

JOÃO PAULO DA SILVA FONSECA

**ANALISADOR DE PLANOS PARA SISTEMAS
AUTOMATIZADOS BASEADOS EM CLPs**



UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA

2013

JOÃO PAULO DA SILVA FONSECA

**ANALISADOR DE PLANOS PARA SISTEMAS AUTOMATIZADOS
BASEADOS EM CLPs**

Dissertação apresentada ao Programa de Pós-graduação em Engenharia Mecânica da Universidade Federal de Uberlândia, como parte dos requisitos para obtenção do título de **MESTRE EM ENGENHARIA MECÂNICA**.

Área de concentração: Mecânica dos Sólidos e Vibrações

Orientador: Prof. Dr. José Jean-Paul Zanlucchi de Souza Tavares

UBERLÂNDIA - MG

2013

F676a Fonseca, João Paulo da Silva, 1988-
2013 Analisador de planos para sistemas automatizados baseados em CLPs /
João Paulo da Silva Fonseca. - 2013.
180 f. : il.

Orientador: José Jean-Paul Zanlucchi de Souza Tavares.
Dissertação (mestrado) – Universidade Federal de Uberlândia, Programa de Pós-Graduação em Engenharia Mecânica.
Inclui bibliografia.

1. Engenharia mecânica - Teses. 2. Automação industrial - Teses. 3. Controladores programáveis - Teses. 4. TCP/IP (Protocolo de rede de computação) - Teses. I. Tavares, José Jean-Paul Zanlucchi de Souza, 1962- II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Engenharia Mecânica. III. Título.

CDU: 621

JOÃO PAULO DA SILVA FONSECA

**ANALISADOR DE PLANOS PARA SISTEMAS AUTOMATIZADOS
BASEADOS EM CLPs**

Dissertação **APROVADA** pelo Programa de
Pós-graduação em Engenharia Mecânica da
Universidade Federal de Uberlândia.

Área de concentração: Mecânica dos Sólidos e
Vibrações

Banca Examinadora:

Prof. Dr. José Jean Paul Zanlucchi de Souza Tavares – UFU – Orientador

Prof. Dr. João Carlos Mendes Carvalho – UFU

Prof. Dr. José Reinaldo Silva – USP

Uberlândia, 23 de Agosto de 2013

DEDICATÓRIA

Àqueles que me trouxeram ao mundo e dedicaram suas vidas à minha criação, meus pais. E
àqueles que fizeram o mesmo por estes, meus avós.

AGRADECIMENTOS

Sinceros agradecimentos a Deus, por abençoar a minha vida cedendo-me a oportunidade de colaborar para o desenvolvimento científico e tecnológico do meu país, por iluminar o meu caminho e por poder conviver ao lado de pessoas tão especiais.

Aos meus pais e meus avós pela minha criação e meus valores, pelo carinho e pelo suporte.

À minha namorada Regianne pelo amor, carinho, companheirismo e paciência, sobretudo durante os momentos de dificuldade. Momentos ao seu lado sempre revigoraram meu ânimo. Sua presença foi e sempre será essencial para tamanha dedicação.

A todos aqueles que aceitaram o desafio de revisar meu texto, especialmente ao Jean e à Regianne, que não mediram esforços para prestar tal ajuda.

Ao meu orientador José Jean-Paul pela oportunidade de desenvolver este trabalho, pelo companheirismo e dedicação, pela transmissão do conhecimento, pelos ensinamentos e pelos conselhos que conduziram ao meu desenvolvimento acadêmico e pessoal.

À Universidade Federal de Uberlândia e à Faculdade de Engenharia Mecânica por acreditarem neste trabalho, pelo espaço físico e pelos materiais e equipamentos disponibilizados para a realização deste projeto.

À CAPES pelo apoio financeiro, essencial para a manutenção do exercício ao longo desses dois anos.

Ao Ricardo Fortes de Miranda e ao Guilherme Ameloti, por nos receberem em seu Laboratório, cedendo parte do seu espaço para o desenvolvimento deste projeto.

Aos colegas do *Manufacturing Automated Planning Lab* (MAPL), especialmente ao Alexandre pelas orientações no que tange o desenvolvimento de software e pelo suporte na configuração de Redes, ao Guilherme Silva pela ajuda em programação, ao Rodrigo pelos conselhos na parte eletrônica, ao William pela ajuda em instalação física, e aos demais pelo companheirismo, pela amizade e pelos momentos de descontração.

FONSECA, J.P.S. **Analizador de Planos para Sistemas Automatizados baseados em CLPs**. 2013. 180 f. Dissertação de Mestrado, Universidade Federal de Uberlândia, Uberlândia.

Resumo

A aplicação de sistemas “inteligentes” em ambientes industriais vai à contramão dos fatores culturais e técnicos do “chão de fábrica”. Apesar deste tipo de sistema já ser conhecido há mais de 50 anos, sua aplicação prática em problemas industriais ainda é restrita e, muitas vezes, um desafio. Especificamente falando sobre o tipo de sistema “inteligente” denominado planejamento automático, a ruptura de linguagens é o principal obstáculo para que as soluções obtidas possam ser implementadas na prática. Via de regra, soluções de planejamento automático são caracterizadas por uma linguagem formal, denominada PDDL (*Planning Domain Definition Language*), enquanto os sistemas industriais operam com linguagens de baixo nível, tais como *Ladder* ou SFC (*Sequential Flow Chart*) aplicadas em equipamentos conhecidos como Controladores Lógicos Programáveis (CLPs). CLPs operando via *Ladder* ou SFC, além de centralizar os dados e informações referentes ao domínio, operam em *loop* fechado com uma lógica de ataque a domínios genéricos. Soluções de planejamento automático, por sua vez, abordam problemas específicos (com cenários inicial e final pré-definidos) e atendem um único caso por solução. Esta particularidade parece ser interessante para alguns casos no meio fabril, visto que a modelagem de domínios genéricos pode não prever todos os cenários possíveis de um domínio. A aplicabilidade de sistemas “inteligentes” em casos industriais está, portanto, diretamente relacionada ao desenvolvimento de aplicativos que facilitem a integração dessas linguagens discrepantes, garantindo acesso às informações, antes centralizadas no CLP, e analisando situações de domínios específicos e não mais apenas de um domínio genérico. Este trabalho apresenta um aplicativo que integra soluções de planejamento automático com CLPs e analisa a funcionalidade do CLP em sistemas com e sem este tipo de integração. Os resultados da estrutura proposta para o aplicativo de integração são apresentados usando a abordagem Cliente / Servidor OPC e Cliente / Servidor TCP/IP.

Palavras-chave: Projeto e Integração de Sistemas. Planejamento Automático. Controlador Lógico Programável. Arquitetura Cliente/Servidor OPC. Arquitetura Cliente/Servidor TCP/IP.

FONSECA, J.P.S. **Plans Parser for Automated Systems based on PLCs**. 2013. 180 f. M. Sc. Dissertation, Universidade Federal de Uberlândia, Uberlândia.

Abstract

Intelligent systems application in industrial environments goes against cultural and technical factors of the "shop floor". Although this kind of system has been developed for more than 50 years, its practical applications in industrial fields are still limited and will often be challenging. Talking specifically about automated planning, a language rupture hinders the practical implementation. In general, automated planning solutions are characterized by a formal language called PDDL (Planning Domain Definition Language), while industrial systems have a simple, centralized, closed loop and low-level language, such as Ladder or SFC (Sequential Flow Chart) based on Programmable Logic Controllers (PLCs). Automated planning solutions, in turn, address specific problems (with initial and goal scenarios predefined) and solve only a case per solution. This feature seems to be interesting in some cases, since the generic domains models cannot predict all possible scenarios of a domain. Therefore, the applicability of intelligent systems in industrial cases is directly linked to the development of applications that makes the integration between these languages easier, ensuring access to information, previously centered on PLC and analyzing specific situations and not just a generic domain. This paper presents an application that integrates automated planning solution with PLCs and analyzes the functionality of PLC with and without this kind of integration. Preliminary results of the proposed structure for the integrator application are shown using OPC and TCP/IP in Client / Server approaches.

Keywords: System Design and Integration, Automated Planning. Programmable Logic Controller. OPC Client/Server. TCP/IP Client/Server.

LISTA DE FIGURAS

Figura 2.1. Regulador de Velocidade de Watt (modificado de Ogata (2003)).....	8
Figura 2.2. Diagrama de blocos resumido de um controlador programável (baseado em Oliveira (1993))	10
Figura 2.3. Ciclo de processamento de um CLP (baseado em Moraes e Castrucci (2001)).	11
Figura 2.4. Exemplos de linguagens de programação utilizadas em CLPs (baseado em Silveira e Santos (2003))	13
Figura 2.5. Representação gráfica de um SFC (SILVEIRA; SANTOS, 2003).....	13
Figura 2.6. Diagrama <i>Ladder</i> . implementado a partir de um SFC (SILVEIRA; SANTOS, 2003)	15
Figura 2.7: Diagrama <i>Ladder</i> referente à transição “Botão de Partida”. (TAVARES, 2010)..	16
Figura 2.8: Diagrama <i>Ladder</i> referente à “Etapa Intermediária”. (TAVARES, 2010)	16
Figura 2.9: Diagrama <i>Ladder</i> referente à ação “Abre Válvula”. (TAVARES, 2010).....	16
Figura 2.10. Fluxograma para a automação de um equipamento (OLIVEIRA, 1993).....	17
Figura 2.11. Redes industriais utilizadas em cada nível da pirâmide de automação (CASSIOLATO, 2011).....	19
Figura 2.12. O modelo de referência OSI (baseado em Tanenbaum (2003)).....	22
Figura 2.13. Pilha de protocolo TCP/IP (baseado em Tanenbaum (2003))	23
Figura 2.14. Representação esquemática da arquitetura Cliente/Servidor OPC	27
Figura 2.15. Estrutura interna dos objetos no padrão OPC (FONSECA, 2002).....	29
Figura 2.16. Diagrama de classe UML para a classe <i>GradeBook</i> (DEITEL; DEITEL, 2004)	32
Figura 2.17. Diagrama de atividades da estrutura de sequência (DEITEL; DEITEL, 2004)..	32
Figura 2.18. O modelo conceitual para o planejamento clássico (GHALLAB; NAU; TRAVERSO, 2004).....	41
Figura 2.19. Estrutura de linguagens do itSIMPLE (VAQUERO et al., 2013)	44
Figura 2.20. Proposta de manufatura de um sistema de montagem de peças (PEIXOTO, 2012)	46
Figura 2.21. Esquema do projeto <i>Fuzzy-PLC Control System</i> (DUMITRU et al., 2010)	48
Figura 2.22. Sistemas de transporte usando MAS (VRBA; MARIK, 2010)	49
Figura 2.23. Integração dos componentes físicos no MAS (BARATA et al., 2001).....	50
Figura 3.1. Representação esquemática do projeto PlanPAS (traduzido de Fonseca et al. (2013)).....	54
Figura 3.2. Representação da Ação LOAD (V1 F1 LEVEL8) em XML sem pré e pós-condições	55

Figura 3.3. Representação da Ação LOAD (V1 F1 LEVEL8) em XML após a inserção de pré e pós-condições.....	56
Figura 3.4. Esquema de comunicação presente no PlanPAS (traduzido de Fonseca et al. (2013)).....	58
Figura 3.5. Funcionalidade do PlanPAS na pirâmide da automação.....	60
Figura 3.6. Diagrama de classes simplificado da <i>PLC Interface</i>	62
Figura 3.7. Servidor OPC e <i>PLC Interface</i> rodando lado a lado	63
Figura 3.8. Diagrama de classes para o <i>Parser</i> (FONSECA et al., 2013)	64
Figura 3.9. Parte do código desenvolvido para o PlanPAS na IDE do NetBeans	65
Figura 3.10. Interface do usuário do aplicativo PlanPAS.....	65
Figura 3.11. Diagrama de atividades para o usuário do PlanPAS	66
Figura 4.1. Representação esquemática do domínio utilizado para análise (adaptada de Fonseca e Tavares (2011)).....	69
Figura 4.2. Foto da bancada didática.....	70
Figura 4.3. Diagrama de casos de uso da bancada didática (FONSECA et al., 2012)	71
Figura 4.4. Diagrama de classes da bancada didática (FONSECA et al., 2012)	72
Figura 4.5. Diagrama de estados para o veículo (FONSECA et al., 2012).....	73
Figura 4.6. Diagrama de estados para o distribuidor (FONSECA et al., 2012).....	74
Figura 4.7. <i>Snapshot</i> inicial (FONSECA et al., 2012).....	75
Figura 4.8. <i>Snapshot</i> objetivo (FONSECA et al., 2012).....	75
Figura 4.9. Plano solução obtido com o planejador SGPlan6 (FONSECA et al., 2012).....	76
Figura 4.10. Interface do PlanPAS configurada para implementar este estudo de caso	80
Figura 4.11. <i>Parser</i> , <i>PLC Interface</i> e <i>Matrikon OPC Client</i> rodando lado a lado	80
Figura 4.12. Programa principal.....	82
Figura 4.13. Discretização dos sensores de nível	84
Figura 4.14. Intertravamentos	85
Figura 4.15. Programa auxiliar	86
Figura 4.16. SFC genérico do sistema real	88
Figura 4.17. Diagrama <i>Ladder</i> parcial de acordo com a abordagem clássica	90
Figura A.1. Arquitetura de comunicação para o exemplo abordado.....	115
Figura A.2. Carregando um arquivo de configuração prévia	117
Figura A.3. Servidor OPC configurado.....	117
Figura A.4. Abrir <i>PLC Interface</i> (OPCBridge).....	118
Figura A.5. Abrir arquivo de configuração do cliente OPC	118
Figura A.6. Cliente OPC configurado	119
Figura A.7. Carregar “tabela de-para”	119

Figura A.8. Abrindo arquivo de configuração da “tabela de-para”	120
Figura A.9. Carregar arquivo com <i>snapshot</i> inicial e “plano-solução”	120
Figura A.10. Abrindo arquivo de configuração do <i>snapshot</i> inicial e “plano-solução”	121
Figura A.11. Interface do PlanPAS configurada	121
Figura A.12. Saída do PlanPAS para a tarefa “Analisar”	122
Figura A.13. Acionamento da tarefa “Enviar”	122
Figura C.1. Diagrama <i>Ladder</i> completo para a abordagem com PA	127
Figura D.1. Diagrama <i>Ladder</i> completo para a abordagem clássica	135

LISTA DE TABELAS

Tabela 2.1. Requisitos de comunicação de sistemas de automação industrial	20
Tabela 2.2. Análise das métricas abordadas (PEIXOTO, 2012).....	47
Tabela 3.1. Exemplo genérico de uma tabela de relacionamento	59
Tabela 4.1. Tabela de Sensores e atuadores	77
Tabela 4.2. “Tabela de-para” simplificada para o estudo de caso analisado	78
Tabela 4.3. Relacionamento entre o parâmetro de acesso no servidor OPC e a saída digital a ser habilitada via memória auxiliar no CLP	81
Tabela 4.4. Tabela de dispositivos de campo vs endereço no módulo de E/S	89
Tabela 4.5. Tabela Etapa vs Memória de trabalho no CLP	89
Tabela 4.6. Tabela Transição vs Memória de trabalho no CLP	90
Tabela 5.1. Evolução do sistema para a abordagem utilizando PA.....	92
Tabela 5.2. Evolução do sistema para a abordagem clássica	93
Tabela 5.3. Comparativo entre os métodos de implantação utilizando PA e CLP clássica...	96
Tabela 5.4. Desempenho do sistema de acordo com as métricas analisadas.....	97
Tabela 5.5. Comparação do estado esperado para os métodos testados.....	99
Tabela 5.6. . Comparativo entre o PlanPAS e os trabalhos relacionados.....	100
Tabela B.1. “Tabela de-para” para o estudo de caso analisado	123

LISTA DE SÍMBOLOS

Σ – sistema de transição de estados

ζ – conjunto de estados

A – conjunto de ações

E – conjunto de eventos exógenos

γ – função de transição de estados

LISTA DE ABREVIATURAS

AGV	<i>Automated Guided Vehicle</i>
AIPS	<i>Artificial Intelligence Planning Systems</i>
ARP	<i>Address Resolution Protocol</i>
ATM	<i>Asynchronous Transfer Mode</i>
CLP	Controlador Lógico Programável
CD	<i>Compact Disc</i>
CNC	Comando Numérico Computadorizado
COM	<i>Component Object Model</i>
CPU	<i>Central of Processing Unit</i>
DCOM	<i>Decentralized Component Object Model</i>
DDE	<i>Dynamic Data Exchange</i>
DHCP	<i>Dynamic Host Configuration Protocol</i>
DNS	<i>Domain Name Service</i>
ERP	<i>Enterprise Resource Planning</i>
E/S	Entrada/Saída
FDB	<i>Function Diagram Block</i>
FDDI	<i>Fiber Distributed Data Interface</i>
FTP	<i>File Transfer Protocol</i>
GCC	<i>GNU Compiler Collection</i>
GM	<i>General Motors</i>
GPS	<i>General Problem Solver</i>
HART	<i>Highway Addressable Remote Transducer</i>
HSP	<i>Heuristic Search Planner</i>
HTTP	<i>Hyper Text Transfer Protocol</i>
IA	Inteligência artificial
ICMP	<i>Internet Control Message Protocol</i>
IDE	<i>Integrated Development Environment</i>
IEC	<i>International Electrotechnical Commission</i>
IHM	Interface Homem-Máquina
I/O	<i>Input/Output</i>
IP	<i>Internet Protocol</i>
IPC	<i>International Planning Competition</i>
itSIMPLE	<i>Integrated Tools Software Interface for Modeling Planning Environments</i>

Java API	<i>Java Application Programming Interface</i>
JDK	<i>J2SE Development Kit</i>
MAS	<i>Multi-agent Systems</i>
MDP	<i>Markovian Decision Process</i>
MES	<i>Manufacturing Execution System</i>
MODICON	<i>Modular Digital Controller</i>
NA	Normalmente Aberto
NF	Normalmente Fechado
OLE	<i>Object Linking and Embedding</i>
OMG	<i>Object Management Group</i>
OPC	<i>OLE for Process Control</i>
OPC AE	<i>OPC Alarm and Events</i>
OPC DA	<i>OPC Data Access</i>
OPC HDA	<i>OPC Historical Data Access</i>
OPCTI	<i>OPC Training Institute</i>
OPC XML DA	<i>OPC eXtensible Markup Language Data Access</i>
OSI	<i>Open System Interconnection</i>
PA	Planejamento Automático
PDDL	<i>Planning Domain Definition Language</i>
PI	Proporcional-Integral
PID	Proporcional-Integral-Derivativo
PlanPAS	<i>Plans Parser for Automated Systems</i>
PNML	<i>Petri Nets Markup Language</i>
POMDP	<i>Partial Observable Markovian Decision Process</i>
POP3	<i>Post Office Protocol version 3</i>
RARP	<i>Reverse Address Resolution Protocol</i>
SCADA	<i>Supervisory Control and Data Acquisition</i>
SFC	<i>Sequential Flow Chart</i>
SMTP	<i>Simple Mail Transfer Protocol</i>
SNMP	<i>Simple Network Management Protocol</i>
SOA	<i>Service Oriented Architectures</i>
SOAP	<i>Simple Object Access Protocol</i>
STRIPS	<i>Stanford Research Problem Solver</i>
TCP	<i>Transfer Control Protocol</i>
TFTP	<i>Trivial File Transfer Protocol</i>
UDP	<i>User Datagram Protocol</i>

UML	<i>Unified Modeling Language</i>
UTC	<i>Universal Time Coordinated</i>
XML	<i>eXtensible Markup Language</i>
XPDDL	<i>eXtensible Planning Domain Definition Language</i>
WSDL	<i>Web Service Description Language</i>

SUMÁRIO

CAPÍTULO I – INTRODUÇÃO	1
1.1. Objetivos	3
1.2. Justificativa	4
1.3. Estrutura da Dissertação	5
 CAPÍTULO II – FUNDAMENTAÇÃO TEÓRICA E REVISÃO DA LITERATURA	 7
2.1. Controladores Lógicos Programáveis	7
2.1.1. <i>Histórico</i>	9
2.1.2. <i>Arquitetura</i>	10
2.1.3. <i>Linguagens de programação</i>	11
2.2. Redes Industriais	18
2.2.1. <i>O modelo OSI</i>	20
2.2.2. <i>O modelo TCP/IP</i>	22
2.3. O Padrão OPC	24
2.3.1. <i>Conceitos básicos</i>	24
2.3.2. <i>Surgimento do OPC Foundation</i>	25
2.4. Programação Orientada a Objetos	30
2.4.1. <i>Programação estruturada em Java</i>	33
2.4.2. <i>Módulos em Java</i>	34
2.4.3. <i>Aplicações de redes em Java</i>	35
2.5. Planejamento Automático	38
2.5.1. <i>Histórico do planejamento automático</i>	38
2.5.2. <i>Planejamento clássico</i>	40
2.5.3. <i>O sistema itSIMPLE</i>	43
2.6. Trabalhos Relacionados	45
2.6.1. <i>SOA/MAS em sistemas de automação da manufatura</i>	45
2.6.2. <i>Lógica fuzzy em sistemas controlados por CLP</i>	48
2.6.3. <i>MAS aplicado em sistemas de transportes</i>	48
2.6.4. <i>MAS em sistemas de manufatura integrados e distribuídos</i>	49
 CAPÍTULO III – PROJETO PROPOSTO PLANPAS	 51
3.1. PlanPAS	53
3.2. Desenvolvimento do Software Aplicativo PlanPAS	60

3.2.1. <i>Desenvolvimento da PLC Interface</i>	60
3.2.2. <i>Desenvolvimento do Parser</i>	63
CAPÍTULO IV – ESTUDO DE CASO	68
4.1. Apresentação do Sistema Analisado no Estudo de Caso	68
4.2. Modelagem do Sistema Real.....	70
4.2.1. <i>Modelagem do Domínio de Planeamento</i>	70
4.2.2. <i>Modelagem do Problema</i>	74
4.3. Resposta dos Planejadores.....	76
4.4. Configuração do sistema PlanPAS	76
4.5. Configuração do CLP como Driver e Intertravamentos	81
4.6. Análise do Sistema usando a Abordagem Clássica IEC 61131-3	86
CAPÍTULO V – RESULTADOS E ANÁLISES.....	91
5.1. Resultados	91
5.2. Análise dos Resultados	94
5.3. Comparativo com os Trabalhos Relacionados.....	99
CAPÍTULO VI - CONCLUSÃO	102
REFERÊNCIAS BIBLIOGRÁFICAS.....	106
APÊNDICE A – PASSO A PASSO PARA A CONFIGURAÇÃO DO PLANPAS.....	115
APÊNDICE B – TABELA DE PARA COMPLETA.....	123
APÊNDICE C – DIAGRAMA LADDER PARA A ABORDAGEM COM PA	126
APÊNDICE D – DIAGRAMA LADDER PARA A ABORDAGEM CLÁSSICA.....	134
ANEXO A – TRABALHO FONSECA E TAVARES (2012).....	145
ANEXO B – TRABALHO FONSECA ET AL (2012).....	156

CAPÍTULO I

INTRODUÇÃO

Há quem diga que a Inteligência Artificial (IA) emergiu como área de estudo no início da década de 1940, quando McCulloch e Pitts (1943) propuseram uma rede neural artificial, cujo objetivo era simular o cérebro humano nas operações computacionais. No entanto, apenas em 1956, durante uma conferência de verão em *Dartmouth College*, nos Estados Unidos – com a participação dos estudiosos Jonh McCarthy, Marvin Minsky, Nathaniel Rochester e Claude Shannon – foi que o termo Inteligência Artificial veio a surgir oficialmente. De lá pra cá, três linhas de pesquisa distintas se destacaram em IA: a primeira ficou conhecida como linha connexionista, a segunda como linha simbólica e a última como linha evolutiva.

A linha connexionista, preocupada especificamente com a modelagem da inteligência humana, teve McCulloch e Pitts como seus principais idealizadores, especificamente no ramo que ficou conhecido com Redes Neurais. A linha simbólica segue a tradição lógica e teve em McCarthy e Newell seus principais pesquisadores. Nesta linha, destacam-se, principalmente, os Sistemas Multiagentes e os Sistemas baseados em Conhecimento, tais como os Sistemas Especialistas e Planejadores Automáticos. Os princípios dessa linha de pesquisa são apresentados no artigo "*Physical symbol systems*" de Newell (1980). Por fim, a linha evolutiva é baseada nos moldes dos mecanismos de evolução biológica e seleção natural. Nesta linha se destacam, principalmente, os Algoritmos Genéticos propostos por Holland (1975) e a Programação Evolutiva proposta por Fogel (1966).

Apesar de mais de 50 anos de desenvolvimento, os maiores avanços em IA estão presentes em simulações computacionais ou em ambientes controlados, na grande maioria das vezes, em comunidades acadêmicas, distante das incertezas e variações dos ambientes industriais ou comerciais. Isso nos leva a crer que a utilização de sistemas "inteligentes" em aplicações práticas continua, ainda, sendo um grande desafio à engenharia.

Um bom exemplo é o Planejamento Automático que, embora tenha surgido há mais de 40 anos, ainda encontra certo tipo de resistência para ser utilizado em aplicações práticas. Algumas vezes essa resistência está diretamente relacionada à própria complexidade dos sistemas em estudo, como é o caso dos sistemas de planejamento baseados nos processos de decisão de Markov (MDPs) ou ainda, dos sistemas de planejamento sobre domínios parcialmente observáveis em MDP (POMDP). Contudo, muitas das vezes o próprio posicionamento desfavorável de alguns pesquisadores e estudiosos mais conservacionistas em relação ao Planejamento Automático, visto como uma ferramenta para a inserção de “comportamento inteligente” aos sistemas e, ainda, como uma ferramenta para o auxílio ao desenvolvimento de sistemas automatizados, possui grande responsabilidade na restrição do desenvolvimento de aplicações práticas desta área da IA.

Sob o ponto de vista técnico, uma das principais razões para a baixa aplicação prática desta área da IA está fundamentada na discrepância entre as linguagens abordadas nos níveis tático e operacional. Por um lado, uma solução de planejamento automático apresenta um plano com sequências pré-estabelecidas baseadas em ações definidas previamente, como se os objetos fossem autônomos, em uma linguagem formal padronizada denominada *Planning Domain Definition Language* (PDDL) (MCDERMOTT, 1998); já a grande maioria dos sistemas reais faz uso de controladores como o CLP, caracterizado por uma linguagem de baixo nível, sendo a mais conhecida o diagrama *Ladder*, que centralizam as operações de sensoramento e acionamentos. Com o surgimento de ferramentas de modelagem de domínio de planejamento, a visualização do modelo, bem como do resultado dos planos, passou a ser facilitada. Todavia, a centralização das operações em CLPs dificulta a aplicação direta dos planos desenvolvidos.

De acordo com Moraes e Castrucci (2001) os sistemas de controle baseados em eventos atuais fazem uso de uma única sequência em *loop* para realizar as operações programadas. Nesse caso, através de um *clock* pré-definido, o equipamento varre as entradas, executa a lógica interna, para enfim, atualizar as saídas. Inicialmente, este equipamento centralizava as ligações entre sensores e atuadores. Com o desenvolvimento de novos tipos de comunicação, como o padrão OPC (OLE (*Object Linking and Embedding*) *for Process Control*), surgem redes como a *Fieldbus*, onde os sensores e atuadores são conectados num único barramento, porém, a lógica e processamento continua centralizada. Há a tendência para que atuadores e sensores passem a ser mais autônomos, porém, isso ainda está em desenvolvimento e não pode ser considerado, do ponto de vista global, uma tecnologia atuante no chão de fábrica.

Por outro lado, desde 1971 com o surgimento do *Stanford Research Institute Problem Solver* (STRIPS) (FIKES; NILSSON, 1971), sistemas baseados em lógica são criados para

solução automática de problemas. O conceito desses sistemas está na existência de um modelo do sistema real descrevendo as ações e regras de cada objeto que o compõe, um conjunto de entradas (situação inicial) e um estado desejado (situação final). Esse modelo faz uso de algoritmos de busca no modelo definido para encontrar um caminho que parta da situação inicial e atinja o estado desejado. Há diversos quesitos que podem ser considerados para encontrar o plano mais adequado como, por exemplo, o tempo de resposta, custo e restrições impostas.

Tantas discrepâncias acerca da linguagem e do método de atuação dificultam a integração entre estas duas abordagens (Planejamento Automático e Sistemas reais baseados em CLP), o que compromete o desenvolvimento de sistemas automatizados mais “inteligentes”.

1.1. Objetivos

A aplicação do Planejamento Automático em sistemas práticos baseados em CLPs consiste em um assunto com pouco estudo anterior a seu respeito. Daí surge a necessidade de uma pesquisa exploratória para realizar um levantamento bibliográfico a cerca do assunto e, a partir de um estudo de caso específico, levantar resultados qualitativos e quantitativos que permitam avaliar o desempenho do Planejamento Automático na solução de um problema de automação.

Seguindo o caráter de uma pesquisa exploratória, este projeto possui como principal objetivo o desenvolvimento de um aplicativo que integre uma solução de planejamento automático com um sistema real controlado por CLP. Deveras, este trabalho aborda a integração de soluções de planejamento automático, também denominados “planos-solução”, em um estudo de caso específico: uma bancada didática que simula um sistema real controlado por CLP e que já foi parcialmente apresentada por Fonseca e Tavares (2012), Tavares e Fonseca (2011) e Fonseca (2011).

Face à enorme distância entre a modelagem de domínios de planejamento automático e as aplicações práticas baseadas em CLPs, o objetivo desta proposta é o desenvolvimento de um aplicativo computacional capaz de aplicar as ações sequenciais de um “plano-solução” diretamente em um sistema controlado por CLP.

Este aplicativo deverá operar como uma “ponte” entre o nível tático e o nível operacional do sistema de automação. Para tanto, faz-se necessário o conhecimento a respeito de redes industriais no sentido de implementar a comunicação entre os dispositivos

de campo e dispositivos de nível tático. Como critério de aplicabilidade e validação do sistema, a solução proposta deverá ser testada em um sistema prático de fato.

Dentre diversos problemas práticos da engenharia que são passíveis de técnicas de planejamento para a sua solução, existem os problemas de distribuição logística. Um problema clássico de distribuição logística que exige técnicas de planejamento diz respeito às operações de cabotagem, incluindo carga e descarga de contêineres bem como carga e descarga de reservatórios de óleo cru, na indústria petroleira (LI et al., 2006). Para que o trabalho desenvolvido fosse de fato avaliado em um sistema prático, optou-se por utilizar uma bancada para fins didáticos, capaz de simular um sistema de distribuição de produto entre fornecedor, distribuidores e clientes, inseridos em uma cadeia de suprimentos.

Além da aplicação prática, é interessante uma comparação entre a abordagem de solução proposta com uma abordagem clássica, que provavelmente seria adotada fora do meio acadêmico. Neste sentido, os resultados obtidos com a abordagem proposta devem ser confrontados com aqueles obtidos utilizando a abordagem clássica de programação de CLPs. A comparação dos resultados qualitativos e quantitativos permitirá o levantamento do desempenho de cada método ao longo das métricas de avaliação do domínio específico.

Dessa forma, como objetivos específicos destacam-se:

- 1) Utilizar padrões de comunicação atuantes em ambientes industriais, como OPC e Ethernet TCP/IP;
- 2) Implementar a solução em uma bancada didática controlada por CLP;
- 3) Realizar testes e ensaios funcionais;
- 4) Comparar o tipo de abordagem proposta com a abordagem clássica (programação clássica de CLPs);
- 5) Avaliar o comportamento do sistema para as duas abordagens, apontando para cada etapa do desenvolvimento, aquela que obtiver a melhor performance.

1.2. Justificativa

A situação atual do setor industrial leva a crer que se torna atraente a possibilidade de se utilizar métodos de inteligência artificial para solução de seus problemas. Por outro lado, é fato que a abordagem de soluções em planejamento automático é muito pouco utilizada, principalmente no que diz respeito à solução de problemas de automação industrial baseados em CLPs.

Por muitos anos os pesquisadores da comunidade de planejamento automático utilizaram a linguagem padrão PDDL para a definição dos seus domínios de planejamento.

No entanto, os formalismos da PDDL caracterizavam-na mais como uma linguagem de programação estritamente acadêmica do que como uma linguagem de especificação e modelagem de domínios reais. Recentemente, Vaquero (2007) propôs uma nova abordagem utilizando uma representação diagramática através da *Unified Modelling Language* (UML) (OMG, 2010) para as fases de análise, especificação e modelagem de domínios através do itSIMPLE (VAQUERO, 2007). Entretanto, nunca foi abordada nenhuma forma de se aplicar os “planos-solução”, obtidos de um planejador automático, diretamente em um controlador industrial, tal como um CLP.

A integração destas duas abordagens, através de um aplicativo desenvolvido computacionalmente, promoveria a inserção de um “comportamento inteligente” a diversos tipos de sistemas industriais baseados em CLP, tais como na indústria automotiva, petroleira, sucroalcooleira, de mineração, etc. Além do que, espera-se que um sistema “inteligente” promova inúmeras vantagens ao sistema de automação industrial, tais como adaptabilidade, redução de incertezas e previsão de comportamento.

Logo, a principal ideia deste projeto é introduzir “inteligência” ao processo da manufatura; o aplicativo proposto será avaliado através de ensaios realizados com uma bancada didática de distribuição de produtos entre fornecedores, distribuidores e clientes – sendo um CLP Omron CJ1M CPU13 (OMRON CORPORATION, 2001), o equipamento de controle do sistema.

1.3. Estrutura da Dissertação

Esta dissertação apresenta o desenvolvimento de um aplicativo computacional para a integração de soluções de planejamento automático em um caso prático específico, a bancada didática que simula um sistema de distribuição de produtos apresentada parcialmente em Fonseca e Tavares (2012), Tavares e Fonseca (2011) e Fonseca (2011). A apresentação do desenvolvimento, dos resultados e das conclusões deste trabalho foi organizada neste documento da forma descrita a seguir.

No Capítulo 2, introduzem-se conceitos fundamentais para o entendimento do projeto. Além dos controladores lógicos programáveis, fala-se sobre redes industriais, sobre o padrão OPC, programação orientada a objetos e planejamento automático. Além disso, a seção 2.6 apresenta resumidamente os principais trabalhos relacionados, encontrados na literatura recente.

O sistema proposto, denominado Analisador de Planos para Sistemas Automatizados Baseados em CLPs ou *Plans Parser for Automated Systems* (PlanPAS), é apresentado no

Capítulo 3. Nele são definidos os requisitos do software, bem como a estruturação do código e as etapas necessárias para a implementação deste projeto.

O Capítulo 4 reserva o estudo de caso utilizado para aplicação do sistema proposto. As primeiras seções focam na modelagem do domínio de planejamento e os resultados obtidos pelos planejadores automáticos (“planos-solução”). Na sequência, dá-se ênfase à configuração da interface especificamente para a aplicação do problema analisado, bem como para a configuração do CLP como *driver* do ambiente e os seus intertravamentos necessários para garantir a segurança operacional do sistema.

Resultados e análises são ponderados no Capítulo 5, seguidos da conclusão e sugestões de trabalhos futuros no Capítulo 6. A dissertação contém ainda, após o Capítulo 6, as referências bibliográficas estudadas, apêndices desenvolvidos e anexos utilizados.

CAPÍTULO II

FUNDAMENTAÇÃO TEÓRICA E REVISÃO DA LITERATURA

Este projeto aborda a integração entre os níveis de planejamento e operacional de uma determinada aplicação de automação. A presente seção apresenta uma breve revisão bibliográfica das principais áreas da engenharia abordadas neste projeto. Inicialmente, a seção 2.1 apresenta as características essenciais dos CLPs, passando desde o seu surgimento, suas aplicações, até as suas linguagens tradicionais de programação, definidas pela norma IEC 61131-3. Em seguida, na seção 2.2 serão ressaltados alguns conceitos básicos relacionados à arquitetura de redes industriais, seguida por uma breve descrição a respeito do padrão de comunicação em tempo real entre equipamentos e dispositivos de campo, conhecido como padrão OPC. Na sequência, a seção 2.4 elucida alguns conceitos básicos, porém necessários, relacionados à programação orientada a objetos, abordando especificamente a linguagem Java. A seção 2.5 traz os principais conceitos acerca do planejamento automático, através de um breve histórico e o levantamento do estado da arte. Finalmente, a seção 2.6 levanta o estado da arte através da apresentação de alguns trabalhos relacionados.

2.1. Controladores Lógicos Programáveis

Um dos primeiros dispositivos automáticos criados pelo homem para automação industrial foi o regulador centrífugo de velocidade para máquinas a vapor, desenvolvido por James Watt em 1769, durante a Revolução Industrial. O esquema da Fig. 2.1 ilustra o dispositivo desenvolvido por Watt. À medida que a rotação da máquina varia, o regulador centrífugo altera a posição do carretel (1), fazendo com que o cilindro hidráulico (2) altere a abertura da válvula de vapor de água, corrigindo automaticamente a velocidade da máquina

a vapor. A velocidade a ser atingida pela máquina será função das massas $m1$ e $m2$ do regulador.

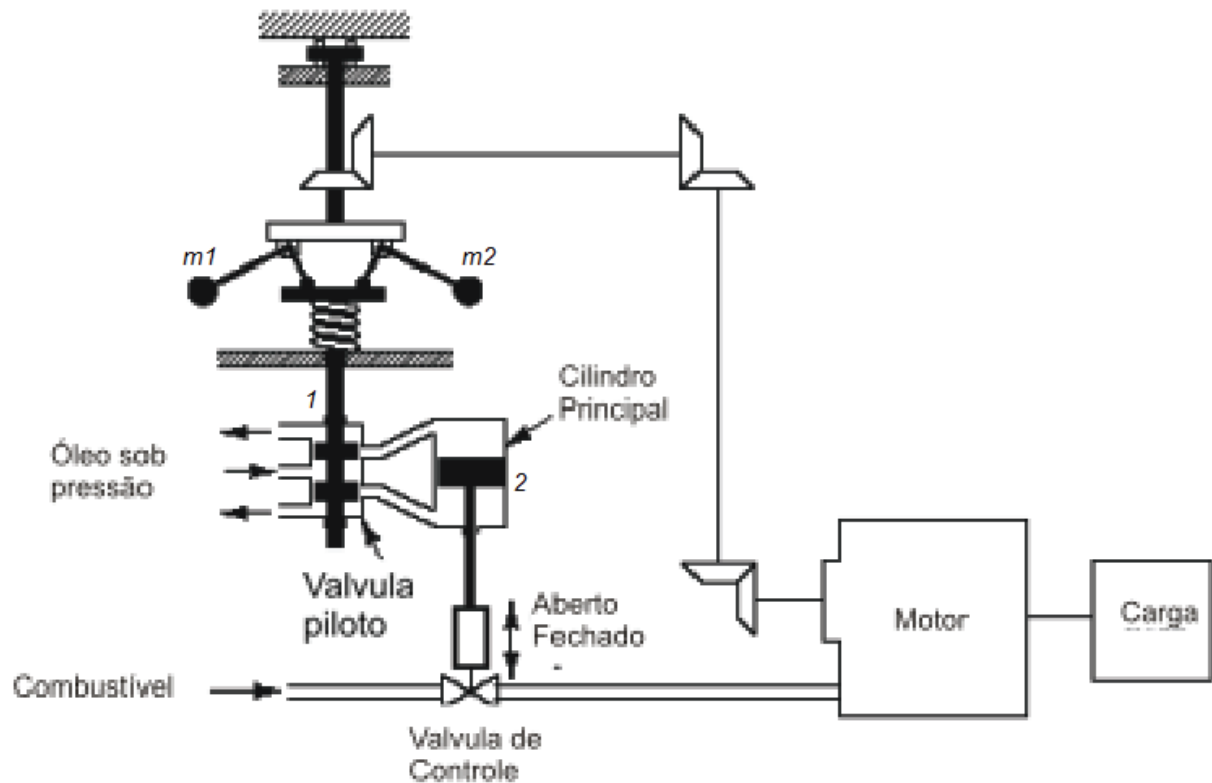


Figura 2.1. Regulador de Velocidade de Watt (modificado de Ogata (2003)).

O início do século XX foi marcado pelo surgimento de vários dispositivos capazes de controlar alguns processos simples como termostatos e chaves de nível. Nos anos 30 surgiram os primeiros controladores, e com a evolução da instrumentação todo processo tornou-se passível de ser automatizado. As primeiras aplicações de controle baseadas em computador e aquisição de dados emergiram na década de 60; os anos 80 foram marcados pelo aparecimento de sensores e atuadores “inteligentes”, robôs, máquinas a Comando Numérico Computadorizado (CNC), eficientes sistemas de supervisão, além dos protocolos de redes que permitiram a integração destes dispositivos. Impulsionado pela evolução da eletrônica, e com o aumento da capacidade de processamento e armazenamento, os dispositivos de controle atuais permitem que plantas industriais de elevada complexidade sejam controladas automaticamente, inclusive com a disponibilidade dos dados para supervisão e controle através de redes sem fio (*wireless*), internet, *tablets* e *smartphones*.

Segundo Seixas Filho (2000), o conceito embutido na palavra automação passa por uma revolução. A automação rompeu os grilhões do “chão-de-fábrica” e buscou fronteiras

mais amplas, se abrangendo a automação do negócio ao invés da simples automação dos processos e equipamentos.

2.1.1. Histórico

Em 1968 a parte lógica da linha de montagem da General Motors era composta basicamente por painéis de controle a relé. De acordo com Moraes e Castrucci (2001) os primeiros controladores surgiram baseados numa especificação da GM, resumida a seguir:

- Facilidade de programação;
- Facilidade de manutenção com conceito de plug-in;
- Alta confiabilidade;
- Dimensões menores que painéis de relés, para redução de custo;
- Envio de dados para processamento centralizado;
- Preço competitivo;
- Sinais de entrada de 115 VCA;
- Sinais de saída de 115 VCA;
- Expansão em módulos;
- Mínimo de 400 palavras na memória.

O primeiro controlador que atendia a estas especificações foi construído em 1969 pela Gould Modicon (*Modular Digital Controller*) e em 1971 sua aplicação foi ampliada a outras indústrias.

Dick Morley, o inventor dos CLPs conta que no início era normal que os usuários quisessem confrontar a robustez do novo equipamento comparando-o com a solução convencional. Era comum que os clientes quisessem comprovar se ocorria algum dano depois de provocar uma queda de dois metros no equipamento. Ou testar seu funcionamento quando próximo a um arco voltaico, produzido por uma máquina de solda industrial. No início, o CLP teve que provar que era um “bugre”, apto para as tarefas mais árduas. Com isso o equipamento sofreu muita influência do tempo para suavizar sua aparência e ser reconhecido pela sua parte interna, sua parte lógica (SEIXAS FILHO, 2000)

O primeiro sistema de comunicação desenvolvido exclusivamente para que os CLPs trocassem informação entre si foi lançado em 1973. Em 1975 passou a incorporar o controle PID (Proporcional - Integral - Derivativo), responsável por um controle contínuo mais eficiente. A partir de 1977 os microprocessadores passaram a fazer parte dos componentes de construção de CLPs; com a evolução da microeletrônica e, conseqüentemente, dos microprocessadores ocorreu o aumento da capacidade de processamento e de memória dos CLPs, o que o tornou um equipamento mais atrativo, não somente em ambiente

industrial, como também na área de automação predial atuando no controle de climatização, alarmes, iluminação, etc.

2.1.2.Arquitetura

Os CLPs são constituídos basicamente por uma fonte de alimentação, uma unidade Central de Processamento (CPU) e módulos de entrada e saída. De acordo com os valores das entradas e com a lógica armazenada na memória, a CPU realiza todo processamento necessário para definição dos valores que serão impostos às saídas. A Fig. 2.2 ilustra a estrutura básica de um CLP.

O CLP possui ainda, um terminal de programação. Este dispositivo é substituído, em geral, por um microcomputador, mas também pode ser integrado através de uma Interface Homem-Máquina (IHM) que possibilita o envio ou edição de programas, modificação de parâmetros de sintonia ou mesmo consulta aos valores de variáveis do processo, ligadas às entradas.

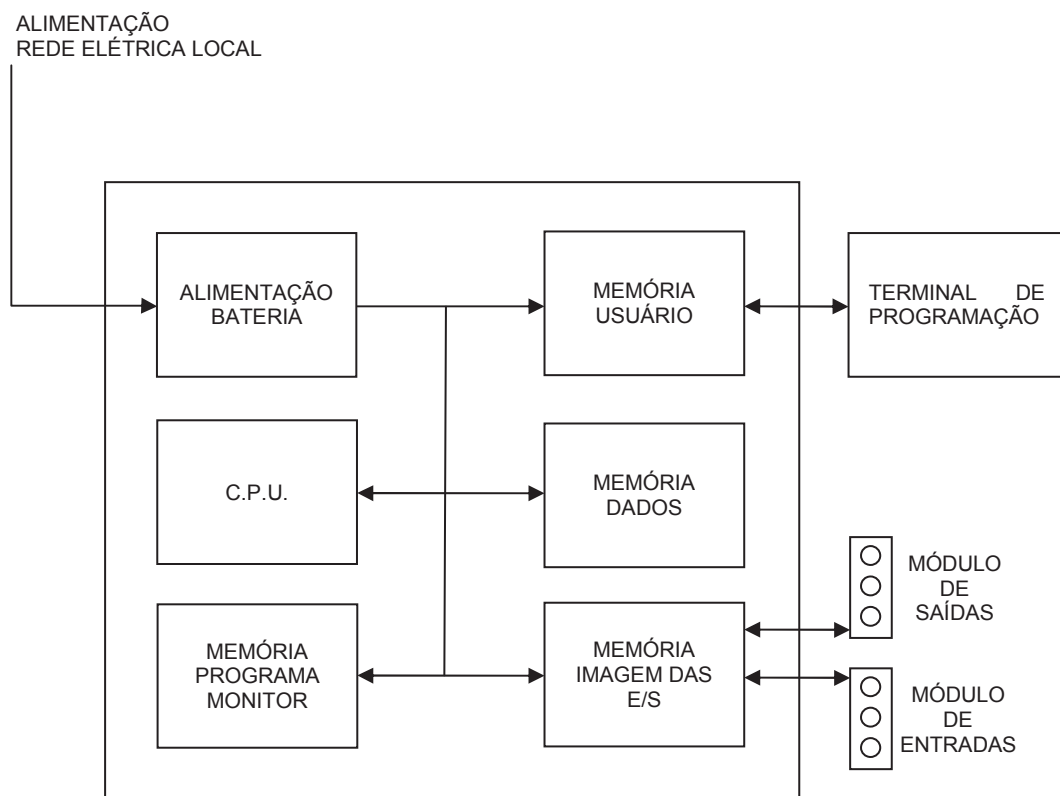


Figura 2.2. Diagrama de blocos resumido de um controlador programável (baseado em Oliveira (1993)).

O princípio de funcionamento de um CLP é definido por Moraes e Castrucci (2001) como sequencial e cíclico, a Fig. 2.3 ilustra o ciclo de processamento de um CLP de Moraes e Castrucci (2001) modificado. Inicialmente, o software interno verifica o funcionamento da CPU, memórias, circuitos auxiliares, estado das chaves, a existência de um programa de usuário, emitindo um aviso de erro em caso de falha e, por fim, desativa todas as saídas. A conclusão desta etapa habilita o início das tarefas que possibilitam o funcionamento do CLP: antes de tudo, cada uma das entradas é lida, verificando-se se houve acionamento; em seguida, os dados são transferidos para a memória e comparados com o programa do usuário; então, o CLP atualiza a memória imagem das saídas e as saídas são acionadas ou desativadas conforme a determinação da CPU (de acordo com o programa do usuário) e um novo ciclo é reiniciado. O tempo total para execução dessas tarefas, denominado ciclo de varredura, ou *scan*, depende, dentre outros fatores, da velocidade e características do processador utilizado, do tamanho do programa de controle do usuário, além da quantidade e tipo de pontos de entrada/saída. Como regra geral, tal tempo se encontra na faixa média de milissegundos (podendo chegar até microssegundos nos CLPs de última geração).

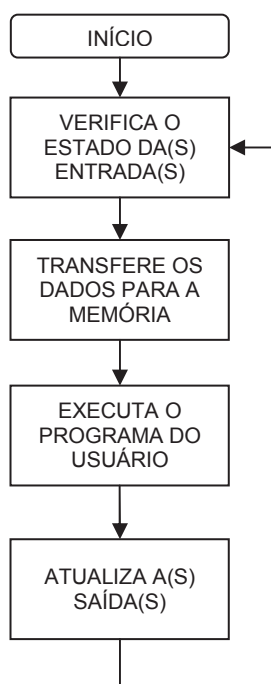


Figura 2.3. Ciclo de processamento de um CLP (baseado em Moraes e Castrucci (2001)).

2.1.3. Linguagens de programação

Do ponto de vista técnico, pode-se afirmar que o CLP surgiu devido à grande dificuldade de mudar a lógica de controle dos painéis de comando a cada mudança na linha

de montagem. Tais mudanças implicavam em altos gastos de tempo e dinheiro. Esta lógica de controle está diretamente relacionada com o tipo de linguagem que os fabricantes adotariam para programar os CLPs.

A fim de evitar a proliferação de métodos distintos de programação de equipamentos eletrônicos na área industrial, a comissão eletrotécnica internacional, em inglês *International Electrotechnical Commission* (IEC) – um grupo de trabalho responsável pela padronização de tecnologias elétricas, eletrônicas e afins – publicou, em 1993, a norma IEC 61131-3. Esta norma é o único padrão global para programação industrial, sendo que a programação de CLPs também se enquadra nesta norma. Desde então, a linguagem utilizada para a programação de qualquer CLP é basicamente uma das cinco definidas pela IEC 61131-3, que são: a linguagem *Ladder*, Diagrama de Bloco de Funções (FDB), GRAFCET ou Fluxograma Sequencial (SFC), além das linguagens textuais como Lista de Instruções e Texto Estruturado.

De acordo com Silveira e Santos (2003), um diagrama *Ladder* (ou diagrama de contatos) é a forma mais clara de apresentar uma lógica de controle aos técnicos e engenheiros eletricitas, uma vez que a própria representação dos contatos normalmente aberto (NA), normalmente fechado (NF) e relés lhes é familiar. Já um FDB tem muita semelhança com os diagramas lógicos utilizados pelos eletrônicos em seus diagramas de portas digitais. O texto estruturado é caracterizado pela sua semelhança no modo como os próprios processadores operam, através de mnemônicos booleanos, e a lista de instruções é considerada uma linguagem de programação de alto nível, com recursos capazes de solucionar problemas de maior complexidade. A Fig. 2.4 apresenta o aspecto de cada uma das formas de linguagem discutidas. No exemplo, o acionamento da saída Q1 é dado pela expressão lógica:

$$Q1 = (I0 + I1) \cdot \overline{I2} \quad (2.1)$$

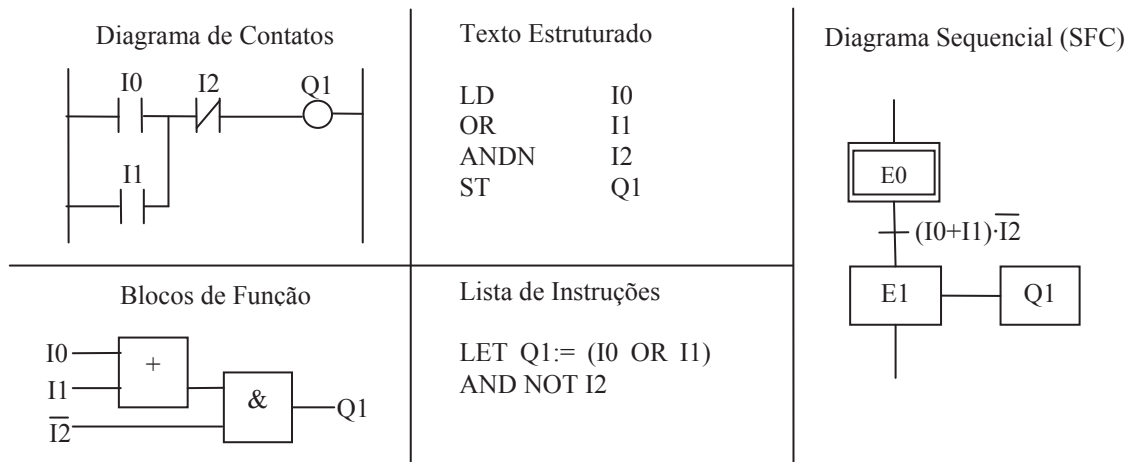


Figura 2.4. Exemplos de linguagens de programação utilizadas em CLPs (baseado em Silveira e Santos (2003)).

Comercialmente, *Ladder* e SFC são as duas linguagens mais representativas dentro do universo de linguagens para CLPs. De acordo com Miyagi (1996), o diagrama *Ladder* corresponde a uma representação lógica baseada no diagrama de circuitos de relés, cuja utilização foi generalizada antes do surgimento do CLP, o que explica a preferência por este tipo de linguagem para a maioria dos programadores. Por outro lado, o desenvolvimento do SFC foi motivado pelo interesse de ferramentas gráficas para representar explicitamente as funções para descrever os processos sequenciais para aplicações industriais.

Basicamente, o SFC é um método gráfico de apoio à concepção de sistemas industriais automatizados que permite representar, através de modelos do tipo dos gráficos de estados, o comportamento de sistemas sequenciais. A representação básica dos elementos do SFC é representada na Fig. 2.5.

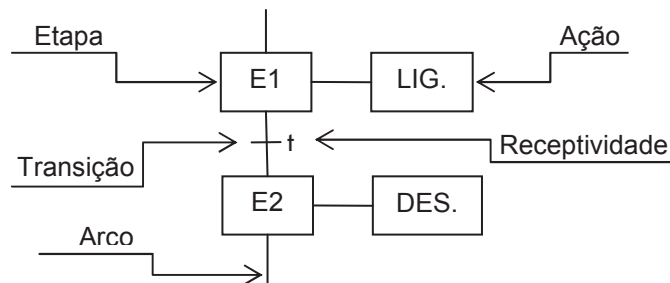


Figura 2.5. Representação gráfica de um SFC (SILVEIRA; SANTOS, 2003).

De acordo com Silveira e Santos (2003), uma compreensão clara do significado de cada uma dessas representações é imprescindível, a fim de implementar o controle

sequencial em CLPs. Neste sentido, Silveira e Santos (2003) descrevem estas representações como segue:

- Uma *etapa* significa um estado no qual o comportamento do circuito de comando não se altera frente a suas entradas e saídas. As *etapas* são representadas graficamente por um quadrilátero, e devem ser identificadas com números, seguidos ou não por abreviaturas;
- Uma *transição* é representada por um traço perpendicular aos *arcos orientados* e significa a possibilidade de evolução do SFC de uma situação para outra. A passagem de uma situação à seguinte, portanto, só é possível com a validade de uma transição que separa estas duas;
- Os *arcos orientados* indicam a sequencialização do SFC pela interligação de uma etapa a uma transição e desta a outra etapa sucessivamente. A interpretação normal é de cima para baixo, sendo que em casos diferentes deste, é recomendável a indicação com flechas para orientação de sentido;
- As *ações* representam os efeitos que devem ser obtidos sobre o mecanismo controlado em uma determinada situação. Cada ação é representada graficamente no interior de retângulos associados a uma etapa, e esta será realizada somente e apenas quando sua etapa correspondente estiver ativa;
- *Receptividade* é a função lógica combinacional associada a cada transição. Quando em estado lógico verdadeiro, uma *receptividade* vai ocasionar a ocorrência de uma transição válida. Uma *receptividade* pode então ser encarada como o elo de ligação existente entre a lógica combinacional e a lógica sequencial.

Tradicionalmente, os processos que necessitam de automação através do uso de CLPs são inicialmente modelados em SFC e posteriormente traduzidos para diagramas *Ladder*. Não que esta seja a maneira mais adequada de se programar um CLP, visto que os profissionais mais experientes realizam a programação dos CLPs em linguagem *Ladder* de forma intuitiva. Alguns CLPs disponíveis no mercado são aptos a receber diretamente o SFC sem a necessidade de tradução externa *SFC-Ladder*, no entanto, o que se vê na maioria dos casos é a utilização do diagrama *Ladder* para representar o modelo desenvolvido em SFC.

Na estrutura de um SFC, observa-se claramente a existência de três elementos básicos: as transições, as etapas e as ações. Assim, se cada um desses elementos for corretamente especificado na programação do CLP, obter-se-á como resultado uma implementação isenta de erros (desde que o SFC seja corretamente modelado). Ou seja, as

dificuldades inerentes à formação da sequência lógica na programação tornam-se transparentes, obtendo-se rapidamente uma implementação prática e funcional. (SILVEIRA; SANTOS, 2003)

No caso de se transpor um diagrama SFC em *Ladder*, o diagrama *Ladder* deve ser dividido em três partes distintas, uma responsável pela ocorrência das transições (representa em maior grau o fluxo do processo), outra responsável pelas etapas (sequencialização do processo) e, finalmente, uma que realizará as ações operativas (em função da etapa ativa), conforme ilustra a Fig. 2.6.

De acordo com Silveira e Santos (2003), para transformar um SFC em diagrama *Ladder*, é necessário criar, inicialmente, as tabelas de relacionamento entre sensores (sensibilidade do SFC) e entradas do CLP (I0, I1,... In), transições e memórias internas (R0, R1,... Rj), etapas e memórias internas (R10, R11,... R1j), e ações com saídas do CLP (O0, O1,... Ok).

As transições serão saídas (bobinas) de entradas que são a(s) etapa(s) anterior(es) (Rx) e a sensibilidade da transição. A Fig. 2.7 mostra como a transição (R0) é acionada pela etapa inicial (R10) e pelo botão de partida (S0).

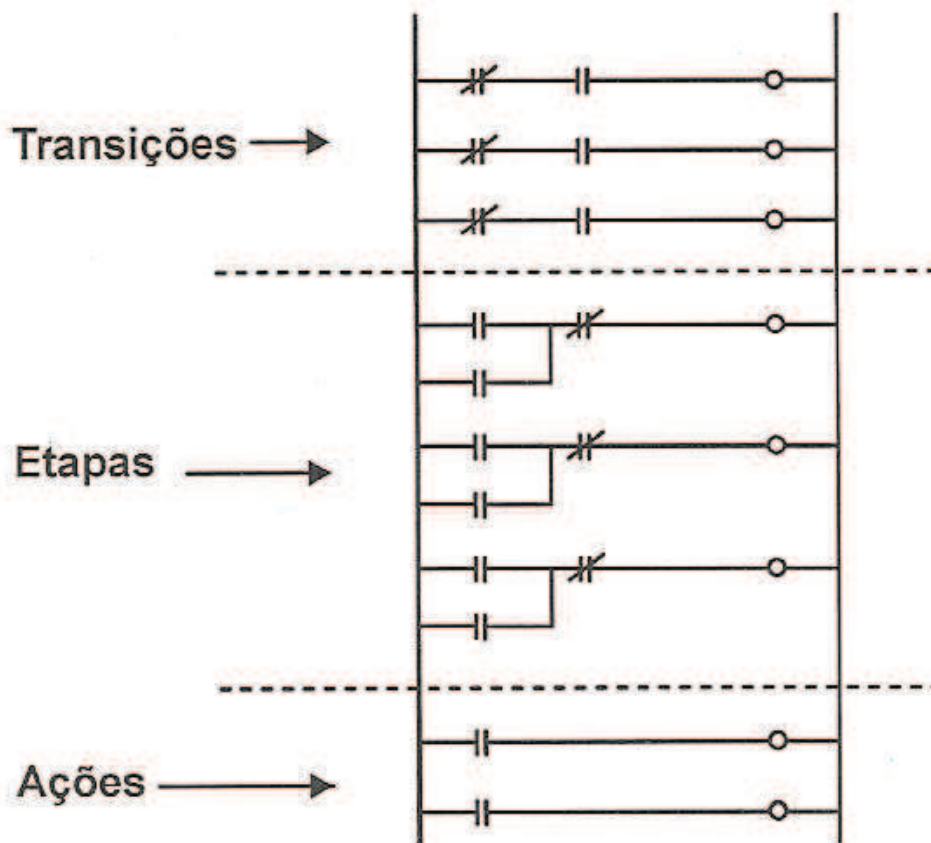


Figura 2.6. Diagrama *Ladder*. implementado a partir de um SFC (SILVEIRA; SANTOS, 2003).

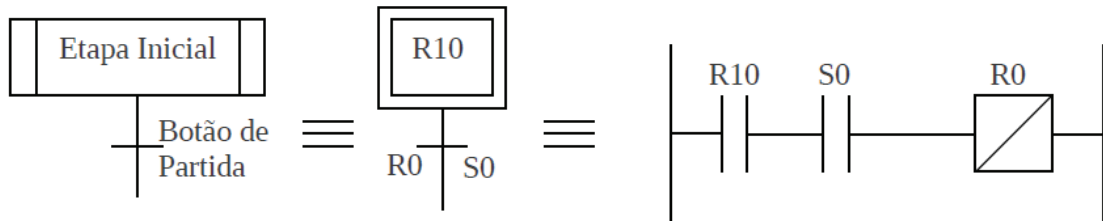


Figura 2.7: Diagrama *Ladder* referente à transição “Botão de Partida”. (TAVARES, 2010)

Em seguida devem-se montar as partes referentes às etapas, ligando-as quando a(s) transição(ões) anterior(es) for(em) válida(s) e desligando-as quando a(s) transição(ões) posterior(es) for(em) válida(s). Vale ressaltar que, para que as etapas se mantenham ativas, elas mesmas devem estar colocadas como entradas, recursivamente. A Fig. 2.8 ilustra esta representação.

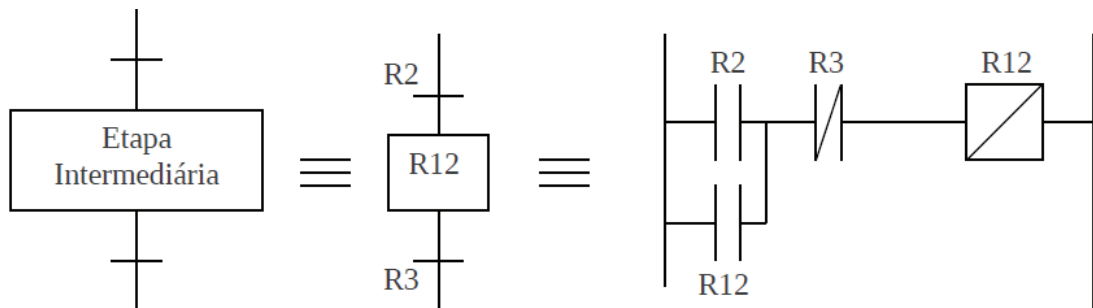


Figura 2.8: Diagrama *Ladder* referente à “Etapa Intermediária”. (TAVARES, 2010)

Finalmente, o diagrama referente às ações deverá acioná-las (bobinas) quando a etapa estiver ativa. A Fig. 2.9 serve de referência à etapa intermediária com a ação de acionar uma válvula (Saída O2 do CLP).

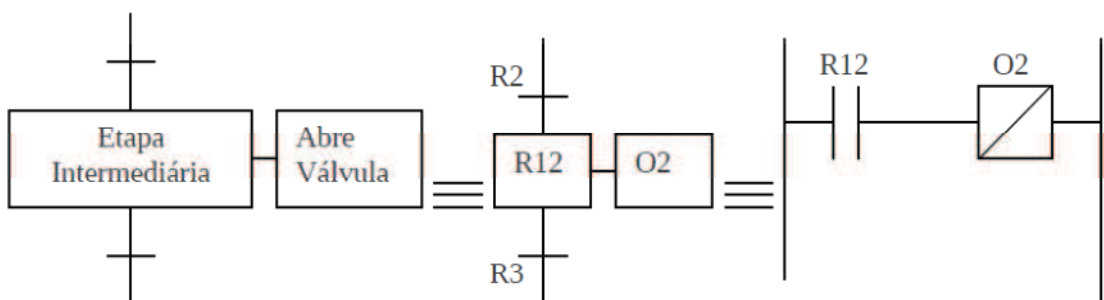


Figura 2.9: Diagrama *Ladder* referente à ação “Abre Válvula”. (TAVARES, 2010)

Oliveira (1993) propõe um método para a automação de um equipamento através do uso de CLP, a Fig. 2.10 apresenta o fluxograma proposto pelo autor. O primeiro passo é

definir quantas entradas e quantas saídas o CLP deverá ter para a automação do equipamento, bem como os atuadores e relés de interfaces necessários entre CLP e equipamento. Em seguida, deve-se projetar o programa que controlará o equipamento levando-se em consideração, além de todas as condições previstas para o funcionamento do equipamento, os intertravamentos e alertas de emergência. Após a elaboração do programa, o mesmo deverá ser submetido a um teste elétrico (sem a operação do equipamento), simulando todas as condições como se o equipamento estivesse operando.

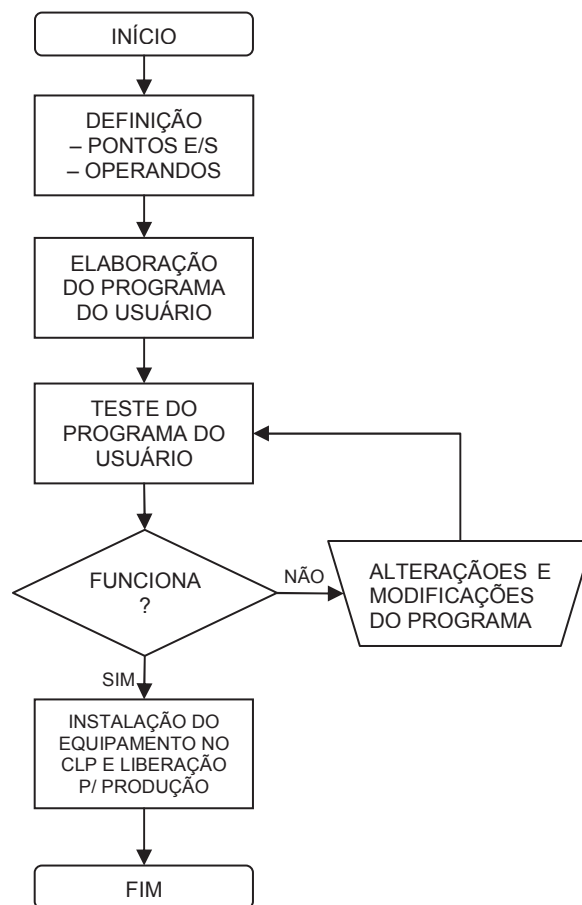


Figura 2.10. Fluxograma para a automação de um equipamento (OLIVEIRA, 1993).

Após os testes, antes de prosseguir para a próxima etapa, deve-se verificar se o sistema está operando corretamente. Caso o teste não forneça os resultados esperados, devem-se realizar alterações no programa ou projetar um mais eficiente, levando em conta o problema de controle que o anterior não solucionou. Somente após todas estas etapas, é que se deve instalar o equipamento e liberá-lo para a produção. O autor aconselha ainda, a preparação de uma listagem do programa, descrevendo linha a linha as instruções e

operações das condicionantes e das saídas, além do arquivamento desta listagem próximo ao CLP para manutenção ou alterações futuras.

2.2. Redes Industriais

Como discutido na seção 2.2, os primeiros sistemas de automação industrial surgiram durante a revolução industrial, mais precisamente no século XIX. De acordo com Cassiolato (2011), naquela época as funções de controle eram implementadas através de dispositivos mecânicos que automatizavam algumas tarefas críticas e repetitivas. Estes dispositivos eram desenvolvidos para cada tarefa e devido à natureza mecânica dos mesmos, tinham vida útil reduzida e alta manutenção.

Na década de 40 o processo de transmissão e controle de sinais em ambiente industrial evoluiu para a transmissão pneumática, caracterizada por um controle puramente analógico e capaz de transmitir uma única informação por vez. Posteriormente, com o advento dos relés e contadores, os dispositivos automáticos em linhas de montagens contaram com um grande avanço na época – a lógica a relés viabilizou o desenvolvimento de funções de controle mais complexas e sofisticadas. Esse tipo de transmissão predominou até a década de 80, quando os avanços da eletrônica trouxeram o padrão 4-20mA, responsável por grande parte do processo de transmissão até os dias atuais.

Segundo Cassiolato (2011), a transmissão eletrônica emergiu principalmente para suprir as necessidades de transmissão e controle dos sensores e atuadores microprocessados “inteligentes”. Como o avanço da eletrônica foi muito rápido, o processo de transmissão e controle teve que acompanhar esta evolução. Assim que os dispositivos de campo passaram a trabalhar com sinais digitais, as principais empresas passaram a discutir formas de padronizar o processo de transmissão de sinais através de um conjunto de regras específicas, denominado protocolo de comunicação. O primeiro protocolo de comunicação para dispositivos de campo foi o HART (da língua inglesa, *Highway Addressable Remote Transducer*), cuja principal contribuição foi a possibilidade de transmissão de mais de uma informação, ao mesmo tempo, dos dispositivos de campo. Os protocolos industriais de campo mais conhecidos atualmente são os da família *Fieldbus*, principais responsáveis pelo surgimento das redes digitais de campo dentro do processo industrial.

Do ponto de vista corporativo, as redes industriais têm aplicação não apenas no nível dos dispositivos de campo, como também é extremamente útil na integração entre os níveis de controle, supervisão, planejamento da produção e gerenciamento de recursos. A Fig.

2.11 ilustra os tipos de redes utilizadas em ambiente industrial de acordo com os níveis da pirâmide da automação.

De acordo com Cassiolato (2011), no nível de atuadores/sensores os dados têm volume de bits e são transmitidos ciclicamente, de uma maneira extremamente eficiente e rápida, na ordem de milissegundos. No nível de campo, os módulos de Entrada/Saída (E/S), transdutores, *drivers*, de operação, comunicam-se com sistemas de automação via um eficiente sistema de comunicação em tempo real (PROFIBUS-DP ou PA, *Foundation Fieldbus*, HART, etc.) com transmissão de dados do processo (na casa dos Bytes) efetuada ciclicamente, na ordem de milissegundos. No nível de célula, os controladores programáveis, tais como CLPs e PCs comunicam-se uns com os outros, o que requer grandes pacotes de dados (na ordem de *Kbytes*) e um maior tempo de transmissão. Já no nível de gerenciamento, os *mainframes* e *Workstations* comunicam-se entre si através de pacotes de dados da ordem de *Mbytes* e com tempo de transmissão na ordem de horas ou minutos. Os requisitos de comunicação para os diversos níveis industriais são apresentados na Tab. 2.1.

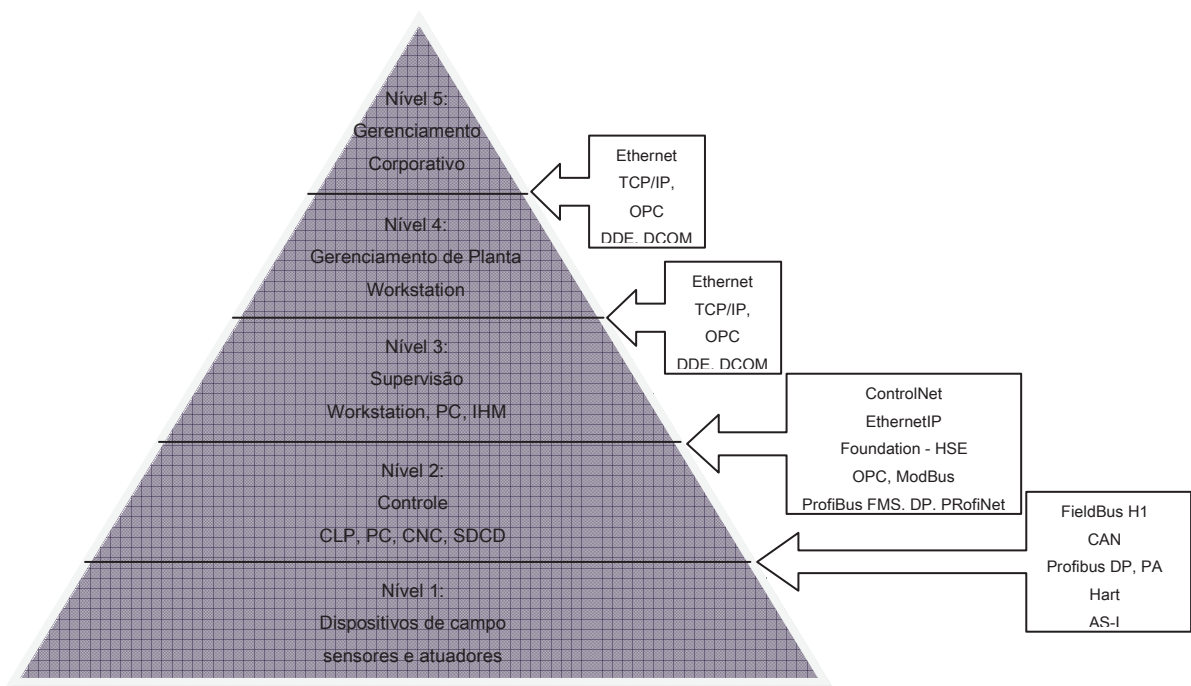


Figura 2.11. Redes industriais utilizadas em cada nível da pirâmide de automação (baseado em Cassiolato (2011)).

Tabela 2.1. Requisitos de comunicação de sistemas de automação industrial (CASSIOLATO, 2011).

	Volume de dados	Tempo de transmissão	Frequência de transmissão
Nível de Gerenciamento	<i>Mbytes</i>	Hora/Minuto	Dia / Turno
Nível de Célula	<i>Kbytes</i>	Segundos	Horas / Minutos
Nível de campo	<i>Bytes</i>	100 μ s ... 100ms	10 ... 10ms
Nível de sensores	<i>Bits</i>	1 ... 10ms	milissegundos

As próximas subseções apresentam rapidamente dois modelos de referência essenciais para o entendimento de aplicações através de camadas. De acordo com Tanenbaum (2003), embora os protocolos associados ao modelo OSI raramente sejam usados nos dias de hoje, o modelo em si é de fato bastante geral e ainda válido, e as características descritas em cada camada ainda são muito importantes. O modelo TCP/IP tem características opostas: o modelo propriamente dito não é muito utilizado, mas os protocolos têm uso geral. A subseção 2.3.1 apresenta o modelo de referência OSI enquanto o modelo de referência TCP/IP é apresentado na subseção 2.3.2.

2.2.1. O modelo OSI

De acordo com Tanenbaum (2003) o modelo é denominado Modelo de Referência OSI (*Open Systems Interconnection*), pois ele trata da interconexão de sistemas abertos — ou seja, sistemas que estão abertos à comunicação com outros sistemas. O modelo OSI (exceto o meio físico) é mostrado na Fig. 2.12.

O modelo OSI contém sete camadas, sendo elas (TANENBAUM, 2003):

- Camada física: a camada física trata da transmissão de bits brutos por um canal de comunicação. Nessa situação, as questões de projeto lidam em grande parte com interfaces mecânicas, elétricas e de sincronização, e com o meio físico de transmissão que se situa abaixo da camada física.
- Camada de enlace de dados: a principal tarefa da camada de enlace de dados é transformar um canal de transmissão bruta em uma linha que pareça livre de erros de transmissão não detectados para a camada de rede. Para executar essa tarefa, a camada de enlace de dados faz com que o transmissor divida os dados de entrada em quadros de dados (que, em geral, têm algumas centenas ou alguns milhares de *bytes*), e transmita os quadros sequencialmente. Se o serviço for confiável, o receptor confirmará a recepção correta de cada quadro, enviando de volta um quadro de confirmação.

- Camada de rede: a camada de rede controla a operação da sub-rede. Uma questão fundamental de projeto é determinar a maneira como os pacotes são roteados da origem até o destino. As rotas podem se basear em tabelas estáticas, "amarradas" à rede e raramente alteradas. Elas também podem ser determinadas no início de cada conversação; por exemplo, uma sessão de terminal (como um *logon* em uma máquina remota). Por fim, elas podem ser altamente dinâmicas, sendo determinadas para cada pacote, com o objetivo de refletir a carga atual da rede.
- Camada de transporte: A função básica da camada de transporte é aceitar dados da camada de sessão, dividi-los em unidades menores caso necessário, repassar essas unidades à camada de rede e assegurar que todos os fragmentos chegarão corretamente à outra extremidade. Além do mais, tudo isso deve ser feito com eficiência e de forma que as camadas superiores fiquem isoladas das inevitáveis mudanças na tecnologia de hardware.
- Camada de sessão: A camada de sessão permite que os usuários de diferentes máquinas estabeleçam sessões entre eles. Uma sessão oferece diversos serviços, inclusive o controle de diálogo (mantendo o controle de quem deve transmitir em cada momento), o gerenciamento de símbolos (impedindo que duas partes tentem executar a mesma operação crítica ao mesmo tempo) e a sincronização (realizando a verificação periódica de transmissões longas para permitir que elas continuem a partir do ponto em que estavam ao ocorrer uma falha).
- Camada de apresentação: Diferentemente das camadas anteriores, que se preocupam principalmente com a movimentação de bits, a camada de apresentação está relacionada à sintaxe e à semântica das informações transmitidas. Para tornar possível a comunicação entre computadores com diferentes representações de dados, as estruturas de dados a serem intercambiadas podem ser definidas de maneira abstrata, juntamente com uma codificação padrão que será usada durante a conexão. A camada de apresentação gerencia essas estruturas de dados abstratas e permite a definição e o intercâmbio de estruturas de dados de nível mais alto (por exemplo, registros bancários).
- Camada de aplicação: A camada de aplicação contém uma série de protocolos comumente necessários para os usuários. Um protocolo de aplicação amplamente utilizado é o HTTP (*HyperText Transfer Protocol*), que constitui a base para a *World Wide Web*. Quando um navegador deseja uma página da

Web, ele envia o nome da página desejada ao servidor, utilizando o HTTP. Então, o servidor transmite a página de volta. Outros protocolos de aplicação são usados para transferências de arquivos, correio eletrônico e transmissão de notícias pela rede.

APLICAÇÃO	Serviços de rede e aplicações
APRESENTAÇÃO	Representação dos dados
SESSÃO	Comunicação entre dispositivos da rede
TRANSPORTE	Conexão ponto a ponto e confiança dos dados
REDE	Endereçamento de rota e IP (endereçamento lógico)
ENLACE	Endereçamento físico e transmissão de quadros
FÍSICA	Interface mecânica e transmissão de bits

Figura 2.12. O modelo de referência OSI (baseado em Tanenbaum (2003)).

2.2.2. O modelo TCP/IP

O TCP/IP consiste basicamente em uma família de protocolos desenvolvidos para interconectar e compartilhar dispositivos através de redes, independentemente da configuração, do fabricante, do modelo ou sistema operacional da máquina.

Segundo Stevens (1994), as principais características do TCP/IP são:

- Trata-se de um padrão aberto, livremente disponível e desenvolvido independentemente do hardware ou sistema operacional do computador;
- É um padrão independente de qualquer hardware específico de rede;
- Possui um esquema de endereçamento comum que possibilita todo dispositivo comunicar-se a outro em toda rede, mesmo que esta seja tão grande quanto a Internet;
- É composto por protocolos padronizados de alto nível para serviços consistentes e amplamente disponíveis.

O nome TCP/IP pode gerar certa confusão, dado que não se trata de um único protocolo, ou do relacionamento somente entre os dois presentes no nome. Na verdade, o TCP/IP trata-se de um conjunto de diversos protocolos, tais como: TCP (*Transmission control Protocol*), UDP (*User Datagram Protocol*), ARP (*Address Resolution Protocol*), RARP (*Reverse Address Resolution Protocol*), DHCP (*Dynamic Host Configuration Protocol*), ICMP

(*Internet Control Message Protocol*), DNS (*Domain Name Service*), FTP (*File Transfer Protocol*), TFTP (*Trivial File Transfer Protocol*), HTTP (*HyperText Transfer Protocol*), POP3 (*Post Office Protocol version 3*), SMTP (*Simple Mail Transfer Protocol*), SNMP (*Simple Network Management Protocol*) e IP (*Internet Protocol*), que em conjunto facilitam a conexão e o compartilhamento entre dispositivos em uma rede – maiores informações a respeito dos protocolos que compõem o TCP/IP podem ser obtidas em Tanenbaum (2003).

Uma rede trabalhando com TCP/IP utiliza apenas quatro das camadas do modelo OSI, como ilustrado na Fig. 2.13.

Aplicação	Tarefas e aplicações com comunicação entre os dispositivos da rede
Transporte	Conexão ponto a ponto e confiança dos dados (TCP e UDP)
Rede	Endereçamento de rota, nº de porta e IP (endereçamento lógico)
Acesso à Rede	Interface mecânica e transmissão de pacotes IP

Figura 2.13. Pilha de protocolo TCP/IP (baseado em Tanenbaum (2003)).

Para Tanenbaum (2003), as quatro camadas do modelo TCP/IP podem ser descritas como segue:

- **Acesso à Rede:** Permite o transporte de dados pelos diversos tipos de redes (X.25, ATM, FDDI, *Ethernet*, *Token Ring*, *Frame Relay*, etc.). Como há uma grande variedade de soluções utilizando diferentes velocidades, meios de transmissão, etc., esta camada não é normatizada pelo modelo, o que provê uma das grandes virtudes do TCP/IP, que é a possibilidade de conexão e interoperação de redes heterogêneas.
- **Rede:** Neste nível, todos os dados são tratados como pacotes. É responsável pelo envio e roteamento dos pacotes entre os *hosts*, ou seja, tem por função encontrar o caminho entre os computadores, função esta exercida pelo protocolo IP.
- **Transporte:** Oferece a comunicação mais confiável entre os “hosts” de uma rede. Caso algum pacote se perca, é obrigação da camada enviar um pedido de retransmissão ao emissor. O TCP aloca essas informações em cada pacote e o IP os envia, quando chegam ao destino, o TCP é quem vai colocá-los em ordem. Outra importante função é o controle de fluxo evitando-se congestionamentos na rede.

- **Aplicação:** Aqui se incluem as aplicações, os programas. Quando se faz uma requisição de uma página da Internet, por exemplo, o navegador processa os pacotes que chegam e monta a página para visualização. Isso não ocorre apenas com o destino, ou seja, para receber essas informações outro programa na origem (também na camada de aplicação) teve que processá-las inicialmente.

O TCP/IP foi inicialmente desenvolvido para aplicações domésticas, em redes de escritórios, distante das irregularidades presentes em ambientes industriais. No entanto, este modelo foi muito bem aceito e se adequou rapidamente à indústria. A rede ethernet faz uso do TCP/IP e é hoje uma das principais redes encontradas nas indústrias ao redor do planeta, desde o nível operacional até o nível estratégico da pirâmide da automação.

2.3. O Padrão OPC

2.3.1. Conceitos básicos

Antes de se entender como surgiu a OPC *Foundation* e o Padrão OPC, deve-se ter o entendimento de alguns conceitos fundamentais. De acordo com Cândido (2004):

COM (*Component Object Model*) é um padrão criado pela Microsoft© para que uma aplicação cliente possa acessar os serviços de uma “aplicação servidor”. O servidor implementa seus serviços como “objetos” que podem ser acessados pelos clientes. COM especifica as interfaces para clientes e para a comunicação padronizada. A interface é uma coleção de métodos ou funções e procedimentos relacionados que implementam os serviços específicos providos por COM.

DCOM (*Decentralised Component Object Model*) provê os mesmos serviços do COM, porém utiliza como meio de acesso uma rede de comunicação de dados.

OLE (*Object linking and Embedding*): “*object*” é uma unidade de informação que pode ser criada e manipulada pelos usuários. Em outras palavras, objetos são módulos de dados/software que podem ser incluídos nos pacotes de softwares. Podem ser “*linked*” ou “*embedded*”. Enquanto no primeiro tipo os objetos são armazenados separadamente e repartidos por várias aplicações, no segundo eles são armazenados junto com as aplicações, tornando-se de uso exclusivo das mesmas.

O sistema SCADA (*Supervisory Control And Data Acquisition*) é responsável pela melhora do processo de monitoração e controle, disponibilizando em tempo útil (tempo real) o estado atual do sistema, através de um conjunto de previsões, gráficos e relatórios, de

modo a permitir a tomada de decisões operacionais apropriadas, quer automaticamente, quer por iniciativa do operador.

2.3.2. Surgimento do OPC Foundation

Até meados da década de 90, apesar de existirem alguns tipos de redes específicas para comunicação em ambientes industriais, tais como: *Profibus*, *ControlNet*, *Fieldbus*, etc., a maioria das aplicações já utilizavam a rede *Ethernet* com o protocolo TCP/IP para transmissão de dados. Como a *Ethernet* é considerada uma rede multi-protocolar e a maioria das aplicações já utilizavam TCP/IP, era possível interligar vários dispositivos de fabricantes diferentes na mesma rede, porém apenas os dispositivos que possuísem o mesmo *driver* de comunicação poderiam trocar informações.

Para solucionar o problema de comunicação de dados, os sistemas tradicionais como os SCADA, utilizados normalmente para comunicar com dispositivos de controle, possuíam um conjunto de diversos *drivers* para então permitir a comunicação entre estes dispositivos de fabricantes distintos. Apesar do uso de *drivers* ter resolvido parcialmente os problemas de comunicação, estes apresentam algumas desvantagens (LEITÃO, 2006):

- Cada *driver* implica em custo adicional;
- Cada *driver* dispõe de mecanismos específicos para comunicação dos dados, com diferentes desempenhos.
- Cada *driver* tem sua própria interface com o usuário dificultando o treinamento e a manutenção do sistema.

Em 1995, buscando uma padronização nas operações de comunicação em tempo real, algumas empresas se reuniram com o objetivo de desenvolver um padrão baseado na tecnologia OLE/DCOM para acesso de dados em tempo real dentro do sistema operacional Windows. Com o resultado dessa reunião, formou-se um grupo que hoje conta com mais de 300 membros em todo mundo, incluindo quase todos os maiores provedores de controle de sistemas, instrumentação e controle de processos. Essa organização, sem fins lucrativos, chama-se *OPC Foundation* e a partir dela são definidos os padrões OPC (PAVEI, 2009).

A tecnologia OLE foi desenvolvida pela Microsoft© em meados de 1990, para suprir a necessidade de se integrar diferentes aplicações dentro da plataforma Windows, de forma a solucionar os problemas de desempenho e confiabilidade do até então utilizado padrão DDE (*Dynamic Data Exchange*) (OPCTI, 2013).

Como uma continuação da tecnologia OLE, o DCOM surgiu junto com o sistema operacional Windows NT e foi logo aceito pela indústria. Basicamente, o DCOM é um conjunto de definições para permitir a implementação de aplicações distribuídas em uma

arquitetura cliente-servidor. Desta forma, um cliente pode acessar diferentes servidores ao mesmo tempo e um servidor pode disponibilizar suas funcionalidades para diferentes clientes ao mesmo tempo (OPCTI, 2013).

O surgimento do padrão OPC possibilitou uma gama de vantagens (OPC Foundation, 2012):

- Padronização das interfaces de comunicação entre os servidores e clientes de dados de tempo real, facilitando a integração e manutenção dos sistemas.
- Eliminação da necessidade de *drivers* de comunicação específicos (proprietários).
- Melhoria do desempenho e otimização da comunicação entre dispositivos de automação.
- Interoperabilidade entre sistemas de diversos fabricantes.
- Integração com sistemas de execução da manufatura ou *Manufacturing Execution Systems* (MES1), sistemas integrados de gestão empresarial ou *Enterprise Resource Planning* (ERP2) e aplicações Windows (Excel, entre outras).
- Redução dos custos e tempo para desenvolvimento de interfaces e *drivers* de comunicação, com conseqüente redução do custo de integração de sistemas.
- Facilidade de desenvolvimento e manutenção de sistemas e produtos para comunicação em tempo real;
- Facilidade de treinamento.

No mesmo contexto, a OPC Foundation lida com diversos tipos de especificação de manipulação de dados. Dentre tantas é possível citar: a OPC Data Access (OPC DA), a OPC Alarm and Events (OPC AE), a OPC Historical Data Access (OPC HDA), a OPC batch e a OPC XML DA.

Com respeito especificamente à OPC XML DA, temos uma especificação cujo objetivo é desenvolver regras e formatos flexíveis e coerentes expondo dados de chão de fábrica utilizando o padrão XML. Esta especificação é baseada nos padrões dos serviços web XML, *Simple Object Access Protocol* (SOAP) e *Web Service Description Language* (WSDL) e padroniza as mensagens SOAP trocadas entre clientes e servidor. A padronização de mensagens frente a uma interface de aplicativo permite a implementação em diferentes sistemas operacionais e não somente naqueles baseados exclusivamente nos modelos COM/DCOM da Microsoft© (HONG; JIANHUA, 2006).

Sobre a arquitetura OPC, para Borges (2007), existem três elementos básicos: servidor, grupo e itens. Do ponto de vista do cliente, o servidor é essencialmente uma

estrutura de armazenamento para grupos, que por sua vez, têm como função básica o armazenamento de itens. Estes itens representam a ligação a pontos de entrada ou saída. Assim o item OPC não é um valor, mas apenas um meio de acesso a um valor. Desta forma, uma única variável de entrada ou saída, pode ser representada por itens diferentes, com diferentes propriedades e compartilhadas por mais que um cliente. A Fig. 2.14 ilustra a representação esquemática da arquitetura cliente/servidor OPC

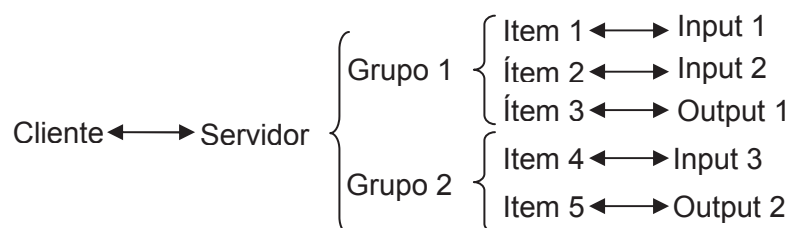


Figura 2.14. Representação esquemática da arquitetura Cliente/Servidor OPC.

Segundo Fonseca (2002), normalmente, os produtos para monitoração de dados (IHMs; sistemas supervisórios, etc.) são clientes OPC, enquanto que os produtos que fazem a comunicação direta com os dispositivos de campo utilizando protocolos proprietários são servidores OPC. Ainda de acordo com Fonseca (2002), o número de servidores OPC necessários para uma determinada aplicação irá depender do produto a ser utilizado. Normalmente, os fabricantes de dispositivos de campo (robôs, CLPs, dispositivos “inteligentes”, etc.) fornecem um servidor OPC capaz de comunicar com todos os protocolos dos seus produtos de linha. Este servidor é um software para o ambiente Windows que é executado em um microcomputador, normalmente PC. Ou seja, um servidor OPC da *Rockwell*, o *RSLinx* por exemplo, permite que diversos *drivers* de comunicação sejam configurados para as diversas redes (*ControlNet*, *DeviceNet*, *Ethernet*, *DH+*, etc.). Neste caso, o *RSLinx* funciona como um único servidor OPC, capaz de comunicar com diversos clientes OPC sendo executados na mesma máquina ou em máquinas remotas. Existem ainda servidores OPC de terceiros que permitem a configuração dos *drivers* de comunicação para diversas redes e protocolos de diferentes fabricantes. É o caso dos servidores da *Kepware* e da *Matrikon*. Do ponto de vista da arquitetura cliente/servidor, qualquer cliente OPC poderá se comunicar com quaisquer servidores OPC de quaisquer fabricantes.

Pela especificação do padrão OPC, todo servidor de dados deve enviar o dado OPC no formato apresentado a seguir:

- *Valor do dado*: Todos os tipos de dados *variant* definidos pela interface DCOM são suportados.

- *Time Stamp*: Esta informação é fornecida pelo servidor através da leitura do *time stamp* dos dispositivos de campo ou por geração interna. É utilizada a estrutura padrão do Windows para o UTC (*Universal Time Coordinated*).
- *Informação de estado*: São reservados 2 *bytes* para codificação do estado do dado fornecido pelo servidor. Por enquanto, apenas o uso do *byte* menos significativo foi definido. Dois bits definem a qualidade do dado que pode ser:
 - *Good* – Dado válido;
 - *Bad* – No caso de perda do *link* de comunicação com o dispositivo de campo, por exemplo;
 - *Uncertain* – No caso de existir o *link* de comunicação, mas o dispositivo de campo estiver fora de operação.

Quatro *bits* fornecem um detalhamento do estado apresentado, tais como *Not Connected* e *Last Usable Value*. Os últimos dois *bits* podem conter dados de diagnóstico no caso de falha de um sensor, por exemplo.

Normalmente, os produtos de mercado não permitem muita flexibilidade para a configuração dos dados solicitados pelo cliente, o que pode ser explicado pelo aspecto cultural dos *drivers* de comunicação específicos. Para Fonseca (2002), isto pode ser uma armadilha para os usuários.

Considerando o caso mais comum que consiste nos servidores de dados OPC (OPC Data Access), os clientes podem definir basicamente as seguintes configurações:

- Criação de grupos e itens OPC: Basicamente, todos os dados OPC são denominados itens. Cada item pode ser de um tipo diferente de dado compatível com a especificação OPC. Os diversos itens são organizados em grupos OPC, os quais definem as principais características de leitura dos itens (Taxa de Atualização, Estado Ativo/Inativo, Banda Morta, Leitura Síncrona/Assíncrona);
- Leitura Síncrona ou Assíncrona: Para um determinado grupo OPC pode ser definido se a leitura dos dados é feita de forma síncrona, a qual depende de uma confirmação de execução antes de uma nova leitura, ou assíncrona, a qual não depende da confirmação. Normalmente é utilizada a leitura assíncrona, a qual garante um melhor desempenho. Leitura de dados direto do dispositivo: A partir da versão 2.0 da especificação para o servidor de dados, é possível fazer a seleção no cliente OPC para leitura dos dados da memória *cache* do servidor ou diretamente do dispositivo de campo;

- Estado Ativo/Inativo: Cada item ou grupo pode ter o seu estado alterado pelo cliente para Ativo, habilitando a comunicação do mesmo, ou Inativo;
- Leitura Cíclica ou por Mudança de Estado: O cliente OPC pode definir se os dados do servidor serão lidos de forma cíclica ou por mudança (transição) de estado. Na leitura cíclica, o cliente faz a requisição de leitura regularmente, independentemente se os dados sofreram alteração de valor ou não. No caso de leitura por mudança de estado, o servidor fica responsável por enviar para os clientes os itens que sofrerem alteração de seu estado (qualidade do dado) ou quando os valores dos itens de um determinado grupo ultrapassarem o valor da banda morta;
- Banda Morta: É utilizado para definir os valores limites de transição para os itens de um determinado grupo, para os quais o servidor fará o envio para os clientes quando a alteração dos valores dos itens estiver fora da banda especificada.

A Fig. 2.15 apresenta a estrutura dos objetos para a comunicação OPC.

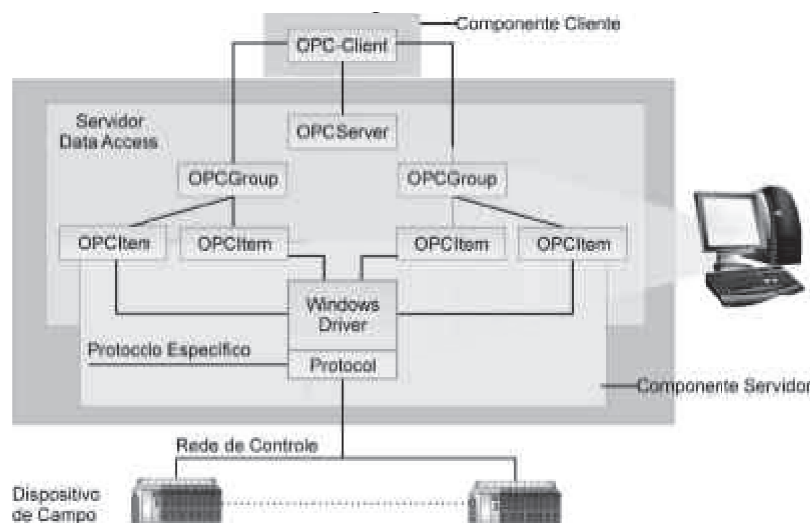


Figura 2.15. Estrutura interna dos objetos no padrão OPC (FONSECA, 2002).

Fonseca (2002) afirma que a escrita de dados OPC funciona de forma independente da leitura. Assim como na leitura, a escrita pode ser síncrona ou assíncrona. Entretanto, os comandos de escrita são executados imediatamente pelo servidor, sendo enviados diretamente para os dispositivos de campo.

2.4. Programação Orientada a Objetos

Antes de elaborar um programa para resolver qualquer problema, é necessário que o programador tenha conhecimento total do problema e saiba de antemão, através de um planejamento prévio, todas as etapas que deverá executar para escrever aquele programa. Ao escrever um programa, também é essencial entender os tipos de blocos de construção que estão disponíveis e empregar técnicas comprovadas de construção de programa.

De acordo com Deitel e Deitel (2004), antes de começar a programar, é essencial que o programador tenha conhecimento sobre os blocos de construção e as “palavras reservadas” da linguagem a qual está disposto a utilizar. Em Java, ao declarar um método, a palavra chave *void* indica que o método realizará uma tarefa, mas não retornará nenhuma informação ao completá-la; a palavra chave *public* é um modificador de acesso que indica que o método está disponível para o público, isto é, pode ser chamado de fora do corpo da declaração de classe por métodos de outras classes; já a palavra-chave *static* indica que o método pode ser chamado sem primeiro criar um objeto da classe em que o método é declarado.

Nas declarações de variável de instância, na maioria das vezes esta é precedida pela palavra-chave *private*. Assim como a palavra-chave *public*, a palavra-chave *private* é um modificador de acesso. As variáveis ou métodos declarados com o modificador de acesso *private* só são acessíveis a métodos da classe em que são declarados, enquanto que aquelas declaradas com *public* são acessíveis nos métodos de qualquer classe do projeto.

Na programação orientada a objetos, objetos são instâncias de classes, que determinam qual informação um objeto contém e como ele pode manipular tal informação. Além disso, as classes podem relacionar-se entre si, serem parte de outras classes ou herdar características de classes mais genéricas.

De acordo com Ricarte (2001), um dos grandes diferenciais da programação orientada a objetos em relação a outros paradigmas de programação, que também permitem a definição de estruturas e operações sobre essas estruturas, está no conceito de herança, mecanismo através do qual, definições existentes podem ser facilmente estendidas. Juntamente com a herança deve ser enfatizada a importância do polimorfismo, que permite selecionar funcionalidades que um programa irá utilizar de forma dinâmica, durante sua execução.

Na programação orientada a objetos, um objeto é um elemento que representa, no domínio da solução, alguma entidade (abstrata ou concreta) do domínio de interesse do problema sob análise. Objetos similares são agrupados em classes. Em outras palavras, os objetos são instâncias de classes. É através deles que (praticamente) todo o processamento

ocorre em sistemas implementados com linguagens de programação orientadas a objetos. O uso racional de objetos, obedecendo aos princípios associados à sua definição conforme estabelecido no paradigma de desenvolvimento orientado a objetos é chave para o desenvolvimento de sistemas complexos e eficientes (RICARTE, 2001).

Uma maneira muito difundida em Engenharia de Software utilizada para modelar projetos e auxiliar os programadores é através da UML. A UML é uma linguagem gráfica utilizada por programadores para representar sistemas orientados a objetos de maneira padronizada. Segundo Deitel e Deitel (2004), na UML cada classe é modelada em um diagrama de classes como um retângulo com três compartimentos, a Fig. 2.16 ilustra um exemplo de um diagrama de classes UML apresentado por Deitel e Deitel (2004) em sua obra. O compartimento superior contém o nome da classe centralizado horizontalmente no tipo negrito. O compartimento do meio contém os atributos da classe, que correspondem às variáveis de instância em Java. Na Fig. 2.16, o compartimento central contém a variável de instância *courseName* da classe *GradeBook*. A UML representa as variáveis de instância como atributos listando o nome do atributo, seguido por um caractere de dois-pontos e o tipo de atributo. O tipo UML do atributo *courseName* é *String*. A variável de instância *courseName* é *private* em Java, então o diagrama de classe lista um sinal de subtração (-) na frente do nome do atributo correspondente. O compartimento inferior contém as operações da classe, que correspondem aos métodos em Java. A UML modela operações listando o nome da operação seguido por um conjunto de parênteses. A classe *GradeBook* contém três métodos *public*, então o diagrama de classe lista três operações no terceiro compartimento. O sinal de adição (+) antes de cada nome de operação indica que a operação é *public* da UML (isto é, um método *public* em Java). O método *displayMessage* não requer informações adicionais para realizar suas tarefas, então os parênteses que seguem *displayMessage* no diagrama de classe estão vazios. A operação *setCourseName* tem um parâmetro *String* denominado *name*. A UML indica o tipo de retorno de uma operação colocando um dois-pontos e o tipo de retorno depois dos parênteses que se seguem ao nome da operação. O método *getCouseName* da classe *GradeBook* tem um tipo de retorno *String* em Java, então o diagrama de classe mostra um tipo de retorno *String* na UML. Observe que as operações *setCourseName* e *displayMessage* não retornam valores (isto é, elas retornam *void* – em Java, a palavra chave *void* indica que o método realizará uma tarefa mas não retornará nenhuma informação ao completá-la), então o diagrama de classe UML não especifica um tipo de retorno depois dos parênteses dessas operações.

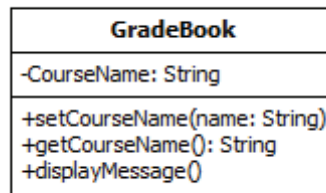


Figura 2.16. Diagrama de classe UML para a classe *GradeBook* (DEITEL; DEITEL, 2004).

Outro diagrama essencial para modelar em UML é o diagrama de atividades. Um diagrama de atividades modela o fluxo de trabalho de uma parte de um sistema de software. Esses fluxos de trabalho podem incluir parte de um algoritmo, como a estrutura de sequência da Fig. 2.17, que corresponde à tarefa de acrescentar uma nota à nota total e incrementar o contador. Os diagramas de atividades são compostos de símbolos de uso especial, como símbolos do estado da ação (retângulos com os lados esquerdos e direitos substituídos por arcos curvados para fora), losangos e pequenos círculos. Esses símbolos são conectados por setas de transição que representam o fluxo da atividade, isto é, a ordem em que as ações devem ocorrer. Cada estado da ação pode ser vinculado ainda a um comentário, que deve ser inserido em uma nota de texto e ancorado ao respectivo estado.

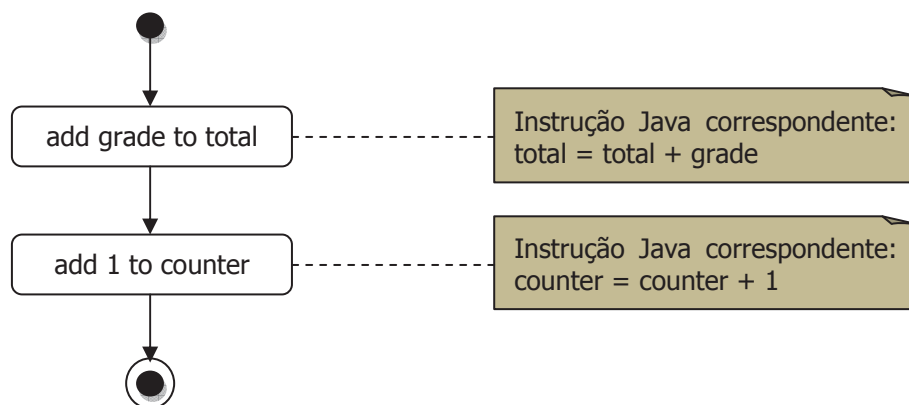


Figura 2.17. Diagrama de atividades da estrutura de sequência (DEITEL; DEITEL, 2004).

Para escrever um programa em Java, além das “palavras reservadas” e do processo de modelagem, é igualmente importante que o programador tenha conhecimento sobre as instruções de seleção e instruções de repetição em Java. Essas instruções são a base para a programação estruturada. A próxima subseção aborda brevemente estes dois tipos de instruções essenciais para o desenrolar de qualquer programa, seja ele em Java ou qualquer outra linguagem de programação. A subseção 2.5.2 apresenta os módulos existentes em Java – sendo eles: métodos, classes e pacotes – e o relacionamento

necessário para o correto funcionamento de um programa. Já a subseção 2.5.3 aborda conceitos básicos da aplicação de redes em Java, abordando principalmente o relacionamento Cliente/Servidor através de *sockets* TCP/IP.

2.4.1. Programação estruturada em Java

2.4.1.1. Instruções de seleção em Java

O Java contém três tipos de instruções de seleção. A instrução *if* realiza uma ação se uma condição for verdadeira ou pula a ação se a condição for falsa. A instrução *if... else* realiza uma ação se uma condição for verdadeira e realiza uma ação diferente se a condição for falsa. A instrução de seleção *switch* realiza uma de muitas ações diferentes, dependendo do valor de uma expressão.

A instrução *if* é uma instrução de uma única seleção porque seleciona ou ignora uma única ação (ou um único grupo de ações). A instrução *if... else* é denominada instrução de seleção dupla porque seleciona entre duas ações diferentes (ou grupos de ações). A instrução *switch* é denominada instrução de seleção múltipla uma vez que seleciona entre muitas ações diferentes (ou grupos de ações) (DEITEL; DEITEL, 2004).

2.4.1.2. Instruções de repetição em Java

O Java fornece três instruções de repetição (também denominadas instruções de *loop*) que permitem aos programas executar instruções repetidamente, contanto que uma condição (denominada condição de continuação do *loop*) permaneça verdadeira. As instruções de repetição são as instruções *while*, *do...while* e *for*. As instruções *while* e *for* realizam a ação (ou grupo de ações) no seu corpo zero ou mais vezes – se a condição de continuação de *loop* for inicialmente falsa, a ação (ou grupo de ações) não será executada. A instrução *do...while* realiza a ação (ou grupo de ações) no seu corpo uma ou mais vezes.

As palavras *if*, *else*, *switch*, *while*, *do* e *for* são palavras-chave Java. Palavras-chave são utilizadas para implementar vários recursos Java, como instruções de controle, e não podem ser utilizadas como identificadores, por exemplo, nos nomes de variáveis (DEITEL; DEITEL, 2004).

2.4.1.3. Generalização da programação estruturada

Para Deitel e Deitel (2004), programação estruturada promove a simplicidade. De uma forma generalista, apenas três formas de controle são necessárias para implementar um algoritmo:

- Sequência;

- Seleção;
- Repetição.

A estrutura de sequência é trivial. As instruções são listadas de forma que sua execução ocorra em uma ordem definida. A seleção pode ser implementada através de uma destas três maneiras:

- instrução *if* (seleção única);
- instrução *if...else* (seleção dupla);
- instrução *switch* (seleção múltipla).

De fato, é simples e direto provar que a instrução simples *if* é suficiente para fornecer qualquer forma de seleção – tudo o que pode ser feito com a instrução *if...else* e a instrução *switch* pode ser implementado combinando-se instruções *if*, embora talvez não de modo tão claro e eficiente.

Já a repetição pode ser implementada de uma destas três maneiras:

- instrução *while*;
- instrução *do...while*;
- instrução *for*.

Assim como a instrução de seleção *if*, na repetição também é simples provar que a instrução *while* é suficiente para fornecer qualquer forma de repetição. Tudo o que pode ser feito com a instrução *do...while* e a instrução *for* pode ser feito com a instrução *while*, embora talvez não tão convenientemente.

A combinação desses resultados ilustra que qualquer forma de controle que possa ser necessária um dia em um programa Java pode ser expressa em termos de

- sequência;
- instrução *if* (seleção);
- instrução *while* (repetição).

e que essas podem ser combinadas apenas de duas maneiras – empilhamento e aninhamento. De fato, programação estruturada concentra-se essencialmente na simplicidade (DEITEL; DEITEL, 2004).

2.4.2. Módulos em Java

Há três tipos de módulos em Java – métodos, classes e pacotes. Programas Java são escritos combinando novos métodos e classes que o programador escreve com métodos e

classes predefinidos disponíveis na *Java Application Programming Interface* (também conhecida com API do Java ou bibliotecas de classe Java) e em várias outras bibliotecas de classes. Em geral, classes relacionadas são agrupadas em pacotes de modo que possam ser importadas nos programas e reutilizadas. A API do Java fornece uma rica coleção de classes predefinidas que contém métodos para realizar cálculos matemáticos comuns, manipulações de *string*, manipulações de caractere, operações de entrada/saída, operações de banco de dados, operações de rede, processamento de arquivo, verificação de erros e muitas outras operações úteis. As classes da API do Java fazem parte do J2SE Development Kit (JDK) 5.0 (DEITEL; DEITEL, 2004).

2.4.3. Aplicações de redes em Java

Os recursos fundamentais das redes em Java são declarados pelas classes e interfaces do pacote *java.net*, por meio do qual o Java oferece comunicações baseadas em: (a) fluxo, que permitem aos aplicativos visualizar as redes como fluxos de dados; (b) pacotes, para transmitir pacotes individuais de informações – comumente utilizados para transmitir áudio e vídeo pela Internet (DEITEL; DEITEL, 2004).

Em um relacionamento cliente-servidor, o cliente solicita que alguma ação seja realizada e o servidor realiza a ação e responde para o cliente.

2.4.3.1. Comunicação baseada em *socket*

As comunicações baseadas em *socket* do Java permitem aos aplicativos visualizar a rede como se fosse uma E/S de arquivo – um programa pode ler ou gravar em um *socket* tão simplesmente quanto lê ou grava em um arquivo. O *socket* é simplesmente uma construção de software que representa uma extremidade final de uma conexão.

Com *sockets* de fluxo um processo estabelece uma conexão com outro processo. Enquanto a conexão estiver no ar, os dados fluem entre processos em fluxos contínuos. Este serviço é orientado a conexão e o protocolo utilizado para transmissão é o TCP.

Com “*sockets* de datagrama”, são transmitidos pacotes individuais de informações. Isso não é apropriado para programadores rotineiros, porque o protocolo utilizado, o UDP, é um serviço sem conexão e, assim, não garante que pacotes cheguem em uma ordem particular. Com o UDP, os pacotes podem até mesmo ser perdidos ou duplicados.

Basicamente, o TCP, o UDP e os protocolos relacionados permitem que uma grande variedade de sistemas de computadores heterogêneos (isto é, sistemas de computadores com diferentes processadores e diferentes sistemas operacionais) se intercomuniquem.

De acordo com Deitel e Deitel (2004), estabelecer um servidor simples em Java, utilizando *sockets* de fluxo, requer cinco passos:

No *Passo1* deve-se criar um objeto *ServerSocket*. Uma chamada ao construtor *ServerSocket* como:

```
ServerSocket server = new ServerSocket( númeroDaPorta, comprimentoDaFila ); /*instancia  
o objeto server da classe ServerSocket como um novo objeto ServerSocket com o número  
da porta e comprimento de fila definidos*/
```

A instrução acima registra um número de porta TCP disponível e especifica um número máximo de clientes que podem esperar para se conectarem ao servidor (isto é, o comprimento da fila). O número de porta é utilizado pelos clientes para localizar o aplicativo servidor no computador servidor. Isso costuma ser denominado ponto de *handshake*. Se a fila estiver cheia, o servidor recusa as conexões do cliente. Somente um aplicativo por vez pode ser vinculado a uma porta específica no servidor.

Os programas gerenciam cada conexão de cliente com um objeto *Socket*.

No *Passo2*, o servidor ouve indefinidamente (ou bloqueia) uma tentativa de conexão por um cliente. Para ouvir uma conexão de cliente, o programa chama o método *ServerSocket accept*, como em:

```
Socket connection = Server.accept(); /*chama o método ServerSocket accept para o objeto  
connection*/
```

A instrução acima retorna um *Socket* quando uma conexão com um cliente é estabelecida. O *Socket* permite ao servidor interagir com o cliente. Na verdade, as interações com o cliente ocorrem em uma porta diferente de servidor a partir do ponto de *handshake*. Isso permite que a porta especificada no *Passo1* seja utilizada novamente em um servidor de múltiplas *threads* para aceitar outra conexão de cliente.

O *Passo3* é obter os objetos *OutputStream* e *InputStream* que permitem ao servidor se comunicar com o cliente enviando e recebendo bytes. O servidor envia informações ao cliente via um *OutputStream* e recebe informações do cliente via um *InputStream*. O servidor invoca o método *getOutputStream* no *Socket* para obter uma referência ao *OutputStream* do *Socket* e invoca o método *getInputStream* no *Socket* para obter uma referência ao *InputStream* do *Socket*.

Os objetos *Stream* (fluxo) podem ser utilizados para enviar ou receber bytes individuais ou sequências de bytes com o método *write* de *OutputStream* e com o método *read* de *InputStream*, respectivamente. Frequentemente é útil enviar ou receber valores de tipos primitivos (por exemplo, *int* e *double*) ou objetos *Serializable* (por exemplo, *Strings* ou

outros tipos serializáveis) em vez de enviar *bytes*. Nesse caso, podemos utilizar as técnicas de empacotamento de outros tipos *Stream* (por exemplo, *ObjectOutputStream* e *ObjectInputStream*) em torno do *OutputStream* e *InputStream* associados com o *Socket*. Por exemplo:

```
ObjectInputStream input = new ObjectInputStream( connection.getInputStream() );
/*instancia o objeto input da classe ObjectInputStream como um novo objeto
ObjectInputStream com chamada do método connection.getInputStream().*/
```

```
ObjectOutputStream output = new ObjectOutputStream( connection.getOutputStream() );
/*instancia o objeto output da classe ObjectOutputStream como um novo objeto
ObjectOutputStream com chamada do método connection.getOutputStream().*/
```

A vantagem de estabelecer esses relacionamentos é que: o que o servidor gravar no *ObjectOutputStream* é enviado via *OutputStream* e estará disponível no *InputStream* do cliente, e o que o cliente gravar em seu *OutputStream* (com um *ObjectOutputStream* correspondente) estará disponível via *InputStream* do servidor. A transmissão dos dados sobre a rede é transparente e tratada completamente pelo Java.

O *Passo4* é a fase de processamento em que o servidor e o cliente se comunicam via objetos *OutPutStream* e *InputStream*.

No *Passo5*, quando a transmissão está completa, o servidor fecha a conexão invocando o método *close* nos fluxos e no *Socket*.

Com *sockets*, a E/S da rede aparece para programas Java como sendo similar à E/S de arquivo sequencial. Isso facilita muito o trabalho, visto que os *sockets* ocultam muito da complexidade de programação da rede do programador, podendo este se concentrar somente na maneira mais adequada para se transmitir os dados.

2.4.3.2. O *multithreading* do Java

Com o *multithreading* do Java é possível criar servidores com múltiplas *threads* que podem gerenciar várias conexões simultâneas com vários clientes. Segundo Deitel e Deitel (2004), essa arquitetura de servidor com múltiplos *threads* é precisamente a que é utilizada nos servidores de rede populares.

Um servidor com múltiplas *threads* pode receber o *Socket* retornado por cada chamada *accept* e criar uma nova *thread* que gerencia a E/S de rede por meio desse *Socket*. Alternativamente, um servidor com múltiplos *threads* pode manter um *pool* de

threads (um conjunto de *threads* existente) pronto para gerenciar a E/S da rede por meio dos novos *Sockets* à medida que são criados.

Em sua obra, Deitel e Deitel (2004) afirmam que estabelecer um cliente simples em Java exige quatro passos:

No *Passo1*, cria-se um *Socket* para estabelecer a conexão com o servidor. O construtor de *Socket* estabelece a conexão com o servidor. Por exemplo, a instrução:

```
Socket connection = new Socket( endereçoDoServidor, porta ); /*instancia o objeto connection do tipo Socket como um novo objeto Socket com endereço do servidor e porta definidos*/
```

A instrução acima utiliza o construtor de *Socket* com dois argumentos – o endereço do servidor e o número da porta. Se a tentativa de conexão for bem sucedida, essa instrução retorna um *Socket*.

No *Passo2*, o cliente utiliza os métodos *getInputStream* e *getOutputStream* de *Socket* para obter referências ao *InputStream* e *OutputStream* do *Socket*. Se o servidor está enviando informações na forma de tipos reais, o cliente deve receber as informações no mesmo formato. Portanto, se o servidor envia valores com um *ObjectOutputStream*, o cliente deve ler esses valores como um *ObjectInputStream*.

O *Passo3* é a fase de processamento em que o cliente e o servidor comunicam-se via objetos *InputStream* e *OutputStream*.

No *Passo4*, o cliente fecha a conexão quando a transmissão está completa invocando o método *close* nos fluxos e no *Socket*.

2.5. Planejamento Automático

Planejar é o processo abstrato e deliberativo de escolha e organização de ações antecipando seus efeitos esperados. Esse processo tem como missão atingir, da melhor forma possível, os objetivos pré-estabelecidos. O planejamento automático é uma área da Inteligência Artificial que estuda este processo deliberativo computacionalmente (VAQUERO, 2007).

2.5.1. Histórico do planejamento automático

O surgimento do planejamento automático se iniciou na década de 60 a partir de trabalhos científicos focados na criação de solucionadores gerais de problemas

(principalmente com o uso de lógica de primeira ordem), como por exemplo, o GPS (*General Problem Solver*) (ERNST; NEWELL, 1969) (NEWELL; SIMON, 1972) e QA3 (GREEN, 1969). No entanto, o primeiro planejador capaz de usufruir, de forma eficaz, das representações dos domínios durante a obtenção das soluções dos problemas surgiu no início da década de 70, quando um grupo de pesquisadores do Instituto de Pesquisa de Stanford criou um sistema de planejamento automático, denominado STRIPS (*Stanford Research Institute Problem Solver*) (FIKES; NILSSON, 1971). De formulação simples, este planejador marcou o início da Era Clássica do Planejamento Automático, que se estendeu até o começo da década de 90.

Até 1995, os avanços foram significativos, no entanto, os planejadores não conseguiam solucionar diversos problemas, mesmo que simples, em tempo computacional satisfatório. Foi quando Avrim Blum apresentou um planejador que utilizava um método de extração de planos diferenciado através de grafos. Este novo planejador foi denominado GRAPHPLAN (BLUM; FURST, 1997). Sua simplicidade aliada ao seu desempenho superior aos planejadores da época estimulou o desenvolvimento e a pesquisa de novas técnicas de planejamento e marcou o início da Era Neoclássica do Planejamento Automático.

O desenvolvimento de técnicas neoclássicas de planejamento trouxe novos espaços de busca e novos algoritmos que permitiram um aumento significativo no tamanho e na complexidade dos problemas de planejamento que podiam ser resolvidos. Em 1998 o Planejamento Automático recebeu um significativo impulso com o surgimento de uma nova técnica de planejamento, apresentada por Hector Geffner, onde foi possível encontrar um novo paradigma para esta área de pesquisa: *Planejamento por Busca Heurística* (BONET; GEFFNER, 1999). O HSP - *Heuristic Search Planner* (BONET; GEFFNER, 2000) - se tornou uma das técnicas mais rápidas de planejamento e sua grande contribuição foi a definição de uma função de custo que guia o processo de busca por uma solução (GHALLAB; NAU; TRAVERSO, 2004).

Como o grande foco da comunidade de planejamento ainda era o desenvolvimento e pesquisa de planejadores, no mesmo ano de 1998 começaram a surgir esforços para definição de uma linguagem comum para modelagem dos domínios de planejamento. Com o objetivo de comparar os planejadores existentes e incentivar ainda mais a pesquisa nesta área da IA, criou-se a linguagem de definição de domínios de planejamento denominada PDDL – *Planning Domain Definition Language* (MCDERMOTT, 1998). Esta linguagem foi utilizada na primeira competição de planejadores denominada IPC – *International Planning Competition* – que ocorreu durante um dos principais congressos na área de Planejamento Automático, o AIPS (*Artificial Intelligence Planning Systems* - 1998). Nesta competição os

planejadores resolviam problemas clássicos de planejamento, bem como problemas reais simplificados (VAQUERO et al., 2009).

Com os avanços, planejadores cada vez mais eficientes surgiram nas competições, como, por exemplo, o FF (HOFFMAN; NEBEL, 2001) que se destacou na competição de 2000 utilizando planejamento por busca heurística; o LPG, destaque da competição de 2002, que incorporava tanto técnicas do GRAPHPLAN quanto técnicas de busca heurísticas; e o METRIC-FF (HOFFMANN, 2003) que se caracterizava como uma evolução do FF capaz de tratar restrições numéricas e funções de otimização. Em 2004 os planejadores *Fast (Diagonally) Downward* (HELMERT, 2004), SGPLAN (WAH; CHEN, 2004) e SATPLAN'04 (KAUTZ; SELMAN; HOFFMANN, 2006) obtiveram ótimos resultados, surpreendendo a todos com suas performances, principalmente no tempo de resposta (VAQUERO, 2007).

2.5.2. Planejamento clássico

Para Ghallab et al. (2004), o planejamento automático é uma crescente área da IA que está presente em cenários como: planejamento de trajetória e movimentação de sistemas automáticos móveis; planejamento de percepção envolvendo ações de sensoriamento para captação de informação do ambiente; planejamento de navegação que combina sensoriamento e definição de trajetórias; planejamento de manipulação relacionado com movimentação de objetos como, por exemplo, montagem de peças, organização de contêineres, entre outros.

De acordo com Ghallab et al. (2004) o planejamento clássico requer oito hipóteses restritivas:

- Σ Finito: O *sistema estado-transição* Σ possui um conjunto finito de estados;
- Σ Totalmente observável: O sistema Σ é totalmente observável, isto é, tem-se total conhecimento do estado de Σ ;
- Σ Determinístico: o sistema Σ é determinístico;
- Σ Estático: o sistema Σ é estático, isto é, o conjunto de Eventos exógenos E é vazio;
- Objetivos restritos: os planejadores lidam apenas com objetivos restritos que são especificados explicitamente no estado objetivo (*goal state*). Os objetivos estendidos, tais como: estados a serem evitados, restrições na trajetória da solução ou funções de otimização, não são permitidos;
- Planos sequenciais: um “plano-solução” de um problema de planejamento é uma sequência finita de ações linearmente ordenada;

- Tempo implícito: Ações e eventos não possuem duração. Elas são transições de estado instantâneas;
- Planejamento *offline*: o planejador não percebe as mudanças do sistema Σ enquanto está planejando. Ele planeja apenas com as condições iniciais e os objetivos, independentemente da dinâmica que ocorre em Σ . O planejador não recebe o *status de execução*.

O sistema de transição de estados Σ é uma *tupla* de (ζ, A, E, γ) , onde ζ é um conjunto de estados, A é um conjunto de ações, E é um conjunto de eventos exógenos e γ é uma função de transição de estado.

Este modelo pode ser descrito por meio da interação de três componentes:

- Um sistema estado-transição Σ evolui como especificado pela sua função de transição de estado γ , de acordo com os eventos e ações que este recebe;
- Um controlador, dado como entrada o estado s do sistema, fornece como saída uma ação a de acordo com algum plano.
- Um planejador, dado como entrada a descrição do sistema Σ , uma situação inicial, e algum objetivo, sintetiza um plano para o controle de maneira a atingir o objetivo.

O modelo conceitual para o planejamento clássico está ilustrado na Fig. 2.18.

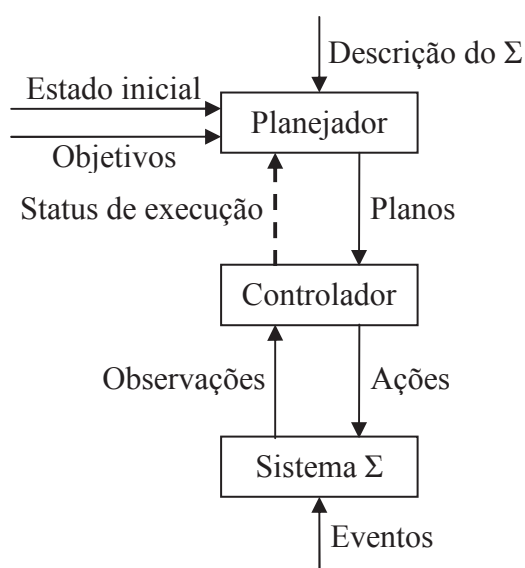


Figura 2.18. O modelo conceitual para o planejamento clássico (traduzido de Ghallab, et al. (2004)).

De acordo com Ghallab et al. (2004), o planejamento clássico é fundamentado em diversas hipóteses restritivas, dentre as quais destacam-se:

- Determinismo: ações têm efeitos determinísticos, ou seja, cada ação, quando aplicável em um estado, levará a um único novo estado.
- Domínio Completamente Observável: o controlador possui conhecimento total sobre o estado corrente do sistema, ou seja, observações resultam em um único estado, o estado atual do sistema.
- Objetivos alcançáveis: objetivos são conjuntos de estados, ou seja, o objetivo é construir um plano que leve a um dos estados objetivo.

Como consequência, no planejamento clássico os planos são vistos como uma sequência de ações e a realimentação fornecida pelas observações não são necessárias. Como forma de aproximar os problemas de planejamento um pouco mais às incertezas da realidade, Ghallab et al. (2004) apresentaram o planejamento sob incerteza, que nada mais é que uma relaxação aplicada às hipóteses do planejamento clássico. Para considerar a incerteza no planejamento, propõem-se a hipótese de sistemas não deterministas, parcialmente observáveis e com objetivos estendidos.

O determinismo trata-se de uma visão simplificada do mundo que assume que este evolui ao longo de um único caminho totalmente previsível. Para Ghallab et al. (2004), esta é uma hipótese bastante irreal e impraticável, visto que prever todas as variáveis de um sistema, bem como a maneira como elas evoluem, é praticamente impossível.

No que diz respeito à “observabilidade” do domínio, em diversas aplicações, o estado do sistema é apenas parcialmente observável em tempo real e, como uma consequência, diferentes estados do sistema são indistinguíveis para o controlador.

O conceito de objetivos estendidos exalta que, em domínios não deterministas os objetivos precisam especificar requisitos de diferentes forças que levem em conta o não determinismo e possíveis falhas. Por exemplo, quando um sistema tentar atingir um determinado estado, e o mesmo não conseguir alcançar tal estado, este sistema deverá garantir que algum estado seguro seja mantido.

O planejamento baseado em Processos de Decisão de Markov – *Markov Decision Processes* (MDPs) – foi desenvolvido para lidar com não determinismo, probabilidades, domínios parcialmente observáveis e objetivos estendidos. A ideia chave deste tipo de planejamento é representar o problema de planejamento como um problema de otimização.

O planejamento baseado em processos de decisão Markovianos é dividido em duas frentes: uma que considera a hipótese de domínio totalmente observável, denominada de

planejamento em MDP; e outra que assume que o domínio é parcialmente observável, denominado de planejamento em POMDP, o qual relaxa a consideração de que o controlador possui total conhecimento sobre o estado do sistema.

2.5.3. O sistema *itSIMPLE*

Segundo Tavares et al. (2011), é possível pensar em um processo de fabricação na manufatura moderna como uma sucessão de ações independentes iniciadas por um acionamento e cuja finalização pode ser detectada por sensores. Estas ações podem ser realizadas por dispositivos, máquinas de comando numérico, veículos autônomos ou AGVs (*Automated Guided Vehicle*), esteiras rolantes, robôs manipuladores, etc. Portanto estas ações são programas relativamente complexos, em diferentes linguagens, incluindo-se aí também as linguagens de programação de CLPs.

Assim, o sequenciamento aqui referido pode ser visto como uma integração de diferentes atividades, dispositivos e máquinas, para a obtenção de um processo mais sofisticado. Mesmo que o processo de detecção e acionamento possa também ser feito por CLPs, nota-se que uma linguagem deste nível não é adequada a este tipo de integração. O próprio processo de projeto não é factível em uma linguagem de baixo nível como a linguagem *Ladder*, ainda que no final do processo, se deseje justamente ter uma forma de acionar automaticamente a entrada destas ações usando um CLP (e detectando a condição de término das ações com sensores).

Torna-se assim bastante atraente a possibilidade de se aplicar novas ferramentas de *design* para sistemas de planejamento e escalonamento neste tipo de integração, com as vantagens em precisão, análise de dependências, adaptabilidade e inserção de um “comportamento inteligente” no sistema.

Até recentemente esta possibilidade era muito remota, dado que os problemas de planejamento automático eram tratados apenas como problemas modelo, sendo extraídos já diretamente em linguagens de especificação formal como a PDDL. Entretanto, a partir da virada do século, uma discussão passou a dominar a comunidade de planejamento, e que se refere justamente à possibilidade do uso destas técnicas da IA a problemas reais, entre estes os problemas de manufatura.

Problemas reais de grande interesse têm sido abordados com estas técnicas como na logística dos sistemas portuários (DAHALL et al., 2003), ou na logística de carga, descarga e “tanqueamento” de petróleo no porto de São Sebastião (SETTE et al., 2008), ou ainda no roteamento de óleo cru em oleodutos (LI et al., 2006).

A partir disso, emergiu a necessidade de um processo de projeto que permitisse associar o plano gerado pelas técnicas de planejamento a cada um dos subprocessos que

substanciam as ações. Neste contexto, surge o sistema itSIMPLE, que consiste em uma proposta de processo de projeto apresentada por Vaquero et al. (2007). Este sistema propõe um tratamento inicial, passando pela eliciação e análise de requisitos, modelagem do domínio e análise do modelo, antes mesmo da escolha do planejador adequado para geração do plano.

O ambiente do itSIMPLE incorpora um conjunto de linguagens de representação e teorias capazes de lidar com requisitos e engenharia do conhecimento, em conjunto com o ciclo de vida do projeto, como mostrado na Fig. 2.19. O ambiente foi implementado em JAVA e é caracterizado por uma ferramenta integrada que utiliza UML com uma abordagem de planejamento para as fases de análise de requisitos, especificação e modelagem (principalmente os aspectos estáticos); Redes de Petri (MURATA, 1989) para a análise de requisitos e análise dinâmica do modelo do domínio; a PDDL, linguagem padrão em planejamento, como uma linguagem intermediária entre a ferramenta e as técnicas de planejamento existentes; a XML (*eXtensible Markup Language*) (W3C., 2008) como uma linguagem de armazenamento e representação dos modelos, assim como uma linguagem que integra e descreve todas as outras, como por exemplo a PNML (*Petri Nets Markup Language*) (WEBER; KINDLER, 2003) que descreve o formalismo de Redes de Petri em XML e a XPDDL (*eXtensible Planning Domain Definition Language*) (GOUGH, 2004) que nada mais é que a representação da PDDL em XML.

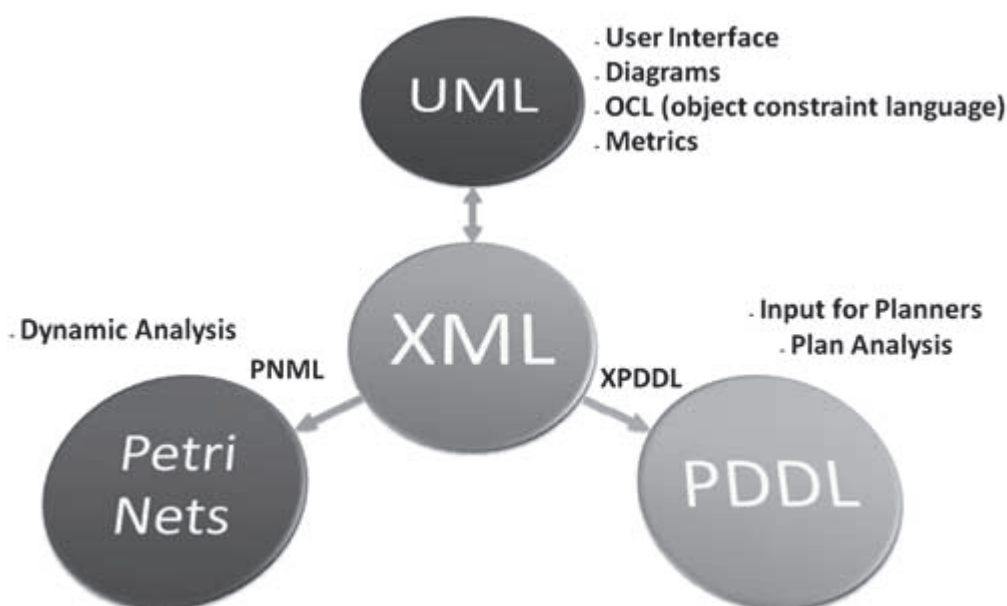


Figura 2.19. Estrutura de linguagens do itSIMPLE (VAQUERO et al., 2013).

Segundo Vaquero (2007), com esta ferramenta é possível que diversos planejadores recebam o mesmo problema (composto pelos mesmos requisitos, restrições, condições

iniciais e situações objetivo), de forma que seja justa uma comparação dos planos gerados e a opção de escolha para aquele que mais se ajuste às características do sistema requerido.

Na versão atual, o itSIMPLE pode se comunicar com os principais planejadores disponíveis na literatura, tais como SGPlan6 (HSU; WAH, 2008), Metric-FF (HOFFMANN, 2003), MIPSXXL (EDELKAMP; JABBAR, 2008), LPG-td (GEREVINI et al., 2004) e HSP (BONET; GEFFNER, 2000). Entretanto, um processo de escolha do melhor planejador para o problema ainda não foi implementado, sendo que esta versão do sistema apenas prepara o domínio para todos os planejadores embarcados.

Por fim, Vaquero et al. (2013) afirmam ainda que a tradução de UML para PDDL oferece um mecanismo para testar e analisar modelos com planejadores, proporcionando a oportunidade de aperfeiçoar os modelos e, consequentemente, a qualidade dos planos. Para os autores, o princípio da flexibilidade que permite usar um conjunto diversificado de linguagens de representação e uma variedade de técnicas de planejamento torna a ferramenta uma interessante *sandbox* para novas representações formais de planejamento de domínios e para o desenvolvimento de novas técnicas de planejamento.

2.6. Trabalhos Relacionados

Abordagens de inteligência artificial são muito recorrentes em aplicações computacionais, sendo já bem aceita pela comunidade acadêmica e com resultados condizentes. No entanto, quando se fala na aplicação deste tipo de tecnologia em ambiente industrial, principalmente na área de Automação, o que se vê é um baixo número de pesquisas relacionadas e um número menor ainda de pesquisas que já aplicaram seus métodos de abordagem no chão de fábrica propriamente dito de alguma empresa de produto ou de processo. Dentre os trabalhos pesquisados, destacam-se quatro estudos de caso que apresentam aplicação prática dos paradigmas em sistemas de automação, utilizando Arquitetura Orientada a Serviços, ou *Service Oriented Architectures* (SOA); Sistemas Multiagentes, ou *Multi-agent Systems* (MAS) e lógica *fuzzy*. As seguintes subseções detalham rapidamente cada um destes estudos de caso.

2.6.1. SOA/MAS em sistemas de automação da manufatura

Este trabalho foi abordado por João Peixoto em sua dissertação de Mestrado, sob o título “Desenvolvimento de Sistemas de Automação da Manufatura usando Arquiteturas Orientadas a Serviço e Sistemas Multiagentes” (PEIXOTO, 2012). Neste trabalho o autor propõe os paradigmas de SOA e MAS em conjunto para o gerenciamento dos sistemas de

automação da produção. O estudo de caso escolhido para validar este método consiste em um sistema de manufatura para a montagem de um produto composto por quatro blocos coloridos em posições definidas de um palete. Os paletes circulam por uma esteira transportadora que passa em frente a seis estações de montagem, cada estação de montagem tem a capacidade de montar algum ou alguns tipos de blocos nos paletes. A Fig. 2.20 apresenta o esquema deste processo.

Os agentes considerados neste sistema foram os agentes de montagem, pois realizam ações autônomas de inserir ou retirar um paleta da estação através do acionamento dos seus cilindros. Estes agentes comunicam-se entre si e agem de forma a otimizar o processo de fabricação de produtos, reduzindo custos e tempo de execução. O sistema foi simulado computacionalmente e os resultados foram satisfatórios.

O autor faz ainda uma análise comparativa do comportamento do sistema de acordo com o método de controle utilizado: SOA/MAS ou programação clássica de CLP. Para isso, ele analisa seis métricas tanto qualitativas quanto quantitativas. Seus resultados apontaram desempenho superior de SOA/MAS em três casos, em outros dois o sistema de melhor desempenho foi o tradicional, havendo ainda um caso isolado com desempenho semelhante. A Tab. 2.2 traz em detalhes os resultados obtidos pelo autor.

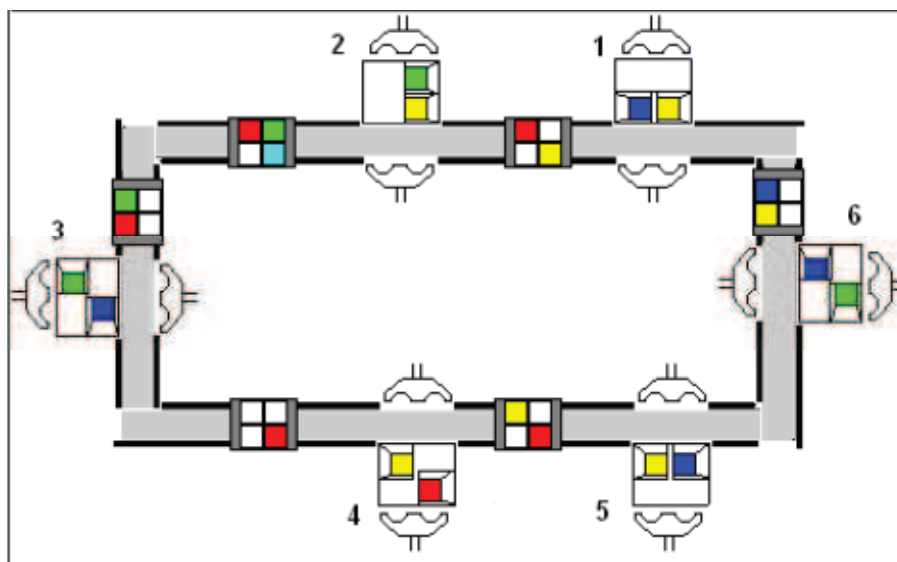


Figura 2.20. Proposta de manufatura de um sistema de montagem de peças (PEIXOTO, 2012).

Tabela 2.2. Análise das métricas abordadas (PEIXOTO, 2012).

Métrica	Análise	Sistema de melhor desempenho
Variação do Tempo de produção com aumento das estações	Em CLP o acréscimo de estações não é significativo para o tempo de produção, diferente do SOA, onde o acréscimo de estações afeta o tempo de produção.	CLP
Manter a produção com a retirada de estações	A retirada de estações em ambos os sistemas não prejudica a continuidade de produção, desde que outras estações tenham habilidades idênticas às da estação que saiu.	Desempenho igual
Manter a produção com a inserção de estações	No CLP para inserção de uma nova estação foi necessário desligar e reprogramar novamente. Já no SOA bastou instanciar a nova estação.	SOA
Quantidade de memória ocupada pelo programa	Com CLP o acréscimo de estações pouco interferiu na quantidade de memória ocupada (4% de 1 para 6 estações). Já em SOA há uma diferença bem mais significativa (82% de 1 para 6 estações).	CLP
Competências necessárias ao operador do sistema de montagem	Em SOA, a operação do sistema requer conhecimentos básicos de sistema operacional, mesmo para inserção e retirada de estações. No CLP é necessário um programador para realizar as alterações no sistema.	SOA
Linhas de código de programação.	Em SOA, por se tratar de instâncias, o esforço de programação é único, independente do número de estação. Já em CLP a cada inserção de estação torna-se necessário acrescentar códigos, onerando mais esforço de programação.	SOA

2.6.2. Lógica fuzzy em sistemas controlados por CLP

Este trabalho foi abordado por Iulia Dumitru, Nicoleta Arghira, Ioana Fagarasan e Sergiu Iliescu, sob o título “A fuzzy PLC control system for a servomechanism” (DUMITRU et al., 2010). Neste trabalho os autores propõem uma abordagem diferente para controle de dispositivos industriais através de aplicação direta de lógica *fuzzy* em um controlador industrial, tipo CLP. A representação esquemática do sistema utilizado como estudo de caso pode ser visualizada na Fig. 2.21. O esquema ilustra um servomecanismo conectado a um aos módulos de I/O do CLP através de um modulo de conexão, uma estação de simulação utilizada para descrever o estado dos sensores e interruptores do CLP, e um PC equipado com um software para programação do CLP e implementação do controlador *fuzzy*.

De acordo com os resultados dos autores, os parâmetros do controlador proposto foram projetados tanto que o servomecanismo apresentou uma resposta rápida e mínimo erro em regime permanente. Os distúrbios (diferentes cargas aplicadas ao motor) foram rejeitados por este tipo de controle. Para os autores, o sistema projetado conecta controle clássico com tecnologia moderna, suportando um vasto campo de aplicações de controle baseado em lógica *fuzzy*.

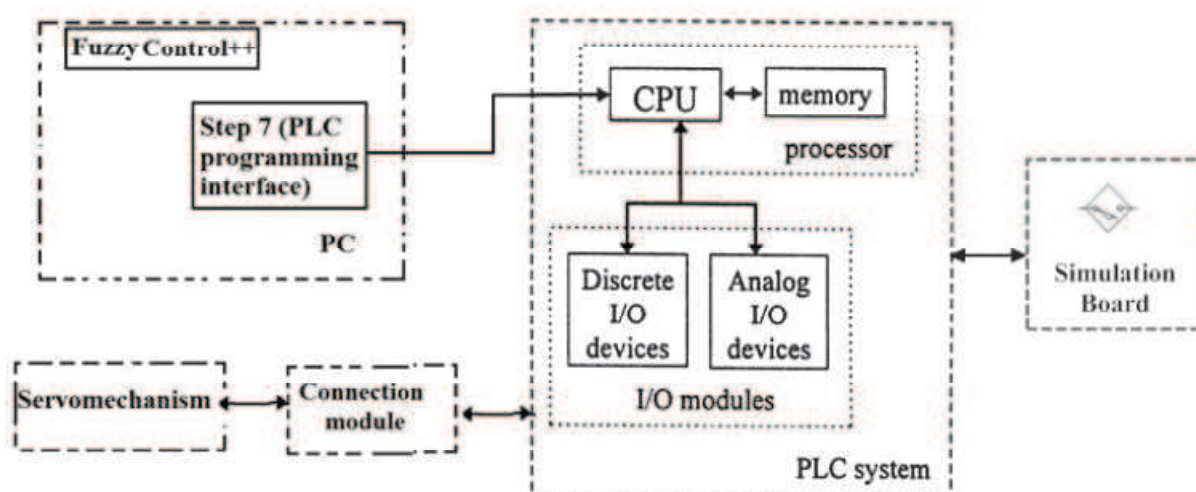


Figura 2.21. Esquema do projeto *Fuzzy-PLC Control System* (DUMITRU et al., 2010).

2.6.3. MAS aplicado em sistemas de transportes.

Em 2010, Pavel Vrba e Vladimír Marík publicaram no IEEE *Transactions on Systems, Man, And Cybernetics*, um trabalho com o título “Capabilities of Dynamic Reconfiguration of Multiagent-Based Industrial Control Systems” (VRBA; MARIK, 2010). Com este artigo, os autores propõem o uso de sistemas multiagentes em sistemas de transportes de materiais. Nas suas análises, os autores consideraram agentes que se integram modularmente e

compõe rotas, sendo que cada rota possui um custo atribuído. O sistema de controle se vale das funcionalidades dos agentes para determinar a rota mais econômica. Assim que o sistema encontra a rota economicamente mais rentável, os agentes que compõe esta rota se agrupam para executar o traslado.

O sistema de transporte foi implantado em uma bancada da Rockwell Automation e possui uma simulação baseada em agentes reais com um controle de tempo de execução exclusivo, baseado em interfaces com CLP. Dessa forma, os agentes podem interagir com qualquer subsistema de simulação ou sistema físico, compartilhando os valores dos sensores e atuadores na memória do CLP. Do ponto de vista da tarefa de transporte, as células de trabalho são consideradas como locais no chão da fábrica ligadas através de uma rede de transportadores, entre as quais as peças de trabalho são transportadas. Uma imagem e uma simulação do sistema são apresentadas na Fig. 2.22.

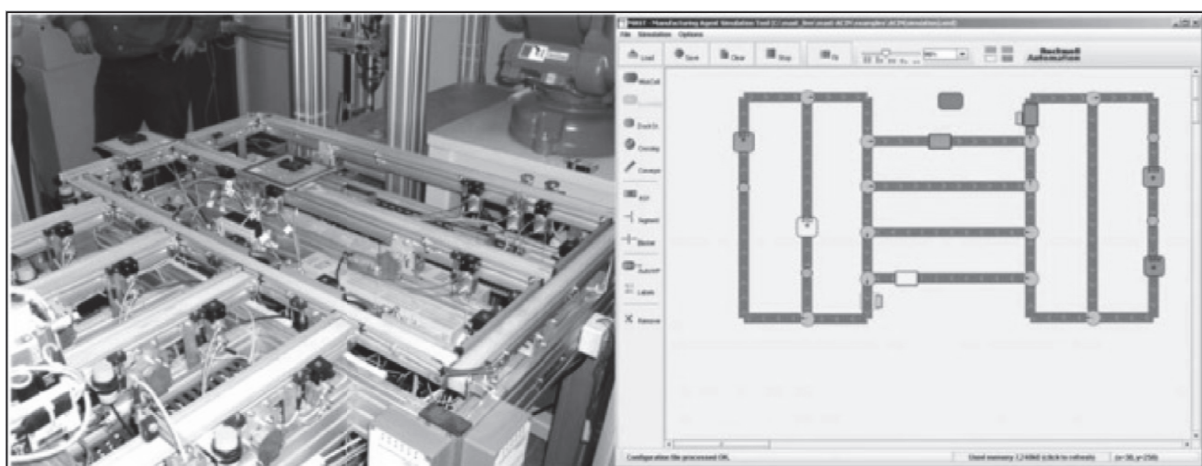


Figura 2.22. Sistemas de transporte usando MAS (VRBA; MARIK, 2010).

2.6.4. MAS em sistemas de manufatura integrados e distribuídos.

Este é um estudo realizado por José Barata, Raymond Boissier, Camarinha-Matos, Mohammed Raddadi e Francisco Restivo, sob o título “Integrated And Distributed Manufacturing, A Multi-Agent Perspective” (BARATA et al., 2001). Neste trabalho os autores propõem o desenvolvimento de aplicações de manufatura distribuída com o uso de sistemas multiagentes. Os agentes podem ser tanto um componente de software como de hardware, capaz de agir autonomamente e de tomar decisões que facilitem a realização das tarefas esperadas.

Os dispositivos físicos como sensores, robôs e máquinas CNC necessitam de uma interface que permita a comunicação com a lógica de algum agente inteligente. Os autores apontam este processo como “agentificação”, ou seja, tornar um componente físico um agente. A Fig. 2.23 apresenta o esquema deste processo.

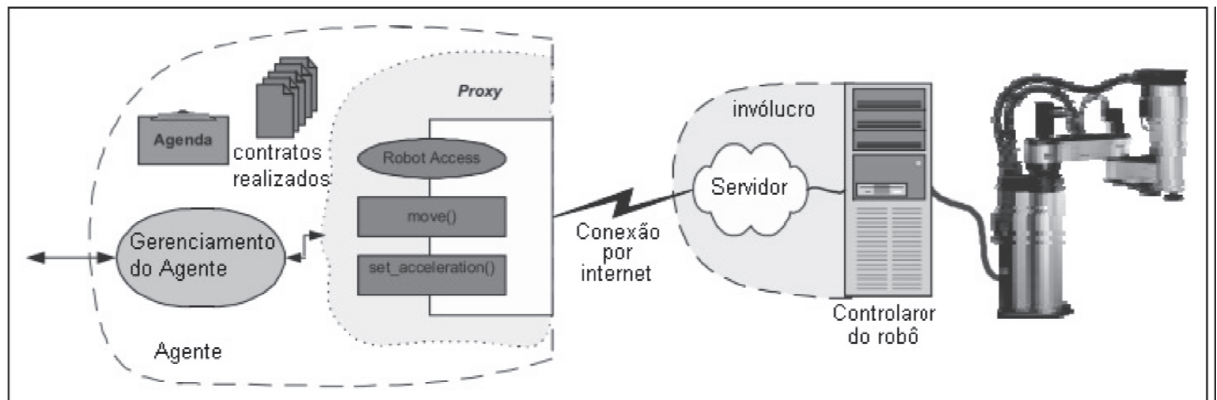


Figura 2.23. Integração dos componentes físicos no MAS (BARATA et al., 2001).

No estudo de caso abordado, os autores mostram o gerenciamento do agente se conectando ao controlador do robô através de um *proxy* e servidor, por meio da internet. O processo de “agentificação” ocorre de forma remota.

CAPÍTULO III

PROJETO PROPOSTO: PLANPAS

Para Sahin e Bolat (2009), as palavras “manufatura” e “qualidade” são sinônimos virtuais. Garantir maior qualidade ao produto e agir em prol de um processo de manufatura mais eficiente são ainda temas em aberto. Talvez por isso, cada vez mais as grandes companhias do ramo industrial vêm dando importância ao desenvolvimento de sistemas de controle de processo.

Historicamente, pode-se dizer que as principais empresas do setor industrial, no intuito de automatizar seus processos produtivos, concentraram praticamente todas as suas energias em sistemas de controle baseados em CLPs e PCs com sistema operacional Windows®. Em 1973, o surgimento do sistema SCADA permitiu a comunicação entre diversos dispositivos de controle. Entretanto, essa comunicação era dependente de *drivers* privados, o que onerava o processo de integração. Este tipo de *driver* ficou caracterizado negativamente, principalmente pelo alto custo e pela especificidade, visto que, embora houvesse vários fornecedores, cada um era caracterizado por equipamentos e linguagens específicos, dificultando a integração de sistemas.

Na segunda metade da década de 90, o surgimento do padrão OPC revolucionou o processo de comunicação entre dispositivos de controle em um ambiente industrial. O padrão OPC permitiu que usuários monitorassem e controlassem diferentes tipos e modelos de equipamentos através da utilização de um servidor OPC instalado em uma plataforma Windows®. O preço mais acessível, quando comparado ao sistema SCADA, aliado com a falta de imposição com relação ao tipo ou modelo de equipamentos utilizados conferiu grande vantagem aos pacotes de software OPC frente ao SCADA.

Similarmente, há anos diversos estudos têm apresentado técnicas de inteligência artificial (IA), tais como: redes neurais, algoritmos genéticos, sistemas especialistas, lógica *fuzzy*, planejamento automático, entre outras, como propostas de melhorias para sistemas práticos, inclusive em ambiente industrial. Por exemplo, em estudo recente, Dumitru et al.

(2010) mostraram que técnicas de IA, mais especificamente, lógica *fuzzy*, podem ser aplicadas diretamente em sistemas controlados por CLP, integrando controle clássico com tecnologia moderna. Já Trsek et al. (2007) comprovaram que a utilização de rede sem fio vem se mostrando uma poderosa ferramenta para integrar dispositivos de campo, tais como sensores e atuadores *wireless*. Além desses, na mesma linha de desenvolvimento, é possível citar ainda o desenvolvimento de uma nova plataforma de software que promove a integração de máquinas CNC, como apresentado por Nassehi et al. (2006).

Da mesma maneira, sabe-se que a comunidade de planejamento automático está muito comprometida em aplicar os avanços obtidos computacionalmente e academicamente em aplicações reais de complexidade relevante. Entretanto, problemas de planejamento complexos trazem grandes desafios, não somente para projetistas durante o processo de modelagem, como também para os planejadores automáticos durante o processo de busca pelo melhor plano.

Conforme já relatado anteriormente, recentemente alguns trabalhos tiveram relevância ao focar a utilização de planejamento automático em aplicações práticas a partir da utilização do sistema itSIMPLE para modelagem dos domínios. Dentre estes trabalhos podem ser citados: resolução de problemas logísticos em portos (DAHALL et al., 2003), logística inerente ao processo de carga e descarga de petróleo no porto de São Sebastião (SETTE et al., 2008), bombeamento de petróleo cru em tubulações (LI et al., 2005) e o desenvolvimento de uma bancada didática que avalia soluções de planejamento automático em um sistema controlado por CLP (FONSECA, 2011).

Entretanto, quando a análise é restrita a controladores industriais, tipo os CLPs, nota-se que os dados fornecidos por planejadores automáticos vêm sendo utilizados apenas como uma ferramenta auxiliar no nível tático, servindo somente como uma referência para o programador escrever o programa no controlador, em outras palavras, ainda não há uma assistência direta do planejamento automático no nível operacional das aplicações de automação. Apesar das soluções fornecidas pelos planejadores serem válidas e condizentes com o esperado, avaliando sobretudo minimização de custos, atualmente este tipo de abordagem só foi encontrado em ambientes acadêmicos, computacionalmente ou em ambientes laboratoriais com condições controladas, longe das incertezas inerentes ao ambiente industrial. É necessário, então, um desenvolvimento maior no sentido de aplicar as soluções de planejamento automático diretamente nos dispositivos de campo para que as indústrias passem a avaliar a utilização desta técnica como uma alternativa para solução de seus problemas.

Esta abordagem foi apresentada em Tavares et al. (2011). Nela, propõem-se o desenvolvimento de um aplicativo computacional capaz de promover a integração entre uma

solução fornecida por um planejador automático e o nível operacional de sistemas reais baseados em CLPs. Porém, até então, nenhum sistema com estas características havia sido desenvolvido.

A principal ideia deste trabalho é desenvolver um software aplicativo, denominado “Analisador de Planos para Sistemas Automatizados Baseados em CLPs” ou “*Plans Parser for Automated Systems*” (PlanPAS), capaz de transcrever um “plano-solução”, dado por algum planejador automático, em uma série de acionamentos, diretamente relacionados aos atuadores de um sistema controlado por CLP. O enfoque, neste caso, é em orientação a objetos e análise de casos específicos. Em outras palavras, o PlanPAS deverá ser uma interface entre um planejador automático (nível tático) e um CLP (nível operacional), com a meta de submeter as ações do “plano-solução” diretamente às E/S digitais do CLP e, conseqüentemente, aos sensores e atuadores do sistema real.

3.1. PlanPAS

O PlanPAS consiste em um software aplicativo capaz de executar as ações planejadas, habilitando e desabilitando saídas digitais do CLP, a partir de uma comparação realizada entre as pré-condições e os efeitos de cada ação planejada com os valores das entradas digitais/analógicas do CLP. Quando os efeitos de uma determinada ação forem válidos, ou seja, quando o estado planejado for igual ao estado atingido, o sistema disparará a próxima ação. Caso o sistema real atinja um estado não esperado, o problema deverá ser replanejado, considerando um novo cenário inicial. A representação esquemática do projeto é ilustrada na Fig. 3.1. As setas alaranjadas indicam processos que ocorrem *online*, enquanto as setas azuis indicam os processos que ocorrem *offline*. É importante salientar que a operação de replanejamento ainda está em fase de desenvolvimento e não será abordada neste trabalho.

Na Fig. 3.1 o retângulo tracejado na cor azul realça as atividades desenvolvidas dentro do sistema itSIMPLE, enquanto que as funcionalidades do sistema PlanPAS estão inseridas no retângulo tracejado na cor cinza. A abordagem para representação do sistema real leva em consideração a modelagem do domínio de planejamento através de diagramas UML utilizando o itSIMPLE. Os cenários, inicial e final, também são definidos em diagramas de objetos UML, denominados *snapshots*. A partir desta modelagem, o itSIMPLE é capaz de traduzir o problema de planejamento de uma representação em UML para PDDL. A partir da representação do domínio em PDDL, os planejadores automáticos estão aptos a analisar os problemas e chegar a uma solução possível, caso esta exista. A saída destes planejadores

corresponde a um arquivo XML com uma lista de ações que leva o sistema modelado do estado inicial ao estado final, considerando as restrições e métricas de otimização impostas.

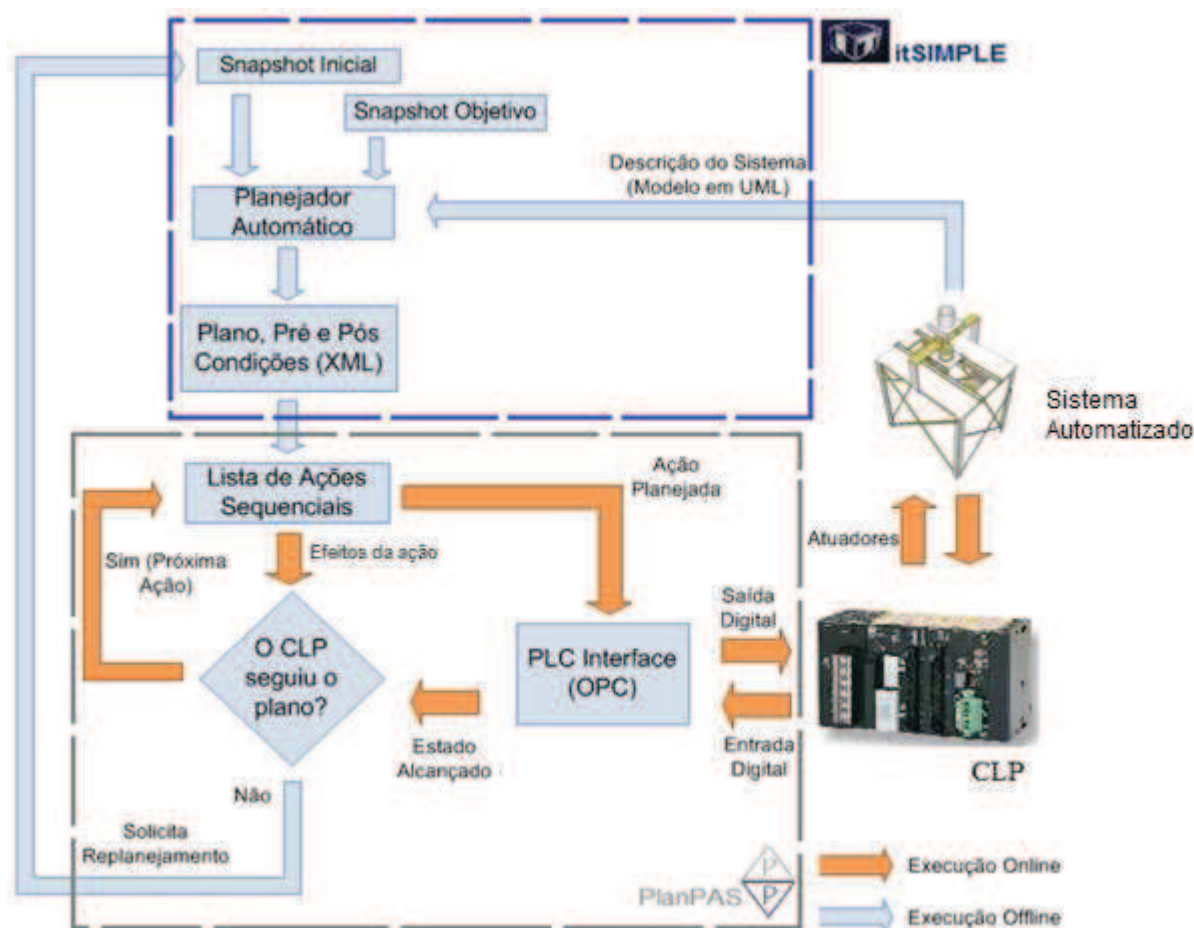


Figura 3.1. Representação esquemática do projeto PlanPAS (traduzido de Fonseca et al. (2013)).

Para aplicar as ações planejadas em um sistema prático de forma sequencial, é necessário ter a informação dos efeitos ou pós-condições de cada ação do plano. Ou seja, na prática, as pós-condições contém os estados intermediários, e operam como transições para a habilitação uma nova ação do plano. No entanto, o arquivo XML com o “plano-solução” gerado pelo itSIMPLE fornece apenas informações referentes às ações planejadas. Para ter acesso a um arquivo XML contendo, além das ações referentes ao plano, as pré-condições e os efeitos (estado previsto) de cada ação, foi necessária uma pequena implementação dentro do código fonte do itSIMPLE. Com isso, o arquivo XML que armazenava somente as ações do “plano-solução”, passou a inserir as pré-condições e os efeitos para cada uma dessas ações no corpo do arquivo. A Fig. 3.2 apresenta a estrutura original de armazenagem dos parâmetros em XML para a ação LOAD (V1 F1 LEVEL8)

enquanto a Fig. 3.3 apresenta a estrutura de armazenagem para a mesma ação, considerando, porém, as suas pré e pós-condições.

Na Fig. 3.2, a tag *<parameters>* armazena os parâmetros da ação LOAD, sendo eles: V1, F1, e LEVEL8, gerando a ação LOAD (V1 F1 LEVEL8). Na Fig. 3.3, além da tag *<parameters>*, existem duas outras tags (*<precondition>* e *<effect>*) necessárias, pois armazenam informações relevantes para a correta operação do PlanPAS. A tag *<precondition>* armazena as pré-condições para disparar a ação LOAD (V1 F1 LEVEL8), enquanto a tag *<effect>* armazena os efeitos da execução desta ação. Após a ação ser disparada, o aplicativo proposto deve ser capaz de comparar os valores das entradas do CLP, os quais representam o estado real do sistema, com as informações armazenadas na tag *<effect>*. A partir do momento em que o sistema atingir o estado esperado, ou seja, quando os valores das entradas do CLP se equipararem com aqueles armazenados na tag *<effect>*, o aplicativo deverá desabilitar o atuador responsável pela ação LOAD e disparar a próxima ação do plano. Na prática, este sistema irá desligar uma saída digital do CLP e “setar” outra.

Uma vez que o plano seja analisado, faz-se necessário estabelecer comunicação com os dispositivos que compõe o chão de fábrica para transpor a lista de ações teóricas em um processo de fato. Em automação industrial, arquiteturas de comunicação complexas e restrições quanto à escolha de dispositivos de campo tem forçado o setor de manufatura a buscar por novas soluções para seus problemas. Esta foi uma das principais razões para o surgimento do padrão OPC, lançado em 1996. Além disso, de acordo com Sahin e Bolat (2009) a tecnologia OLE/COM, uma tecnologia cliente/servidor da Microsoft®, opera de acordo com a estrutura base do padrão OPC.

```
<action id="LOAD">
  <parameters>
    <parameter id="V1" />
    <parameter id="F1" />
    <parameter id="LEVEL8" />
  </parameters>
  <startTime>0.001</startTime>
  <duration>1</duration>
  <notes />
</action>
```

Figura 3.2. Representação da Ação LOAD (V1 F1 LEVEL8) em XML sem pré e pós-condições.

```

<action id="LOAD">
  <parameters>
    <parameter id="V1" />
    <parameter id="F1" />
    <parameter id="LEVEL8" />
  </parameters>
  <startTime>0</startTime>
  <duration>1</duration>
  <notes />
  <precondition>    //pré-condição
    <and>
      <predicate id="isAt">
        <parameter id="V1" />
        <parameter id="F1" />
      </predicate>
      <ge>
        <function id="maxlevel">
          <parameter id="V1" />
        </function>
        <add>
          <function id="level">
            <parameter id="V1" />
          </function>
          <function id="amount_transferred">
            <parameter id="LEVEL8" />
          </function>
        </add>
      </ge>
    </and>
  </precondition>    //pré-condição
  <effect>            //pós-condição
    <increase>
      <function id="level">
        <parameter id="V1" />
      </function>
      <function id="amount_transferred">
        <parameter id="LEVEL8" />
      </function>
    </increase>
  </effect>            //pós-condição
</action>

```

Figura 3.3. Representação da Ação LOAD (V1 F1 LEVEL8) em XML após a inserção de pré e pós-condições.

O acesso ao CLP é realizado através de uma interface baseada nos recursos do padrão OPC. Na verdade, foi desenvolvido um cliente que se comunica com um servidor através da arquitetura cliente/servidor OPC. O acesso às *tags* do CLP pode ser realizado de dois modos distintos: somente leitura ou leitura/escrita. Normalmente o modo de “leitura/escrita” é habilitado nas *tags* correspondentes às saídas digitais, enquanto que as *tags* correspondentes às entradas analógicas e digitais utilizam o modo de acesso “somente leitura” para acessar as informações dos sensores do ambiente. Esta interface passará a ser denominada “PLC Interface”.

A *PLC Interface* comunica-se diretamente com um “Comparador” de estados, denominado aqui de *Parser* – representado no fluxograma da Fig. 3.1 pela decisão “O CLP seguiu o plano?”. Na prática, o *Parser* é o responsável por avaliar se o sistema está seguindo as ações planejadas, ou seja, se os estados intermediários atingidos correspondem àqueles planejados. Esta avaliação ocorre através da comparação entre os dados provenientes das *tags* correspondentes às entradas analógicas/digitais definidas no servidor OPC e as informações de pré-condições e efeitos para cada ação, obtidas do arquivo XML contendo o “plano-solução”. O *Parser* também é encarregado de enviar mensagens para a *PLC Interface* disparando novas ações. Para cada estado esperado atingido, o *Parser* envia uma mensagem habilitando uma *tag* e desabilitando outra, correspondentes às saídas digitais vinculadas às ações que se deseja habilitar e desabilitar.

Do ponto de vista da arquitetura cliente/servidor, a *PLC Interface* pode assumir dois papéis distintos, dependendo do padrão de conexão considerado. Na arquitetura TCP/IP, a *PLC Interface* age como um servidor, ouvindo alguma porta específica aguardando por uma mensagem composta por comandos XML e retorna uma resposta, formatada da mesma maneira, contendo os dados requisitados, enquanto o *Parser* executa papel de cliente. Já na arquitetura OPC, a *PLC Interface* assume o papel de cliente, enquanto o servidor passa a ser um servidor OPC. Neste trabalho, optou-se por utilizar o servidor OPC para CLPs Omron fornecido pela Matrikon®. Tal servidor é denominado *MatrikonOPC Server for Omron.PLCs* (MATRIKON OPC, 2011).

A *PLC Interface* é basicamente uma ponte capaz de receber mensagens com comandos estruturados em XML do *Parser*, estabelecendo comunicação com o servidor OPC de modo a impor as ações planejadas e retornar uma mensagem contendo os dados adquiridos das *tags* relacionadas às entradas do CLP, que representam o estado do sistema real. A Fig. 3.4 detalha o esquema de comunicação observada no projeto do PlanPAS.

Através desta ideia de comunicação via ponte – no caso a *PLC Interface* – o PlanPAS pôde ser concebido sem maiores preocupações com respeito aos protocolos de comunicação específicos dos equipamentos de campo, bem como com uma maior autonomia em suporte, ao permitir que sistemas adicionais possam ser facilmente adicionados. Este foi o caso do *Parser* que foi desenvolvido em Java, uma linguagem de programação aberta e independente de Sistema Operacional.

O desenvolvimento do PlanPAS focou nas iterações entre ações planejadas e saídas digitais do CLP, bem como entre as pré-condições e efeitos de cada ação e as entradas digitais/analógicas do CLP. Para estabelecer esta ligação, fez-se necessário o desenvolvimento de uma tabela de relacionamento, ou uma “tabela de-para”, com o levantamento de todas as ações possíveis no sistema a ser controlado. A Tab. 3.1

apresenta um exemplo genérico de uma tabela de relacionamento para um sistema de distribuição e logística. Neste caso, a execução da ação “Load” está diretamente relacionada com a saída digital “output1”, a ação “Unload” com a saída digital “output2” e as ações “MoveLeft” e “MoveRigth” com as saídas “output3” e “output4”, respectivamente.

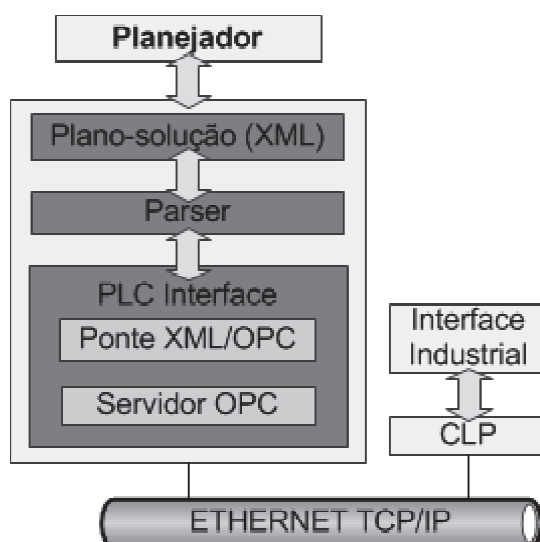


Figura 3.4. Esquema de comunicação presente no PlanPAS (traduzido de Fonseca et al. (2013)).

Na prática, o *Parser* acessa um arquivo XML com o “plano-solução” de um determinado problema e, para cada ação, envia uma mensagem através de conexão TCP/IP com a saída específica (que deverá ser habilitada) para a *PLC Interface*. Esta, por sua vez, estabelece comunicação através do padrão OPC com o servidor OPC do CLP em serviço, modificando o valor da *tag* correspondente à saída relacionada com a ação planejada. Por exemplo, para executar a ação *Load*, o *Parser* verifica qual a próxima ação e acessa a tabela de relacionamento para encontrar a saída do CLP correspondente, neste caso, a saída “output1”. Em seguida, o *Parser* inicia uma conexão TCP/IP com a PLC interface e envia uma mensagem contendo a saída a ser habilitada (“outuput1”) e permanece aguardando uma mensagem de confirmação. Neste momento, a PLC interface analisa qual *tag* possui o mesmo valor da mensagem recebida (“output1”), modificando seu valor, que inicialmente era *false* para *true* – isto dispara um sinal de tensão que habilita o atuador responsável por executar a ação “Load” no sistema real. Neste momento, a *PLC Interface* envia uma resposta ao *Parser* com a mensagem “Ação enviada com sucesso” e, a partir de então, a *PLC Interface* passa a enviar os dados armazenados nas entradas analógicas/digitais do CLP, indicando o estado do sistema real. Concomitantemente, o *Parser* passa então a comparar os dados dos sensores com as pós-condições da ação

“*Load*”; no momento em que o sistema atinge o estado planejado, o *Parser* percebe e envia uma nova mensagem para a PLC Interface para desabilitar a saída “*output1*” e, caso esta não seja a última ação do plano, disparar a próxima ação planejada.

Tabela 3.1. Exemplo genérico de uma tabela de relacionamento.

Ação Planejada	Saída do CLP
<i>Load</i>	<i>output1</i>
<i>Unload</i>	<i>output2</i>
<i>MoveLeft</i>	<i>output3</i>
<i>MoveRigth</i>	<i>output4</i>

Dessa forma, nota-se que a lógica de controle do sistema é definida pela sequência de ações dada pelo planejador automático, enquanto o CLP passa a operar apenas como um *driver* do ambiente, habilitando ou desabilitando as suas saídas e acessando os dados das suas entradas.

Para garantir maior segurança operacional, optou-se por inserir os intertravamentos diretamente no programa do CLP, uma vez que esta lógica não será afetada por atrasos causados por congestionamento ou indisponibilidade da rede, aumentando com isso a robustez do sistema.

Do ponto de vista funcional, o sistema PlanPAS proposto se adapta perfeitamente entre os níveis de planejamento e operacional na pirâmide da automação. A Fig. 3.5 apresenta uma pirâmide da automação modificada com a execução do PlanPAS. Os protocolos de comunicação utilizados neste caso são rede Ethernet TCP/IP para comunicação entre o *Comparador (Parser)* e a *Ponte OPC (PLC Interface)* e Ethernet OPC para acesso da *Ponte OPC* diretamente ao CLP, através de um servidor OPC.

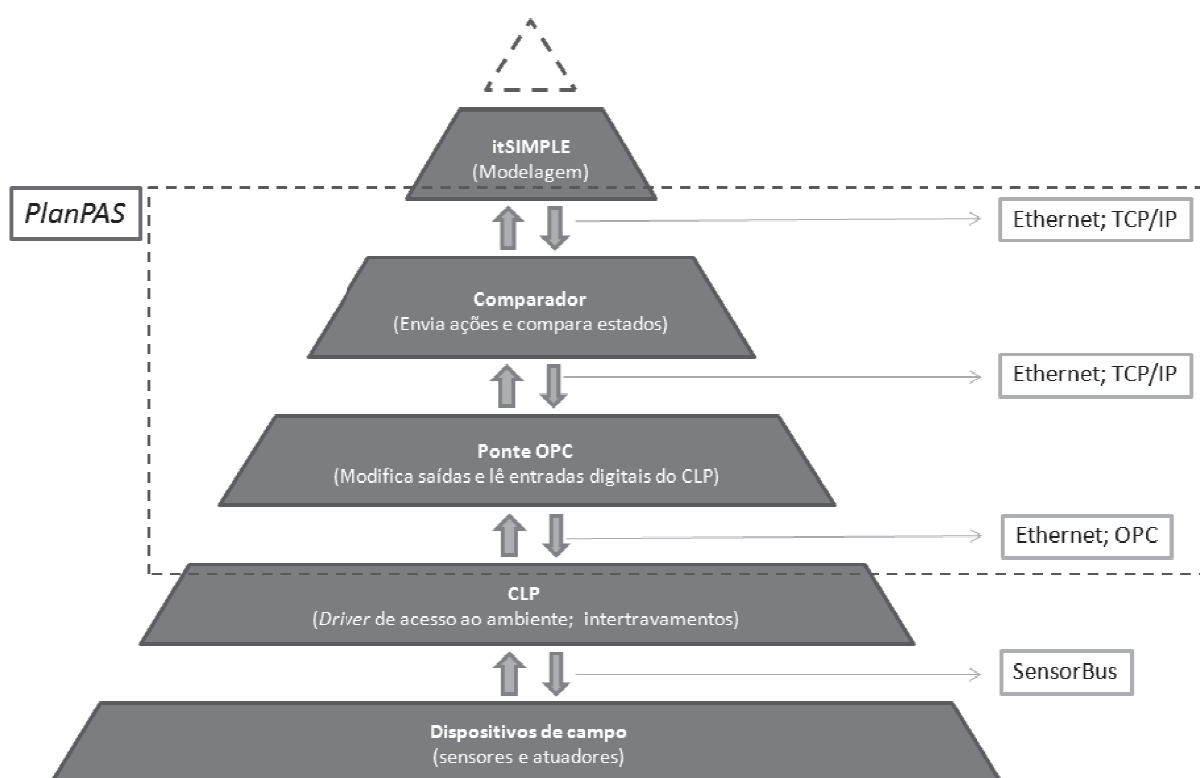


Figura 3.5. Funcionalidade do PlanPAS na pirâmide da automação.

3.2. Desenvolvimento do Software Aplicativo PlanPAS

O projeto do PlanPAS foi desenvolvido em duas partes. Primeiramente focou-se no desenvolvimento da *PLC Interface*, principalmente com comunicação OPC – esta etapa foi realizada em parceria com um aluno de graduação sob orientação e supervisão do autor. Em seguida, o foco passou para o desenvolvimento do *Parser*, principalmente com a tabela de relacionamento e a comunicação TCP/IP. Dessa forma, esta seção está dividida em duas subseções. Na subseção 3.2.1 é apresentado o processo de desenvolvimento da *PLC Interface*, enquanto que o desenvolvimento do *Parser* é abordado na subseção 3.2.2.

3.2.1. Desenvolvimento da PLC Interface

Para atingir os objetivos propostos, o código da *PLC Interface* foi dividido em três etapas. A primeira delas foi a comunicação OPC-DA e respectivos testes. Em seguida, partiu-se para a criação de classes auxiliares responsáveis por desafogar o gerenciamento das conexões de rede, múltiplas *threads* e arquivos XML. Por fim, levantou-se a necessidade de se ter algum tipo de interface visual, logo, a terceira etapa se concentrou na

interface do aplicativo cliente e a sua correta integração com as classes criadas anteriormente.

A preparação do programa para a comunicação OPC consiste, inicialmente, na criação de classes para encapsular os objetos COM do servidor. Dessa maneira, não há necessidade que o código principal execute chamadas de OLE/COM a todo o momento durante a execução, visto que as classes foram desenvolvidas para gerenciar todos estes detalhes. Além disso, as classes foram estruturadas de uma forma tal que o restante do código não é responsável pela limpeza de memória temporal ou pela alocação ou liberação de recursos. Centralizando estas operações nas novas classes, reduz-se a probabilidade de vazamento de recursos em caso de falhas.

Um contratempo com a arquitetura COM exigiu uma solução alternativa para a manipulação de objetos. Os objetos COM não podem ser usados em *threads* diferentes daquelas onde foram instanciados. Tal problema pôde ser contornado através da utilização de um artifício conhecido como *marshalling*. Esta solução foi implementada de uma maneira tão transparente que a função chamada foi desincumbida desta tarefa; a função passa a realizar simplesmente a chamada de métodos, e qualquer necessidade de *marshalling* é processada automaticamente.

Foi implementada uma classe denominada *OPCServer* cujos objetos são responsáveis pelo estabelecimento da comunicação entre o software e um servidor OPC e pela criação de grupos de itens. Estes grupos, representados pela classe *OPCItemGroup*, que contém os dados dos itens propriamente ditos, são incorporados por objetos da classe *OPCItem*, que contém métodos para ler e gravar dados no servidor.

Para uma melhor manipulação dos recursos, diversas classes auxiliares foram criadas. O primeiro grupo deste tipo de classes é responsável pela comunicação TCP/IP através de *sockets*. Tanto os fluxos de dados, quanto as conexões de escuta foram implementados em classes *Socket* e, por meio de especialização, nas classes *ServerSocket*, respectivamente. Além disso, foram realizadas implementações de modo que a leitura e escrita de dados em XML através de *sockets* fossem realizadas mais facilmente.

Para implementar suporte para múltiplas *threads*, foi criada uma classe genérica denominada *Thread*. A principal característica desta classe é que ela contém um método que é executado em uma *thread* exclusiva para o objeto. Por se tratar de uma classe especializada, surgiu a necessidade de novas classes derivadas, sendo uma para a tarefa de escuta da rede e outra para transações cliente/servidor. Ambas fazendo uso das classes de comunicação apresentadas anteriormente.

Finalmente, as classes de manipulação XML em OLE/COM empacotadas com o sistema operacional foram encapsuladas de tal forma que possam ser facilmente utilizadas por outros componentes.

Uma classe denominada *NetBridge* foi implementada para instanciar todas outras classes funcionais, não incluindo a interface, sendo a responsável pelo gerenciamento de objetos. Esta centralização de gerenciamento permite que o software opere de uma maneira unificada. Esta integração é representada através de um diagrama de classes simplificado na Fig. 3.6.

Como dito, a terceira etapa do desenvolvimento da *PLC Interface* se concentrou na interface do aplicativo cliente e a sua correta integração com as classes criadas anteriormente. A Fig. 3.7 apresenta a imagem da *PLC Interface* rodando lado a lado com o servidor *MatrikonOPC Server for Omron PLCs* (MATRIKON OPC, 2011).

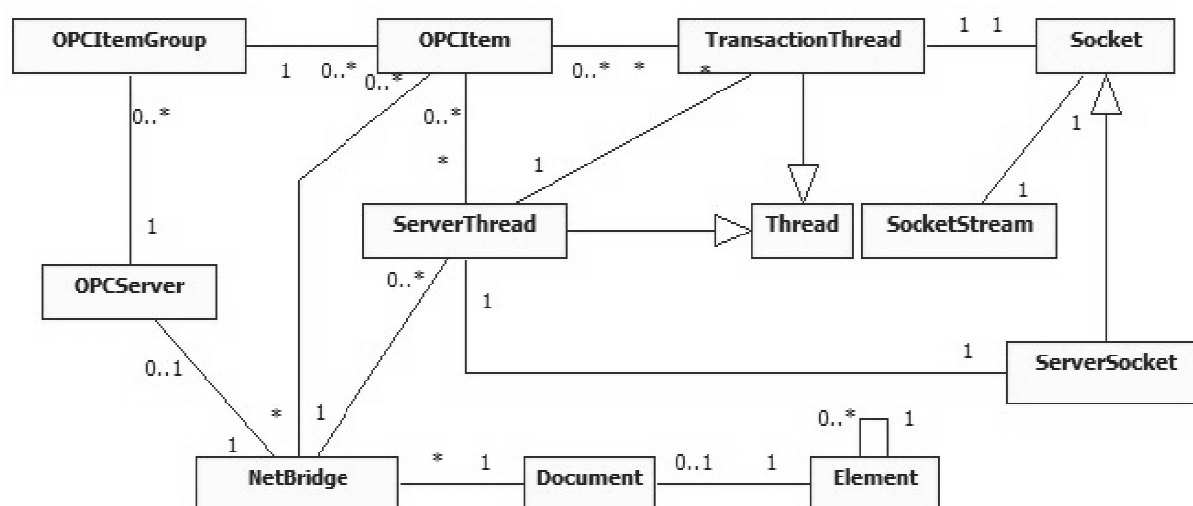


Figura 3.6. Diagrama de classes simplificado da *PLC Interface*.

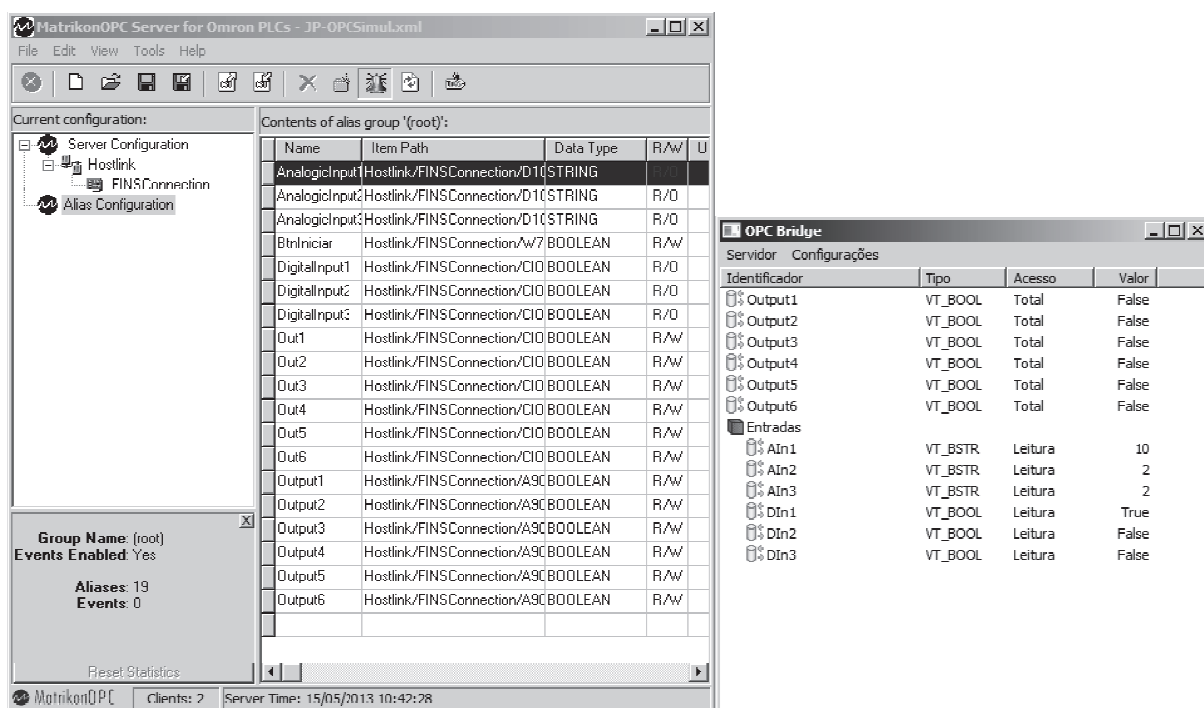


Figura 3.7. Servidor OPC e PLC Interface rodando lado a lado.

3.2.2. Desenvolvimento do Parser

Assim como no desenvolvimento da *PLC Interface*, o código do *Parser* também foi dividido em etapas. Primeiramente, o principal objetivo foi implementar a leitura e interpretação de arquivos de extensão .XML, visto que o “plano-solução” – dado por algum planejador embarcado no itSIMPLE – é armazenado em um arquivo deste tipo. Para isto, foram implementadas classes Java responsáveis pelas tarefas de leitura e escrita de arquivos e fluxos no padrão XML.

Em seguida, os esforços foram concentrados na comunicação TCP/IP entre o *Parser* e a *PLC Interface*. Nesta etapa, definiram-se os endereços IP aonde cada aplicativo (*Parser* e *PLC Interface*) viria a operar, bem como a porta de comunicação e troca de mensagens.

A última etapa foi o desenvolvimento da interface do aplicativo cliente integrando as classes criadas até então. Aqui, foi criado um formulário Java com a estrutura da interface do *Parser*, contendo uma tabela de relacionamentos, uma tabela de monitoração e diversos botões que promovem as funcionalidades de configuração do *Parser* e comunicação com a *PLC Interface*. Todo o software foi desenvolvido através do ambiente NetBeans (IDE 7.1.1). O diagrama de classes para *Parser* é mostrado na Fig. 3.8.

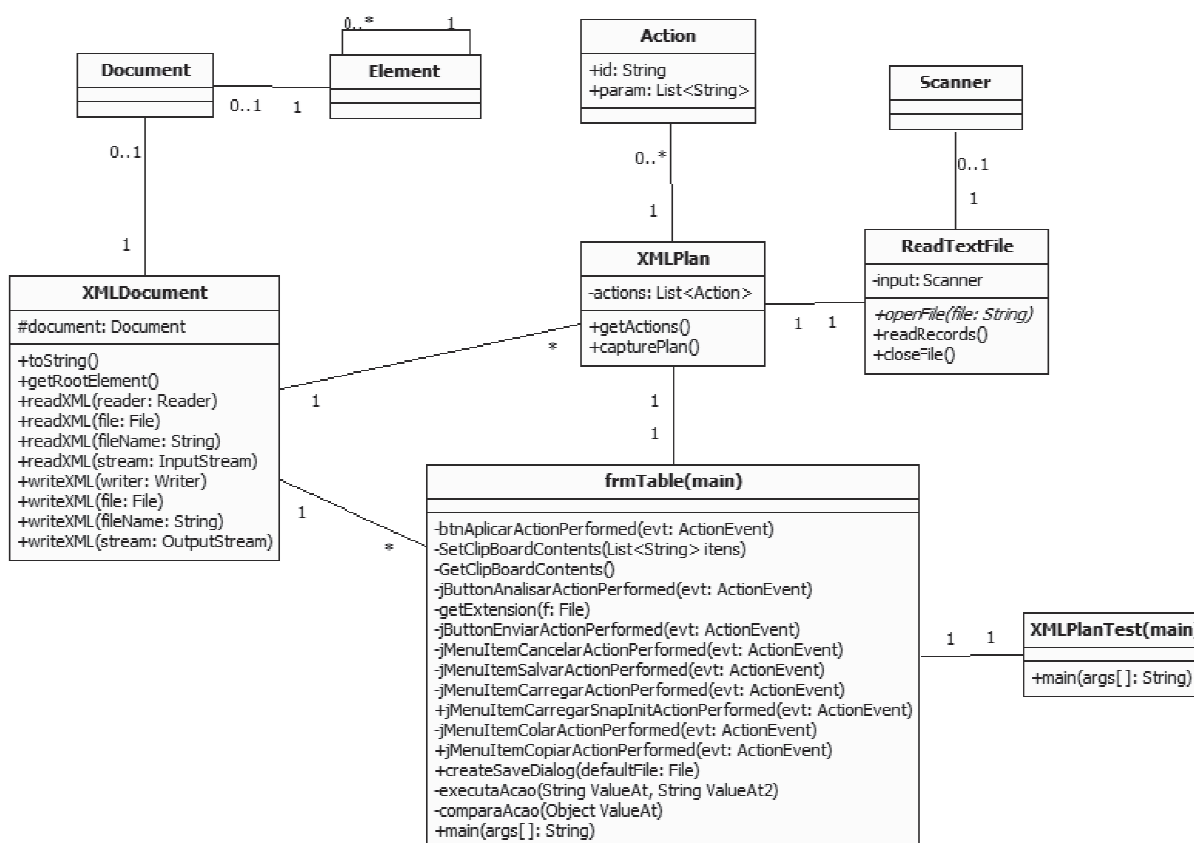


Figura 3.8. Diagrama de classes para o *Parser* (FONSECA et al., 2013).

Sintetizando o funcionamento do *Parser*, podemos dizer que o mesmo opera da seguinte maneira: a classe *XMLDocument* é a principal responsável pela tarefa de ler e escrever arquivos e fluxos no padrão XML; a classe *XMLPlan* é responsável por obter as ações e capturar o “plano-solução” do arquivo XML analisado; a classe *frmTable* implementa a interface do *Parser* bem como as funcionalidades de comunicação; *jButtons* e *jMenuItems* foram criados de maneira a permitir que diversas funcionalidades, tais como: aplicar determinado número de linhas à tabela de relacionamentos, analisar a tabela de relacionamento em busca de saídas do CLP correspondentes às ações planejadas, enviar uma mensagem para a *PLC Interface* contendo qual saída deverá ser habilitada e receber um *socket* de confirmação, entre outras funcionalidades. A Fig. 3.9 ilustra parte do código referente à programação da interface do PlanPAS, ilustrada na Fig. 3.10.

Através da interface ilustrada na Fig. 3.10, o usuário é capaz de realizar as atividades apresentadas no diagrama de atividades da Fig. 3.11. Inicialmente, o usuário deve carregar um arquivo de extensão .XML contendo o “plano-solução” para o problema que se deseja solucionar. Em seguida, o usuário deve inserir a tabela de relacionamento, ou “tabela de-para”, referente ao domínio abordado. Para isso, o usuário pode escrever diretamente a tabela, linha por linha ou, caso esta tabela já tenha sido armazenada em um arquivo de extensão .XML ou .TXT, o usuário pode carregar este arquivo para a tabela de relacionamentos da interface.

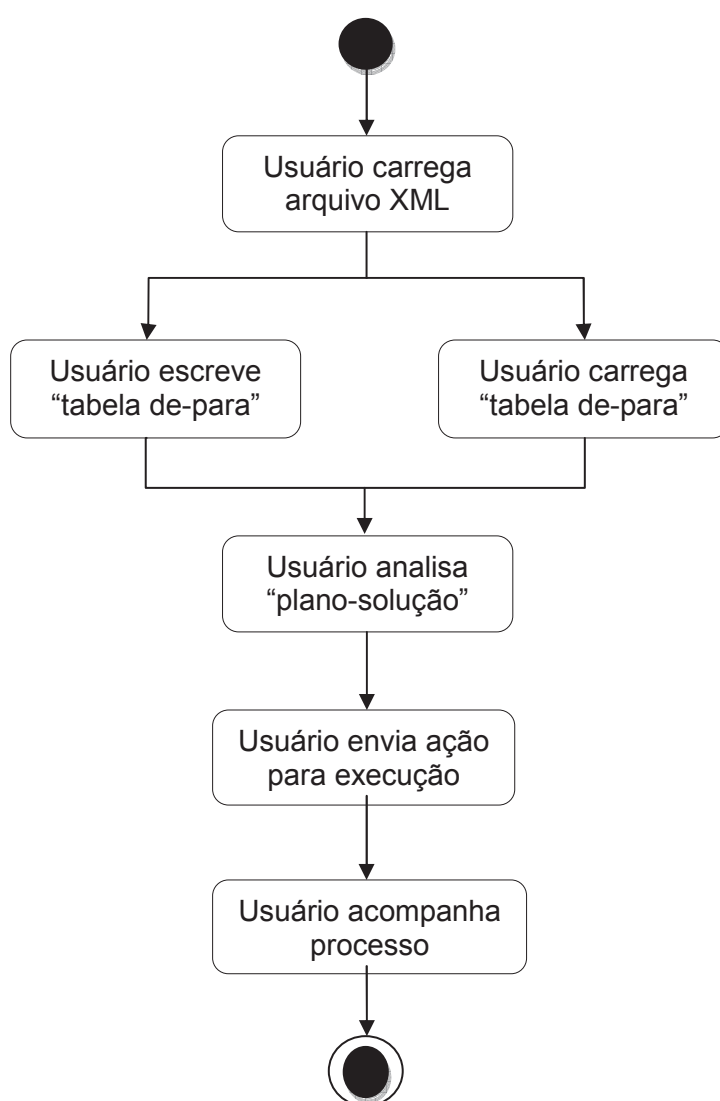


Figura 3.11. Diagrama de atividades para o usuário do PlanPAS.

Carregados o “plano-solução” e a “tabela de-para”, dizemos que o PlanPAS encontra-se configurado para operar no domínio definido no modelo. A partir daí, o usuário deve “analisar” o “plano-solução” – tarefa na qual o sistema PlanPAS determina as saídas digitais

do CLP correspondentes às ações planejadas. Após “analisar” o “plano-solução”, o usuário está habilitado a enviar a primeira ação planejada para execução no sistema real. A partir deste momento, o sistema age no sentido de executar todas as ações planejadas e o usuário passa a acompanhar o processo e monitorar a evolução dos estados intermediários até que o cenário objetivo seja atingido, ou alguma mensagem de erro seja lançada na tela. Esta mensagem de erro indicará que um estado intermediário planejado não foi atingido pelo sistema real e a operação é abortada. No melhor dos mundos, o sistema deveria ser capaz de perceber esta falha e solicitar um replanejamento ao planejador sem, no entanto, interromper a operação do sistema real. Esta tarefa está em estudo e ainda não foi implementado no projeto do PlanPAS.

O procedimento de configuração do PlanPAS para um problema genérico é apresentado detalhadamente no Apêndice A.