida aos equipamentos.
- Variáveis Internas
  - Não existem variáveis de estado, apenas são utilizados dois *triggers* para detecção de transições das variáveis REQ e COMPLETE.

De seguida apresenta-se o código que define estas variáveis.

```
FUNCTION_BLOCK RECIPE
VAR_INPUT
(*Variáveis Software*)
    REQ:BOOL; (*Pedido de produção de tinta*)
    COMPLETE:BOOL; (*Informção de que terminou um ciclo de produção de tinta*)
    QUANTITY:INT; (*Definição da quantidade de doses a produzir*)
    COLOR:INT; (*Definição da cor de tinta a produzir*)
END_VAR
VAR_OUTPUT
(*Variáveis Software*)
    DOSE:BOOL := FALSE; (*Pedido de arranque para os doseadores*)
    RED_LEVEL:INT; (*Quantidade de tinta vermelha*)
    GREEN_LEVEL:INT; (*Quantidade de tinta verde*)
    BLUE_LEVEL:INT; (*Quantidade de tinta azul*)
    MIX_TIME:INT; (*Tempo de mistura*)
    QUANTITY_LEFT:INT; (*Quantidade de ciclos em falta*)
END_VAR
VAR
(*Triggers*)
    R_Req: R_TRIG;
    R_Complete: R_TRIG;
END_VAR
```

A lógica interna do bloco, resume-se a três partes:

- Definição dos níveis de cada tanque doseador de acordo com a cor definida através da variável de entrada COLOR
- Definição do tempo de mistura
- Definição, contagem e gestão dos ciclos de produção

```
IF COLOR = 1 THEN
    RED_LEVEL := 1;
    GREEN_LEVEL := 1;
    BLUE_LEVEL := 0;
END_IF;
```

Para cada valor de COLOR, encontra-se definida uma cor que por sua vez necessita de quantidades específicas de tinta vermelha, verde e azul para ser produzida. Estas definições encontram-se codificadas dentro do bloco, através de ciclos IF. Alternativamente podia-se usar o operador CASE.

Notar que, se fosse preciso aplicar esta solução num sistema com mais doseadores concorrentes para o mesmo misturador, seria necessário alterar o código que define as combinações de cores e adicionar as variáveis de saída necessárias.

A lógica interna do bloco pode ser vista como um conjunto de configurações e de lógica de gestão da produção e não unicamente máquina de estados, pelo que a sua representação gráfica no bloco funcional é diferente dos blocos até aqui apresentados.

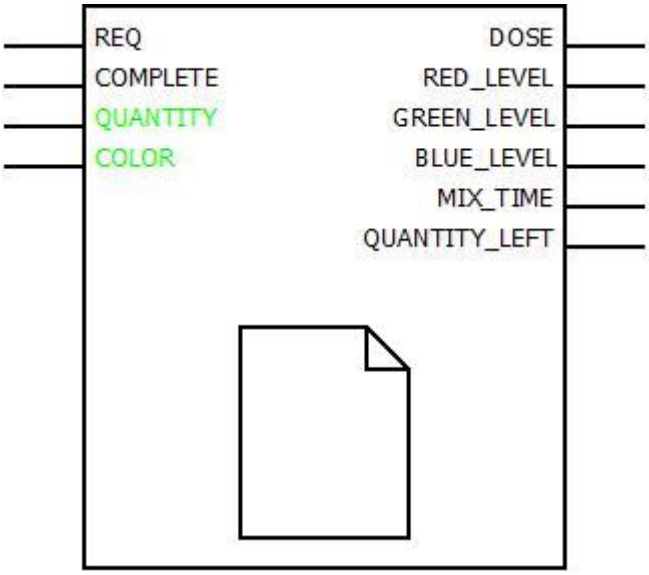


Figura 101 – Bloco funcional da receita de produção

Para simplificação e objectividade, a representação simbólica da receita omite que, na realidade, como se verá de seguida, existe alguma lógica interna associada à gestão da produção.

O código do bloco Receita é de seguida apresentado na sua totalidade.

```
(*Inicialização de Edge Detections*)
R_Req (CLK:=REQ);
R_Complete (CLK:= COMPLETE);

(*Definição de Receita*)
IF R_Req.Q  AND QUANTITY_LEFT = 0 THEN
    QUANTITY_LEFT := QUANTITY;
    DOSE := TRUE;
    IF COLOR = 1 THEN
        RED_LEVEL := 1;
        GREEN_LEVEL := 1;
        BLUE_LEVEL := 0;
    END_IF;
    IF COLOR = 2 THEN
        RED_LEVEL := 1;
        GREEN_LEVEL := 0;
        BLUE_LEVEL := 1;
    END_IF;
    IF COLOR = 3 THEN
        RED_LEVEL := 0;
        GREEN_LEVEL := 1;
        BLUE_LEVEL := 1;
    END_IF;
    IF COLOR = 4 THEN
        RED_LEVEL := 1;
        GREEN_LEVEL :=1;
        BLUE_LEVEL := 1;
    END_IF;
    (*Podem ser configuradas 19 cores
     diferentes com a combinação dos vários sensores de nível*)
    MIX_TIME:=5500;
END_IF;

(*Contador de Ciclos*)
IF R_Complete.Q AND  QUANTITY_LEFT <> 0 THEN
    QUANTITY_LEFT := QUANTITY_LEFT - 1;
END_IF;

(*Final de Ciclo*)
IF  QUANTITY_LEFT = 0 THEN
    DOSE:=FALSE;
    MIX_TIME:=0;
END_IF;
```

Doseador

O bloco Doseador é em tudo similar ao da solução inicial, com a alteração do booleano DOSE para o mais genérico inteiro RECIPE_LEVEL. Foi alterada também a consequente cópia dessa variável de entrada.

```
RECIPE_LEVEL:INT; (*Definição da quantidade de tinta*)
Recipe_Level1 : INT; (* Cópia da variável RECIPE_LEVEL, para garantir que mesmo que
    seja dada uma ordem durante o processamento de outra, esta não é comprometida *)
```

A lógica interna do bloco é similar à abordagem inicial. A sequência de estados é a mesma. A única alteração é uma das condições de arranque de produção (RECIPE_LEVEL diferente de zero).

```
(*Inicialização de Edge Detections*)
R_Start (CLK :=REQ_DOSE);

(*Controlo de Estados*)
IF Inactivo AND R_Start.Q AND RECIPE_LEVEL <>0  THEN (* Quando produção requisita tinta *)
    Recipe_Level1 := RECIPE_LEVEL;
    Inactivo := FALSE;
    Enche := TRUE;
END_IF;

IF Enche AND (HIGH_LEVEL AND Recipe_Level1 = 2 OR MID_LEVEL AND  Recipe_Level1 = 1) THEN
(* Quando cheio *)
    Enche := FALSE;
    Cheio := TRUE;
END_IF;

IF Cheio AND REQ_DELIVERY THEN (* Quando produção autoriza a descarregar *)
    Cheio:= FALSE;
    Esvazia := TRUE;
END_IF;

IF Esvazia AND NOT LOW_LEVEL THEN (* Quando fica vazio *)
    Esvazia := FALSE;
    Inactivo := TRUE;
END_IF;

(* Afectação das Saídas *)
UPSTREAM_VALVE := Enche;
DOWNSTREAM_VALVE := Esvazia;
CNF_COMPLETE := Inactivo;
```

Graficamente o bloco funcional é representado da seguinte forma:

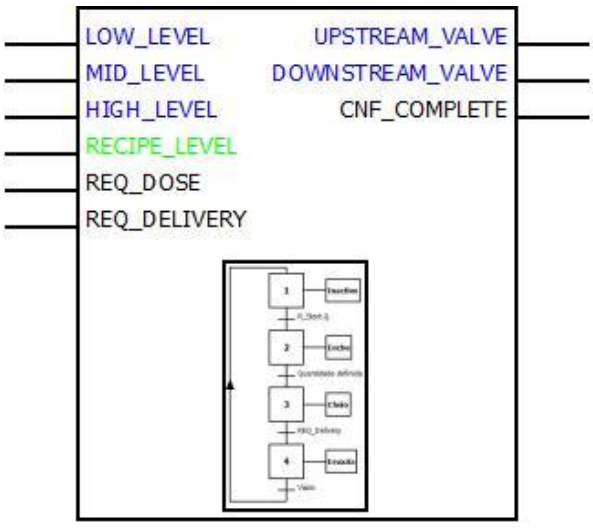


Figura 102 – Bloco funcional do doseador

As variáveis REQ_DOSE e REQ_DELIVERY, deixam de ser necessárias integrar verticalmente, pois serão usadas para coordenação directa entre os equipamentos.

Misturador

O bloco funcional Misturador sofreu poucas alterações. Foi criada uma variável booleana de saída que informa quando o estado do misturador é de inactividade. A variável CNF_COMPLETE foi renomeada para COMPLETE por motivos de clarificação de nomenclatura. Foram criados *triggers* para auxilio na detecção de alguns eventos importantes e para resolver um problema de sincronismo entre blocos (Pulse_Descarga). Este *trigger*, quando activado, mantém-se ligado durante um intervalo de tempo pré-definido, de forma a existir garantia que a comunicação decorre sem problemas.

```
IDLE: BOOL; (*Informação de que o misturador está em espera*)
COMPLETE:BOOL := FALSE; (*Confirmação de término da mistura*)
F_Descarga : F_TRIG; (*Falling trigger da descarga*)
Pulse_Descarga : TP; (*Pulso que mantém o sinal de descarga terminada activo durante 2seg*)
```

O código interno foi elaborado com base na solução inicial, com as alterações de variáveis já anunciadas. A sequenciação de estados manteve-se inalterada.

```
(*Inicialização de Edge Detections*)
R_Start (CLK :=REQ_MIX);
F_Descarga (CLK := Descarga);
Pulse_Descarga (IN:= F_Descarga.Q, PT:=TIME#2s);

COMPLETE := Pulse_Descarga.Q;

(*Definição do tempo de mistura*)
IF R_Start.Q THEN
Mix_Time1 := MIX_TIME;
END_IF;

(*Controlo de Estados*)
IF Inactivo AND LOW_LEVEL THEN  (* Quando começa a receber tinta *)
    COMPLETE := FALSE;
    Inactivo := FALSE;
    Enche := TRUE;
END_IF;

IF Enche AND REQ_MIX THEN
(* Quando produção informa que foi entregue toda a tinta *)
    Enche := FALSE;
    Mistura := TRUE;
END_IF;

IF Mistura AND Timer_mistura.Q THEN
(* Ao fim do tempo de mistura *)
    Mistura := FALSE;
    Descarga := TRUE;
END_IF;

IF Descarga AND NOT LOW_LEVEL THEN  (* Quando vazio *)
    Descarga := FALSE;
    Inactivo := TRUE;
END_IF;
```

```
(*Afectação das Saídas*)
DOWNSTREAM_VALVE:=Descarga OR REQ_DELIVERY;
MIXER:=Mistura;
IDLE := Inactivo;

(*Temporizador com base no tempo de mistura definido*)
Timer_mistura(IN := Mistura, PT := INT_TO_TIME (Mix_Time1));
```

Graficamente o bloco Misturador é o seguinte:

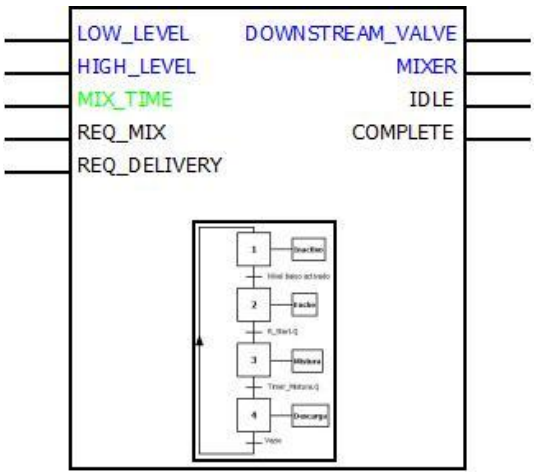


Figura 103 – Bloco funcional do misturador

Para todo o sistema ser controlado, é necessário que cada bloco funcional seja instanciado e configurado da seguinte forma:

```
PROGRAM Main
VAR
    DosingRed:DOSING;
    DosingGreen:DOSING;
    DosingBlue:DOSING;
    Mixing1:MIXING;
    Recipe_Manager:RECIPE;
END_VAR

(*Doseador Vermelho*)
DosingRed (RECIPE_LEVEL := Recipe_Manager.RED_LEVEL);
DosingRed (LOW_LEVEL:= In_0);
DosingRed (MID_LEVEL :=In_1);
DosingRed (HIGH_LEVEL:=In_2);
DosingRed (REQ_DOSE:= Recipe_Manager.DOSE);
DosingRed (REQ_DELIVERY := Mixing1.IDLE);
Out_0 := DosingRed.UPSTREAM_VALVE;
Out_1 := DosingRed.DOWNSTREAM_VALVE;

(*Doseador Verde*)
DosingGreen (RECIPE_LEVEL := Recipe_Manager.GREEN_LEVEL);
DosingGreen (LOW_LEVEL:= In_3);
DosingGreen (MID_LEVEL :=In_4);
DosingGreen (HIGH_LEVEL:=In_5);
DosingGreen (REQ_DOSE:= Recipe_Manager.DOSE);
DosingGreen (REQ_DELIVERY := Mixing1.IDLE);
Out_2 := DosingGreen.UPSTREAM_VALVE;
Out_3 := DosingGreen.DOWNSTREAM_VALVE;
```

```
(*Doseador Azul*)
DosingBlue (RECIPE_LEVEL := Recipe_Manager.BLUE_LEVEL);
DosingBlue (LOW_LEVEL:= In_6);
DosingBlue (MID_LEVEL :=In_7);
DosingBlue (HIGH_LEVEL:=In_8);
DosingBlue (REQ_DOSE:= Recipe_Manager.DOSE);
DosingBlue (REQ_DELIVERY := Mixing1.IDLE);
Out_4 := DosingBlue.UPSTREAM_VALVE;
Out_5 := DosingBlue.DOWNSTREAM_VALVE;

(*Misturador*)
Mixing1 (LOW_LEVEL:= In_9);
Mixing1 (HIGH_LEVEL := In_10);
Mixing1 (MIX_TIME := Recipe_Manager.MIX_TIME);
Mixing1 (REQ_MIX:= DosingRed.CNF_COMPLETE AND DosingGreen.CNF_COMPLETE AND DosingBlue.CNF_COMPLETE);
Out_6 := Mixing1.MIXER;
Out_7 := Mixing1.DOWNSTREAM_VALVE;

(*Recipe Manager*)
Recipe_Manager (COMPLETE := DosingRed.CNF_COMPLETE AND DosingGreen.CNF_COMPLETE AND DosingBlue.CNF_COMPLETE);
Recipe_Manager (REQ:= In_12);
Recipe_Manager (COLOR :=2);
Recipe_Manager (QUANTITY := 1);
```

Integração Horizontal

A ligação entre dois equipamentos é alcançada de forma que esta seja independente do resto do sistema, ou seja cada equipamento só dependa do equipamento a jusante ou a montante para encadear o seu correcto funcionamento.

O encadeamento entre um doseador e o misturador é garantido pelas seguintes duas simples linhas de código:

```
DosingRed (REQ_DELIVERY := Mixing1.IDLE);
Mixing1 (REQ_MIX:= DosingRed.CNF_COMPLETE AND DosingGreen.CNF_COMPLETE AND DosingBlue.CNF_COMPLETE);
```

Esta ligação bilateral garante que um dado doseador descarregue a sua dose para o misturador assim que este esteja disponível (IDLE) e o misturador apenas lança a sua ordem de mistura, assim que garanta que, todos os doseadores existentes a montante já descarregaram toda a tinta que a receita exige.

A restante configuração necessária é toda efectuada fora dos blocos dos equipamentos.

Integração Vertical

Sendo toda a configuração da receita encapsulada no bloco funcional Receita, esta tem que ser transmitida ao sistema através da ligação entre variáveis de entrada e saída dos blocos funcionais. O seguinte código ilustra essa ligação.

```
DosingRed (RECIPE_LEVEL := Recipe_Manager.RED_LEVEL);
DosingRed (REQ_DOSE:= Recipe_Manager.DOSE);
Mixing1 (MIX_TIME := Recipe_Manager.MIX_TIME);
```

Cada doseador (estando representado o de tinta vermelha – DosingRed - neste extracto de código) recebe a informação do nível a que deve dosear a tinta (RED_LEVEL) e recebe a ordem de início de enchimento (DOSE). O misturador recebe o tempo definido de mistura (MIX_TIME).

Desta forma permite-se que qualquer ordem de produção de tinta (cor, número de partidas e tempo de mistura), ou seja a receita, seja configurada de forma completamente externa aos blocos de equipamento. Esta é a separação receita/equipamento tão importante para a correcta implementação de padrões de desenho.

A redução da informação transmitida verticalmente é importante para assim minimizar a configuração inicial e ligação entre blocos.

A imagem seguinte demonstra graficamente toda a solução de controlo, com todos os blocos funcionais configurados e ligados entre si.

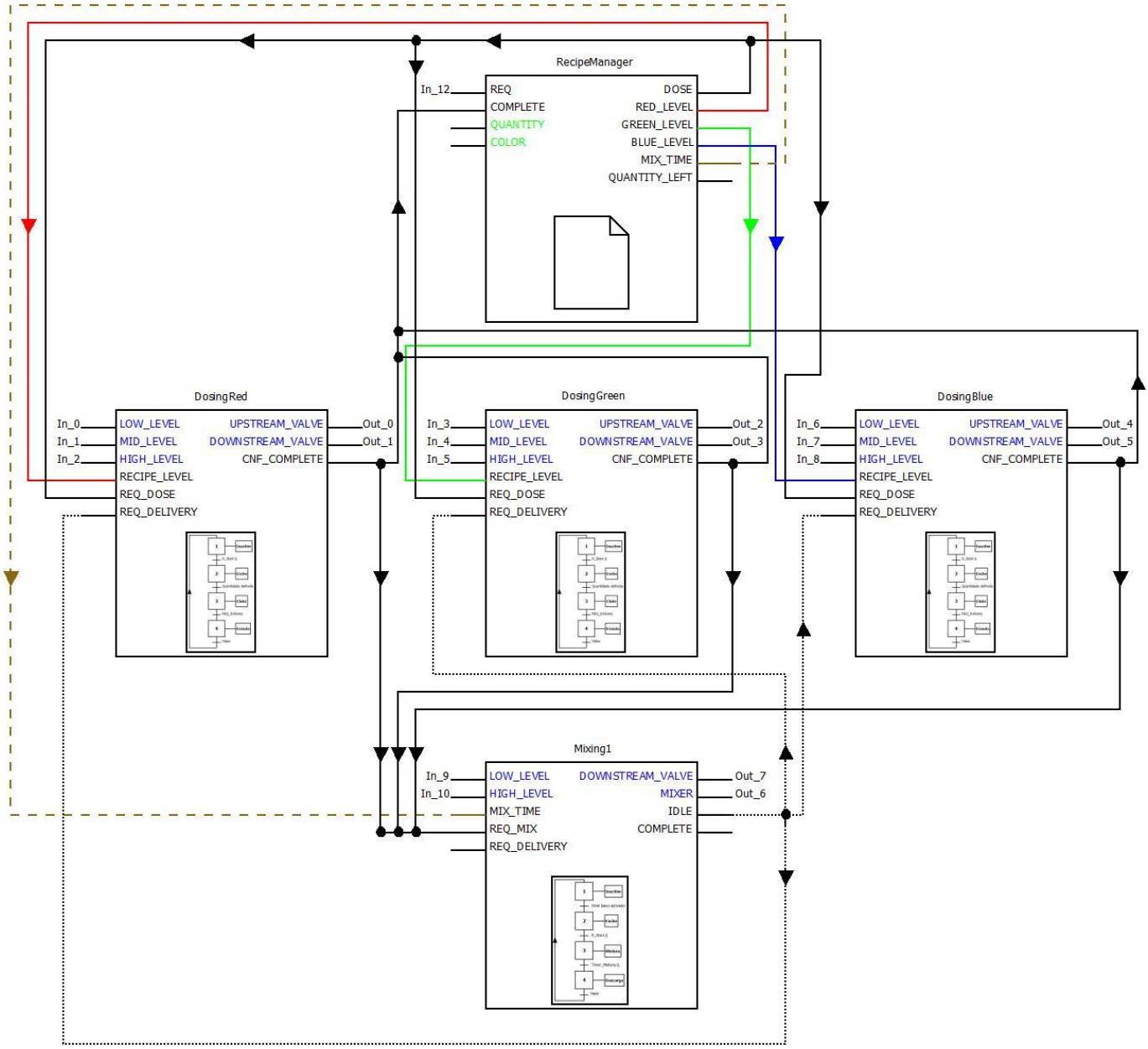


Figura 104 – Diagrama de blocos da solução final de controlo para o Batching

O diagrama de blocos é de relativamente difícil análise, tal deve-se ao considerável número de ligações de integração. Contudo o que é feito para um doseador é exactamente igual aos restantes, pelo que o esforço de compreensão só é feito uma vez.

De toda a forma, como se pode observar, tal como ambicionado, a solução para o controlo do sistema espelha a disposição física do próprio sistema (três doseadores e um misturador), com o acrescentar do bloco de definição de receita.

Comparativo de Soluções

Mesmo não havendo um avultado número de soluções efectivamente experimentadas a nível de programação (apenas duas) segue-se uma comparação entre soluções, usando a mesma métrica qualitativa do sistema de sorting analisado previamente.

Tabela 8 – Comparativo qualitativo das várias hipóteses de controlo do sistema Batching

Solução	Flexibilidade / Modularidade	Facilidade de Configuração	Reutilização Software
Inicial	**	****	****
Final	****	***	****

A solução final obteve sucesso na busca da melhor flexibilidade / modularidade em relação à solução inicial, sem comprometer grandemente a facilidade de configuração e a forte reutilização de software.

Seria interessante aplicar a solução encontrada a um sistema real. Contudo devido à inexistência de sistemas semelhantes no ambiente industrial em que o autor exerce a sua profissão, tal não foi equacionado. Contudo para atestar a verdadeira flexibilidade da solução, pode ser feito um esforço suplementar de imaginação, de forma a perspectivarem-se alterações ao sistema analisado.

- **Mais doseadores em paralelo (mais cores base)**

  Esta mudança exigiria que o bloco funcional Receita fosse alterado, de forma a permitir o interface ao número desejado de doseadores – são necessárias tantas variáveis booleanas de saída do tipo "COLOR"_LEVEL quantas as cores bases que existam. O código dentro do bloco teria também que ser alterado de forma às novas variáveis de saída poderem ser configuradas com as devidas quantidades.

  A nível de equipamentos tanto nos doseadores como no misturador tudo se manteria inalterado, bastando a óbvia instanciação do bloco Doseador para cada equipamento novo.

  A integração horizontal e vertical seria exactamente a mesma.

- **Mais doseadores em série (processos em cascata)**

  Encadear doseadores levanta o problema da integração horizontal. Como foi observado, a solução encontrada não permite a ligação directa entre doseadores. Uma solução pouco ortodoxa (mas praticável) seria a válvula que se sobrepõe nos dois blocos ser configurada em ambos os blocos e dessa forma ter dois blocos a dar-lhe ordens (ver Figura 105). Por questões

de sincronismo natural de funcionamento, a operação da válvula acabaria por ser o esperado e cumpriria a sua função.

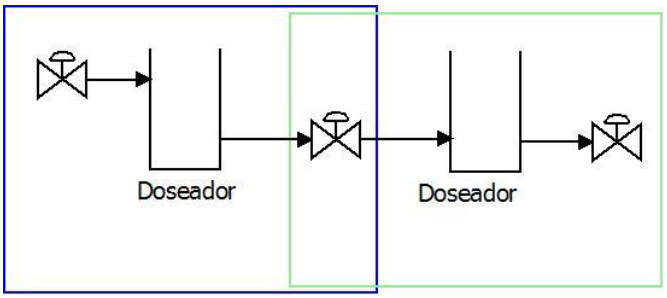


Figura 105 – Encadeamento em série de doseadores

Seria também necessário que o bloco Receita contemplasse os níveis dos novos doseadores.

Globalmente, melhor opção seria implementar a solução com os módulos partilhados de equipamentos. Cada equipamento e cada válvula possuiriam assim um bloco funcional independente, garantindo máxima flexibilidade no que diz respeito a ligações entre equipamentos.

- **Mais misturadores em paralelo**

Introduzir mais misturadores em paralelo em relação ao existente, pressupõe a existência de mais válvulas para permitir a escolha do caminho da tinta após dosagem. Na integração vertical seria necessário definir correctamente qual esse caminho para cada ordem de produção.

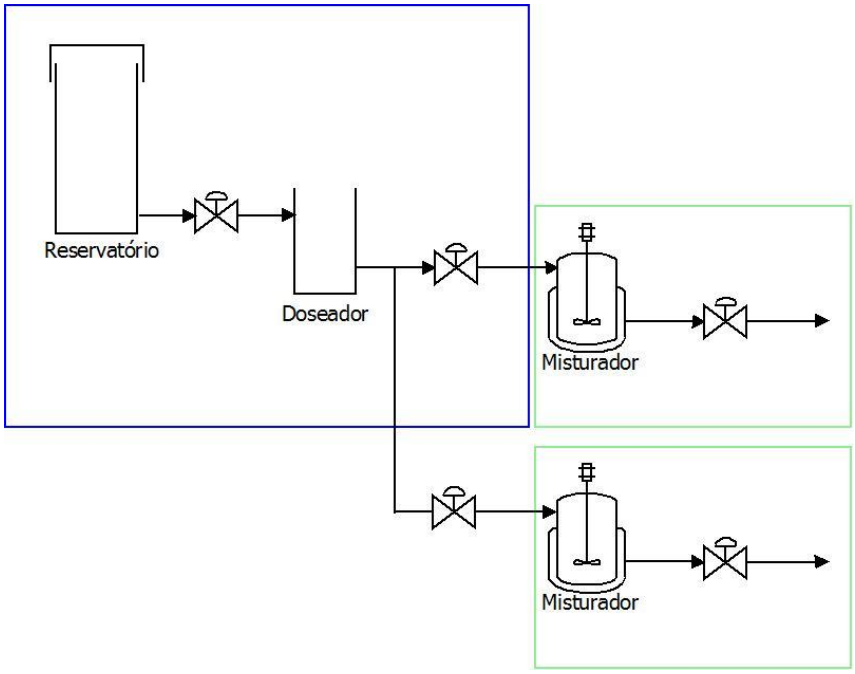


Figura 106 – Disposição de misturadores em paralelo

Essas válvulas, como não fariam parte dos blocos Doseador ou Misturador, necessitam de um bloco específico. Novamente a solução com módulos partilhados de equipamentos resolveria este problema.

- **Mais misturadores em série**

  Inserir mais misturadores encadeados em série com o existente não suscitaria quaisquer alterações nos três blocos funcionais. Neste caso a existência de apenas uma válvula na saída do equipamento permitiria flexibilidade máxima.

  A integração entre misturadores seria feita aquando da configuração inicial através das variáveis de saída IDLE. Um misturador só descarregaria no seguinte quando este estivesse disponível.

- **Outro sistema de batching que não consista em misturar tintas**

  Pensando num sistema de batching mais genérico, qualquer equipamento que tenha o funcionamento típico de encher uma quantidade de material previamente medida, efectuar um processamento ao material e esvaziar, pode ser controlado com o bloco misturador. Em vez de se accionar um misturador, pode-se accionar uma resistência eléctrica, um processo químico, etc. O bloco funcional é aplicável em qualquer dessas situações tipo.

  Um doseador genérico pode possuir inúmeras formas de efectuar a medição de nível (seja analógico ou digital), sendo por isso necessário para cada caso configurar a lógica interna que relaciona o medidor de nível com o nível pedido pelo bloco Receita. Em todo o caso o fundamento é o mesmo.

Resumindo e completando esta informação numa conveniente tabela:

Tabela 9 – Síntese das alterações necessárias aos blocos funcionais encontrados, no caso de alteração do *layout* do sistema

	Receita	**Doseador**	**Misturador**	**Integração Horizontal**	**Integração Vertical**
+ Doseadores em paralelo	Alterações consideráveis	Mantém-se	Mantém-se	Mantém-se	Mantém-se
+ Doseadores em série	Alterações consideráveis	Mantém-se	Mantém-se	Algumas alterações [1]	Mantém-se
+ Misturadores em paralelo	Alterações mínimas	Mantém-se	Mantém-se	Algumas alterações [1]	Alterações consideráveis
+ Misturadores em série	Mantém-se	Mantém-se	Mantém-se	Mantém-se	Mantém-se
Batching genérico	Algumas alterações	Algumas alterações	Mantém-se	Pode manter-se	Pode manter-se

Pode-se então constatar que a solução é suficientemente abrangente e flexível, contudo seria interessante considerar em trabalhos futuros a efectiva implementação de uma solução baseada em módulos partilhados de equipamento.

Adicionalmente, a aplicação da solução a sistemas reais seria uma mais-valia à validação deste trabalho.

Após toda a experimentação no sistema de batching com blocos funcionais segundo a norma IEC 61131-3, surge novamente a questão…

E então a norma IEC 61499?

Além das tendências e características já descritas no capítulo 5.5, pode-se salientar que nesta solução, a forma de integração entre blocos segue muito de perto o conceito "*event-driven*" da IEC 61499. Tomando por exemplo o bloco Doseador, a variável de entrada REQ_DELIVERY pode ser vista como um *event input* e a variável RECIPE_LEVEL como um *data input*. Pode-se representar, a título de exemplo, o bloco doseador segundo a norma IEC 61499.

[1] As alterações podem ser de número bastante elevado, se for implementada uma solução alternativa com módulos partilhados de equipamento para as válvulas.

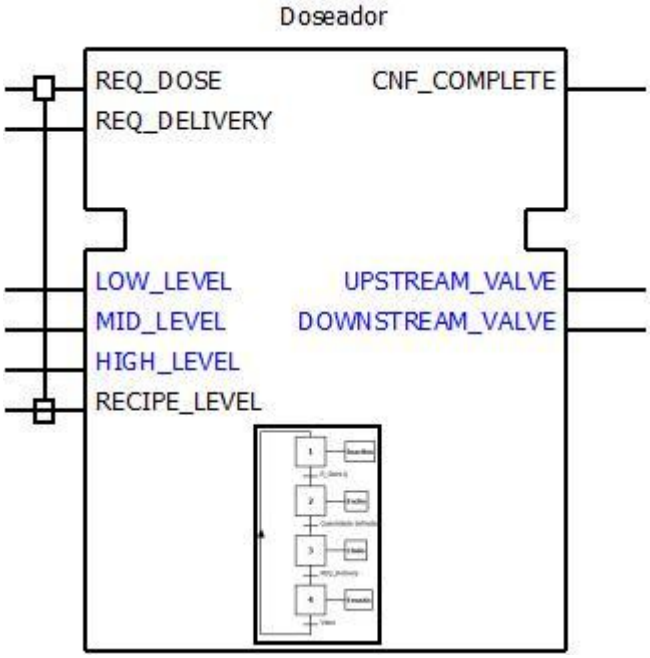


Figura 107 – Bloco funcional segundo a IEC 61449 para o doseador

Ao pensar-se em controlo distribuído segundo a IEC 61499 – Figura 108 – é notória a necessidade de uma aplicação poder ser distribuída por diferentes controladores, podendo até cada controlador dedicar-se a um só equipamento. Sendo cada equipamento controlado por um bloco funcional bem definido e integrável em qualquer sistema, esta distribuição é fácil de implementar, segura e eficiente. Os aspectos relacionados com a rede e forma de comunicação entre os vários controladores não foram abordados nesta dissertação.

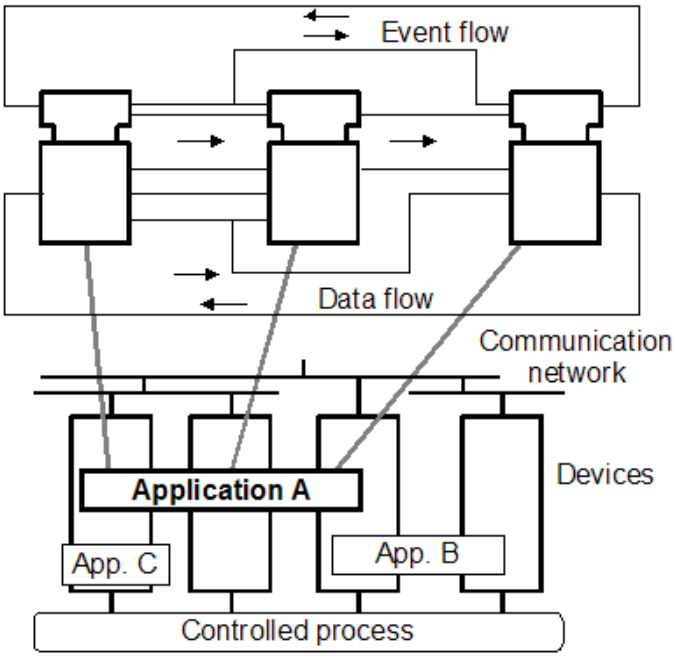


Figura 108 – Aplicações de controlo distribuídas por vários dispositivos (IEC, 2005)

7. Conclusões

O objectivo máximo desta dissertação foi a busca de padrões de desenho em soluções programadas de automação flexível. Pelo caminho foi-se procurando a resposta para diversas questões colaterais mas de certa forma relacionadas com o objectivo. Relembrando as questões lançadas na introdução deste trabalho, podem-se agora tecer respostas validadas por toda a experiência adquirida.

Quando encontramos um sistema que precisa de ser controlado automaticamente, quais os passos que se devem tomar?

A metodologia usada de análise do sistema, identificação de equipamentos e processos, subdivisão do sistema em módulos, criação de solução inicial de controlo e expansão da sua aplicação, provou ser eficiente no alcançar de uma solução de automação, seja ela uma solução tradicional ou flexível.

A cada implementação de uma solução de automação deve-se criar algo novo ou devemo-nos sempre basear nas "boas práticas" existentes?

Ao pensar-se em padrões de desenho, estes devem ser aplicados sempre que possível pois oferecem um *template* de base para o desenvolvimento da solução de controlo. Pegar nas boas práticas e sistematiza-las ao mesmo tempo que se alarga o espectro de situações em que são aplicáveis, leva-nos a criar um padrão de desenho.

As normas existem para serem usadas ou apenas para servirem de tábula rasa?

As normas IEC 61131 e IEC 61499, estão presentes na indústria por um motivo, ajudar os engenheiros e os fabricantes de equipamentos a comungarem ideias, conceitos e linguagens. As normas não existem para criar entraves, apenas para facilitar a criação de boas soluções de automação. Dito isto, as normas devem ser usadas em todo o seu escopo sempre que possível. Pela minha experiência tal nem sempre acontece, pois o engenheiro na indústria tende a desligar-se dos fundamentos teóricos por detrás do que faz. Em todo o caso, visto que os fabricantes seguem claramente as normas, no resultado final de uma solução de automação acaba por não ser notório se quem a fez tomou ou não consciência dos fundamentos que as normas prevêem.

Até que ponto é possível traçar paralelismos entre diferentes sistemas?

Olhar-se para dois sistemas de automação distintos e procurar-se semelhanças que permitam o desenvolvimento de soluções de automação comuns, é algo que por si só pode ser tema para uma tese. No presente trabalho apenas se traçaram paralelismos entre os sistemas originais e eventuais *upgrades* ou alterações, com sucesso.

A programação orientada por objectos tem o seu lugar na automação?

Com a utilização da linguagem de blocos funcionais que encapsulam código numa qualquer outra linguagem, podem-se por em prática os princípios da programação orientada por objectos. As suas vantagens são claras, simplicidade de estruturação, modularidade, reconfigurabilidade, extensibilidade e forte e eficaz reutilização de código.

Devemos procurar soluções dedicadas ou flexíveis?

A pergunta aparentemente mais fundamental é a que tem resposta mais fácil de obter. Num momento em que a indústria de produção aplica o princípio de *make-to-orde*r em resposta ao paradigma de customização em massa, as necessidades de alteração de sistemas automatizados de produção são cada vez maiores. A flexibilidade da automação tem que seguir a flexibilidade dos sistemas. Diversos estudos realizados provam-no, e as conclusões desta dissertação apontam no mesmo sentido. Em breves palavras resume-se o pensamento generalizado - soluções flexíveis são o futuro da automação industrial

Ao longo deste trabalho foram analisados dois sistemas típicos da indústria – Sorting e Batching – e foram desenvolvidos padrões de desenho que conferem flexibilidade e modularidade a quaisquer sistemas similares. Os padrões de desenho consistem em blocos funcionais segundo a norma IEC 61131-3 mas estruturados tendo em vista a futura aplicação da norma IEC 61499. Cada sistema pode ser composto por tantos blocos quantos os equipamentos ou processos que contém, e deve existir uma forte preocupação em espelhar na solução de controlo a estrutura física do sistema. A ligação entre os blocos confere a sequenciação de funções e a transmissão da informação relevante ao funcionamento. A separação receita / equipamento é clara e é assinalada pela capacidade de cada bloco funcional interpretar as ordens de produção (receita) e comandar as acções do equipamento físico.

Os padrões de desenho alcançados foram testados em várias aplicações virtuais, idealizadas e reais, com considerável sucesso. Os padrões visaram ser aplicados ao maior número possível de situações, sendo esse número limitado pela crescente dificuldade de configuração inicial dos blocos. Existe então um certo nível de alteração dos sistemas que vai além das capacidades dos blocos funcionais encontrados. Esse problema é contornado com a edição de algumas partes do código do bloco ou pelo adicionar de novas variáveis de entrada ou saída.

No fim do capítulo 5 foi elaborado um exercício inventivo, com o fim de materializar as vantagens económicas de uma solução flexível, e os lucros estimados para uma empresa de laboração contínua como a Europac Kraft Viana são avultados.

Apenas a justificação económica chegaria para validar a aplicação de padrões de desenho na automação, contudo as mais-valias que uma solução de automação bem estruturada, modular e facilmente reconfigurável traz para o melhorar do trabalho do engenheiro que faz automação, são tão ou mais importantes.

Em virtude de todos os factos mencionados, os padrões de desenho têm definitivamente o seu lugar em soluções programadas de automação flexível, e o esforço em desenvolve-los é válido e de manifestos dividendos.

A nível pessoal, cabe-me referir que o que se conclui deste trabalho é tão importante como o que se aprendeu pelo caminho, e quando um trabalho nos permite aprender mais do que estamos inicialmente à espera, é sempre sinal que tiramos o máximo partido da experiência.

Ao longo dos últimos meses, pude pôr em prática muitas ideias que foram surgindo ao longo do meu percurso académico e profissional. É gratificante constatar-se que os anos de faculdade não foram uma mera formalidade teórica mas sim um tempo de adquirir conceitos e conhecimentos vitais, que são realmente aplicados no dia-a-dia de um engenheiro.

Como sinto que qualquer trabalho nunca está verdadeiramente terminado, ficam algumas ideias para futuros estudos que possam buscar inspiração e motivação no presente.

- Seria interessante procurar definir métricas quantitativas para a aferição da modularidade de uma solução. A métrica usada neste trabalho foi fundamentalmente qualitativa (à excepção da tentativa de quantificação monetária do fim do capítulo 5).
- As soluções alcançadas não tiveram em conta a detecção de erros e situações de emergência que qualquer solução de automação deve ter. Poderia ser feito o trabalho de completar os padrões de desenho de forma a atender a estes princípios fundamentais. Igualmente poderia ser estudado com maior profundidade a forma dos equipamentos interagirem com o utilizador, nomeadamente através de comandos locais.
- As diferenças entre sistemas push e sistemas pull não foram devidamente aprofundadas. Seria atraente explorar melhor em que diferem os padrões de desenho para cada um desses tipos de sistemas.
- A abordagem de módulos partilhados de equipamento, discutida no capítulo 6.4, é digna de ser cuidadosamente aprofundada, pois aparentemente pode apresentar resultados muito interessantes para a flexibilidade da solução.

- De forma a comprovar que a solução é totalmente distribuível por vários controladores, sugere-se que seja criada uma rede de controladores que implementem cada um o seu bloco funcional, e que em conjunto controlem um sistema de teste.
- Provavelmente o trabalho mais interessante que esta dissertação pode despoletar é a aplicação de padrões de desenho a outros tipos de sistemas produtivos. Podem ser analisados um sem número de sistemas e inferidos novos padrões de desenho com o objectivo de se alcançar uma vasta biblioteca de blocos funcionais que possam ser prontamente aplicados para resolver problemas de controlo.

Como se pode observar pela data de todas as referências e publicações, o campo de pesquisa de soluções flexíveis normalizadas na automação é relativamente recente. Se a investigação nesta área continuar a proliferar, idealmente chegaremos a um ponto onde existirá um padrão de desenho aplicável a cada tipo de sistema, que contemplará tudo que uma solução de automação deve contemplar, e quem sabe se algum dia as mais-valias económicas e de engenharia, convencem clientes e fornecedores de que devem fazer uma aposta inequívoca em soluções flexíveis alicerçadas em padrões de desenho.

Referências

Brandl, D. (2007). *Design Pattners for Flexible Manufacturing.* EUA: ISA.

Dai, W., & Vyatkin, V. (2010). On Migration from PLCs to IEC 61499: Addressing the Data Handling Issues. 8th International IEEE Conference on Industrial Informatics, Osaka, Japão.

Dai, W., & Vyatkin, V. (2010). Redesign Distributed IEC 61131-3 PLC System in IEC 61499 Function Blocks. IEEE International Conference on Emerging Technologies and Factory Automation, Bilbao, Espanha.

Fletcher, M., Garcia-Herreros, E., Christensen, J., Deen, S., & Mittmann, R. (2000). *An Open Architecture for Holonic Cooperation and Autonomy.* DEXA '00 Proceedings of the 11th International Workshop on Database and Expert Systems Applications, Londres, Reino Unido .

Frey, G., & Hussain, T. (2006). Modeling techniques for distributed control systems based on the IEC 61499 standard - current approaches and open problems. 8th International Workshop on Discrete Event Systems, Michigan, E.U.A.

Hayslip, N., Sastry, S., & Gerhardt, J. (2006). Networked embedded automation. *Assembly Automation, pag 235 - 241 , 26.*

IEC. (2003). *IEC 61131-3 Programmable Controllers - Part 3 - Programming languages, International Standard, Second Edition.* Genébra: IEC.

IEC. (2005). *IEC 61499 - Function Blocks for industrial-process measurement and control systems.* Genébra: IEC.

Magalhães, A. (2010). *Práticas de Automação Industrial: especificação e programação de soluções de controlo lógico no ambiente de treino "ITS PLC".* Porto: Real Games Lda.

OPC Foundation. (n.d.). (Consultado em Junho de 2011) Retrieved from http://www.opcfoundation.org/

Oracle Sun Developer Network (SDN). (Consultado em Maio de 2011). Retrieved from http://java.sun.com/developer/onlineTraining/Programming/BasicJava2/oo.html

Schillo, M., & Fischer, K. *Holonic Multiagent Systems.* Alemanha.

Sunder, C., Zoitl, A., & Dutzler, C. (2006). Functional structure-based modelling of automation systems. *International Journal of Manufacturing Research pag 405 - 420 , 1.*

Vyatkin, V. (Consultado em Março de 2011). *Function Blocks -- IEC 61499 Standard - Engineering Distributed Embedded Automation Systems with the New Generation Component Architecture*. Retrieved from http://www.fb61499.com/

Zoitl, A., & Vyatkin, V. (2009). IEC 61499 Architecture for Distributed Automation: the 'Glass Half Full' View. *IEEE Industrial Electronics Magazine, pag 7-23* .

Bibliografia

Christensen, J.H. Design patterns for systems engineering with IEC 61499. E.U.A.

Black, G., & Vyatkin, V. (2010). Intelligent Component-Based Automation of Baggage Handling Systems With IEC 61499.

CoDeSys, Controller Development System, 3S Software. Consultado em Janeiro de 2011, de http//.www.3s-software.com

Dubinin V., Vyatkin, V. On Definition of a Formal Model for IEC 61499 Function Blocks. EURASIP Journal of Embedded Systems. Janeiro 2007. Nova Zelândia

FBDK, The Function Block Development Kit. Consultado em Abril 2011, de http://www.holobloc.com/

FBench, Fbench Project. Consultado em Abril 2011, de http://www.ece.auckland.ac.nz/~vyatkin/fbench/

Fletcher, M. Holonic Manufacturing Systems: Some Scenarios and Issues. Agent Oriented Software Ltd. United Kingdom

Gerber, C., Hanisch H., Ebbinghaus, S. From IEC 61131 to IEC 61499 for Distributed Systems: A Case Study. Alemanha.

Hall, K., Staron, R., & Zoitl, A. (2007). Challenges to Industry Adoption of the IEC 61499 Standard on Event-based Function Blocks. 5th IEEE International Conference on Industrial Informatics, Nova Iorque.

Hallenborg, K., Demazeau, Y. (2006). Dynamical Control in Large Scale Material Handling Systems Through Agent Technology. Conference on Intelligent Agent Technology (IAT'06), Hong-Kong.

ISaGRAF, ICS Triplex ISaGRAF. Consultado em Abril 2011, de http://www.isagraf.com/

ITS PLC, Interactive Training Systems for PLC, Real Games. Consultado em Junho 2011, de http://www.realgames.pt

Marik, V., Vrba, P., Hall, K.H., Maturana, F.P. Rockwell Automation Agents for Manufacturing. República Checa.

Pang, C. IEC 61499 Function Block Implementation of Intelligent Mechatronic Component. Nova Zelândia.

Vyatkin, V. (2007) IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design. ISA. ISBN: 978-0-9792343-0-9.

Vyatkin, V. On comparisons of the ISaGRAF implementation of IEC 61499 with FBDK and other implementations. Nova Zelândia.

Vyatkin, V. Refactoring of Execution Control Charts in Basic Function Blocks of the IEC 61499 Standard. Nova Zelândia.

Vyatkin, V., Alsafi, Y. Ontology-based Reconfiguration Agent for Intelligent Mechatronic Systems in Flexible Manufacturing. Nova Zelândia.

Vyatkin, V., Hanisch, H. Design of Controllers for Plug-and-Play Composition of Automated Systems from Smart Mechatronic Components. Nova Zelândia.

Vyatkin, V., Veber, C. (2007) Alternatives for Execution Semantics of IEC 61499. Nova Zelândia.

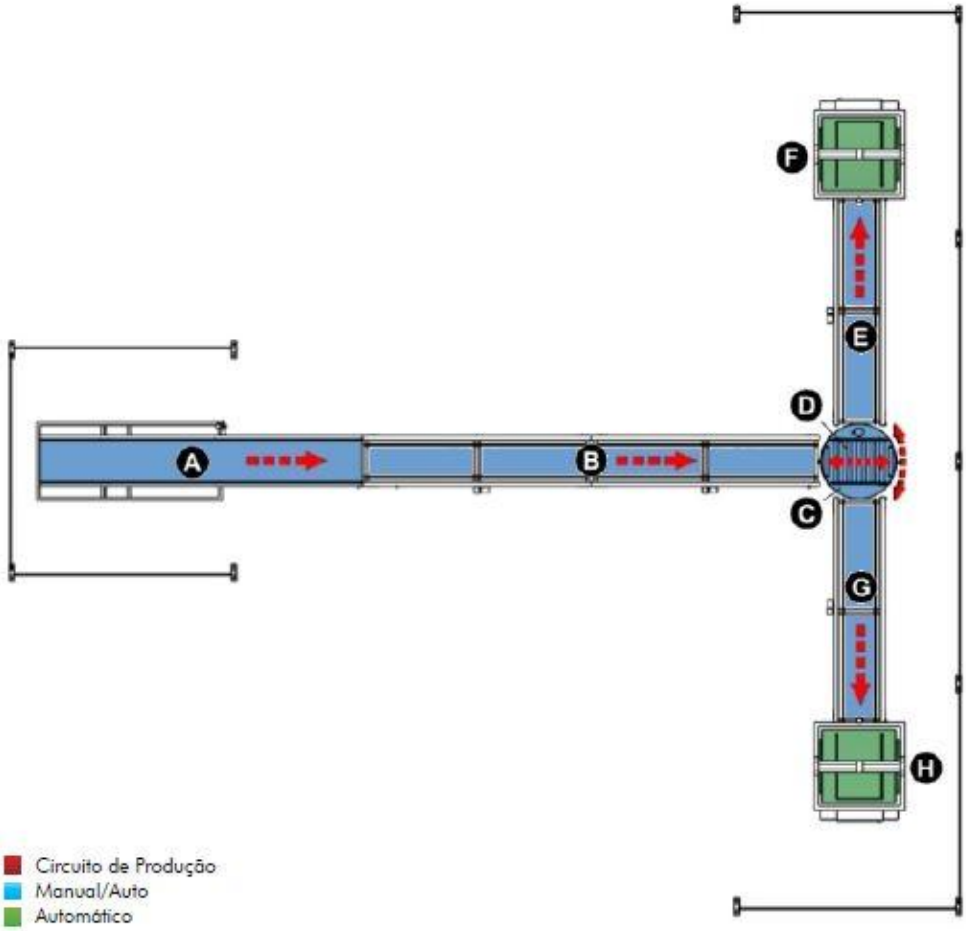
Anexos

Anexo 1 – ITS PLC – Manual do Utilizador

Anexo 1.1 – ITS PLC - Manual do Utilizador - Sorting

SORTING - DESCRIÇÃO DO SISTEMA

Este é um sistema de sorting em que o objectivo é transportar caixas desde o cais de entrada até aos elevadores, separando-as por altura.

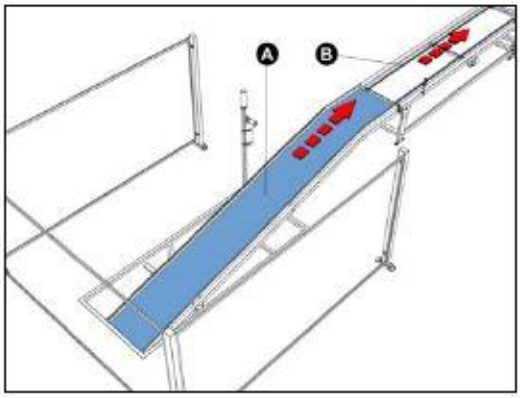


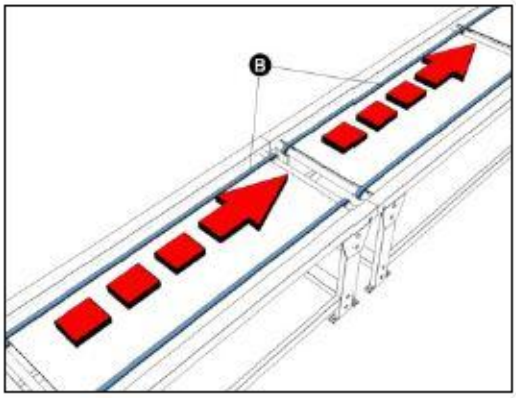
Este sistema é constituído por um cais de entrada, mesas transportadoras e dois cais de saída.

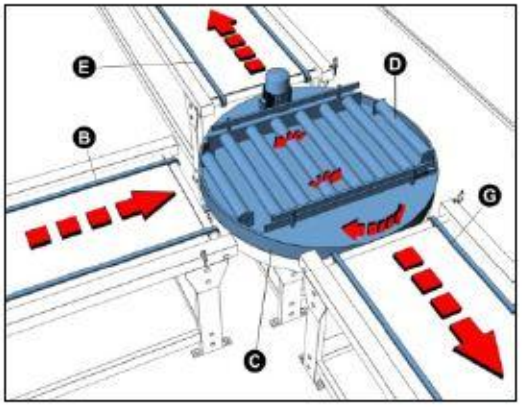
O tapete alimentador (A) transporta paletes carregadas com caixas baixas ou altas que são fornecidas aleatoriamente. As paletes são encaminhadas pelas mesas transportadoras (B) até à mesa rotativa (C) sendo carregadas através dos rolos (D). De acordo com a altura das caixas detectada no inicio das mesas (B) as paletes são rodadas 90° pela mesa rotativa (C) e descarregadas através dos rolos (D) para as mesas transportadoras (E ou G). Finalmente as paletes são enviadas para os elevadores automáticos (F ou H).

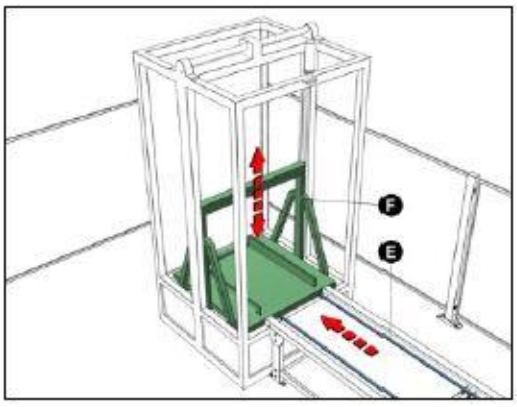
CAIXA BAIXA

CAIXA ALTA







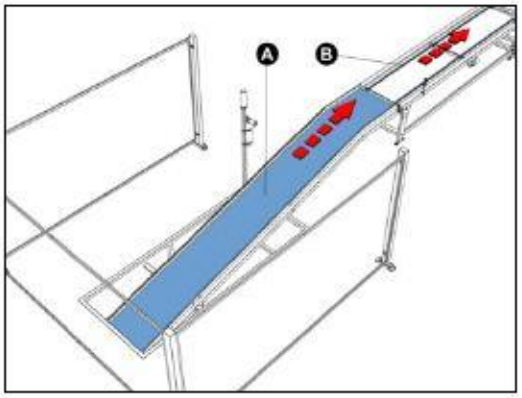


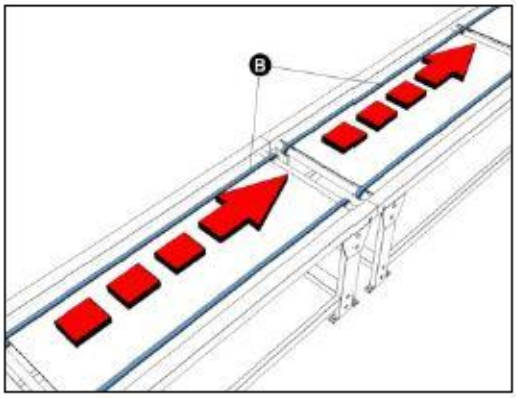
Sugestões:

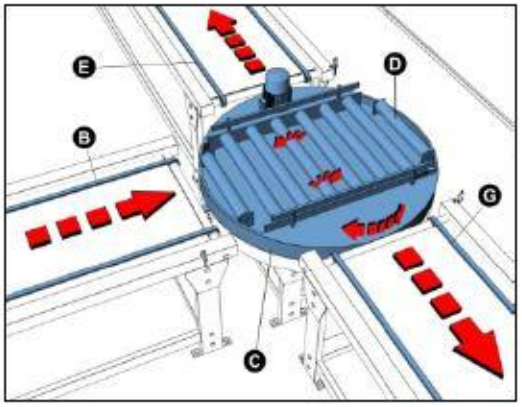
- Comece por separar uma caixa de cada vez. Pare o tapete alimentador (A) após uma caixa entrar no tapete transportador (B). Repita o processo após a caixa ser descarregada para o elevador automático (F ou H).
- Utilize a mesa transportadora (B) como um buffer de caixas. Note que a medição da altura das caixas é realizada à entrada da mesa (B).
- Use uma consola HMI ou um SCADA para mudar a ordem de separação das caixas.

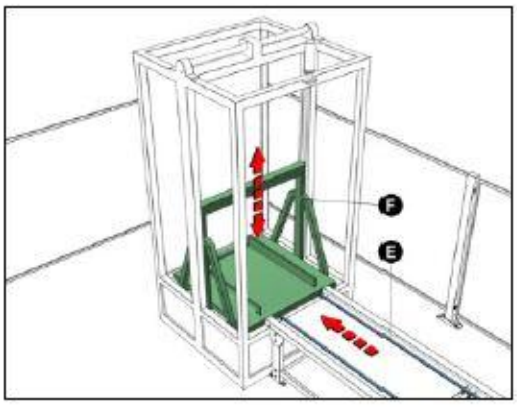
CAIXA BAIXA

CAIXA ALTA







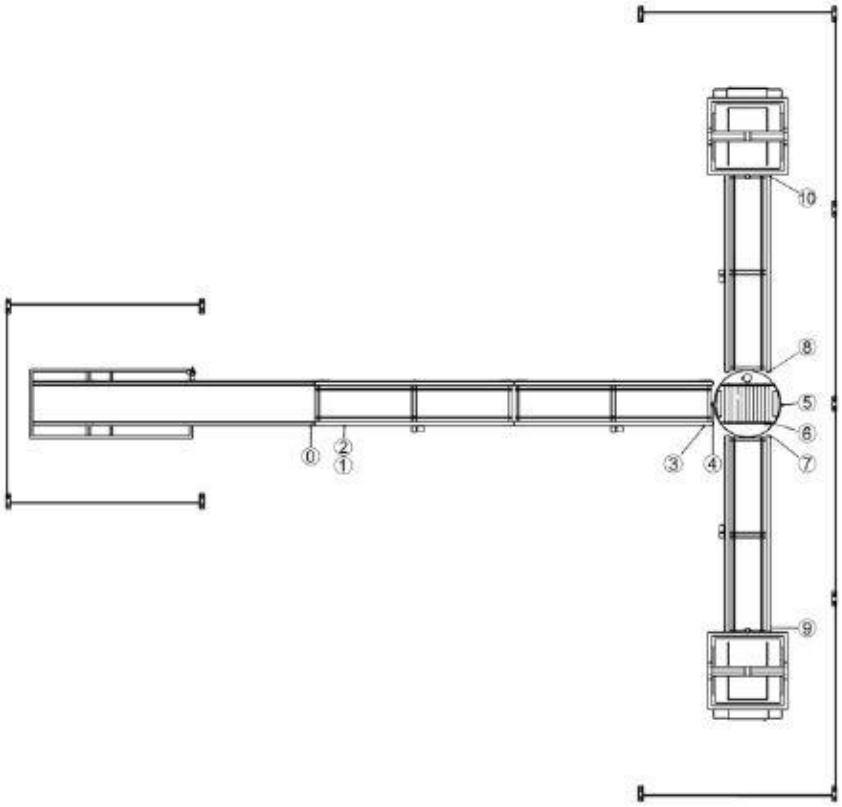


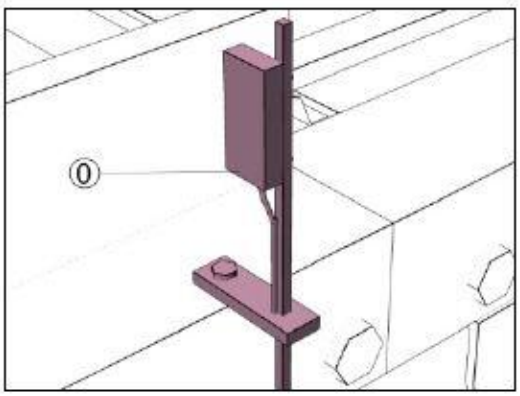
Sugestões:

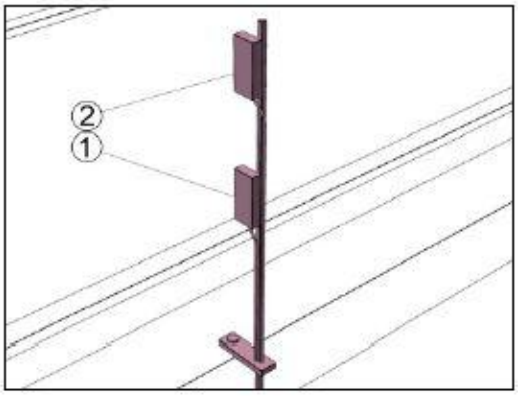
• Comece por separar uma caixa de cada vez. Pare o tapete alimentador (A) após uma caixa entrar no tapete transportador (B). Repita o processo após a caixa ser descarregada para o elevador automático (F ou H).
• Utilize a mesa transportadora (B) como um buffer de caixas. Note que a medição da altura das caixas é realizada à entrada da mesa (B).
• Use uma consola HMI ou um SCADA para mudar a ordem de separação das caixas.

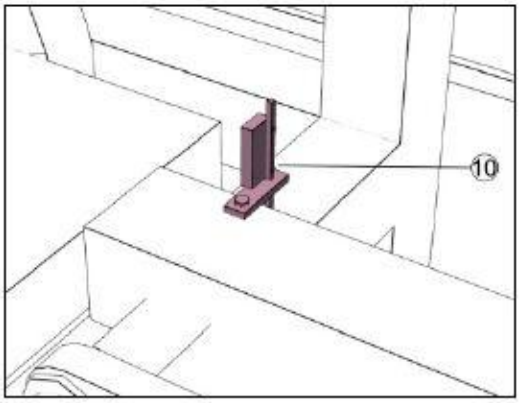
SORTING - SENSORES

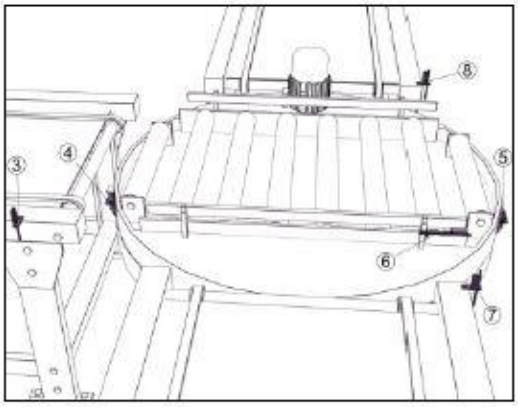
Sensor	Descrição
0	Detector de saída do tapete alimentador
1	Detector de altura inferior
2	Detector de altura superior
3	Detector de saída das mesas transportadoras de entrada
4	Detector da posição de carga da mesa rotativa
5	Detector da posição de descarga da mesa rotativa
6	Detector de palete na mesa rotativa
7	Detector de entrada da mesa transportadora de saída
8	Detector de entrada da mesa transportadora de saída
9	Detector de saída da mesa transportadora de saída
10	Detector de saída da mesa transportadora de saída





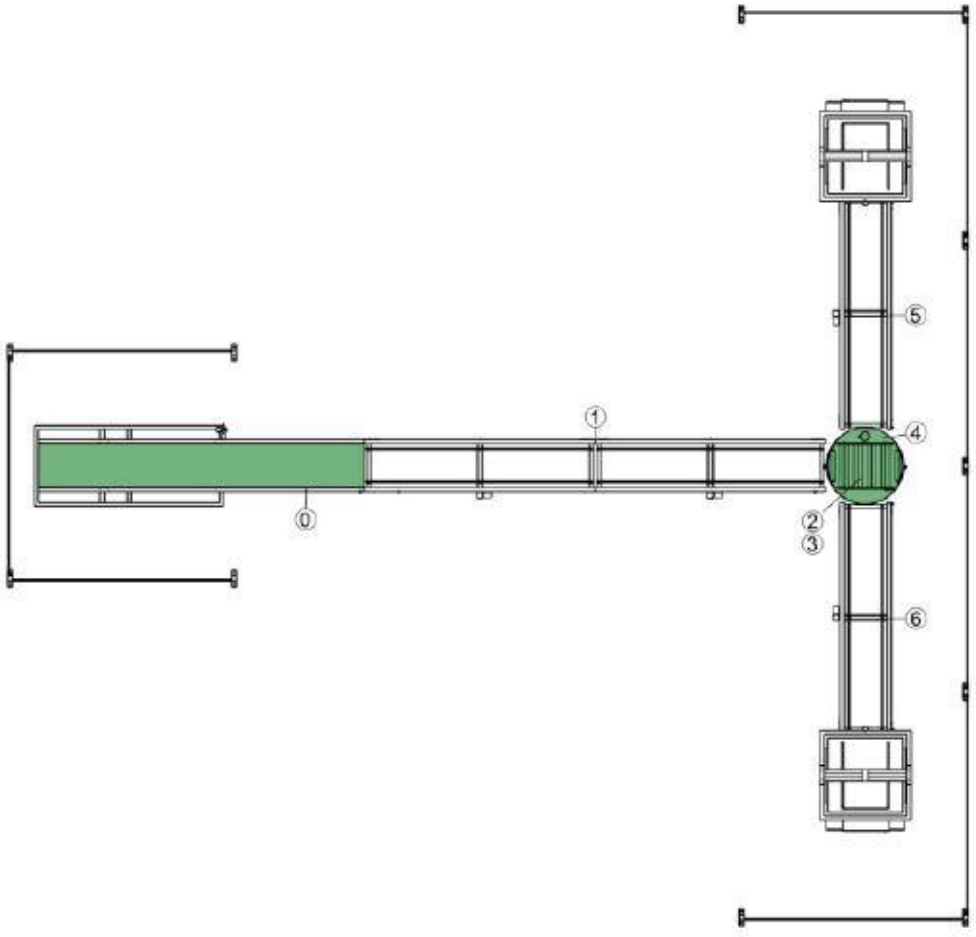


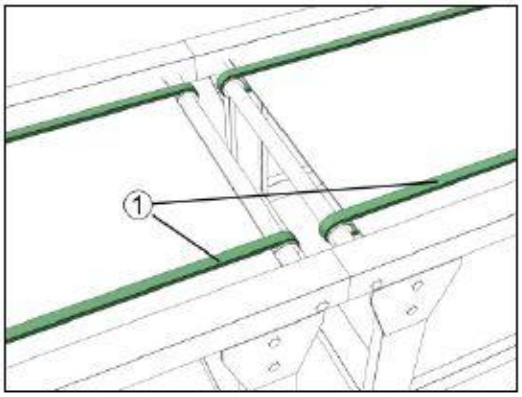


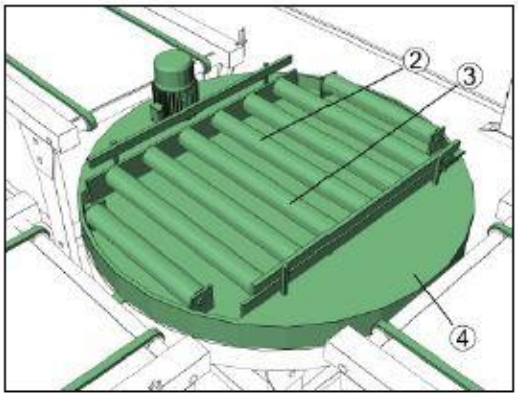


SORTING - ACTUADORES

Actuador	Descrição
0	Tapete alimentador
1	Mesa transportadora de entrada
2	Rolos da mesa rotativa (carga)
3	Rolos da mesa rotativa
4	Mesa rotativa
5	Mesa transportadora de saída
6	Mesa transportadora de saída



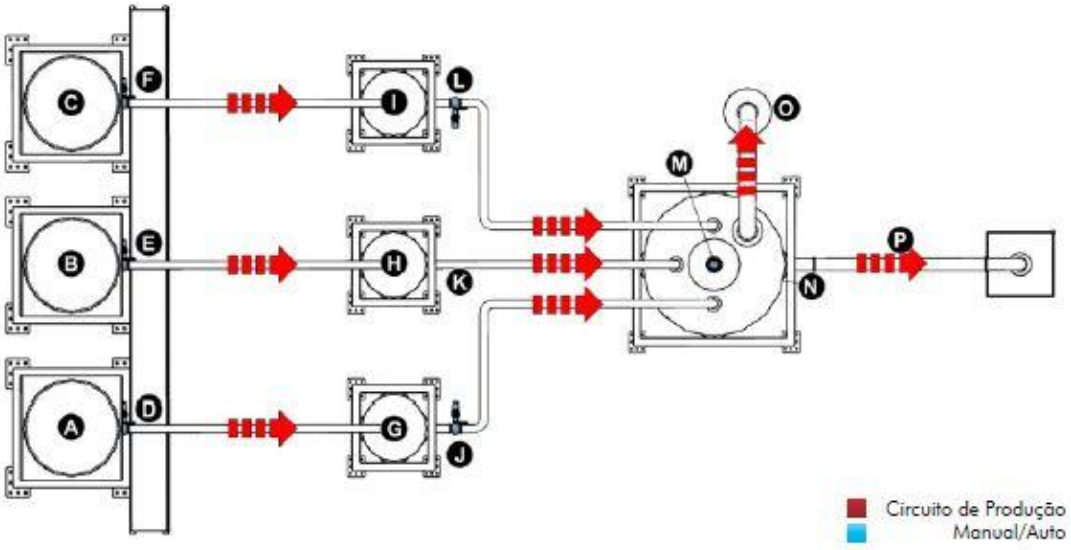




Anexo 1.2 – ITS PLC - Manual do Utilizador – Batching

BATCHING - DESCRIÇÃO DO SISTEMA

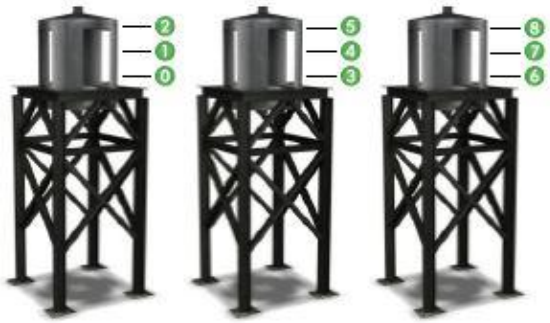
Este sistema simula um processo de mistura de tintas. O objectivo é misturar três tintas de cores primárias (vermelho, verde e azul) de forma a obter a tinta de cor desejada.



Este sistema é constituído por três reservatórios de tinta, três tanques de medição e um tanque de mistura.

Os reservatórios (A, B, C) contêm tintas de cor vermelha, verde e azul respectivamente. A descarga dos reservatórios é feita através das válvulas (D, E, F) para os tanques de medição (G, H, I). Cada tanque possui três pontos de medição. A tinta contida nesses tanques é descarregada através das válvulas (J, K, L) para o tanque de mistura (M). Se o volume de tinta descarregado para o tanque de mistura for superior à sua capacidade, o excedente é descarregado pela purga (O). O processo de mistura tem que durar no mínimo cinco segundos. A tinta produzida é descarregada através da válvula (N) para o tubo de descarga (P).

Cores de tinta obtidas através da combinação dos detectores de nível.



Tanque de Medição de Tinta Vermelha | Tanque de Medição de Tinta Verde | Tanque de Medição de Tinta Azul

Detectores de Nível		
1	4	7
1	5	8
1	4	8
2	4	8
1		8
2	4	7
2		7
1		7
1		
2	5	7
1	4	
2	4	
1	5	7
	5	7
	4	
1	5	
	4	8
	4	7
		7

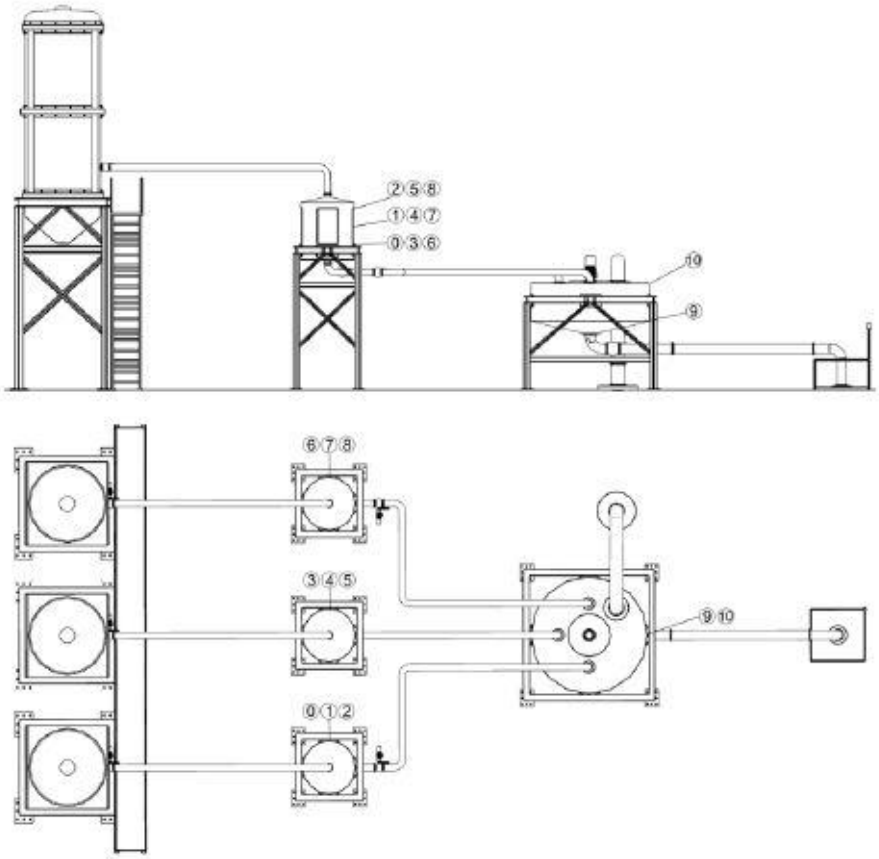
Nota: São necessários cinco segundos para misturar a tinta correctamente.

Sugestões:

- Para começar crie um programa que produza uma só cor, de forma repetida.
- Use temporizadores para efectuar a medição da quantidade de cada tinta para criar outras cores que não as apresentadas na tabela acima.
- Use uma consola HMI ou um SCADA para mudar a receita da tinta a ser produzida.

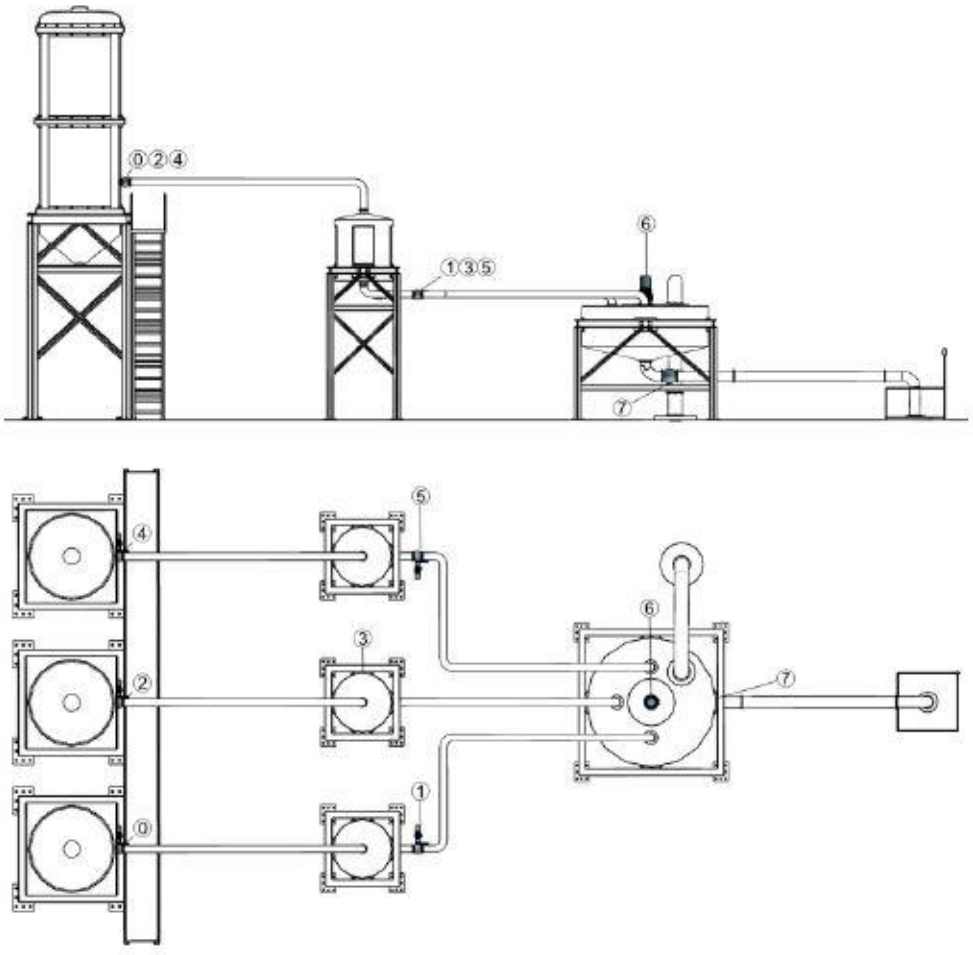
BATCHING - SENSORES

Sensor	Descrição
0	Detector de nível inferior do tanque de medição de tinta vermelha (tanque vazio)
1	Detector de nível médio do tanque de medição de tinta vermelha
2	Detector de nível superior do tanque de medição de tinta vermelha (tanque cheio)
3	Detector de nível inferior do tanque de medição de tinta verde (tanque vazio)
4	Detector de nível médio do tanque de medição de tinta verde
5	Detector de nível superior do tanque de medição de tinta verde (tanque cheio)
6	Detector de nível inferior do tanque de medição de tinta azul (tanque vazio)
7	Detector de nível médio do tanque de medição de tinta azul
8	Detector de nível superior do tanque de medição de tinta azul (tanque cheio)
9	Detector de nível inferior do tanque de mistura
10	Detector de nível superior do tanque de mistura



BATCHING - ACTUADORES

Actuador	Descrição
0	Válvula de descarga do reservatório de tinta vermelha
1	Válvula de descarga do tanque de medição de tinta vermelha
2	Válvula de descarga do reservatório de tinta verde
3	Válvula de descarga do tanque de medição de tinta verde
4	Válvula de descarga do reservatório de tinta azul
5	Válvula de descarga do tanque de medição de tinta azul
6	Misturador
7	Válvula de descarga do tanque de mistura



Anexo 2 – CoDeSys – Código Sorting

```
VAR_GLOBAL

(* Entradas *)
  In_0   : BOOL ; (* Input 0 group 0 *)
  In_1   : BOOL ; (* Input 1 group 0 *)
  In_2   : BOOL ; (* Input 2 group 0 *)
  In_3   : BOOL ; (* Input 3 group 0 *)
  In_4   : BOOL ; (* Input 4 group 0 *)
  In_5   : BOOL ; (* Input 5 group 0 *)
  In_6   : BOOL ; (* Input 6 group 0 *)
  In_7   : BOOL ; (* Input 7 group 0 *)
  In_8   : BOOL ; (* Input 0 group 1 *)
  In_9   : BOOL ; (* Input 1 group 1 *)
  In_10  : BOOL ; (* Input 2 group 1 *)
  In_11  : BOOL ; (* Input 3 group 1 *)
  In_12  : BOOL ; (* Input 4 group 1 *)
  In_13  : BOOL ; (* Input 5 group 1 *)
  In_14  : BOOL ; (* Input 6 group 1 *)
  In_15  : BOOL ; (* Input 7 group 1 *)


(* Saídas *)
  Out_0  : BOOL ; (* Output 0 group 0 *)
  Out_1  : BOOL ; (* Output 1 group 0 *)
  Out_2  : BOOL ; (* Output 2 group 0 *)
  Out_3  : BOOL ; (* Output 3 group 0 *)
  Out_4  : BOOL ; (* Output 4 group 0 *)
  Out_5  : BOOL ; (* Output 5 group 0 *)
  Out_6  : BOOL ; (* Output 6 group 0 *)
  Out_7  : BOOL ; (* Output 7 group 0 *)
  Out_8  : BOOL ; (* Output 0 group 1 *)
  Out_9  : BOOL ; (* Output 1 group 1 *)

END_VAR
```

Capítulo 5.3.1 – Hipótese 3~

```
(*Conveyor Linear de Entrada*)
Conveyor0 (INPUT_IN := TRUE);
Conveyor0 (INPUT_EXIT := In_0);
Conveyor0 (RECEIVE_BUSY:= Conveyor1.SEND_BUSY);
Out_0 := Conveyor0.OUTPUT;
Conveyor0 (MAX_BOX:= 2);

(*Conveyor Linear de Identificação de Caixas*)
Conveyor1 (INPUT_IN := In_0);
Conveyor1 (INPUT_IN_HIGH:= In_2);
Conveyor1 (INPUT_EXIT := In_3);
Conveyor1 (RECEIVE_BUSY:=ConveyorRotary.SEND_BUSY);
Out_1 := Conveyor1.OUTPUT;
Conveyor1 (MAX_BOX:= 2);
```

```
FUNCTION_BLOCK CONVEYOR
VAR_INPUT
(*Inputs Físicos*)
    INPUT_IN: BOOL; (*Sensor de Entrada*)
    INPUT_IN_HIGH:BOOL; (*Sensor de Entrada - Caixa Alta*)
    INPUT_EXIT:BOOL; (*Sensor de Saída*)

(*Inputs Software*)
    MAX_BOX:INT; (*Máximo de caixas presentes simultaneamente no conveyor*)
    RECEIVE_BUSY:BOOL; (*Variável de entrada que indica que o próximo conveyor
                         não se encontra em posição de receber mais caixas*)
END_VAR
VAR_OUTPUT
(*Outputs Físicos*)
    OUTPUT: BOOL; (*Comando do motor do conveyor*)

(*Outputs Software*)
    SEND_BUSY:BOOL;(*Variável que indica ao conveyor antecedente que não se
                     encontra em posição de receber mais caixas*)
END_VAR
VAR
(*Triggers*)
    R_In    : R_TRIG; (*Rising trigger do sensor de entrada*)
    F_In    : F_TRIG; (*Falling trigger do sensor de saída*)
    R_Out : R_TRIG; (*Rising trigger do sensor de saída*)
    F_Out  : F_TRIG;(*Falling trigger do sensor de saída*)
    Counter :CTUD; (*Contador do Max_Box*)

(*Estados Internos*)
    Idle:BOOL:=TRUE; (*Parado e pronto a iniciar marcha*)
    Loading:BOOL :=FALSE; (*A carregar uma caixa*)
    Transporting:BOOL:=FALSE; (*Atransportar uma caixa*)
    Unloading:BOOL:=FALSE; (*A descarregar uma caixa*)
END_VAR
```

```
(* Inicialização de Edge Detections e Counters*)

R_In (CLK := INPUT_IN);
F_Out (CLK := INPUT_EXIT);
F_In (CLK := INPUT_IN);
R_Out(CLK := INPUT_EXIT);

(*Contador de Max_Box*)

Counter(CU := F_In.Q, CD:=F_Out.Q, Reset := In_14, PV := MAX_BOX);

(*Implementação de Max_Box*)

IF Counter.QU THEN
    SEND_BUSY := TRUE;
END_IF;

IF NOT Counter.QU THEN
    SEND_BUSY :=FALSE;
END_IF;

(*Controlo de Estados*)

IF Idle AND INPUT_IN THEN
    Idle:=FALSE;
    Loading:=TRUE;
END_IF;

IF Loading AND F_In.Q THEN
    Loading:=FALSE;
    Transporting := TRUE;
END_IF;

IF Transporting AND R_Out.Q THEN
    Transporting := FALSE;
    Unloading :=TRUE;
END_IF;

IF Unloading AND F_Out.Q THEN
    Unloading := FALSE;
    IF NOT  Counter.QU THEN
        Idle:=TRUE;
    ELSE
        Transporting := TRUE;
    END_IF;
END_IF;

(*Afectação das saídas*)

OUTPUT:=((Loading OR Transporting OR Unloading) AND NOT ((RECEIVE_BUSY AND INPUT_EXIT)));
```

Capítulo 5.3.2 – Hipótese 3

```
(*Conveyor Linear de Entrada*)
Conveyor0 (IS_ROTARY := FALSE);
Conveyor0 (INPUT_IN := TRUE);
Conveyor0 (INPUT_EXIT := in_0);
Conveyor0 (RECEIVE_BUSY:= Conveyor1.SEND_BUSY);
Out_0 := Conveyor0.OUTPUT_LINEAR;
Conveyor0 (MAX_BOX :=2);

(*Conveyor Linear de Identificação de Caixas*)
Conveyor1 (IS_ROTARY := FALSE);
Conveyor1 (INPUT_IN := In_0);
Conveyor1 (INPUT_EXIT := in_3);
Conveyor1 (MAX_BOX :=2);
Conveyor1 (RECEIVE_BUSY:=ConveyorRotary.SEND_BUSY);
Out_1 := Conveyor1.OUTPUT_LINEAR;

(*Conveyor Rotativo*)
ConveyorRotary (IS_ROTARY := TRUE);
ConveyorRotary (INPUT_IN := In_3);
ConveyorRotary (INPUT_EXIT := In_6);
ConveyorRotary (INPUT_ROTARY_IN := In_4);
ConveyorRotary (INPUT_ROTARY_EXIT := In_5);
ConveyorRotary (MAX_BOX :=1);
Out_2 := ConveyorRotary.OUTPUT_LINEAR;
Out_4 := ConveyorRotary.OUTPUT_ROTARY;

Conveyor2 (IS_ROTARY := FALSE);
Conveyor2 (INPUT_IN := In_8);
Conveyor2 (INPUT_EXIT := in_10);
Out_5 := Conveyor2.OUTPUT_LINEAR;
```

```
FUNCTION_BLOCK CONVEYOR
VAR_INPUT
(*Inputs Físicos*)
    INPUT_IN: BOOL; (*Sensor de Entrada*)
    INPUT_IN_HIGH:BOOL; (*Sensor de Entrada - Caixa Alta*)
    INPUT_EXIT:BOOL; (*Sensor de Saída*)
    IS_ROTARY:BOOL; (*Flag que define se o conveyor é rotativo ou não*)
    INPUT_ROTARY_IN:BOOL; (*Sensor de tapete rotativo na posição de carga*)
    INPUT_ROTARY_EXIT:BOOL;(*Sensor de tapete rotativo na posição de descarga*)

(*Inputs Software*)
    MAX_BOX:INT; (*Máximo de caixas presentes simultaneamente no conveyor*)
    RECEIVE_BUSY:BOOL; (*Variável de entrada que indica que o próximo conveyor
                         não se encontra em posição de receber mais caixas*)
END_VAR
VAR_OUTPUT
(*Outputs Físicos*)
    OUTPUT_LINEAR: BOOL; (*Comando do motor do accionamento linear para a frente*)
    OUTPUT_ROTARY:BOOL; (*Comando do motor do accionamento rotativo*)

(*Outputs Software*)
    SEND_BUSY:BOOL;

END_VAR
VAR
(*Triggers*)
    R_In   : R_TRIG; (*Rising trigger do sensor de entrada*)
    F_In   : F_TRIG; (*Falling trigger do sensor de saída*)
    R_Out : R_TRIG; (*Rising trigger do sensor de saída*)
    F_Out  : F_TRIG;(*Falling trigger do sensor de saída*)
    Counter :CTUD; (*Contador do Max_Box*)

(*Estados Internos*)
    Idle:BOOL:=TRUE; (*Parado e pronto a iniciar marcha*)
    Loading:BOOL :=FALSE;(*A carregar uma caixa*)
    Transporting:BOOL:=FALSE; (*A transportar uma caixa*)
    Rotating1:BOOL:=FALSE; (*A rodar para a posição de descarga*)
    Unloading:BOOL:=FALSE;(*a descarregar uma caixa*)
    Rotating2:BOOL:=FALSE; (*A rodar para a posição de carga*)
END_VAR
```

```
(* Inicialização de Edge Detections e Counters*)

R_In (CLK := INPUT_IN);
F_Out (CLK := INPUT_EXIT);
F_In (CLK := INPUT_IN);
R_Out(CLK := INPUT_EXIT);

Counter(CU := F_In.Q, CD := F_Out.Q, Reset := In_14, PV := MAX_BOX);

(*Controlo de Estados*)

IF Idle AND R_In.Q THEN
    Idle:=FALSE;
    Loading:=TRUE;
END_IF;

IF Loading AND F_In.Q AND NOT IS_ROTARY THEN
    Loading:=FALSE;
    Transporting := TRUE;
END_IF;

IF Loading AND R_Out.Q AND IS_ROTARY THEN
    Loading:=FALSE;
    Rotating1 := TRUE;
END_IF;

IF Transporting AND R_Out.Q THEN
    Transporting := FALSE;
    Unloading :=TRUE;
END_IF;

IF Rotating1 AND INPUT_ROTARY_EXIT THEN
    Rotating1 :=FALSE;
    Unloading:=TRUE;
END_IF;

IF Unloading AND F_Out.Q THEN
    Unloading := FALSE;
    IF NOT  IS_ROTARY AND Counter.QD THEN
    Idle:=TRUE;
    END_IF;
    IF NOT IS_ROTARY AND NOT Counter.QD THEN
    Transporting := TRUE;
    END_IF;
    IF IS_ROTARY THEN
    Rotating2:=TRUE;
    END_IF;
END_IF;
```

```
IF Rotating2 AND INPUT_ROTARY_IN THEN
    Rotating2:=FALSE;
    Idle:=TRUE;
END_IF;

(*Implementação de Max_Box*)

IF Counter.QU THEN
    SEND_BUSY := TRUE;
END_IF;

IF NOT Counter.QU THEN
    SEND_BUSY :=FALSE;
END_IF;

(*Afectação das saídas*)

OUTPUT_LINEAR:=(Loading OR Transporting OR Unloading) AND NOT RECEIVE_BUSY;
OUTPUT_ROTARY:=(Rotating1 OR Unloading) AND NOT RECEIVE_BUSY;
```

Capítulo 5.3.2 – Hipótese 2

```
(*Conveyor Linear de Entrada*)
Conveyor0 (INPUT_IN := TRUE);
Conveyor0 (INPUT_EXIT := In_0);
Conveyor0 (RECEIVE_BUSY:= Conveyor1.SEND_BUSY);
Out_0 := Conveyor0.OUTPUT;
Conveyor0 (MAX_BOX:= 2);

(*Conveyor Linear de Identificação de Caixas*)
Conveyor1 (INPUT_IN := In_0);
Conveyor1 (INPUT_IN_HIGH:= In_2);
Conveyor1 (INPUT_EXIT := In_3);
Conveyor1 (RECEIVE_BUSY:=ConveyorRotary.SEND_BUSY);
Out_1 := Conveyor1.OUTPUT;
Conveyor1 (MAX_BOX:= 2);

(*Conveyor Rotativo*)
ConveyorRotary (INPUT_IN := In_3);
ConveyorRotary (INPUT_EXIT := In_6);
ConveyorRotary (INPUT_ROTARY_IN := In_4);
ConveyorRotary (INPUT_ROTARY_EXIT := In_5);
Out_2 := ConveyorRotary.OUTPUT_LINEAR;
Out_4 := ConveyorRotary.OUTPUT_ROTARY;
ConveyorRotary (RECEIVE_BUSY:= (Conveyor2.SEND_BUSY OR Conveyor3.SEND_BUSY));
```

```
FUNCTION_BLOCK CONVEYOR
VAR_INPUT
(*Inputs Físicos*)
    INPUT_IN: BOOL; (*Sensor de Entrada*)
    INPUT_IN_HIGH:BOOL; (*Sensor de Entrada - Caixa Alta*)
    INPUT_EXIT:BOOL; (*Sensor de Saída*)

(*Inputs Software*)
    MAX_BOX:INT; (*Máximo de caixas presentes simultaneamente no conveyor*)
    RECEIVE_BUSY:BOOL; (*Variável de entrada que indica que o próximo conveyor
                         não se encontra em posição de receber mais caixas*)
END_VAR
VAR_OUTPUT
(*Outputs Físicos*)
    OUTPUT: BOOL; (*Comando do motor do conveyor*)

(*Outputs Software*)
    SEND_BUSY:BOOL;(*Variável que indica ao conveyor antecedente que não
                     se encontra em posição de receber mais caixas*)
END_VAR
VAR
(*Triggers*)
    R_In   : R_TRIG; (*Rising trigger do sensor de entrada*)
    F_In   : F_TRIG; (*Falling trigger do sensor de saída*)
    R_Out : R_TRIG; (*Rising trigger do sensor de saída*)
    F_Out  : F_TRIG;(*Falling trigger do sensor de saída*)
    Counter :CTUD; (*Contador do Max_Box*)

(*Estados Internos*)
    Idle:BOOL:=TRUE; (*Parado e pronto a iniciar marcha*)
    Loading:BOOL :=FALSE; (*A carregar uma caixa*)
    Transporting:BOOL:=FALSE; (*Atransportar uma caixa*)
    Unloading:BOOL:=FALSE; (*A descarregar uma caixa*)
END_VAR
```

```
(* Inicialização de Edge Detections e Counters*)

R_In (CLK := INPUT_IN);
F_Out (CLK := INPUT_EXIT);
F_In (CLK := INPUT_IN);
R_Out(CLK := INPUT_EXIT);
R_In_Rotary(CLK := INPUT_ROTARY_IN);
R_Out_Rotary(CLK := INPUT_ROTARY_EXIT);

(*Controlo de Estados*)

IF Idle AND INPUT_IN THEN
    Idle:=FALSE;
    Loading:=TRUE;
END_IF;

IF Loading AND R_Out.Q  THEN
    Loading:=FALSE;
    Rotating1 := TRUE;
END_IF;

IF Rotating1 AND R_Out_Rotary.Q THEN
    Rotating1 :=FALSE;
    Unloading:=TRUE;
END_IF;

IF Unloading AND RECEIVE_BUSY THEN
    Unloading := FALSE;
    Rotating2:=TRUE;
END_IF;

IF Rotating2 AND INPUT_ROTARY_IN THEN
    Rotating2:=FALSE;
    Idle:=TRUE;
END_IF;

(*Afectação das saídas*)

SEND_BUSY:= Rotating1 OR Rotating2 OR Unloading;
OUTPUT_LINEAR:=((Loading OR Unloading) AND NOT (RECEIVE_BUSY AND INPUT_EXIT));
OUTPUT_ROTARY:=(Rotating1 OR Unloading) AND NOT RECEIVE_BUSY;
```

```
(* Inicialização de Edge Detections e Counters*)

R_In (CLK := INPUT_IN);
F_Out (CLK := INPUT_EXIT);
F_In (CLK := INPUT_IN);
R_Out(CLK := INPUT_EXIT);

(*Contador de Max_Box*)

Counter(CU := F_In.Q, CD:=F_Out.Q, Reset := In_14, PV := MAX_BOX);

(*Implementação de Max_Box*)

IF Counter.QU THEN
    SEND_BUSY := TRUE;
END_IF;

IF NOT Counter.QU THEN
    SEND_BUSY :=FALSE;
END_IF;


(*Controlo de Estados*)

IF Idle AND R_In.Q THEN
    Idle:=FALSE;
    Loading:=TRUE;
END_IF;

IF Loading AND F_In.Q THEN
    Loading:=FALSE;
    Transporting := TRUE;
END_IF;

IF Transporting AND R_Out.Q THEN
    Transporting := FALSE;
    Unloading :=TRUE;
END_IF;

IF Unloading AND F_Out.Q THEN
    Unloading := FALSE;
    IF NOT  Counter.QU THEN
        Idle:=TRUE;
    ELSE
        Transporting := TRUE;
    END_IF;
END_IF;

(*Afectação das saídas*)

OUTPUT:=((Loading OR Transporting OR Unloading) AND NOT ((RECEIVE_BUSY AND INPUT_EXIT)));
```

```
FUNCTION_BLOCK CONVEYOR_ROT
VAR_INPUT
(*Inputs Físicos*)
    INPUT_IN: BOOL; (*Sensor de Entrada*)
    INPUT_EXIT:BOOL; (*Sensor de Saída*)
    INPUT_ROTARY_IN:BOOL; (*Sensor de tapete rotativo na posição de carga*)
    INPUT_ROTARY_EXIT:BOOL;(*Sensor de tapete rotativo na posição de descarga*)

(*Inputs Software*)
    RECEIVE_BUSY:BOOL; (*Variável de entrada que indica que o próximo conveyor
                         não se encontra em posição de receber mais caixas*)
END_VAR
VAR_OUTPUT
(*Outputs Físicos*)
    OUTPUT_LINEAR: BOOL; (*Comando do motor do accionamento linear para a frente*)
    OUTPUT_ROTARY:BOOL; (*Comando do motor do accionamento rortativo*)

(*Outputs Software*)
    SEND_BUSY:BOOL; (*Variável que indica ao conveyor antecedente que não
                      se encontra em posição de receber mais caixas*)
END_VAR
VAR
(*Triggers*)
    R_In    : R_TRIG;(*Rising trigger do sensor de entrada*)
    F_In    : F_TRIG;(*Falling trigger do sensor de saída*)
    R_Out : R_TRIG; (*Rising trigger do sensor de saída*)
    F_Out  : F_TRIG;(*Falling trigger do sensor de saída*)
    R_In_Rotary : R_TRIG; (*Rising trigger do sensor de tapete em posição de carga*)
    R_Out_Rotary : R_TRIG; (*Rising trigger do sensor de tapete em posição de descarga*)

(*Estados Internos*)
    Idle:BOOL:=TRUE; (*Parado e pronto a iniciar marcha*)
    Loading:BOOL :=FALSE;(*A carregar uma caixa*)
    Rotating1:BOOL:=FALSE; (*A rodar para a posição de descarga*)
    Unloading:BOOL:=FALSE;(*a descarregar uma caixa*)
    Rotating2:BOOL:=FALSE; (*A rodar para a posição de carga*)

END_VAR
```

```
(* Inicialização de Edge Detections e Counters*)

R_In (CLK := INPUT_IN);
F_Out (CLK := INPUT_EXIT);
F_In (CLK := INPUT_IN);
R_Out(CLK := INPUT_EXIT);
R_In_Rotary(CLK := INPUT_ROTARY_IN);
R_Out_Rotary(CLK := INPUT_ROTARY_EXIT);

(*Controlo de Estados*)

IF Idle AND INPUT_IN THEN
    Idle:=FALSE;
    Loading:=TRUE;
END_IF;

IF Loading AND R_Out.Q  THEN
    Loading:=FALSE;
    Rotating1 := TRUE;
END_IF;

IF Rotating1 AND R_Out_Rotary.Q THEN
    Rotating1 :=FALSE;
    Unloading:=TRUE;
END_IF;

IF Unloading AND RECEIVE_BUSY THEN
    Unloading := FALSE;
    Rotating2:=TRUE;
END_IF;

IF Rotating2 AND INPUT_ROTARY_IN THEN
    Rotating2:=FALSE;
    Idle:=TRUE;
END_IF;

(*Afectação das saídas*)

SEND_BUSY:= Rotating1 OR Rotating2 OR Unloading;
OUTPUT_LINEAR:=((Loading OR Unloading) AND NOT (RECEIVE_BUSY AND INPUT_EXIT));
OUTPUT_ROTARY:=(Rotating1 OR Unloading) AND NOT RECEIVE_BUSY;
```

Capítulo 5.3.4

```
PROGRAM ProceduralControl
VAR
    Conveyor0 : CONVEYOR;
    Conveyor1 : CONVEYOR;
    ConveyorRotary:  CONVEYOR_ROT;
    Conveyor2: CONVEYOR;
    Conveyor3: CONVEYOR;
END_VAR

(*Conveyor Linear de Entrada*)
Conveyor0 (INPUT_IN := TRUE);
Conveyor0 (INPUT_EXIT := In_0);
Conveyor0 (RECEIVE_BUSY:= Conveyor1.SEND_BUSY);
Out_0 := Conveyor0.OUTPUT;
Conveyor0 (MAX_BOX:= 2);

(*Conveyor Linear de Identificação de Caixas*)
Conveyor1 (INPUT_IN := In_0);
Conveyor1 (INPUT_IN_HIGH:= In_2);
Conveyor1 (INPUT_EXIT := In_3);
Conveyor1 (RECEIVE_BUSY:=ConveyorRotary.SEND_BUSY);
Out_1 := Conveyor1.OUTPUT;
Conveyor1 (MAX_BOX:= 2);
Conveyor1 (IS_QUEUE := TRUE);

(*Conveyor Rotativo*)
ConveyorRotary (INPUT_IN := In_3);
ConveyorRotary (INPUT_EXIT := In_6);
ConveyorRotary (INPUT_ROTARY_IN := In_4);
ConveyorRotary (INPUT_ROTARY_EXIT := In_5);
Out_2 := ConveyorRotary.OUTPUT_LINEAR;
Out_3 := ConveyorRotary.OUTPUT_LINEAR_BWD;
Out_4 := ConveyorRotary.OUTPUT_ROTARY;
ConveyorRotary (RECEIVE_BUSY:= (Conveyor2.SEND_BUSY OR Conveyor3.SEND_BUSY));

(*Conveyor Linear de Saída Esquerdo*)
Conveyor2 (INPUT_IN := In_8);
Conveyor2 (INPUT_EXIT := In_10);
Out_5 := Conveyor2.OUTPUT;
Conveyor2 (MAX_BOX:= 1);

(*Conveyor Linear de Saída Direito*)
Conveyor3 (INPUT_IN := In_7);
Conveyor3 (INPUT_EXIT := In_9);
Out_6 := Conveyor3.OUTPUT;
Conveyor3 (MAX_BOX:= 1);
```

```
FUNCTION_BLOCK CONVEYOR
VAR_INPUT
(*Inputs Físicos*)
    INPUT_IN: BOOL; (*Sensor de Entrada*)
    INPUT_IN_HIGH:BOOL; (*Sensor de Entrada - Caixa Alta*)
    INPUT_EXIT:BOOL; (*Sensor de Saída*)

(*Inputs Software*)
    IS_QUEUE:BOOL := FALSE; (*Variável de escolha se o conveyor é um
                              identiicador de alturas de caixas ou não*)
    MAX_BOX:INT; (*Máximo de caixas presentes simultaneamente no conveyor*)
    RECEIVE_BUSY:BOOL; (*Variável de entrada que indica que o próximo conveyor
                         não se encontra em posição de receber mais caixas*)
END_VAR
VAR_OUTPUT
(*Outputs Físicos*)
    OUTPUT: BOOL; (*Comando do motor do conveyor*)

(*Outputs Software*)
    SEND_BUSY:BOOL;(*Variável que indica ao conveyor antecedente que não se
                     encontra em posição de receber mais caixas*)
END_VAR
VAR
(*Triggers*)
    R_In    : R_TRIG; (*Rising trigger do sensor de entrada*)
    F_In    : F_TRIG; (*Falling trigger do sensor de saída*)
    R_Out : R_TRIG; (*Rising trigger do sensor de saída*)
    F_Out  : F_TRIG;(*Falling trigger do sensor de saída*)
    Counter :CTUD; (*Contador do Max_Box*)

(*Estados Internos*)
    Idle:BOOL:=TRUE; (*Parado e pronto a iniciar marcha*)
    Loading:BOOL :=FALSE; (*A carregar uma caixa*)
    Transporting:BOOL:=FALSE; (*A transportar uma caixa*)
    Unloading:BOOL:=FALSE; (*A descarregar uma caixa*)
END_VAR
```

```
(* Inicialização de Edge Detections e Counters*)

R_In (CLK := INPUT_IN);
F_Out (CLK := INPUT_EXIT);
F_In (CLK := INPUT_IN);
R_Out(CLK := INPUT_EXIT);

(*Contador de Max_Box*)

Counter(CU := F_In.Q, CD:=F_Out.Q, Reset := In_14, PV := MAX_BOX);

(*Implementação de Max_Box*)

IF Counter.QU THEN
    SEND_BUSY := TRUE;
END_IF;

IF NOT Counter.QU THEN
    SEND_BUSY :=FALSE;
END_IF;

(*Detecção de Alturas*)

IF IS_QUEUE THEN
    IF F_In.Q THEN
        COUNT := ROL (COUNT,1);
        QUEUE := SHL (QUEUE,1);       (* Desloca QUEUE para a esquerda introduzindo um 0 *)
        IF INPUT_IN_HIGH THEN
        QUEUE := QUEUE OR WORD#1; (* Modifica para 1 se a palete que entrou é alta *)
        END_IF;
    END_IF;
END_IF;

(*Controlo de Estados*)

IF Idle AND R_In.Q THEN
    Idle:=FALSE;
    Loading:=TRUE;
END_IF;

IF Loading AND F_In.Q THEN
    Loading:=FALSE;
    Transporting := TRUE;
END_IF;
```

```
IF Transporting AND R_Out.Q THEN
    Transporting := FALSE;
    Unloading :=TRUE;
END_IF;

IF Unloading AND F_Out.Q THEN
    Unloading := FALSE;
    IF NOT  Counter.QU THEN
        Idle:=TRUE;
    ELSE
        Transporting := TRUE;
    END_IF;
END_IF;

(*Afectação das saídas*)

OUTPUT:=((Loading OR Transporting OR Unloading) AND NOT ((RECEIVE_BUSY AND INPUT_EXIT)));
```

```
FUNCTION_BLOCK CONVEYOR_ROT
VAR_INPUT
(*Inputs Físicos*)
    INPUT_IN: BOOL; (*Sensor de Entrada*)
    INPUT_EXIT:BOOL; (*Sensor de Saída*)
    INPUT_ROTARY_IN:BOOL; (*Sensor de tapete rotativo na posição de carga*)
    INPUT_ROTARY_EXIT:BOOL;(*Sensor de tapete rotativo na posição de descarga*)

(*Inputs Software*)
    RECEIVE_BUSY:BOOL; (*Variável de entrada que indica que o próximo conveyor
                         não se encontra em posição de receber mais caixas*)
END_VAR
VAR_OUTPUT
(*Outputs Físicos*)
    OUTPUT_LINEAR: BOOL; (*Comando do motor do accionamento linear para a frente*)
    OUTPUT_LINEAR_BWD:BOOL;(*Comando do motor do accionamento linear para trás*)
    OUTPUT_ROTARY:BOOL; (*Comando do motor do accionamento rortativo*)

(*Outputs Software*)
    SEND_BUSY:BOOL; (*Variável que indica ao conveyor antecedente que
                         não se encontra em posição de receber mais caixas*)
END_VAR
VAR
(*Triggers*)
    R_In    : R_TRIG;(*Rising trigger do sensor de entrada*)
    F_In    : F_TRIG;(*Falling trigger do sensor de saída*)
    R_Out : R_TRIG; (*Rising trigger do sensor de saída*)
    F_Out  : F_TRIG;(*Falling trigger do sensor de saída*)
    R_In_Rotary : R_TRIG; (*Rising trigger do sensor de tapete em posição de carga*)
    R_Out_Rotary : R_TRIG; (*Rising trigger do sensor de tapete em posição de descarga*)
    R_Choose_Side : R_TRIG; (*Rising trigger da estado em que o conveyor
                              aguarda que seja definido o destino da caixa*)
    Counter :CTUD; (*Contador do Max_Box*)

(*Estados Internos*)
    Idle:BOOL:=TRUE; (*Parado e pronto a iniciar marcha*)
    Loading:BOOL :=FALSE;(*A carregar uma caixa*)
    Rotating1:BOOL:=FALSE; (*A rodar para a posição de descarga*)
    Unloading:BOOL:=FALSE;(*a descarregar uma caixa*)
    Rotating2:BOOL:=FALSE; (*A rodar para a posição de carga*)
    Choose_Side:BOOL:=FALSE; (*Estado de transição até que seja definida para onde vai a caixa*)
    Divert:BOOL:=FALSE; (*Variável que definie se a caixa vai para a direita ou esquerda*)
                         (*Pode-se generalizar como uma situação de divert*)
END_VAR
```

```
(* Inicialização de Edge Detections e Counters*)

R_In (CLK := INPUT_IN);
F_Out (CLK := INPUT_EXIT);
F_In (CLK := INPUT_IN);
R_Out(CLK := INPUT_EXIT);
R_In_Rotary(CLK := INPUT_ROTARY_IN);
R_Out_Rotary(CLK := INPUT_ROTARY_EXIT);

R_Choose_Side (CLK:= Choose_Side);

(*Controlo de Estados*)

IF Idle AND INPUT_IN THEN
    Idle:=FALSE;
    Loading:=TRUE;
END_IF;

IF Loading AND R_Out.Q  THEN
    Loading:=FALSE;
    Rotating1 := TRUE;
END_IF;

IF Rotating1 AND R_Out_Rotary.Q THEN
    Rotating1 :=FALSE;
    Choose_Side :=TRUE;
    Unloading:=TRUE;
END_IF;

IF R_Choose_Side.Q THEN
    IF (COUNT AND QUEUE) <> WORD#0 THEN  (* Se a palete que sai é alta ... *)
        Diverl := TRUE;    (*... será descarregada para a esquerda *)
    ELSE
        Diverl := FALSE;   (* senão, para a direita *)
    END_IF;
    COUNT := ROR(COUNT,1);          (* O contador só agora pode ser decrementado *)
END_IF;

IF Unloading AND RECEIVE_BUSY THEN
    Unloading := FALSE;
    Choose_Side:=FALSE;
    Diverl:=FALSE;
    Rotating2:=TRUE;
END_IF;

IF Rotating2 AND INPUT_ROTARY_IN THEN
    Rotating2:=FALSE;
    Idle:=TRUE;
END_IF;

(*Afectação das saídas*)

SEND_BUSY:= Rotating1 OR Rotating2 OR Unloading;
OUTPUT_LINEAR:=((Loading OR Unloading) AND NOT ((RECEIVE_BUSY AND INPUT_EXIT) OR Diverl));
OUTPUT_LINEAR_BWD:= Unloading AND Diverl;
OUTPUT_ROTARY:=(Rotating1 OR Unloading) AND NOT RECEIVE_BUSY;
```

Capítulo 5.5

```
PROGRAM ProceduralControl
VAR
    Conveyor0 : CONVEYOR;
    Conveyor1 : CONVEYOR;
    ConveyorRotary:  CONVEYOR_ROT;
    Conveyor2: CONVEYOR;
    Conveyor3: CONVEYOR;
    ID_Block:ID;
END_VAR
(*Conveyor Linear de Entrada*)
Conveyor0 (INPUT_IN := TRUE);
Conveyor0 (INPUT_EXIT := In_0);
Conveyor0 (RECEIVE_BUSY:= Conveyor1.SEND_BUSY);
Out_0 := Conveyor0.OUTPUT;
Conveyor0 (MAX_BOX:= 2);

(*Identificação de Caixas*)
ID_Block (INPUT_IN := In_1);
ID_Block (INPUT_IN_HIGH := In_2);

(*Conveyor Linear de Identificação de Caixas*)
Conveyor1 (NEW_BOX:=ID_Block.BOX_SENT);
Conveyor1 (BOX_HEIGHT_IN:=ID_Block.BOX_HEIGHT);
Conveyor1 (INPUT_IN := In_0);
Conveyor1 (INPUT_EXIT := In_3);
Conveyor1 (RECEIVE_BUSY:=ConveyorRotary.SEND_BUSY);
Out_1 := Conveyor1.OUTPUT;
Conveyor1 (MAX_BOX:= 2);

(*Conveyor Rotativo*)
ConveyorRotary (NEW_BOX :=Conveyor1.BOX_SENT);
ConveyorRotary (BOX_HEIGHT_IN := Conveyor1.BOX_HEIGHT_OUT);
ConveyorRotary (INPUT_IN := In_3);
ConveyorRotary (INPUT_EXIT := In_6);
ConveyorRotary (INPUT_ROTARY_IN := In_4);
ConveyorRotary (INPUT_ROTARY_EXIT := In_5);
Out_2 := ConveyorRotary.OUTPUT_LINEAR;
Out_3 := ConveyorRotary.OUTPUT_LINEAR_BWD;
Out_4 := ConveyorRotary.OUTPUT_ROTARY;
ConveyorRotary (RECEIVE_BUSY:= (Conveyor2.SEND_BUSY OR Conveyor3.SEND_BUSY));

(*Conveyor Linear de Saída Esquerdo*)
Conveyor2 (INPUT_IN := In_8);
Conveyor2 (INPUT_EXIT := In_10);
Out_5 := Conveyor2.OUTPUT;
Conveyor2 (MAX_BOX:= 1);
Conveyor2 (NEW_BOX:=ConveyorRotary.BOX_SENT_LEFT);
Conveyor2 (BOX_HEIGHT_IN:=ConveyorRotary.BOX_HEIGHT_OUT);

(*Conveyor Linear de Saída Direito*)
Conveyor3 (INPUT_IN := In_7);
Conveyor3 (INPUT_EXIT := In_9);
Out_6 := Conveyor3.OUTPUT;
Conveyor3 (MAX_BOX:= 1);
Conveyor3 (NEW_BOX:=ConveyorRotary.BOX_SENT_RIGHT);
Conveyor3 (BOX_HEIGHT_IN:=ConveyorRotary.BOX_HEIGHT_OUT);
```

```
FUNCTION_BLOCK ID
VAR_INPUT
    INPUT_IN: BOOL; (*Sensor detecçao Caixa*)
    INPUT_IN_HIGH:BOOL; (*Sensor detecçao Caixa Alta*)
END_VAR
VAR_OUTPUT
    BOX_HEIGHT:INT:=0; (*Altura da caixa*)
    BOX_SENT:BOOL:=FALSE; (*Evento de caixa identificada*)
END_VAR
VAR
    R_In    : R_TRIG; (*Rising  trigger do sensor de entrada*)
    F_In:F_TRIG; (*Falling trigger do sensor de caixa alta de entrada*)
    F_Identifying :F_TRIG; (*Falling trigger da flag Identifying*)

    Idle:BOOL :=TRUE; (*Estado idle*)
    Identifying :BOOL:= FALSE; (*Estado de caixa a ser identificada*)
END_VAR


R_In (CLK := INPUT_IN);
F_In (CLK:=  INPUT_IN);
F_Identifying (CLK:= Identifying);

IF Idle AND R_In.Q THEN
    Idle:=FALSE;
    BOX_HEIGHT := 1;
    Identifying:=TRUE;
END_IF;

IF Identifying THEN
    IF INPUT_IN_HIGH THEN
    BOX_HEIGHT := 2;
    END_IF;
END_IF;

IF Identifying AND F_In.Q THEN
    Identifying := FALSE;
    Idle :=TRUE;
END_IF;

BOX_SENT := F_Identifying.Q;
```

```
FUNCTION_BLOCK CONVEYOR
VAR_INPUT
(*Inputs Físicos*)
    INPUT_IN: BOOL; (*Sensor de Entrada*)
    INPUT_EXIT:BOOL; (*Sensor de Saída*)

(*Inputs Software*)
    MAX_BOX:INT; (*Máximo de caixas presentes simultaneamente no conveyor*)
    RECEIVE_BUSY:BOOL; (*Variável de entrada que indica que o próximo conveyor
                         não se encontra em posição de receber mais caixas*)
    NEW_BOX:BOOL;(*Evento que indica que uma nova caixa entrou no conveyor*)
    BOX_HEIGHT_IN:INT:=0;(*Altura da caixa que entrou*)
END_VAR
VAR_OUTPUT
(*Outputs Físicos*)
    OUTPUT: BOOL; (*Comando do motor do conveyor*)

(*Outputs Software*)
    SEND_BUSY:BOOL;(*Variável que indica ao conveyor antecedente que não
                     se encontra em posição de receber mais caixas*)
    BOX_HEIGHT_OUT:INT;(*Altura da caixa que sai*)
    BOX_SENT:BOOL:=FALSE;(*Evento de caixa enviada para o conveyor seguinte*)
END_VAR
VAR
(*Triggers*)
    R_In    : R_TRIG; (*Rising trigger do sensor de entrada*)
    F_In    : F_TRIG; (*Falling trigger do sensor de saída*)
    R_Out : R_TRIG; (*Rising trigger do sensor de saída*)
    F_Out  : F_TRIG;(*Falling trigger do sensor de saída*)
    Counter :CTUD; (*Contador do Max_Box*)
    R_New_Box :R_TRIG;(*Rising trigger do evento New_Box*)
    F_Unloading:F_TRIG; (*Falling trigger do estado Unloading*)
    Pulse_Box_Sent :TP; (*Timer Pulse que garante que a informação de caixa
                         enviada é correctamente recebida pelo conveyor seguinte*)

(*Estados Internos*)
    Idle:BOOL:=TRUE; (*Parado e pronto a iniciar marcha*)
    Loading:BOOL :=FALSE; (*A carregar uma caixa*)
    Transporting:BOOL:=FALSE; (*A transportar uma caixa*)
    Unloading:BOOL:=FALSE; (*A descarregar uma caixa*)

(*Lista de caixas e suas alturas*)
    Count:WORD:=16#8000; (*Contador caixas*)
    Queue:WORD; (*Fila caixas*)
    Is_Queue: BOOL; (*Flag que faz com que a queue / count sejam actualizadas
                     à saida do conveyor.*)
END_VAR
```

```
(* Inicialização de Edge Detections e Counters*)

R_In (CLK := INPUT_IN);
F_Out (CLK := INPUT_EXIT);
F_In (CLK := INPUT_IN);
R_Out(CLK := INPUT_EXIT);
R_New_Box(CLK:=NEW_BOX);
F_Unloading (CLK:=Unloading);
Pulse_Box_Sent (IN:=F_Unloading.Q, PT:=T#1s);

(*Contador de Max_Box*)

Counter(CU := F_In.Q, CD:=F_Out.Q, Reset := In_14, PV := MAX_BOX);

(*Implementação de Max_Box*)

IF Counter.QU THEN
    SEND_BUSY := TRUE;
END_IF;

IF NOT Counter.QU THEN
    SEND_BUSY :=FALSE;
END_IF;

(*Gestão da Informação de Alturas*)

IF R_New_Box.Q THEN
    Is_Queue:=TRUE;
    Count := ROL (Count,1);
    Queue := SHL (Queue,1);
    IF BOX_HEIGHT_IN = 2 THEN
        Queue := Queue OR WORD#1;
    END_IF;
END_IF;

IF Is_Queue AND F_Out.Q THEN
        IF (Count AND Queue) <> WORD#0 THEN
        BOX_HEIGHT_OUT := 2;
        ELSE
        BOX_HEIGHT_OUT := 1;
        END_IF;
        Count := ROR(Count,1);
        Is_Queue:=FALSE;
END_IF;
```

```
(*Controlo de Estados*)

IF Idle AND INPUT_IN THEN
    Idle:=FALSE;
    Loading:=TRUE;
END_IF;

IF Loading AND F_In.Q THEN
    Loading:=FALSE;
    Transporting := TRUE;
END_IF;

IF Transporting AND R_Out.Q THEN
    Transporting := FALSE;
    Unloading :=TRUE;
END_IF;

IF Unloading AND F_Out.Q THEN
    Unloading := FALSE;
    IF NOT  Counter.QU THEN
        Idle:=TRUE;
    ELSE
        Transporting := TRUE;
    END_IF;
END_IF;

(*Afectação das saídas*)

BOX_SENT := Pulse_Box_Sent.Q;
OUTPUT:=((Loading OR Transporting OR Unloading) AND NOT ((RECEIVE_BUSY AND INPUT_EXIT)));
```

```
FUNCTION_BLOCK CONVEYOR_ROT
VAR_INPUT
(*Inputs Físicos*)
    INPUT_IN: BOOL; (*Sensor de Entrada*)
    INPUT_EXIT:BOOL; (*Sensor de Saída*)
    INPUT_ROTARY_IN:BOOL; (*Sensor de tapete rotativo na posição de carga*)
    INPUT_ROTARY_EXIT:BOOL;(*Sensor de tapete rotativo na posição de descarga*)

(*Inputs Software*)
    RECEIVE_BUSY:BOOL; (*Variável de entrada que indica que o próximo conveyor
                         não se encontra em posição de receber mais caixas*)
    NEW_BOX:BOOL; (*Evento que indica que uma nova caixa entrou no conveyor*)
    BOX_HEIGHT_IN:INT:=0; (*Altura da caixa que entrou*)
END_VAR
VAR_OUTPUT
(*Outputs Físicos*)
    OUTPUT_LINEAR: BOOL; (*Comando do motor do accionamento linear para a frente*)
    OUTPUT_LINEAR_BWD:BOOL;(*Comando do motor do accionamento linear para trás*)
    OUTPUT_ROTARY:BOOL; (*Comando do motor do accionamento rortativo*)

(*Outputs Software*)
    SEND_BUSY:BOOL; (*Variável que indica ao conveyor antecedente que não
                      se encontra em posição de receber mais caixas*)
    BOX_SENT_LEFT:BOOL; (*Evento de caixa enviada para para a esquerda*)
    BOX_SENT_RIGHT:BOOL; (*Evento de caixa enviada para para a direita*)
    BOX_HEIGHT_OUT:INT;(*Altura da caixa que sai*)
END_VAR
VAR
(*Triggers*)
    R_In    : R_TRIG;(*Rising trigger do sensor de entrada*)
    F_In    : F_TRIG;(*Falling trigger do sensor de saída*)
    R_Out : R_TRIG; (*Rising trigger do sensor de saída*)
    F_Out  : F_TRIG;(*Falling trigger do sensor de saída*)
    R_In_Rotary : R_TRIG; (*Rising trigger do INPUT_ROTARY_IN*)
    R_Out_Rotary : R_TRIG; (*Rising trigger do INPUT_ROTARY_OUT*)
    R_New_Box :R_TRIG; (*Rising trigger do evento New_Box*)
    F_Unloading :F_TRIG; (*Falling trigger do estado Unloading*)
    Pulse_Box_Sent :TP; (*Timer Pulse que garante que a informação de caixa
                         enviada é correctamente recebida pelo conveyor seguinte*)


(*Estados Internos*)
    Idle:BOOL:=TRUE; (*Parado e pronto a iniciar marcha*)
    Loading:BOOL :=FALSE;(*A carregar uma caixa*)
    Rotating1:BOOL:=FALSE; (*A rodar para a posição de descarga*)
    Unloading:BOOL:=FALSE;(*a descarregar uma caixa*)
    Rotating2:BOOL:=FALSE; (*A rodar para a posição de carga*)

(*Lista de caixas e suas alturas*)
    Count:WORD:=16#8000; (*Contador caixas*)
    Queue:WORD; (*Fila caixas*)
    Is_Queue: BOOL; (*Flag que faz com que a queue / count sejam actualizadas
                     à saida do conveyor.*)
END_VAR
```

```
(* Inicialização de Edge Detections e Counters*)

R_In (CLK := INPUT_IN);
F_Out (CLK := INPUT_EXIT);
F_In (CLK := INPUT_IN);
R_Out(CLK := INPUT_EXIT);
R_In_Rotary(CLK := INPUT_ROTARY_IN);
R_Out_Rotary(CLK := INPUT_ROTARY_EXIT);
R_New_Box (CLK:=NEW_BOX);
F_Unloading (CLK := Unloading);
Pulse_Box_Sent (IN:=F_Unloading.Q, PT:=T#1s);

(*Gestão da Informação de Alturas*)

IF R_New_Box.Q THEN
    Is_Queue:=TRUE;
    Count := ROL (Count,1);
    Queue := SHL (Queue,1);
    IF BOX_HEIGHT_IN = 2 THEN
        Queue := Queue OR WORD#1;
    END_IF;
END_IF;

IF Is_Queue AND F_Unloading.Q THEN
        IF (Count AND Queue) <> WORD#0 THEN
        BOX_HEIGHT_OUT := 2;
        ELSE
        BOX_HEIGHT_OUT := 1;
        END_IF;
        Count := ROR(Count,1);
        Is_Queue:=FALSE;
END_IF;
```

```
(*Controlo de Estados*)

IF Idle AND INPUT_IN THEN
    Idle:=FALSE;
    Loading:=TRUE;
END_IF;

IF Loading AND R_Out.Q  THEN
    Loading:=FALSE;
    Rotating1 := TRUE;
END_IF;

IF Rotating1 AND R_Out_Rotary.Q THEN
    Rotating1 :=FALSE;
    Unloading:=TRUE;
END_IF;

IF Unloading AND RECEIVE_BUSY THEN
    Unloading := FALSE;
    Rotating2:=TRUE;
END_IF;

IF Rotating2 AND INPUT_ROTARY_IN THEN
    Rotating2:=FALSE;
    Idle:=TRUE;
END_IF;


(*Afectação das saídas*)

SEND_BUSY:= Rotating1 OR Rotating2 OR Unloading;
OUTPUT_LINEAR:=(Loading OR (Unloading AND BOX_HEIGHT_IN=1));
OUTPUT_LINEAR_BWD:= Unloading AND BOX_HEIGHT_IN = 2;
OUTPUT_ROTARY:=(Rotating1 OR Unloading) AND NOT RECEIVE_BUSY;
BOX_SENT_LEFT:= BOX_HEIGHT_IN = 1 AND Pulse_Box_Sent.Q;
BOX_SENT_RIGHT:= BOX_HEIGHT_IN = 2 AND Pulse_Box_Sent.Q;
BOX_HEIGHT_OUT := BOX_HEIGHT_IN;
```

Anexo 3 – CoDeSys – Código Batching

Capítulo 6.3

```
PROGRAM Main
VAR
    DosingRed:DOSING;
    DosingGreen:DOSING;
    DosingBlue:DOSING;
    Mixing1:MIXING;
END_VAR

(*Doseador Vermelho*)
DosingRed (REQ_DOSE:= In_12); (*Simulação de pedido de doseamento de tinta*)
DosingRed (REQ_DELIVERY := In_14); (*Simulação de pedido de entrega de tinta doseada*)
DosingRed (LOW_LEVEL:= In_0);
DosingRed (MID_LEVEL :=In_1);
DosingRed (HIGH_LEVEL:=In_2);
DosingRED (DOSE := TRUE); (*Simuação de pedido de dose completa*)
Out_0 := DosingRed.UPSTREAM_VALVE;
Out_1 := DosingRed.DOWNSTREAM_VALVE;

(*Doseador Verde*)
DosingGreen (REQ_DOSE:= In_12); (*Simulação de pedido de doseamento de tinta*)
DosingGreen (REQ_DELIVERY := In_14); (*Simulação de pedido de entrega de tinta*)
DosingGreen (LOW_LEVEL:= In_3);
DosingGreen (MID_LEVEL :=In_4);
DosingGreen (HIGH_LEVEL:=In_5);
DosingGreen (DOSE := FALSE); (*Simuação de pedido de meia dose*)
Out_2 := DosingGreen.UPSTREAM_VALVE;
Out_3 := DosingGreen.DOWNSTREAM_VALVE;

(*Doseador Azul*)
DosingBlue (REQ_DOSE:= In_12); (*Simulação de pedido de doseamento de tinta*)
DosingBlue (REQ_DELIVERY := In_14); (*Simulação de pedido de entrega de tinta*)
DosingBlue (LOW_LEVEL:= In_6);
DosingBlue (MID_LEVEL :=In_7);
DosingBlue (HIGH_LEVEL:=In_8);
DosingBlue (DOSE := TRUE); (*Simuação de pedido de dose completa*)
Out_4 := DosingBlue.UPSTREAM_VALVE;
Out_5 := DosingBlue.DOWNSTREAM_VALVE;

(*Misturador*)
Mixing1 (REQ_MIX:= DosingRed.CNF_COMPLETE AND DosingGreen.CNF_COMPLETE AND DosingBlue.CNF_COMPLETE);
Mixing1 (MIX_TIME := 5500);
Mixing1 (LOW_LEVEL:= In_9);
Mixing1 (HIGH_LEVEL := In_10);
Out_6 := Mixing1.MIXER;
Out_7 := Mixing1.DOWNSTREAM_VALVE;
```

```
FUNCTION_BLOCK DOSING
VAR_INPUT
(*Variáveis Físicas*)
    LOW_LEVEL:BOOL; (*Sensor de nível baixo*)
    MID_LEVEL:BOOL; (*Sensor de nível médio*)
    HIGH_LEVEL:BOOL; (*Sensor de nível alto*)

(*Variáveis Software*)
    REQ_DOSE:BOOL; (*Pedido para que seja processada a Dose*)
    REQ_DELIVERY:BOOL; (*Pedido para que seja entregue a dose*)
    DOSE:BOOL; (*Definição de quantidade -> 1 DOSE = TRUE | 1/2 DOSE = FALSE*)
END_VAR
VAR_OUTPUT
(*Variáveis Físicas*)
    UPSTREAM_VALVE:BOOL; (*Comando da válvula a montante*)
    DOWNSTREAM_VALVE:BOOL; (*Comando da válvula a jusante*)

(*Variáveis Software*)
    CNF_COMPLETE:BOOL; (*Confirmação de término do doseamento*)
END_VAR
VAR
(*Triggers*)
    R_Start: R_TRIG; (* Transição ascendente do Start*)

(*Estados Internos*)
    Inactivo  : BOOL := TRUE;  (*Estado de doseador Idle*)
    Enche  : BOOL := FALSE; (* Estado de doseador a encher*)
    Cheio :BOOL :=FALSE; (*Estado de dosesador cheio*)
    Esvazia : BOOL := FALSE; (* Estado de doseador a esvaziar*)

(*Varíaveis Auxiliares*)
    Dose_1 : BOOL; (* Cópia da variável Dose, para garantir que mesmo que seja dada uma ordem
                      durante o processamento de outra, esta não é comprometida *)
END_VAR
```

```
(*Inicialização de Edge Detections*)
R_Start (CLK :=REQ_DOSE);

(*Definição de quantidade*)
IF R_Start.Q THEN
Dose_1 := Dose;
END_IF;

(*Controlo de Estados*)
IF Inactivo AND R_Start.Q THEN (* Quando é requisitada tinta *)
    Inactivo := FALSE;
    Enche := TRUE;
END_IF;

IF Enche AND (HIGH_LEVEL OR MID_LEVEL AND NOT Dose_1) THEN
(* Quando a quantidade desejada de tinta é atingida *)
    Enche := FALSE;
    Cheio := TRUE;
END_IF;

IF Cheio AND REQ_DELIVERY THEN (* Quando há autorização para descarregar *)
    Cheio:= FALSE;
    Esvazia := TRUE;
END_IF;

IF Esvazia AND NOT LOW_LEVEL THEN (* Quando fica vazio *)
    Esvazia := FALSE;
    Inactivo := TRUE;
END_IF;

(* Afectação das Saídas *)
UPSTREAM_VALVE := Enche;
DOWNSTREAM_VALVE := Esvazia;
CNF_COMPLETE := Inactivo;
```

```
FUNCTION_BLOCK MIXING
VAR_INPUT
(*Variáveis Físicas*)
    LOW_LEVEL:BOOL; (*Sensor de nível baixo*)
    HIGH_LEVEL:BOOL; (*Sensor de nível alto*)

(*Variáveis Software*)
    REQ_MIX:BOOL; (*Pedido de mistura*)
    REQ_DELIVERY: BOOL;(*Pedido de entrega, independentemente do tempo de mistura*)
    MIX_TIME:INT; (*Tempo de mistura*)
END_VAR
VAR_OUTPUT
(*Variáveis Físicas*)
    DOWNSTREAM_VALVE:BOOL; (*Comando da válvula a jusante*)
    MIXER:BOOL; (*Comando do accionamento do misturador*)

(*Variáveis Software*)
    CNF_COMPLETE:BOOL; (*Confirmação de término da mistura*)
END_VAR
VAR
(*Triggers*)
    R_Start: R_TRIG; (* Transição ascendente do Start*)

(*Timers*)
    Timer_mistura : TON; (* Temporizador On-delay *)

(*Estados Internos*)
    Inactivo  : BOOL := TRUE; (* Estado de misturador idle*)
    Enche : BOOL := FALSE; (* Estado de misturador a encher*)
    Mistura : BOOL := FALSE; (* Estado de misturador a misturar*)
    Descarga : BOOL := FALSE; (* Estado de misturador a descarregar*)

(*Varíaveis Auxiliares*)
    Mix_Time_1: INT;(* Cópia da variável Mix_Time, para garantir que mesmo que seja dada uma ordem
                       durante o processamento de outra, esta não é comprometida *)
END_VAR
```

```
(*Inicialização de Edge Detections*)
R_Start (CLK :=REQ_MIX);

(*Definição do tempo de mistura*)
IF R_Start.Q THEN
Mix_Time_1 := MIX_TIME;
END_IF;

(*Controlo de Estados*)
IF Inactivo AND LOW_LEVEL THEN (* Quando começa a receber tinta *)
    Inactivo := FALSE;
    Enche := TRUE;
END_IF;

IF Enche AND R_Start.Q THEN (* Quando recebe informação que foi entregue toda a tinta *)
    Enche := FALSE;
    Mistura := TRUE;
END_IF;

IF Mistura AND Timer_mistura.Q THEN (* Ao fim do tempo de mistura *)
    Mistura := FALSE;
    Descarga := TRUE;
END_IF;

IF Descarga AND NOT LOW_LEVEL THEN (* Quando vazio *)
    Descarga := FALSE;
    Inactivo := TRUE;
END_IF;

(*Afectação das Saídas*)
DOWNSTREAM_VALVE:=Descarga OR REQ_DELIVERY;
MIXER:=Mistura;
CNF_COMPLETE := Inactivo;

(*Temporizador com base no tempo de mistura definido*)
Timer_mistura(IN := Mistura, PT := INT_TO_TIME (Mix_Time_1));
```

Capítulo 6.5

```
PROGRAM Main
VAR
    DosingRed:DOSING;
    DosingGreen:DOSING;
    DosingBlue:DOSING;
    Mixing1:MIXING;
    Recipe_Manager:RECIPE;
END_VAR

(*Doseador Vermelho*)
DosingRed (RECIPE_LEVEL := Recipe_Manager.RED_LEVEL);
DosingRed (LOW_LEVEL:= In_0);
DosingRed (MID_LEVEL :=In_1);
DosingRed (HIGH_LEVEL:=In_2);
DosingRed (REQ_DOSE:= Recipe_Manager.DOSE);
DosingRed (REQ_DELIVERY := Mixing1.IDLE);
Out_0 := DosingRed.UPSTREAM_VALVE;
Out_1 := DosingRed.DOWNSTREAM_VALVE;

(*Doseador Verde*)
DosingGreen (RECIPE_LEVEL := Recipe_Manager.GREEN_LEVEL);
DosingGreen (LOW_LEVEL:= In_3);
DosingGreen (MID_LEVEL :=In_4);
DosingGreen (HIGH_LEVEL:=In_5);
DosingGreen (REQ_DOSE:= Recipe_Manager.DOSE);
DosingGreen (REQ_DELIVERY := Mixing1.IDLE);
Out_2 := DosingGreen.UPSTREAM_VALVE;
Out_3 := DosingGreen.DOWNSTREAM_VALVE;

(*Doseador Azul*)
DosingBlue (RECIPE_LEVEL := Recipe_Manager.BLUE_LEVEL);
DosingBlue (LOW_LEVEL:= In_6);
DosingBlue (MID_LEVEL :=In_7);
DosingBlue (HIGH_LEVEL:=In_8);
DosingBlue (REQ_DOSE:= Recipe_Manager.DOSE);
DosingBlue (REQ_DELIVERY := Mixing1.IDLE);
Out_4 := DosingBlue.UPSTREAM_VALVE;
Out_5 := DosingBlue.DOWNSTREAM_VALVE;


(*Misturador*)
Mixing1 (LOW_LEVEL:= In_9);
Mixing1 (HIGH_LEVEL := In_10);
Mixing1 (MIX_TIME := Recipe_Manager.MIX_TIME);
Mixing1 (REQ_MIX:= DosingRed.CNF_COMPLETE AND DosingGreen.CNF_COMPLETE AND DosingBlue.CNF_COMPLETE);
Out_6 := Mixing1.MIXER;
Out_7 := Mixing1.DOWNSTREAM_VALVE;

(*Recipe Manager*)
Recipe_Manager (COMPLETE := DosingRed.CNF_COMPLETE AND DosingGreen.CNF_COMPLETE AND DosingBlue.CNF_COMPLETE);
Recipe_Manager (REQ:= In_12);
Recipe_Manager (COLOR :=2);
Recipe_Manager (QUANTITY := 1);
```

```
FUNCTION_BLOCK DOSING
VAR_INPUT
(*Variáveis Físicas*)
    LOW_LEVEL:BOOL; (*Sensor de nível baixo*)
    MID_LEVEL:BOOL; (*Sensor de nível médio*)
    HIGH_LEVEL:BOOL; (*Sensor de nível alto*)

(*Variáveis Software*)
    REQ_DOSE:BOOL; (*Pedido para que seja processada a dose*)
    REQ_DELIVERY:BOOL; (*Pedido para que seja entregue a dose*)
    RECIPE_LEVEL:INT; (*Definição da quantidade de tinta*)
END_VAR
VAR_OUTPUT
(*Variáveis Físicas*)
    UPSTREAM_VALVE:BOOL; (*Comando da válvula a montante*)
    DOWNSTREAM_VALVE:BOOL; (*Comando da válvula a jusante*)

(*Variáveis Software*)
    CNF_COMPLETE:BOOL; (*Confirmação de término do doseamento*)
END_VAR

VAR
(*Triggers*)
    R_Start: R_TRIG; (* Transição ascendente do Start*)

(*Estados Internos*)
    Inactivo  : BOOL := TRUE;  (*Estado de doseador Idle*)
    Enche  : BOOL := FALSE; (* Estado de doseador a encher*)
    Cheio :BOOL :=FALSE; (*Estado de dosesador cheio*)
    Esvazia : BOOL := FALSE; (* Estado de doseador a esvaziar*)

(*Varíaveis Auxiliares*)
    Recipe_Level1 : INT; (* Cópia da variável RECIPE_LEVEL, para garantir que mesmo que
        seja dada uma ordem durante o processamento de outra, esta não é comprometida *)
END_VAR
```

```
(*Inicialização de Edge Detections*)
R_Start (CLK :=REQ_DOSE);

(*Controlo de Estados*)
IF Inactivo AND R_Start.Q AND RECIPE_LEVEL <>0  THEN (* Quando produção requisita tinta *)
    Recipe_Level1 := RECIPE_LEVEL;
    Inactivo := FALSE;
    Enche := TRUE;
END_IF;

IF Enche AND (HIGH_LEVEL AND Recipe_Level1 = 2 OR MID_LEVEL AND  Recipe_Level1 = 1) THEN
(* Quando cheio *)
    Enche := FALSE;
    Cheio := TRUE;
END_IF;

IF Cheio AND REQ_DELIVERY THEN (* Quando produção autoriza a descarregar *)
    Cheio:= FALSE;
    Esvazia := TRUE;
END_IF;

IF Esvazia AND NOT LOW_LEVEL THEN (* Quando fica vazio *)
    Esvazia := FALSE;
    Inactivo := TRUE;
END_IF;

(* Afectação das Saídas *)
UPSTREAM_VALVE := Enche;
DOWNSTREAM_VALVE := Esvazia;
CNF_COMPLETE := Inactivo;

FUNCTION_BLOCK MIXING
VAR_INPUT
(*Variáveis Físicas*)
    LOW_LEVEL:BOOL; (*Sensor de nível baixo*)
    HIGH_LEVEL:BOOL; (*Sensor de nível alto*)

(*Variáveis Software*)
    REQ_MIX:BOOL; (*Pedido de mistura*)
    REQ_DELIVERY: BOOL;(*Pedido de entrega, independentemente do tempo de mistura*)
    MIX_TIME:INT; (*Tempo de mistura*)
END_VAR
VAR_OUTPUT
(*Variáveis Físicas*)
    DOWNSTREAM_VALVE:BOOL; (*Comando da válvula a jusante*)
    MIXER:BOOL; (*Comando do accionamento do misturador*)

(*Variáveis Software*)
    IDLE: BOOL; (*Informação de que o misturador está em espera*)
    COMPLETE:BOOL := FALSE; (*Confirmação de término da mistura*)
END_VAR
```

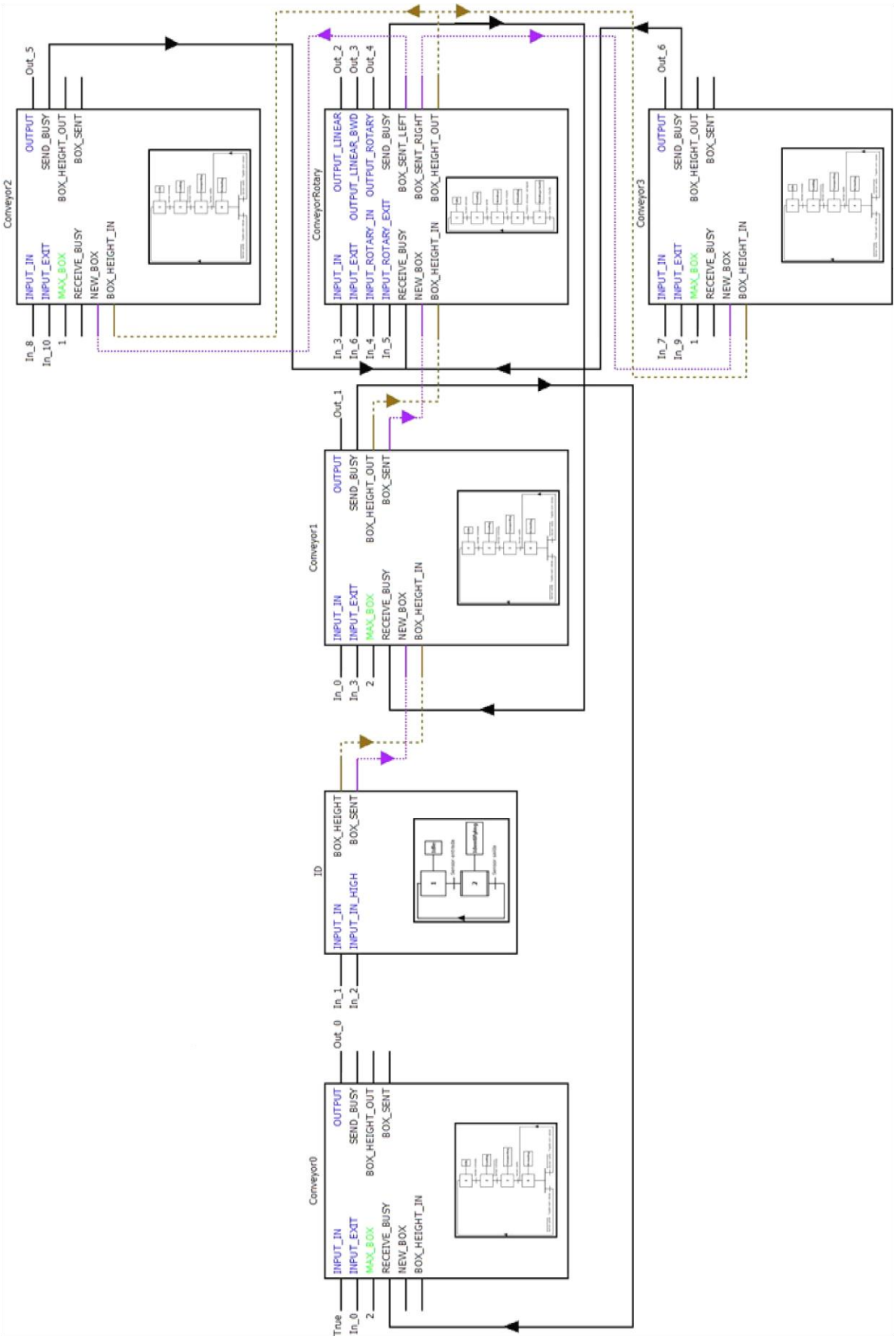
```
VAR
(*Triggers*)
    R_Start: R_TRIG; (* Transição ascendente do Start*)

(*Timers*)
    Timer_mistura : TON; (* Temporizador On-delay *)

(*Estados Internos*)
    Inactivo  : BOOL := TRUE; (* Estado de misturador idle*)
    Enche  : BOOL := FALSE; (* Estado de misturador a encher*)
    Mistura : BOOL := FALSE; (* Estado de misturador a misturar*)
    Descarga : BOOL := FALSE; (* Estado de misturador a descarregar*)

(*Varíaveis Auxiliares*)
    Mix_Time1: INT;(* Cópia da variável Mix_Time, para garantir que mesmo que seja dada uma ordem
                      durante o processamento de outra, esta não é comprometida *)

(*Triggers*)
    F_Descarga : F_TRIG; (*Falling trigger da descarga*)
    Pulse_Descarga : TP; (*Pulso que mantém o sinal de descarga terminada activo durante 2seg*)
END_VAR

(*Inicialização de Edge Detections*)
R_Start (CLK :=REQ_MIX);
F_Descarga (CLK := Descarga);
Pulse_Descarga (IN:= F_Descarga.Q, PT:=TIME#2s);

COMPLETE := Pulse_Descarga.Q;

(*Definição do tempo de mistura*)
IF R_Start.Q THEN
Mix_Time1 := MIX_TIME;
END_IF;

(*Controlo de Estados*)
IF Inactivo AND LOW_LEVEL THEN  (* Quando começa a receber tinta *)
    COMPLETE := FALSE;
    Inactivo := FALSE;
    Enche := TRUE;
END_IF;

IF Enche AND REQ_MIX THEN
(* Quando produção informa que foi entregue toda a tinta *)
    Enche := FALSE;
    Mistura := TRUE;
END_IF;

IF Mistura AND Timer_mistura.Q THEN
(* Ao fim do tempo de mistura *)
    Mistura := FALSE;
    Descarga := TRUE;
END_IF;

IF Descarga AND NOT LOW_LEVEL THEN  (* Quando vazio *)
    Descarga := FALSE;
    Inactivo := TRUE;
END_IF;
```

```
(*Afectação das Saídas*)
DOWNSTREAM_VALVE:=Descarga OR REQ_DELIVERY;
MIXER:=Mistura;
IDLE := Inactivo;

(*Temporizador com base no tempo de mistura definido*)
Timer_mistura(IN := Mistura, PT := INT_TO_TIME (Mix_Time1));


FUNCTION_BLOCK RECIPE
VAR_INPUT
(*Variáveis Software*)
    REQ:BOOL; (*Pedido de produção de tinta*)
    COMPLETE:BOOL; (*Informção de que terminou um ciclo de produção de tinta*)
    QUANTITY:INT; (*Definição da quantidade de doses a produzir*)
    COLOR:INT; (*Definição da cor de tinta a produzir*)
END_VAR
VAR_OUTPUT
(*Variáveis Software*)
    DOSE:BOOL := FALSE; (*Pedido de arranque para os doseadores*)
    RED_LEVEL:INT; (*Quantidade de tinta vermelha*)
    GREEN_LEVEL:INT; (*Quantidade de tinta verde*)
    BLUE_LEVEL:INT; (*Quantidade de tinta azul*)
    MIX_TIME:INT; (*Tempo de mistura*)
    QUANTITY_LEFT:INT; (*Quantidade de ciclos em falta*)
END_VAR
VAR
(*Triggers*)
    R_Req: R_TRIG;
    R_Complete: R_TRIG;
END_VAR
```

```
(*Inicialização de Edge Detections*)
R_Req (CLK:=REQ);
R_Complete (CLK:= COMPLETE);

(*Definição de Receita*)
IF R_Req.Q  AND QUANTITY_LEFT = 0 THEN
    QUANTITY_LEFT := QUANTITY;
    DOSE := TRUE;
    IF COLOR = 1 THEN
        RED_LEVEL := 1;
        GREEN_LEVEL := 1;
        BLUE_LEVEL := 0;
    END_IF;
    IF COLOR = 2 THEN
        RED_LEVEL := 1;
        GREEN_LEVEL := 0;
        BLUE_LEVEL := 1;
    END_IF;
    IF COLOR = 3 THEN
        RED_LEVEL := 0;
        GREEN_LEVEL := 1;
        BLUE_LEVEL := 1;
    END_IF;
    IF COLOR = 4 THEN
        RED_LEVEL := 1;
        GREEN_LEVEL :=1;
        BLUE_LEVEL := 1;
    END_IF;
    (*Podem ser configuradas 19 cores
     diferentes com a combinação dos vários sensores de nível*)
    MIX_TIME:=5500;
END_IF;

(*Contador de Ciclos*)
IF R_Complete.Q AND  QUANTITY_LEFT <> 0 THEN
    QUANTITY_LEFT := QUANTITY_LEFT - 1;
END_IF;

(*Final de Ciclo*)
IF  QUANTITY_LEFT = 0 THEN
    DOSE:=FALSE;
    MIX_TIME:=0;
END_IF;
```

Anexo 4 – Padrões de desenho - Diagramas de blocos funcionais finais

Anexo 4.1 - Sorting



Anexo 4.2 - Batching