SENAI CIMATEC

PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM COMPUTACIONAL E TECNOLOGIA INDUSTRIAL

Mestrado em Modelagem Computacional e Tecnologia Industrial

Projeto de Dissertação

Modelagem de sistemas para controladores lógicos programáveis utilizando o desenvolvimento baseado em componentes e IEC-61131-3: Aplicação em indústria de petróleo e gás

Apresentada por: Nuno César Silva Mattos
Orientador: Dr. Roberto Luiz de Souza Monteiro

Agosto de 2017

Nuno César Silva Mattos

Modelagem de sistemas para controladores lógicos programáveis utilizando o desenvolvimento baseado em componentes e IEC-61131-3: Aplicação em indústria de petróleo e gás

Projeto de Dissertação apresentado ao Programa de Pós-graduação em Modelagem Computacional e Tecnologia Industrial, Curso de Mestrado em Modelagem Computacional e Tecnologia Industrial do SENAI CIMATEC, como requisito parcial para a obtenção do título de **Mestre em Modelagem Computacional e Tecnologia Industrial**.

Área de conhecimento: Interdisciplinar

Orientador: Dr. Roberto Luiz de Souza Monteiro
SENAI CIMATEC

Salvador
SENAI CIMATEC
2017

Ficha catalográfica elaborada pelo Centro Universitário SENAI CIMATEC

M435m Mattos, Nuno César Silva

Modelagem de sistemas para controladores lógicos programáveis utilizando o desenvolvimento baseado em componentes e IEC 61131-3: aplicação em indústria de petróleo e gás / Nuno César Silva Mattos. – Salvador, 2017.

131 f. : il. color.

Orientador: Prof. Dr. Roberto Luiz de Souza Monteiro.

Dissertação (Mestrado em Modelagem Computacional e Tecnologia Industrial) – Programa de Pós-Graduação, Centro Universitário SENAI CIMATEC, Salvador, 2017.
Inclui referências.

1. Componentes de software. 2. Controladores lógicos programáveis. 3. Engenharia de software. 4. IEC 61131-3. I. Centro Universitário SENAI CIMATEC. II. Monteiro, Roberto Luiz de Souza. III. Título.

CDD: 005.1

SENAI CIMATEC

Programa de Pós-graduação em Modelagem Computacional e Tecnologia Industrial

Mestrado em Modelagem Computacional e Tecnologia Industrial

A Banca Examinadora, constituída pelos professores abaixo listados, leram e recomendam a aprovação do Projeto de Dissertação, intitulada "Modelagem de sistemas para controladores lógicos programáveis utilizando o desenvolvimento baseado em componentes e IEC-61131-3: Aplicação em indústria de petróleo e gás", apresentada no dia 26 de Agosto de 2017, como requisito parcial para a obtenção do título de **Mestre em Modelagem Computacional e Tecnologia Industrial**.

Orientador:

Prof°. Dr. Roberto Luiz de Souza Monteiro
SENAI CIMATEC

Membro interno da Banca:

Prof°. Dr. Marcelo Albano Moret Simões Gonçalves
SENAI CIMATEC

Membro interno da Banca:

Prof°. Dr. Herman Augusto Lepikson
SENAI CIMATEC

Membro externo da Banca:

Prof°. Dr. José Roberto de Araujo Fontoura
Universidade do Estado da Bahia

Dedico esta dissertação para aqueles brasileiros que sonham com uma nação mais justa e com uma educação de qualidade disponível para todos.

Agradecimentos

Agradeço em primeiro lugar à Deus, Rei dos reis e Senhor dos senhores, pela sua maravilhosa graça, me concedendo: A salvação, por intermédio do seu filho Jesus Cristo, e a vida.
Agradeço a minha família, em especial aos meus pais e esposa, por todo o apoio fornecido.
Ao meu orientador Prof. Dr. Roberto Monteiro por toda dedicação e respeito durante as etapas deste processo, compreendendo sempre a minha dificuldade em conciliar os horários das orientações, estando sempre disposto a colaborar e transmitir conhecimentos.
Aos demais professores do programa de pós-graduação em modelagem computacional, pela boa vontade em ensinar e pelo compromisso com a pesquisa e produção cientifica dos participantes deste mestrado.
Aos membros da banca pela disposição em colaborar também com este trabalho.
À instituição de ensino SENAI CIMATEC, a quem devo grande respeito e admiração, principalmente por ter sido um ex-aluno do sistema de integração SESI-SENAI, onde descobri o prazer pelo desenvolvimento tecnológico e tive a oportunidade de me transformar em um profissional.
Aos colegas de classe por terem compartilhado tão boas ideias e experiências, e por terem acreditado em nossos trabalhos e pesquisas.

Salvador, Brasil — Nuno César Silva Mattos
26de Agosto de 2017

Resumo

Este projeto aborda a aplicação da engenharia de *software* em soluções para automação industrial, mais precisamente a utilização do conceito de desenvolvimento baseado em componentes e do padrão internacional IEC 61131-3, como ferramentas para uma modelagem computacional no contexto dos CLPs (controladores lógicos programáveis) aplicáveis na indústria de exploração e produção de petróleo e gás.
É apresentado, como resultado, uma modelagem de solução corporativa para equipamentos industriais, construída a partir dos fluxogramas de engenharia P&ID (Piping and Instrumentation Diagram - Diagrama de tubulação e instrumentação); tais como transmissores analógicos, válvulas de bloqueio, bombas com acionamento elétrico e malhas de controle PID (Controlador Proporcional, Integral e Derivativo). Para representação desses modelos foi utilizado a diagramação SysML (Systems Modeling Language - Linguagem de modelagem de sistemas), elicitados requisitos comportamentais dos componentes e codificados em linguagem compatível com os CLPs da plataforma RSLogix5000. Por fim, é possível considerar que estes componentes de softwares são reutilizáveis em toda a instalação industrial, sendo assim, estes se apresentam como uma alternativa à padronização de lógicas destinadas aos controladores programáveis.
No quesito metodologia, parte da pesquisa é considerada uma revisão bibliográfica e está relacionada a busca de informações em livros e artigos publicados, objetivando o domínio do conhecimento técnico para analisar o problema e modelar uma solução computacional. O método de pesquisa é experimental e modelagem, estando relacionada ao objeto de estudo: Componentes para controle industrial.
Palavras-chave: Componentes de software, Controladores lógicos programáveis, Engenharia de Software, IEC 61131-3.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Siglas

CLP	Controlador lógico programável
IEC	*International Electrotechnical Commission* (Comissão Eletrotécnica Internacional)
IHM	Interface Homem Máquina
CPU	Unidade Central de Processamento
ROM	*Read-Only Memory* (Memória apenas de leitura)
EPROM .	*Erasable Programmable Read-Only Memory* (Memória somente de leitura, porém programável e apagável)
EEPROM	*Electrically-Erasable Programmable Read-Only Memory* (Memória somente de leitura, porém programável e apagável eletronicamente)
RAM	*Random Access Memory* (Memória de acesso aleatório)
CMOS ...	*Complementary Metal Oxide Semiconductor* (Semi-condutor de óxido de metal complementar)
E/S	*Módulos de entradas e saídas*
FB	*Function Block* (Bloco de função)
OTAN ...	*Organização do Tratado do Atlântico Norte*
IEEE	*Instituto de Engenheiros Elétricos e Eletrônicos*
ESW	*Engenharia de Software*
DBC	*Desenvolvimento Baseado em Componentes*
ESBC	*Engenharia de Software Baseada em Componentes*
OMG	*Grupo de Gerenciamento de Objetos*
COTS ...	*Componentes comercialmente disponível fora da prateleira*
DB	*Data Base* (Base de dados)
CTUD ...	*Counter up/ down* (Contador de subida e descida)
POU	*Unidade de organização de programa*
LD	*Ladder*
FBD	*Function Block Diagram* (Diagrama de bloco de funções)
ST	*Texto Estruturado*
IL	*Lista de instruções*
SFC	*Sequencial Function Chart*
CFC	*Funções Gráficas Contínuas*
POO	*Programação Orientada a Objeto*
UDT	*User Defined Type* (Tipo definido pelo usuário)
IHM	*Interface Homem-Máquina*
SW	*Abreviação de Software*
TIA	*Totally Integrated Automation* (Automação totalmente integrada)
MDE	*Model-Driven Engineering* (Engenharia orientada por modelos)

CORBA . *Common Object Request Broker Architecture* (Arquitetura intermediaria para requisições de objetos comuns)
DCOM ... *Distributed Component Object Model* (Modelo de componente-objeto distribuído)
OMF .... *Object Management Facility* (Facilitador de gerenciamento de objetos)
CBA ..... *Component based automation* (Automação baseada em componentes)
P&ID .... *Piping and Instrumentation Diagram* (Diagrama de instrumentação e tubulação)
SCADA .. *Sistemas de Supervisão e Aquisição de Dados*
BDD ..... *Diagrama para Definição de Bloco*
A/D ..... *Conversor Analógico-Digital*
IBD ..... *Diagrama de Bloco Interno*
ACT ..... *Diagrama de atividades*
SysML ... *Systems Modeling Language* (Linguagem de modelo de sistemas)
SED ..... *Sistemas a Evento Discreto*
KA ..... *Knowlegde Areas* (Áreas de conhecimento)
SWEBOK *Software Engineering Body of Knowledge* (Estrutura de conhecimentos da engenharia de software)
INCOSE . *Conselho Internacional de Engenharia de Sistema*
OMG .... *Object Management Group* (Grupo de gerenciamento de objetos)
MES ..... *Manufacturing Execution System* (Sistema de execução de manufatura)
ERP ..... *Enterprise Resource Planning* (Planejamento de recursos empresariais)
IDE ..... *Integrated Development Environment* (Ambiente de desenvolvimento integrado)
CASE .... *Computer-Aided Software Engineering* (Engenharia de software auxiliada por comput

Capítulo Um

Introdução

1.1 Definição do problema

Em ambientes industriais nem sempre é possível mensurar a qualidade dos softwares implementados nos equipamentos eletrônicos baseados na computação; como os CLPs, utilizados para controle do sistema e processo fabril. Em muitos casos, a falta desta modelagem e padronização nos sistemas de controle comprometem não somente a manutenção e diagnósticos de falhas, como também a expansão e modificações dos processos produtivos.
As perdas relacionadas a esta questão extrapolam quesitos meramente econômicos, provocados por atrasos na retomada de operação da planta ou na entrega dos novos sistemas e unidades; podendo envolver questões estratégicas de uma organização, como a segurança industrial, a gestão do conhecimento e da tecnologia de fabricação.

Apesar de existirem diversas metodologias para validar uma lógica que entrará em funcionamento, um projeto com qualidade duvidosa aplicado nestes dispositivos - partindo do pressuposto que estão mais suscetíveis a erros de programação quando elaborados sem uma padronização - impactará diretamente na segurança dessas instalações industriais, podendo causar danos ao patrimônio, pessoas e até ao meio ambiente. Os denominados *bugs* ou erros numa lógica para controle industrial embarcada em um CLP poderá anular intertravamentos de segurança, causar descontrole operacional e até acidentes fatais.

Neste contexto, é possível afirmar que a falta de um padrão de projeto corporativo aplicado à automação, descrevendo os critérios a serem adotados nos sistemas, é considerado um problema nas instalações industriais, pois além de prejudicar a administração da mesma com uma integração insuficiente, contendo pontos com informações imprecisas ou desconhecidas; provoca perdas na produção, compromete a segurança e gestão da tecnologia de processo em uma organização.

1.2 Objetivo

Este trabalho tem como objetivo geral demonstrar como a engenharia de software baseada em componente poderá colaborar com a utilização de um padrão em lógicas para CLPs.

Para isto, pretende-se apresentar uma modelagem utilizando conceitos definidos pela IEC-61131-3 e IEC-61499, no que se refere ao uso dos *function blocks* (blocos de função) como unidade elementar para um sistema desenvolvido baseado em componente. A SYSML (uma linguagem de modelagem com propósito geral para aplicação em engenharia de sistemas) será utilizada como ferramenta para auxiliar na compreensão do funcionamento destes componentes e posterior programação dos controladores lógicos programáveis.
Este trabalho tem como objetivo específico apresentar um padrão de projeto que atenda ao desenvolvimento dos algoritmos de controle aplicados na indústria de petróleo e gás, implementando a representação de componentes amplamente encontrados nestas plantas industriais.
É considerada como solução desta pesquisa identificar quais as vantagens são obtidas na adoção dos modelos propostos, na etapa de desenvolvimento de lógicas para CLPs.

1.3 Importância da pesquisa

Desde o inicio da criação dos primeiros CLPs, ao final da década de 60 do século passado até o inicio da sua 4ª geração em meados da década de 90, a atividade de programação e configuração de lógicas para tais equipamentos foram elaboradas por técnicos e engenheiros especializados e conhecedores do hardware dos dispositivos. Eram utilizadas linguagens e plataformas de configuração específicas dos seus fornecedores.
Além desta dependência de recursos técnicos, esta atividade era executada de maneira artesanal, mesmo considerando técnicas ainda hoje bem difundidas no meio acadêmico e na indústria, tais como Rede de Petri, máquinas de estado e autômatos finitos. Com o advento da norma internacional IEC 61131-3, os fabricantes deste equipamento passaram a adotar requisitos específicos, como por exemplo a utilização de 5 diferentes tipos de linguagens de programação. Esta norma possibilitou a diminuição das limitações por dependência dos recursos técnicos, entretanto não conseguiu evitar que os desenvolvedores e integradores de sistemas continuassem a atuar de maneira artesanal.
Em 2004, a VDMA (Associação Alemã de Fabricação de Máquinas e Instalações Industriais) apresentou um relatório indicando a crescente proporção do desenvolvimento de *softwares* nos custos das máquinas (STETTER, 2004), a proporção de *software* dobrou em uma década de 20% a 40%. Isto sinaliza a importância do tratamento desta questão, se esta tendência continuar, a principal atividade de fornecedores e desenvolvedores de sistemas de automação será a produção deste artefato.

Atualmente é possível encontrar pesquisas abordando o uso da ESW (Engenharia de *Software*) no âmbito da automação industrial, no capitulo 4 são apresentadas algumas publicações científicas que abordam o tema do reúso, da ESW e do avanço de padrões internacionais neste ambiente. Foram encontrados trabalhos utilizando conceitos oriundos

da UML, inclusive com a introdução da UML-PA e SYSML, no tratamento de derivações destinadas à automação de processos e sistemas. Outras concepções, tais como a CBA (Component Based Automation - Automação Baseada em Componentes) vinculada a norma IEC 61499, a utilização de padrão de projeto (*Design Pattern*) e a engenharia dirigida por modelos (MDE); têm colaborado cada vez mais para a evolução da modelagem dos softwares nos CLPs.

Um modelo de *software* consistente é útil para inúmeras categorias industriais, em diferentes processos produtivos. Além disto, contribui, de forma valiosa, com as atividades exercidas por empresas fornecedoras de tecnologias da automação. A maneira como os requisitos dos componentes foram elicitados nesta pesquisa, permite considerar que a mesma tem um grau de importância e aplicação maior em indústrias de exploração e produção de petróleo e gás; especialmente nas estações coletoras, injeção de água e compressão. Foram considerados atributos específicos para esta finalidade, mas devido a similaridade dos equipamentos e instalações, a abordagem proposta poderá ser adotada também em estações de tratamento e distribuição de água e, em alguns casos, indústrias petroquímicas. Não foi considerada a utilização dos componentes em processos de fabricação estritamente manufaturados.
A indústria de petróleo e gás foi escolhida com um ambiente para análise dos componentes propostos, por conter uma extensa gama de componentes e possuir - como característica da natureza de seu negócio - a reutilização e portabilidade de equipamentos em diferentes condições. Por exemplo, é muito comum que uma indústria de grande porte deste segmento descomissione suas instalações ou parte delas e reinstalá-las em outros locais. Automaticamente estes dois requisitos se tornam fundamentais na elaboração deste projeto.

1.4 Motivação

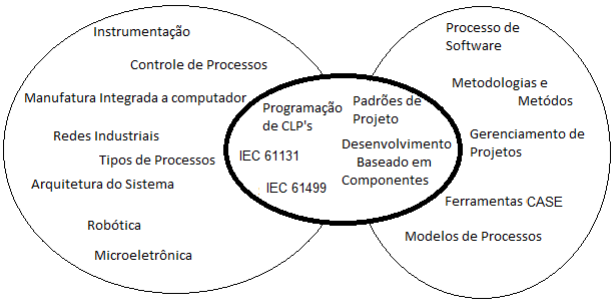
O conceito de componente de software individual para descrever e encapsular o controle de um equipamento na indústria já foi apresentado e discutido por alguns autores. A exemplo de trabalhos já publicados, Lee, Harrison & West (2004), Crnkovic & Larsson (2000), Ljungkrantz *et al.* (2010) discutem os diferentes níveis de reutilização dos componentes em sistemas distribuídos e aspectos do desenvolvimento, tais como a generalidade e a eficiência, problemas de compatibilidade, demandas no ambiente de desenvolvimento, manutenção e outros.
Fora do ambiente manufaturado e puramente SED (sistemas a eventos discretos), já observados por outras pesquisas, este trabalho tem como principal motivação explorar a utilização de componentes individuais, com interfaces bem definidas, de modo que favoreça a diminuição do tempo necessário para o desenvolvimento e minimize erros na

programação.
Estes componentes precisam ter a capacidade para participar do sistema automatizado, preferencialmente sem a necessidade de um controlador mestre. Por fim, a área pesquisada escolhida é uma indústria com a produção mais próxima da continuada, especificamente estações de coleta e tratamento de petróleo e gás, por ter sido uma área com maior espaço para exploração do conteúdo encontrado em pesquisas similares.

1.5 Limites e limitações

Este trabalho abordará conteúdos relacionados a automação industrial e engenharia de *software*. Observa-se na figura 1.1, na área sombreada, o enfoque deste trabalho com seu escopo principal, englobando a programação de CLPs, utilizando-se do conceito de desenvolvimento baseado em componentes e recursos providos pela IEC 61131 e IEC 61499. Quanto aos demais aspectos da engenharia de *software*(Descritos ao lado direito da área sombreada) e automação industrial (descritos ao lado esquerdo da área sombreada), não serão abordados por não pertencerem ao escopo deste trabalho.

Figura 1.1: Enfoque deste trabalho



Fonte: do autor

Esta pesquisa limita-se a modelagem e simulação de cinco componentes amplamente utilizados na indústria de petróleo e gás e outras com produção de fluxo em linha. São estes:

- Componente transmissor analógico;
- Componente válvula de bloqueio manual;
- Componente válvula pneumática de bloqueio;
- Componente bomba com inversor de frequência;
- Componente malha de controle PID.

A linguagem adotada para modelagem deste sistema foi a SysML. Foram considerados conceitos como o da automação baseada em componentes, dos princípios da reusabilidade e o uso dos blocos de funções - propostos pela IEC 61131-1 e revisados pela IEC 61499. No ambiente de implementação/ simulação, a ferramenta utilizada foi o RSLogix5000 fornecido pela Rockwell Software.

1.6 Aspectos metodológicos

Inicialmente foi necessário buscar bases teóricas com conceitos importantes no processo de desenvolvimento do projeto, realizado um levantamento bibliográfico, este consiste em uma seleção qualitativa de livros, documentos, arquivos fotográficos que discutam o tema investigado pelo pesquisador em seu trabalho acadêmico. Trata-se de um prévia seleção de bibliografias ou documentos que poderão ser utilizados como referências ou "mapas" na construção do trabalho acadêmico.
De posse deste material, os capítulos 2 ao 4 apresentam os conceitos fundamentais que nortearam este trabalho. Por esta razão, parte da pesquisa pode ser classificada, segundo o que preconizam Cervo, Silva & Bervian (2007), como sendo uma revisão de literatura com cunho histórico e documental; lastreada em livros, artigos, dissertações e teses sobre as seguintes temáticas:

- **Apresentação de tópicos da engenharia de software, relacionados com os temas:*design*(Projeto) e reúso:** Os princípios preconizados pela engenharia de software baseada em componentes (tais como: Modularização, encapsulamento, separação de interfaces/interesses, escalabilidade e possibilidade de reúso) favoreceram a aceitação desta forma de desenvolvimento como a principal abordagem adotada nesta dissertação;
- **Apresentação dos avanços promovidos pelos padrões internacionais IEC 61131-3 e IEC 61499:** Os principais pontos destas duas normas que foram aproveitados por este trabalho, diz respeito ao uso dos blocos de funções (*function blocks*), tipos de dados estruturados pelo usuário (*user-defined data type*) e a distribuição de dados e eventos de entrada/saída;
- **Apresentação de pesquisas publicadas com finalidade semelhantes a deste trabalho:** Foi relacionado o tema engenharia de software com projetos de automação industrial;

Por fim, a modelagem, e de posse desta, os componentes são implementados e analisados em um ambiente de simulação. Neste aspecto, o método de pesquisa é considerado

experimental e modelagem/ simulação, pois se detém a um objeto de estudo, a automação baseada em componentes. Como etapas para o processo de modelagem, foram consideradas:

1. Análise dos requisitos (realizado o levantamento e extraído informações a partir dos fluxogramas de engenharia do processo - PI&D):

   Identificação dos componentes;

   Especificação dos requisitos (em situações normais/ condições de falha);

2. Interação dos componentes e interfaces, apresentada através da linguagem de modelagem e diagramas SYSML;

3. Descrição das funcionalidades e estrutura de dados, adotando os critérios abordados pela IEC 61131-3 e IEC 61499;

No processo de simulação foram analisados questões-chave qualitativas, como: abstração, acoplamento, decomposição e modularização, encapsulamento, e suficiência. Foram também avaliados os seus benefícios, através dos fatores de reusabilidade, flexibilidade e portabilidade.

1.7 Organização do Projeto de Dissertação

Este documento apresenta 6 capítulos e está estruturado da seguinte forma:

- **Capítulo 1 – Introdução**: Contextualiza o âmbito no qual a pesquisa proposta está inserida. Apresenta, portanto, a definição do problema, objetivos e justificativas da pesquisa e como este trabalho está estruturado;
- **Capítulo 2 – A Engenharia de *software***: Trata-se de uma breve revisão bibliográfica, onde são abordados apenas os principais conceitos utilizados nesta pesquisa, como por exemplo o de desenvolvimento baseado em componentes e outros relacionados ao projeto de *software*;
- **Capítulo 3 – IEC61131-3: Funtion Block e a IEC61499**: Este capitulo apresenta padrões internacionais aplicados ao desenvolvimento de programas para os CLPs, destacando a sua importância e os principais conceitos destas normas para o favorecimento da modelagem proposta nesta pesquisa;
- **Capítulo 4 – Estado da arte da engenharia e modelagem de software aplicada à automação industrial**: São apresentados resumos de artigos publicados, principalmente em revistas patrocinadas pela IEEE, correlacionados ao tema

proposto neste trabalho, por fim são apresentados superficialmente os conceitos de alguns diagramas da tecnologia SysML (uma linguagem de modelagem de propósito geral para aplicação em engenharia de sistemas) que serão propostos no capitulo 5

- **Capítulo 5 – Modelagem do sistema:** tem-se a modelagem computacional, assim como a descrição das suas interfaces e funcionamento. Por fim, são apresentadas as análises dos modelos em ambientes de simulação;
- **Capítulo 6 – Considerações Finais**: Apresenta as conclusões, contribuições e algumas sugestões de atividades de pesquisa a serem desenvolvidas no futuro.

Capítulo Dois

A Engenharia de software

Em virtude da dinâmica de sua evolução, o elemento "software"necessita de documentações padronizadas e técnicas de criação, manutenção e gerenciamento que incorporem as melhores praticas e metodologias. Em seu livro engenharia de software 6a Edição, Pressman (2006) no capítulo1 página 13, diz: "O software tornou-se o elemento chave na evolução de sistemas e produtos baseados em computador, e uma das tecnologias mais importantes em todo o mundo. Ao longo dos últimos 50 anos, o software evoluiu de um ferramental especializado em solução de problemas e análise de informações para um produto de indústria. Mas ainda temos problemas na construção de software de alta qualidade no prazo e dentro do orçamento. O software - programas, dados e documentos - dirigido a uma ampla variedade de tecnologias e aplicações, continua a obedecer a uma série de leis que permanecem as mesmas há cerca de 30 anos. O intuito da engenharia de software é fornecer uma estrutura para a construção de software com alta qualidade."

O termo "Engenharia de Software"(ESW) foi utilizado no título de uma conferência da Organização do Tratado do Atlântico Norte (OTAN), realizada na Alemanha em 1968. A partir desta data, o Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) publica pela primeira vez seu relatório sobre Engenharia de Software. E em 1976 foi fundada uma comissão na IEEE *Computer Society* com a incumbência de desenvolver padrões da ESW. Após o desenvolvimento de uma serie destes padrões, em 1993 a comissão estabelece um guia para o desenvolvimento do corpo de conhecimento e práticas recomendadas da engenharia de software, também conhecido como SWEBOK.

2.1 O SWEBOK

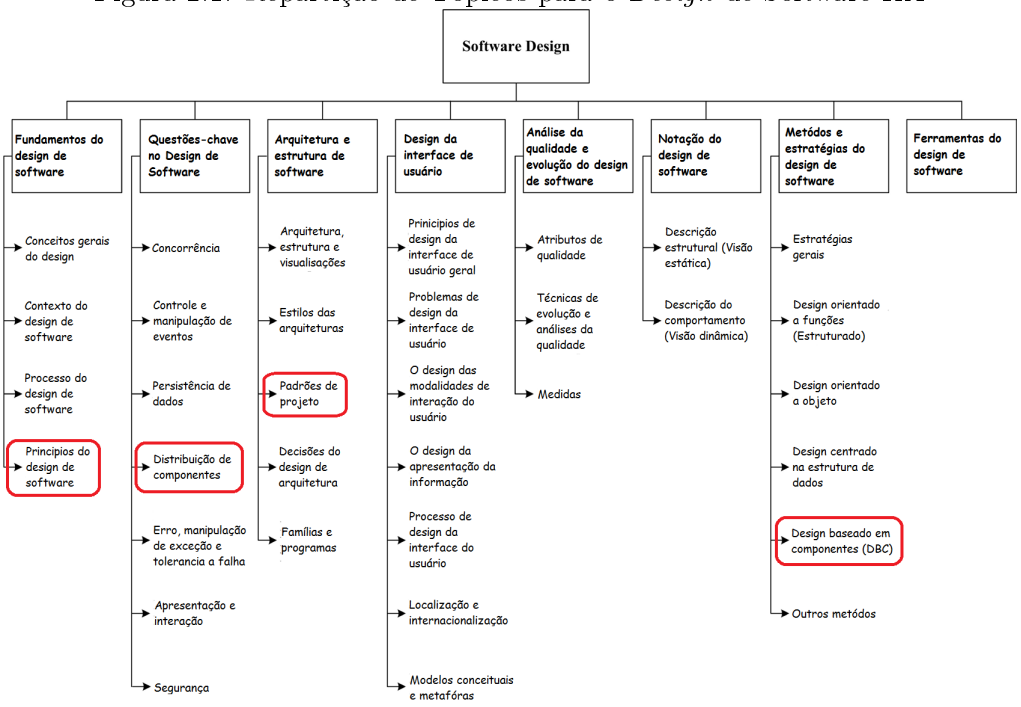
O guia SWEBOK estabelece, dentre outras coisas, dez áreas de conhecimento (KA - *Knowledge Area*) para tratar o tema:

- Requisitos de Software
- *Design* ou projeto de Software
- Construção de Software
- Teste de Software

- Manutenção de Software
- Gerência de Configuração de Software
- Gerência da Engenharia de Software
- Processo de Engenharia de Software
- Ferramentas e Métodos da Engenharia de Software
- Qualidade de Software

Pelo fato do SWEBOK possuir uma abordagem com espectro abrangente, este referencial teórico apresentará apenas algumas questões relacionadas ao reúso e projeto ou modelagem de software, por ser a área de conhecimento com maior relevância para este trabalho. Segundo Santiago (2011) esta KA é dividida em seis subáreas, a primeira subárea apresenta os fundamentos de projeto de Software, que formam uma base subjacente à compreensão do papel e do escopo do projeto de *software*. A segunda subárea agrupa o conjunto de Questões-chave em projeto de *Software*. Essas questões chaves incluem colaboração, controle e tratamento de eventos, a distribuição de componentes, tratamento de exceções e erros e tolerância à falha, interação e apresentação, e persistência de dados. A terceira subárea é Estrutura e Arquitetura de Software, os tópicos da qual são estruturas de arquiteturas e pontos de vista, estilos de arquitetura, padrões de projetos, e, finalmente, as famílias dos programas. A quarta subárea descreve Análise de Qualidade do projeto e Avaliação do software. Enquanto existe uma KA inteira dedicada à qualidade de software, esta subárea apresenta os tópicos especificamente relacionados ao projeto de software. Esses aspectos são atributos de qualidade, análise de qualidade, e técnicas de avaliação e medidas.
A quinta subárea é Notações de projeto de Software, que é dividida em descrições estruturais e comportamentais. A última subárea descreve Estratégias de projeto de Software e Métodos. Primeiro, as estratégias gerais são descritas, seguidas por métodos de projetos orientados por função, métodos de *design* orientados a objeto, *design* centrando na estrutura de dados, *design* baseado em componentes e outros. A figura 2.1 apresenta como estão distribuídos os tópicos sobre a KA *design* de software.

Figura 2.1: Repartição de Tópicos para o *Design* de Software KA



Fonte: Traduzido de Bourque & Fairley (2013)

2.1.1 *Princípios e fundamentos do design de software*

Para o guia SWEBOK (BOURQUE; FAIRLEY, 2013), princípios de *design* de software são noções-chave que fornecem a base para muitas abordagens diferentes e conceitos. Estes incluem abstração; acoplamento e coesão; decomposição e modularização; Encapsulamento ou ocultamento de informações; Separação da interface e implementação; suficiência, integridade e primitividade; e separação de interesses.

- **Abstração** é "uma visão de um objeto que se concentra na informação relevante para um propósito particular e ignora o restante da informação" [Allen *et al.* (2008)]. No contexto do projeto de *software*, dois mecanismos principais de abstração são a parametrização e a especificação. Abstração por parametrização são resumos a partir dos detalhes das representações de dados, representando os dados como parâmetros nomeados. A abstração por especificação leva a três tipos principais de abstração: abstração processual, de dados e de controle (iteração).

- **Acoplamento e Coesão** o primeiro é definido como "uma medida da interdependência entre módulos em um programa de computador", enquanto a coesão é definida como "uma medida da força de associação dos elementos dentro de um módulo"[Allen *et al.* (2008)].

- A **Decomposição e modularização** significa que o software grande pode ser dividido em um número de componentes nomeados menores que têm interfaces bem definidas que descrevem interações de componentes. Geralmente o objetivo é colocar diferentes funcionalidades e responsabilidades em diferentes componentes.
- **Encapsulamento** de informações significa agrupar e empacotar os detalhes internos de uma abstração e torná-los inacessíveis a entidades externas.
- **Separação de interface e implementação** envolve a definição de um componente, especificando uma interface pública (conhecida pelos clientes) que é separada dos detalhes de como o componente é realizado.
- **Suficiência, integridade e primitividade**. Conseguir suficiência e integridade significa garantir que um componente de software capte todas as características importantes de uma abstração e nada mais. Já a primitividade significa que o *design* deve ser baseado em padrões fáceis de implementar.
- **Separação de interesses**. Um *concern* é uma "área de interesse com relação a um projeto de software" [Clements, Bass & Nord (2010)]. Esta é uma área do *design* relevante para uma ou mais partes interessadas, cada visão de arquitetura enquadra uma ou mais *concern*, separa-los por visões permite que os *stakeholders* se concentrem em poucas coisas ao mesmo tempo e oferece um meio de gerenciar a complexidade.

2.1.2 Estrutura e Arquitetura de Software: Padrões de projeto

Em um simples resumo, os padrões de projetos são formas de descrever as melhores práticas e capturar as boas experiências de maneira que se torne possível à outros reusa-las. As ideias inicias para o uso de padrões na construção começaram a ser apresentados por Alexander, Silverstein & Ishikawa (1977), que sugeriu haver pontos comuns de projeto em prédios que eram inerentemente agradáveis e eficazes, o próprio Christopher Alexander afirma: "Cada padrão descreve um problema no nosso ambiente e o cerne de sua solução, de tal forma que você possa usar essa solução mais de um milhão de vezes, sem nunca fazê-lo da mesma maneira". Um padrão é uma descrição do problema e da essência de sua solução, de modo que a solução possa ser reusada em diferentes contextos, não é necessariamente uma especificação detalhada, em vez disso, pode-se pensar nele como uma descrição de conhecimento e experiências em uma solução já aprovada para um problema comum.

Em seu livro "Padrões de projeto: Soluções reutilizáveis de software orientado a objeto" Gamma *et al.* (2000) inicia explicando a diferença entre os projetistas experientes e os

novatos, que se distanciam através da capacidade individual do primeiro em capturar experiências ou soluções que funcionaram no passado podendo ser reutilizadas rapidamente, de maneira que novos profissionais precisariam resolver cada problema a partir de princípios elementares ou do zero. Esta análise se repete em outras profissões, por isso os padrões de projeto são elaborados vislumbrando aproveitar os valores que as experiências promovem, sobre este aspecto o padrão de projeto pode ser visto também como uma forma de gestão do conhecimento.

Segundo Dantas *et al.* (2013) Podemos pensar em um padrão como a reutilização da essência de uma solução para determinados problemas similares. Sintetizando as definições encontradas na literatura, podemos dizer que um padrão resolve um problema recorrente, em um determinado contexto, fornecendo uma solução que comprovadamente funcione, além de informar os resultados e compromissos da sua aplicação, e subsídios para que seja possível adaptar esta solução a uma variante do problema.

Os padrões tiveram um enorme impacto no projeto de software orientado a objetos. Além de serem soluções já testadas para problemas comuns, tornaram-se um vocabulário para falar sobre um projeto. É possível, portanto, explicar o projeto por meio de descrições dos padrões que foram utilizadas. Isso é particularmente verdadeiro para os padrões de projeto mais conhecidos que foram originalmente descritos por Gamma *et al.* (2000) como 'Gangue dos Quatro' em seu livro. Outras descrições de padrões particularmente importantes são as publicadas em uma série de livros por autores da Siemens, uma grande empresa europeia de tecnologia.
Como mencionado no paragrafo anterior, os quatro elementos essenciais dos padrões de projeto foram definidos pela 'Gangue dos Quatro', por Gamma *et al.* (2000):

- Um **nome** que seja uma referência significativa para o padrão. Dar um nome aumenta imediatamente o vocabulário do projeto e permite utilizar um nível mais alto de abstração. Ter uma identificação permite documentar, comunicar e também torna-se mais fácil pensar e discutir sobre ele.
- Uma descrição da área que delimita o **problema** e que explique quando o modelo pode ser aplicado. Algumas vezes, o problema incluirá uma lista de condições que devem ser satifeitas para que faça sentido aplicar o padrão.
- A descrição da **solução** das partes da solução de projeto, seus relacionamentos, responsabilidades e colaborações. Essa não é uma descrição do projeto concreto; é um modelo para uma solução de projeto que pode ser instanciado de diferentes maneiras. Costuma ser expresso graficamente e mostra os relacionamentos entre os objetos e suas classes na solução.

- Uma declaração das **consequências** — os resultados e compromissos — da aplicação do padrão, ou seja as vantagens e desvantagens do uso, assim como os custos e benefícios. Esta declaração pode ajudar os projetistas a entenderem quando um padrão pode ou não ser usado em uma situação particular. As consequências para o uso de um padrão de projeto frequentemente envolvem o balanceamento entre espaço e tempo, elas também podem abordar aspectos sobre linguagens e implementação. Uma vez que a reutilização é frequentemente um fator no projeto orientado a objetos, as consequências de um padrão incluem o seu impacto sobre a flexibilidade, a extensibilidade ou portabilidade de um sistema. Relacionar essas consequências explicitamente ajuda a compreendê-las e avaliá-las

Segundo Sommerville (2011) os padrões suportam reúso de conceito em alto nível. Ao tentar reusar componentes executáveis existirá uma limitação inevitável nas decisões, feitas pelos implementadores desses componentes. Elas vão desde os algoritmos particulares que têm sido usados para implementar os componentes até objetos e tipos de interface dos componentes. Quando essas decisões de projeto entram em conflito com seus requisitos específicos, o reúso de componentes é impossível ou apresenta ineficiências em seu sistema. O uso de padrões significa reúso de ideias, mas em alguns casos pode se adaptar a aplicação para se adequar ao sistema que está sendo desenvolvido.
Ao começar a projetar um sistema, pode ser difícil saber, antecipadamente, se vai precisar de determinado padrão. Portanto, muitas vezes, o uso destes em um processo envolve o desenvolvimento de um projeto, a experimentação com algum problema e, em seguida, o reconhecimento de que um padrão pode ser usado. Isso certamente é possível ao se concentrar nos 23 padrões de uso geral documentados no livro original de padrões Gamma *et al.* (2000). No entanto, se o problema for diferente, poderão existir dificuldades para encontrar um padrão de projeto adequado entre as centenas diferentes que têm sido propostos.
Os padrões são uma ótima ideia, mas, para usá-los efetivamente, é necessário experiências de desenvolvimento e reconhecer as situações em que um padrão pode ser aplicado. Para Sommerville (2011) programadores inexperientes sempre acharão difícil decidir se podem reusar um padrão ou se é necessário desenvolver uma solução especial.

2.1.3 Estratégias e métodos de projetos de software: desenvolvimento baseado em componente (DBC)

- **O que são componentes**
  Discussões na academia e na indústria indicam que há uma grande lacuna relacionada às terminologias, aplicações, usos, efeitos colaterais e outras características pertinentes a Engenharia de *Software* Baseada em Componentes (ESBC), inclusive

a própria definição de "componentes", também denominados artefatos reutilizáveis. Será possível identificar nesta seção a visão de diferentes autores, fonte: Resende & Cunha (2006).

Segundo Szyperski (1997), construir novas soluções a partir de combinações de componentes já desenvolvidos e adquiridos, aumenta a qualidade e dá suporte ao rápido desenvolvimento, levando a diminuição do tempo de entrega do produto final ao mercado. Os sistemas definidos por meio da composição de componentes permitem que sejam adicionadas, removidas ou substituídas partes dele sem a necessidade de sua completa substituição. Com isso, o DBC auxilia na manutenção dos sistemas de *software* por permitir que sejam atualizados por meio da integração de novos componentes ou atualização dos já existentes.

Para Brown & Wallnau (1996), um componente é uma unidade funcional significativa de um sistema, podendo representar desde uma função de alto nível (Tarefas elaboradas por linguagens de programação próxima à humana) até uma tarefa de baixo nível (Tarefas elaboradas por linguagens de programação próxima ao hardware). Pode-se classificar um componente como atômico ou composto. Portanto, a arquitetura de um sistema completo apresenta-se como uma organização hierárquica de componentes agregados ou únicos. Além disso, define-se também como unidades reutilizáveis e compartilhadas, podendo ser utilizadas em diversos projetos.

Segundo Kruchten (1998), um componente representa um elemento significativo e independente, uma parte substituível que cumpre uma clara função no contexto de uma arquitetura bem definida.

Para Szyperski (1997), o que identifica um componente não é sua aplicação nem uma tecnologia de implementação específica. Assim, qualquer dispositivo de *software* pode ser considerado um componente, desde que possua uma interface bem definida.

Segundo Pressman (2006), componente é um bloco construtivo modular para *software* de computador. Mais formalmente, a especificação da linguagem de modelagem unificada pelo Grupo de Gerenciamento de Objetos (OMG - Unified Modeling Language Specification, Group (2003)) define componente como "... Uma parte modular, possível de ser implantada e substituível de um sistema que encapsula implementação e expõe um conjunto de interfaces". No contexto da engenharia de *software* tradicional, componente é o elemento funcional de um programa que incorpora a lógica de processamento, as estruturas de dados internas necessárias para implementar a logica de processamento, e uma interface que habilita os dados a serem passados. Um componente tradicional também denominado *módulo*, reside na arquitetura de *software* e se presta a um dos três importantes papeis: (1) um "componente de controle" que coordena a chamada de todos os demais componentes do domínio do problema, (2) um "componente de domínio do problema" que implementa uma função completa ou parcial solicitada pelo cliente ou (3) um "componente de infraestrutura" responsável por funções que dão suporte ao processamento necessário ao

domínio do problema.

Para Sametinger (1997), "componentes de *software* reutilizáveis são artefatos autocontidos, facilmente identificáveis que descrevem e/ou executam funções específicas e possui interfaces claras, documentação apropriada e uma condição de reúso definida", O termo artefato apresentado nesta definição indica que componentes podem ter diferentes formas, como por exemplo, um código fonte, uma documentação ou um código executável. A seguir, destacam-se alguns elementos desta definição, discutidos em Sametinger (1997), que contribuem para um melhor entendimento da abordagem dada pelo autor, retirado de Redolfi *et al.* (2004):

- **Autocontido:** diz respeito à capacidade de um componente ser reutilizável por si só, ou seja, não depende de outros para ser reusado. As dependências entre eles, se existirem, devem ser vistas como um único componente reutilizável.
- **Identificação:** componentes devem ser claramente identificáveis, ou seja, devem estar contidos em um único local ao invés de distribuídos e misturados com outros artefatos de *software* ou documentação.
- **Funcionalidades:** diz respeito às funcionalidades de um componente, ou seja, componentes podem descrever um funcionamento ou executar funções. Assim, pode-se considerar toda a documentação do ciclo de vida do *software* como um componente, embora ela não abrigue uma codificação.
- **Interfaces:** componentes devem ter interfaces claras a fim de facilitar o reúso e a conexão com outros, além de esconder detalhes que não são necessários para o reúso.
- **Documentação:** uma documentação clara é indispensável para o reúso, pois é ela quem indica o tipo e sua complexidade. A falta de uma documentação apropriada torna o componente menos útil para o reúso.
- **Condição de reúso:** condições de reúso devem ser estabelecidas, contendo informações sobre quem é o proprietário do componente, quem o mantém, quem deve ser contactado em caso de problemas, e qual é o estado de qualidade do componente.

Como já foi mencionado, a ideia de desenvolvimento baseado em componentes precisa ser expandida a outros artefatos do *software*, não limitando o seu uso à programação orientada a objeto. Mesmo sendo as classes, consideradas pela programação orientada a objeto como um componente, não deve ser associado a única ideia de que todos os componentes são objetos ou ainda que o conceito de ambos são iguais. Vale salientar ainda que, para este trabalho, considerou-se que os componentes possuem as características básicas apresentadas por Sametinger (1997), abrangente o suficiente para enfatizar o reúso em todo o processo de desenvolvimento. Além disso, foi considerado neste trabalho a questão da disponibilidade para aquisição na própria organização ou no mercado, permitindo ou não o acesso ao seu código-fonte.

- **Motivação para o uso de componentes**

  Existem diversos fatores que explicam o interesse no desenvolvimento de sistemas, usando componentes, entre elas é possível observar em Alves (2001):

  – O aumento da competição para prover soluções de *software* com maior qualidade, e desenvolvidas rapidamente;
  – A crescente demanda por *software* extensos e complexos, que usualmente não podem ser desenvolvidos por uma única organização;
  – A diversidade no mercado, tanto de produtos COTS (Commercial off-the-shelf - Componentes comercialmente disponível fora da prateleira) - expressão utilizada para descrever a compra de soluções prontas que são então adaptadas para satisfazer as necessidades do cliente, ao invés da construção de uma solução customizada desde o inicio - de domínio específico, quanto de componentes genéricos;
  – O aumento do grau de interoperabilidade e de concordância de padrões entre os produtos COTS, o que garante uma significativa redução de tempo e esforço gastos durante a fase de integração do produto; e
  – O crescente esforço em pesquisas na área, fornecendo novas técnicas e métodos para suportar DBC.

  Ainda sobre o uso de componentes, Golubev (2004) cita como vantagens esperadas:

  – A flexibilidade, onde os componentes podem trabalhar independentemente, e caso projetados corretamente, tornam-se menos dependentes de seu ambiente de *hardware*, *software*, outras aplicações ou componentes. Consequentemente, sistemas baseados em componentes apresentam-se menos sensíveis às mudanças e se mantêm num nível funcional mais adaptável e estendível do que nos sistemas tradicionais;
  – A reutilização de funcionalidades, técnicas ou orientadas ao negócio, desenvolvidas e implementadas apenas uma vez, ao contrário do caso tradicional;
  – A sustentabilidade num sistema baseado em componentes, em que uma funcionalidade, idealmente implementada apenas uma vez, resulta numa manutenção mais fácil, levando a custo mais baixo de produção e a vida mais longa para esses sistemas;
  – A maior produtividade percebida a longo prazo. A curto prazo, os frutos da reutilização são menores do que o custo da introdução de uma nova forma de desenvolvimento de sistemas;
  – A distribuição, onde o DBC possibilita o projeto de sistemas para distribuição em rede Internet.

2.1.4 *Os pontos chave no projeto de Software: Distribuição de Componentes*

Uma série de questões-chave devem ser tratadas ao projetar software, algumas estão relacionadas a problemas de qualidade em que todos devem abordar - por exemplo, desempenho, segurança, confiabilidade, usabilidade, etc. Outro problema importante é como decompor, organizar e compactar componentes de software, isso é tão fundamental que todas as abordagens de projeto tratam o assunto de uma forma ou de outra. Em contraste, outras questões lidam com algum aspecto do comportamento do software que não está no domínio da aplicação, mas que aborda alguns dos domínios de suporte. Essas questões, que muitas vezes cruzam a funcionalidade do sistema, segundo Bourque & Fairley (2013) têm sido referidas como aspectos que "tendem a não ser unidades de decomposição funcional do software, mas a serem propriedades que afetam o desempenho ou a semântica dos componentes de maneira sistêmica".

Para Pressman (2006), no contexto de engenharia de *software*, a reutilização é uma ideia ao mesmo tempo antiga e nova. Os programadores têm reutilizado ideias, abstrações de processos desde os primórdios da computação, mas a abordagem para reutilização era *ad hoc*. Hoje em dia, sistemas computacionais complexos de alta qualidade em funcionamento devem ser construídos em prazos muito curtos e exigem uma abordagem mais organizada para reutilização.

Clements, Bass & Nord (2010) descrevem a ESBC (Engenharia de software baseada em componentes) da seguinte maneira: A [EBSC] incorpora a filosofia "Compre, não construa" defendida também por Fred Brooks e outros. Da mesma forma que sub-rotinas iniciais liberavam o programador de pensar em detalhes, a [ESBC] transfere a ênfase de programar *software* para compor sistemas de *software*. A implementação deu lugar à integração como foco.

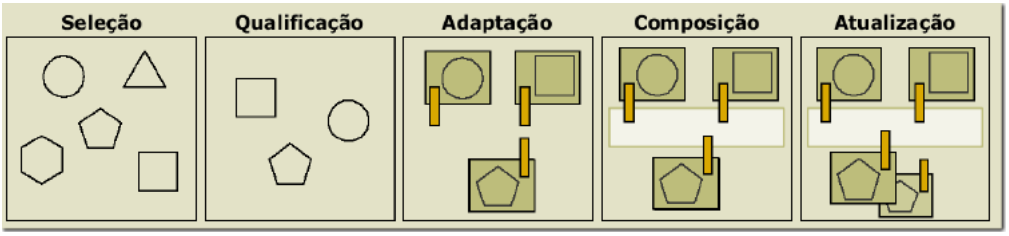
Para Resende & Cunha (2006) O DBC está, senão revolucionando, ao menos mudando significativamente o desenvolvimento e o uso do *software* em geral. Acredita-se que componentes e serviços baseados em componentes terá seu uso tão facilitado, que até mesmo não programadores poderão utilizá-los extensamente para construir suas aplicações, com o auxílio de ferramentas específicas para este fim. Observa-se que a atualização (*update*) automática de componentes via *Internet*, encontra-se presente em muitas aplicações como uma forma-padrão de melhoria da aplicação

Quando trata de DBC, deve-se notar a diferença entre o desenvolvimento de componentes e desenvolvimento com componentes. No primeiro caso, eles são especificados e implementados, existindo a preocupação em gerar a documentação e em projeta-los de forma a serem reusados. No segundo, os componentes já produzidos são utilizados para conceber um novo sistema. Seguindo este conceito, o DBC dispõe de duas perspectivas principais:

a da Engenharia de Componentes e a da Engenharia da Aplicação. A engenharia de componentes tem por objetivo desenvolver estes, de forma reutilizável para a construção dos sistemas. Suas principais responsabilidades consistem em identificar a necessidade da construção dos componentes, desenvolvê-los, catalogá-los e disseminá-los a um domínio específico da aplicação.

A engenharia da aplicação tem como objetivo propiciar a utilização dos componentes de *software*. Suas principais responsabilidades consistem em selecionar, qualificar, adaptar, integrar e atualizar componentes para utilização em novos sistemas. A figura 2.2 representa a metodologia utilizada pela engenharia de aplicação, na distribuição e uso dos componentes.

Figura 2.2: Sequência de passos para a seleção e utilização de componentes



Fonte: Feijó (2007)

2.2 Reúso

Certamente o reúso é um dos principais benefícios para a adoção de estratégias da engenharia de software durante o processo de desenvolvimento de novos sistemas. Os conceitos apresentados até o momento, como padrões de projeto, e o desenvolvimento baseado em componentes (DBC) contribuem neste sentido.

Na década de 1980, a abordagem da orientação a objetos apresentou-se como o mecanismo capaz de promover a reutilização, chegando-se a considerar a classe como a unidade atômica para esse fim. A partir daí, verificou-se que a reutilização não era uma característica inerente apenas da orientação a objetos, mas obtida a partir do uso de técnicas para construção de *software* reutilizável.

Segundo Sommerville (2011) somente em 2000 o "desenvolvimento com reúso" se tornou a norma para novos sistemas de negócios. A mudança para o desenvolvimento baseado em reúso foi uma resposta às exigências de menores custos de produção e manutenção de software, entregas mais rápidas de sistemas e softwares de maior qualidade. Cada vez

mais, empresas consideram o software como um ativo valioso, o reúso tem sido promovido para aumentar o retorno sobre os investimentos em software. Algumas grandes empresas fornecem uma variedade de componentes reusáveis para seus clientes. Padrões, como o *web service*, tornaram mais fácil o desenvolvimento de serviços gerais e reutilização destes em uma variedade de aplicações.

Sobre as principais vantagens para o reúso nas aplicações industriais, destacam-se:

- Velocidade no desenvolvimento;
- Atendimento aos requisitos do sistema;
- Facilidade na ampliação, graças ao conceito de Hot-Swapping, onde devido a desassociação dos componentes no programa de controle, puderam ser inseridos novos em tempo de execução sem a necessidade de parada do processo. Além disso, foi possível evitar a repetição de testes dos subsistemas em que já estavam em funcionamento;
- Facilidade na manutenção dos sistemas e treinamento das equipes responsáveis;
- Facilidade na localização dos defeitos e diagnósticos de falhas, diminuindo assim perdas por tempos de processo parado;
- Redução no período de testes e comissionamentos para sistemas novos;
- Redução de problemas por documentações insuficientes e lógicas pobremente comentadas;

Ainda sobre o reúso de software, Sommerville (2011) menciona que é possível adotá-lo em vários níveis diferentes:

1. **O nível de abstração**. Nesse nível o software não é reutilizado diretamente, mas usa o conhecimento das abstrações de sucesso no seu projeto. Os padrões de projeto e de arquitetura são formas de representar o conhecimento abstrato para reúso.
2. **O nível de objeto**. Nesse nível, os objetos são reutilizados diretamente de uma biblioteca em vez de escrever um código. Para implementar esse tipo de reúso, é preciso encontrar bibliotecas adequadas e descobrir se os objetos e métodos oferecem a funcionalidade necessária. Por exemplo, uma aplicação que precisa processar mensagens de correio em um programa Java, poderá ser usado objetos e métodos de uma biblioteca JavaMail.
3. **O nível de componentes**. Muitas vezes, se faz necessário adaptar e ampliar o componente adicionando um código próprio. Um exemplo de reúso em nível de componente é aquele no qual se constrói uma interface de usuário usando um *framework*.

Este é um conjunto de classes de objetos em geral que implementam manipulação de eventos, gerencimento de displays etc. São adicionadas as conexões com os dados a serem exibidos e escreve o código para definir detalhes específicos do display, como o arranjo da tela e as cores.

4. **O nível de sistema**. Nesse nível, os sistemas de aplicação inteiros são reutilizados, o que geralmente envolve algum tipo de configuração desses sistemas. Essas configurações podem ser feitas por meio da adição e modificação do código ou pelo uso de interface de configuração do próprio sistema. A maioria dos sistemas comerciais são criados dessa forma, em que sistemas genéricos de COTS (commercial off-the-shelf) são adaptados e reusados. Às vezes, essa abordagem pode envolver o reúso e a integração de diversos sistemas para criar um novo.

Ao reusar softwares existentes, é possível desenvolver novos sistemas mais rapidamente, com menos riscos de desenvolvimento e custos mais baixos. Como o software reusado foi testado em outras aplicações, deve ser mais confiável que o novo software. No entanto, existem custos associados ao reúso:

1. Custos de tempo gasto na procura do software para reúso e na avaliação sobre ele atender ou não às necessidades. Em alguns casos é preciso testar o software para ter certeza de que vai funcionar em seu ambiente, sobretudo se ele for diferente do ambiente de desenvolvimento.
2. Quando se aplicam os custos de aquisição do software reusável, para grandes sistemas de prateleira, esses custos podem ser muito elevados.
3. Custos de adaptação e configuração dos componentes de software reusável ou sistemas para refletir os requisitos do sistema que está desenvolvendo.
4. Custos de integração dos componentes reusáveis (por uso de softwares com diferentes fontes) com o novo código que foi desenvolvido. A integração de softwares reusáveis de diferentes fornecedores pode ser difícil e cara, pois cada um pode fazer suposições conflitantes sobre como seus respectivos softwares serão reusados.

Reusar um conhecimento ou software existente deve ser a primeira coisa pensada antes de iniciar um projeto de desenvolvimento. É necessário considerar as possibilidades de reúso antes de projetar o sistema em detalhes, podendo assim adaptar o projeto para reusar ativos existentes.
Em um processo de desenvolvimento orientado ao reúso, existe a procura por elementos reusáveis e, em seguida, alteração do projeto e requisitos para fazer melhor uso desses. Para um grande número de sistemas e aplicações, a ESW realmente significa o reúso de software, por isso, faz-se necessário avaliar os custos, as vantagens e desvantagens para a adoção desta metodologia.

Capítulo Três

IEC 61131-3: *Funtion Block* e a IEC 61499

As formas de programação dos controladores lógicos programáveis (CLP) foram desenvolvidas em diferentes sintaxes, desde a sua criação, sendo peculiares aos fabricantes. Isto impulsionou o surgimento de muitos projetos elaborados de maneira artesanal, não particionada e não modularizada, destacando-se assim alguns complicadores: como o alto custo para ampliação dos sistemas; necessidade de domínio e conhecimento específico para cada código; e dificuldades nos diagnósticos de falhas.

Em dezembro de 1993 foi publicada a IEC 61131 (que determina a padronização dos controladores lógicos programáveis), ela contribuiu de forma imprescindível no processo de uniformização global das ferramentas para desenvolvimento de lógicas para os CLPs, possibilitando a construção de blocos de funções, estruturas de dados e o instanciamento entre estes artefatos.

3.1 IEC 61131

Com o aparecimento de diversos fabricantes de CLPs, gerou-se uma grande variedade de equipamentos e como consequência uma incompatibilidade das características referentes à programação dos mesmos. Para atender às demandas da comunidade industrial internacional, foi formado um grupo de trabalho dentro da International Electrotechnical Commission (IEC) para avaliar e especificar critérios para o projeto completo de controladores lógicos programáveis, incluindo hardware, instalação, testes, documentação, programação e comunicação. Atualmente a IEC 61131 é dividido em oito partes:

1. Definição da terminologia e conceitos.
2. Teste de verificação e fabricação eletrônica e mecânica.
3. Estrutura do software do CLP, linguagens e execução de programas.
4. Orientações para seleção, instalação e manutenção de CLP's.
5. Funcionalidades para comunicação com outros dispositivos.
6. Reservada
7. Funcionalidades de software, incluindo blocos funcionais padrões para tratamento de lógica nebulosa dentro de CLPs.

8. Orientações para implementação das linguagens IEC 61131-3.

Sobre as formas de programação dos CLPs, a linguagem mais difundida é o Ladder, esta possui uma representação gráfica muito semelhante ao diagrama elétrico e de painéis por controle a relés. Entretanto as partes 3 e 8 da IEC 61131 estabelecem novas formas de programação e outros critérios para tipos de variáveis e estrutura de dados. Atualmente as linguagens disponíveis nos CLPs que seguem esta norma, em sua ultima versão publicada em fevereiro de 2013, são:

1. **Diagrama Ladder (LD)**: Representação gráfica, "diagrama de circuito" de variáveis booleanas (contatos e bobinas), vista geométrica de um circuito similar aos antigos controles a relé.
2. **Diagrama de bloco de função (FBD)**: Representação gráfica, conexão de elementos aritméticos, booleanos ou outros elementos funcionais e blocos de função.
3. **O texto estruturado (ST)**: Representação textual, linguagem de alto nível (semelhante ao PASCAL) para tarefas de controle, bem como cálculos complexos (matemáticos).
4. **Lista de instruções (IL)**: Representação textual, linguagem orientada a máquina de baixo nível oferecida pela maioria dos sistemas de programação
5. **Sequencial de funções (SFC)**: Representação gráfica, possui elementos para organizar programas de controle sequencial e/ou paralelo. SFC descreve muito claramente o fluxo do programa definindo quais ações do processo controlado serão habilitadas, desabilitadas ou encerradas a qualquer momento.
6. **Funções Gráficas Contínuas (CFC)**: Extensão da IEC 61131-3, que dá liberdade à posição dos elementos do gráfico.

Não será apresentada neste referencial teórico a descrição detalhada de cada uma das sintaxes de programação enumeradas acima, estas informações poderão ser encontradas na literatura (COMMISSION, 2003). Apenas serão discutidas, a partir dos próximos tópicos, as particularidades desta norma que favoreceram o reúso nas aplicações dos CLPs.

3.1.1 Unidades de Organização do Programa(POU)

A IEC 61131-3 denomina os blocos dos quais os programas e projetos são construídos como Unidades de Organização do Programa (POUs). As POUs correspondem aos blocos de

Tabela 3.1: Tipos de POU e significados

Tipo - POU	Palavra-chave	Significado
Program	PROGRAM	Programa principal incluindo atribuição de E / S, variáveis globais e pastas de acesso
Function Block	FUNCTION_BLOCK	Bloco com variáveis de entrada e saída; Este é o tipo de POU mais utilizado
Function	FUNCTION	Bloco com valor de função para extensão do conjunto de operação básica do CLP

Fonte: John & Tiegelkamp (1995)

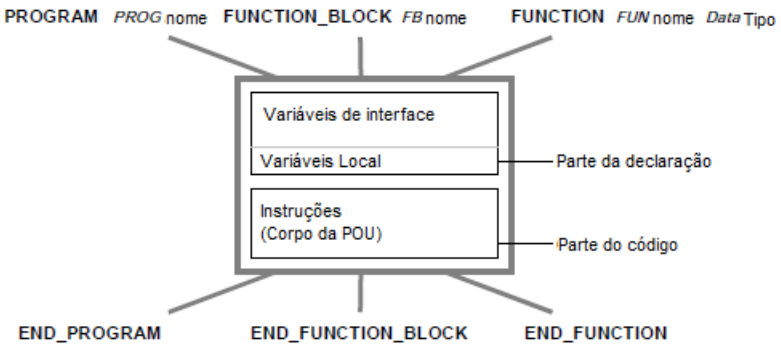
programa, de organização, de sequência e blocos de função da programação convencional. Um objetivo muito importante do padrão é restringir a variedade e, muitas vezes, significados implícitos dos tipos de blocos, unificar e simplificar seu uso. A tabela 3.1 representa os tipos e seus significados.

Estes três tipos de POUs diferem uns dos outros com determinadas características:

- **Função (FUN)** - POU que pode ser atribuído parâmetros, mas não tem variáveis estáticas (sem memória), que, quando invocado com os mesmos parâmetros de entrada, sempre produz o mesmo resultado que seu valor de função (saída).
- **Bloco de função (FB)** - POU que pode ser atribuído parâmetros e tem variáveis estáticas (com memória). Um FB (por exemplo um contador ou um bloco temporizador), quando invocado com os mesmos parâmetros de entrada, produzirá valores que também dependem do estado de suas variáveis internas (VAR) e externas (VAR_EXTERNAL), que são retidas de uma execução do Bloco de funções para o próximo.
- **Programa (PROG)** - Este tipo de POU representa o "programa principal". Todas as variáveis que são atribuídas a endereços físicos (por exemplo entradas e saídas de CLP) devem ser declaradas nesta POU ou acima dela (Resource, Configuration). Em todos os outros aspectos, ela se comporta como um FB.

Esta independência de POUs facilita a modularização extensiva de tarefas da automação, bem como a reutilização de unidades do software já implementadas e testadas. A figura 3.1 demonstra a estrutura comum dos três tipos de unidades do programa, e essas POUs podem ser descritas através das 05 formas de programação que foram enumeradas na subseção 3.1.

Figura 3.1: Estrutura comum das POUs



Fonte: John & Tiegelkamp (1995)

Existem restrições para os tipos de variáveis acessíveis por cada POU. O “PROGRAM” pode acessar todos os tipos, o “FUNCTION_BLOCK” não pode acessar os dados do tipo “global” disponibilizado por outras funções, para isso é necessário usar a variável tipo “VAR_EXTERNAL”. O “FUNCTION” tem mais restrições porque somente as variáveis locais e de entrada são permitidas nelas, elas disponibilizam o resultado de seu cálculo usando o valor de retorno da função.
Exceto para variáveis locais, todos os outros tipos podem ser usados para importar e exportar dados de uma POU. Isso possibilita a troca de informações entre POUs, este recurso pode ser chamado de interface da POU.

3.1.2 Function Block (FB)

Atualmente os *Function Blocks* são os principais elementos para construção e estruturação de programas no CLP. Na prática, eles são chamados por meio de outras funções e podem chamar rotinas, bem como outros FBs. O conceito de "instanciação de FBs"é de grande importância na IEC 61131-3 e é um critério distintivo essencial entre os três tipos de POU. Este conceito será, portanto, introduzido antes de explicar as outras características das POUs.
Os blocos de função também podem ser instanciados como variáveis: por exemplo em 3.2, a instância FB Motor1 é declarada como uma “variável” do tipo de função definida pelo usuário (tipo FB) MotorType na parte da declaração de uma POU. Após instanciado, um FB pode ser usado (como uma “Variável”) e chamado dentro da POU em que é declarado.

Este princípio pode parecer incomum à primeira vista, mas, de fato, não é nada de novo. Até antes da IEC 61131, os blocos de função para contagem ou temporização, conhecidos brevemente como contadores e temporizadores, respectivamente, foram definidos principalmente pelo seu tipo (tal como a direção de contagem ou o comportamento de

Tabela 3.2: Exemplo de declaração de variável comum e do tipo FB

Nome	**tipo da variável**	**Descrição**
Valve:	BOOL	(*Variável Booleana*)
Motor1:	MotorType	(*Instancia de FB*)

Fonte: John & Tiegelkamp (1995)

temporização) e por uma numeração fornecida pelo utilizador, como por exemplo contador "C19".
Em vez deste número absoluto, o padrão requer um nome de variável (simbólico) combinado com a especificação do temporizador ou tipo de contador desejado. Isso deve acontecer na parte de declaração da POU, mas o sistema de programação pode gerar automaticamente números internos absolutos para eles ao compilar o código em linguagem de máquina para o CLP. Com auxílio desses nomes de variáveis, o programador de CLP pode utilizar temporizadores ou contadores diferentes do mesmo tipo, de forma transparente e sem a necessidade de verificar conflitos por nomenclatura.
Através da instanciação, a IEC 61131-3 unifica o uso de FBs, antigamente dependentes de fabricantes (normalmente timers e contadores) e FBs definidos pelo usuário. Os nomes das instâncias correspondem aos chamados símbolos, usados por muitas IDEs (*Integrated Development Environment* - Ambiente de desenvolvimento integrado) de programação. Da mesma forma, um tipo FB é correspondente com a sua interface de chamada.

A estrutura de dados no exemplo 3.2 mostra os parâmetros formais (interface de chamada) e valores de retorno do padrão FB CTUD, ela representa a visão do invocador do FB. Variáveis locais ou externas da POU são mantidas ocultas.

Figura 3.2: Representação alternativa da estrutura de dados do contador up/ down

```
TYPE CTUD :    (* data structure of an FB instance of FB type CTUD *)
  STRUCT
    (* inputs *)
    CU :        BOOL;   (* count up *)
    CD :        BOOL;   (* count down *)
    R  :        BOOL;   (* reset *)
    LD :        BOOL;   (* load *)
    PV :        INT;    (* preset value *)
    (* outputs *)
    QU :        BOOL;   (* output up *)
    QD :        BOOL;   (* output down *)
    CV :        INT;    (* current value *)
  END_STRUCT;
END_TYPE
```

Fonte: John & Tiegelkamp (1995)

Esta estrutura de dados é gerenciada automaticamente pela IDE ou em tempo de execução, simplificando a utilização quanto a atribuição de parâmetros às FBs, via linguagem

de programação por lista de instrução. A imagem 3.3 explica as formas de uso desta estrutura de dados no programa:

Figura 3.3: Parametrização e invocação do contador up/ down

```
LD    34
ST    Counter.PV    (* preset count value *)
LD    %IX7.1
ST    Counter.CU    (* count up *)
LD    %M3.4
ST    Counter.R     (* reset counter *)
CAL   Counter       (* invocation of FB with actual parameters *)
LD    Counter.CV    (* get current count value *)
```

Fonte: John & Tiegelkamp (1995)

Neste exemplo, o contador instanciado recebe como parâmetro para o limite máximo da contagem o valor 34, o comando para cada ciclo de contagem feito através do "%IX7.1" e o reset por "%M3.4", isto tudo antes que o FB seja invocado pela instrução "CAL". Por fim, o valor do contador pode então ser lido pela variável "Counter.CV".

As figuras 3.4 e 3.5 representam, respectivamente, a declaração e invocação de FBs no ambiente de programação, de forma textual e gráfica.

Figura 3.4: Declaração do FB em linguagem gráfica e textual

```
(* a) Textual declaration in ST language (see 3.3) *)

FUNCTION_BLOCK DEBOUNCE
(*** External Interface ***)
VAR_INPUT
  IN : BOOL ;                    (* Default = 0 *)
  DB_TIME : TIME := t#10ms ;     (* Default = t#10ms *)
END_VAR
VAR_OUTPUT OUT : BOOL ;         (* Default = 0 *)
    ET_OFF : TIME ;             (* Default = t#0s *)
END_VAR
VAR DB_ON : TON ;               (** Internal Variables **)
    DB_OFF : TON ;              (**  and FB Instances  **)
    DB_FF : SR ;
END_VAR

(** Function Block Body **)
DB_ON(IN := IN, PT := DB_TIME) ;
DB_OFF(IN := NOT IN, PT:=DB_TIME) ;
DB_FF(S1 :=DB_ON.Q, R := DB_OFF.Q) ;
OUT := DB_FF.Q ;
ET_OFF := DB_OFF.ET ;

END_FUNCTION_BLOCK
```

```
(* b) Graphical declaration in FBD language (see 4.3) *)

FUNCTION_BLOCK
(** External Interface **)
                  +----------------+
                  |   DEBOUNCE     |
          BOOL---|IN           OUT|---BOOL
          TIME---|DB_TIME   ET_OFF|---TIME
                  +----------------+
(** Function Block Body **)

                DB_ON         DB_FF
               +------+      +----+
               | TON  |      | SR |
 IN----+-------|IN  Q|------|S1 Q|---OUT
       |  +----|PT ET|   +--|R   |
       |  |    +------+   |  +----+
       |  |               |
       |  |     DB_OFF    |
       |  |    +------+   |
       |  |    | TON  |   |
       |  +--|--O|IN  Q|--+
 DB_TIME--+---|PT ET|--------------ET_OFF
               +------+
 END_FUNCTION_BLOCK
```

Fonte: Commission (2003)

Figura 3.5: Parametrização e invocação do FB em linguagem gráfica e textual

Graphical (FBD language)	Textual (ST language)
FF75 +------+ | SR | %IX1---|S1 Q1|---%QX3 %IX2---|R | +------+	VAR FF75: SR; END_VAR (* Declaration *) FF75(S1:=%IX1, R:=%IX2); (* Invocation *) %QX3 := FF75.Q1 ; (* Assign Output *)
MyTon +-------+ +----+ | TON | a--| NE |---O|EN ENO|-- b--| | r--|IN Q|O-out +----+ --|PT ET|-- +-------+	VAR a,b,r,out : BOOL; MyTon : TON; END_VAR MyTon(EN := NOT (a <> b), IN := r, NOT Q => out);

Fonte: Commission (2003)

3.1.3 *FBs reutilizáveis*

Os blocos de função estão sujeitos a certas restrições que os tornam reutilizáveis em programas de CLP:

- A declaração de variáveis com atribuição fixa aos endereços de hardware do CLP (Variáveis representadas diretamente: %Q, %I, %M) como variáveis "locais" não é permitida em blocos de função. Isso garante que os FBs sejam independentes de um hardware específico.
- A declaração dos caminhos de acesso do tipo de variável VAR_ACCESS ou variáveis globais com VAR_GLOBAL também não são permitidos nos FBs. Dados globais e, portanto, indiretamente, caminhos de acesso, podem ser acessados por meio de VAR_EXTERNAL.
- Os dados externos só podem ser passados para o FB por meio da interface POU usando parâmetros e variáveis externas. Não há "herança", como em algumas outras linguagens de programação.

Como resultado dessas características, os blocos de função também são referidos como encapsulados, indicando que eles podem ser usados universalmente e estão livres de efeitos colaterais indesejados durante a execução, uma propriedade importante para partes de programas de CLP. Os dados FBs locais e, portanto, a função FB não dependem diretamente de variáveis globais, caminhos de comunicação I/O ou do sistema. Os FBs podem manipular tais áreas de dados apenas indiretamente através de sua interface (bem

documentada).

O modelo de instância FB com as propriedades de "estrutura" e "memória" foi introduzido na seção anterior, juntamente com a propriedade de encapsulamento para reutilização. Isso pode ser resumido da seguinte forma: "Um bloco de função é uma estrutura de dados encapsulada independente com um algoritmo definido trabalhando nestes dados."

O algoritmo é representado pela parte de código do FB. A estrutura de dados corresponde à instância FB e pode ser "invocada", algo que não é possível com as estruturas de dados normais. Para cada tipo de FB qualquer, números de instâncias podem ser derivados, cada um independente do outro. Cada instância tem um nome exclusivo com sua própria área de dados. Devido a isso, a IEC 61131-3 considera os blocos de função como sendo orientados a objetos. Esses recursos não devem, contudo, ser confundidos com os das modernas "linguagens de programação orientada a objetos" ("POO"), como, por exemplo, C ++ com sua hierarquia de classes.
Para resumir, os FBs trabalham na sua própria área de dados contendo entrada, saída e variáveis locais. Em sistemas de programação CLP anteriores, os FBs normalmente trabalhavam em áreas de dados globais, como sinalizadores, memória compartilhada, E/S e blocos de dados.

3.1.4 Data type

Cada CLP suporta um determinado grupo padrão de tipos de dados. A lista normalmente inclui *bit, bytes, Word, DoubleWord, Real, time, string*, entre outros. Os nomes para cada tipo de dado elementar, o número de bits por elemento de dados e o intervalo de valores para cada tipo de dados elementar são indicados na tabela 3.3.

3.1.5 Derived Data Type e User Data Type (UDT)

Dentre os diferentes tipos de dados, o mais importante para a construção da modelagem desta dissertação são as estruturas definidas pelo próprio usuário ou User Data Type (UDT), esta tem uma organização semelhante às variáveis *STRUCT* em linguagem de programação "C". Uma declaração *STRUCT* especifica que os elementos de dados desse tipo devem conter subelementos de tipos especificados que podem ser acessados pelos nomes específicos. Por exemplo, um elemento do tipo ANALOG_CHANNEL_CONFIGURATION conforme declarado na tabela 3.4 conterá um subelemento RANGE do tipo ANALOG_SIG-

Tabela 3.3: Tipos de dados elementares

Nome	**Tipo de dados**	N^a
BOOL	Booleano	1^h
SINT	Inteiro curto	8^c
INT	Inteiro	16^c
DINT	Duplo inteiro	32^c
LINT	Longo inteiro	64^c
USINT	Inteiro curto sem sinal	8^d
UINT	Inteiro sem sinal	16^d
UDINT	Duplo inteiro sem sinal	32^d
ULINT	Longo inteiro sem sinal	64^d
REAL	Número real	32^e
LREAL	Número real longo	64^f
TIME	Duração de tempo	$--^b$
DATE	Data (Apenas)	$--^b$
TIME_OF_DAY ou TOD	Tempo do dia (Apenas)	$--^b$
DATE_AND_TIME ou DT	Data e tempo do dia	$--^b$
STRING	Cadeia de Caracteres de um byte de comprimento variável	$8^{i,g}$
BYTE	Cadeia de bits de comprimento 8	$8^{j,g}$
WORD	Cadeia de bits de comprimento 16	$16^{j,g}$
DWORD	Cadeia de bits de comprimento 32	$32^{j,g}$
LWORD	Cadeia de bits de comprimento 64	$64^{j,g}$
WSTRING	Cadeia de caracteres duplos de bytes de comprimento variável	$16(i, g)$

^{a}	As entradas nesta coluna devem ser interpretadas como especificado nas notas de rodapé.
^{b}	O intervalo de valores e precisão de representação nestes tipos de dados é dependente da implementação.
^{c}	O intervalo de valores para variáveis deste tipo de dados é de - (2N-1) a (2N-1) -1.
^{d}	O intervalo de valores para variáveis deste tipo de dados é de 0 a (2N) -1.
^{e}	O intervalo de valores para as variáveis deste tipo de dados deve ser conforme definido na IEC 60559 para o formato de ponto flutuante de uma única largura.
^{f}	O intervalo de valores para as variáveis deste tipo de dados deve ser conforme definido na IEC 60559 para o formato de ponto flutuante de largura dupla básica.
^{g}	Um intervalo numérico de valores não se aplica a esse tipo de dados.
^{h}	Os valores possíveis das variáveis deste tipo de dados devem ser 0 e 1, correspondendo às palavras-chave FALSE e TRUE, respectivamente.
^{i}	O valor de N indica o número de bits/ caracteres para este tipo de dados.
^{j}	O valor de N indica o número de bits na seqüência de bits para esse tipo de dados.

Commission (2003)

NAL_RANGE, um subelemento MIN_SCALE do tipo ANALOG_DATA e um elemento MAX_SCALE do tipo ANALOG_DATA. O número máximo de elementos da estrutura, a quantidade máxima de dados que podem ser contidos em uma estrutura e o número máximo de níveis aninhados de endereçamento do elemento da estrutura são parâmetros dependentes da implementação.

Uma declaração *ARRAY* especifica que uma quantidade suficiente de armazenamento de dados deve ser alocada para cada elemento desse tipo armazenar todos os dados indexados pela(s) subrange(s) de índice especificada. Assim, qualquer elemento do tipo ANALOG_16_INPUT_CONFIGURATION como mostrado na tabela 3.4 contém (entre outros elementos) armazenamento suficiente para 16 elementos CHANNEL do tipo ANALOG_CHANNEL_CONFIGURATION. O número máximo de subscritos de matriz, o tamanho e o intervalo máximo de valores do subíndice são parâmetros dependentes da implementação.

As estruturas de dados são tipos de dados personalizados que são construídos agrupando dados nativos ou outras *struct*, elas são conhecidas também como *User Defined Types* (UDT) e são fundamentais para a implementação do modelo proposto neste trabalho. Semelhante as *structs* utilizadas por outras linguagens de programação, este conceito permite que a organização e melhor disposição dos dados em programas. Este conceito será novamente explorado e utilizado no capitulo 5, onde os dados serão encapsulados em uma variável personalizando informações relevantes para o modelo. Estes dados podem ser facilmente passados de um lugar para outro em um pedaço grande e há muitos benefícios para organizar variáveis em estruturas de dados:

- **Redução de erros na programação:** Movimentação de dados em grandes grupos reduz as chances de erros que estariam associados à passagem de diversas variáveis individuais;
- **Evita a programação redundante:** Menor quantidade de linhas de códigos necessárias para movimentar dados de um lugar para outro;
- **Aplica coerência:** Definida a estrutura de dados uma vez é possível utilizá-la em todo o programa.
- **Facilita alterações rápidas no programa:** Uma vez que a estrutura de dados é definida em uma única parte do projeto, pode-se facilmente fazer alterações na *struct* que se propagam automaticamente em todo o programa.

Tabela 3.4: Recursos para declaração dos tipos de dados

No.	Recurso/ Exemplo textual
1	Derivação direta de um tipo de dado elementar: TYPE RU_REAL : REAL END_TYPE
2	Tipo de dado enumerated: TYPE ANALOG_SIGNAL_TYPE : (SINGLE_ENDED, DIFFERENTIAL) END_TYPE
3	Tipo de dado Subrange: TYPE ANALOG_DATA : INT (-4095..4095) END_TYPE
4	Tipo de dado array: TYPE ANALOG_16_INPUT_DATA : ARRAY [1..16] OF ANALOG_DATA END_TYPE
5	Tipo de dado struct: TYPE ANALOG_CHANNEL_CONFIGURATION : STRUCT RANGE : ANALOG_SIGNAL_RANGE ; MIN_SCALE : ANALOG_DATA ; MAX_SCALE : ANALOG_DATA ; END_STRUCT ; ANALOG_16_INPUT_CONFIGURATION : STRUCT SIGNAL_TYPE : ANALOG_SIGNAL_TYPE ; FILTER_PARAMETER : SINT (0..99) ; CHANNEL : ARRAY [1..16] OF ANALOG_CHANNEL_CONFIGURATION ; END_STRUCT ; END_TYPE

Commission (2003)

3.1.6 Utilização da UDT

Um elemento de uma variável com múltiplos elementos pode ser usado em qualquer lugar onde o tipo "pai" for usado, por exemplo, dado a declaração de ANALOG_16_INPUT_DATA na tabela 3.4, sendo:

VAR INS: ANALOG_16_INPUT_DATA; END_VAR

As variáveis INS[1] a INS[16] podem ser usadas em qualquer lugar onde uma variável do tipo INT for permitida. Esta regra também pode ser aplicada recursivamente, por exemplo, dadas as declarações de ANALOG_16_INPUT_CONFIGURATION, ANALOG_CHANNEL_CONFIGURATION e ANALOG_DATA na tabela 12 e a declaração:

VAR CONF: ANALOG_16_INPUT_CONFIGURATION; END_VAR

A variável CONF.CHANNEL[2].MIN_SCALE pode ser usada em qualquer lugar onde uma variável do tipo INT tiver a utilização permitida.

3.2 Justificativas e a importância do uso da IEC 61131 para esta pesquisa

A utilização de um padrão internacionalmente conhecido favoreceu a adoção de especificações técnicas bem definidas e delimitadas, é inegável que a IEC 61131 evoluiu a tenologia e desenvolveu as aplicações dos controladores lógicos nos ambientes industriais. John & Tiegelkamp (1995) expõem algumas vantagens, estas também estão associadas diretamente com este trabalho e serão relacionadas a seguir:

3.2.1 Conveniência e segurança com variáveis e tipos de dados

- **Variáveis locais e globais em vez de endereços de hardware**: Anteriormente, toda a memória de dados em um CLP era acessada utilizando endereços globais, o programador precisava estar atento para que uma parte de um programa não substituísse os dados de outra. Isso se aplicava particularmente aos endereços de E/ S, flags e blocos de dados. O padrão IEC 61131-3 substituiu todos os endereços de hardware global por variáveis nomeadas com um escopo definido: o sistema de programação distingue automaticamente variáveis globais e locais de uma POU. Endereços globais passaram a ser acessados atribuindo o seu endereço para uma variável nomeada na parte de declaração.

- **Acesso aos dados no CLP orientado por tipo:** Os programadores precisavam ser cuidadosos ao usar os mesmos dados de leitura ou escrita nos endereços de programa. Era possível acessar o mesmo endereço de memória como um número inteiro em uma parte do programa e como um ponto flutuante em outro. A IEC 61131-3 evita que tais erros de programação ocorram, uma vez que cada variável (incluindo endereços de hardware diretos) deve ser atribuída a um tipo de dado. O sistema de programação pode verificar se todos os acessos usam dados apropriados.

- **Valores iniciais definidos pelo usuário:** Todos os dados são explicitamente declarados sob a forma de uma variável, e atribuído um tipo de dado na declaração. Cada tipo de dados tem, por padrão ou conforme especificado pelo usuário, um valor inicial definido, de modo que cada variável usada em um programa é sempre inicializada corretamente de acordo com suas propriedades. As variáveis podem ser declaradas como retentivas (com um qualificador RETAIN). Elas são atribuídas automaticamente a uma área de memória RAM com suporte de bateria pelo sistema de programação.

- ***Arrays* e estruturas de dados para cada aplicação:** Com base nos tipos de dados predefinidos, o programador de CLP projeta *arrays* e outras estruturas de dados complexas para coincidir com a aplicação, como é a prática com linguagens de alto nível. Limites de índices de matriz e intervalos de valores das variáveis são verificados pelo sistema de programação, bem como pelo CLP em tempo de execução. Este recurso será amplamente explorado no capítulo 5 onde serão apresentados os diagramas de definição de bloco para cada componente.

- **Declaração unificada de variáveis:** As extensas facilidades para o uso de variáveis são geralmente idênticas em todas as línguas definidas pela IEC 61131.

3.2.2 Blocos com capacidade estendida

- **Reutilização de blocos:** Os Blocos (POUs), como funções e FBs, podem ser projetados para serem independentes do sistema de destino usado. Isto possibilita a construção de bibliotecas com blocos reutilizáveis, disponíveis para múltiplas plataformas. Os parâmetros de entrada, bem como de saída e dados locais para cada instância, de um FB mantém seus valores entre as suas chamadas. Cada FB tem sua própria área de dados na memória, onde poderá executar seus cálculos independente dos dados externos, não sendo obrigatório instanciar uma base de dados para executar o FB. Esta reutilização favorecerá a aplicação do modelo apresentado no capítulo 5.

- **Atribuição eficiente dos parâmetros de blocos de funções:** O padrão fornece uma variedade de mecanismos para passar dados de e para os blocos de funções:

VAR_INPUT: Valor de uma variável

VAR_IN_OUT: Ponteiro para uma variável

VAR_OUTPUT: Valor de retorno

VAR_EXTERNAL: variável global de outra POU

VAR_ACCESS: caminho de acesso dentro de uma configuração.

Graças ao mecanismo de endereçar variáveis do tipo entrada e saída, tornou-se possível associar as memórias tipo UDT, mencionadas anteriormente.

- **Funcionalidades padronizada no CLP:** Para padronizar a funcionalidade típica do CLP, a IEC 61131-3 define um conjunto de funções e FBs, a interface de chamada, o arranjo gráfico e o comportamento em tempo de execução, esta "biblioteca" padrão para os CLPs é uma base importante para treinamento, programação, documentação uniforme e independente do fabricante.

3.2.3 Linguagens de programação uniformes

IEC 61131-3 define cinco linguagens de programação que podem cobrir uma ampla gama de aplicações. Como resultado desta norma internacional, os especialistas passaram a receber treinamentos mais uniformes e começaram a falar "a mesma linguagem" onde quer que estejam empregados. Com isto o custo da formação foi reduzido, uma vez que apenas as características específicas de cada novo controlador precisariam ser aprendidas, além disso a documentação passou a ser mais uniforme, mesmo em hardwares diferentes.

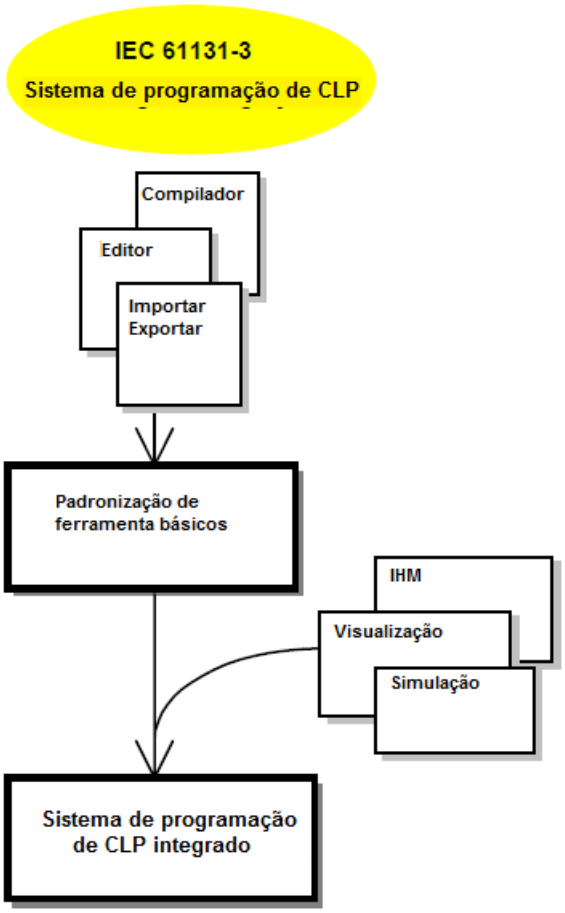
3.2.4 Tendência para Sistemas de Programação de CLP Abertos

A padronização das linguagens de programação levaram à uniformização dos softwares, o que tornou viáveis os programas portáteis independentes do fornecedor. A "tabela de características" da IEC 61131-3 fornecem uma base para comparar sistemas de programação com a mesma funcionalidade básica de diferentes fornecedores. Ainda haverá diferenças entre os sistemas de diversos fabricantes, mas estas serão encontradas principalmente em ferramentas adicionais, como analisadores lógicos ou nos simuladores *off-line*, e não nas próprias linguagens de programação.

A nova geração dos sistemas de programação dos CLPs terá uma funcionalidade básica padronizada, além de ferramentas altamente sofisticadas para cobrir uma ampla gama de aplicações. Conforme mostrado na Figura 3.6, a padronização pela IEC 61131-3 promove sistemas integrados, construídos a partir de componentes padronizados como editores,

compiladores, exportação e importação de utilitários, com interfaces abertas e reutilizáveis.

Figura 3.6: Tendência para componentes abertos e padronizados baseados em sistemas de programação compatíveis com IEC 61131-3



Fonte: John & Tiegelkamp (1995)

Atualmente já são comercializados IDEs destinadas ao desenvolvimento de lógicas para CLPs, se valendo destes princípios, como é o caso do CODESYS.

3.3 IEC 61499

A programação utilizando elementos gráficos extraídos do "mundo real" está se tornando cada vez mais importante. Com as linguagens gráficas, o fluxo de dados e a sequência de execução lógica podem ser programados e documentados usando símbolos e nomes. No entanto, a capacidade de exibir graficamente a distribuição topológica dos programas é também desejável, além da sua configuração geral e das interligações a outras partes de um projeto em automação distribuída. Isso ocorre em um nível mais alto e abstrato do que meramente a programação de POUs, descrita anteriormente.

As ferramentas para configurar aplicativos complexos e distribuídos são chamadas como editores de configuração. As partes do programa, a exemplo dos blocos funcionais (FBs), são combinadas para formar unidades maiores. Isto é feito através da interconexão de blocos de função.
Segundo John & Tiegelkamp (1995) a arquitetura IEC 61499 representa uma solução componente para sistemas de automação industrial distribuídos, com o objetivo de permitir portabilidade, interoperabilidade, reutilização, reconfiguração de aplicações distribuídas. Este padrão fornece um modelo genérico para sistemas distribuídos, nele estão incluídos processos e redes de comunicação como um ambiente para dispositivos embarcados, recursos e aplicações. Sendo assim as aplicações são construídas como uma rede de blocos de funções.

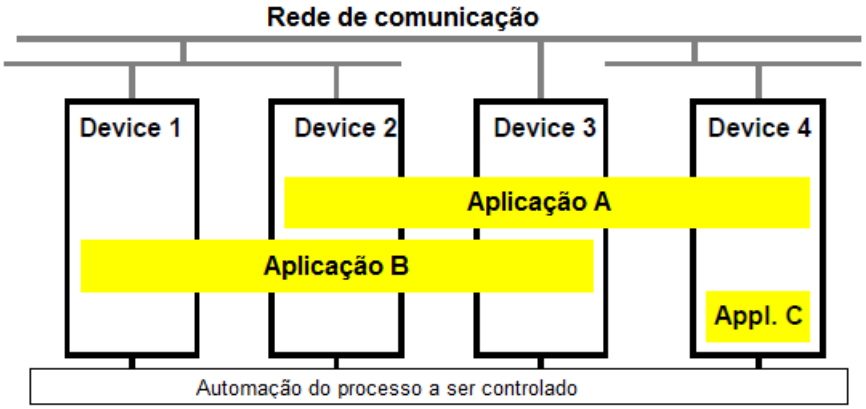
A invocação sequencial de blocos definidos na IEC 61131-3 não é um método adequado para a estruturação de programas em sistemas distribuídos. O objetivo de um sistema descentralizado distribuído é distribuir programas entre várias unidades de controle e executá-los em paralelo (em contraste com a execução sequencial). Aqui é essencial assegurar a consistência dos dados entre os nós do sistema em rede, isto é, definir tempos exatos para a troca mútua de dados.
Dois tipos de troca de informações desempenham um papel essencial na IEC 61499:

- Fluxo de dados do usuário;
- Controle de fluxo e a validade dos dados de usuário, como informações sobre evento.

3.3.0.1 Modelo de sistema

Num ambiente de automação real, várias unidades de controle, aqui referidas como dispositivos, executam as mesmas ou diferentes aplicações em paralelo. A figura 3.7 descreve isto.

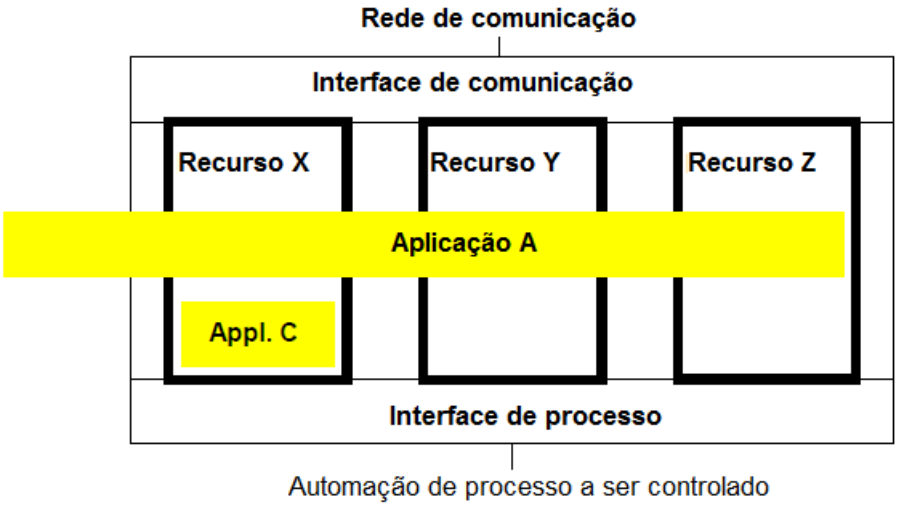
Figura 3.7: Controle de um processo real distribuído entre vários dispositivos.



Fonte: John & Tiegelkamp (1995)

Um dispositivo pode conter vários recursos, que usam interfaces comuns para trocar informações com outras unidades de controle e o processo de automação. A figura 3.8 descreve como são distribuídos os recursos dentro de um dispositivo.

Figura 3.8: Distribuição dos recursos em um dispositivo



Fonte: John & Tiegelkamp (1995)

3.3.0.2 Modelo de recurso

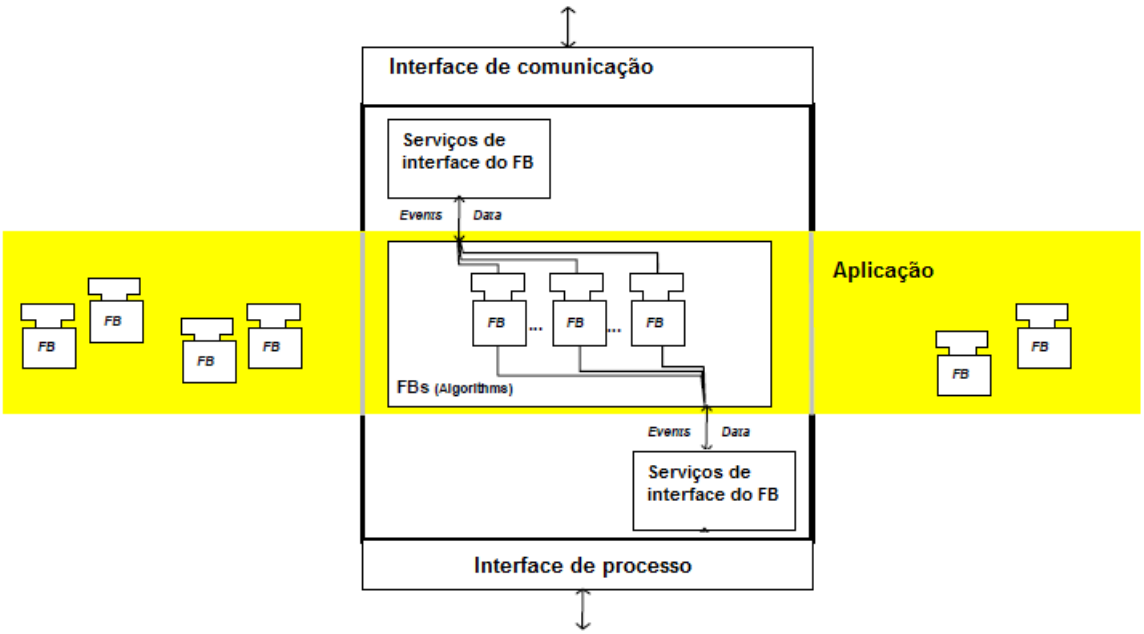
Um recurso consiste em blocos de funções, que trocam eventos, bem como informações usando interfaces especiais. Existem dois tipos de bloco de função:

1. FBs interface de serviço, que são FBs padrão e formam as interfaces para o processo de automação e a rede de comunicação.

2. Bloco de função definidos pelo usuário, que compõem o programa de aplicação real (algoritmo).

Assim como na IEC 61131-3, existe uma distinção entre o tipo FB e a instância FB. As propriedades em tempo de execução, como o número máximo de instâncias, tempo de execução, número de conexões e outros, podem ser atribuídas a cada bloco de função dentro do recurso. A interconexão de FBs pelo usuário não é realizada no nível do recurso, mas no nível do aplicativo, apresentado a seguir. O verdadeiro intercâmbio de informações entre os FBs do programa de aplicação ocorre "invisivelmente" para o usuário, através das interfaces de comunicação e processo. Um aplicativo pode ser implementado em um ou mais recursos.

Figura 3.9: Um modelo de recurso

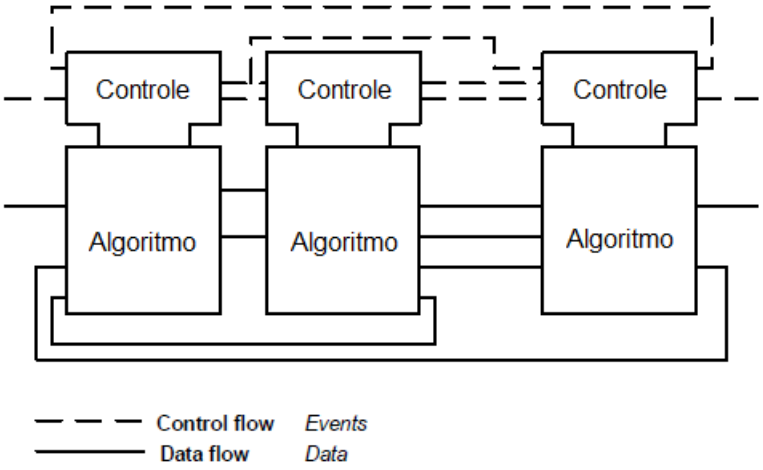


Fonte: John & Tiegelkamp (1995)

3.3.0.3 Modelo de Aplicação

O nível de aplicação representa o nível de programação real, porque é neste nível que os FBs são interconectados uns com os outros, independente dos recursos em que eles são executados. Depois que o programa aplicativo, composto por vários FBs, foi distribuído aos recursos e o programa iniciado, a comunicação entre os FBs ocorrerá de forma transparente, através das interfaces de serviço com as conexões especificadas pelo usuário. A figura 3.10 mostra como uma aplicação é composta por partes controladoras (com eventos) e de processamento de dados (algoritmos). Aqui pode ser vista a representação gráfica diferente da IEC 61131-3.

Figura 3.10: Modelo de aplicação



Fonte: John & Tiegelkamp (1995)

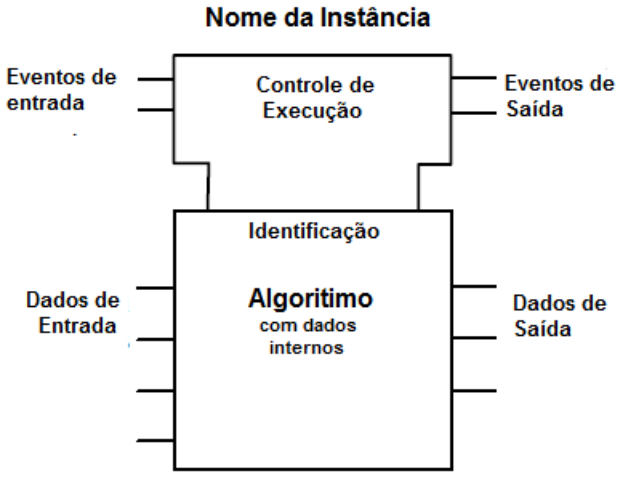
3.3.0.4 Modelo de bloco de funções

O bloco de função é a menor unidade de programa na IEC 61499. Um bloco funcional na IEC 61499 consiste geralmente de duas partes:

1. Controle de execução: criação e processamento de eventos como entradas e saídas de controle
2. Fluxo de dados e processo: algoritmo com entrada e saída de dados.

Estes blocos funcionais podem ser especificados em forma textual ou gráfica. Os FBs são instanciados para programação, da mesma forma como na IEC 61131-3. A figura 3.11 mostra a representação gráfica de um bloco funcional de acordo com a IEC 61499. A parte destinada ao algoritmo de controle do processo é programada baseada na IEC 61131-3 (como um corpo de POU). A parte do controle de execução é programada usando um diagrama de estado ou gráfico de funções sequenciais (SFC na IEC 61131-3). Os eventos são valores de entrada para diagramas de estado, ou gráficos de controle de execução (ECC). Estes ECCs controlam os tempos de execução do algoritmo ou partes dele dependendo do estado real e dos eventos de entrada.

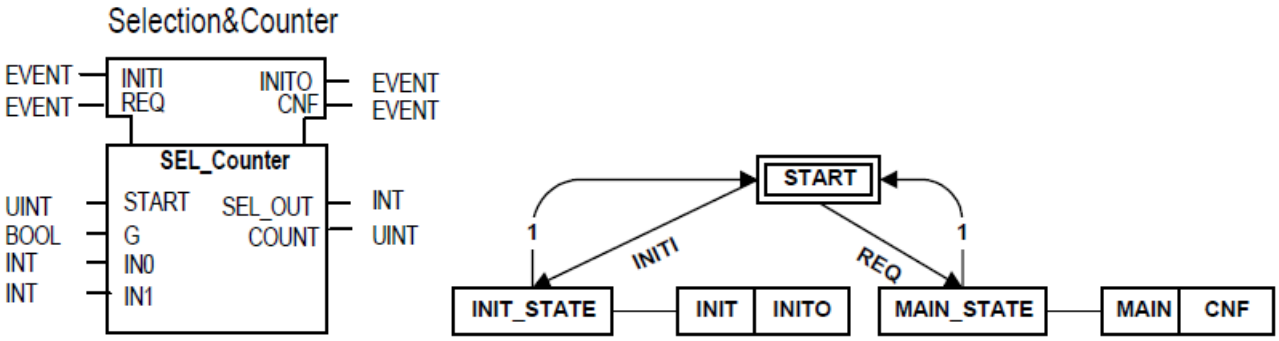
Figura 3.11: Representação gráfica de um bloco de função na IEC 61499.



Fonte: John & Tiegelkamp (1995)

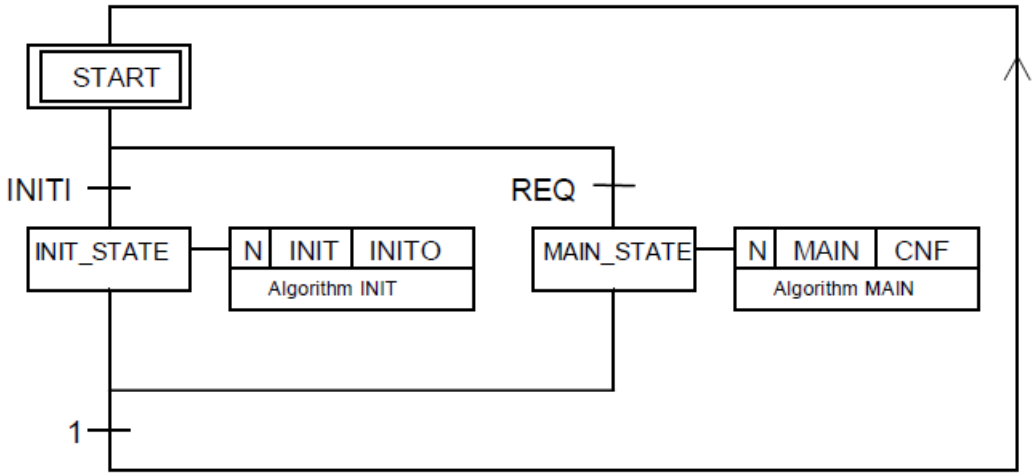
As figuras 3.12 e 3.13 representam a lógica para controle de execução num bloco de função.

Figura 3.12: Bloco de funções com parâmetros formais digitados e diagrama de estado (ECC)



Fonte: John & Tiegelkamp (1995)

Figura 3.13: Controle de execução do Exemplo 3.12 usando o SFC (Sequential Function Chart), conforme definido na IEC 61131-3.



Fonte: John & Tiegelkamp (1995)

O exemplo 3.12 contém o bloco de função Sel_Counter (nome da instância Selection&Counter), em uma parte de controle ECC e a parte do algoritmo, que consiste em dois algoritmos INIT e MAIN. O controle de execução determina qual parte do algoritmo estará ativa e em qual horário.
No Exemplo 3.12, quando ocorre o evento INITI, o controle FB mudará do estado inicial START para o estado INIT_STATE e o algoritmo INIT será executado. Depois, a variável de saída do evento INITO é ajustada (acção "N"), seguida por um RESET (isto é, um impulso de sinal). Agora, o controle de execução avalia a próxima transição. Isto tem o parâmetro constante "1"neste exemplo, significa que a condição é sempre verdadeira e leva de volta ao estado START. O evento de entrada REQ é processado analogamente.

Esse comportamento é equivalente às ações do SFC (Sequential Function Chart) na IEC 61131-3 e é ilustrado pelo exemplo 3.13. IEC 61499 pressupõe que é mais favorável especificar o controle de execução usando diagramas de estado, já que com esse método apenas um estado poderá estar ativo por vez. No SFC isto pode ser conseguido proibindo ramos simultâneos.

O bloco de função é o modelo elementar do padrão IEC 61499. Um FB geralmente fornece uma interface para entradas e saídas por eventos e dados. Existem dois tipos de FBs: básico e função composta, por outro, este segundo poderá conter outros blocos de funções compostas e/ ou FBs básicas. Assim, permitindo metodologias de projeto modulares. FBs básicas incluem gráficos de controle de execução acionados por eventos (ECC), que

são máquinas de estado. Os elementos da ECC são estados e transições acionados por evento. Um ECC pode desencadear a execução de algoritmos pela ocorrência de eventos.

3.4 Importância da IEC 61499 para este trabalho

A arquitetura IEC 61499 representa uma solução componente para sistemas de automação industrial distribuído, com o objetivo de permitir portabilidade, interoperabilidade, reutilização, reconfiguração de aplicações distribuídas. Este padrão fornece um modelo genérico para sistemas distribuídos, inclui processos e redes de comunicação como um ambiente para dispositivos embarcados, recursos e aplicações. Sendo assim as aplicações são construídas por redes de Blocos de funções (FB). O bloco de função (FB) é o modelo elementar do padrão IEC 61499. Um bloco de função geralmente fornece uma interface para entradas e saídas por eventos e dados. Um bloco de função composto pode conter outros FBs compostas e/ ou básicos. Assim, FBs compostas permitem metodologias de projeto modulares. As FBs básicas incluem gráficos de controle de execução acionados por eventos (ECC), que são máquinas de estado. Os elementos da ECC são estados e transições acionados por evento. Um ECC pode desencadear a execução de algoritmos pela ocorrência de eventos.

Os pontos principais do padrão IEC 61499 que serão aproveitados no trabalho são:

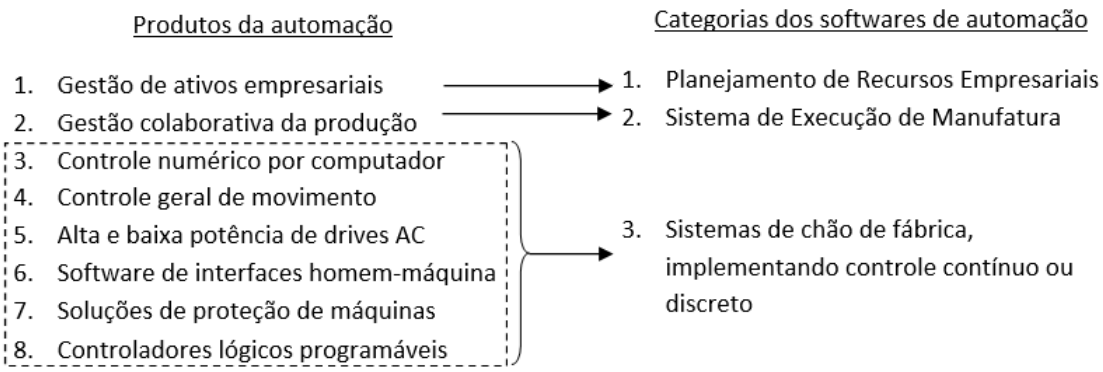
- **Arquitetura baseada em FB** - A arquitetura do sistema baseado em *function block*, como unidade elementar da lógica de controle, também será adotada na modelagem deste trabalho e apresentada no capítulo 6. Cada FB representará um componente do sistema de automação e a interação entre eles ocorrerá sempre através da interconexão de suas interfaces. Deste modo, o programador precisará apenas adicionar estes elementos e realizar a interação entre cada interface para atingir o controle e os intertravamentos do processo.
- **Eventos de entrada e saída** - Embora os eventos de E/ S dos componentes, a serem apresentados no capítulo 6, não tenham sido implementados em linguagem *sequencial fuction chart*, eles foram concebidos com o mesmo propósito e finalidade sugeridos por esta norma. Com isto, cada componente terá o seu comportamento definido pela atuação das interfaces de entrada e o seu estado será indicado pelos eventos de saída.
- **Dados de entrada e saída** - Todas as variáveis dos componentes propostos serão endereçadas como dados de entrada e saída. Sendo assim, para que o programador tenha acesso a escrita de algum ponto de ajuste ou leitura dos parâmetros dos componentes, ele precisará acessar a área destinada aos dados de entrada e saída.

Capítulo Quatro

Estado da arte da engenharia e modelagem de software aplicada à automação industrial

Este capítulo apresenta uma perspectiva sobre as recentes produções cientificas relacionadas com a engenharia e modelagem de software no âmbito da automação industrial, que abrange desde sistemas de controle automático para processos de fabricação a automação de sistemas de geração e distribuição de energia. A pesquisa de Vyatkin (2013) traz toda esta abordagem e se concentra principalmente em publicações de revistas creditadas e representativas nas avançadas práticas industriais, o trabalho de Vyatkin (2013) foi fundamental para elaboração deste capítulo por organizar produções científicas relacionadas com o uso da engenharia de software na automação industrial, contemplando ainda a utilização da IEC 61131 e IEC 61499 neste contexto.
Refletindo sobre a tendência da crescente importância do software na automação, há um grande número de projetos de pesquisas e publicações correspondentes abordando vários aspectos do processo de desenvolvimento do software, no domínio de automação industrial. Conforme ilustrado na figura 4.1 as categorias de SW dependem da camada TIA aplicada.

Figura 4.1: Relação de produtos de automação com categorias de software



Fonte: Vyatkin (2013)

4.1 Pesquisas experimentando a IEC61131

No capítulo 3 foram apresentados dois padrões com relevante influência para modelagem de softwares na categoria dos sistemas de chão de fábrica. O padrão IEC61131-3 capturou as formas mais comuns de pensamento dos engenheiros de controle, e contribuiu

para a portabilidade dos softwares de automação. Esta norma está em constante evolução para refletir as últimas tendências na prática do desenvolvimento e em sua terceira edição inclui extensões para programação orientada a objetos, como discutido preliminarmente por Werner (2009).

A crítica à IEC 61131-3, Bauer *et al.* (2004) menciona suas ambiguidades semânticas e difícil integração em sistemas distribuídos e flexíveis de automação (VYATKIN *et al.*, 2005). Quando dois ou mais controladores se comunicam via rede, o modelo de controle programável centralizado dos CLPs podem causar resultados indesejados e despesas diversas na ocorrência de falha na comunicação, tais como parada do processo, o modelo de execução do programa verificado ciclicamente limita a reutilização dos componentes em caso de reconfiguração.

Katzke & Vogel-Heuser (2007) traz uma combinação das representações UML com o padrão IEC61131-3, para ele a apresentação visual da norma para sistemas de automação embarcados pode dificilmente transmitir estruturas de soluções complexas. A UML como uma apresentação alternativa do software convencional é confundida entre os engenheiros deste domínio, pois contém muitos diagramas e detalhes de notação, que são projetados para outros fins. A UML para automação de processos (UML-PA), que começou a ser introduzida por Ritala & Kuikka (2007), estendendo a UML v2, com vários domínios de conceitos específicos para modelagem de aplicações de automação de forma eficiente, oferece um subconjunto personalizado da UML, simplificando a sua estrutura gráfica, complementando ou limpando elementos de notações ambíguos. A UML-PA refere-se a especificações de comportamento previstos na IEC 61131-3 para oferecer noções familiarizadas no âmbito de novas estruturas.

Outros trabalhos foram realizados aproximando a IEC61131-3 da UML, Vogel-Heuser *et al.* (2013) faz um experimento de campo investigando a avaliação, o ensino e a aplicação de duas abordagens diferentes para o controle automático em CLPs, em particular, comparando a UML (Unified Modeling Language) com o paradigma processual clássico (IEC 61131-3). Participaram no treinamento e experimento 85 aprendizes de uma escola vocacional de engenharia de produção com especialização em mecatrônica.

Neste artigo, Vogel-Heuser *et al.* (2013) detalha os resultados do treinamento usando ambas as abordagens e as correlações encontradas entre o desempenho de modelagem e/ ou programação com as habilidades cognitivas de interesse, carga de trabalho, *expertise* e notas escolares. Em geral, os resultados mostraram que os alunos puderam ser treinados para realizar autênticas tarefas de programação dentro de um dia e meio, mesmo para iniciantes em programação.

A programação de diagramas de blocos funcionais (IEC 61131-3) pôde ser melhor prevista pela grade matemática, experiência de programação e demanda cognitiva. Para desempenho no diagrama de classe UML e modelagem de gráfico de estado, o desempenho em matemática tornou-se com papel ainda mais relevante. Isso explicou a maior variação no desempenho da modelagem no grupo UML do que no grupo 61131/ *Function Block Diagram*. Com relação a outros pontos, o artigo concluiu que habilidades especiais de

resolução de problemas e para o pensamento abstrato devem ser ensinadas a partir da modelagem UML.

4.2 Pesquisas experimentando a IEC61499

Sobre a IEC 61499, a norma apresenta uma arquitetura de referência que explora a familiaridade entre engenheiros de controle acostumados a uma forma bloco-diagramada de pensar. O principal artefato do *design*, bloco de funções (FB), foi estendido a partir da estrutura sub-rotina à abstração utilizada em processo semelhante na teoria dos sistemas de computação distribuída, onde representa uma atividade computacional independente com seu próprio conjunto de variáveis (contexto) e comunicação com outros processos através de mensagens. O padrão foi desenvolvido para permitir uma automação onde a inteligência é genuinamente descentralizada e incorporada em componentes de software, que podem ser distribuídos livremente entre dispositivos em rede. As áreas de aplicação promissoras incluem sistemas flexíveis de manuseio de materiais, em particular logística das bagagens nos aeroportos, automação de fabricação reconfigurável e flexível, redes inteligentes de distribuição de energia (*SmartGrid*), bem como a ampla gama de sistemas embarcados em rede.

A norma aborda diretamente a tendência da importância crescente do software no projeto de sistemas de automação, melhorando a portabilidade, configuração e interoperabilidade de sistemas de automação. Segundo Vyatkin (2013), a IEC61499 ainda está em estágios iniciais de sua adoção, com três ambientes de desenvolvimento integrado (IDE) comercialmente lançados no mercado.

Por outro lado, esta norma tem sido criticada por uma série de deficiências, tais como as ambiguidades semânticas, alto limiar de aprendizagem e harmonização insuficiente com as tecnologias existentes nos CLPs. No entanto, a maioria destas questões foram abordadas na segunda edição lançada em 2011.

Thramboulidis (2004) propõe a geração de FBs IEC61499 a partir de diagramas UML. Dubinin, Vyatkin & Pfeiffer (2005) descreveu a arquitetura UML-FB com ambas as formas de geração dos diagramas UML de delineamento em blocos de função e vice-versa. Em (THRAMBOULIDIS, 2005) propõe o conceito, baseado na IEC61499, da arquitetura mecatrônica integradas ao modelo de *design* de sistemas automatizados. Vyatkin *et al.* (2009) descreve o conceito de componentes mecatrônicos inteligentes (IMC) decorrente das ideias do *design* de sistemas de automação orientada a objetos (VYATKIN *et al.*, 2005) enriquecido com a simulação e outras funcionalidades, o IMC desenvolve o conceito objeto de automação.

O desenvolvimento de arquitetura de automação baseada em componentes da norma IEC 61499 estimulou o grande número de trabalhos de pesquisa. Por exemplo, Strasser, Rooker & Ebenhofer (2008) estuda o uso de linguagens de programação gráfica no domínio da automação para lidar com aplicações cada vez mais complexas, com o objetivo de reduzir

os custos de engenharia para a sua instalação e manutenção. Eles usam o conceito de *subapplications* para fornecer uma solução de custo eficaz para aplicações de automação industrial baseadas em componentes. O artigo apresenta uma abordagem de engenharia simples, mas eficaz para estruturar programas de controle distribuído em grande escala com base em um modelo de estrutura da planta hierárquica com IEC 61499 *subapplications*.

4.3 Pesquisas sobre o *design* de software e DBC

A engenharia de software baseada em componentes (CBSE), em um sentido mais amplo, é uma abordagem baseada no reúso para definir, implementar e compor componentes independentes acoplados em sistemas. Um componente de software individual é uma unidade de software que pode ser reutilizada sem quaisquer modificações em diferentes sistemas, semelhantes aos componentes de hardware como circuitos integrados.
Dois aspectos principais da automação baseada em componentes (CBA - *Component Based Automation*) estão sendo discutidos na literatura: questões de nível de implementação e de aplicação.
No lado da implementação, o principal objetivo é mascarar detalhes específicos da localização do componente, que pode ser um servidor remoto, conectado por uma variedade de redes e controlado por um sistema operacional diferente do cliente.
Outro aspecto da CBA está relacionado ao planejamento da modularidade no nível da aplicação. Projetar, desenvolver e manter componentes para reutilização é um processo muito complexo que coloca altos requisitos não apenas para a funcionalidade e flexibilidade dos componentes, mas também para a organização do desenvolvimento. A decisão, cuja granularidade é apropriada para um modelo reutilizável, depende do domínio considerado e da reutilização pretendida do modelo em questão. Como recomendação geral, os autores sugerem usar modelos bem documentados, hierárquicos e aninhados, que fornecem diferentes níveis de granularidade, dependendo da funcionalidade necessária.
Lee, Harrison & West (2004) descrevem a implementação de um sistema de controle industrial baseado em rede distribuída para manufatura. Os autores enfatizam as limitações dos modelos de componentes comuns, tais como a falta de garantias em tempo real e a alta complexidade. Eles propõem um modelo DBC próprio para automação distribuída, onde encapsula uma máquina de estado. O sistema de controle é composto por componentes autônomos e inteligentes que têm a capacidade de participar da automação sem a necessidade de um controlador mestre. É considerado pelos autores que o sistema de automação de manufatura que adota essa abordagem de controle pode ter maior agilidade, reutilização e redução nos custos de desenvolvimento.
A evolução dos requisitos para os produtos gera novos recursos para os componentes, tais como o planejamento do ciclo de vida, onde primeiro atinge sua estabilidade e posterior-

mente degenera em um recurso difícil de usar, de adaptar e manter.
Crnkovic & Larsson (2000) discutem ambos os aspectos de implementação e aplicação da CBA, referindo-se à experiência industrial. Foram identificados diferentes níveis de reutilização, juntamente com certos aspectos do desenvolvimento de componentes no contexto do controle de processos industriais, incluindo questões relacionadas com a normalização. Como ilustração é apresentada uma implementação bem-sucedida de um sistema de controle de processo industrial baseado em componentes. Os autores concluem que a crescente adoção de arquiteturas de componentes levanta novas questões, tais como: generalidade e eficiência de componentes, problemas de compatibilidade, demandas em ambiente de desenvolvimento e manutenção. Por exemplo, no que diz respeito à compatibilidade, os autores descrevem vários níveis, como a compatibilidade entre diferentes versões dos componentes, entre suas implementações para diferentes plataformas. Apesar da existência de vários modelos de implementação populares, muitas vezes eles não são igualmente suportados em todas as plataformas de hardware-software necessárias, o que levanta o problema da migração dos componentes.

4.3.1 SYSML(System Modeling Language)

A SysML (System Modeling Language) é uma ferramenta de modelagem com propósito geral para aplicação em engenharia de sistemas. Trata-se do resultado de um trabalho iniciado em 2001 pela INCOSE (Conselho Internacional de Engenharia de Sistema) e OMG (Object Management Group - Grupo de gerenciamento de projetos) visando customizar o UML para atender a esta finalidade, teve sua primeira especificação emitida pela OMG como SysML1.0 em 2007.

Foram definidos os seguintes diagramas, divididos em três grupos: estruturais, comportamentais, transversais:

- **Diagramas Estruturais:**

  Diagrama de definição de Bloco - Utilizado para definir as características dos blocos em termos estruturais e comportamentais, bem como suas relações;

  Diagrama de Bloco Interno - Usado para descrever a estrutura interna de um bloco em termos de como as suas partes estão interligadas;

  Diagrama paramétrico - Representa as restrições entre os elementos estruturais, as restrições são criadas a partir de sistemas de equações;

  Diagrama de Pacotes - Descreve as partes ou pacotes do sistema dividido em agrupamentos lógicos mostrando as dependências entre eles. Pode ser utilizado para ilustrar a arquitetura de um sistema;

- **Digramas Comportamentais:**

  Diagrama de Atividades - Usado para demonstrar através dos fluxos de entrada, saída e controle, o comportamento;

  Diagrama de Sequência - Utilizado para representar as interações entre objetos através de mensagens;

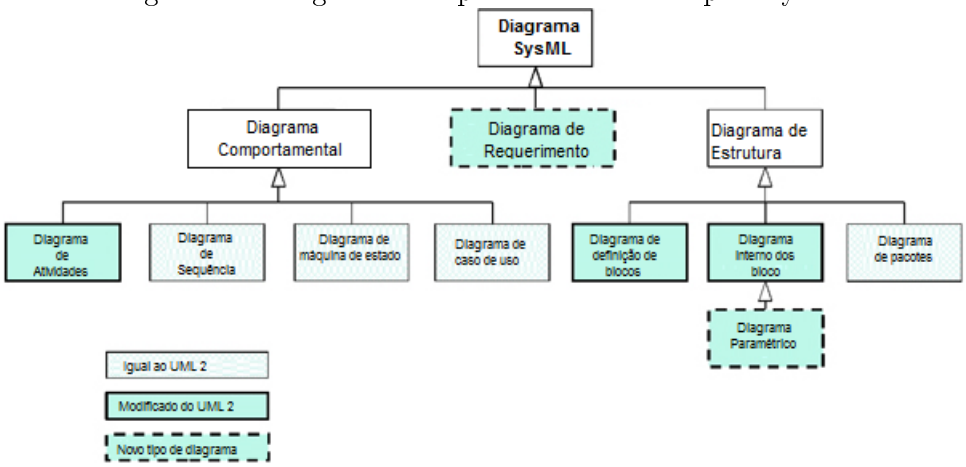
  Diagrama de Máquina de Estados - Tem o propósito de exibir o comportamento do sistema em sequências de estados que um componente ou uma experiência de interação apresenta em resposta a eventos;

  Diagrama de Caso de Uso - Usado para representar as funcionalidades propostas para o sistema. Casos de uso descrevem as interações entres usuários do sistema e os próprios sistemas. Casos de uso descrevem as interações entre usuários do sistema e o próprio sistema;

- **Diagrama Transversal:**

  Diagrama de Requisitos - Utilizado para representar hierarquias entre requisitos ou mostrar uma exigência individual e sua relação com outros elementos do modelo;

Figura 4.2: Diagramas adaptados e utilizados pela SysML



Fonte: Friedenthal, Moore & Steiner (2008)

A SysML oferece algumas vantagens, face ao uso da UML, incluindo a expansão da utilização em diferentes aplicações, outrora restrita e centralizada no software. Ela é capaz de modelar uma grande variedade de sistemas, que pode incluir hardware, software, informação, processos, pessoal e instalações, além disso a linguagem SysML é menor e mais simples, tendo um melhor aprendizado e aplicação.

Como citado neste referencial teórico anteriormente, o uso de linguagens de modelagens, tais como o SysML e UML, demonstraram uma vantagem pedagógica na compreensão

e representação do funcionamento dos sistemas. Por este mesmo motivo, no capitulo 6 foi necessário utilizar os diagramas SysML para facilitar a divulgação da modelagem proposta nesta dissertação, pois a representação diagramada apenas pelo *function block* não seria suficiente para descrever o comportamento de cada componente, suas estruturas de controle, algorítimos e suas interfaces organizadas por estruturas de dados definida pelo usuário (UDT) em multiníveis.

Capítulo Cinco

Modelagem do sistema

Este capitulo apresenta os modelos e procedimentos propostos para o desenvolvimento dos componentes lógicos, explorando conceitos e métodos citados nos capítulos anteriores.
A estrutura básica das etapas implementadas, necessárias para a definição dos componentes envolve as seguintes fases:

1. Análise dos requisitos
   - Identificação dos componentes
   - Especificação dos requisitos/ Diagramas de requerimento
     - Requisitos em situações normais
     - Requisitos em condições de falha
   - Interação dos componentes e interfaces
2. Modelagem
3. Análise dos modelos

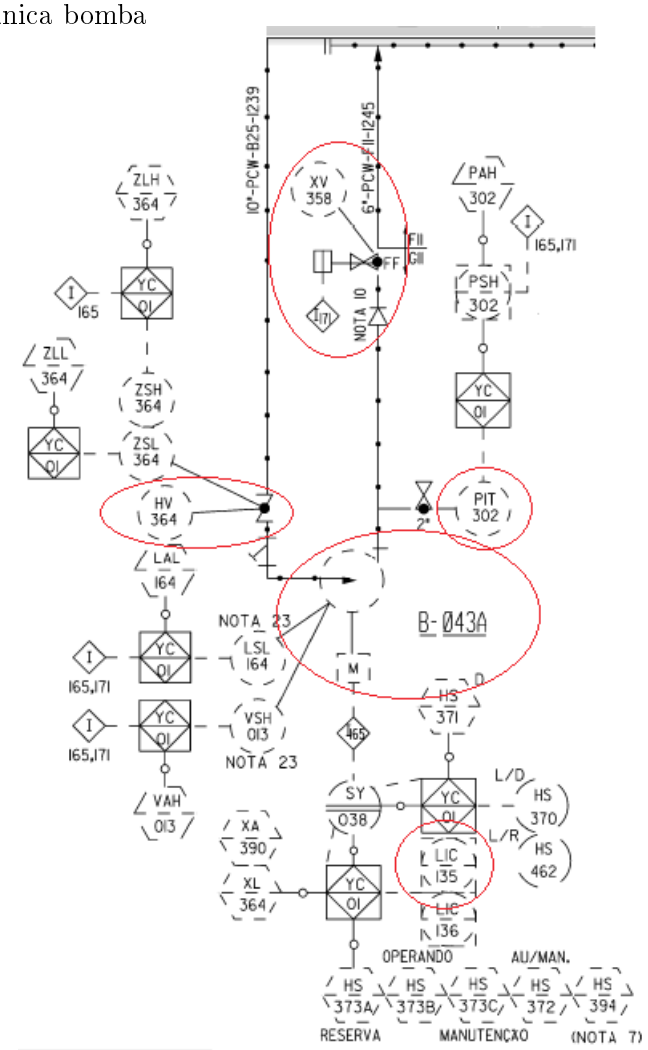
5.1 Análise dos requisitos

O objetivo nesta etapa é identificar quais os tipos de componentes serão criados, especificando quais os requisitos para representar a automação em uma estação de injeção de água em poços. Vale ressaltar que são características necessárias a reusabilidade e independência entre as partes, de modo que permita uma distribuição no sistema sem exigência de um controlador mestre.

5.1.1 Identificação dos componentes

Para identificação foram utilizados os diagramas P&ID como fonte de requisitos, conforme proposto por Thramboulidis & Frey (2011). Portanto apenas na figura 5.1 é possível extrair 05 componentes lógicos distintos, destacados através de um elipse em vermelho.

Figura 5.1: Diagrama P&ID utilizado para representar uma estação de injeção de água - Recorte do trecho de uma única bomba



Fonte: Adaptado pelo autor

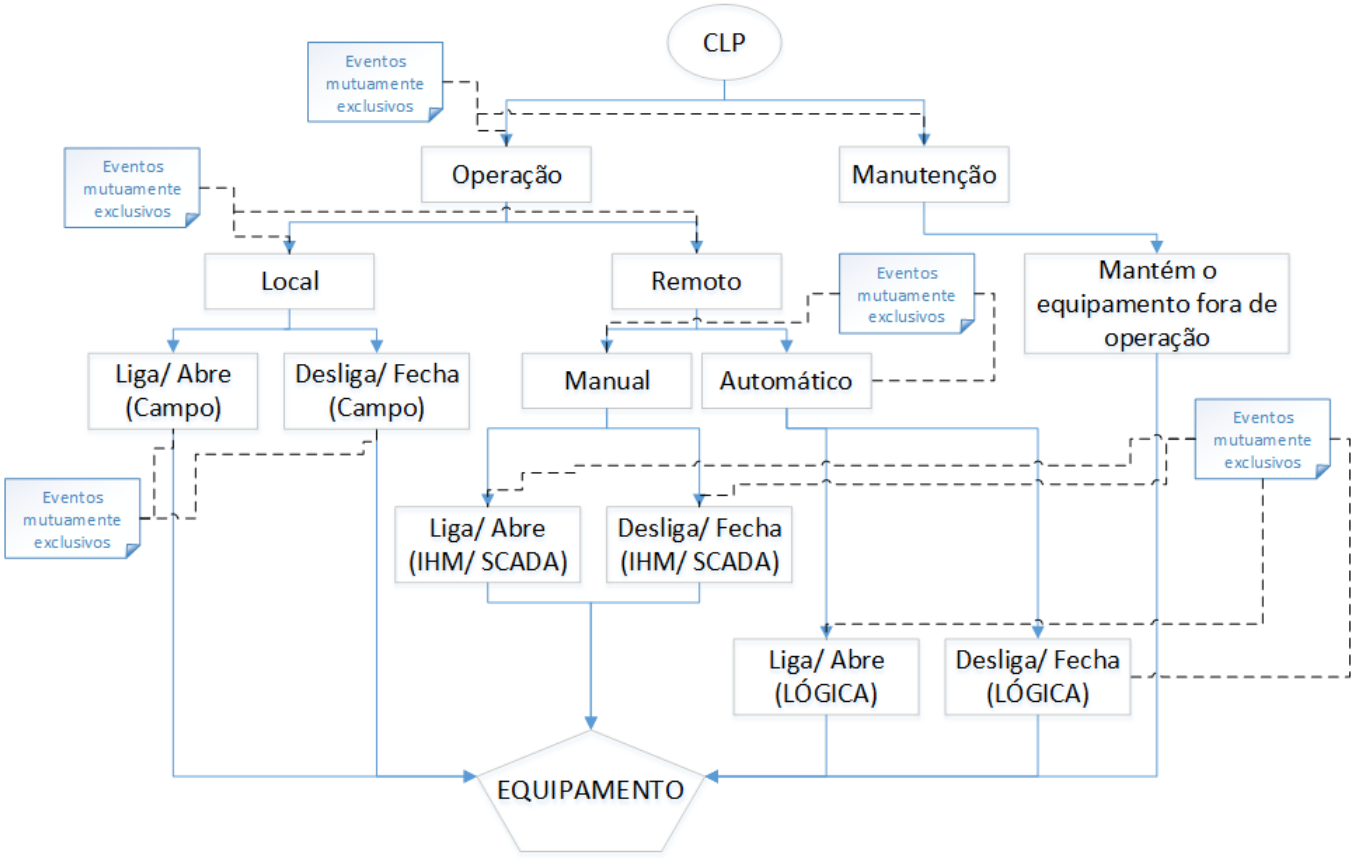
Exceto o LIC-135, que representa uma malha de controle PID, cada um dos componentes destacados em vermelho estão relacionados a um objeto fisicamente instalado. Todos estes possuem interfaces com um CLP (Neste exemplo indicado como YC-01), deste modo é possível extrair os seguintes itens da figura 5.1:

- Componente transmissor analógico (PIT-302)
- Componente válvula de bloqueio manual (HV-364)
- Componente válvula pneumática de bloqueio (XV-358)
- Componente bomba com inversor de frequência (B-043A)
- Componente malha de controle PID (LIC-135)

5.1.2 *Interação dos componentes e suas interfaces*

Na modelagem proposta para os componentes deste trabalho, as interfaces foram sempre particionadas e inspiradas pelo padrão IEC 61499, no que tange ao tratamento utilizado para elaboração dos blocos de funções. Sendo assim, todas as informações foram divididas em eventos de entrada/ saída e dados de entrada/ saída. Os eventos de entrada foram utilizados para agrupar comandos enviados por um operador ou qualquer outra intercorrência capaz de modificar o comportamento/ estado do componente, seja local ou remotamente; os eventos de saída estão sendo utilizados para representar o estado ativado no componente. Este conceito foi adotado para permitir, além de uma boa reusabilidade, uma flexibilização voltada à distribuição dos dispositivos de controle na indústria de petróleo e gás.
Nos próximos subcapítulos serão apresentados os diagramas SysML para definição de blocos (BDD) representando as interfaces de cada componente modelado. Alguns termos e definições das variáveis incluídas nestes BDDs são semelhantes, principalmente no que se refere aos eventos de entrada e saída, contudo algumas considerações serão apresentadas. A figura 5.2 pretende resumir as similaridades na forma de operação dos componentes propostos para controle do processo.

Figura 5.2: Modo de operação dos equipamentos na automação de processo



Fonte: Do autor

Os eventos mutuamente exclusivos indicam que, para uma dupla de estados, cada um estará ativado de forma única e isolada em seu par; por exemplo: quando o modo "manutenção" estiver selecionado, logicamente o estado "operação" nunca estará simultaneamente habilitado, e vice-versa. Isto indica uma relação de dependência por selecionamento entre cada um desses pares. Sobre os estados e termos apresentados na figura 5.2, compreende-se:

- **Liga/ Abre (Campo):** Como um evento de entrada que será aceito apenas quando o componente estiver com o modo "local" habilitado. Além disto, este comando será enviado pelo operador somente através das chaves/ botoeiras instaladas em campo, individualmente, para ligar cada equipamento ou abrir cada válvula;
- **Desliga/ Fecha (Campo):** Um evento de entrada que será aceito apenas quando o componente estiver com o modo "local" habilitado. Além disto, este comando será enviado pelo operador somente através das chaves/ botoeiras instaladas em campo, individualmente, para desligar cada equipamento ou fechar cada válvula;
- **Liga/ Abre (IHM/ SCADA):** Um evento de entrada que será aceito apenas quando o componente estiver com o modo "remoto" e "manual" simultaneamente habilitados. Além disto, este comando será enviado pelo operador somente através de chave lógica - via sistema SCADA (Supervisory Control and Data Acquisition - Supervisão controle e aquisição de dados) - para ligar cada equipamento ou abrir cada válvula;
- **Desliga/ Fecha (IHM/ SCADA):** Um evento de entrada que será aceito apenas quando o componente estiver com o modo "remoto" e "manual" simultaneamente habilitados. Além disto, este comando será enviado pelo operador somente através de chave lógica - via sistema SCADA - para desligar cada equipamento ou fechar cada válvula;
- **Local:** Este estado será selecionado a partir de uma chave em campo. Com este modo habilitado, será permitido acionar o equipamento apenas através dos eventos de entrada "Liga/ Desliga" ou "Abre/ Fecha" - Via chaves ou boteiras instaladas em campo - sempre respeitando os intertravamentos de segurança;
- **Remoto:** Este estado será selecionado a partir de uma chave em campo. Com este modo habilitado, será permitido atuar no componente manualmente via sistema SCADA, através das chaves lógicas: "Desliga", "liga", "abre" ou "Fecha"; ou automaticamente - obedecendo condições pré-estabelecidas na lógica de controle - sempre respeitando os intertravamentos de segurança;
- **Manual:** Este estado será selecionado através de um comando via sistema SCADA e terá validade apenas se o modo "Remoto" estiver conjuntamente selecionado. Com

o estado "Manual" habilitado, a intervenção no equipamento será realizada através das chaves "Liga/ Desliga" ou "Abre/ Fecha", novamente via sistema SCADA, respeitando os intertravamentos de segurança do componente;

- **Automático:** Este estado será selecionado através de um comando via sistema SCADA e terá validade apenas se o modo "Remoto" estiver conjuntamente selecionado. Com o estado "Automático" habilitado, a operação do equipamento respeitará apenas as condições de controle e segurança pré-estabelecidas na lógica do componente.
- **Operação:** Esta condição será selecionada pelo operador através do sistema de supervisão. Sendo esta escolhida, o equipamento estará disponível para operar em "Local" ou "Remoto", dependendo da opção desejada.
- **Manutenção:** Esta condição será selecionada pelo operador através do sistema de supervisão, ela estará disponível para todos os equipamentos. Sendo escolhido este estado, o componente ficará impedido de operar pelo CLP para evitar danos físicos aos técnicos de manutenção que estiverem intervindo no equipamento.

5.1.3 Especificação dos requisitos/ Diagramas de requerimento

Apesar da figura 5.1 possuir um recorte capaz de localizar componentes em potencial, a mesma ainda não reproduz com detalhamento os requisitos exigidos para cada um destes. Isto acontece porque normalmente estas informações são ocultadas para não sobrecarregar o desenho ou são apresentadas em documentos complementares: tais como descritivos de lógica de automação, diagramas de malha, entre outros.
Alguns P&ID discriminam estas funcionalidades em uma única parte do desenho, agrupando equipamentos do mesmo tipo, esta aglomeração é normalmente denominada como detalhes típicos de projeto.
As seções a seguir detalharão cada típico, apresentando as particularidades, requisitos em situações normais e em condições de falha, bem como os diagramas de requerimentos.

5.2 Modelagem do componente transmissor analógico

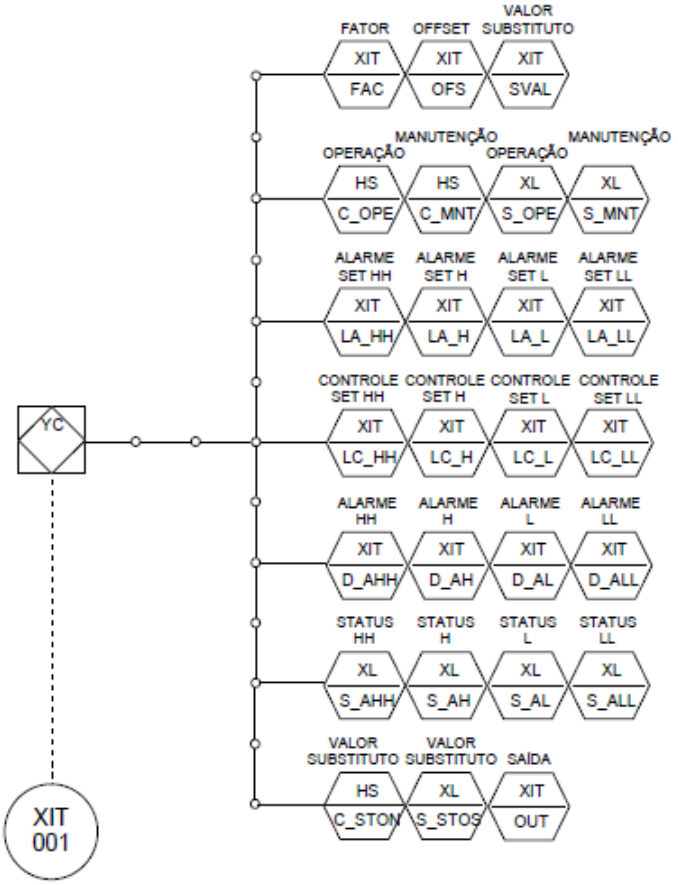
5.2.1 Requisitos do componente transmissor analógico

Os transmissores analógicos comunicam com os CLPs através das entradas analógicas ou rede - Consultar a literatura para maiores detalhes (RIBEIRO, 1999) - estes são capazes de realizar medições e leituras em diferentes variáveis do processo, tais como: pressão

temperatura, nível, vazão, entre outras, e convertê-las em sinais padronizados e entendíveis pelo controlador.
Na literatura existem outras terminologias, como transdutor, sensor e instrumento de medição. Neste trabalho, o termo transmissor analógico foi o escolhido para nomear este componente, dada a necessidade de que o seu sinal de saída esteja sempre condicionado a um padrão elétrico de transmissão.

No que diz respeito as interfaces do transmissor analógico com o controlador lógico e/ou sistema SCADA, a figura 5.3 representa os detalhes normalmente solicitados, quando utilizados na automação de processos. A abreviação "XIT" simboliza um transmissor e indicador de diferentes variáveis, segundo a normatização pela ISA 5.1. Com isto, é possível compreender que os requisitos explicitados e exigidos no diagrama típico de um "transmissor analógico" descreve uma aplicação na qual será reutilizada pelo CLP para conversão e escalonamento, em diferentes tipos de grandezas físicas e químicas.

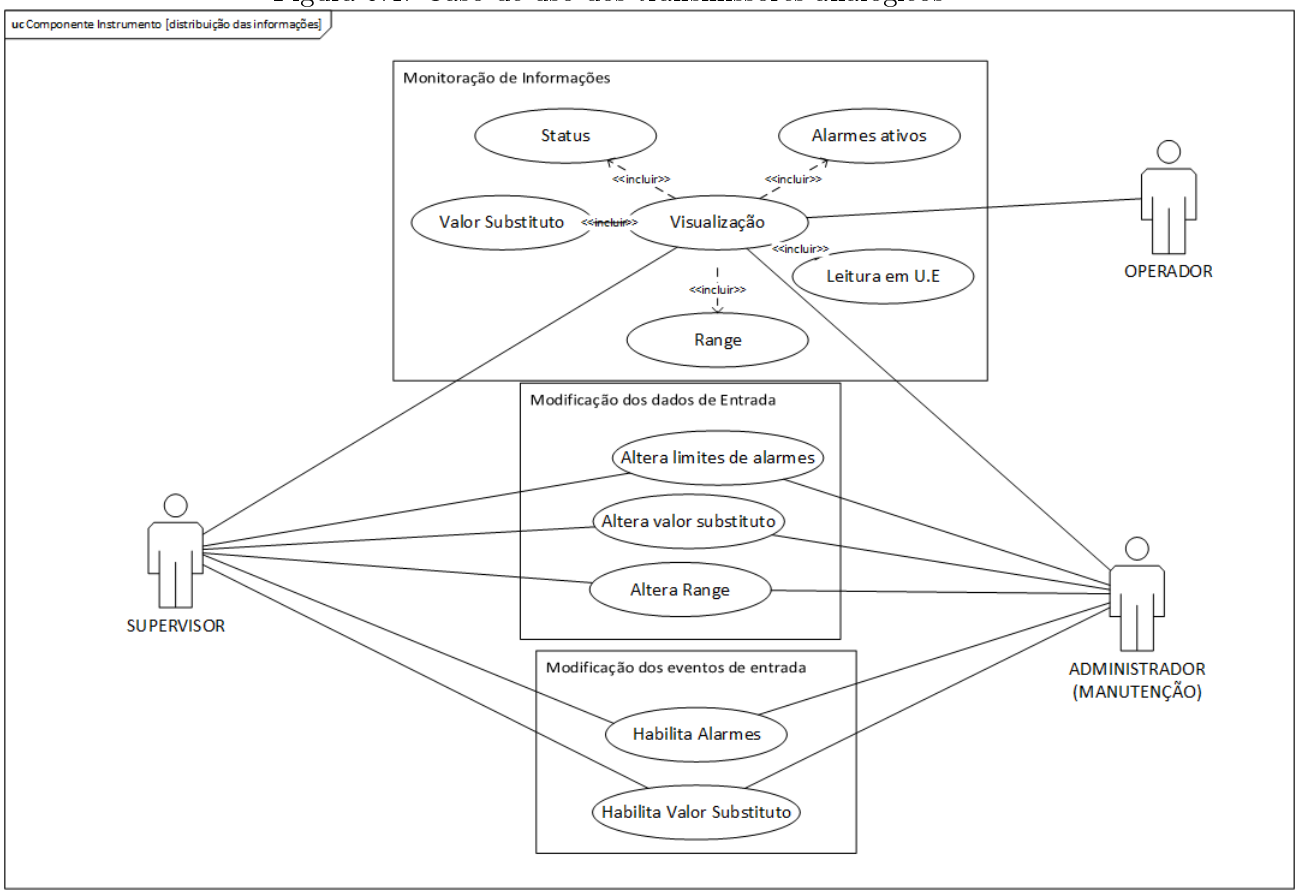
Figura 5.3: Diagrama típico de transmissor analógico



Fonte: Adaptado pelo autor

Através da figura 5.3 é possível observar que este componente comportará diferentes tipos de dados para escrita e leitura (Incluindo, por exemplo: alterações nos parâmetros limites para alarmes, escala do instrumento, leitura do valor convertido em unidade de engenharia, dentre outros), além dos eventos de entrada e saída (Incluindo, por exemplo: habilitação dos alarmes, do valor substituto, seleção dos modos de operação, dentre outros).
Entende-se como eventos de entrada a solicitação para que cada modo de funcionamento seja habilitado no componente, eles possuem uma atuação direta nos eventos de saída. Por sua vez, os eventos de saída sinalizam quais os estados estão realmente ativos na operação de cada componente, internamente e externamente. A figura 5.4 descreve, utilizando o diagrama de casos de uso, as principais funcionalidades e as interações com os usuários do mesmo sistema. Neste diagrama ainda não são aprofundados os detalhes técnicos do componente.

Figura 5.4: Caso de uso dos transmissores analógicos



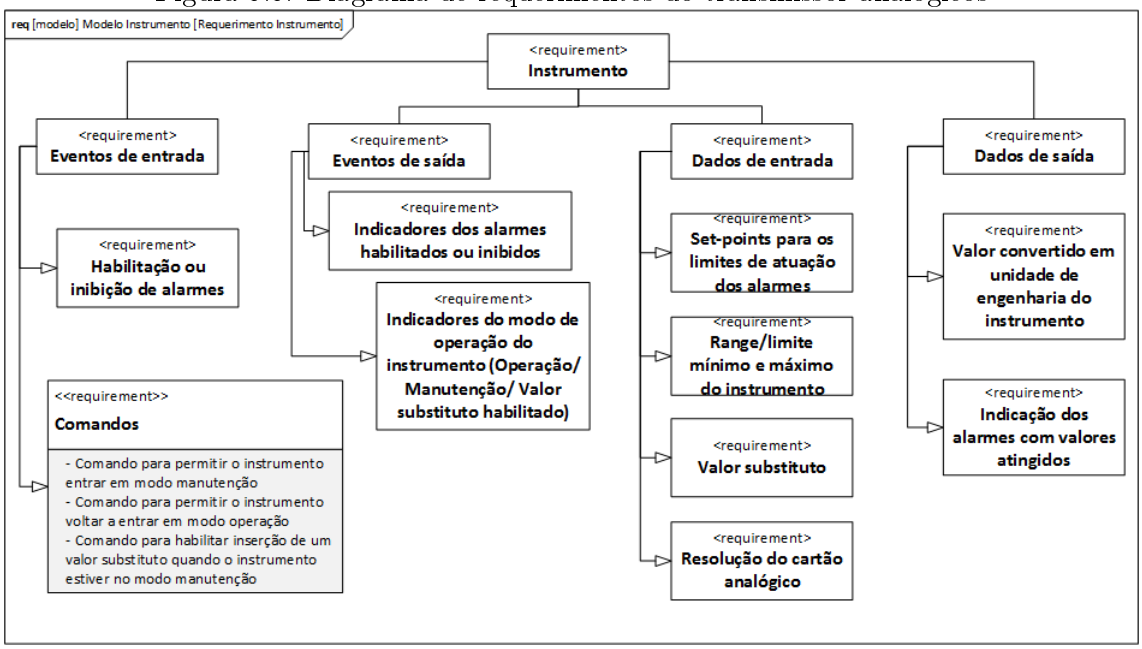
Fonte: Do autor

Para garantir uma integridade, na modelagem do componente “Transmissor analógico”, foi necessário prever requisitos para condições de falha. Sendo apresentados da seguinte forma:

- **Falha interna:** Os transmissores analógicos normalmente comunicam com os CLPs através de um padrão conhecido de corrente, variando entre valores de 4 à 20mA. Quando estes medidores apresentam algum tipo de falha em seu funcionamento, assumem um comportamento previsível definido e passam a indicar um valor fixo de corrente, normalmente acima do limite superior da sua escala, a saber 20,5mA. A identificação desta ocorrência permitirá um tratamento adequado na lógica de controle, conduzindo o processo para uma condição de segurança;
- **Falha de comunicação:** Se o CLP "lê" os transmissores analógicos por meio do padrão elétrico de corrente estabelecido em 4 à 20mA, eventualmente poderá perder a comunicação deste sinal. Isto ocorrerá por meio de uma ruptura indesejada do seu cabo elétrico ou mesmo por atuação do fusível de proteção do instrumento. A identificação deste tipo de falha pelo CLP poderá ser sinalizada todas as vezes que a leitura analógica assumir valores menores que o limite inferior da escala de corrente, normalmente abaixo de 3,5mA;
- **Falha de comunicação em rede:** Se a leitura dos transmissores analógicos forem efetuadas por meio de redes de comunicação, eventualmente poderá perder esta informação por diferentes motivos. Esta condição de falha poderá ser identificada através de *time out* do sinal ou por meio de outros diagnósticos disponibilizados nos canais de comunicação, distintos para cada protocolo de comunicação.

Por fim, a figura 5.5 apresenta o diagrama de requerimentos para o transmissor analógico.

Figura 5.5: Diagrama de requerimentos do transmissor analógicos



Fonte: Do autor

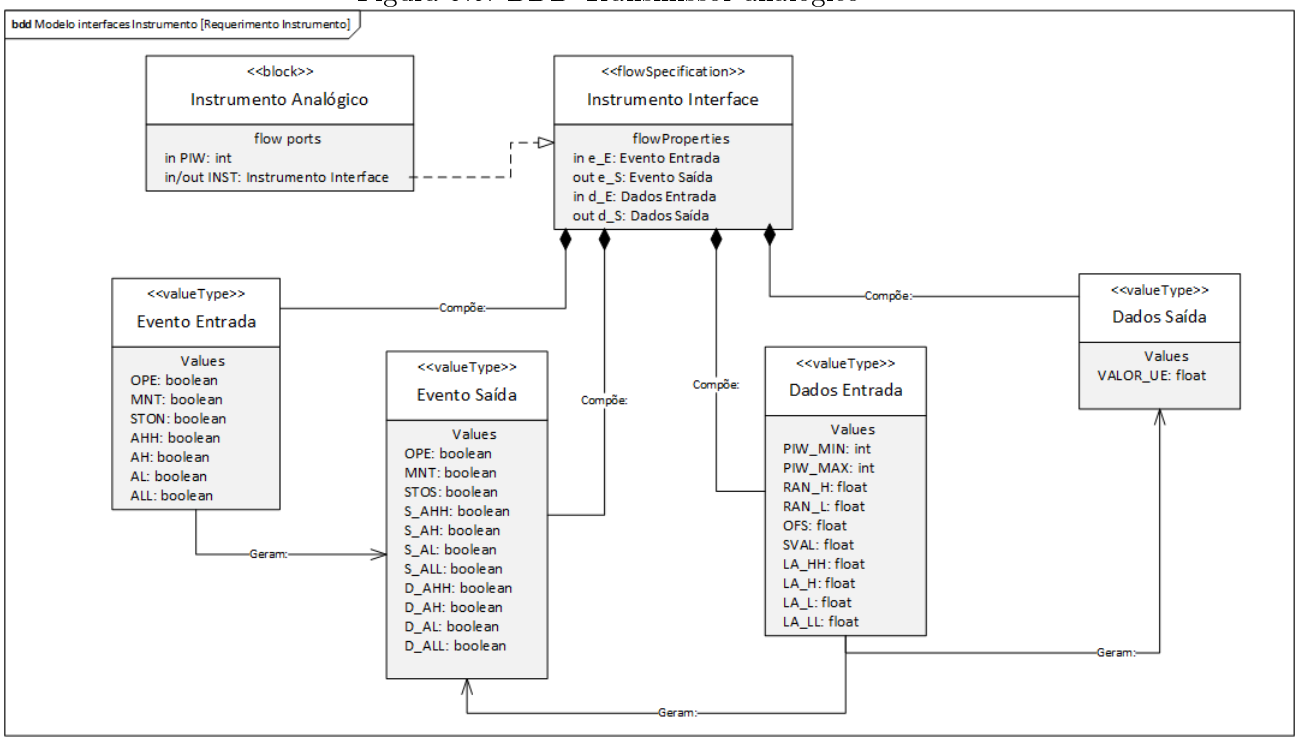
Por meio do diagrama representado na figura 5.5 é possível observar a transição que

ocorrerá entre os requisitos - ora solicitados pelos diagramas PI&D - e a organização das variáveis que serão utilizadas na interface do componente. Apesar de não considerado, ainda neste momento, os tipos de dados e suas memórias no programa, é possível notar mais uma vez o agrupamento que ocorrerá entre os dados e eventos de entrada e saída, relembrando as exigências mencionadas na IEC 61499.

5.2.2 BDD Transmissor analógico

A figura 5.6, representa o diagrama de definição de bloco (BDD) do componente transmissor analógico.

Figura 5.6: BDD Transmissor analógico



Fonte: Do autor

A seguir os eventos de entrada e saída do BDD transmissor analógico serão explicados, contemplando as seguintes informações:

- **OPE:** Ao ser ativado, o componente passará a obedecer o critério de funcionamento explicado anteriormente em 5.1.2.
- **MNT:** Ao ser ativado, o componente passará a obedecer o critério de funcionamento explicado anteriormente em 5.1.2.

- **STON:** Este estado somente poderá ser ativado quando o modo "MNT" estiver habilitado, nesta condição o valor configurado pelo dado de entrada "SVAL" (Valor Substituto) será enviado para saída de indicação do instrumento "VALOR_UE". Este procedimento permite ao operador simular as variáveis de processo.
- **S_AHH/ S_AH/ S_AL/ S_ALL:** Estes estados habilitam a comparação entre os alarmes "muito alto", "alto", "baixo", "muito baixo", respectivamente na lógica do componente transmissor analógico. Ao serem individualmente desabilitados, o CLP passará a desconsiderar essas comparações entre "VALOR_UE" com os limites fornecidos para cada alarme. Esta função possibilita a customização dos alarmes que são realmente necessários para bloco de função do sistema de automação.
- **D_AHH/ D_AH/ D_AL/ D_ALL:** Estes eventos de saída simbolizam os diagnósticos dos alarmes ocorridos, ou seja, são os resultados das operações de comparação entre o valor convertido em unidade de engenharia "VALOR_UE" com os limites para cada alarme LA_HH, LA_H, LA_L, LA_LL, respectivamente.

A seguir os dados de entrada e saída do BDD transmissor analógico serão explicados:

- **PIW_MIN:** Representa o valor mínimo, em decimal, da resolução do canal de comunicação analógica no CLP, este valor será considerado pela função escalonamento. Por exemplo, se o canal possuir resolução de 12 bits, deverá ser incluído o valor 0 neste parâmetro.
- **PIW_MAX:** Representa o valor máximo, em decimal, da resolução do canal de comunicação analógica no CLP, este valor será considerado pela função escalonamento. Por exemplo, se o canal possuir resolução de 12 bits, deverá ser incluído o valor 4095 neste parâmetro.
- **RAN_L:** Representa o valor minimo em unidade de engenharia para o range ou alcance do instrumento (O formato do dado deverá ser em ponto flutuante), este valor será considerado pela função escalonamento. Por exemplo, para transmissão da informação de um sensor de pressão ao CLP, onde a sua faixa de medição oscila entre -10,0 PSI à 30,0 PSI; deverá ser incluído o valor -10,0 neste parâmetro.
- **RAN_H:** Representa o valor máximo em unidade de engenharia para o range ou alcance do instrumento (O formato do dado deverá ser em ponto flutuante), este valor será considerado pela função escalonamento. Por exemplo, para transmissão da informação de um sensor de pressão ao CLP, onde a sua faixa de medição oscila entre -10,0 PSI à 30,0 PSI, deverá ser incluído o valor 30,0 neste parâmetro.
- **OFS:** Representa o valor *offset* do instrumento em unidade de engenharia (O formato do dado deverá ser em ponto flutuante), por fim este valor será sempre somado ao dado de saída "VALOR_UE".

- **SVAL:** Representa o valor substituto, este valor será utilizado para simulação das indicações do instrumento e deverá variar entre os valores configurados no Range; podendo inclusive acionar os alarmes dependendo dos valores configurados em seus limites.
- **LA_HH/ LA_H/ LA_L/ LA_LL:** São valores utilizados para definir o ponto de acionamento dos alarmes "muito alto", "alto", "baixo", "muito baixo" em cada instrumento. Ou seja, se o estado do evento de saída para cada alarme estiver habilitado, todas as vezes que o "VALOR_UE" ultrapassar os limites configurados nos parâmetros, o respectivo alarme do evento de saída será acionado.
- **VALOR_UE:** Representa o dado de saída com o valor convertido em unidade de engenharia do transmissor analógico.

5.2.3 Descrição do comportamento interno - Componente transmissor analógico

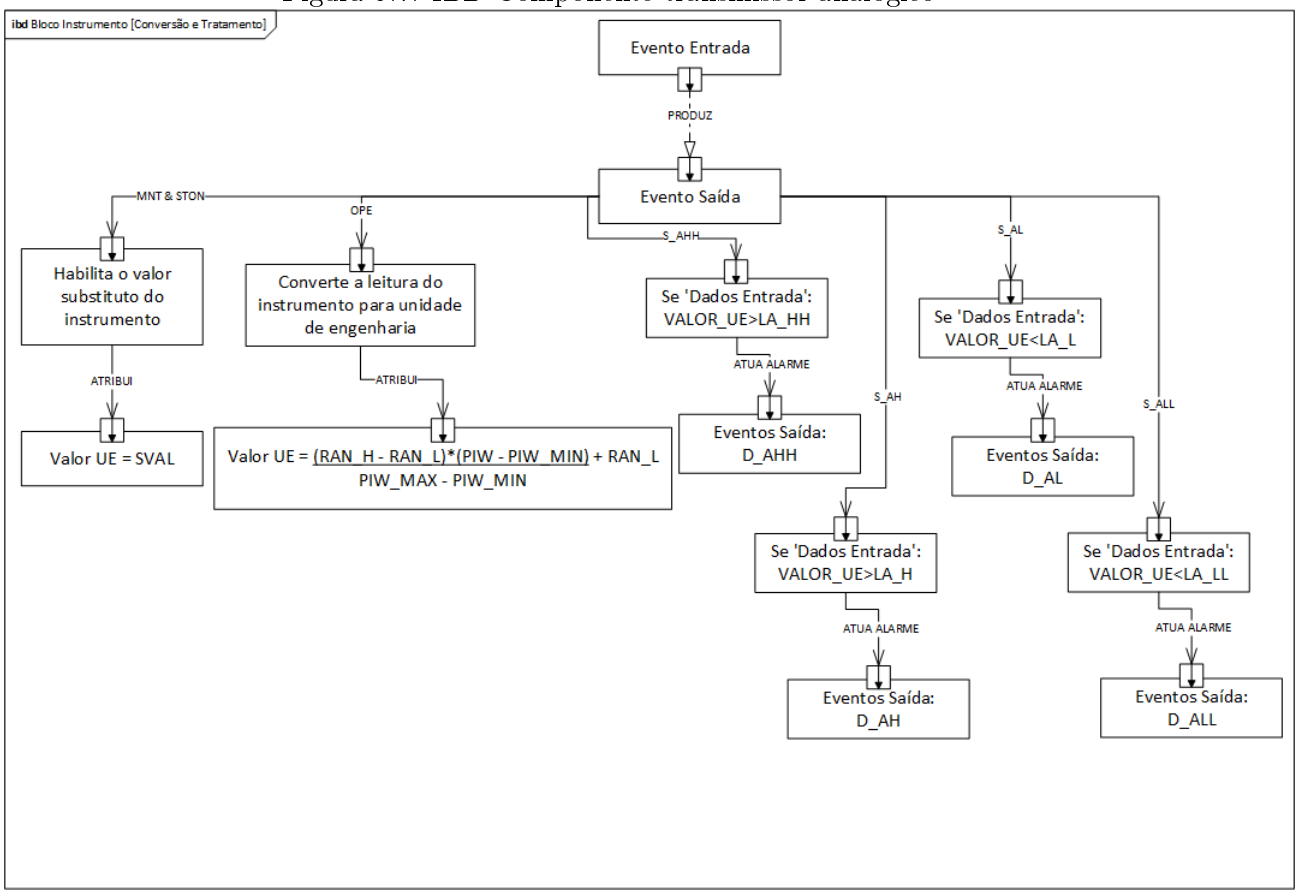
No programa de CLP, este componente tem como principal finalidade a conversão do sinal A/D proveniente do canal periférico de comunicação analógica para a unidade de engenharia da variável medida. Isto pode ser realizado através da equação de escalonamento:

$$VALOR\_UE = \frac{(RAN\_H - RAN\_L) * (PIW - PIW\_MIN)}{(PIW\_MAX - PIW\_MIN)} + RAN\_L$$

.

Outras funcionalidades importantes para este componente são os tratamentos de cada alarme e do modo de operação do transmissor. A figura 5.7 apresenta como são correspondidos cada evento de saída e suas interfaces, através do diagrama de bloco interno (IBD).

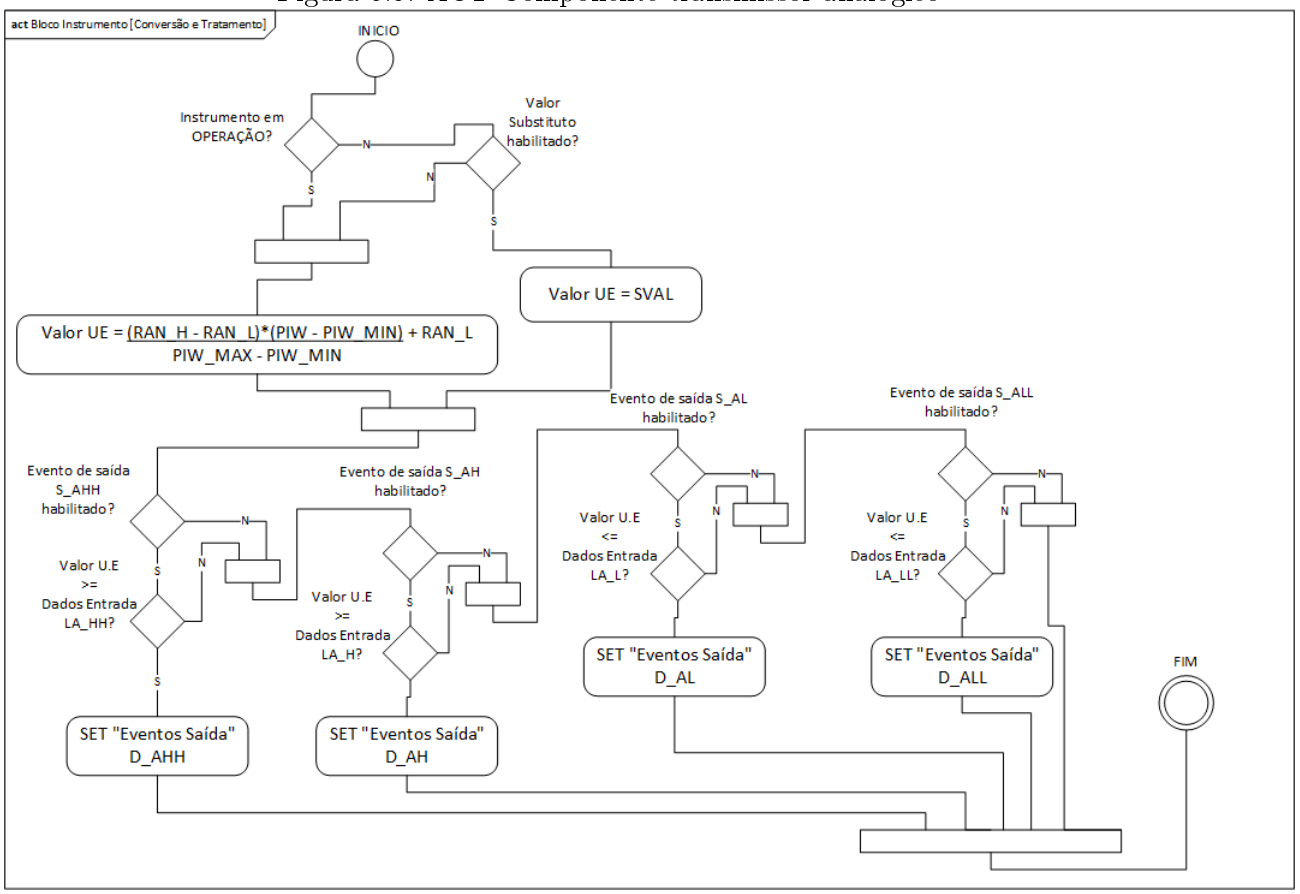
Figura 5.7: IBD Componente transmissor analógico



Fonte: Do autor

A figura 5.8 é denominada diagrama de atividades (ACT) do componente instrumento, ela faz uma representação gráfica das etapas do processo de funcionamento e ajuda a modelar os fluxos de trabalho através de uma sequência controlada de ações.

Figura 5.8: ACT Componente transmissor analógico



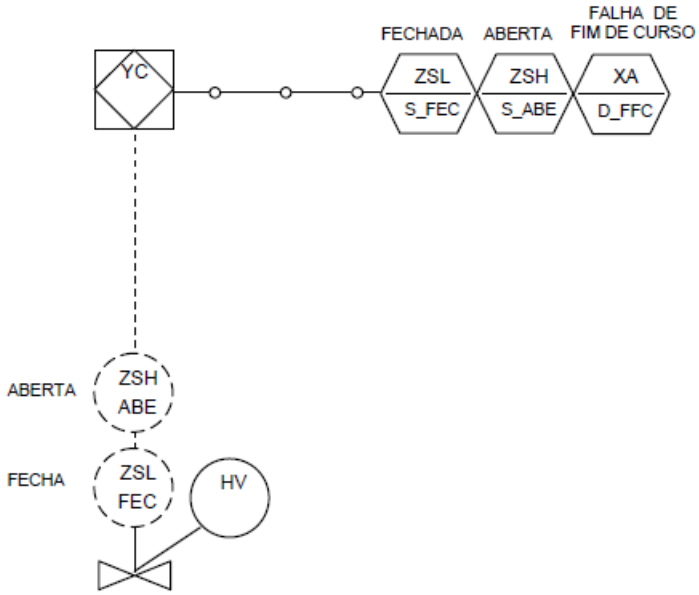
Fonte: Do autor

5.3 *Modelagem do componente válvula de bloqueio manual*

5.3.1 *Requisitos do Componente válvula de bloqueio manual*

A válvula de bloqueio manual é um componente lógico simples, por possuir poucas interfaces com o CLP, consequentemente possui poucos requisitos funcionais. Estas válvulas são manipuladas fisicamente em campo por operadores de processo, assumindo apenas duas posições: completamente fechada ou aberta, estas informações são lidas através de sensores e transmitidas ao CLP por meio dos canais de entrada digital, consultar a literatura para maiores detalhes (RIBEIRO, 1999). A figura 5.9 representa o diagrama típico P&ID, com as exigências normalmente solicitadas para este componente.

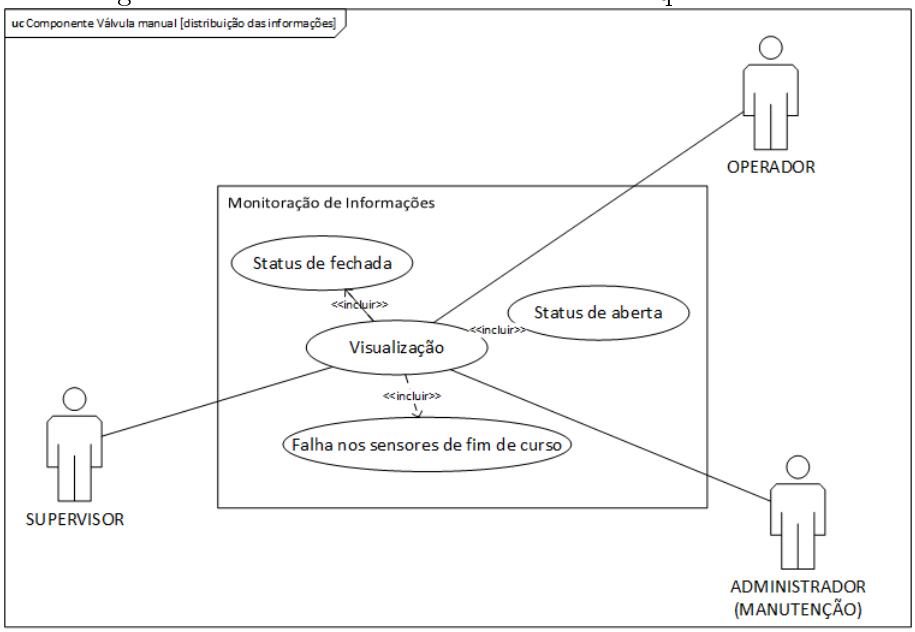
Figura 5.9: Diagrama típico de válvula de bloqueio manual



Fonte: Adaptado pelo autor

A figura 5.10 representa o diagrama caso de uso, especificando os requisitos para este componente.

Figura 5.10: Caso de uso das válvulas de bloqueio manual



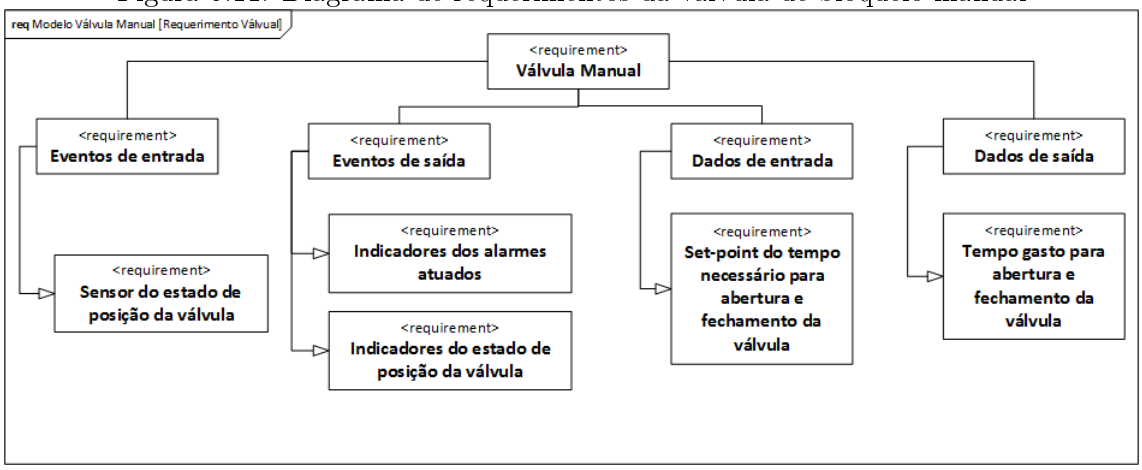
Fonte: Do autor

Como representado nos diagramas acima 5.9 e 5.10, o seguinte requisito é previsto para condição de falha deste componente:

- **Falha no fim de curso:** Ocorrerá quando os sensores de posição aberta ou fechada da válvula estiverem simultaneamente acionados ou desativados em conjunto após a finalização do tempo pré-definido pelo usuário.

Por fim, a figura 5.11 apresenta o diagrama de requerimentos para a válvula de bloqueio manual.

Figura 5.11: Diagrama de requerimentos da válvula de bloqueio manual

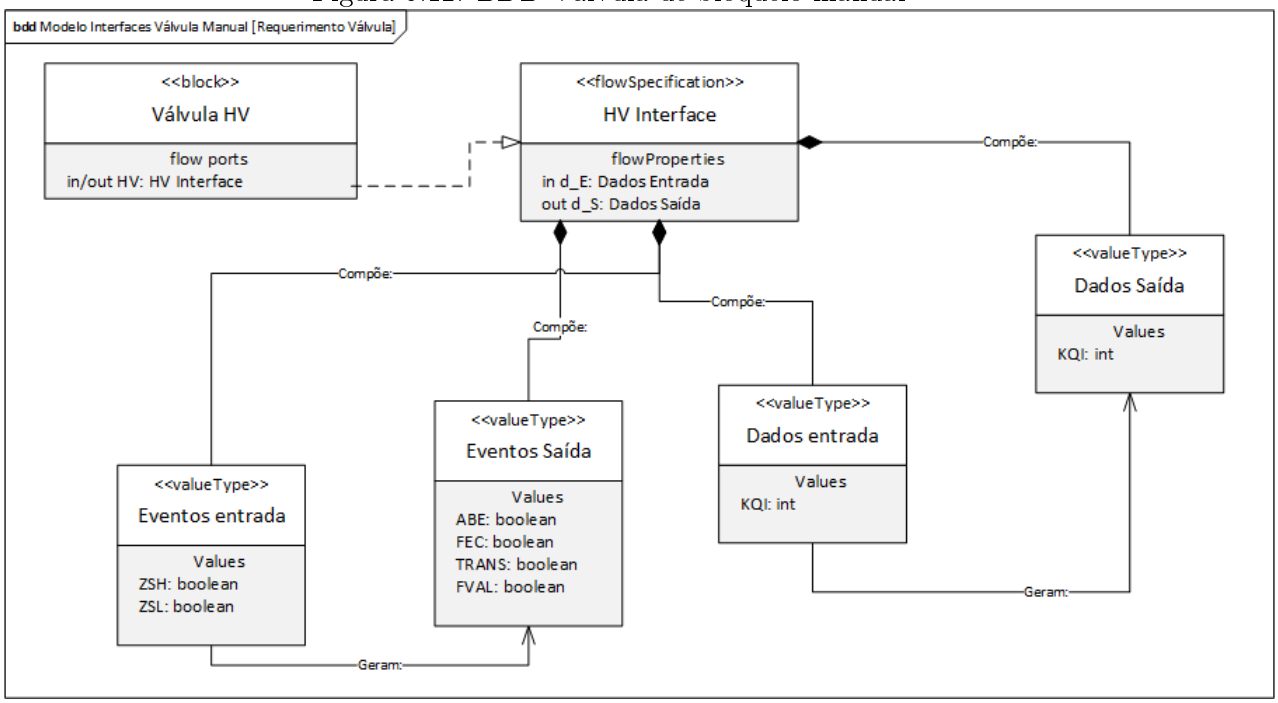


Fonte: Do autor

5.3.2 BDD Válvula de bloqueio manual

A figura 5.12, representa o diagrama de definição de bloco (BDD) do componente válvula de bloqueio manual.

Figura 5.12: BDD Válvula de bloqueio manual



Fonte: Do autor

A seguir os eventos de entrada e saída do BDD válvula de bloqueio manual serão explicados:

- **ZSL:** Sensor de posição fechada.
- **ZSL:** Sensor de posição aberta.
- **ABE:** Indicação do estado de válvula aberta.
- **FEC:** Indicação do estado de válvula fechada.
- **TRANS:** Indicação do estado de transição da válvula "abrindo" ou "fechando".

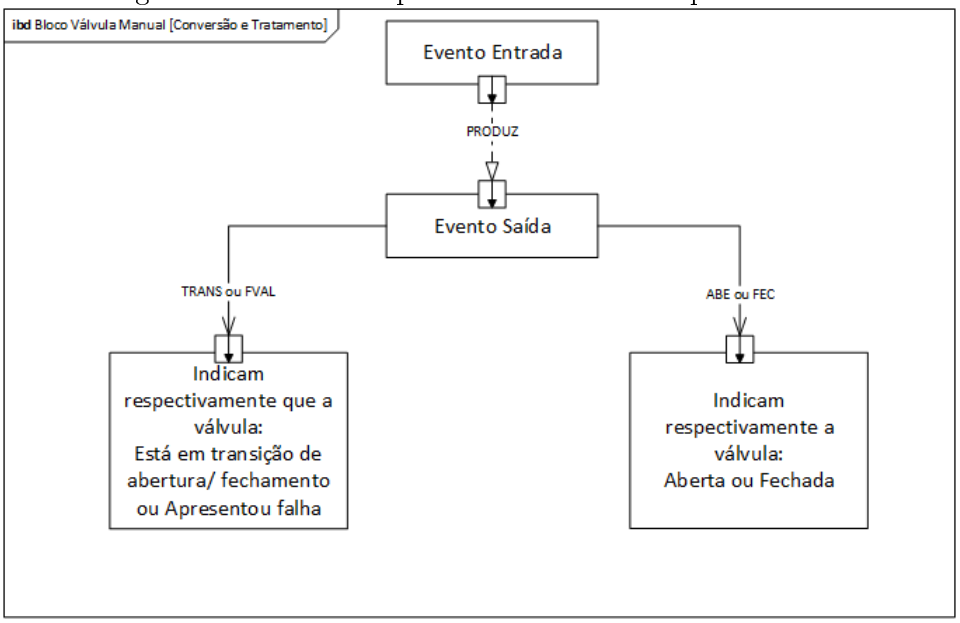
A seguir os dados de entrada e saída do BDD válvula de bloqueio manual serão explicados:

- **KQI:** Representa a quantidade de tempo em segundos necessários para manobrar o volante da válvula. Caso este tempo informado exceda, será acionado o alarme "FVAL".
- **FVAL:** Indica a presença do alarme: "falha de posição na válvula". A condição para que este ocorra é que os sensores ZSL e ZSH fiquem simultaneamente desativados ou atuados, por um tempo maior do que o definido em "KQI".

5.3.3 *Descrição do comportamento interno - Componente válvula de bloqueio manual*

No programa de CLP, este componente tem como principal finalidade o tratamento dos sinais provenientes dos canais periféricos de comunicação digital, referente ao estado de posição aberta ou fechada destas válvulas. Basicamente, além desta funcionalidade, é acrescido apenas o alarme por fim da contagem de tempo da transição entre os estados de posição.
A figura 5.13 apresenta como são correspondidos cada evento de saída e suas interfaces, através do diagrama de bloco interno (IBD).

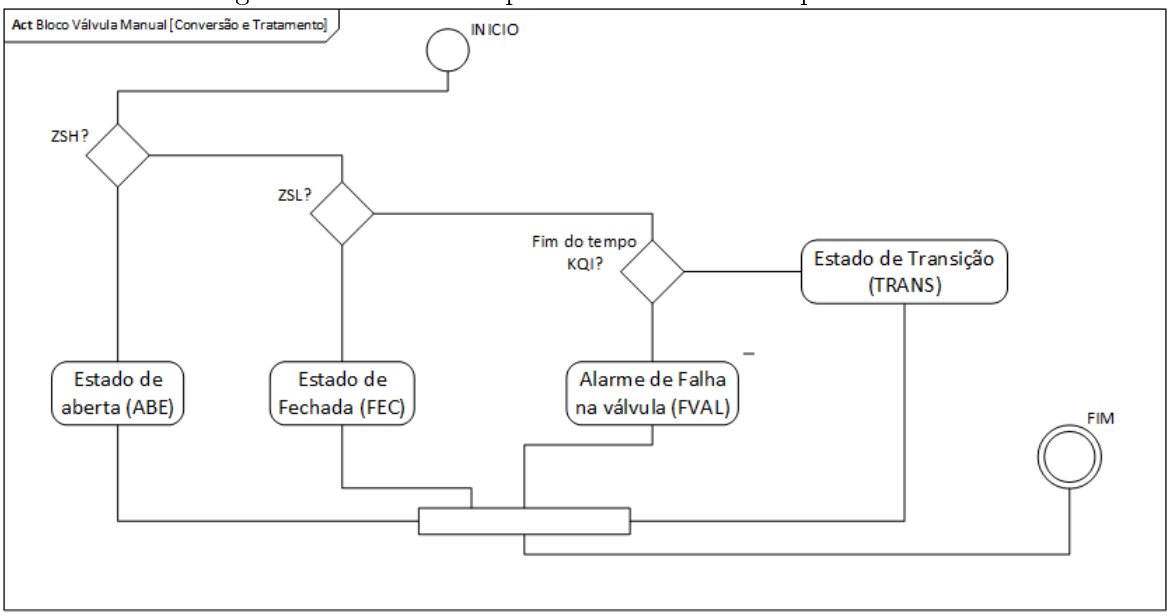
Figura 5.13: IBD Componente válvula de bloqueio manual



Fonte: Do autor

A figura 5.14 é denominada diagrama de atividades (ACT) do componente válvula de bloqueio manual, ela faz uma representação gráfica das etapas do processo de funcionamento e ajuda a modelar os fluxos de trabalho através de uma sequência controlada de ações.

Figura 5.14: ACT Componente válvula de bloqueio manual



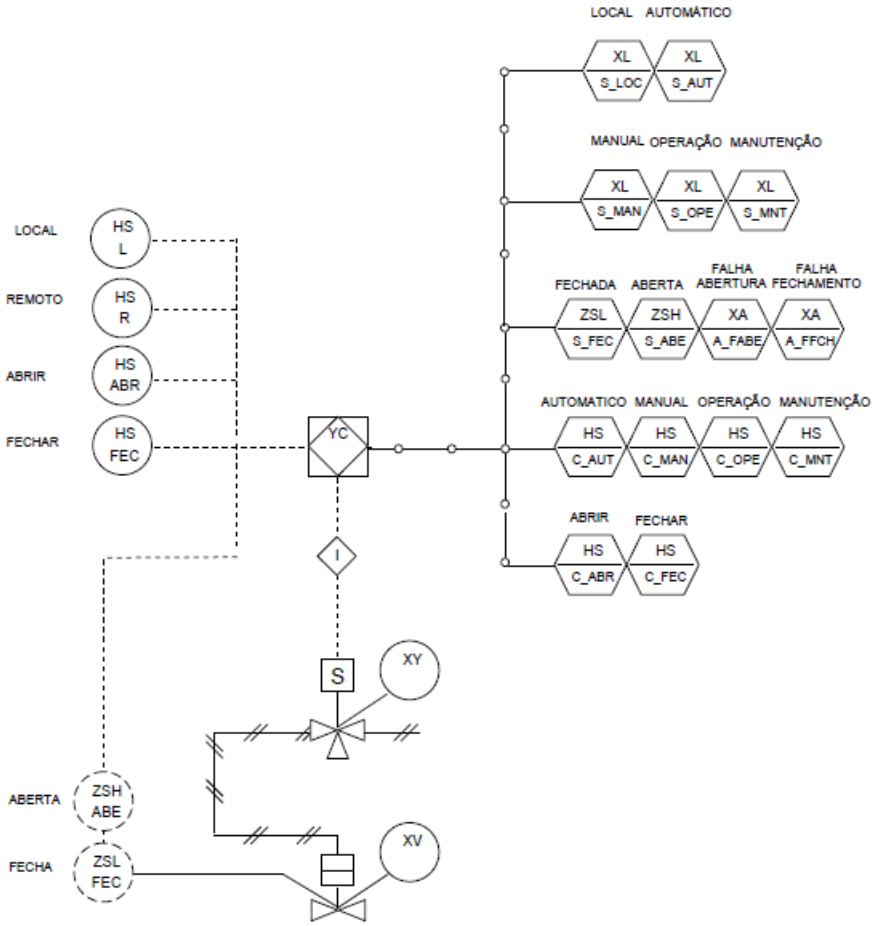
Fonte: Do autor

5.4 *Modelagem do componente válvula pneumática de bloqueio*

5.4.1 *Requisitos do Componente válvula pneumática de bloqueio*

A válvula pneumática de bloqueio é um componente que fisicamente possui acionamento pelo CLP. Através da saída digital, uma solenoide é atuada e libera ar comprimido para o diafragma da válvula, provocando assim o deslocamento do atuador até a posição totalmente aberta ou fechada.
Além de chaves seletoras em campo para decidir o seu modo de operação, estas válvulas possuem também sensores para indicação das suas duas únicas posições: completamente fechada ou aberta. Estas informações são lidas por estes sensores e transmitidas ao CLP, por meio dos canais de entrada digital, a figura 5.15 representa o tipico baseado no P&ID deste componente.

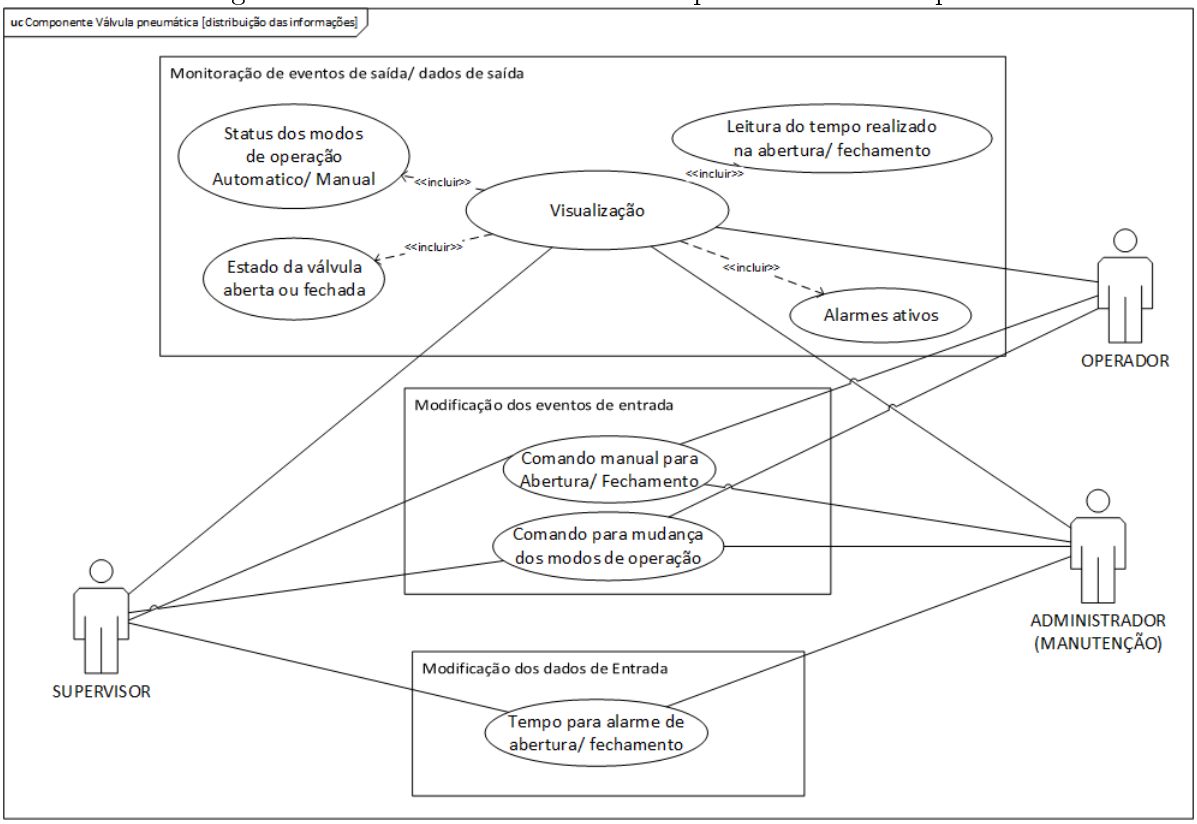
Figura 5.15: Diagrama típico de válvula pneumática de bloqueio



Fonte: Adaptado pelo autor

A figura 5.16 representa o diagrama de caso de uso, especificando os requisitos para este componente.

Figura 5.16: Caso de uso das válvulas pneumática de bloqueio



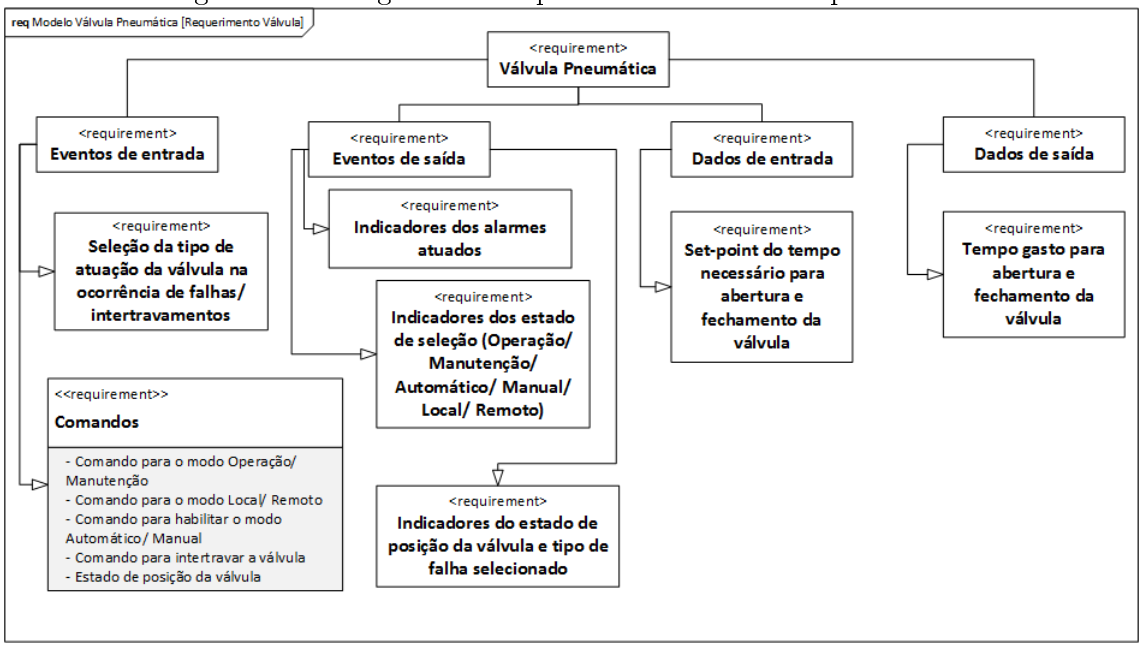
Fonte: Do autor

Como representado nos diagramas acima 5.15 e 5.16, os seguintes requisitos foram previstos para condições de falha deste componente:

- **Falha no fim de curso:** Esta ocorrerá quando os sensores das posições aberta ou fechada na válvula estiverem acionados simultaneamente ou conjuntamente desativados, após a finalização da contagem de tempo para abertura ou fechamento; pré-definido pelo usuário.
- **Falha na abertura:** Após a finalização do tempo pré-definido pelo usuário - suficiente para abertura da válvula - esta falha ocorrerá se o sinal do sensor de posição aberta não for confirmado pelo CLP. É importante considerar que o comando para atuação da mesma já tenha sido enviado pelo controlador.
- **Falha no fechamento:** Após a finalização do tempo pré-definido pelo usuário - suficiente para fechamento da válvula - esta falha ocorrerá se o sinal do sensor de posição fechada não for confirmado pelo CLP. É importante considerar que o comando para atuação da mesma já tenha sido retirado pelo controlador.

Por fim, a figura 5.17 apresenta o diagrama de requerimentos para a válvula de bloqueio pneumática.

Figura 5.17: Diagrama de requerimentos da válvula pneumática

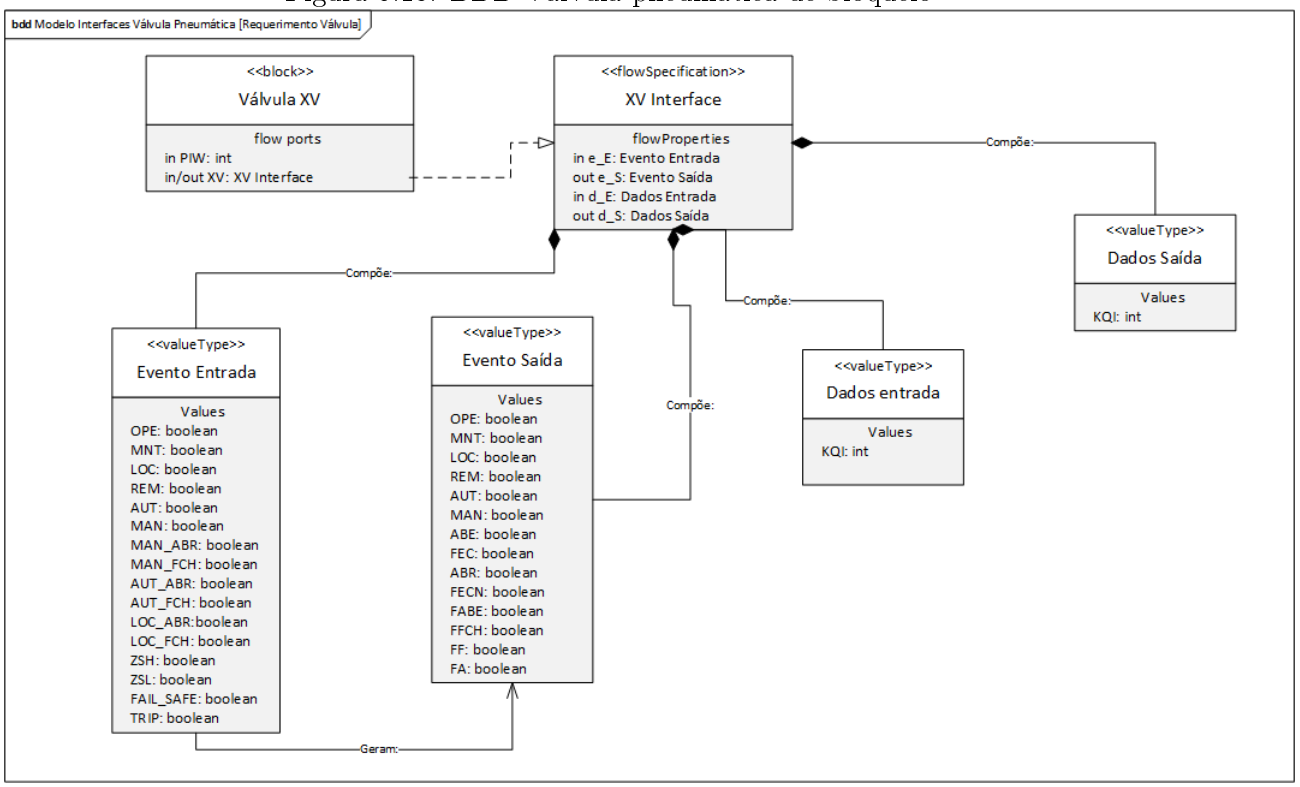


Fonte: Do autor

5.4.2 *BDD Válvula pneumática de bloqueio*

A figura 5.18, representa o diagrama de definição de bloco (BDD) do componente válvula pneumática de bloqueio.

Figura 5.18: BDD Válvula pneumática de bloqueio



Fonte: Do autor

Os eventos de entrada e saída descritos no BDD válvula pneumática de bloqueio, e que não foram apresentados na figura 5.1.2, serão explicados a seguir:

- **MAN_ABR:** Comando enviado pelo sistema SCADA para realizar a abertura da válvula, quando os modos “OPE”, “REM” e “MAN” estiverem simultaneamente selecionados.
- **MAN_FCH:** Comando enviado pelo sistema SCADA para realizar o fechamento, quando os modos “OPE”, “REM” e “MAN” estiverem simultaneamente selecionados.
- **AUT_ABR:** Interface que será utilizada pela própria lógica de controle para realizar a abertura, desde que os modos “OPE”, “REM” e “AUT” estejam todos simultaneamente selecionados.
- **AUT_FCH:** Interface que será utilizada pela própria lógica de controle para realizar o fechamento, desde que os modos “OPE”, “REM” e “AUT” estejam todos simultaneamente selecionados.
- **LOC_ABR:** Comando que poderá ser enviado pelo operador, através de uma chave em campo, para realizar a abertura. Desde que os modos “OPE” e “LOC” estejam todos simultaneamente selecionados.

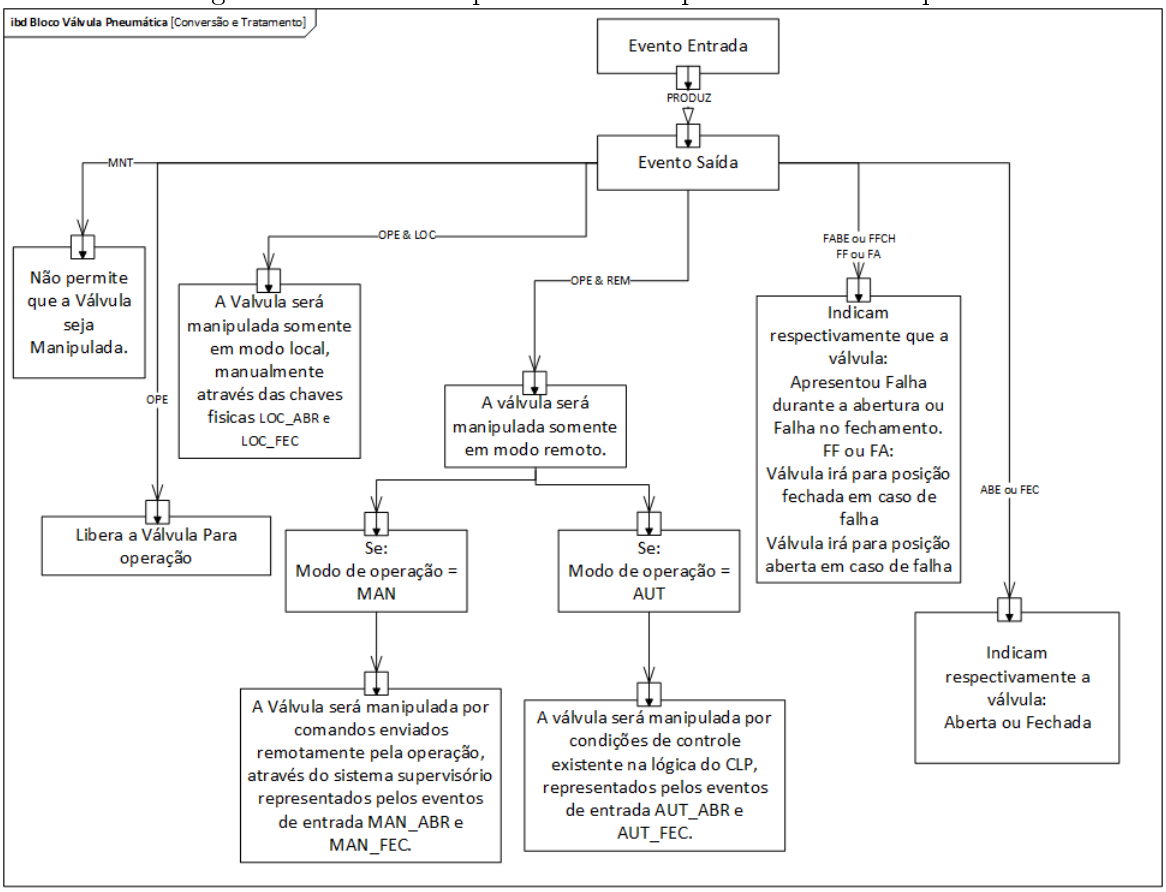
- **LOC_FCH:** Comando que poderá ser enviado pelo operador, através de uma chave em campo, para realizar o fechamento. Desde que os modos "OPE" e "LOC" estejam todos simultaneamente selecionados.
- **ZSL:** Sensor indicador de posição fechada.
- **ZSH:** Sensor indicador de posição aberta.
- **FAILSAFE:** Esta interface deverá ser utilizada para determinar qual posição é segura para válvula assumir, em caso de um "TRIP" no componente. Sendo: 0 = "FF - Feche a válvula, em caso de falha" e 1 = "FA - Abra a válvula, em caso de falha"
- **TRIP:** Utilizado na ocorrência de uma falha do processo ou numa condição insegura para o sistema, este é um evento de entrada capaz de anular todos os modos de operação ativos no componente. A válvula será conduzida para uma posição segura definida anteriormente em "FAILSAFE".
- **FF:** Este item é um evento de saída. Se o seu valor indicar 1, esta válvula terá como comportamento seguro a posição fechada, sempre nas ocorrências de falha ou emergências.
- **FA:** Este item é um evento de saída. Se o seu valor indicar 1, esta válvula terá como comportamento seguro a posição aberta, sempre nas ocorrências de falha ou emergências.
- **ABE:** Indicação do estado de válvula aberta (evento de saída).
- **FEC:** Indicação do estado de válvula fechada (evento de saída).
- **ABR:** Indica a válvula abrindo.
- **FECN:** Indica a válvula fechando.
- **FABE:** Indica o alarme de falha na abertura.
- **FFCH:** Indica o alarme de falha no fechamento.

Foi considerado como dados de entrada e saída da válvula pneumática de bloqueio apenas o "KQI" para informar o tempo necessário para a válvula realizar completamente o seu deslocamento no curso de abertura e fechamento, sem que sejam acionados os alarmes.

5.4.3 *Descrição do comportamento interno - Componente válvula pneumática de bloqueio*

No programa de CLP, este componente tem como principal finalidade o tratamento dos sinais provenientes de canais periféricos digitais - referente ao estado de posição aberta ou fechada deste tipo de válvula e os modos de operação local ou remoto. Outras funcionalidades foram acrescidas, como o tratamento dos alarme após o fim da contagem do tempo de transição entre os estados de aberta e fechada, vice e versa. Por fim, outra funcionalidade acrescida foi a condução da válvula para uma posição segura para o processo, em caso de ocorrência de alguma falha ou intertravamento externo.
A figura 5.19 apresenta como são correspondidos cada evento de saída e suas interfaces, através do diagrama de bloco interno (IBD).

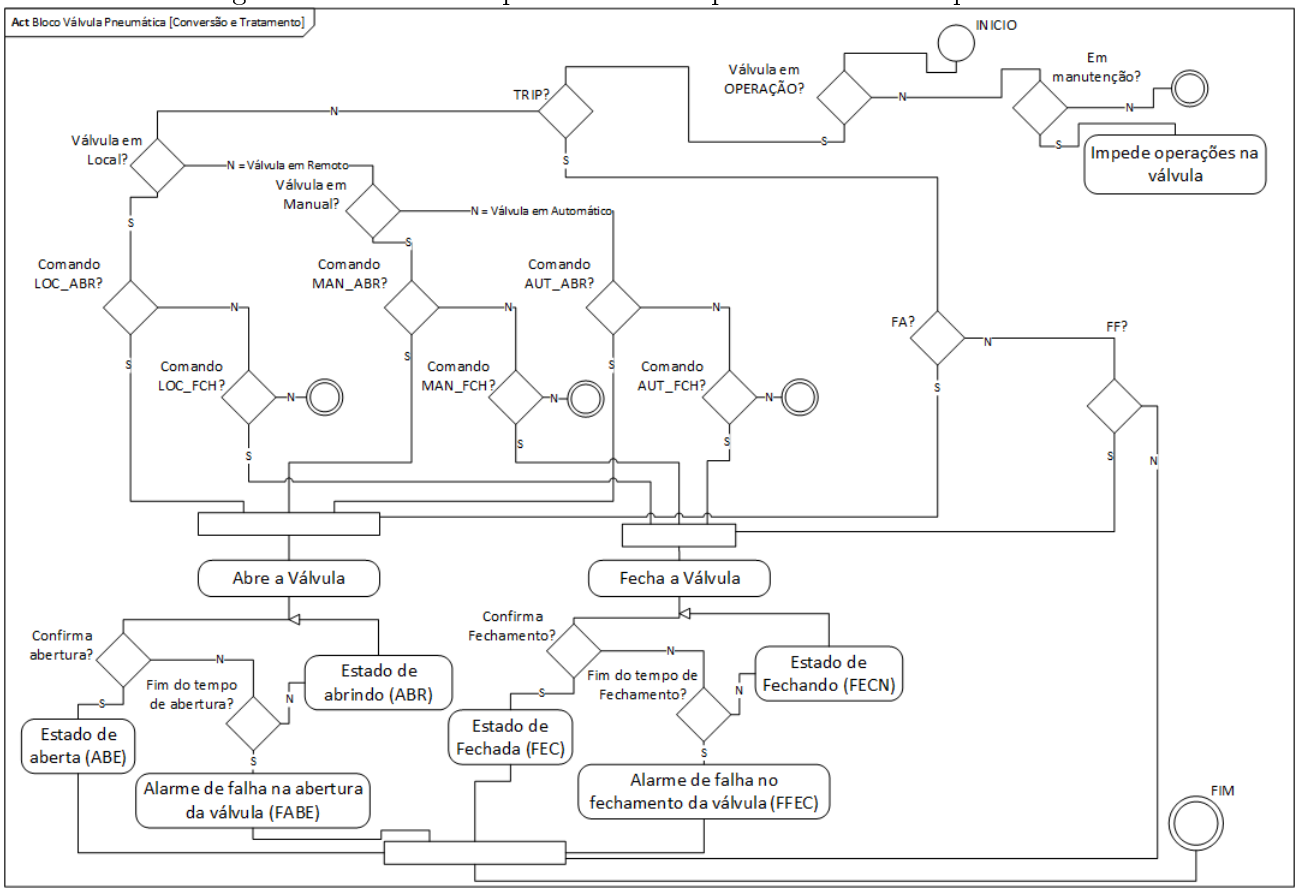
Figura 5.19: IBD Componente válvula pneumática de bloqueio



Fonte: Do autor

A figura 5.20 é denominada diagrama de atividades (ACT) do componente válvula pneumática de bloqueio, ela faz uma representação gráfica das etapas do processo de funcionamento e ajuda a modelar o fluxo de trabalho através de uma sequência controlada de ações.

Figura 5.20: ACT Componente válvula pneumática de bloqueio



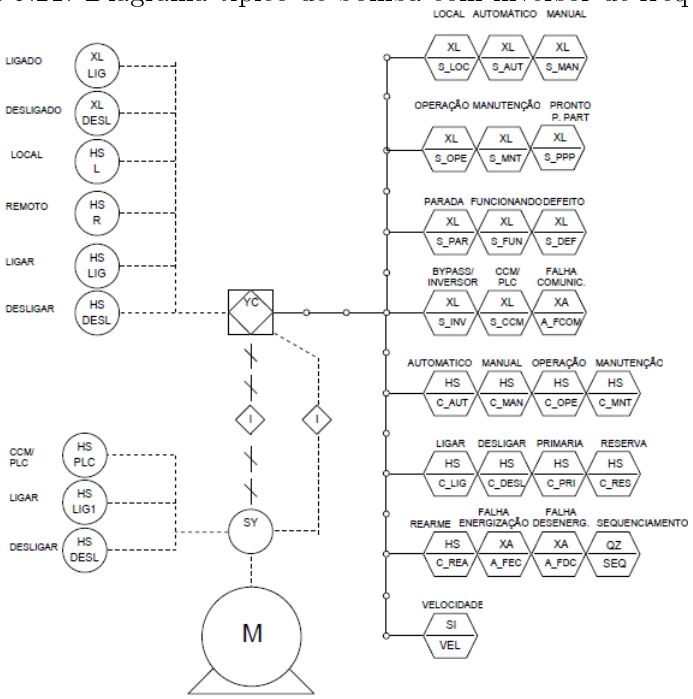
Fonte: Do autor

5.5 *Modelagem do componente bomba com inversor de frequência*

5.5.1 *Requisitos do Componente bomba com inversor de frequência*

A bomba com inversor de frequência é um componente que possui acionamento físico pelo CLP, através da saída digital ou rede de comunicação, liberando o inicio de partida do motor elétrico. O inversor de frequência é um dispositivo eletrônico com capacidade de determinar a variação de frequência e amplitude do nível de tensão aplicada nas bobinas do motor. Este equipamento tem a capacidade de fornecer uma série de grandezas elétricas envolvidas na operação da bomba. Além disso, normalmente em sua instalação industrial, possui chaves seletoras para definir o seu modo de operação.
Estas informações são lidas e transmitidas ao CLP por meio dos canais de entrada digital ou rede de comunicação. A figura 5.21 representa o diagrama tipico baseado no P&ID.

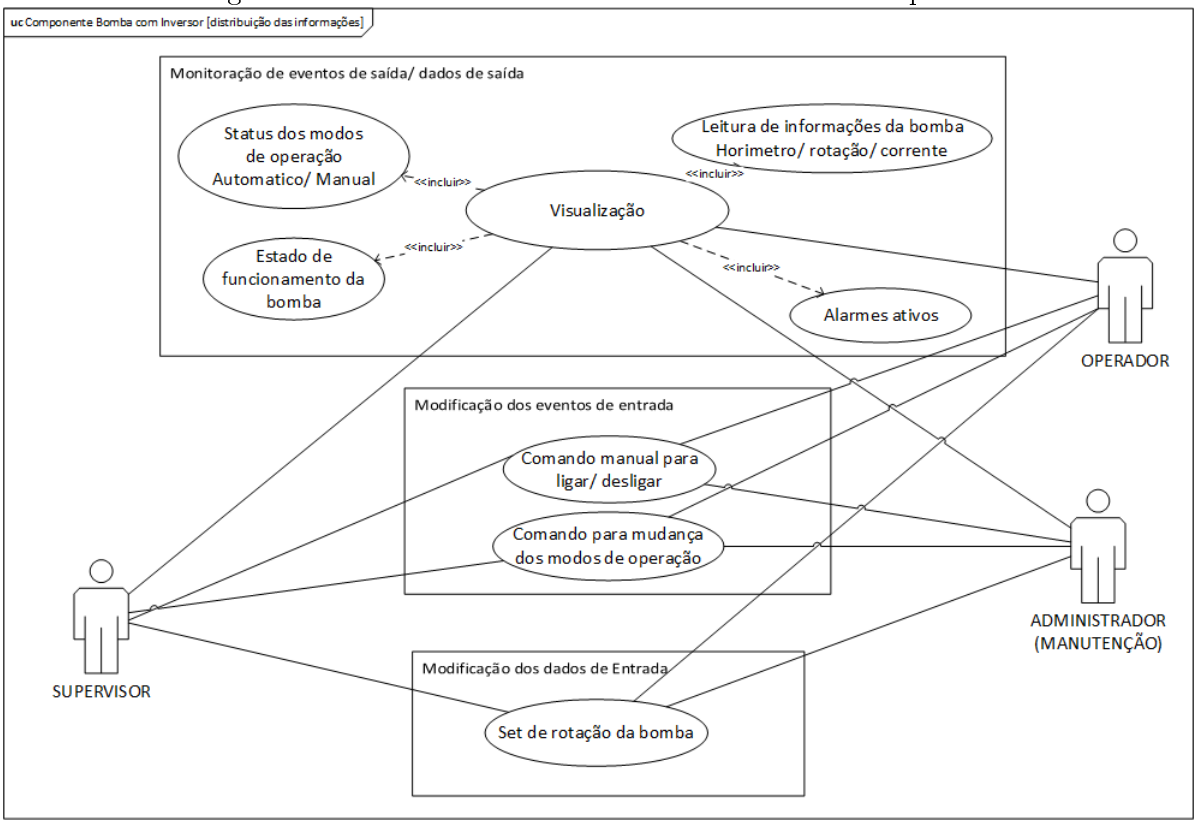
Figura 5.21: Diagrama típico de bomba com inversor de frequência



Fonte: Adaptado pelo autor

A figura 5.22 representa o diagrama “caso de uso”, demonstrando as formas interação e uso dos requisitos para este componente.

Figura 5.22: Caso de uso de bomba com inversor de frequência



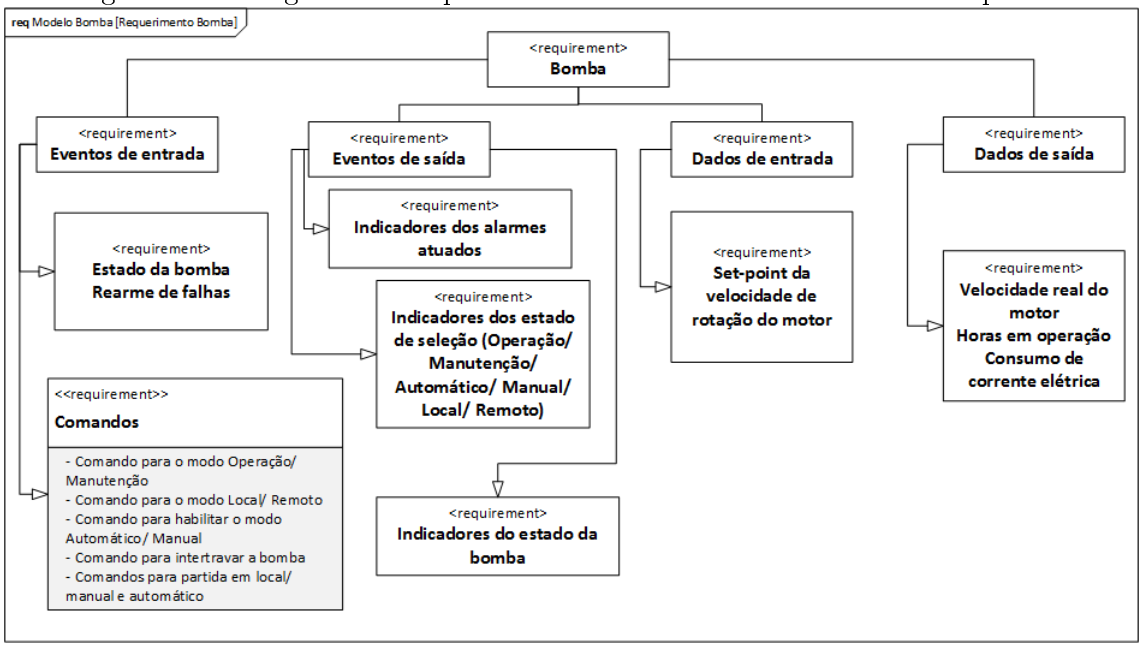
Fonte: Do autor

Os seguintes requisitos foram previstos para condições de falha deste componente:

- **Resumo de falhas no inversor:** Sempre que o inversor identificar algum problema interno que impossibilite a operacionalidade do equipamento, tais sobrecorrente, sobretensão, falta de fase, defeito, dentre outros; Este sinal indicará o resumo destes tipos de falhas.
- **Falha na energização:** Ocorrerá quando, após o acionamento do motor via CLP, não for identificado pelo mesmo controlador lógico o sinal de confirmação "Motor Ligado"; sempre após o fim da contagem de tempo pré-definido pelo usuário.
- **Falha na desenergização:** Ocorrerá quando, após o desligamento do motor via CLP, não for identificado pelo mesmo controlador lógico o sinal de confirmação "Motor Desligado"; novamente sempre após o fim da contagem de tempo pré-definido pelo usuário.

Por fim, a figura 5.23 apresenta o diagrama de requerimentos para a bomba com inversor de frequência.

Figura 5.23: Diagrama de requerimentos da bomba com inversor de frequência

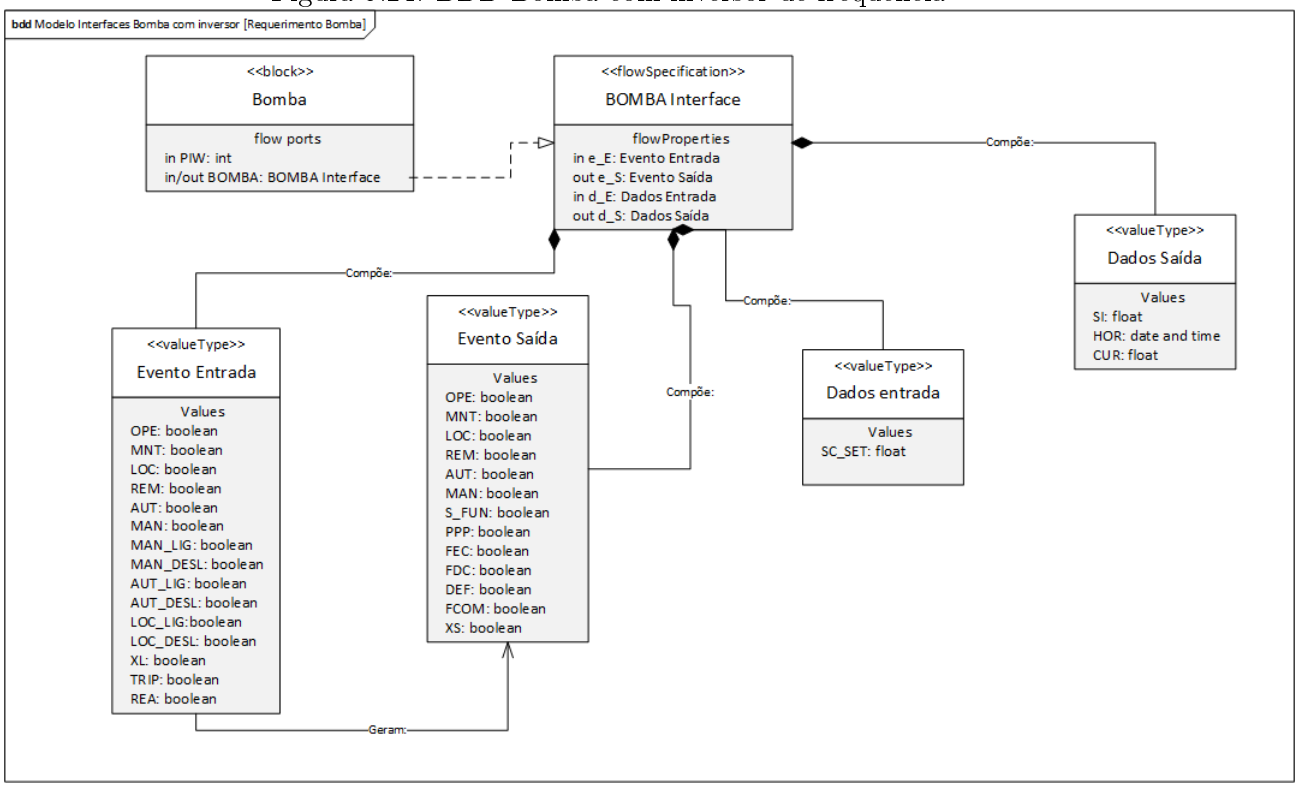


Fonte: Do autor

5.5.2 *BDD Bomba com inversor de frequência*

A figura 5.24, representa o diagrama de definição de bloco (BDD) do componente bomba com inversor de frequência.

Figura 5.24: BDD Bomba com inversor de frequência



Fonte: Do autor

A seguir os eventos de entrada e saída do BDD deste componente, que não foram apresentados em 5.1.2, serão explicados:

- **MAN_LIG:** Comando via sistema SCADA para acionamento da bomba, desde que os modos “OPE”, “REM” e “MAN” estejam simultaneamente selecionados.
- **MAN_DESL:** Comando via sistema SCADA para realizar o desligamento da bomba, desde que os modos “OPE”, “REM” e “MAN” estejam simultaneamente selecionados.
- **AUT_LIG:** Interface utilizada pela própria lógica de controle para realizar o acionamento da bomba, desde que os modos “OPE”, “REM” e “AUT” estejam simultaneamente selecionados.
- **AUT_DESL:** Interface utilizada pela própria lógica de controle para realizar o desligamento da bomba, desde que os modos “OPE”, “REM” e “AUT” estejam simultaneamente selecionados.
- **LOC_LIG:** Comando realizado pelo operador através de uma botoeira em campo para o acionamento da bomba, desde que os modos “OPE” e “LOC” estejam simultaneamente selecionados.

- **LOC_DESL:** Comando realizado pelo operador através de uma botoeira em campo para desligamento da bomba, desde que os modos "OPE" e "LOC" estejam simultaneamente selecionados.
- **XL:** Interface utilizada no componente para confirmar que fisicamente a bomba foi realmente ligada.
- **TRIP:** Interface utilizada pelo componente para que no surgimento de uma condição insegura do processo, a bomba seja imediatamente desligada e impedida de religar até que a causa geradora seja normalizada.
- **REA:** Evento de entrada utilizado para rearmar a falha na bomba, após ocorrência de um "TRIP", sinalizando que o equipamento estará pronto para uma nova partida.
- **S_FUN:** Evento de saída no qual indica a bomba com o estado de funcionando.
- **PPP:** Evento de saída em que sinaliza o equipamento como pronto para uma nova partida, ou seja, liberando o inicio de uma nova operação.
- **FEC:** Significa "Falha na energização do contator", indicando que a bomba não correspondeu com resultado esperado após o comando para seu acionamento via CLP, permanecendo desligada.
- **FDC:** Significa "Falha na desenergização do contator", indicando que a bomba não correspondeu com resultado esperado após o comando para seu desligamento via CLP, permanecendo ligada.
- **FCOM:** Falha na comunicação. Ocorre quando a troca de informações entre o CLP e inversor, feita via rede de comunicação, começa a apresentar falhas.
- **XS:** Utilizado para o acionamento da saída digital via CLP, responsável pela energização da bomba.

Foi considerado como dado de entrada da bomba apenas o "SC_SET" para informar qual deve ser a velocidade obedecida pelo inversor de frequência. Como elementos do tipo dado de saída, a expressão "SI" representa a velocidade real no equipamento, "HOR" diz respeito ao horímetro e "CUR" a corrente elétrica consumida pela bomba.

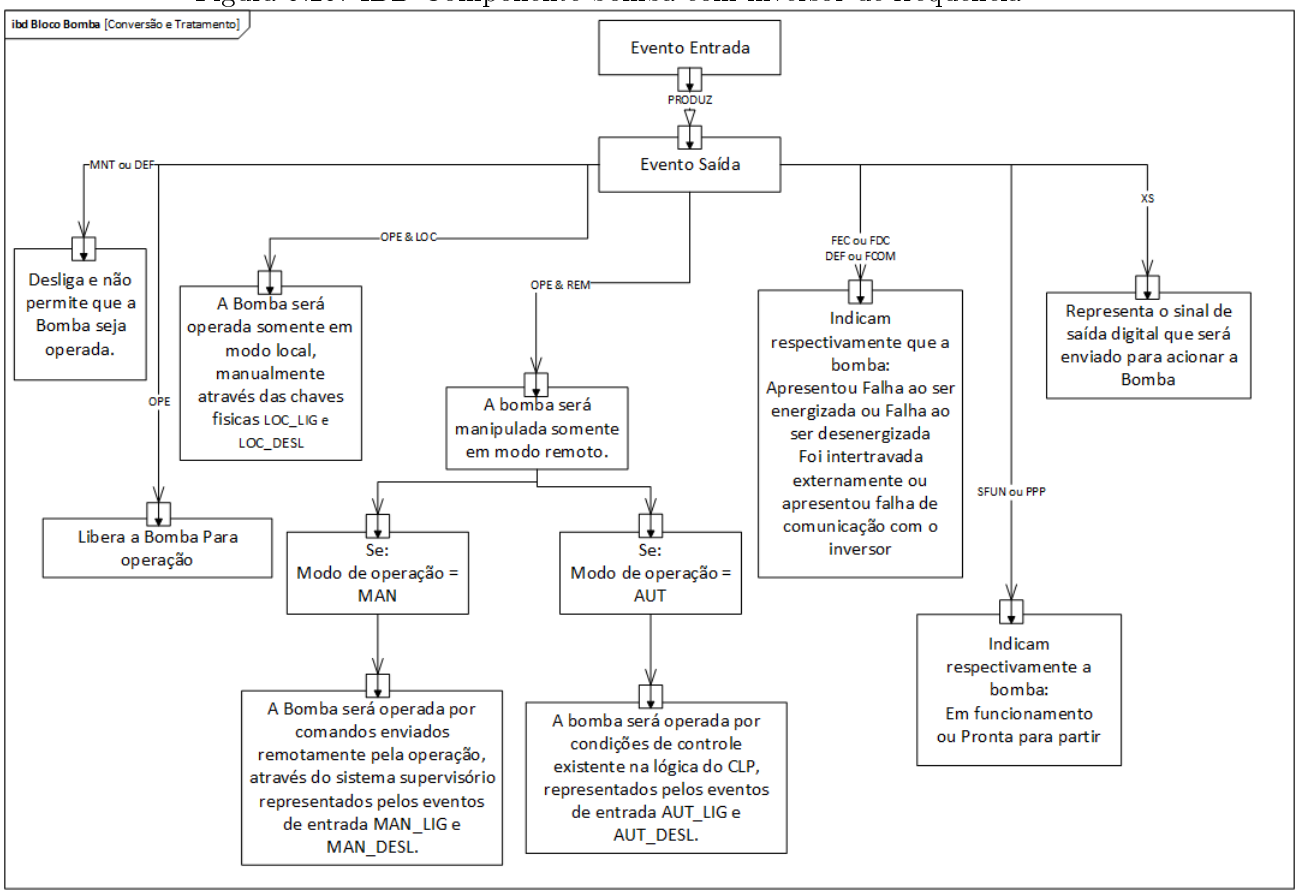
5.5.3 Descrição do comportamento interno - Componente bomba com inversor de frequência

O comportamento do componente bomba está condicionado aos eventos de saída. Sendo assim, ao ser selecionado o modo manutenção ou após sinalização de um intertravamento

externo como eventos de entrada, a mesma ficará desligada e permanecerá impedida de retornar a operação. Por sua vez quando o modo "OPE" for escolhido, sem que algum defeito tenha sido ativado através de intertravamento externo "TRIP"; a mesma estará pronta para partir "PPP", entrando em operação quando uma das formas de partida for iniciada.

As condições de falha propostas no levantamento de requisitos, vistas em 5.5.1 foram contempladas através dos eventos de saída "FEC", "FDC", "DEF", "FCOM"; representando respectivamente: falha ao energizar a bomba, falha ao desenergizar, intertravamento externo ativado, e falha de comunicação. A figura 5.25 representa como são correspondidos cada evento de saída e suas interfaces, através do diagrama de bloco interno (IBD).

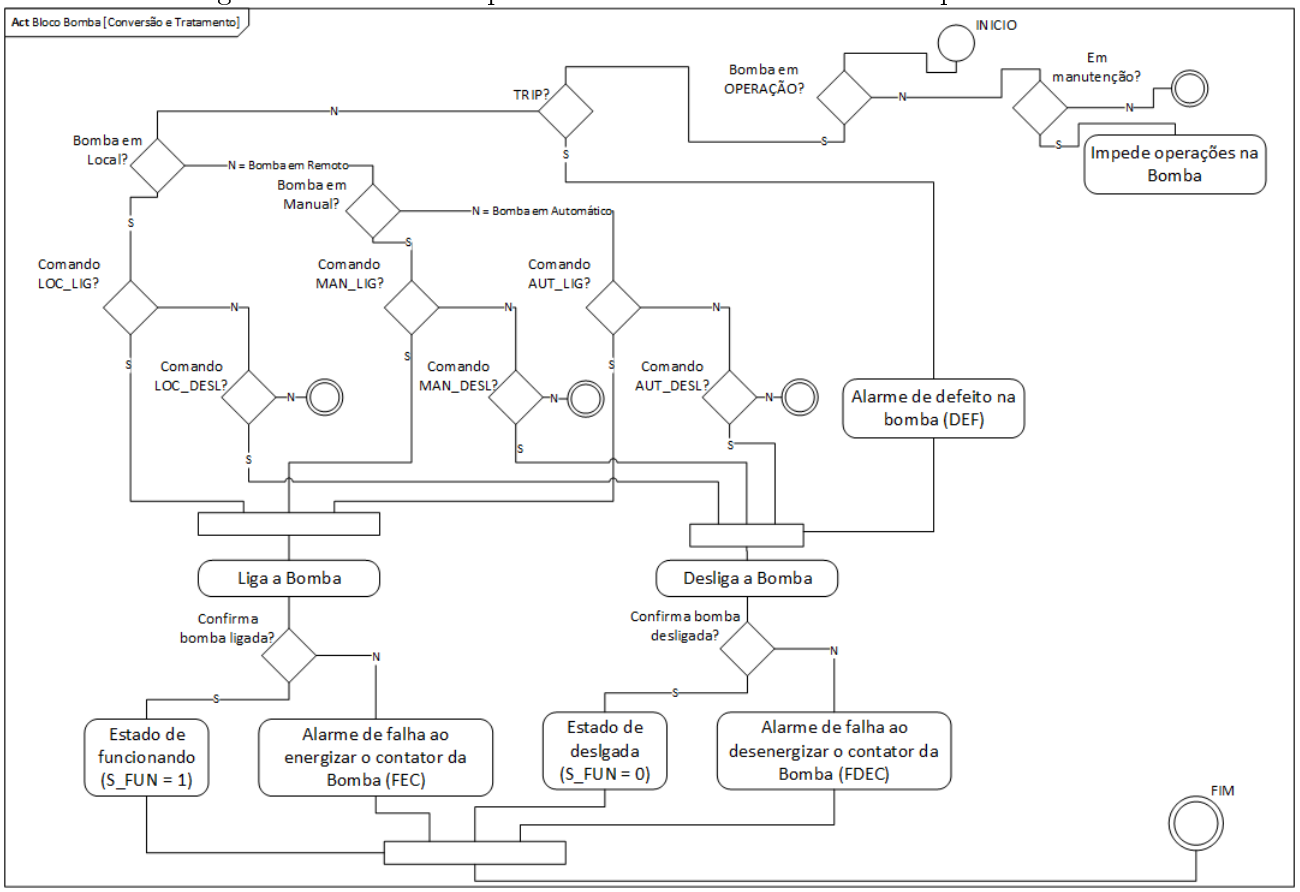
Figura 5.25: IBD Componente bomba com inversor de frequência



Fonte: Do autor

A figura 5.26 é denominada diagrama de atividades (ACT) do componente bomba com inversor de frequência, ela faz uma representação gráfica das etapas do processo de funcionamento e ajuda a modelar os fluxos de trabalho através de uma sequência controlada de ações.

Figura 5.26: ACT Componente bomba com inversor de frequência



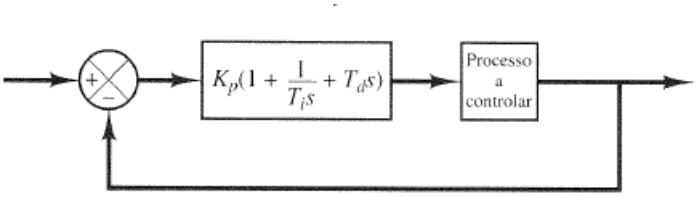
Fonte: Do autor

5.6 *Modelagem do componente malha de controle PID*

5.6.1 *Requisitos do componente malha de controle PID*

A malha de controle PID está estruturada como um componente lógico que possui interfaces físicas com o CLP, através da saída analógica ou rede de comunicação, por meio de um sinal padronizado. Por exemplo, através de uma equação especifica, o controle PID tem por finalidade definir como uma válvula de controle precisa ser posicionada ou em qual velocidade uma bomba deve girar; para que haja um controle sobre o sistema. Tudo isto para proporcionar que a variável de processo "PV" seja controlada e tenha o seu erro compensado. Este tipo de controlador é muito utilizado quando não é possível descobrir a função de transferência da planta ou do processo a ser controlado. A figura 5.27 demonstra através do diagrama de blocos a utilização do controlador PID e sua equação.

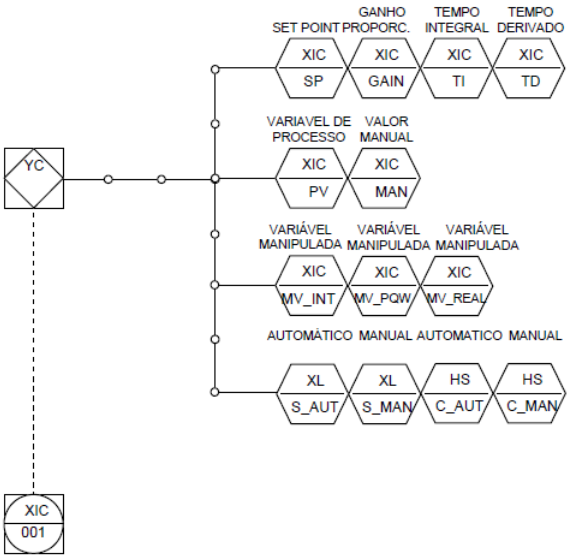
Figura 5.27: Diagrama típico de malha de controle PID



Fonte: Adaptado pelo autor

Atualmente na maioria dos CLPs a equação do PID já está implementada, exigindo ao programador apenas definir seus critérios de operação e os modos de sua utilização. Não é o proposito deste trabalho detalhar o funcionamento do controlador PID ou quais estratégias podem ser adotadas numa malha de controle, devendo o leitor consultar a literatura relacionada ao assunto para maiores esclarecimentos (RIBEIRO, 1999). Quanto aos tipos de requisitos que utilizados, a figura 5.28 demonstra como pode ser a interface com este componente, baseado no P&ID (*Piping and Instrumentation Diagram/Drawing* - diagrama de engenharia de tubulação e instrumentação).

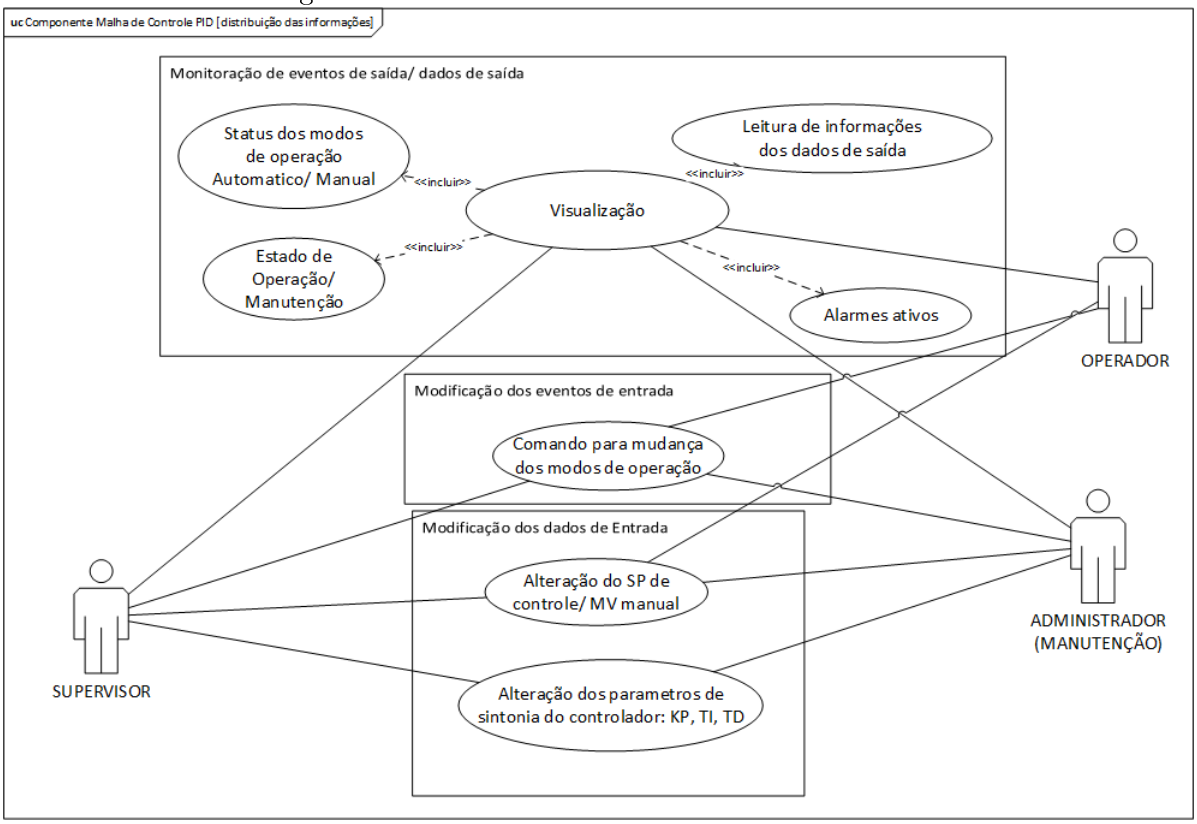
Figura 5.28: Diagrama típico de malha de controle PID



Fonte: Adaptado pelo autor

A figura 5.29 representa o diagrama de caso de uso, especificando os requisitos para este componente.

Figura 5.29: Caso de uso da malha de controle PID



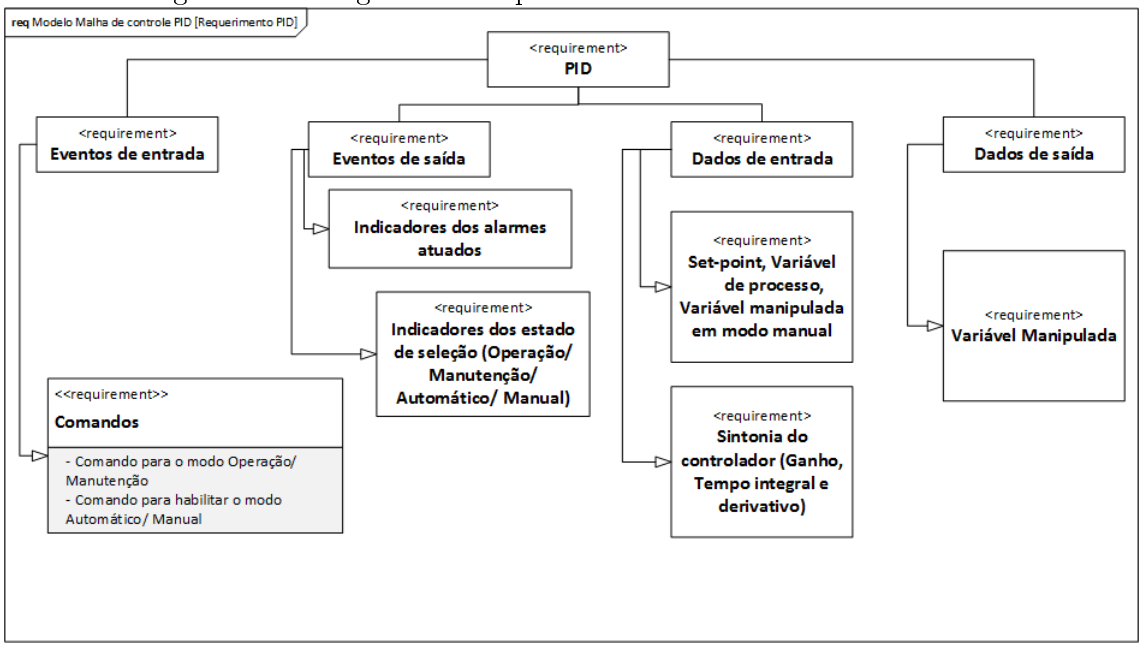
Fonte: Do autor

Os requisitos previstos para condições de falha deste componente são aos indicações de saturação da malha, onde:

- **HLM:** Indica limite máximo da MV atingido
- **LLM:** Indica limite mínimo da MV atingido

Por fim, a figura 5.30 apresenta o diagrama de requerimentos para a malha de controle PID.

Figura 5.30: Diagrama de requerimentos da malha de controle PID

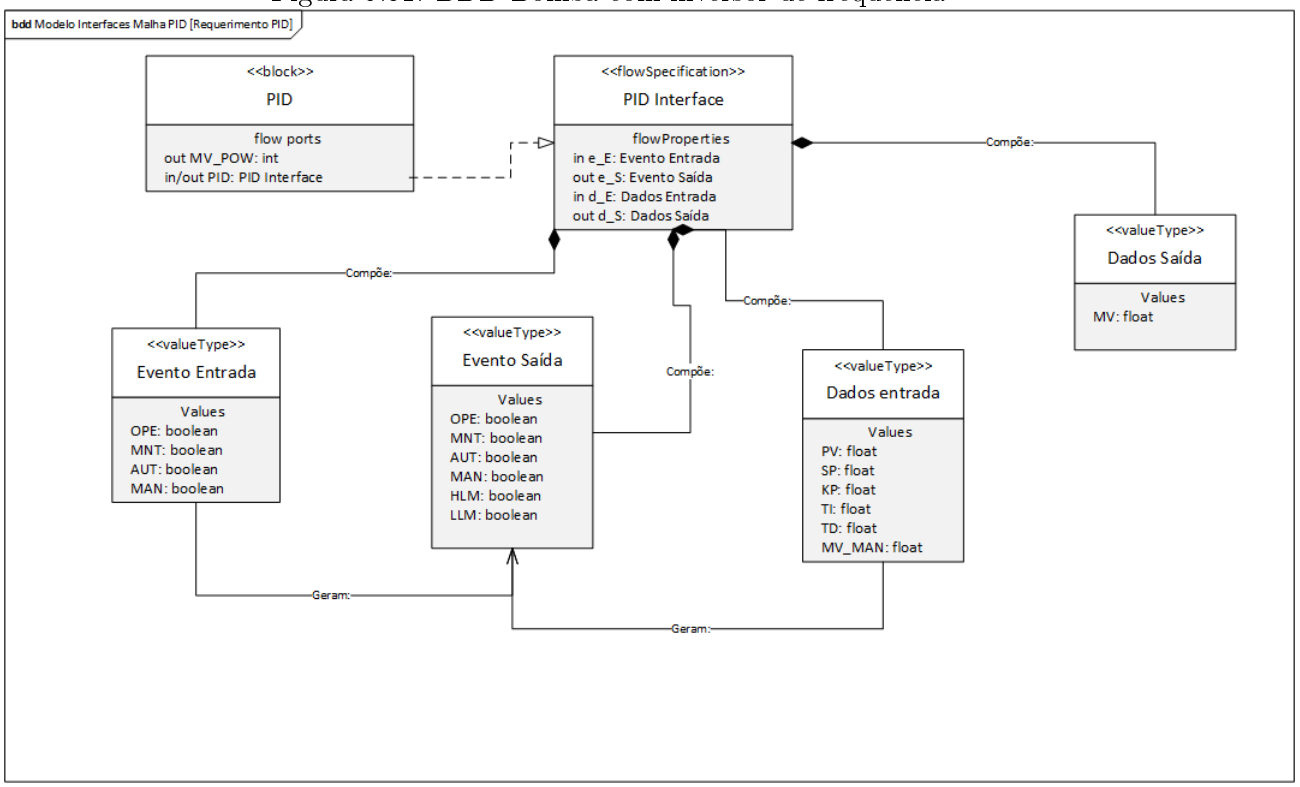


Fonte: Do autor

5.6.2 *BDD Malha de controle PID*

A figura 5.31, representa o diagrama de definição de bloco (BDD) do componente malha de controle PID.

Figura 5.31: BDD Bomba com inversor de frequência



Fonte: Do autor

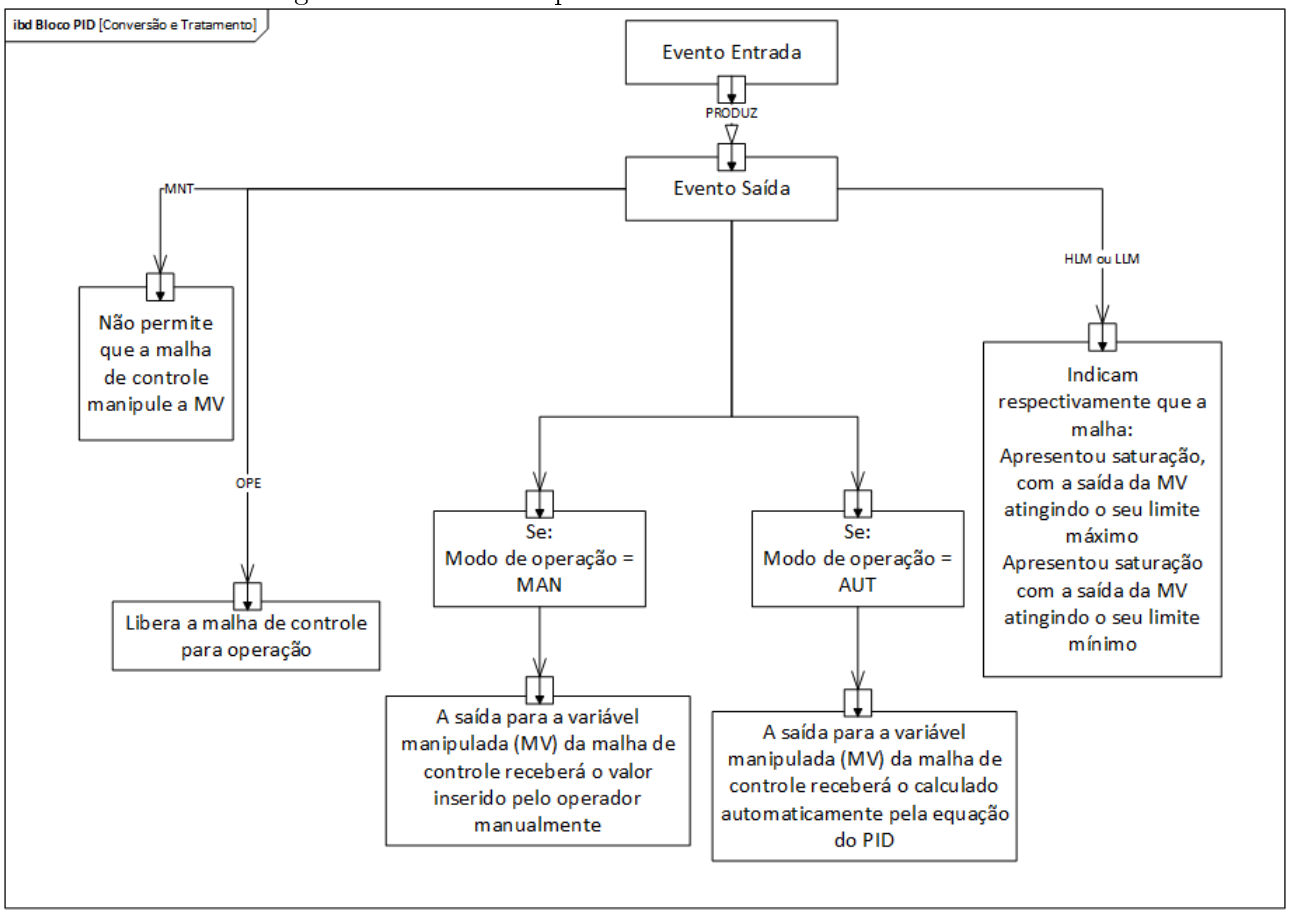
A seguir os dados de entrada e saída do BDD malha de controle PID serão explicados contemplando as seguintes informações:

- **PV:** Representa a variável de processo.
- **SP:** Representa SET-POINT da malha.
- **KP:** Representa ganho proporcional.
- **TI:** Representa tempo integral.
- **TD:** Representa tempo derivativo.
- **MV:** Representa a variável manipulada.
- **HLM:** Indica limite máximo da MV atingido
- **LLM:** Indica limite mínimo da MV atingido

5.6.3 *Descrição do comportamento interno - Componente malha de controle PID*

O PID, após a convenção adotada pela IEC 61131-3, possui atualmente uma estrutura funcional consolidada, já incluindo informações previstas inclusive neste trabalho.
Portanto está sendo apresentado uma modelagem para este componente, neste trabalho, no intuito de adequar o formato das variáveis com o dos outros já demonstrados. Logo as descrições do comportamento deste componente são básicas contemplando o tratamento da variável manipulada nos modos automático e manual. A figura 5.32 demonstra como são correspondidos cada evento de saída e suas interfaces, através do diagrama de bloco interno (IBD).

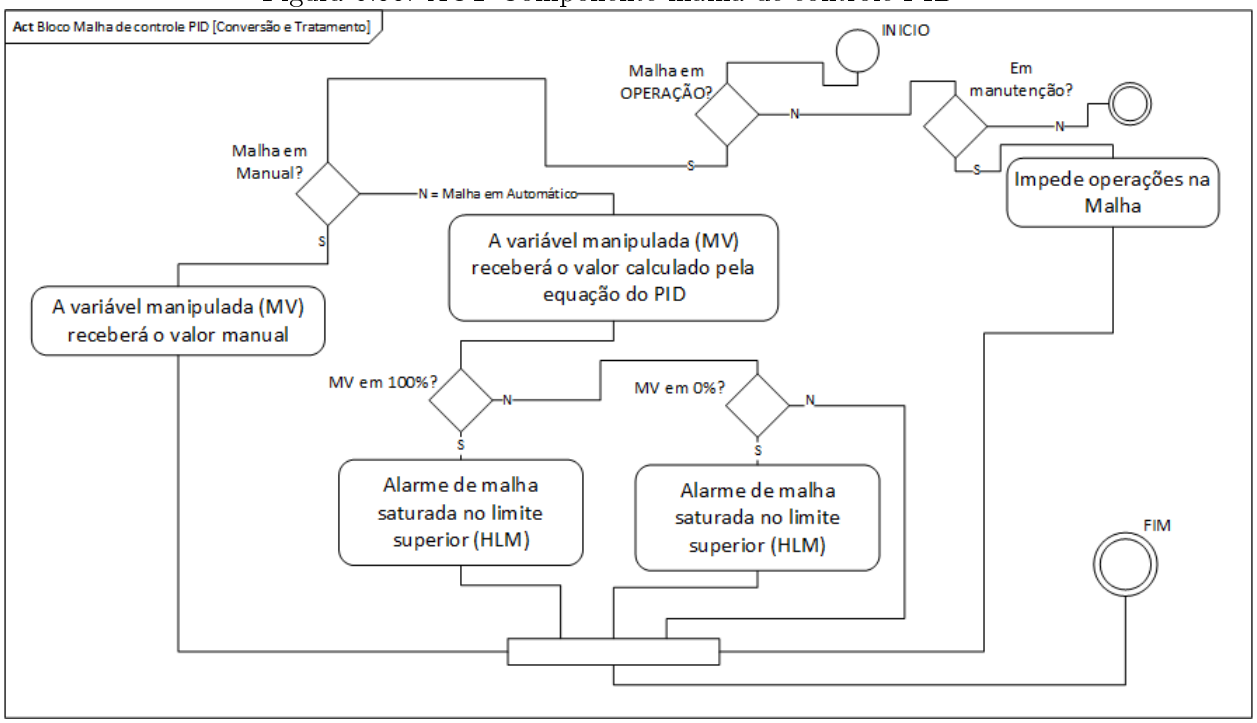
Figura 5.32: IBD Componente malha de controle PID



Fonte: Do autor

A figura 5.33 é denominada diagrama de atividades (ACT) do componente malha de controle PID, ela faz uma representação gráfica das etapas do processo de funcionamento e ajuda a modelar os fluxos de trabalho através de uma sequência controlada de ações.

Figura 5.33: ACT Componente malha de controle PID



Fonte: Do autor

5.7 Construção do código

Com o objetivo de avaliar os resultados alcançados na utilização dos modelos propostos por este trabalho, os componentes foram implementados em ambiente de simulação através da plataforma RSLogix 5000, fornecida pela Rockwell Software; esta ferramenta atende aos requisitos do padrão IEC 61131-3.

5.7.1 Implementação do transmissor analógico

Acompanhando as explicações apresentadas no capítulo 3, a imagem 5.34 representa a estrutura de dados do transmissor analógico, utilizando como variável um tipo de dado definido pelo usuário *"User-Defined DataType"*, semelhante ao *"Struct"* da linguagem "C".

Figura 5.34: Base de dados do transmissor analógico no programa RsLogix5000

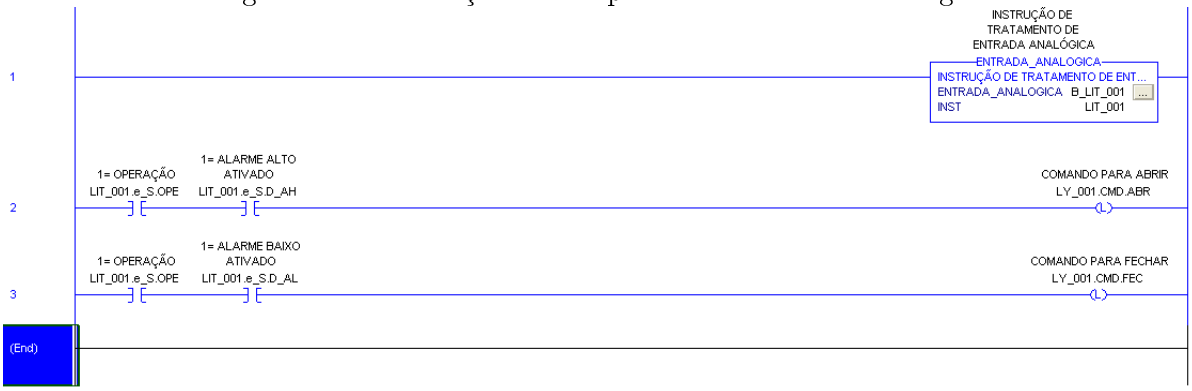
Name	Alias For	Base Tag	Data Type	Style	Description
B_LIT_001			ENTRADA_ANALOGICA		INSTRUÇÃO DE ...
B_M_001			MOTOR_ELET		
B_XV_001			VALVULA_ONOFF		
LIT_001			INSTRUMENTO		
LIT_001.PIW			DINT	Decimal	VALOR VINDO D...
LIT_001.e_E			e_E_INST		COMANDO DOS ...
LIT_001.d_E			d_E_INS		ESCRITA DOS V...
LIT_001.d_S			d_S_INS		REGISTRO DE L...
LIT_001.e_S			e_S_INST		EVENTOS DE S...
LIT_001.e_S.OPE			BOOL	Decimal	1= OPERAÇÃO
LIT_001.e_S.MNT			BOOL	Decimal	1= MANUTENÇÃO
LIT_001.e_S.STOS			BOOL	Decimal	1= VALOR SUBS...
LIT_001.e_S.S_AHH			BOOL	Decimal	1= ALARME MUI...
LIT_001.e_S.S_AH			BOOL	Decimal	1= ALARME ALT...
LIT_001.e_S.S_AL			BOOL	Decimal	1= ALARME BAI...
LIT_001.e_S.S_ALL			BOOL	Decimal	1= ALARME MUI...
LIT_001.e_S.D_AHH			BOOL	Decimal	1= ALARME MUI...
LIT_001.e_S.D_AH			BOOL	Decimal	1= ALARME ALT...
LIT_001.e_S.D_AL			BOOL	Decimal	1= ALARME BAI...
LIT_001.e_S.D_ALL			BOOL	Decimal	1= ALARME MUI...
M_001			MOTOR		
M_001_HS_CCM			BOOL	Decimal	
M_001_HS_DES			BOOL	Decimal	
M_001_HS_LIG			BOOL	Decimal	
M_001_HS_LOC			BOOL	Decimal	

Monitor Tags | Edit Tags

Fonte: Do autor

A imagem 5.35 representa a lógica no CLP com o uso do "*Function Block*" transmissor analógico, a interação com as interfaces e sua utilização como um bloco de controle (A Rockwell Software denominou o "*Function Block*" como "*Add-on Instruction*").

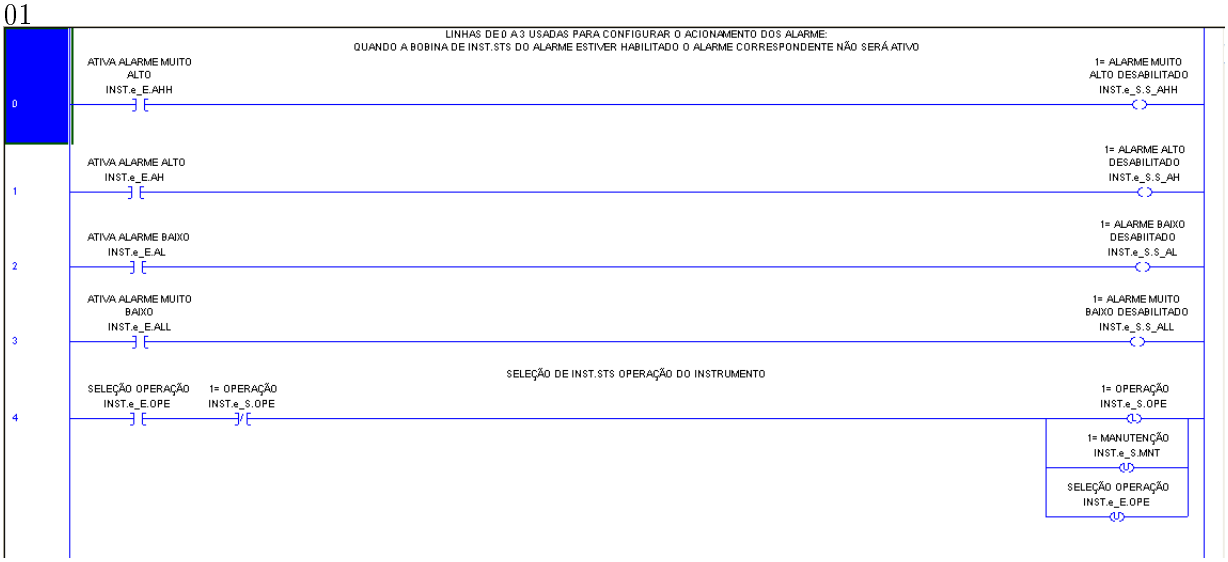
Figura 5.35: Utilização do componente transmissor analógico



Fonte: Do autor

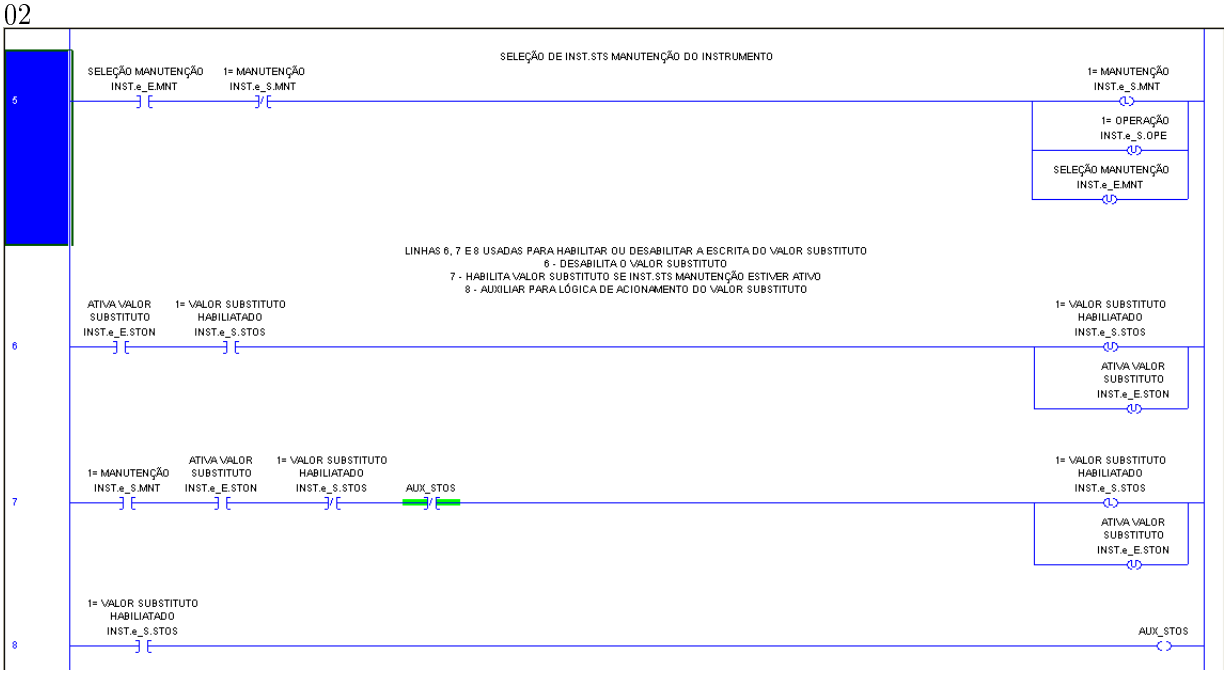
As imagens 5.36, 5.37, 5.38, 5.39 demonstram a lógica para um único transmissor analógico, caso a mesma fosse elaborada puramente em ladder convencional, sem a utilização dos adventos de *"Function Block"* e *"User-Defined DataType"*.

Figura 5.36: Exemplo de lógica convencional para o tratamento do transmissor analógico - Parte 01



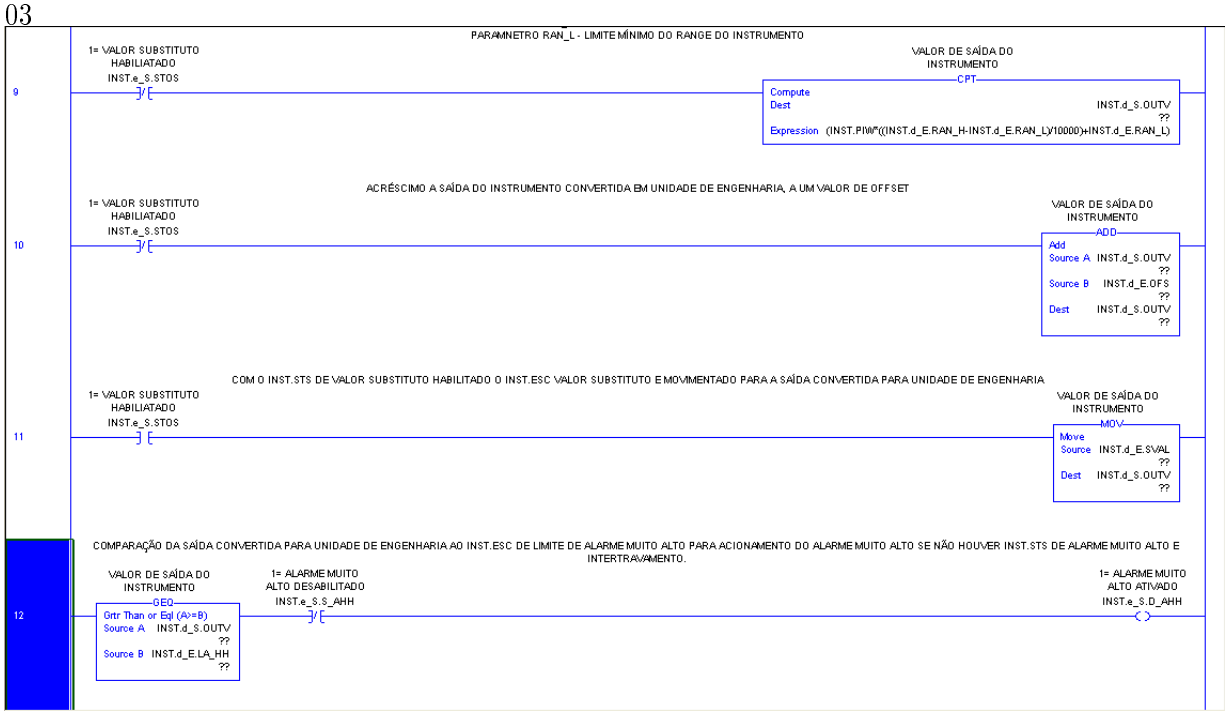
Fonte: Do autor

Figura 5.37: Exemplo de lógica convencional para o tratamento do transmissor analógico - Parte 02



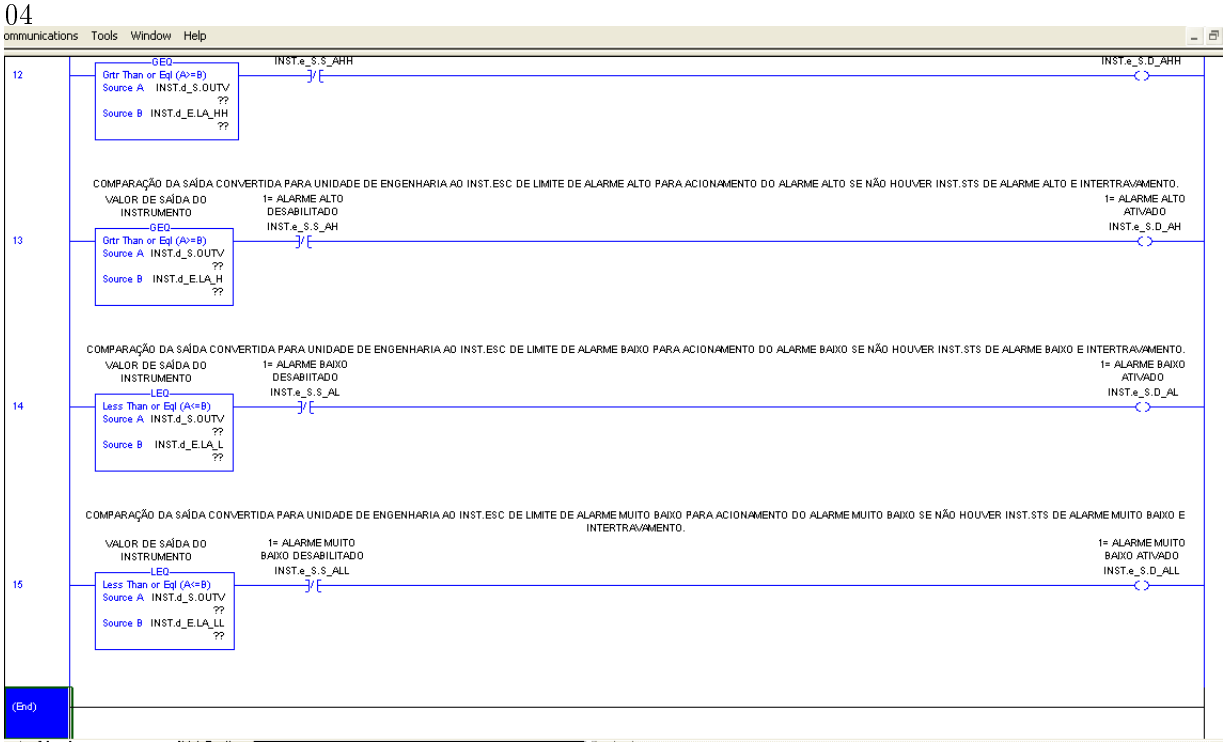
Fonte: Do autor

Figura 5.38: Exemplo de lógica convencional para o tratamento do transmissor analógico - Parte 03



Fonte: Do autor

Figura 5.39: Exemplo de lógica convencional para o tratamento do transmissor analógico - Parte 04



Fonte: Do autor

As estruturas exibidas anteriormente refletem o custo da programação tradicional para

elaboração de um programa de CLP. Ou seja, para um sistema de controle com centenas de transmissores analógicos, seria necessário repetir pela mesma quantidade de vezes todos os diagramas "Ladder" exibidos nas figuras 5.36, 5.37, 5.38, 5.39.

5.7.2 *Implementação da válvula de bloqueio manual*

A imagem 5.40 representa a estrutura de dados do transmissor analógico, utilizando como variável um tipo de dado definido pelo usuário *"User-Defined DataType"*, semelhante ao *"Struct"* da linguagem "C".

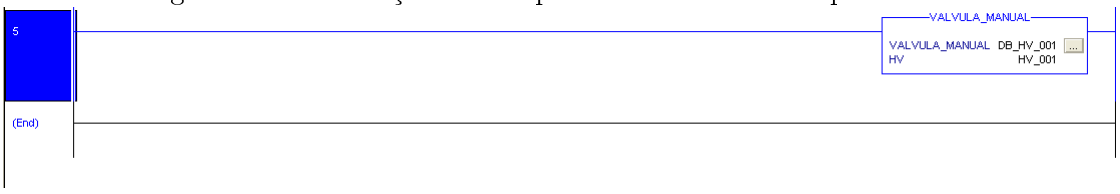
Figura 5.40: Base de dados válvula de bloqueio manual no programa RsLogix5000

HV_001	{...}	{...}		HV_TYPE	VALVULA MANUAL
HV_001.e_E	{...}	{...}		e_E_HV	VALVULA MANUAL
HV_001.e_E.ZSH	0		Decimal	BOOL	VALVULA MANUAL
HV_001.e_E.ZSL	0		Decimal	BOOL	VALVULA MANUAL
HV_001.e_S	{...}	{...}		e_S_HV	VALVULA MANUAL
HV_001.e_S.ABE	0		Decimal	BOOL	VALVULA MANUAL ABERTA
HV_001.e_S.FEC	0		Decimal	BOOL	VALVULA MANUAL FECHADA
HV_001.e_S.TR...	0		Decimal	BOOL	VALVULA MANUAL EM TRANSICAO
HV_001.e_S.FVAL	0		Decimal	BOOL	VALVULA MANUAL FALHA NA VALVULA
HV_001.d_E	{...}	{...}		d_E_HV	VALVULA MANUAL
HV_001.d_E.KQI	0		Decimal	INT	VALVULA MANUAL
HV_001.d_S	{...}	{...}		d_S_HV	VALVULA MANUAL
HV_001.d_S.KQI	0		Decimal	INT	VALVULA MANUAL

Fonte: Do autor

A imagem 5.41 representa a lógica no CLP com o uso do "*Function Block*" transmissor analógico, a interação com as interfaces e sua utilização como um bloco de controle (A Rockwell Software denominou o "*Function Block*" como "*Add-on Instruction*").

Figura 5.41: Utilização do componente válvula de bloqueio manual



Fonte: Do autor

5.7.3 *Implementação da válvula pneumática de bloqueio*

A imagem 5.42 representa a estrutura de dados do transmissor analógico, utilizando como variável um tipo de dado definido pelo usuário *"User-Defined DataType"*, semelhante ao *"Struct"* da linguagem "C".

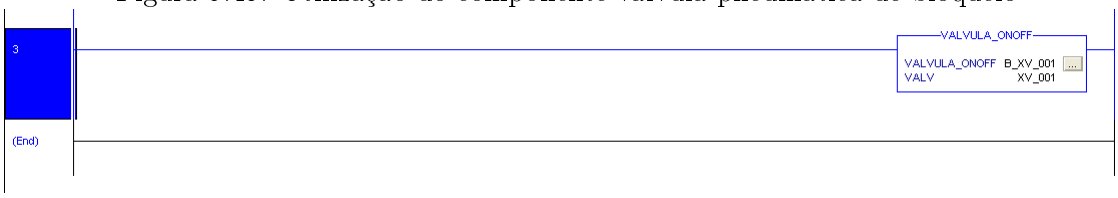
Figura 5.42: Base de dados da válvula pneumática de bloqueio no programa RsLogix5000

⊟ XV_001			VALVULA		
⊞ XV_001.e_E			e_E_XV		COMANDO DE INTERTRAVAMENTO PARA FECHAR
⊟ XV_001.e_S			e_S_XV		RECONHECER ALARME
XV_001.e_S.FABE			BOOL	Decimal	FALHA DE ABERTURA
XV_001.e_S.FFCH			BOOL	Decimal	FALHA DE FECHAMENTO
XV_001.e_S.FCOM			BOOL	Decimal	FALHA DE COMUNICAÇÃO
XV_001.e_S.AUT			BOOL	Decimal	1= AUTOMÁTICO
XV_001.e_S.MAN			BOOL	Decimal	1= MANUAL
XV_001.e_S.OPE			BOOL	Decimal	1= OPERAÇÃO
XV_001.e_S.MNT			BOOL	Decimal	1= MANUTENÇÃO
XV_001.e_S.ABE			BOOL	Decimal	1= ABERTA
XV_001.e_S.FEC			BOOL	Decimal	1= FECHADA
XV_001.e_S.LOC			BOOL	Decimal	RECONHECER ALARME
XV_001.e_S.REM			BOOL	Decimal	RECONHECER ALARME
XV_001.e_S.ABR			BOOL	Decimal	ABRINDO
XV_001.e_S.FECN			BOOL	Decimal	FECHANDO
XV_001.e_S.FF			BOOL	Decimal	VALVULA CONFIGURADA COMO FALHA FECHA
XV_001.e_S.FA			BOOL	Decimal	VALVULA CONFIGURADA COMO FALHA ABRE
⊞ XV_001.d_E			d_E_INS		ESCRITA DOS VALORES DE PARÂMETROS PARA TRATAMENTO DAS ENTRADAS ANALÓGIC
⊞ XV_001.d_S			d_S_INS		REGISTRO DE LEITURA DOS FORNECIDO PELO TRATAMENTO DE ENTRADA ANALÓGICA

Fonte: Do autor

A imagem 5.43 representa a lógica no CLP com o uso do "*Function Block*" transmissor analógico, a interação com as interfaces e sua utilização como um bloco de controle (A Rockwell Software denominou o "*Function Block*" como "*Add-on Instruction*").

Figura 5.43: Utilização do componente válvula pneumática de bloqueio



Fonte: Do autor

5.7.4 *Implementação da bomba com inversor de frequência*

A imagem 5.44 representa a estrutura de dados do transmissor analógico, utilizando como variável um tipo de dado definido pelo usuário *"User-Defined DataType"*, semelhante ao *"Struct"* da linguagem "C".

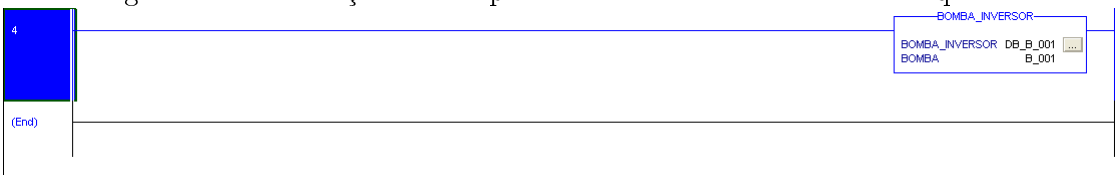
Figura 5.44: Base de dados da bomba com inversor de frequência no programa RsLogix5000

B_001	{...}	{...}		BOMBA_INV	
B_001.e_E	{...}	{...}		e_E_BO	EVENTOS DE ENTRADA
B_001.e_S	{...}	{...}		e_S_BO	EVENTOS DE SAIDA
B_001.e_S.S_FUN	0		Decimal	BOOL	BOMBA EM FUNCIONAMENTO
B_001.e_S.PPP	0		Decimal	BOOL	BOMBA PRONTA PARA PARTIR
B_001.e_S.FCOM	0		Decimal	BOOL	FALHA DE COMUNICAÇÃO
B_001.e_S.AUT	0		Decimal	BOOL	1= AUTOMÁTICO
B_001.e_S.MAN	0		Decimal	BOOL	1= MANUAL
B_001.e_S.OPE	0		Decimal	BOOL	1= OPERAÇÃO
B_001.e_S.MNT	0		Decimal	BOOL	1= MANUTENÇÃO
B_001.e_S.LOC	0		Decimal	BOOL	LOCAL
B_001.e_S.REM	0		Decimal	BOOL	REMOTO
B_001.e_S.FEC	0		Decimal	BOOL	FALHA AO ENERGIZAR O CONTATOR DA BOMBA
B_001.e_S.FDC	0		Decimal	BOOL	FALHA AO DESENERGIZAR O CONTATOR
B_001.e_S.DEF	0		Decimal	BOOL	BOMBA COM DEFEITO
B_001.e_S.XS	0		Decimal	BOOL	SAIDA DIGITAL PARA ENERGIZACAO DA BOMBA
B_001.d_E	{...}	{...}		d_E_BO	DADOS DE ENTRADA
B_001.d_S	{...}	{...}		d_S_BO	DADOS DE SAIDA

Fonte: Do autor

A imagem 5.45 representa a lógica no CLP com o uso do "*Function Block*" transmissor analógico, a interação com as interfaces e sua utilização como um bloco de controle (A Rockwell Software denominou o "*Function Block*" como "*Add-on Instruction*").

Figura 5.45: Utilização do componente bomba com inversor de frequência



Fonte: Do autor

5.7.5 *Implementação da malha de controle PID*

A imagem 5.46 representa a estrutura de dados do transmissor analógico, utilizando como variável um tipo de dado definido pelo usuário *"User-Defined DataType"*, semelhante ao *"Struct"* da linguagem "C".

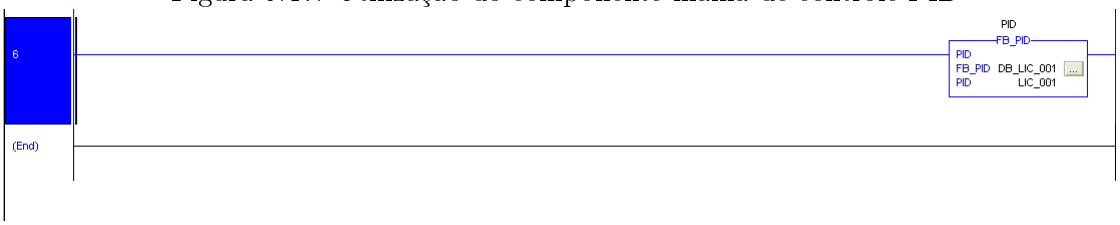
Figura 5.46: Base de dados da malha de controle PID no programa RsLogix5000

LIC_001	{...}	{...}		PID_TYPE	PID
LIC_001.e_E	{...}	{...}		e_E_PID	PID
LIC_001.e_E.OPE	0		Decimal	BOOL	PID OPERACAO
LIC_001.e_E.MNT	0		Decimal	BOOL	PID MANUTENCAO
LIC_001.e_E.AUT	0		Decimal	BOOL	PID AUTOMATICO
LIC_001.e_E.MAN	0		Decimal	BOOL	PID MANUAL
LIC_001.e_S	{...}	{...}		e_S_PID	PID
LIC_001.e_S.OPE	0		Decimal	BOOL	PID OPERACAO
LIC_001.e_S.MNT	0		Decimal	BOOL	PID MANUTENCAO
LIC_001.e_S.AUT	0		Decimal	BOOL	PID AUTOMATICO
LIC_001.e_S.MAN	0		Decimal	BOOL	PID MANUAL
LIC_001.e_S.HLM	0		Decimal	BOOL	PID LIMITE MAXIMO DA VARIAVEL MANIPULADA A...
LIC_001.e_S.LLM	0		Decimal	BOOL	PID LIMITE MINIMO DA VARIAVEL MANIOULADA AT...
LIC_001.d_E	{...}	{...}		d_E_PID	PID
LIC_001.d_E.PV	0.0		Float	REAL	PID VARIAVEL DE PROCESSO
LIC_001.d_E.SP	0.0		Float	REAL	PID SET-POINT
LIC_001.d_E.KP	0.0		Float	REAL	PID GANHO PROPORCIONAL
LIC_001.d_E.TI	0.0		Float	REAL	PID TEMPO INTEGRAL
LIC_001.d_E.TD	0.0		Float	REAL	PID TEMPO PROPORCIONAL
LIC_001.d_E.MV...	0.0		Float	REAL	PID VALOR DA VARIAVEL MANIPULADA EM MANUAL
LIC_001.d_S	{...}	{...}		d_S_PID	PID
LIC_001.d_S.MV	0.0		Float	REAL	PID VARIAVEL MANIPULADA

Fonte: Do autor

A imagem 5.47 representa a lógica no CLP com o uso do "*Function Block*" transmissor analógico, a interação com as interfaces e sua utilização como um bloco de controle (A Rockwell Software denominou o "*Function Block*" como "*Add-on Instruction*").

Figura 5.47: Utilização do componente malha de controle PID



Fonte: Do autor

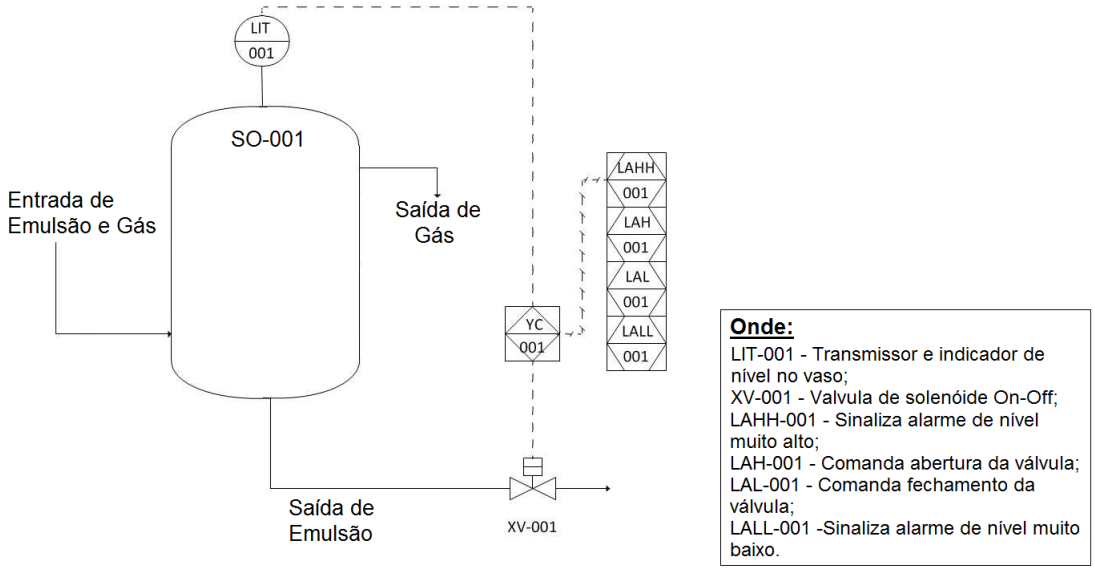
5.7.6 *Integração dos componentes*

Na figura 2.2, apresentada no capitulo 2.1.4, foi demonstrado a metodologia adotada para aplicação dos componentes. Após a qualificação e adaptação destes elementos, onde são definidos as suas interfaces e requisitos, a fase de composição é iniciada. Nesta etapa, os blocos de funções são integrados, fazendo com que haja interação entre as interfaces dos componentes. A partir deste momento, a automação do processo começará a ser alcançada, pois é através do relacionamento entre cada módulo que atingiremos o propósito de controlar a planta industrial.

No inicio deste capítulo foi apresentado o recorte de um sistema para injeção de água

em poços, com o intuito de elicitar requisitos e identificar componentes que poderiam ser modelados neste trabalho. Por questões didáticas, será apresentado - conforme a figura 5.48 abaixo - a integração dos componentes para controlar outro sistema muito encontrado nas instalações da indústria petróleo e gás, trata-se do controle On-Off de um vaso separador bifásico; composto apenas por uma válvula pneumática de bloqueio e um transmissor de nível.

Figura 5.48: Diagrama P&ID utilizado para representar um separador bifásico

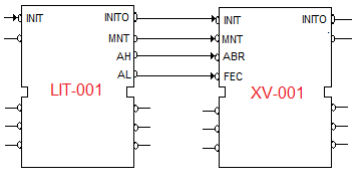


Fonte: Do autor

O funcionamento deste sistema é bastante simplório, a princípio o CLP representado por YC-001 tem apenas a função de monitorar a altura do líquido no vaso e não permitir que este extravase ou esgote, evitando assim a saída de líquido pela linha de gás, bem como a saída do gás pela linha de emulsão. Porem os requisitos técnicos exigidos para cada um dos componentes (LIT-001 e XV-001) são os mesmos apresentados no inicio deste capítulo, por meio dos diagramas típicos. Sendo assim, para atender esta composição será utilizado um bloco de função para o LIT-001 e outro para a XV-001 na lógica do CLP, e consequentemente haverá um relacionamento entre eles através de suas interfaces de eventos e dados.

A distribuição dos componentes e interfaces que serão conectadas para atender as especificações do vaso separador bifásico, estão representadas na 5.49 (Fazendo abstrações das demais funcionalidades), de maneira que a XV-001 recebe o comando para abrir sempre que atuado o alarme de nível alto e fechar quando ativado o alarme de nível baixo. Além disso, é possível observar que os eventos de saída que indicam LIT-001 em manutenção, também habilita o evento de entrada da XV-001 para manutenção.

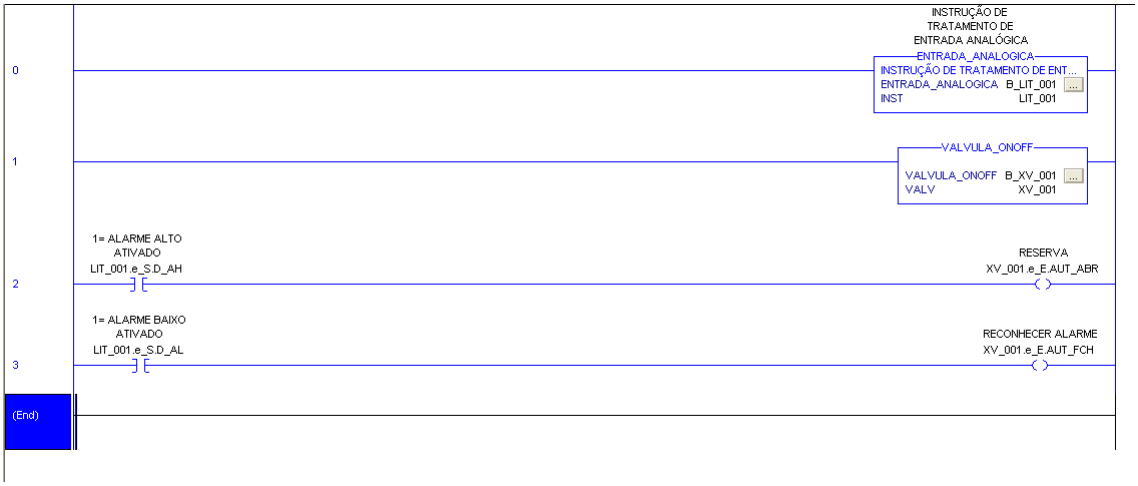
Figura 5.49: Integração dos FBs para controle do separador bifásico



Fonte: Do autor

Conforme visto na figura 3.11 pelo capítulo 3.3, por convenção, os diagramas dos FBs descritos pela IEC 61499 possuem os eventos de entrada representados na parte superior esquerda do bloco e os eventos de saída na parte superior direita. No processo de implementação ladder via software RsLogix5000 será necessário acrescentar um FB para o LIT-001 e outro para a XV-001; após isto, será necessário apenas a parametrização destes componentes em sua base de dados e interligar as suas interfaces. A figura 5.50 exemplifica a estrutura necessária para atender a automação do vaso separador.

Figura 5.50: Desenvolvimento baseado em componente aplicado em controle On-Off de vaso separador



Fonte: Do autor

Após inseridos os blocos de funções “ENTRADA ANALÓGICA” e “VALVULA_ON-OFF”, todas as condições de funcionamento serão automaticamente executadas, conforme foram descritas na modelagem destes componentes. O endereço de memória “LIT_001.e_S.D_AH” é o alarme de nível alto do vaso (LIT-001), com ele ativado iniciará a abertura da válvula XV-001, através do endereço “XV_001.e_E.AUT_ABR”. Consecutivamente, a válvula fechará quando o “LIT_001.e_S.D_AL” acionar a memória “XV_001.e_E.AUT_FCH”.

Capítulo Seis

Considerações Finais

Para avaliação dos resultados, inicialmente se faz necessário desassociar os efeitos desta modelagem do ponto de vista da engenharia de componentes com o da engenharia de aplicação, apresentadas no capitulo 2. A primeira tem como principal objetivo implementar individualmente cada bloco de função que será aproveitado no sistema, suas responsabilidades consistem em identificar a necessidade de construção e desenvolvimento, catalogar e disseminar componentes de software pertinentes a um domínio específico.
Por outro lado, a engenharia de aplicação visa propiciar o uso dos componentes de software. Suas principais responsabilidades consistem em selecionar, qualificar, adaptar, integrar e atualizar estes módulos para reutilização em novos sistemas.

A principal vantagem obtida, no uso do componentes propostos, está centralizada na forma de organização dos programas de CLPs. Tal estruturação propõe aos desenvolvedores de sistemas automatizados um fortalecimento da visão de que esta atividade poderá ser, na maior parte do tempo, aplicada à configuração dos componentes; em detrimento da programação tradicional, propriamente dita.
Investir em uma engenharia de software que valoriza a parametrização dos blocos de funções existentes e consolidados, ao invés de conceber todo o algoritmo, contribui com a redução dos custos globais de desenvolvimento. Menos artefatos de software precisariam ser especificados, concebidos, implementados e validados. Sommerville (2011) faz uma análise semelhante sobre as principais vantagens e desvantagens com o uso desta tecnologia.

6.1 Avaliação do modelo: Perspectiva da engenharia de componentes

Baseado na analise dos resultados obtidos com uso da modelagem proposta pela engenharia de componentes e observando a avaliação de Sommerville (2011), destacam-se como vantagens:

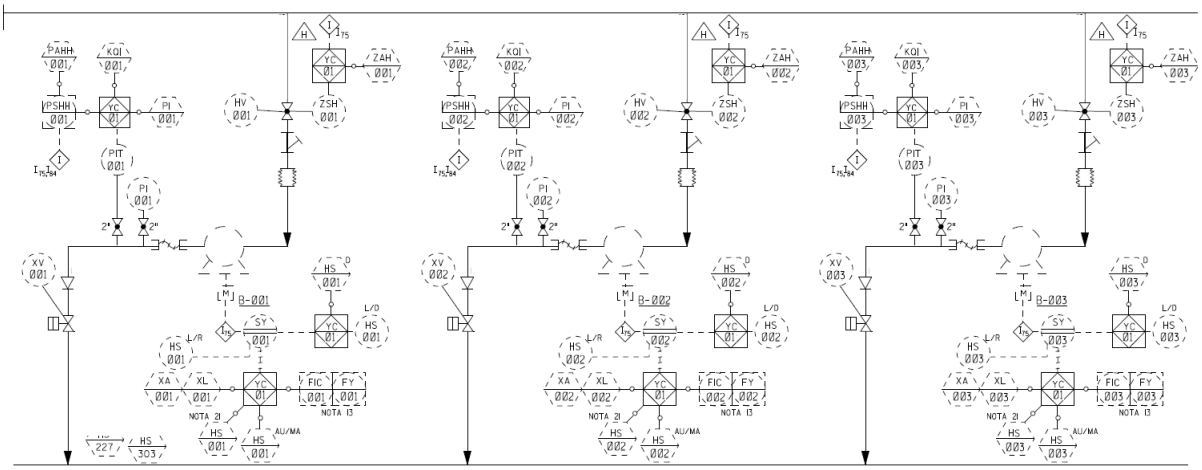
6.1.1 Confiança aumentada

O uso contínuo dos blocos de funções naturalmente ocasionará sua evolução. Por meio da experimentação e testes com sistemas em funcionamento, os componentes se tornarão cada vez mais confiáveis do que um novo software; a medida que seus defeitos de projeto e implementação forem sendo encontrados e corrigidos. Não foi mencionado explicitamente nesta dissertação, mas as versões apresentadas para os componentes transmissores analógicos, válvulas, bombas e malhas de controle foram elaboradas seguindo o modelo de processo prescritivo de software em cascata. Com exceção da etapa de implantação junto ao cliente, foram obedecidas as seguintes etapas:

- **Comunicação:** Levantamento de requisitos em colaboração com o cliente;
- **Planejamento:** Estabelece as tarefas, os riscos, os recursos, os produtos e um cronograma;
- **Modelagem:** Criação de modelos que permitam ao desenvolvedor entender melhor o projeto e seus requisitos;
- **Construção:** Geração de código e testes;
- **Implantação:** Entrega do software ao cliente.

Apesar dos componentes apresentados demonstrarem uma confiabilidade possível de ser validada e implantada em campo, esta etapa não foi aplicada em um ambiente real de controle, sendo apenas simulado em nível de programação até a fase de testes.
As empresas de exploração e produção de petróleo e gás simulam o funcionamento de novos sistemas ou modificações, principalmente devido aos riscos inerentes com a atividade. Esta simulação é denominada comumente por Testes de Aceitação em Bancada (TAB), visando garantir a integridade do software. No TAB são confeccionados os relatórios ou cadernos de testes, e validados para assegurar que os requisitos exigidos foram rigorosamente cumpridos.
Antes de iniciar a operação de um novo sistema são aplicados novos testes à lógica do CLP, dessa vez denominado como Testes de Aceitação em Campo (TAC). Após esta última etapa, inicia a fase de operação assistida visando corrigir algumas anormalidades ou eventualidades não previstas.
Para mensurar a questão "aumento da confiança", foi utilizado um relatório de testes em bancada criado para validar uma lógica programada "artesanalmente", da maneira convencional; em um sistema de injeção de água em poços, composto por: 03 bombas com inversor de frequência, 03 XVs (Válvula pneumática de bloqueio), 03 HVs (Válvula manual de bloqueio), 03 Transmissores de pressão, representado na figura 6.1.

Figura 6.1: Fluxograma de engenharia do sistema de injeção



Fonte: Do Autor

Através da programação convencional, foram encontrados os seguintes resultados:

- Ao aplicar o teste foram localizados 30 desvios em um total de 288 verificações;
- Ao repetir o teste pela segunda vez, foram novamente localizados 08 desvios em um total de 288 verificações;
- Na terceira avaliação do programa, foi conferido que as 288 verificações foram atendidas.

Este mesmo relatório de TAB foi aplicado sobre uma lógica desenvolvida utilizando o conceito dos componentes mencionados por esta dissertação. Foi analisado o resultado dos relatórios de testes em bancada, e na primeira verificação foram encontrados um total de 05 desvios; sendo estes corrigidos, foi atestado o atendimento a todas as verificações. Apesar de inicialmente as diferenças obtidas nas duas metodologias parecerem mínimas diante dos resultados desta avaliação, em aplicações reais, onde a quantidade de componentes e a complexidade das relações entre as partes são superiores a apresentada, os benefícios desta modelagem certamente serão destacados e o seu uso fortalecido. A utilização de um processo prescritivo de software incremental ou evolucionário, com direito a correções e ampliações após a sua implantação e entrega ao cliente para operação, permitirá um aumento ainda maior na confiança destes componentes.

6.1.2 Uso eficaz de especialistas

O desenvolvimento das lógicas para cada um desses equipamentos passarão a ser realizados por técnicos especializados. Profissionais que dominam tecnologias envolvidas nas malhas de controle, compressores ou bombas desenharão uma lógica para contemplar o melhor funcionamento e a proteção de cada um desses, encapsulando assim seu conhecimento e participando da evolução dos componentes. A separação e divisão das atividades entre a engenharia de componentes com a engenharia de aplicação, possui uma relação direta com a utilização eficaz dos especialistas. Sobre esta questão, alguns benefícios foram observados e relacionados a seguir:

- Em grandes projetos é comum a contratação de especialistas com domínio do conhecimento sobre o equipamento, não sendo necessariamente profissionais do segmento de automação, formando uma equipe multidisciplinar para atender as exigências do empreendimento. Esta contratação poderá ser limitada apenas aos períodos de criação e/ ou durante as atualizações dos blocos de funções;
- O dimensionamento da equipe de desenvolvimento é diretamente influenciado por esta questão. Com a definição das variáveis representadas através dos diagramas BDD do transmissor analógico, das válvulas, bombas e malha de controle; é possível antecipar de forma independente a construção da base de dados de cada componente. Sendo assim, a implementação do sistema SCADA pode ser iniciada em paralelo ou até mesmo antecipadamente à programação do CLP. No desenvolvimento convencional, esta condição não é possível;

Pode-se concluir que a utilização eficaz dos especialistas proporciona além dos ganhos econômicos, com a otimização do quadro técnico, a redução do prazo para conclusão das atividades; a medida em que tarefas poderão iniciar em momentos independentes, sendo que anteriormente eram iniciadas apenas após o término de outras.

6.2 Avaliação do modelo: Perspectiva da engenharia de aplicação

6.2.1 Risco de custo do processo reduzido

Considerando que o custo do desenvolvimento de um software é sempre uma questão que envolve julgamento, dimensionar o tamanho de uma aplicação e calcular o prazo de entrega do sistema é um importante fator para o gerenciamento do projeto. A métrica do software para elaboração de cada componente poderá ser mensurada com antecedência, porem se os blocos de funções já estiverem sido elaborados previamente e disponíveis em

uma biblioteca ou servidor na corporação, será necessário apenas calcular o tempo gasto para configuração de cada um destes elementos. Considere o exemplo a seguir:

- O tempo gasto para um programador invocar e configurar uma unidade do bloco de função para **transmissor analógico** é em média 15 minutos;
- O tempo gasto para um programador invocar e configurar uma unidade do bloco de função para **válvula de bloqueio manual** é em média 10 minutos;
- O tempo gasto para um programador invocar e configurar uma unidade do bloco de função para **válvula pneumática de bloqueio** é em média 15 minutos;
- O tempo gasto para um programador invocar e configurar uma unidade do bloco de função para **bomba com inversor de frequência** é em média 20 minutos;
- O tempo gasto para um programador invocar e configurar uma unidade do bloco de função para **malha de controle PID** é em média 20 minutos;

Em um sistema de controle contendo 80 transmissores analógicos, 50 válvulas manuais, 30 válvulas pneumáticas de bloqueio, 15 bombas e 15 malhas de controle PID; seguindo a métrica demonstrada no tópico acima, seriam necessários 2750 minutos para configurar todos os componentes deste projeto. Além das parametrizações mencionadas para estes elementos na lógica de controle, a equipe responsável pelo orçamento precisaria ainda definir qual o custo *versus* tempo necessário para integrar as interfaces virtuais de cada componente; ou seja, seria necessário dimensionar o relacionamento restante entre todos eles no programa do CLP. Certamente a modelagem proposta não consegue eliminar todos os riscos envolvidos em um orçamento, porem reduz a margem de erro na estimativa dos custos do projeto; o que é particularmente verdadeiro quando componentes de software relativamente extensos, como subsistemas, são reutilizados.

6.2.2 Conformidade com padrões

O uso dos componentes pela engenharia de aplicação assegura o atendimento aos requisitos dos padrões de projeto. Muito embora na subseção 1.1 tenha sido discutido que: "A falta de um padrão corporativo aplicado na automação industrial, além de prejudicar a administração da mesma com uma integração insuficiente, contendo pontos desconhecidos; poderá provocar perdas na produção e comprometer a gestão da tecnologia de processo em uma organização"; não é possível afirmar a inexistência de tais padrões para programação de CLPs nas indústrias de petróleo e gás, ainda que em muitos casos eles não estejam descritos explicitamente.

No capítulo 2 foi mencionado que um padrão de projeto possui quatro elementos essenciais, no qual inclui um nome, problema, solução, consequência; conhecidos como "Gangue dos quatro". Existe uma certa precariedade de padrões voltados à automação em empresas do segmento de petróleo e gás, desta forma, podemos dizer que as exigências apresentadas nos diagramas típicos das figuras 5.3, 5.21, 5.9, 5.15, 5.28 são demonstrações de como um "padrão de projeto" voltado à automação pode ser encontrado nesta cadeia produtiva, ainda que não tenha sido declarado propriamente com esta finalidade, pois não expressam de forma clara os elementos essenciais aos padrões.
Sendo assim, a modelagem proposta para cada componente nesta dissertação, não somente atende a todos os requisitos que foram exigidos nos diagramas típicos, como complementa a solução proposta por estes. A experiência dos relatórios de testes, descritas em 6.1.1, ajudou a comprovar também que o desenvolvimento baseado em componentes assimila as exigências destes padrões mais rapidamente que a programação tradicional; de maneira que após o primeiro teste de aceitação em bancada foi obtido um resultado inicial de 98,92% do atendimento ao doravante padrão.
Ao organizar as memórias de programa em dados e eventos de entrada e saída, inspirados nos conceitos da IEC 61499 e encapsulados em variáveis definidas pelo usuário, os blocos de funções passaram também a compor uma solução para o próprio padrão. Isto porque foram incorporadas condições de falhas para cada equipamento, não elicitadas inicialmente. Se os componentes forem implementados e estruturados em uma biblioteca para utilização corporativa, teremos uma reutilização dessas ideias da mesma forma como a sugerida por (ALEXANDER; SILVERSTEIN; ISHIKAWA, 1977): "Cada padrão descreve um problema no nosso ambiente e o cerne de sua solução, de tal forma que você possa usar essa solução mais de um milhão de vezes, sem nunca fazê-lo da mesma maneira".

6.2.3 Desenvolvimento acelerado

Em alguns casos, os custos gerais de desenvolvimento não são tão importantes quanto entregar um sistema operando ao mercado o mais rápido possível. Realizar uma experiência que comprove a eficiência para o uso de uma técnica, que colabore com um desenvolvimento ágil em detrimento a outra, não é algo trivial.
Considerando um experimento inicial em que seja comparado o tempo gasto por diferentes programadores, no qual cada um utilizará modelagens distintas para solucionar um mesmo problema; e outra experiência contabilizando o tempo necessário para que as duas soluções sejam desenvolvidas pelo mesmo programador; em ambos os casos, o resultado não conseguirá ser completamente preciso. Na primeira situação, por mais que um modelo se sobressaia ao outro, fatores como a experiência ou até mesmo aspectos psicológicos de cada profissional poderá influenciar nos resultados. No outro experimento, após um programador ter se debruçado ao desafio de projetar um sistema, o tempo para concluir a segunda solução - utilizando uma nova modelagem - tenderá a ser menor; pelo fato do

profissional já estar ambientado com o problema.

Ainda assim, na experiência mencionada em 6.1.1 foram computados os tempos para elaboração das duas lógicas. Foi possível observar que através do desenvolvimento baseado em componentes, o tempo necessário para finalizar a lógica do CLP foi inferior ao desenvolvimento convencional, esta diferença tenderá a ser maior na medida em que os sistemas de controle forem maiores e mais complexos. É notável que o reúso de um software pode acelerar a produção do sistema, podendo reduzir o tempo de implementação e também de validação. Porem as alternativas existentes para mensurar esta questão não são consideradas suficientemente conclusivas.

Entretanto, apesar das dificuldades desta avaliação, utilizando a modelagem proposta nesta dissertação - além de obter a vantagem óbvia e já mencionada do reúso - o tempo de desenvolvimento pode ser ainda mais reduzido se forem utilizados os recursos de importação e exportação da ferramenta RsLogix5000. Este é um recurso válido e possível de ser adotado em diferentes IDEs(*Integrated Development Environment* - Ambiente de desenvolvimento integrado), a seguir foram enumeradas duas formas de aplicação:

1. **Utilizando a opção de importação dos TAGs e comentários:** Este recurso está disponível na maioria das IDEs para programação dos CLPs baseados na IEC 61131-3. Um arquivo no formato CSV (*Comma-Separated Values* - Valores separados por vírgulas) é gerado pelo RsLogix5000. Este formato CSV é facilmente editado por softwares manipuladores de planilhas. Com essa disponibilidade, é possível elaborar toda a base de dados do programa em tempo reduzido, já que as planilhas permitem concatenação de informações e criações de fórmulas para geração automática de texto.

2. **Utilização dos arquivos de importação em formatos XML:** Assim como no caso dos documentos CSVs, é possível exportar e importar arquivos de textos descritos por linguagem baseada no XML (Formato utilizado para criação de documentos com dados organizados, de forma hierárquica). Sendo assim, toda a lógica do CLP poderá ser manipulada em XML, conforme ilustrado nas figuras 6.2 e 6.3 a seguir:

Figura 6.2: Edição da lógica através de arquivo XML

```
</Tag>
<Tag Name="SY_001" TagType="Base" DataType="BOOL" Radix="Decimal">
<Data>00</Data>
<Data Format="Decorated"><DataValue DataType="BOOL" Radix="Decimal" Value="0"/>
</Data>
</Tag>
</Tags>
<Routines>
<Routine Name="MainRoutine" Type="RLL">
<RLLContent>
<Rung Number="0" Type="N">
<Text>
<![CDATA[ENTRADA_ANALOGICA(B_LIT_001,LIT_001);]]>
</Text>
</Rung>
<Rung Number="1" Type="N">
<Text>
<![CDATA[XIC(LIT_001.e_S.OPE)XIC(LIT_001.e_S.D_AH)OTL(LY_001.CMD.ABR);]]>
</Text>
</Rung>
<Rung Number="2" Type="N">
<Text>
<![CDATA[XIC(LIT_001.e_S.OPE)XIC(LIT_001.e_S.D_AL)OTL(LY_001.CMD.FEC);]]>
</Text>
</Rung>
</RLLContent>
</Routine>
</Routines>
</Program>
</Programs>
</Controller>
</RSLogix5000Content>
```

Fonte: Do Autor

Figura 6.3: Criação das variáveis através de arquivo XML

```
<Data Format="Decorated"><Structure DataType="ENTRADA_ANALOGICA">
<DataValueMember Name="EnableIn" DataType="BOOL" Value="1"/>
<DataValueMember Name="EnableOut" DataType="BOOL" Value="0"/>
</Structure>
</Data>
</Tag>
<Tag Name="LIT_001" TagType="Base" DataType="INSTRUMENTO">
<Data>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00</Data>
<Data Format="Decorated"><Structure DataType="INSTRUMENTO">
<DataValueMember Name="PIW" DataType="DINT" Radix="Decimal" Value="0"/>
<StructureMember Name="e_E" DataType="e_E_INST">
<DataValueMember Name="R1" DataType="BOOL" Value="0"/>
<DataValueMember Name="R2" DataType="BOOL" Value="0"/>
<DataValueMember Name="OPE" DataType="BOOL" Value="0"/>
<DataValueMember Name="MNT" DataType="BOOL" Value="0"/>
<DataValueMember Name="STON" DataType="BOOL" Value="0"/>
<DataValueMember Name="STOP" DataType="BOOL" Value="0"/>
<DataValueMember Name="AHH" DataType="BOOL" Value="0"/>
<DataValueMember Name="AH" DataType="BOOL" Value="0"/>
<DataValueMember Name="AL" DataType="BOOL" Value="0"/>
<DataValueMember Name="ALL" DataType="BOOL" Value="0"/>
<DataValueMember Name="CHH" DataType="BOOL" Value="0"/>
<DataValueMember Name="CH" DataType="BOOL" Value="0"/>
<DataValueMember Name="CL" DataType="BOOL" Value="0"/>
<DataValueMember Name="CLL" DataType="BOOL" Value="0"/>
<DataValueMember Name="R3" DataType="BOOL" Value="0"/>
<DataValueMember Name="R4" DataType="BOOL" Value="0"/>
```

Fonte: Do Autor

Através dos arquivos XMLs, é possível editar os trechos apresentados nas figuras 6.2 e 6.3, acrescentando tantos componentes quanto forem necessários, conjuntamente com a declaração de suas variáveis. Este processo poderá ainda ser automatizado e proposto como alternativa para construção de uma ferramenta 'CASE' em trabalhos futuros.

6.2.4 Facilidade na manutenção

Sistemas baseados em componentes se apresentam menos sensíveis às mudanças e se mantêm num nível funcional mais adaptável e estendível do que nos sistemas tradicionais. Portanto após o período de aprendizagem e adaptação com o uso dos componentes nos sistemas de controle e realizado os treinamentos das equipes responsáveis pela manutenção, naturalmente haverá uma melhoria na localização dos defeitos e identificação de falhas; esta será proporcionada mediante a organização das variáveis e divisão do programa em blocos de funções, diminuindo assim perdas por tempos de processo parado.

Nesta modelagem, a estrutura do programa de CLP é caracterizada por uma composição destes componentes. A sua alta desassociação, permite que novos elementos sejam inseridos em tempo de execução e sem necessidade de parada do processo. Enquanto na programação convencional, normalmente é necessário mapear todas as memórias e endereços de entrada/ saída antes de serem utilizados, para que não ocorram conflitos entre as versões. Ainda na programação convencional, por não ser composta por módulos com interfaces bem definidas e estruturadas, a busca por endereçamento para total integração da automação é normalmente lenta.

6.2.5 Testes e validação do sistema

Durante o ciclo de construção do software para o CLP, após a fase de desenvolvimento, a engenharia de componentes realiza testes de funcionamento da lógica e sua validação com os requisitos exigidos. No capítulo 2 foi mencionado que para Sametinger (1997) "componentes de *software* reutilizáveis são artefatos auto-contidos, facilmente identificáveis que descrevem e/ou executam funções específicas e possui interfaces claras, documentação apropriada e uma condição de reúso definida", neste contexto a documentação de um componente também está contemplada como um artefato, e os relatórios de testes (TAB e TAC) são partes desta documentação. Sendo assim, é possível não somente reutilizar os componentes na lógica de controle como também a descrição de suas funcionalidades e seu relatório de testes e validação.

Ao ser validado e "homologado" pela engenharia de aplicação, um componente poderá descartar uma grande quantidade de repetição dos testes das suas funcionalidades, por ter sido já aprovadas em etapas anteriores. Por fim, é possível notar avanços neste quesito, ao perceber que através do reúso há uma redução no período de testes e comissionamentos para novos sistemas, seja pela agilidade na entrega da documentação contendo o roteiro dos testes ou pela possibilidade na redução do volume de validações.

6.3 Possíveis riscos observados na modelagem baseada em componentes

Sobre o ponto de vista da engenharia de componentes eventualmente alguns problemas poderão surgir, a exemplo:

6.3.1 Risco de fluxo inverso nos custos da manutenção

Mesmo após ter sido mencionado na seção 6.2.4 as vantagens e facilidades apresentadas na etapa de manutenção do software, há ainda uma possibilidade de reversão nos custos associados a esta fase. Na eventualidade de rotatividade dos especialistas ou mesmo alterações dos sistemas, poderá ocorrer um aumento do custo dependendo da forma como o conhecimento é gerenciado e explicitado na empresa, ou seja, se o funcionamento interno de cada um dos blocos de funções não forem bem documentados, com o seu código fonte acessível, os técnicos novatos necessitarão de mais tempo para compreensão das lógicas dos equipamentos. Entretanto este é um risco inerente a todas as formas de modelagens.

6.3.2 Falta de ferramentas de suporte

Em algumas ferramentas de programação para CLPs não há suporte para o desenvolvimento com reúso baseado em componentes (Blocos de funções). Pode ser difícil ou impossível integrar essas ferramentas com um sistema de biblioteca de componentes. Sendo assim, a utilização desta modelagem está estritamente condicionada a capacidade da ferramenta ao suporte da IEC 61131-3

6.3.3 Síndrome de não inventado aqui

Alguns programadores preferem reescrever componentes ou códigos, pois acreditam poder melhorá-los. Isso tem a ver, parcialmente, com aumentar a confiança e com o fato de que escrever softwares originais é considerado mais desafiador do que reusar de outras pessoas.

6.3.4 Encontrar, compreender e adaptar os componentes reusáveis

Componentes de software precisam ser descobertos em uma biblioteca, compreendidos e, às vezes, adaptados para trabalhar em um novo ambiente. Os engenheiros de aplicação

precisam estar confiantes de que encontrarão, na biblioteca, um componente, antes de incluírem a pesquisa como parte de seu processo normal de desenvolvimento.

6.4 Conclusão

Pode-se afirmar que, de fato, o uso da engenharia de software baseada em componentes colabora com a construção e com o fortalecimento dos padrões de projetos aplicados ao desenvolvimento de lógicas para os CLPs. Foi possível notar uma convergência e harmonia entre: o uso dos blocos de funções; variáveis definidas e estruturadas pelo usuário; além de princípios adotados pela IEC 61499 e pela ESBC durante as etapas de implementação dos componentes. Reunido estes conceitos, foi apresentado - utilizando os diagramas da linguagem de modelagem SYSML - um conjunto de componentes lógicos capazes de serem adotados pelas principais famílias de CLPs com suporte da IEC 61131-3. Desta forma foi possível constatar que houve a criação de um padrão de projeto composto por este conjunto de artefatos e suas documentações, e que este padrão de lógica poderá ser utilizado e evoluído em diferentes corporações, embora as discussões tenham sido voltadas à industria de petróleo e gás.

Foram realizadas avaliações qualitativas que atestaram as condições de reúso destes componentes e principalmente a escalabilidade do padrão apresentado. À primeira vista, seguir todas as recomendações propostas parecem custosas, complexas e muitas das vezes desnecessárias, entretanto elas serão implementadas uma única vez e ao serem aplicadas oferecem consistência aos projetos, deixando-os capazes de manipular uma porção crescente de trabalho ou mesmo permitindo que os sistemas estejam sempre preparados para o crescimento. Dois conceitos chaves que favoreceram o reúso foram o encapsulamento e a modularização, sempre observados pela engenharia de componentes e explorados pela engenharia de aplicação; a nível operacional, estes trazem grandes benefícios à indústria, em virtude da dinâmica dos processos e das organizações. Foi possível notar nesta abordagem que o reúso poderá ser alcançado também nas documentações dos componentes e nos testes de validação do sistema, trazendo uniformidade e garantindo a padronização tão desejada nos projetos.

A principal contribuição desta dissertação está no fortalecimento da mudança de paradigma onde, com o desenvolvimento baseado em componentes, a programação para CLPs deixará cada vez mais de ser produzida de forma artesanal e tenderá para uma produção mais escalar e padronizada. Um programa construído, usufruindo deste conceito, possibilita além do ágil desenvolvimento, eficiência nos testes de aceitação, velocidade no comissionamento e start-up de plantas, facilidade no acoplamento dos componentes com outros sistemas, padronização, maior flexibilidade e melhoria na divisão das atividades pelas equipes de projeto.

Embora tenha sido possível construir os componentes na lógica de CLP, definindo suas interfaces, em contrapartida dentro do contexto acadêmico foi possível observar uma divergência de conceitos, opiniões e abrangência dos componentes em situações reais, pelos principais escritores do assunto. Este tipo de situação dificulta a materialização do conteúdo pelo pesquisador.

Foi observado também que apesar do conceito de componentes já ser uma realidade visível na área da automação industrial, ainda existe um caminho longo a ser percorrido. A automação totalmente baseada em componentes requer uma integração em todas as camadas, onde muita das vezes a falta de informação e conhecimento nos projetos realizados por equipes multidisciplinares, faz com que hajam divergências ou componentes incompletos de uma ponta à outra na pirâmide da automação. Para que um projeto utilizando este conceito dê efetivamente certo é necessário que haja uma coesão e empenho em todas as camadas (Dispositivo, Controle, Supervisão, MES (Manufacturing Execution System), ERP (Enterprise Resource Planning)) e por todas as disciplinas (Civil, Mecânica, Elétrica, instrumentação, entre outras), sobretudo se considerado que uma aplicação para CLP não consegue operar de maneira isolada, sendo ela totalmente dependente dos elementos físicos devidamente instalados e configurados na planta industrial.

Referências Bibliográficas

ALEXANDER, Christopher; SILVERSTEIN, Murray; ISHIKAWA, Sara. *A Pattern Language: Towns, Building, Construction*. Nova Iorque: Oxford University Press, 1977.

ALLEN, Julia H. *et al. Software Security Engineering: A Guide for Project Managers*. [S.l.]: Addison-Wesley, 2008. 1a. ed.

ALVES, Carina Frota. *Seleção de Produtos de Software Utilizando uma Abordagem Baseada em Engenharia de Requisitos*. 2001. Universidade Federal de Pernambuco, Dissertação de Mestrado. Disponível em: <http://intranet.cin.ufpe.br/ler/Our%20Publications/Dissertations/CarinaAlves2001.pdf>. Acesso em: 03 Mar. 2017.

BAUER, Nanette *et al. Integration of Software Specification Techniques for Applications in Engineering*. [S.l.]: Springer, 2004.

BOURQUE, Pierre; FAIRLEY, Richard E. *SWEBOK - Guide to the Software Engineering Body of Knowledge*. [S.l.]: IEEE Computer Society, 2013. V3.0.

BRADLEY, Allen. *ControlLogix System*. 2016. Http://www.ab.com/en/epub/catalogs/12762/2181376/2416247/360807/1837516/ControlLogix-5570-Controllers.html.

BROWN, Alan W.; WALLNAU, Kurt C. *Engineering of Component-Based Systems, Component - Based Software Engineering*. 1996. Software Engineering Institute, IEEE Computer Society Press. Disponível em: <http://ieeexplore.ieee.org/abstract/document/558485/>. Acesso em: 30 Marc. 2017.

CALVO, Isidro *et al.* Control communications with dds using iec61499 service interface function blocks. *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, p. 1–4, 2010. Disponível em: <http://ieeexplore.ieee.org/document/5641207/>. Acesso em: 21 Abr. 2017.

CAMERINI, A. Chauvet J.; BRILL, M. Interface for distributed automation: Ida. *IEEE Conference on Emerging Technologies and Factory Automation*, p. 515–518, 2001. Disponível em: <http://ieeexplore.ieee.org/document/997726/>. Acesso em: 21 Abr. 2017.

CERVO, Amado Luiz; SILVA, Roberto da; BERVIAN, Pedro. *Metodologia Científica*. [S.l.]: Pearson Education - Br, 2007. 6a. ed.

CLEMENTS, Paul; BASS, Len; NORD, Robert L. *Documenting Software Architectures: Views and Beyond*. [S.l.]: Addison-Wesley Professional, 2010. 2a Ed.

COMMISSION, International Electrotechnical. *International Standard IEC61131-3*. 2003. Disponível em: <http://d1.amobbs.com/bbs_upload782111/files_31/ourdev_569653.pdf>. Acesso em: 08 Abr. 2017.

CRNKOVIC, I.; LARSSON, M. A case study: demands on component-based development. *International Conference on Software Engineering, Proceedings of the 2000*, p. 23–31, 2000. Disponível em: <http://ieeexplore.ieee.org/document/870393/>. Acesso em: 21 Abr. 2017.

DANTAS, Alexandre *et al.* Suporte a padrões no projeto de software. *XVI Simpósio Brasileiro de Engenharia de Software*, p. 450 – 455, 2013. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/sbes/2002/038.pdf>. Acesso em: 15 Abr. 2017.

DEREMER, Frank. *PROGRAMMING-IN-THE LARGE VERSUS PROGRAMMING-IN-THE-SMALL*. 1976. IEEE Transactions on Software Engineering. Disponível em: <http://www.cs.kent.edu/~durand/CS43101Fall2004/deremer.pdf>. Acesso em: 29 Mar. 2017.

DUBININ, V.; VYATKIN, Valeriy; PFEIFFER, T. Engineering of validatable automation systems based on an extension of uml combined with function blocks of iec 61499. *Robotics and Automation, IEEE*, p. 3996–4001, 2005. Disponível em: <http://ieeexplore.ieee.org/document/1570732/>. Acesso em: 21 Abr. 2017.

FEIJó, Rafael Heider Barros. Mestrado em engenharia elétrica, *Uma arquitetura de software baseada em componentes para visualização de informações industriais*. 2007. Disponível em: <ftp://ftp.ufrn.br/pub/biblioteca/ext/bdtd/RafaelHBF.pdf>. Acesso em: 31 Mar. 2017.

FRIEDENTHAL, Sanford; MOORE, Alan; STEINER, Rick. *A Practical Guide to SysML: The Systems Modeling Language*. 2008. Acesso em: 29 Mar. 2017.

GAMMA, Erich *et al. Padrões de projeto: Soluções reutilizáveis de software orientado a objeto*. São Paulo: Addison-Wesley, 2000.

GEORGINI, Marcelo. *Automação Aplicada: Descrição e implementação de sistemas sequências com PLCs*. [S.l.]: Érica Editora, 2000. 5a. ed.

GOLUBEV, Alexey. *Components*. 2004. University of Konstanz. Acesso em: 03 Mar. 2017.

GROUP, Object Management. *OMG Unified Modeling Language Specification, Version 1.5*. 2003. Disponível em: <www.rational.com/uml/resources/documentation>. Acesso em: 30 Mar. 2017.

JOHN, Karl-Heinz; TIEGELKAMP, Michael. *IEC61131-3: Programming Industrial Automation Systems*. 1995. Disponível em: <http://www.dee.ufrj.br/controle_automatico/cursos/IEC61131-3_Programming_Industrial_Automation_Systems.pdf>. Acesso em: 30 Mar. 2017.

KATZKE, Uwe; VOGEL-HEUSER, Birgit. Combining uml with iec 61131-3 languages to preserve the usability of graphical notations in the software development of complex automation systems. *IFAC Proceedings Volumes*, p. 90 – 94, 2007. Disponível em: <https://doi.org/10.3182/20070904-3-KR-2922.00016>. Acesso em: 17 Abr. 2017.

KRUCHTEN, Philippe. *Modeling Component Systems with the Unified Modeling Language - A position paper for ICSE?98 Workshop*. 1998. Software Engineering Institute, IEEE Computer Society Press. Disponível em: <http://ieeexplore.ieee.org/abstract/document/558485/>. Acesso em: 29 Mar. 2017.

LEE, Szer-Ming; HARRISON, R.; WEST, A.A. A component-based distributed control system for assembly automation. *2nd IEEE International Conference on Industrial Informatics, INDIN '04*, p. 33–38, 2004. Disponível em: <http://ieeexplore.ieee.org/document/1417298/>. Acesso em: 21 Abr. 2017.

LJUNGKRANTZ, Oscar *et al.* Formal specification and verification of industrial control logic components. *IEEE Transactions on Automation Science and Engineering*, p. 538–548, 2010. Disponível em: <http://ieeexplore.ieee.org/document/5350407/>. Acesso em: 21 Abr. 2017.

MCILROY, Malcolm Douglas. *Mass-produced Software Components*. 1968. In North Atlantic Treaty Organization Conference on Software Engineering. Disponível em: <http://www.cs.dartmouth.edu/~doug/components.txt>. Acesso em: 29 Mar. 2017.

PAREDE, Ismael Moura; GOMES, Luiz Eduardo Lemes. *Eletrônica: automação industrial*. São Paulo: Fundação Padre Anchieta, 2011.

PRESSMAN, Roger. *Engenharia de software*. [S.l.]: McGraw-Hill, 2006. 6a. ed.

REDOLFI, Giliane *et al. Especificando Informações para Componentes Reutilizáveis*. 2004. Programa de Pós-Graduação em Ciência da Computação PUCRS, Relatório Técnico. Disponível em: <http://www3.pucrs.br/pucrs/files/uni/poa/facin/pos/relatoriostec/tr041.pdf>. Acesso em: 30 Mar. 2017.

RESENDE, Ana Rubélia Mendes de Lima; CUNHA, Adilson Marques da. Mestrado Acadêmico, *Um Modelo de Processo para Seleção de Componentes de Software*. 2006.

RIBEIRO, Marcos Antônio. *Automação Industrial*. Salvador: Tek Treinamento e Consultoria, 1999.

RITALA, Tuukka; KUIKKA, Seppo. Uml automation profile: Enhancing the efficiency of software development in the automation industry. *5th IEEE International Conference*, p. 885 – 890, 2007. Disponível em: <http://ieeexplore.ieee.org/document/4384890/>. Acesso em: 17 Abr. 2017.

SAMETINGER, Johannes. *Software Engineering with Reusable Components*. 1997. Disponível em: <http://www.swe.uni-linz.ac.at/publications/pdf/TR-SE-97.04.pdf>. Acesso em: 30 Mar. 2017.

SANTIAGO, Marcos Rogério. Especialização latu sensu em gestão de tecnologia da informação, *Ensaio do SWEBOK – Software Engineering Body Of Knowledge*. 2011. Disponível em: <https://pt.scribd.com/doc/77017069/SWEBOK-traduzido>. Acesso em: 21 Mar. 2017.

SEGOVIA, Vanessa Romero; THEORIN, Alfred. *A HISTÓRIA DO CLP*. 2012. Disponível em: <http://www.control.lth.se/media/Education/DoctorateProgram/2012/HistoryOfControl/Vanessa_Alfred_report.pdf>. Acesso em: 05 Mai. 2016.

SILVEIRA, Paulo R. da; SANTOS, Winderson E. *Automação e controle discreto*. São Paulo: Érica, 1999.

SOMMERVILLE, Ian. *Engenharia de software*. [S.l.]: Pearson Education, 2011. 9a. ed.

SOUZA, Alessandro José de. Mestrado em Automação e Sistemas, *Sistema de Gerência de Informação de Processos Industriais via WEB*. 2005. Disponível em: <https://repositorio.ufrn.br/jspui/bitstream/123456789/15424/1/AlessandroJS.pdf>. Acesso em: 30 Abr. 2016.

STETTER, Rainer. Software im maschinenbau – lästiges anhängsel oder chance zur marktführerschaft? 2004. Disponível em: <http://www.software-kompetenz.de/servlet/is/21700/Stetter-SW_im_Maschinenbau.pdf?command=downloadContent&filename=Stetter-SW_im_Maschinenbau.pdf>. Acesso em: 14 Abr. 2017.

STRASSER, Thomas; ROOKER, Martijn; EBENHOFER, Gerhard. Structuring of large scale distributed control programs with iec 61499 subapplications and a hierarchical plant structure model. *IEEE International Conference on Emerging Technologies and Factory Automation*, p. 934 – 941, 2008. Disponível em: <http://ieeexplore.ieee.org/document/4638507/>. Acesso em: 21 Abr. 2017.

SZYPERSKI, Clemens Alden. *Component Software - Beyond Object-Oriented Programming*. Harlow: Addison-Wesley Professional, 1997. 1. edição.

THRAMBOULIDIS, Kleanthis. Using uml in control and automation: a model driven approach. *Industrial Informatics, 2nd IEEE International Conference*, p. 587–593, 2004. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.4726&rep=rep1&type=pdf>. Acesso em: 21 Abr. 2017.

_____. Model-integrated mechatronics-toward a new paradigm in the development of manufacturing systems. *Industrial Informatics, IEEE Transactions*, p. 54–61, 2005. Disponível em: <https://pdfs.semanticscholar.org/3f9b/998f410a878aac6591554410e0bcbd5da2ce.pdf>. Acesso em: 21 Abr. 2017.

_____. Iec 61499 vs. 61131: A comparison based on misperceptions. *Journal of Software Engineering and Applications*, p. 405–415, 2013. Disponível em: <https://www.researchgate.net/publication/256025296_IEC_61499_vs_61131_A_Comparison_Based_on_Misperceptions>. Acesso em: 21 Abr. 2017.

THRAMBOULIDIS, Kleanthis; FREY, Georg. An mdd process for iec 61131-based industrial automation systems. *IEEE 16th Conference on Emerging Technologies and Factory Automation (ETFA)*, p. 1–8, 2011. Disponível em: <http://ieeexplore.ieee.org/document/6059118/>. Acesso em: 21 Abr. 2017.

TRKAJ, K. Users introduce component based automation solutions. *Computing and Control Engineering Journal*, p. 32–37, 2004. Disponível em: <http://ieeexplore.ieee.org/document/1394900/>. Acesso em: 21 Abr. 2017.

VOGEL-HEUSER, Birgit *et al.* Evaluation of a uml-based versus an iec 61131-3-based software engineering approach for teaching plc programming. *IEEE TRANSACTIONS ON EDUCATION, VOL. 56*, p. 329 – 335, 2013. Disponível em: <http://ieeexplore.ieee.org/document/6363495/>. Acesso em: 17 Abr. 2017.

VYATKIN, Valeriy. "software engineering in factory and energy automation: State of the art review". *IEEE Transactions on Industrial Informatics*, p. 1234 – 1249, 2013. Disponível em: <http://ieeexplore.ieee.org/document/6502240/>. Acesso em: 15 Abr. 2017.

VYATKIN, V. *et al.* Oooneida: an open, object-oriented knowledge economy for intelligent distributed automation. *Industrial Informatics, IEEE Transactions*, p. 4 – 17, 2005. Disponível em: <http://ieeexplore.ieee.org/document/1300207/>. Acesso em: 15 Abr. 2017.

_____. Closed-loop modeling in future automation system engineering and validation. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, p. 17 – 28, 2009. Disponível em: <http://ieeexplore.ieee.org/document/4717249/>. Acesso em: 21 Abr. 2017.

WERNER, Bernhard. Object-oriented extensions for iec 61131-3. *Industrial Electronics Magazine*, p. 36 – 39, 2009. Disponível em: <http://ieeexplore.ieee.org/document/5352481/>. Acesso em: 15 Abr. 2017.

Modelagem de sistemas para controladores lógicos programáveis utilizando o desenvolvimento baseado em componentes e IEC-61131-3: Aplicação em indústria de petróleo e gás

Nuno César Silva Mattos

Salvador, Agosto de 2017.