UNIFEI

Universidade Federal de Itajubá

AMBIENTE DE PROGRAMAÇÃO E INTEGRAÇÃO PARA MANUFATURA VIRTUAL "APIMV"

Área de Automação e Sistemas Elétricos Industriais

Janaina Fernandes Rosa Arantes

AMBIENTE DE PROGRAMAÇÃO E INTEGRAÇÃO PARA MANUFATURA VIRTUAL "APIMV"

Orientador: Dr. Luiz Edival de Souza

Co-orientador: Dr. Leonardo de Mello Honório

DISSERTAÇÃO APRESENTADA À UNIVERSIDADE FEDERAL DE ITAJUBÁ PARA OBTENÇÃO DO TÍTULO DE MESTRE EM ENGENHARIA ELÉTRICA

ITAJUBÁ
ESTADO DE MINAS GERAIS – BRASIL
2007

Ficha catalográfica elaborada pela Biblioteca Mauá –
Bibliotecária Margareth Ribeiro- CRB_6/1700

A662a

Arantes, Janaína Fernandes Rosa
Ambiente de programação e integração para manufatura virtual "APIMV" / Janaína Fernandes Rosa Arantes. – Itajubá, (MG) : UNIFEI, 2007.
123 p. : il.

Orientador : Prof. Dr. Luiz Edival de Souza.
Co-orientador : Prof. Dr. Leonardo de Mello Honório.
Dissertação (Mestrado) – Universidade Federal de Itajubá.

1. Manufatura virtual. 2. Controle de processos. 3. Progra_ mação industrial. 4. Diagrama Ladder. I. Souza, Luiz Edival de, orient. II. Honório, Leonardo de Mello, co-orient. III. Uni_ versidade Federal de Itajubá. IV. Título.

CDU 004.451.25(043)

COMISSÃO DE AVALIAÇÃO

Prof. Dr. Carlos Eduardo Cugnasca
1º Examinador - USP

Prof. Dr. Enio Roberto Ribeiro
2º Examinador - UNIFEI

Prof. Dr. Leonardo de Mello Honório
3º Examinador – (Co-Orientador) - UNIFEI

Prof. Dr. Luiz Edival de Souza
3º Examinador – (Orientador) - UNIFEI

Aprovado em:

14 de março de 2007

Dedico esse trabalho ao meu esposo, Edson, e a meus pais
pelo amor e pela sintonia que nos une.

Agradecimentos

À CAPES e ao CNPq, Projeto CT-Info/Cnpq 01/2003 – Processo: 400842/2003-3, pela disponibilização de bolsas e recursos computacionais sem os quais este projeto não poderia ser realizado.

Aos docentes, amigos e colegas que compõem o CRTi, pela colaboração, participação e interesse em contribuir para o sucesso deste projeto. Principalmente: ao meu Orientador, Professor Luiz Edival, pelo incentivo e orientação; ao meu Co-orientador, Professor Leonardo, pela dedicação ao coordenar o projeto; ao colega e amigo Welinton, por ser prestativo durante a implementação do *software*, à minha colega e amiga Daniele, pela colaboração e apoio e ao amigo Luiz Antônio pelo suporte técnico.

E em especial aos que contribuíram de alguma forma para que este projeto fosse concretizado, dentre os quais agradeço: a meus pais, que incentivaram a iniciativa de dar continuidade à minha formação profissional e ajudaram em questões particulares; meus enteados, que precisaram ser compreensivos e colaboraram muito comigo; e acima de tudo a meu marido, sempre dedicado, participativo e amável, que foi meu equilíbrio durante todo esse período.

Resumo

A constante evolução industrial e a diversividade da tecnologia geram uma demanda crescente de profissionais qualificados que atendam à flexibilidade de soluções para os requisitos e problemas das empresas. Diante disso, faz-se importante um sistema universal de simulação que possibilite reproduzir, programar, integrar e controlar diferentes objetos e instrumentos numa tela de computador, tal que proporcione uma idéia bastante verdadeira da realidade de um projeto, antes mesmo de sua implementação.

Com essa visão, o Centro de Referência em Tecnologia da Informação (CRTi), da Universidade Federal de Itajubá (UNIFEI), com o apoio do Centro Nacional de Pesquisas (CNPq), está desenvolvendo um sistema de manufatura virtual para simular processos industriais. O projeto se aplica tanto a instituições de ensino quanto a empresas, com propósito de transmitir a idéia real de um contexto idealizado para um sistema industrial servindo como base de treinamento de profissionais e auxílio de especialistas em tomadas de decisões no planejamento da produção.

O sistema conta com: maquetes virtuais, que simulam o mundo real através de animações gráficas, e um ambiente de programação para comandá-las.

O ambiente de programação é o tema dessa dissertação. Trata-se de um *software* padronizado, para desenvolvimento de programas nas linguagens industriais previstas na Norma IEC61131-3. Integrado a Maquetes Virtuais de simulação de processos, responsabiliza-se pela programação da lógica de controle que comanda a movimentação dos objetos nela representados, criando assim uma realidade virtual.

Destinado, inicialmente, ao desenvolvimento de lógicas de controle empregando linguagem Ladder, foi desenvolvido nesta dissertação uma ferramenta gráfica de programação com recursos de edição, interpretação e simulação da lógica baseada em símbolos de relés e blocos de função. A ferramenta de programação desenvolvida oferece um conjunto de instruções que permite ao usuário programar a lógica de funcionamento de um processo ou de uma máquina através de funções booleanas (E, OU e Negação), temporizadores, contadores, comparadores e funções aritméticas. Este conjunto é formado por vários elementos de programação, entre eles, os símbolos de contato de relé NA (normal aberto) e NF (normal fechado) que quando

adequadamente combinado em ligações série e paralelo executam as funções booleanas E e OU respectivamente. Juntamente às funções de cada elemento de programação, administradas por um algoritmo de interpretação e gerenciamento, foram criadas as dinâmicas de tela e imagens (figuras ilustrativas) necessárias para compor a relação lógica entre os objetos e, assim, implementarem as linhas de comando.

Para que um elemento seja adicionado ao diagrama é apresentada uma imagem genérica segundo o tipo de componente escolhido, que pode ser movida no espaço de edição para determinar o ponto de fixação do componente. Uma vez fixado na tela, são atribuídos dados característicos que permitem a identificação desse componente dentro da lógica. O posicionamento na tela especifica pontos de conexão permitindo a ligação entre os elementos, enquanto as simbologias gráficas definem o comportamento dos elementos na lógica de controle.

Os comandos de manipulação do ambiente são totalmente gráficos, acionados por botões. Podem-se editar elementos de programação e linhas de comando, executar ou simular um diagrama completo ou linhas individuais e, além disso, possibilidade de integração com maquetes virtuais de um processo industrial. As variáveis podem ser reconhecidas da maquete, carregadas do simulador vinculado a esse ambiente ou declaradas pelo usuário.

O ambiente de programação é um aplicativo simples, fácil de ser usado, independente de equipamento ou fabricante, e tem a vantagem de se adaptar a qualquer máquina por ser um *software* aberto e contar com uma arquitetura flexível. As lógicas nele implementadas podem ser armazenadas em arquivos de forma completa (diagrama) ou fracionada (linhas de comando independentes) num caminho escolhido pelo usuário.

Possui recursos que a maioria dos *softwares* similares a ele não apresentam, como: módulo simulador de entradas e saídas, incluindo a possibilidade de simulação individual de cada linha de comando; disposição de linguagens industriais de programação regulamentadas; e o mais importante, exerce comunicação com maquetes virtuais.

Abstract

The constant industrial evolution and the variety of technologies generate an increasing demand of qualified professionals for the flexibility of solutions related to the requirements and problems of the companies. Therefore, a universal system of simulation to make possible reproduction, programming, integration and control of different objects and instruments in a computer screen is important, to form a true idea of the reality of a project, before its implementation.

With this vision, the Center of Reference in Technology of the Information (CRTi), at Federal University of Itajubá (UNIFEI), with support of the National Center of Research (CNPq), is developing a system of virtual manufacturing system to simulate industrial processes. The project can be used by institutions of education or by directed for companies, with intention to transmit the real idea of a idealized context for an industrial system serving as base of professionals training and support for specialists in taking of decisions in the planning of the production.

The system counts on: virtual mockups, that simulate the real world using graphical animation based on virtual reality techniques, and an programming environment for developing the control logic.

The programming environment is the subject of this work. It describes a standardized *software*, for development of programs that use the defined by IEC61131-3. Totally integrated in the virtual mockups for simulation of manufacturing processes, it is responsible for the programming of control logics that command the movement of objects, sensors and actuators, in the represented ones, thus creating a virtual reality.

Destined, initially, to the development of control logics using Ladder language, was developed in this work a graphical tool of programming with edition resources, interpretation and simulation of the logic based on symbols of relays and function blocks. The programming tool developed offers a set of instructions that it allows to program the logic of functioning of a process or of a machine with the boolean functions (And, Or, Negation), timers, counters, comparators and arithmetical functions. This set is formed by some elements of programming, between them, the symbols of contact of relay: normal opened and normal closed, that when adequately combined in linkings series and parallel executes the boolean functions AND and OR respectively. Together to

the functions of each element of programming, managed by an algorithm of interpretation and management, the dynamic of screen and images (illustrative figures), that are necessary to compose the logical relation between objects, had been created to implement the command lines.

So that an element composes the diagram, is presented a generic image according to type of chosen component. This image can be moved in the edition space to determine the point of attachment of the component. It settled in the screen, characteristics datas are attributed to the component to allow its identification inside of the logic. The positioning in the screen specifies connection points establishing the linking between the elements, while the graphical symbologies define the behavior of the elements in the control logic.

Manipulation commands of the environment are completely graphical, activated with buttons or tree of options. Elements of programming and lines of command can be edited, a complete diagram or individual lines can be executed or be simulated and there is the possibility of integration with virtual mockups of an industrial process. The variables can be recognized of the mockup, loaded of the entailed simulator to this environment or declared by the user.

This environment is a simple *software*, easy of to be used and to apply, independent of equipment or supplier, and it has the advantage of to adapt itself to any machine, therefore it is an open *software* and to count on a flexible architecture. The logics implemented in it can be stored in archive as diagrams (complete program) or logic bit (independent lines of command) in a way chosen for the user.

It possess resources that the majority of *softwares* similar it does not present, as: simulator module of entrances and exits, including the possibility of individual simulation of each line of command; options of regulated industrial languages of programming; and the most important, it must keeps link of communication with virtual mockups.

Índice

Índice de Figuras

Índice de Tabelas

I INTRODUÇÃO

A crescente evolução da tecnologia incute nas empresas uma necessidade incondicional de se modernizarem movidas pela concorrência e competitividade que aquecem o mercado industrial. Cada vez mais preocupadas com a excelência no atendimento ao cliente, buscam sempre inovações e soluções práticas ideais para suas questões e requisitos. Esse cenário favorece o desenvolvimento de recursos mais produtivos e técnicas de trabalho mais eficientes e elaboradas, além de estimular o aprimoramento de profissionais qualificados.

A automação industrial é um fator marcante na atualização das empresas. Torna-se mais comum e inteligente ao passo que a alta tecnologia se evidencia no chão de fábrica. A participação do computador nesse quadro é essencial. Os recursos modernos de *software*, aliados ao aperfeiçoamento dos dispositivos mecânicos, são importantes responsáveis por esse crescimento e constituem uma poderosa ferramenta capaz de “dar vida” à manufatura. A programação em geral é fundamental para a realização da maioria dos procedimentos automáticos, como por exemplo: funcionamento de robôs; execução de ciclos de produção seqüenciais; inspeção de qualidade de peças. Mas além de participar ativamente na automação dos processos, a programação se destaca também na supervisão e controle da produção, na análise de projetos, na simulação de sistemas, no desenvolvimento de soluções lógicas para os processos de fabricação, na confecção de relatórios, em levantamentos estatísticos, na análise de desempenho, ou seja, exerce um importante papel desde a produção até a gestão empresarial.

Softwares atuam como grandes colaboradores no meio industrial, assim como em qualquer outra área. Sua tecnologia evolui rapidamente incentivada pelo exigente mercado que desfruta de seus recursos. Entre tantas contribuições que os *softwares* podem promover ao meio industrial, uma das mais modernas e importantes é a possibilidade de reproduzir virtualmente um ambiente fabril e criar uma realidade virtual, que retrate a réplica perfeita do modelo real. Seus recursos permitem que se reproduzam condições do meio circunscrito com as reações e interações dos elementos de campo, dando vida às ilusões do projetista, que pode avaliar o quão prático e funcional é seu projeto idealizado.

Concretizar, ainda que virtualmente, a abstração de um projeto, fornecendo recursos ao especialista para analisar os requisitos e avaliar o comportamento do sistema diante de uma aplicação é fundamental. Assim, é possível evitar problemas de estimativas, atender a requisitos, superar limitações, enfim, garantir o máximo de eficiência na conclusão de uma realização prática. A simulação de sistemas, ainda na fase de projeto, torna a realidade de campo muito próxima do ideal, o que contribui bastante para as tomadas de decisões na gestão empresarial. Além disso, simular projetos manipulando seus elementos e sua programação de controle, criando diversas aplicações, é muito interessante também para fins didáticos, importante para desenvolver a visão e o sentimento industrial dos futuros profissionais com foco em automação de processos ou produção. O mercado exige mão-de-obra qualificada, portanto é um ótimo instrumento para aperfeiçoar a formação dos especialistas.

Para desenvolver uma plataforma de simulação voltada a processos industriais, que seja capaz de criar

uma realidade virtual [5], que represente com bastante proximidade cada objeto do mundo real, é necessário muito estudo, pesquisa, dedicação e ferramentas de *software* com recursos avançados, pois a intenção é representar com fidelidade cada elemento da modelagem, seu comportamento e resposta aos estímulos do meio. Como em aplicações de medicina [4], onde essa sofisticação há muito tempo vem sendo usada, capaz de criar uma representação minuciosamente do foco real no computador, para orientação e auxílio dos médicos em exames, intervenções cirúrgicas e outras tarefas delicadas.

Está sendo desenvolvido pelo Centro de Referência em Tecnologia da Informação (CRTi), na Universidade Federal de Itajubá (UNIFEI), com assistência do Centro Nacional de Pesquisas (CNPq), um sistema de manufatura virtual composto principalmente por: maquetes virtuais, dispostas num ambiente gráfico para representação e animação [29]; e um ambiente de programação, responsável pelo controle das maquetes [30].

O tema dessa dissertação aborda somente parte do sistema. Concentra-se no ambiente de programação, em uma de suas linguagens, e na integração entre a lógica de controle e a maquete.

Objetivo

O objetivo desta dissertação consiste no desenvolvimento de um ambiente de programação da lógica de controle de maquetes virtuais com base nas diretivas da Norma IEC61131-3 [1], que trata da padronização de linguagens de programação de controladores industriais. O ambiente deve permitir a programação de Controladores Lógicos Programáveis (CLP) e robôs, porém, em primeira instância, será disponível apenas uma linguagem gráfica, conhecida como Diagrama Ladder (DL), para programação de CLP. Essa linguagem é bastante conhecida e muito usada devido à semelhança com os diagramas de relés eletromecânicos de painéis elétricos utilizados para controle e intertravamento de processos industriais. A Figura 1 ilustra a representação de uma mesma lógica usando relés e seu similar em DL.

(a)Diagrama de relés eletromecânicos[23] (b)Diagrama Ladder

Figura 1 – Comparação entre o diagrama de relés e o Diagrama Ladder.

A intenção é apresentar ao usuário recursos necessários para uma boa programação e um ambiente amigável que torne simples e fácil a manipulação dos componentes na tela.

O ambiente de programação deve trocar informações com a maquete virtual de modo a ilustrar cada elemento com sua respectiva animação em resposta às lógicas do DL. Para comprovar a funcionalidade do ambiente, foi criada uma maquete virtual que modela uma mesa de processamento de peças e desenvolvido um DL para uma aplicação simples dessa mesa.

A idéia do projeto é possibilitar a simulação de um sistema antes de sua implementação em campo, o que facilita visualizar a aplicação e auxilia nas avaliações.

I.1 O Estado da Arte

A modelagem virtual de situações reais causou um avanço significativo na visualização de sistemas físicos complexos, devido à considerável aproximação da realidade na representação das características dinâmicas dos objetos [17]. Os modelos criados no computador permitiram supervisionar sistemas complicados e delicados de maneira mais eficaz, além estabelecerem um novo conceito de intervenção do usuário usando as tecnologias relacionadas à internet, que possibilitam o acesso e o controle remotos do sistema.

Programas de simulação em três dimensões (3D) são capazes de criar um mundo virtual com o máximo de realismo alcançado pelas tecnologias presentes no mercado atual de *software*. Muitos prevêem expansões de sistemas e dão liberdade de criação ao usuário que pode explorar seus conhecimentos e originalidade ao elaborar aplicações nesse tipo de ambiente [12].

Dispondo de recursos sofisticados de programação, alguns ambientes virtuais são aptos a reproduzir movimentos humanos e reconhecer comandos de voz [14]. Também máquinas têm seu comportamento minuciosamente retratado no mundo digital, permitindo o estudo mais aprofundado de seu comportamento [13]. E a possibilidade de se trabalhar virtualmente com modelos complexos, permite tratar sistemas completos de manufatura flexível [17] com maior confiabilidade e segurança [16].

Manipular um sistema de manufatura virtual é muito mais que simplesmente simular um apanhado de equipamentos interagindo entre si. Envolve um conceito maior em torno do projeto que compreende administrar seus fenômenos e características de acordo com sua disposição, tarefas correlacionadas, tempos de ciclo, comportamento dos agentes (sensores e atuadores), estratégias de processamento de produto, controle e eventos [16], dentre tantos outros itens relacionados aos processos de produção. Por isso o sistema de manufatura virtual tem que ser o mais completo possível, para amarrar todas as condições de avaliação de um projeto.

O mercado disponibiliza uma grande diversidade de *softwares* capazes de reproduzir digitalmente objetos industriais, porém poucos alcançam um nível satisfatório de realidade. A maioria utiliza representação em duas dimensões (2D) e são dedicados a fabricantes e a equipemantos. No entanto, há ferramentas que permitem simulações em 3D e dispõem de recursos interessantes que auxiliam na avaliação de um projeto, ainda que tenham algumas limitações. São exemplos de sistemas virtuais voltados à manufatura:

- Flexman [15] – *software* 3D de modelagem e simulação para manufatura flexível e
- COSIMIR (Cell Oriented Simulation of Industrial Robots) [18 e 19] – *software* 2D de modelagem e simulação mais focado em robótica.

São simuladores parecidos com a proposta do sistema que está em desenvolvimento no CRTi. Estão inseridos no mesmo contexto (industrial) e apresentam uma realidade bastante fiel na modelagem do

comportamento dos objetos.

O Flexman é uma ferramenta em que o sistema é modelado e simulado criando uma maquete virtual, porém não dá condições de se desenvolver uma lógica de controle em qualquer linguagem industrial. Para configurar as diretivas de controle desse sistema e definir o arranjo de seqüências que caracterizam a dinâmica da operação, é usado um conjunto de regras e trajetórias baseadas no formato: SE <condição> então <ação>. Caso o comportamento do sistema não seja satisfatório, muda-se a política de regras e direcionamentos dos objetos da modelagem [16]. A aferição do projeto, para testar a veracidade de funcionamento do fluxograma usado, implica numa execução prática em bancada, independente do sistema de manufatura virtual, envolvendo *software* e equipamento de qualquer fabricante disponível no mercado, como: Siemens, Weg, Phoenix Contact, ou qualquer outro.

Voltado a aplicações com dispositivos robóticos, com alguns recursos pneumáticos [8], o Cosimir é uma ferramenta de modelagem e simulação de plantas industriais. Conta com recursos de zoom que possibilitam analisar detalhes dos objetos em ação que seriam impossíveis de se acompanhar na prática. Suas bibliotecas são restritas a alguns fornecedores e sua configuração não é trivial, implicando numa considerável dificuldade em usá-lo. O custo para aquisição da ferramenta é bastante alto [18], inviável para fins didáticos. E além de tudo, assim como o Flexman, também usa regras e trajetórias para compor as diretivas das operações, não dispondo de módulo programador de controle baseado em linguagens industriais padronizadas.

O ideal é que o sistema seja aberto, universal, independente de fabricante ou tecnologia, e apresente o máximo de recursos de edição e simulação, para que se tenha a idéia completa dos resultados a serem obtidos na implementação. E para que não seja preciso retrabalhar a lógica de controle por estar numa linguagem não enquadrada na norma, o ideal é ter imbutido no próprio sistema um ambiente de programação para elaborar a lógica de controle em linguagem industrial compatível com qualquer fabricante do mercado, ou seja, padronizada pela norma vigente que dita as regras para linguagens usadas em automação [1 e 21], pois assim a representação virtual seria completa e mais real.

A manufatura virtual do CRTi conta com essa vantagem. Num ambiente de programação confecciona-se o controle para comando das maquetes virtuais, respeitando a norma IEC61131-6 [1]. Isso permite o desenvolvimento da lógica de controle no mundo virtual e sua transferência para o mundo físico de maneira direta, sem necessidade de reestruturar o programa, desde que o *software* a ser empregado na prática siga a padronização.

O ambiente de programação incluído na manufatura virtual, comparado a *softwares* industriais dispostos nos mercado, tem como sua principal vantagem o vínculo com maquetes virtuais. Isso permite a visualização do comportamento dos objetos em reação aos estímulos provocados pela lógica de controle durante o desenvolvimento do projeto, possibilitando ajustes e alterações de procedimentos.

A maioria dos programas encontrados para implementação de programas de controle são dedicados a equipamentos, específicos de fabricantes, como: o PCWorx voltado aos controladores da Phoenix Contact [20] ou o Step 7 direcionado para produtos Siemens [22]. Poucos são os aplicativos genéricos desta categoria. O Isagraf é um exemplo de *software* independente de *hardware* baseado no padrão internacional [21 e 24]. Permite a programação e a simulação de entradas e saídas para teste da lógica de controle, porém peca por não ser integrado a um sistema de manufatura virtual.

Na literatura também foi encontrado um simulador de PLC [28] em que o autor se dedica a implementar um *software* para fins didáticos, provido de algumas plantas virtuais, gratuito e independende de fabricante. Ainda na fase de criação (bem primitiva), com uma proposta bastante básica, tem limitações de linguagens e de representação gráfica.

Além de algumas ferramentas de implementação de lógicas de controle, ao contrário do ambiente de programação assunto desse trabalho, não disponibilizarem todas as linguagens de programação previstas na IEC611311-3 [1], outra característica importante que inviabiliza o uso de tais *softwares* amarrados a *hardwares* e fabricantes é o altíssimo custo, pois se torna impraticável a instituições de ensino, a menos que se estabeleçam convênios e parcerias, facilitando a aquisição dos produtos, e é um investimento inicial muito sifgnificativo a empresas na fase de concepção e aprovação de um projeto. E além do preço do produto ser caro, ainda há restrição de licensa para liberar a livre utilização da ferramenta.

O ambiente de programação do sistema de manufatura virtual do CRTi se beneficia com a junção dos recursos oferecidos por essas ferramentas já atuantes no mercado além do aprimoramento e melhoria com vista nas limitações apontadas por elas. E ainda apresenta um custo irrisório além de ser independente de *hardware* e de fabricante, tornando-o viável à demanda do mercado industrial e acadêmico.

I.2 Organização do Trabalho

A organização desse trabalho reúne dez capítulos e cinco 5 apêndices, dispostos da seguinte maneira: no capítulo II encontra-se uma breve descrição do é uma manufatura virtual; os capítulos III e IV descrevem as maquetes virtuais e o ambiente de programação; o capítulo V enfoca o algoritmo de compilação e a tratativa dos elementos gráficos; o capítulo VI apresenta as telas e seus recursos; no capítulo VII é feita uma explanação com respeito à aplicabilidade do projeto; no capítulo VIII é demonstrados alguns resultados e a aprovação do aplicativo; o capítulo IX traz a conclusão do trabalho e a reflexão sobre possíveis trabalhos futuros; o capítulo X lista as referências bibliográficas que serviram de auxílio no desenvolvimento do projeto; e finalizando, seguem em anexo alguns materiais importantes para o entendimento desse trabalho.

II MANUFATURA VIRTUAL

Um sistema de manufatura virtual retrata a realidade de processos industriais, apresentando a visão clara do que será o projeto na prática, mostrando o seqüenciamento das atividades de processo e permitindo visualizar os pontos fracos da aplicação que permite evitar ocorrências de falhas causadas por fenômenos críticos tais como conflito e *deadlock*,

A manufatura virtual por estar imersa num contexto totalmente digital [7], portanto imaginário, oferece aos profissionais a liberdade de criar e simular soluções inovadoras, mais ousadas e criativas, sem o risco de perdas exorbitantes de material ou danos à instituição em função de uma possível idéia frustrada, pois, apesar da fleximbilidade, a ferramenta respeita as limitações reais que a prática deverá impor ao projeto se levado à implementação.

Um *software* como esse é bastante eficaz para instituições de ensino técnico por incutir nos alunos o sentimento prático das teorias de automação que lhes são aplicadas. Além disso, um simulador de processos de fabricação promove ganhos consideráveis às empresas, não só no treinamento de profissionais, tornado-os mais qualificados, mas na própria gestão empresarial diante das vantagens e facilidades que apresenta. A seguir são citados alguns dos principais benefícios da manufatura virtual:

- Abre o campo de visão na busca de soluções e aprimora a percepção do mundo real da automação;
- Possibilita o aprimoramento de técnicas e permite alcançar maior eficiência nos processos industriais, usando a diversividade de componentes e uma vasta gama de possibilidades em implementação;
- Permite estimar com mais precisão os tempos de ciclo de produção, rendimentos e perdas;
- Contribui na identificação do equipamento mais adequado para uma determinada aplicação, uma vez que o comportamento de cada elemento de processo é reproduzido com exatidão e possibilida saber como será sua resposta aos estímulos do meio e às condições a que será submetido;
- Transmite maior segurança na integração de equipamentos, uma vez que permite testar a configuração e reestruturar o projeto para eliminar pontos fracos e de risco;
- Ajuda na definição de um Layout mais confiável e simplificado, com o máximo de aproveitamento de área e espaço;
- Permite avaliar a condição prática de uma aplicação e eventualmente, após inúmeras tentativas de soluções, condenar um projeto que não será bem sucedido;
- Auxilia no estudo de disposições melhores e mais ergonômicas dos equipamentos e máquinas a serem operados manualmente, evitado problemas com Lesões por Esforços Repetitivos (LER) ou Distúrbios Osteomusculares Relacionados ao Trabalho (DORT) [11];
- E outros recursos que contribuem nas avaliações de projetos e agregam valores à formação de

especialistas.

A manufatura virtual que vem sendo desenvolvida na UNIFEI busca oferecer todas essas qualificações. Para isso mobilizam-se equipes de alunos com diversas formações e experiências, reunindo conhecimentos e técnicas avançados num projeto que tende a inovar a categoria dos *softwares* simuladores em nosso país.

O sistema está todo idealizado. É composto basicamente por duas partes principais:

- Maquete virtual[29]: representação e animação gráfica de objetos e
- Ambiente de programação[30]: desenvolvimento da lógica de controle da maquete.

O ambiente de programação prevê ainda outras subdivisões para tratar individualmente de cada linguagem de programação. Cada divisão significa um projeto a parte, que será integrado aos demais para compor ambiente completo.

A troca de informações entre a maquete virtual e o ambiente de programação ilustra na tela do computador cada elemento com sua respectiva animação, simulando o processo industrial, formando, portanto, a manufatura virtual.

III MAQUETE VIRTUAL

Maquetes virtuais são representações gráficas que ilustram a modelagem física e comportamental de objetos que interagem entre si num meio pré-definido.

No caso de maquetes de manufatura virtual, são reproduzidos ambientes industriais incluindo equipamentos, instrumentos e maquinários do meio fabril. Dispõem de alto nível de detalhamento e precisão, criando uma realidade virtual utilizando a mesma tecnologia de *software* aplicada a jogos. A Figura 2 mostra duas maquetes que representam virtualmente mundos reais.

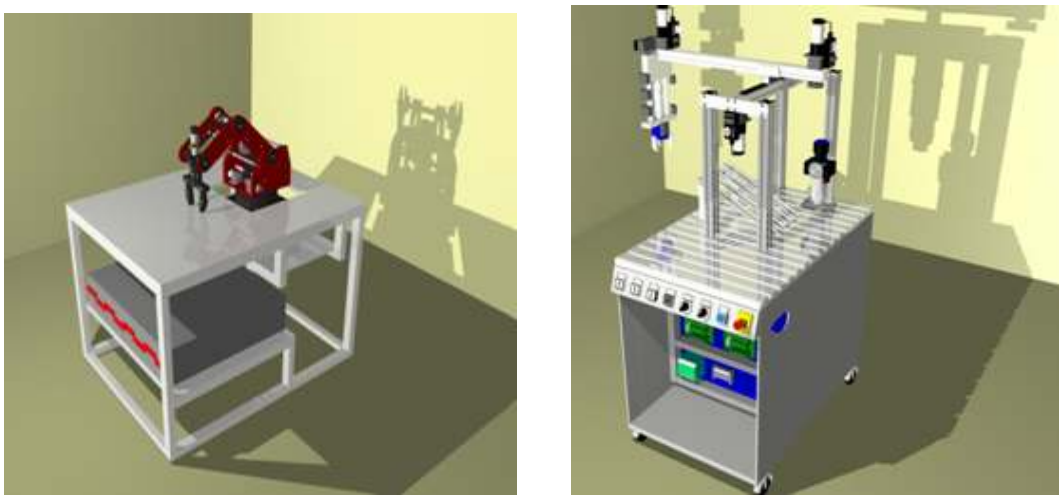


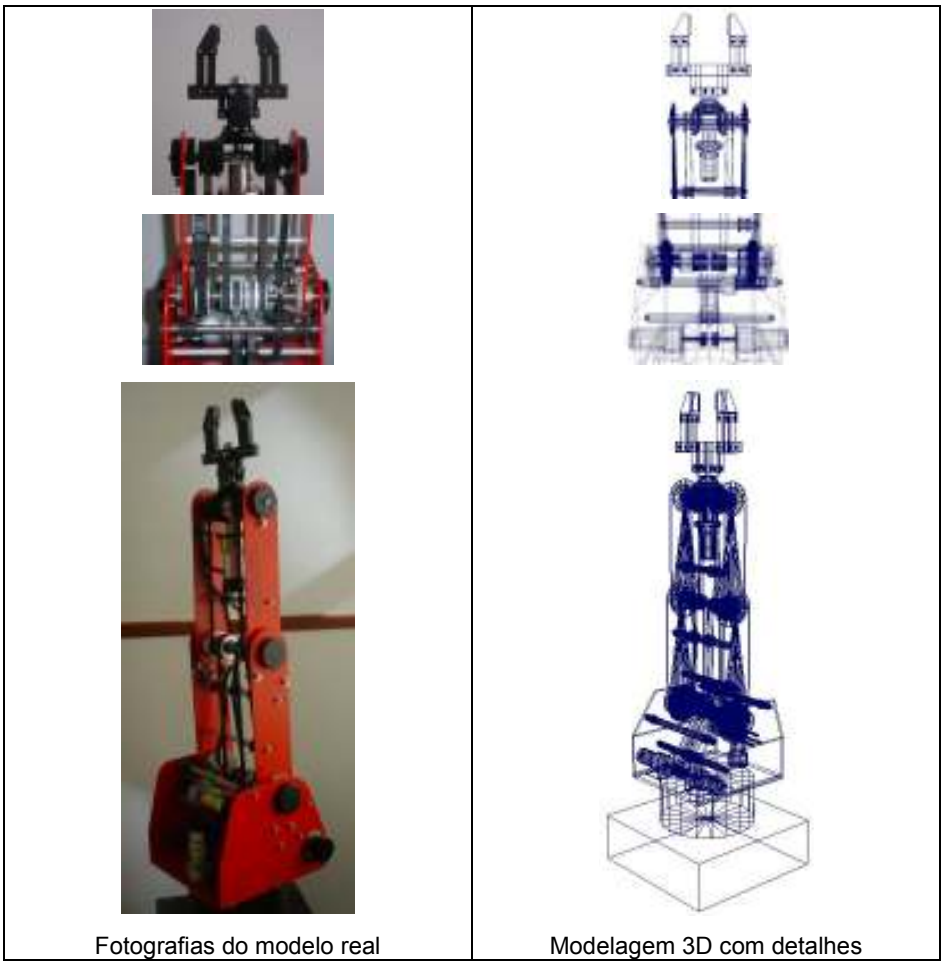
Figura 2 – Exemplos de maquete virtual.

A modelagem dos objetos garante uma precisão minuciosa. As imagens são captadas e reproduzidas com o cuidado de respeitar cada detalhe. O movimento, comportamento e resposta dos objetos são estudados e alimentam uma seqüência de atividades para construir um ambiente semelhante ao mundo real, onde esses objetos vão interagir. Dentre as tarefas envolvidas na criação de um objeto, a reprodução de agentes do meio que interferem em sua dinâmica são de total relevância, como a ação da gravidade e colisões.

A colisão é quem determina a interação entre os elementos da maquete, isto é, a proximidade, o choque, o apoio entre os objetos. Permite ilustrar a ação de um dispositivo robótico pegando algo ou mesmo reproduzir algum objeto sobre uma superfície. Já a representação da gravidade imita a força que age nos corpos atraindo-os para baixo, gerando as quedas dos objetos.

As maquetes virtuais são desenvolvidas numa ferramenta gráfica 3D após muito estudo do modelo real escolhido e diversos cálculos [6]. Como demonstração, pode ser citado o dispositivo robótico [3], cujo modelo real foi minuciosamente estudado, fotografado em detalhes e cuidadosamente cotado, como pode ser visto na Figura 3.

Tudo para se conseguir um alto grau de fidelidade na reprodução do equipamento. Também foi feita uma análise de otimização da *performance* do modelo virtual visando criar uma dinâmica bem próxima da realidade. A principal ferramenta utilizada no projeto dessa maquete foi o DirectX9.



Fotografias do modelo real | Modelagem 3D com detalhes

Figura 3 – Algumas representações do Braço Mecânico [3].

Ao comparar as imagens (a) e (b) da Figura 4, pode-se perceber o nível de realidade já alcançado na modelagem de um dispositivo robótico [3].

(a) Objeto Real (b) Modelo Virtual

Figura 4 – Braço Mecânico (SCORBOT-ER V) [3].

A literatura que trata mais detalhadamente do projeto das maquetes virtuais encontra-se na dissertação de mestrado defendida pelo Welinton Dias[29].

IV AMBIENTE DE PROGRAMAÇÃO E INTEGRAÇÃO PARA MANUFATURA VIRTUAL

O ambiente de programação é uma peça importante no corpo do sistema de manufatura virtual, pois vinculado às maquetes virtuais[30], que representam o modelo real, dará "vida" à dinâmica do processo no computador, controlando os objetos com os comandos resultantes da lógica nele implementada. Com esse recurso é possível testar as lógicas de controle que devem manipular o processo de fabricação antes de serem postas em prática.

O Ambiente de Programação e Integração para Manufatura Virtual (APIMV) é um ambiente independente de fabricante e equipamento. A programação nesse ambiente é bastante simples, e não implica em posteriores conversões de código para linguagens de mercado na aplicação real. Com a intenção de tornar o programa o mais aberto possível, seguindo o propósito de um ambiente universal de programação, tomou-se por base a Norma IEC61131-3 [1], que padroniza as linguagens de programação para automação industrial.

A interface com o usuário é bem simples e intuitiva, que garante a integração amigável com o programa e uma familiarização rápida.

O *software* dispõe de recursos de programação e monitoramento, e deve estabelecer comunicação com outros programas (maquetes virtuais) desenvolvidos em DirectX, responsáveis pela animação gráfica do sistema. O vínculo entre o *software* de programação e as maquetes se dará através de protocolo TCP/IP.

A leitura dos estados das entradas e saídas (*I/O*) é feita por transferência de tabelas gravadas em arquivos. Inicialmente, a maquete virtual passa ao APIMV uma relação de todos os pontos de *I/O*. Durante a execução do programa, esses pontos são lidos de uma tabela atualizada constantemente pela maquete e aplicados à lógica. Os resultados são atualizados do ambiente para a maquete em outra tabela. A varredura de comunicação está ilustrada na Figura 5.

Figura 5 – Varredura de Comunicação.

IV.1 Funcionamento do Ambiente de Programação

O APIMV dispõe de dois modos de operação:

- modo simulação e
- modo *link*.

No modo simulação não é feita a comunicação com a maquete virtual, e nesse caso o número de pontos de entrada e de saída é restrito a: oito entradas digitais e duas analógicas, quatro saídas digitais e duas analógicas. No modo *link* é estabelecida a comunicação com a maquete, e nesse caso é ela quem estipula os pontos do processo a serem manipulados.

Ao criar ou editar um programa de controle, são listados os pontos de entradas e saídas disponibilizados pela maquete ou carregados pelo módulo simulador, e as variáveis internas definidas pelo programador para escolha da variável a ser associada a cada elemento de programação.

Para melhor auxiliar o trabalho do desenvolvedor, o programa pode ser testado a qualquer momento durante sua execução, e as linhas de comando testadas separadamente no momento em que estão sendo editadas. Há um modo de execução tanto para a área do diagrama quanto para a área de edição.

O mesmo ocorre com o armazenamento e resgate de um programa ou partes de uma implementação. A gravação e a restauração de códigos podem ser feitas na tela do diagrama, salvando/abrindo lógicas completas, ou na tela de edição, salvando/abrindo apenas uma linha de comando (*rung*). Com isso, o programador não precisa reeditar linhas inteiras que tenham lógicas iguais ou semelhantes, agilizando o trabalho “braçal”, reduzindo o tempo de codificação e liberando tempo para elaboração da lógica.

Toda a manipulação de arquivos feita pelo ambiente de programação se refere por convenção ao seu diretório raiz, salvo especificação de caminho para a comunicação com a maquete virtual ou indicação de diretório para armazenar/resgatar um código. Os códigos podem ficar alocados em qualquer lugar a que o computador onde ele esteja sendo implementado tenha acesso, desde que especificado.

IV.2 Comunicação com a Maquete Virtual

Quando o APIMV é executado no modo *link* de operação, seu vínculo com a maquete deve ser estabelecido através de um driver de protocolo TCP/IP incorporado à ferramenta de desenvolvimento. Esse recurso permite o acesso direto a um endereço local ou remoto.

A comunicação entre o APIMV e a maquete é simples. Ao iniciar a implementação de um programa, o usuário determina a localização da maquete através do nome da máquina ou do número de IP. Caso não informe o caminho, o programa assume a definição padrão, que aponta para a máquina local. O *link* é transparente ao usuário.

O sistema utiliza uma arquitetura cliente-servidor, de modo que a maquete exerce o papel de servidor e o ambiente de programação trabalha como cliente; o acesso é feito pela porta 8082 da respectiva máquina. A Figura 6, mostra uma ilustração da arquitetura empregada.

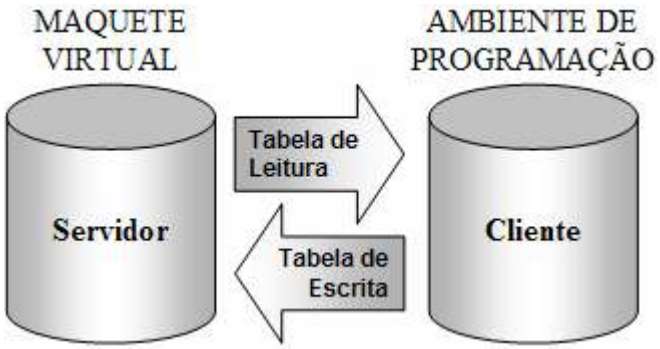


Figura 6 – Ilustração da arquitetura do sistema.

As informações trocadas entre o APIMV e a maquete ficam dispostas em duas tabelas: tabela de leitura e tabela de escrita. No caso de simulação de *I/O*, em que não se usa o *link* com a maquete virtual, as informações são disponibilizadas em arquivos do tipo texto: TI.txt, espelho da tabela de leitura e TO.txt, espelho da tabela de escrita.

A primeira tabela (leitura) é composta por seis campos que representam cada ponto de transferência de dados:

- Entrada/Saída (*I/O*): classificação do ponto como entrada (I) ou saída (O);
- Nome: nomenclatura de referência ao ponto de *I/O*;
- Tipo de dado: identificação do tipo de informação que será transportada;
- Comentário: breve descrição do ponto;
- Valor Inicial: valor a ser carregado no elemento relacionado ao ponto em questão no início da execução do programa;
- Valor Final: referência final para limitar a variação do valor desse ponto.

A segunda tabela (escrita) é criada durante a execução do programa de controle para transmitir à maquete ou ao simulador de *I/O* os valores de atualização dos pontos de saída (já definidos pela maquete na primeira

tabela); portanto possui apenas um campo de informação.

O formato dos pacotes de transmissão (leitura e escrita) pode ser conferido na Tabela 1:

Tabela 1 – Formato do pacote de transferência de informações.

Tabela de Leitura (direção: Maquete → APIMV)						
campos	**1**	**2**	**3**	**4**	**5**	**6**
conteúdo	Entrada/Saída (*I/O*)	Nome	Tipo de Dado	Comentário	Valor Inicial	Valor Final
Tabela de Saída (direção: APIMV → Maquete)						
campos	**Único**					
conteúdo	Valor Atual					

Para exemplificar a formatação das tabelas de transferência de dados, foi montado o seguinte modelo (Tabela 2):

Tabela 2 – Exemplo de pacote de comunicação entre maquete e APIMV.

Tabela de Leitura (direção: Maquete → APIMV)					
I/O	***Nome***	***Tipo de Dado***	***Comentário***	***Valor Inicial***	***Valor Final***
I	DI1	BOOL	Entrada Digital 1	false	false
I	DI2	BOOL	Entrada Digital 2	false	false
I	DI3	BOOL	Entrada Digital 3	false	false
I	AI1	STRING	Entrada Analógica 1	false	false
O	DO1	BOOL	Saída Digital 1	false	false
O	DO2	BOOL	Saída Digital 2	false	false
O	DO3	BOOL	Saída Digital 3	false	false
O	AO1	STRING	Saída Analógica 1	false	false
Tabela de Saída (direção: APIMV → Maquete)					
Valores das Saídas					
false false false false					

As informações fornecidas pela maquete são automaticamente reconhecidas pelo ambiente de programação e os pontos de entrada e saída do processo ficam disponíveis para a confecção da lógica, onde são relacionados a elementos lógicos do Diagrama Ladder.

IV.3 Desenvolvimento da Lógica Ladder

A Lógica Ladder consiste numa coleção de componentes gráficos interligados que determinam o fluxo de energia ao longo da malha de conexões estabelecida pelo programador. Há cinco elementos básicos disponíveis para programação no APIMV. São instruções denominadas:

- Contatos
- Blocos de Funções
- Bobinas
- Conexão Horizontal
- Conexão Vertical

Instruções do tipo contato representam os estados atuais das variáveis de programa e se encarregam de liberar ou bloquear a alimentação.

Os blocos de função têm uma configuração específica para cada um dos diferentes tipos, dependem de condições pré-estabelecidas para habilitá-lo e quando acionam sua saída, comportam-se de forma semelhante aos contatos, liberando ou impedindo a passagem de energia.

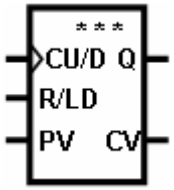
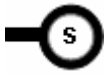
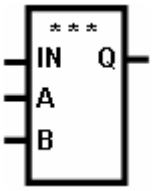
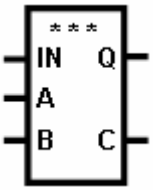
Os fios (conexões horizontal e vertical), sempre liberam o fluxo, funcionam como conectores. A conexão horizontal preenche lacunas entre os elementos na linha ou leva uma ligação até a linha mestre (fonte de alimentação) e a conexão vertical estabelece ligações paralelas entre elementos e conduz a energia no sentido vertical.

Por sua vez, as instruções do tipo bobina são energizadas, ou não, conforme alimentação (energia) que recebem.

A Tabela 3 mostra a representação dos principais elementos gráficos presentes no ambiente de programação.

Cada elemento de programação representa uma variável que pode ser local ou externa (entrada ou saída) e tem um comportamento específico diante do estado de ativação dela. É preciso conhecer esse relacionamento e entender como a dinâmica entre os elementos de programação funciona para compor uma lógica Ladder.

Tabela 3 – Identificação dos elementos de programação

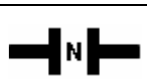
<table>
<tr><th colspan="2">CONTATOS</th><th colspan="2">BOBINAS</th><th colspan="2">BLOCOS</th></tr>
<tr><td> 
Normalmente Aberto (NA)</td><td> 
Normalmente Fechado (NF)</td><td> 
Direta</td><td> 
Inversa</td><td rowspan="2"> 
Temporizador</td><td rowspan="2"> 
Contador</td></tr>
<tr><td> 
Pulso no edge positivo[i]</td><td> 
Pulso no edge negativo[i]</td><td> 
Pulso no edge positivo[i]</td><td> 
Pulso no edge negativo[i]</td></tr>
<tr><td colspan="2">FIOS DE CONEXÃO</td><td rowspan="2"> 
Set</td><td rowspan="2"> 
Reset</td><td rowspan="2">  </td><td rowspan="2">  </td></tr>
<tr><td> 
Fio Horizontal</td><td> 
Fio Vertical</td><td colspan="2">As bobinas Set e Reset são associadas à mesma variável</td><td>Comparador</td><td>Aritmético</td></tr>
<tr><td colspan="6">Nota:
i. Edge positivo: capta a transição de estado de false para true;
ii. Edge negativo: capta a transição de estado de true para false;</td></tr>
</table>

A seguir, da Tabela 4 à Tabela 6, é ilustrada a relação dos elementos com o estado de ativação das variáveis associadas, e seu comportamento pode ser conferido no item V.2. De acordo com as ligações entre os elementos e a resposta de cada qual frente aos estímulos e estados variáveis do sistema dentro da lógica estabelecida, os pontos de saída são atualizados.

Nas tabelas 4 a 6, o campo Variável representa o ponto físico ou interno associado àquele elemento, é quem se encarrega pela habilitação do componente no caso de contato, ou é acionado se estiver ligado a uma bobina. A Alimentação indica a presença ou não de energia, ou seja, se o elemento está ou não habilitado (cabe lembrar que a condição verdadeira é representada por *TRUE* e seu inverso por *FALSE*). O Estado de ativação da variável indica em que situação o ponto físico ou interno representado por ele se encontra: ativado ou desativado. Dependendo dessa afirmação, o elemento responde com um estado de acionamento: ativado ou desativado, conforme peculiaridades de seu comportamento característico, liberando ou não o fluxo de energia.

Alguns elementos são sensíveis a transições de estado (*edge*), ou seja, operam mediante a passagem de um estado para outro. Outros respondem com um pulso na saída em lugar de liberar o fluxo continuamente. Isso vai depender do tipo de elemento.

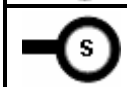
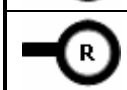
Tabela 4 – Características funcionais dos contatos

Símbolo	Variável	Alimentação[i]	Estado de ativação da variável	Estado do elemento	fluxo de energia	Descrição do Comportamento
	Digital	*true*[ii]	*true*	***true***	***liberado***	Acompanha o estado lógico da variável
			false[ii]	***false***	***bloqueado***	
	Digital	*true*	*true*	***false***	***bloqueado***	Inverso do estado lógico da variável
			false	***true***	***liberado***	
	Digital	*true*	*edge* positivo[iii]	***pulso***	***liberação temporária***	Acompanha o estado lógico da variável
	Digital	*true*	*edge* negativo[iv]	***pulso***	***liberação temporária***	Inverso do estado lógico da variável

Nota:
i. Alimentação: energia chegando até o elemento;
ii. *True*: significa ativado (*ON*) e *False*: significa desativado (*OFF*);
iii. *Edge*: positivo - significa transição de estado de *false* para *true* e negativo - significa transição de estado de *true* para *false*;
iv. *Pulso*: variação de estado, permanece um tempo curto em *true* e depois retorna para *false*.

A lógica de funcionamento de um contato pode ser entendida fazendo uma analogia com o funcionamento de relés eletromecânicos. Considerando o contato normalmente aberto da primeira linha da Tabela 4, observe que se o estado de ativação da variável for *true* (análogo a energização da bobina do relé eletromecânico) o estado do elemento também é *true* (análogo ao fechamento do contato normal aberto do relé eletromecânico). Nesta condição o fluxo de energia no diagrama Ladder está liberado; o que corresponde à passagem de corrente elétrica no circuito a relé.

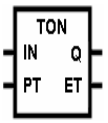
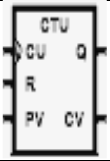
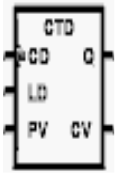
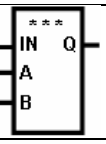
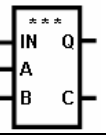
Tabela 5 – Características funcionais das bobinas

Símbolo	VAR	Alim.[i]	Estado	Comportamento	Símbolo	VAR	Alim.	Estado	Comportamento
	Digital	*true*[ii]	***true***	Acompanha o estado lógico da alimentação		Digital	*true*	***false***	Inverso do estado lógico da alimentação
	Digital	*edge*[iii] P	***pulso***[iv]			Digital	*edge*[iii] N	***pulso***	
	Digital	*true*	***true***	Acompanha o estado lógico da alimentação; Estado da variável permanece setado (*true*) até acionamento da bobina R					
		*true*	***false***[ii]	Inverso do estado lógico da alimentação; Estado da variável permanece resetado (*false*) até acionamento da bobina S					

Nota:
i. Alim. = alimentação: energia chegando até o elemento;
ii. *True*: significa ativado (*ON*) e *False*: significa desativado (*OFF*);
iii. *Edge* P: significa transição de estado de *false* para *true* e *Edge* N: significa transição de estado de *true* para *false*;
iv. Pulso: variação de estado, permanece um tempo curto em *true* e depois retorna para *false*.

No caso de uma bobina, sua lógica de funcionamento é semelhante à de uma solenóide. Considerando a bobina simples da primeira linha e primeira coluna da Tabela 5, observe que se o estado de ativação da alimentação for *true* (análogo à energização da bobina da solenóide) o estado do elemento também é *true* (análogo ao acionamento da solenóide).

Tabela 6 - Características funcionais dos blocos de função

Símbolo	Variável (valor)	Acionamento (entradas digitais)	Acionamento Estado	Operação	Q (saída digital)	Descrição do Comportamento
	PT Tempo máximo	IN	*edge* positivo	dispara	*true*	Enquanto habilitado mantém Q ativada. Quando ET = PT, zera contagem e desativa Q
	ET Contagem		*true*	contando		
			false	zerado	*false*	
	PT Tempo máximo	IN	*true*	zerado	*false*	Quando é retirada a habilitação, aciona Q, mantém até ET = PT e zera desativando Q.
	ET Contagem		*edge* negativo	dispara	*true*	
			false	contando		
	PV Valor máximo	CU	*edge* positivo	incrementa	*true* quando CV = PV	Incrementa contagem a cada *edge* positivo em CU. Ativa Q quando CV atinge PV e o desativa quando R é habilitado.
	CV Contagem	R	*true*	zera		
	PV Valor inicial	CD	*edge* positivo	decrementa	*true* quando CV = 0	Incrementa contagem a cada *edge* positivo em CU. Ativa Q quando CV atinge 0 e o desativa quando R é habilitado.
	CV Contagem	LD	*true*	recarrega		
	A	IN	*true*	A *** B	*true*	Quando há energia, compara A e B *** é substituído por: =, <>, >, <, >= ou <=
	B		*false*	nada	*false*	
	A	IN	*true*	A *** B	*true*	Quando há energia, compara A e B *** é substituído por: +, - , * ou /
	B		*false*	nada	*false*	

Nota:
i. *True*: significa ativado (*ON*), *False*: significa desativado (*OFF*);
ii. *Edge* positivo: significa transição de estado de *false* para *true*, *Edge* negativo: significa transição de estado de *true* para *false*;

Já o bloco de função tem sua lógica de funcionamento dependente de uma condição especial conforme o tipo de operação que o define e a categoria a que pertence. Considerando o bloco TON (primeiro da Tabela 6), observe que seu comportamento depende de duas condições para estabelecer continuidade lógica: a alimentação na entrada IN (coluna entradas digitais da Tabela 6,) e o valor máximo para contagem de tempo indicado na entrada PT (coluna Variável da Tabela 6). A presença de alimentação na entrada IN, habilita um contador interno do temporizador que incrementa o tempo (em milissegundos) até o limite estabelecido pelo valor em

PT. No final da contagem, o contador pára e o temporizador libera o fluxo de energia; tal como um contato que se mantém aberto e depois se fecha (como a energização da bobina de um relé eletromecânico), permitindo a continuidade lógica da linha de comando.

IV.3.1 Posicionamento e fixação

Os elementos de programação da linguagem Ladder são objetos gráficos que se resumem num conjunto de parâmetros de configuração alocados numa lista interna do ambiente e que apresentam uma imagem associada na tela na posição determinada pelo usuário ao confeccionar o diagrama de contatos.

Para cada elemento gráfico inserido no diagrama é definido um conjunto padrão de parâmetros. Alguns determinados pelo usuário, como o ponto de conexão a ser vinculado a este elemento e sua posição no diagrama. Outros atributos já são especificados automaticamente pelo ambiente de programação. Essas informações ficam armazenadas numa lista tipo Array, disponível ao programa para consulta dos componentes na interpretação da lógica. Informam as características de cada elemento, identificando o tipo de componente, seu posicionamento na tela, seus pontos de conexão no diagrama, localização nas tabelas de dados, seu comportamento, além de outras informações importantes para sua classificação, interpretação e integração à lógica onde for inserido. A listagem desses parâmetros está disposta na Tabela 22 no APÊNDICE A.

A posição na tela (x e y) é definida dentro do espaço reservado para a edição. Diante desse posicionamento, é calculada a posição do elemento dentro do diagrama, determinando a linha e a coluna de sua localização no Diagrama Ladder. Duas referências são usadas para indicar os pontos de conexão do elemento. A primeira indica o local de fixação dentro do diagrama, tratado como ponto de entrada (Pin), e a segunda é a próxima posição à direita, tratada como ponto de saída (Pout), isso porque a energia chega pela esquerda do elemento gráfico e ao ser liberada passa à direita do mesmo estabelecendo a continuidade lógica.

Os pontos de referência de um componente do tipo contato estão representados a seguir na Figura 7. Os números 150 e 40 formam a coordenada de tela onde é fixado o componente, isto significa posicionamento na coluna 150 e na linha 40 do espaço de edição, ao passo que os números 12 e 13 compõem a identificação dos pontos de entrada e saída do contato, que determinam sua ligação a outros elementos. Essa coordenada refere-se à disposição do Diagrama Ladder indicando a linha que o elemento ocupa e as colunas que o delimitam, assim as conexões mostradas na Figura 7 informam que ele encontra-se na primeira linha do diagrama, pois Pin e Pout iniciam com o algarismo 1, e entre as colunas 2 e 3, pois Pin termina com 2 e Pout é finalizado com 3, ou seja, Pin=12 e Pout=13 localiza o contato na linha 1 entre as colunas 2 e 3.

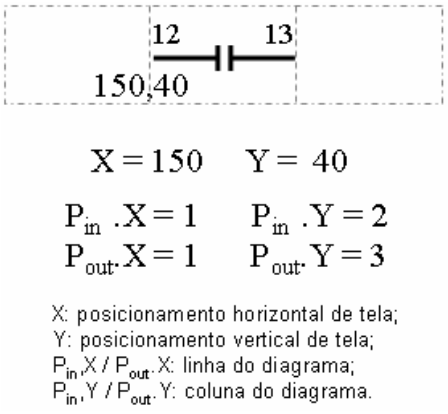


Figura 7 – Exemplo de Localização de um Contato

IV.3.2 Conexão dos elementos

A tela de edição é abordada como uma grade de células de tamanho determinado. A fixação e conexão entre os elementos são feitas pela marcação dos pontos de entrada e de saída (Pin e Pout respectivamente). Segue Figura 8 com ilustração.

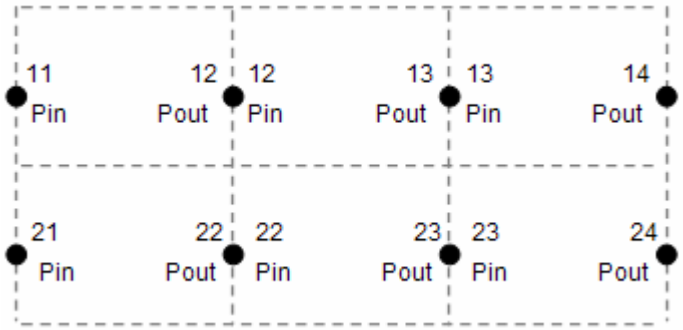
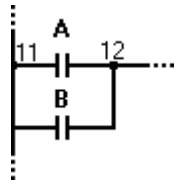
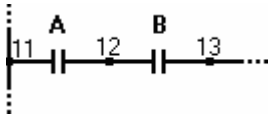


Figura 8 - Pontos de Conexão

Os elementos são considerados conectados entre si quando o ponto de entrada de um é igual ao de saída do outro, ou existem ligações entre eles estabelecidas por fios verticais ligando seus pontos de entrada e/ou de saída. Há dois tipos básicos de conexão entre elementos:

- Paralela: que estabelece a lógica OU (OR), onde fios verticais interligam os pontos de conexão de entrada (Pin) e/ou de saída (Pout) dos elementos (veja item a da Tabela 7).
- Série: que estabelece a lógica E (AND), onde o Pout do primeiro é igual ao Pin do próxim (veja item b da Tabela 7).

Tabela 7 - Exemplo de resolução de lógica

 $Pin_A = Pin_B = 11$ $Pout_A = Pout_B = 12$ A está em paralelo com B *Lógica OR*	 $Pin_A = 11$ $Pout_A = Pin_B = 12$ $Pout_B = 13$ A está em série com B *Lógica AND*
(a)	(b)

IV.3.3 Edição dos elementos

A área de edição é demarcada por células de tamanho definido, de modo que os elementos são fixados em posições conhecidas para que seja possível determinar sua ligação a outros componentes. Cada célula tem por definição 50 pontos de largura e 40 pontos de altura, partindo a contagem da posição 100,40 (x,y) da tela, reservando um espaço para cabeçalhos e procedimentos intermediários de implementação (assunto abordado no item VI.4). A Figura 9 mostra algumas posições de fixação:

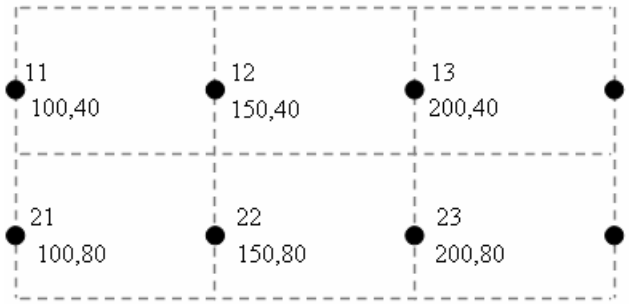


Figura 9 – Exemplos de posições de fixação na tela

Na edição de um elemento, o objeto que o representa (imagem genérica) deve ser arrastado pelo mouse até a posição aproximada de sua fixação. Ao soltar o objeto, o ambiente de programação se encarrega de calcular automaticamente a posição correta da célula onde fora solto e ajusta sua fixação. Por exemplo: se um objeto é solto no meio da segunda célula, em 170,60 (x,y), sua posição é ajustada para 150,40. A Figura 10 mostra uma ilustração do posicionamento de um componente.

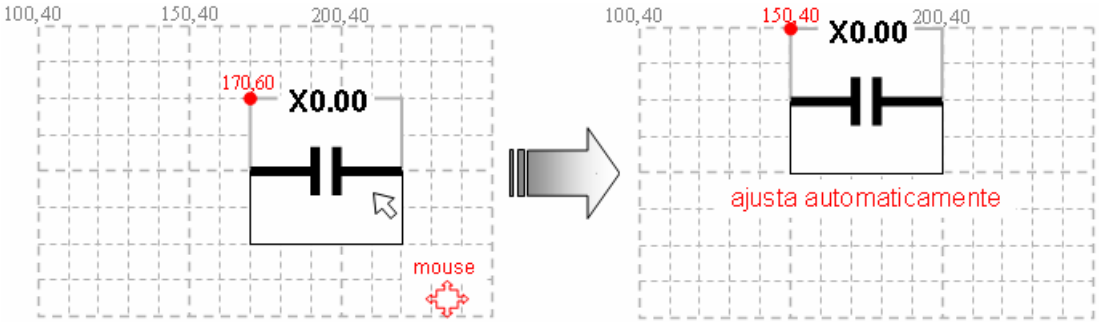


Figura 10 – Exemplo de fixação de componente na tela

IV.4 Gravação de programa

O arquivo de armazenamento de qualquer programa desenvolvido no APIMV é organizado da seguinte maneira (Tabela 8):

Tabela 8 – Formato dos arquivos de armazenamento do diagrama (.LAD ou .RNG)

Atributos	Descrição	Tipos	Objetos
ListaE.count	quantidade de elementos da lista	int	Número de Elementos
label	nome	string	Elemento1
categoria	grupo de classificação		
tipo	Tipo		
atr	Atributo		
td	Tipo de dado		
vi	Valor inicial		
texto	Comentário		
Figura	Imagem do elemento		
id	index	int	
x	coordenada horizontal		
y	coordenada vertical		
pin.X	ponto de conexão à esquerda na horizontal	point	
pin.Y	ponto de conexão na vertical		
pout.X	ponto de conexão na horizontal		
pout.Y	ponto de conexão na vertical		
gb.Name	nome da representação gráfica	groupbox	
gb.Text	texto da representação gráfica		
gb.Width	largura da representação gráfica		
gb.Height	altura da representação gráfica		
gb.Left	posicionamento horizontal da representação gráfica		
gb.Top	posicionamento vertical da representação gráfica		
...	repete o conjunto de atributos para os próximos elementos	…	Elementos2 ... ElementoN
ListaT.Count	quantidade de textos agregados ao programa	int	Número de Textos
tb.Name	nome de referência do texto	string	Texto1
tb.Text	Caracteres que compõem o texto		
tb.Width	largura do texto	int	
tb.Height	altura do texto		
tb.Left	posicionamento horizontal do texto		
tb.Top	posicionamento vertical do texto		
...	repete o conjunto de atributos para os próximos textos	…	Texto 2 ... TextoM

Esse formato é respeitado mesmo na ausência de alguns parâmetros, quando se preenche com espaços as lacunas de que ficaram nas posições de informações irrelevantes para um determinado elemento. Quando o programa é restaurado, essa estrutura servirá de orientação para ambiente de programação interpretar e dispor os dados em seus devidos lugares.

Os arquivos gravados no espaço de edição do ambiente, onde são criadas as linhas de instruções, apresentam extensão .RNG, indicando rung. Guardam apenas uma linha de comando. Já os arquivos que

armazenam diagramas completos, ou seja, uma ou mais linhas de comando, têm extensão .LAD, que o identifica como programa desenvolvido em linguagem Ladder.

IV.5 Fluxo de Energia e Processamento

O fluxo de energia é conduzido pelo elemento de programação da esquerda para a direita e no sentido vertical por toda a extensão da linha, como orienta a Figura 11.

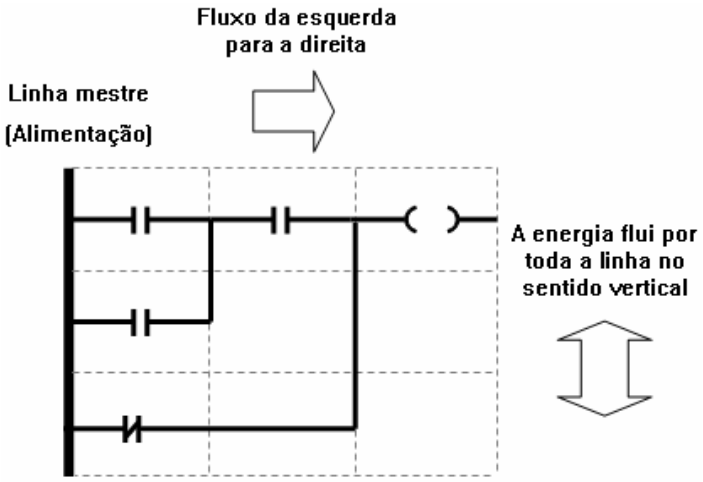


Figura 11 – Orientação do fluxo de energia no Diagrama Ladder

Graficamente, o estado de habilitação de um elemento é representado pela mudança de cor de branco para azul como apresenta a Figura 12. Uma vez habilitado, o elemento libera a passagem de energia conforme suas propriedades, ou seja, de acordo com seu tipo e com a categoria a que pertence. Assim, o estado de ativação da variável associada não é o único fator que comanda a troca de cor do elemento.

Na Figura 12 uma mesma variável (DI1) é associada a dois tipos de contato: normalmente aberto (NA) e normalmente fechado (NF), para mostrar o comportamento de cada um deles diante da troca de estado da variável. Nesse exemplo, quando DI1 assume estado de ativação igual a *true*, o contato NA acompanha o acionamento e sua cor passa a ser azul, enquanto o contato NF tem efeito contrário, já tendo iniciado com a cor azul simbolizando atuação inversa à mudança de estado da variável. Quando DI1 volta ao estado inicial (*false*), o contato NA acompanha a variável e volta à sua cor branca e o NF retorna à cor azul.

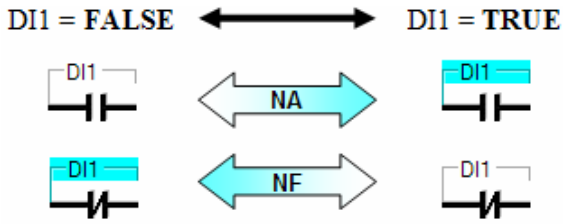


Figura 12 – Exemplo de representação gráfica de mudança de estado

O processamento da lógica varre linha a linha de comando (rungs), seguindo de cima para baixo e da esquerda para a direita de acordo com a ilustração da Figura 13.

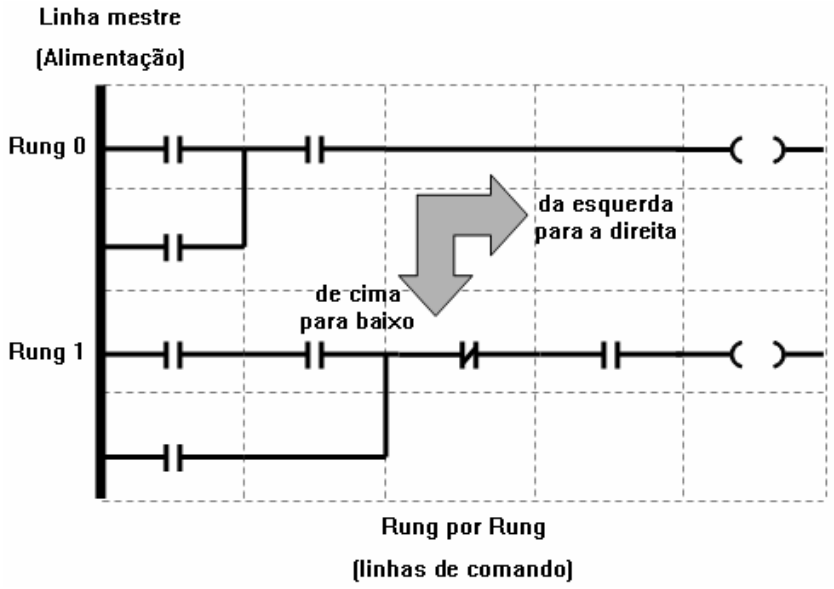
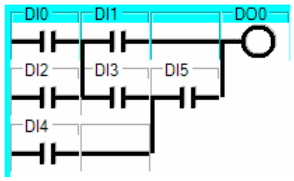


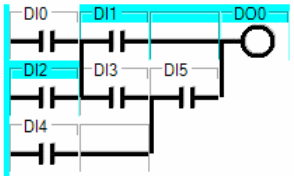
Figura 13 – Sentido e direção do processamento da lógica

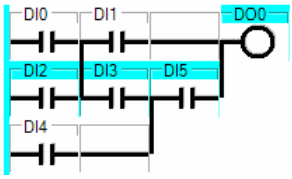
Não é permitido fluxo reverso, isto é, a energia não pode fluir por um caminho estabelecido por elementos anteriores à coluna que está sendo tratada naquele instante do processamento, segue sempre em frente e para baixo, não volta. O APIMV ignora caminhos de retorno, mesmo que no DL haja caminho para energia estabelecer continuidade lógica conduzindo o fluxo inversamente. A Tabela 9 ilustra esta explicação através de um DL que poderia permitir um eventual fluxo contrário. São apresentadas as possíveis alternativas de se estabelecer a continuidade lógica do programa (de i a iv) e uma situação em que poderia ocorrer fluxo reverso através de um contato anterior ao ponto de varredura atual. Na ocasião em que o DI3 oferece uma possível passagem para a energia, a varredura, ao atingi-lo, não encontra alimentação do lado esquerdo do contato, por isso não há fluxo de

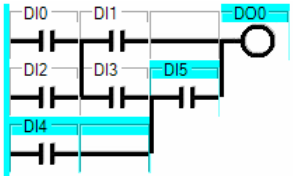
energia através dele. Quando a varredura passa à frente desse contato, há alimentação chegando até o fio vertical habilitando o contato associado a DI5 que encontra-se em estado de ativação *false*, portanto não há fluxo de energia nesse momento também.

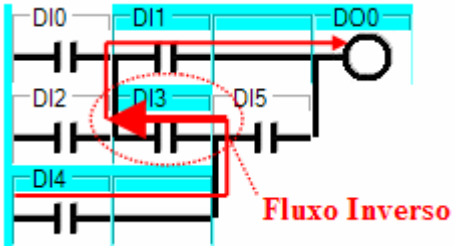
Tabela 9 – Exemplo de Diagrama Ladder com possibilidade de fluxo reverso

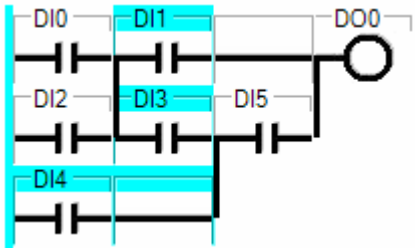
<table>
<tr><th colspan="4">Caminhos possíveis para o fluxo de energia</th></tr>
<tr>
<td> 

(i)</td>
<td> 

(ii)</td>
<td> 

(iii)</td>
<td> 

(iv)</td>
</tr>
<tr><td colspan="2">Alternativa PROIBIDA</td><td colspan="2">Fluxo Bloqueado</td></tr>
<tr>
<td colspan="2"> 

(v)</td>
<td colspan="2"> 

(vi)</td>
</tr>
</table>

IV.6 Varredura de Execução

A varredura de execução é um ciclo de troca e processamento de informações. Quando o programa está rodando, é feita a leitura de status de todos os pontos de entrada do processo, gerando uma tabela da Maquete Virtual para o APIMV com os valores atuais. O programa de controle é executado considerando os valores recebidos e os resultados são dispostos em outra tabela para a maquete, que por sua vez atualiza a situação de cada elemento de processo conforme os novos valores. A seguir, a Figura 14 mostra o ciclo de transferência de dados. Outra ilustração que representa essa comunicação encontra-se na Figura 5.

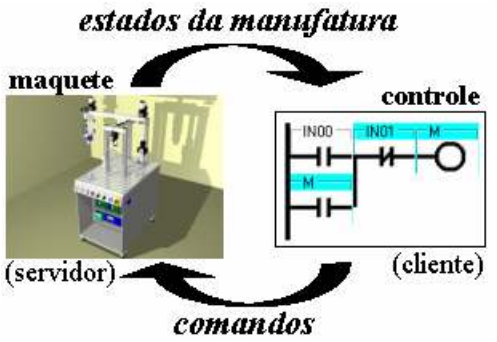


Figura 14 – Transferência de dados

IV.7 Simulador de *I/O*

A simulação de valores para pontos externos (estados de entrada e saída), torna o APIMV independente da integração com qualquer maquete virtual em se tratando de pequenas aplicações. Com isso, antes de partir para o *link* com a maquete, pode-se programar a lógica de controle e testá-la. Esse pequeno módulo, agregado ao ambiente de programação, conta com oito pontos de entradas digitais, quatro pontos de saídas digitais, duas entradas analógicas e duas saídas analógicas para simular os acionamentos. Veja ilustração da janela do simulador na Figura 15.

Figura 15 – Janela de simulação de I/O

Esse é um recurso de grande importância, que diferencia o APIMV de outros aplicativos semelhantes disponíveis no mercado. Alguns *softwares* de programação industrial do mercado não disponibilizam simulador; é o caso do PCWorx da Phoenix Contact [20], do PC12 da WEG [25], do Twido da Schneider [26], e tantos outros.

O simulador só pode ser chamado quando o programa se encontra em execução. Durante a varredura de

execução, em lugar de dados verídicos trocados com uma maquete virtual, o programa busca e atualiza valores na janela do simulador, que é manipulada pelo usuário, e trabalha com dois arquivos: de leitura (TI.txt), que capta os estados atualizados da tela; e outro de escrita (TO.txt), que é atualizado com os estados em resposta ao processamento da lógica. A seqüência de funcionamento do simulador de *I/O* é a seguinte:

- mudam-se os estados de entrada na tela;
- os estados da tela, tanto de entrada quanto de saída são gravados no arquivo de leitura;
- o programa lê os estados do arquivo e os dispõe na tabela de leitura;
- é realizada a varredura de execução (leitura, execução da lógica e escrita);
- os estados resultantes são dispostos na tabela e no arquivo de escrita.

Como ilustração, pode-se considerar um Diagrama Ladder (Figura 16) composto por um contato NA, um comparador do tipo GT (verifica se um número é maior que outro) e uma bobina comum. Nesse caso, quando alimentado pela continuidade lógica estabelecida pelo contato NA que está associado à variável DI0, o comparador deve verificar se a variável de entrada AI0 é maior que a variável de entrada AI1. Se a condição for verdadeira o fluxo de energia é liberado alimentando a bobina que aciona a variável de saída DO0.

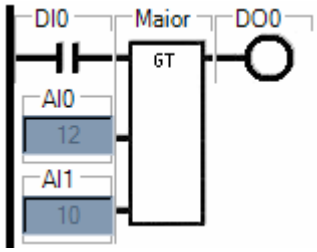


Figura 16 – Diagrama Ladder usado no exemplo de simulação

Quando o programa começa a rodar, o estado de ativação da variável DI0 é *false*, portanto não há continuidade lógica para alimentar o comparador (Figura 17).

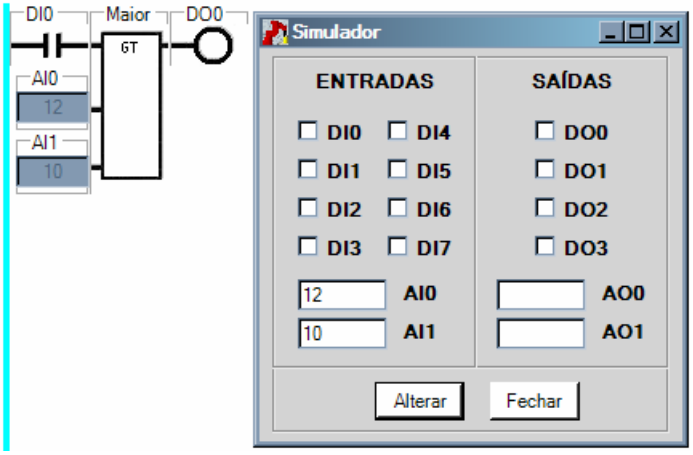


Figura 17 – Exemplo de simulação em estado inicial

Através do simulador, o estado de DI0 passa para *true*, liberando o fluxo de energia para alimentar o bloco de função que pode, assim, realizar sua operação. Alimentado, o comparador verifica que o valor de AI0 é maior que o valor de AI1 e libera o fluxo de energia para alimentar a bobina, que por sua vez aciona DO0 (Figura 18).

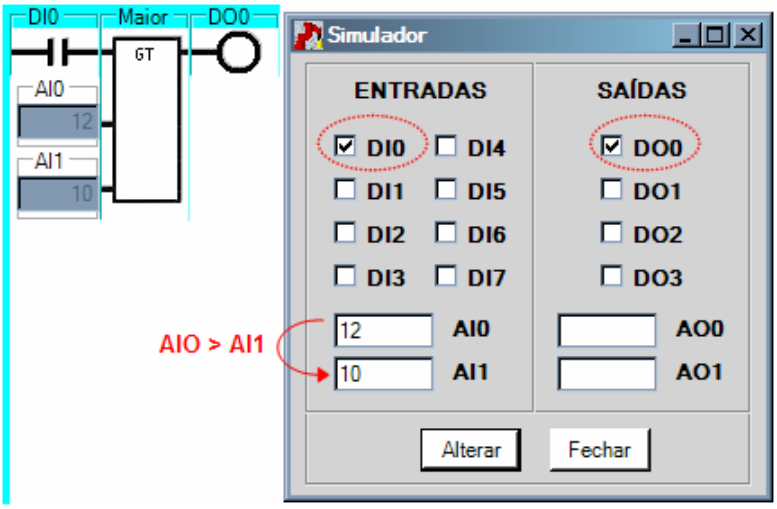


Figura 18 – Exemplo de simulação com acionamento

V COMPILAÇÃO LADDER

A compilação de um programa Ladder é a tradução da linguagem gráfica simbólica para uma linguagem mais estruturada, que combina condições, que caracterizam uma situação, e ações, que transformam o meio para descrever um novo cenário.

Programas em Ladder são compostos de várias linhas de comandos (*rungs*), com seus elementos dispostos em colunas, formando um diagrama e estabelecendo lógicas entre si. Durante a compilação, as *rungs* são lidas separadamente coluna a coluna, linha a linha, interpretando cada elemento e a condição a que é submetido, respeitando o sentido e a direção do fluxo de processamento, como já visto na ilustração da Figura 13, e liberando a passagem da energia pelos componentes, de acordo com a Figura 11, também apresentada anteriormente.

O processamento das linhas atualiza os status dos elementos e disponibiliza os valores para a próxima varredura (vide Figura 5).

V.1 Interpretação lógica

Há várias metodologias para se elaborar um Diagrama Ladder. Optou-se por uma tratativa matricial, como a abordagem do Miyagi (1996) [2].

Ao criar ou editar um programa, o espaço reservado para implementação do diagrama é considerado uma matriz bidimensional (M x N), como ilustra a Figura 19:

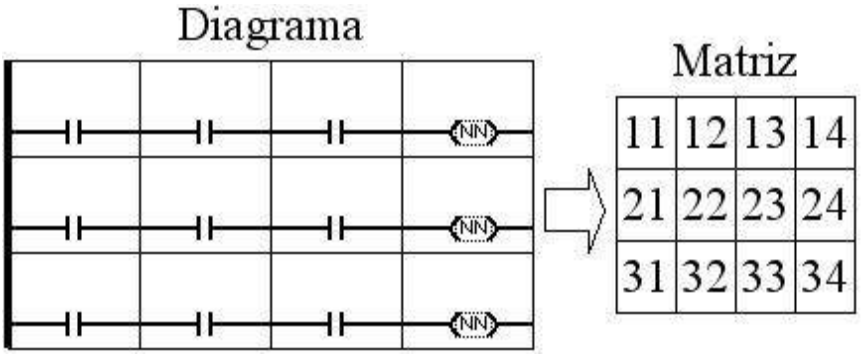


Figura 19 – Disposição matricial dos elementos

A interpretação da lógica Ladder pode seguir diversas metodologias. A adotada nesse projeto é a de Miyagi (1996) [2], onde é feito o tratamento matricial do diagrama, considerando o posicionamento dos contatos, seus estados e conexões.

Um Diagrama Ladder é composto basicamente por contatos, bobinas e ligações paralelas. O posicionamento

dos contatos é interpretado como uma matriz (1) de M linhas por N colunas:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1N} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2N} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3N} \\ \dots & \dots & \dots & \dots & \dots \\ a_{M1} & a_{M2} & a_{M3} & \dots & a_{MN} \end{bmatrix} \quad (1)$$

Para se determinar o valor que cada saída deve assumir, são considerados tanto o estado dos contatos, quanto o comportamento particular de cada um e o fluxo de energia através do diagrama. Para isso, na interpretação da lógica, são considerados os seguintes parâmetros:

- $P_{i, j-1}$: estados de ativação e desativação (ON / OFF) à esquerda dos contatos (após ligações verticais);
- $S_{i, j}$: estados dos contatos conforme comportamento individual de cada componente;
- $R_{i, j}$: estados de ativação e desativação (ON / OFF) à direita dos contatos;
- $L_{ab, j}$: relação de conexão vertical entre as linhas a e b da coluna j à direita;
- C_j: conexões do diagrama de relés.

Onde: i=1.. M, representa linhas; j=1..N,representa colunas; a=1..M-1, representa linha de referência; b=2..M, representa linha de verificação.

Esses dados são trabalhados em vetores e matrizes. A varredura do diagrama será sempre da esquerda para a direita e de cima para baixo, linha por linha de comando. A relação matricial é a seguinte (Tabela 10):

Tabela 10 – Equações que definem a relação entre vetores e matrizes do diagrama

$R_{i,j} = S_{i,j}.P_{i,j-1}$	(a)
$P_{i,j} = L_{ab,j}.R_{i,j}$	(b)
$P_{i,j} = L_{ab,j}.S_{i,j}.P_{i,j-1}$	(c)
$P_{i,j} = C_{i,j}.P_{i,j-1}$	(d)
a) Combinação serial (lógica E) entre estados à esquerda ($P_{i,j-1}$) e atuais ($S_{i,j}$) dos contatos; b) Combinação serial (lógica E) entre os estados em série ($R_{i,j}$) e as conexões verticais ($L_{ab,j}$) entre os contatos da coluna em questão; c) Forma extendida da Equação (b); d) Representação da Equação (b) apontando a relação entre as conexões do Diagrama (C_{ij}) e os estados à esquerda ($P_{i,j-1}$).	

A representação de um diagrama 5x5 é a seguinte (2) e (3):

$$\begin{bmatrix} R_{1,j} \\ R_{2,j} \\ R_{3,j} \\ R_{4,j} \\ R_{5,j} \end{bmatrix} = \begin{bmatrix} S_{1,j} & 0 & 0 & 0 & 0 \\ 0 & S_{2,j} & 0 & 0 & 0 \\ 0 & 0 & S_{3,j} & 0 & 0 \\ 0 & 0 & 0 & S_{4,j} & 0 \\ 0 & 0 & 0 & 0 & S_{5,j} \end{bmatrix} \cdot \begin{bmatrix} P_{1,j-1} \\ P_{2,j-1} \\ P_{3,j-1} \\ P_{4,j-1} \\ P_{5,j-1} \end{bmatrix} \qquad (2)$$

$$\begin{bmatrix} P_{1,j} \\ P_{2,j} \\ P_{3,j} \\ P_{4,j} \\ P_{5,j} \end{bmatrix} = \begin{bmatrix} 1 & L_{12,j} & L_{13,j} & L_{14,j} & L_{15,j} \\ L_{12,j} & 1 & L_{23,j} & L_{24,j} & L_{25,j} \\ L_{13,j} & L_{23,j} & 1 & L_{34,j} & L_{35,j} \\ L_{14,j} & L_{24,j} & L_{34,j} & 1 & L_{45,j} \\ L_{15,j} & L_{25,j} & L_{35,j} & L_{45,j} & 1 \end{bmatrix} \cdot \begin{bmatrix} R_{1,j} \\ R_{2,j} \\ R_{3,j} \\ R_{4,j} \\ R_{5,j} \end{bmatrix} \qquad (3)$$

Onde: A matriz R representa os estados à direita dos contatos e a matriz P representa os estados à esquerda dos contatos.

A localização dos parâmetros nos elementos está representada na Figura 20, que mostra uma coluna genérica num diagrama contendo cinco linhas de comando:

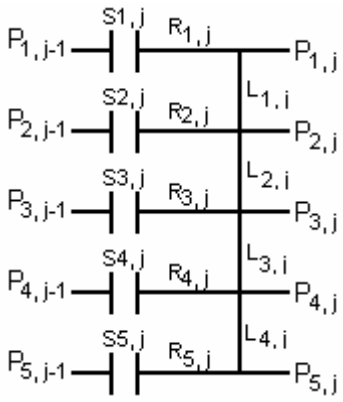


Figura 20 – Localização dos parâmetros dos elementos

Os resultados fornecidos pela matriz P da última coluna são os valores que devem ser assumidos pelas bobinas no final de cada linha.

V.1.1 Ligações entre os elementos

As ligações entre os elementos podem ser diretas ou por fios, que também têm seus pontos de conexão. Os fios horizontais estabelecem ligações seriais (lógica E) e os fios verticais estabelecem ligações paralelas (lógica OU).

No exemplo de diagrama a seguir (Figura 21), os elementos apresentam os seguintes pontos de conexão (Tabela 11):

Tabela 11 – Exemplo de Pontos de Conexão

Elemento	Pe	Ps
Elemento Contato I1.0	Pe=11	Ps=12
Elemento Contato I1.1	Pe=12	Ps=13
Elemento Fio Horizontal_1314	Pe=13	Ps=14
Elemento Bobina Q1.0	Pe=14	Ps=15
Elemento Contato Q1.0	Pe=21	Ps=22
Elemento Fio Vertical_1121	Pe=11	Ps=21
Elemento Fio Vertical_1222	Pe=12	Ps=22

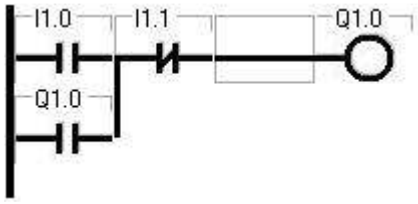


Figura 21 – Exemplo de Diagrama Ladder

Conhecendo os pontos de conexão de cada elemento, é possível identificar as ligações. No caso do exemplo, I1.0 está em paralelo com o contato Q1.0 porque há um fio vertical ligando seus pontos de conexão Pe (respectivamente 11 e 21), e outro fio vertical ligando seus pontos de conexão Ps (respectivamente 12 e 22).

V.1.2 Montagem da Matriz S e da Matriz L_{ab}

Na montagem das matrizes, trabalha-se diretamente com os pontos de conexão dos elementos, o que informa todas as ligações do diagrama.

A matriz S é composta pelos estados dos elementos. É uma matriz atualizada a cada varredura do programa

na sua execução, quando o valor de cada elemento é buscado e inserido na matriz de acordo com o seu posicionamento no Diagrama de Contatos, como mostra a equação 4:

$$\mathbf{S} = \begin{bmatrix} S_{11} & S_{12} & S_{13} & \dots & S_{1n} \\ S_{21} & S_{22} & S_{23} & \dots & S_{2n} \\ S_{31} & S_{32} & S_{33} & \dots & S_{34} \\ \dots & \dots & \dots & \dots & \dots \\ S_{m1} & S_{m2} & S_{m3} & \dots & S_{mn} \end{bmatrix} \quad (4)$$

Sendo: S o estado de ativação; n = coluna e m = linhas.

Assim como a matriz S, a matriz L_{ab} (equação 5) usa como referência os pontos de conexão dos elementos (Pe e Ps), neste caso dos elementos verticais, para determinar a existência ou não de ligações paralelas. Para cada coluna é montada uma matriz de ligações entre as linhas. A equação 5 representa a matriz L_{ab} de uma coluna genérica j, apontando a relação de conexões verticais entre as linhas a e b da coluna j:

$$Lab, j = \begin{bmatrix} 1 & L_{12,j} & L_{13,j} & L_{14,j} & L_{15,j} \\ L_{12,j} & 1 & L_{23,j} & L_{24,j} & L_{25,j} \\ L_{13,j} & L_{23,j} & 1 & L_{34,j} & L_{35,j} \\ L_{14,j} & L_{24,j} & L_{34,j} & 1 & L_{45,j} \\ L_{15,j} & L_{25,j} & L_{35,j} & L_{45,j} & 1 \end{bmatrix} \quad (5)$$

Sendo: j = coluna, 1~5; a = linhas, 1~4 e b = linhas, 2~5

Os estados são interpretados conforme a Tabela 12:

Tabela 12 – Interpretação do elemento

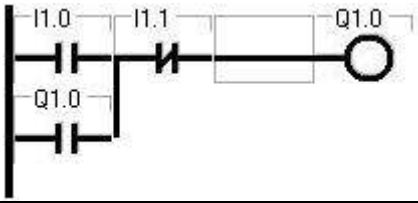
S i, j	Representação Gráfica	X i, j 	X i, j 		
	Representação Matemática	X i, j	$\overline{X}$ i, j	1	0

L k, j	Representação Gráfica		
	Representação Matemática	1	0

Os blocos são interpretados como contatos de acordo com a habilitação e comportamento que cada tipo de função exerce. São elementos condicionais, que agem conforme combinação entre valores. Como exemplo pode-se citar o contador, cuja liberação do fluxo de energia acontece quando o limite de contagem é atingido, ou seja, nessa situação o interpretador lógico o representa matematicamente com o algarismo 1 (comportamento do bloco no APÊNDICE B).

No caso do exemplo (Figura 21), as matrizes e equações ficam da seguinte forma (Tabela 13):

Tabela 13 – Matrizes de estados



Disposição dos Elementos

$$A = \begin{bmatrix} I_{1.0} & \overline{I_{1.1}} & Fio \\ Q_{1.0} & 0 & 0 \end{bmatrix}$$

Estados de Ativação

$$S = \begin{bmatrix} S_{11} & \overline{S_{12}} & 1 \\ S_{21} & 0 & 0 \end{bmatrix}$$

Operações com as Matrizes R

$$\begin{bmatrix} R_{1,1} \\ R_{2,1} \end{bmatrix} = \begin{bmatrix} \mathbf{S}_{11} & 0 \\ 0 & \mathbf{S}_{21} \end{bmatrix} . \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} R_{1,2} \\ R_{2,2} \end{bmatrix} = \begin{bmatrix} \mathbf{S}_{12} & 0 \\ 0 & \mathbf{S}_{22} \end{bmatrix} . \begin{bmatrix} P_{1,1} \\ P_{2,1} \end{bmatrix}$$

$$\begin{bmatrix} R_{1,3} \\ R_{2,3} \end{bmatrix} = \begin{bmatrix} \mathbf{S}_{13} & 0 \\ 0 & \mathbf{S}_{23} \end{bmatrix} . \begin{bmatrix} P_{1,2} \\ P_{2,2} \end{bmatrix}$$

$$R_{1,1} = S_{11}$$
$$R_{2,1} = S_{21}$$
$$R_{1,2} = \overline{S_{12}}.P_{1,1}$$
$$R_{2,2} = S_{22}.P_{2,1} = 0.P_{2,1} = 0$$
$$R_{1,3} = S_{13}.P_{1,2} = 1.P_{1,2} = P_{1,2}$$
$$R_{2,3} = S_{23}.P_{2,2} = 0.P_{2,2} = 0$$

$$R_{i,j} = S_{i,j}.P_{i,j-1}$$

Ligações entre as linhas 1 e 2 (a=1)(b=2)(j=1~3)

$$L_{12,j} = \begin{bmatrix} L_{12,1} & L_{12,2} & L_{12,3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

$$L_{121} = \begin{bmatrix} 1 & I_{1.0}\_ligado\_Q_{1.0} \\ Q_{1.0}\_ligado\_I_{1.0} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$L_{122} = \begin{bmatrix} 1 & I_{1.1}\_ligado\_0 \\ 0\_ligado\_I_{1.1} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$L_{123} = \begin{bmatrix} 1 & Fio\_ligado\_0 \\ 0\_ligado\_Fio & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Resultados das Matrizes P

$$\begin{bmatrix} P_{1,1} \\ P_{2,1} \end{bmatrix} = \begin{bmatrix} 1 & L_{12,1} \\ L_{12,1} & 1 \end{bmatrix} . \begin{bmatrix} R_{1,1} \\ R_{2,1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} . \begin{bmatrix} R_{1,1} \\ R_{2,1} \end{bmatrix} \quad \begin{matrix} P_{1,1} = R_{1,1} + R_{2,1} = S_{1,1} + S_{2,1} \\ P_{2,1} = R_{1,1} + R_{2,1} = S_{1,1} + S_{2,1} \end{matrix}$$

$$\begin{bmatrix} P_{1,2} \\ P_{2,2} \end{bmatrix} = \begin{bmatrix} 1 & L_{12,2} \\ L_{12,2} & 1 \end{bmatrix} . \begin{bmatrix} R_{1,2} \\ R_{2,2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} . \begin{bmatrix} R_{1,2} \\ R_{2,2} \end{bmatrix} \quad \begin{matrix} P_{1,2} = R_{1,2} = \overline{S_{12}}.P_{1,1} \\ P_{2,2} = R_{2,2} = 0 \end{matrix}$$

$$\begin{bmatrix} P_{1,3} \\ P_{2,3} \end{bmatrix} = \begin{bmatrix} 1 & L_{12,3} \\ L_{12,3} & 1 \end{bmatrix} . \begin{bmatrix} R_{1,3} \\ R_{2,3} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} . \begin{bmatrix} R_{1,3} \\ R_{2,3} \end{bmatrix} \quad \begin{matrix} P_{1,3} = R_{1,3} = P_{1,2} \\ P_{2,3} = R_{2,3} = 0 \end{matrix}$$

$$P_{i,j} = L_{ab,j}.R_{i,j}$$
$$P_{i,j} = L_{ab,j}.S_{i,j}.P_{i,j-1}$$

Nota: Operações do tipo "." representam a lógica booleana E (AND) e operações do tipo "+" representam a lógica booleana OU(OR)

No exemplo citado, o resultado (saída) é gerado na coluna 4 (última coluna), portanto essa coluna é ignorada pelos cálculos para obtenção do estado resultante, isto é, trabalha-se apenas com as colunas 1, 2 e 3 para análise de liberação e interrupção do fluxo de energia nas linhas.

Como pode ser visto na Tabela 13, o estado de ativação dos elementos I1.0 e Q1.0 são empregados diretamente nas equações por estarem associados a componentes de programação do tipo normalmente aberto (NA), enquanto o elemento I1.1 tem seu estado de ativação invertido ao ser aplicado às fórmulas, devido ao comportamento reverso do componete de programação normalmente fechado (NF). O comportamento de cada componente é abordado no próximo tópico desse documento.

Assim, supondo uma situação em que o elemento I1.0 esteja habilitado (estado de ativação igual a 1), os elementos I1.1 e Q1.0 estejam desabilitados (estado de ativação igual a 0) e aplicando-se as relações equacionais demonstradas na Tabela 13, é obtido o seguinte resultado: $P_{1,1}$=1, $P_{2,1}$=1, $P_{1,2}$=1, $P_{2,2}$=0, $P_{1,3}$=1 e $P_{2,3}$=0. A saída Q1.0 é portanto ativada, pois $P_{1,1}$= $P_{1,2}$= $P_{1,3}$=1 o que libera a energia até a quarta coluna onde se encontra a bobina (componente de programação associado ao elemento Q1.0).

V.2 Comportamento isolado

Cada elemento de programação tem seu comportamento particular diante do estado de ativação a que é submetido, respeitando a Norma IEC61131-3, conforme visto no item IV.3.

Observando, por exemplo, as formas de onda de comportamento do contato normalmente aberto (NA), nota-se que o estado do elemento acompanha a alimentação e o estado de ativação da variável associada a ele (Figura 22). O comportamento de outros elementos pode ser visto no APÊNDICE B.

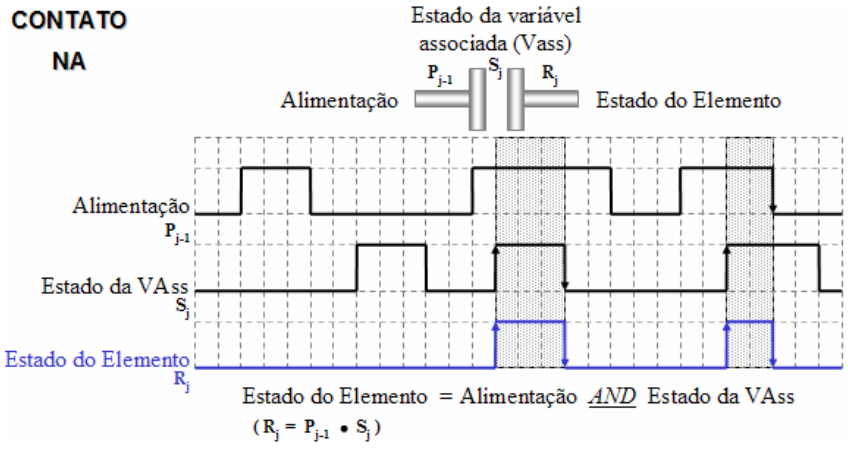


Figura 22 – Comportamento do contato NA

VI TELAS DO APIMV

O ambiente de programação conta com poucas telas, dentre as quais quatro são principais:

- Abertura: apresenta o *software*, com dados das instituições envolvidas no projeto;
- Estrutura: exibe a arquitetura de unidades de programação e recursos de *hardware*;
- Programação: onde se cria, abre e edita diagrama de contatos e se executa o programa;
- Edição: permite a edição de uma linha de comando da lógica e teste da mesma;

Janelas auxiliares são abertas durante a edição para conduzir a montagem da lógica, permitindo a configuração de elementos e outras interações com o usuário.

VI.1 Tela de Abertura

É a tela inicial, que faz a chamada da próxima janela por meio de um botão (APIMV). A Figura 23 mostra a aparência da Tela de Abertura:

Figura 23 – Tela de Abertura

VI.2 Tela de Estrutura

Tela destinada à apresentação, em formato de árvore, da arquitetura da aplicação e do modo operacional do programa. Deve abranger programação e características físicas do sistema, para permitir o acesso às variáveis do sistema, a escolha da linguagem de codificação para implementar o programa de controle, o relacionamento entre programas escritos em linguagens distintas, a ordem de execução entre vários programas, a associação de programas específicos a ocorrências de eventos e alarmes, a definição de programas especiais periódicos e a conexão à maquete virtual.

Essa parte do projeto está esquematizada, porém em desenvolvimento. Visto que é responsável pelo vínculo e integração entre diversos *softwares* que compõem a dinâmica do controle e a manipulação da maquete, aguarda a conclusão de cada módulo a ser acoplado ao ambiente de programação, como: o *software* interpretador da linguagem ladder (tema desse trabalho, já inserido no ambiente), o algoritmo dedicado ao seqüenciamento lógico da interoperabilidade entre programas, e outros *softwares* interpretadores de linguagens de programação.

A Figura 24 mostra um exemplo ilustrativo da janela com as opções de descrição, programação, variáveis e configuração. A Descrição é uma opção para que sejam inseridos comentários que caracterizem o projeto. No item Programação deve ser organizada a lista de programas, agrupados conforme a linguagem de codificação em que foram implementados. O nó principal de cada linguagem permite criar uma nova lógica, mas apenas a linguagem ladder encontra-se disponível, pois as demais se encontram em desenvolvimento. Em Variáveis, apresenta-se a listagem das variáveis já definidas disponíveis à implementação da lógica. Elas podem ser referentes à maquete, definidas pelo usuário ou mesmo ligadas ao simulador do ambiente de programação. E no item Configuração é onde se escolhe a maquete virtual a ser manipulada pelo programa de controle. Pode-se optar por não estabelecer uma comunicação com maquete; nesse caso, adota-se o simulador e os dados são obtidos de um modelo padrão inscrito em arquivos de transferência de dados.

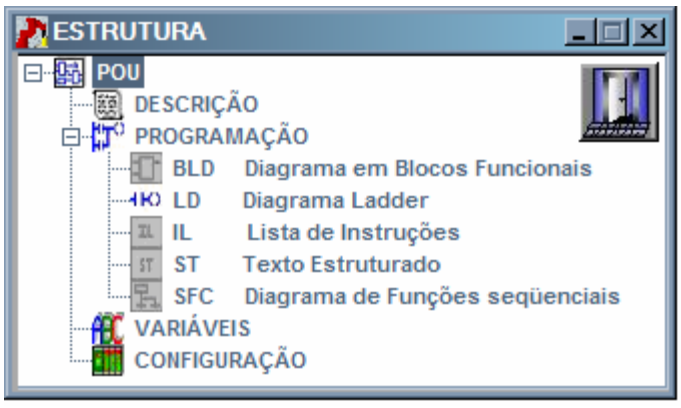


Figura 24 – Tela de Estrutura

VI.3 Tela de Programação

O Diagrama Ladder é montado nessa tela, ou seja, onde ficam dispostas todas as linhas de comando. A execução do programa é feita nesse mesmo local. Os campos que compõem essa tela são:

- Barra de Ferramentas: onde se localizam os botões;
- Área de Programa: onde é exibido o diagrama formado pelas linhas de comando (*rung*);
- Cabeçalhos: região de comentários que intitulam as rungs;

A Figura 25 apresenta a imagem da janela identificando os campos, e a Tabela 14 traz a função dos botões.

Tabela 14 – Botões da Tela de Programa.

	Adicionar *rung*: chama Tela de Edição para criar linha de comando.		Inserir *rung*: chama Tela de Edição para inserir linha de comando.
	Editar *rung*: chama Tela de Edição para editar linha de comando.		Rodar: executa o programa.
	Limpar tela: apaga tudo que há na tela.		Parar: interrompe execução do programa.
	Abrir: chama janela para localizar diagrama armazenado;		Simular: chama simulador de variáveis.
	Gravar: chama janela para especificar local de armazenamento.		Sair: fecha a tela de programa

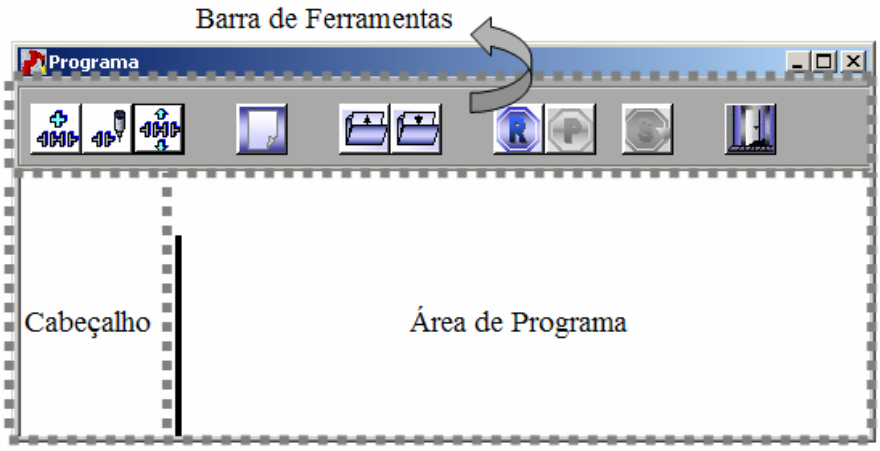


Figura 25 – Tela de Programa.

VI.4 Tela de Edição

A Tela de Edição permite criar, gravar, abrir, editar e executar/simular uma linha de comando. É composta pelos seguintes campos:

- Barra de Ferramentas: disponibiliza os botões;
- Área de trabalho (edição): onde ocorre a edição propriamente dita;
- Cabeçalhos: região de comentários que intitulam as rungs;
- Área reservada para seleção: se reserva aos procedimentos de seleção de objeto; como apresentação de imagens correspondentes a cada tipo de elemento por exemplo.

A Figura 26 mostra a janela com identificação dos campos, e os botões estão identificados na Tabela 15, onde é descrita a função cada um deles.

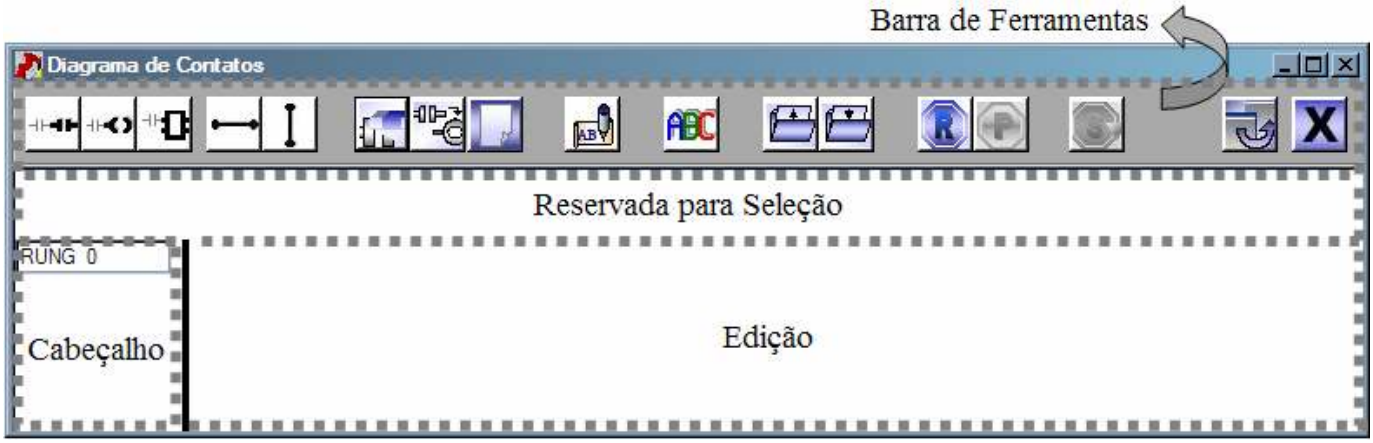


Figura 26 – Campos da Tela de Edição.

Tabela 15– Botões da Tela de Edição.

Botões	Descrição	Botões	Descrição
	Contato: insere elemento tipo contato		Limpar tela: apaga tudo que tem na tela
	Bobina: insere elemento tipo bobina		Abrir: chama janela para localizar diagrama armazenado
	Bloco: insere elemento tipo bloco de função		Gravar: chama janela para especificar local de armazenamento
	Fio_H: insere fio de conexão horizontal		Rodar: executa o programa
	Fio_V: insere fio de conexão vertical		Parar: interrompe execução do programa
	Apagar: elimina o elemento selecionado		Simular: chama simulador de variáveis
	Editar: edita o elemento selecionado		Confirmar: adiciona rung ao diagrama
	Comentário: insere texto		Cancelar: retorna ignorando edição da rung
	Variáveis: lista as variáveis já declaradas		

VI.4.1 Comentários

Os comentários podem ser fixados em qualquer parte do espaço de edição. O campo de preenchimento dos caracteres pode ser aumentado alargando-se a janela lateralmente. A Figura 27 mostra um exemplo.

Figura 27 – Exemplo de inserção de comentário.

VI.4.2 Parametrização

A seleção de qualquer dos três elementos básicos: contato, bobina ou bloco, implica na chamada da Janela de Parâmetros. Nessa janela são definidas as características do elemento conforme categoria e opção do programador.

A maioria dos parâmetros determinados é de configuração interna, dados usados na interpretação da lógica, na exibição do elemento na tela, ou em qualquer outra operação do programa sobre ele.

Ao selecionar um elemento, são listadas as variáveis que podem ser referenciadas por ele, compatíveis com a categoria à qual o componente pertence. No caso de um contato, são disponibilizadas todas as variáveis carregadas e criadas no ambiente (entradas, saídas e internas); se for selecionada uma bobina, as entradas não são listadas; e se for um bloco, são listadas variáveis coerentes com sua categoria e tipo, e ainda haverá listagens correspondentes às variáveis associadas responsáveis pelo seu funcionamento (preset por exemplo).

A Figura 28 ilustra um exemplo dessa janela para um elemento da categoria temporizador. O exemplo define um temporizador com os seguintes parâmetros: tipo TON, cujo funcionamento pode ser conferido no item IV.3; preset (PT) associado a uma variável denominada AI1, com valor igual a 2000[ms]; contagem (ET) associada a uma variável denominada AO1; pontos de conexão de entrada (Pin) e saída (Pout) definidos como 12 e 13 respectivamente; e comentário.

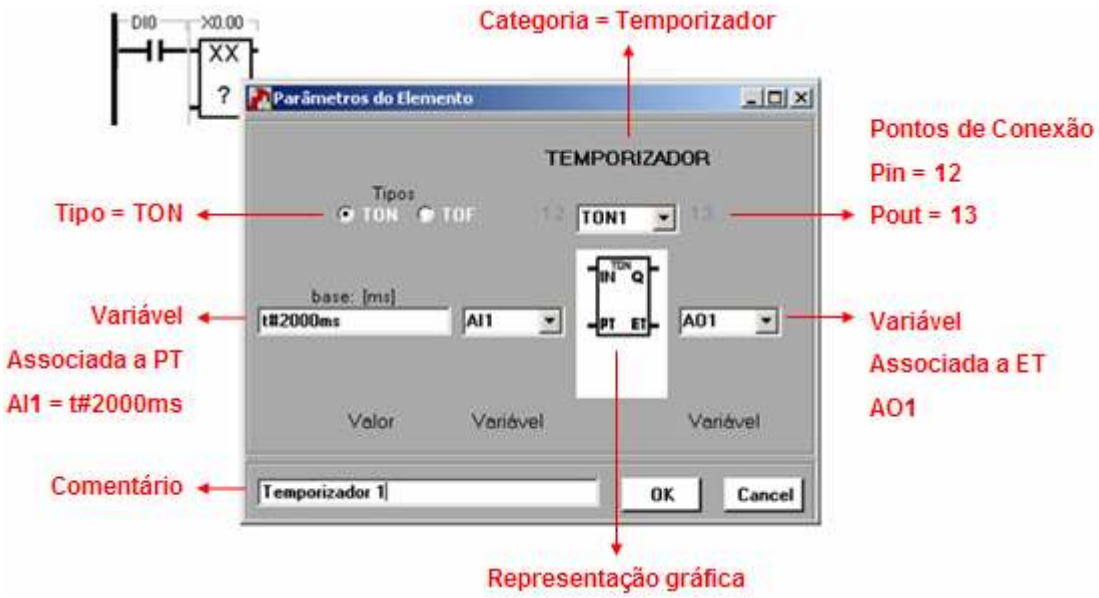


Figura 28 – Exemplo de Janela de Parametrização de um temporizador.

VI.5 Declaração de Variáveis

Quando um elemento é relacionado a uma variável inexistente, ou seja, com nome desconhecido para o programa, a janela de declaração de variáveis é acionada. Os parâmetros são setados conforme categoria do elemento escolhido e o atributo (que indica se o ponto é externo ou interno) é fixado na opção interna, visto que todos os pontos externos são informados pela maquete, ou no caso de simulação, definidos pelo modelo padrão. A Figura 29 apresenta um exemplo de Janela de Declaração de Variável.

Figura 29 – Exemplo de Janela de Declaração de Variável.

VI.6 Tela de Simulação

A Tela de Simulação é a mesma acessada na Área de Programa e na Área de Edição. Fica disponível somente no modo de simulação, isto é, não pode ser chamada durante a execução de programa integrado à maquete. Nessa tela o usuário pode manipular oito entradas digitais e duas entradas analógicas, além da possibilidade de acompanhar resultados através de quatro saídas digitais e duas saídas analógicas; desde que esteja operando em modo simulador e tenha associado devidamente os pontos padrões aos elementos de programação do diagrama ladder.

A Figura 30 ilustra um exemplo de simulação de uma linha de comando contendo um contato normal aberto habilitando um temporizador TON que está ligado a uma bobina comum. Na parametrização do temporizador foram associadas as variáveis analógicas AI0 e AO0 para determinar o limite da contagem de tempo e permitir o acompanhamento da contagem respectivamente. Os valores de entrada tornam-se válidos para o programa mediante confirmação ao pressionar o botão Alterar. Os resultados são atualizados automaticamente conforme resposta da lógica do programa.

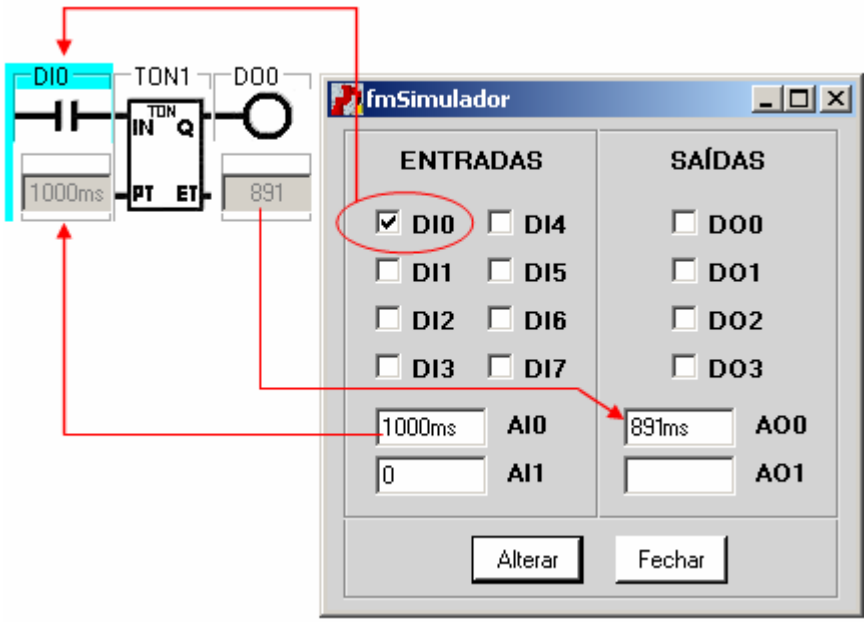


Figura 30 - Exemplo de Janela de Simulação de I/O

VII APLICABILIDADE

Os *softwares* simuladores e de supervisão desempenham papel importante na modelagem de sistemas industriais sendo utilizados em planejamento e treinamento. Seus recursos complementam o projeto teórico com um embasamento prático, permitindo variações na programação de equipamentos, testes, ajustes, simulações, adequação de elementos de processo, reestruturação, otimização de layout e tempo de ciclo, definição de procedimentos a serem seguidos na implementação e análise da resposta de cada elemento.

Para se obter um resultado satisfatório de análise e decisão ao se estudar um modelo através de um simulador, este tem que ser completo e de filosofia ampla, que forneça o máximo de informações sobre o sistema e que seja fiel à sua dinâmica ao representar a realidade. O mercado industrial oferece muitos aplicativos de simulação, mas são raros os que se assemelham ao sistema de manufatura virtual que vem sendo concebido nessa universidade. A maioria dos aplicativos dessa categoria ainda não atingiu um nível ideal; são incompletos, dedicados e/ou limitados.

Muitos simuladores trabalham apenas em duas dimensões, com representação simbólica de componentes e não reproduzem seus comportamentos com fidelidade. Alguns trabalham em 3D, mas ainda são deficientes diante do propósito de imitar a realidade e permitir imaginar situações diversas em cima de uma aplicação. É o caso do COSIMIR [8], que além de não criar uma realidade virtual, não dispõe de recursos para programar lógicas de controle e estabelece um altíssimo custo que implica num investimento considerável das empresas e torna inviável sua aplicação em Universidades (fins didáticos). Outros *softwares* empregam a realidade virtual, como o Flexman[7], porém são poucos e também restritos, não permitirem a edição e programação de novos elementos gráficos, para definir formas, comportamento e dinâmica, e são direcionados a um fabricante ou equipamento específico.

A manufatura virtual que está sendo desenvolvida nessa universidade, além de ter um custo irrisório, perfeitamente cabível no contexto das universidades e interessante para o orçamento das empresas, será um diferencial em sua categoria devido aos recursos que a promovem.

Para construir uma realidade virtual é preciso dispor de uma tecnologia de *software* muito avançada. Portanto está sendo empregado na implementação de cada módulo do sistema o que há de mais recente no mercado. Os objetos de campo (por exemplo: um braço mecânico e uma mesa de perfuração de peças) são representados por maquetes virtuais bastante precisas, todas em 3D, geradas com tecnologia aplicada a jogos, retratando com fidelidade imagens e comportamentos de cada elemento gráfico.

A preocupação em manter uma interface didática e intuitiva com o usuário para facilitar a operação da

manufatura virtual é constante, pois o operador do sistema deve se sentir a vontade e se preocupar apenas em projetar. Esse sistema preza muito a liberdade imaginativa e expressiva na criação das aplicações, com a intenção de explorar ao máximo as possibilidades de implementação de um projeto. A possibilidade de combinar vários dispositivos de diversas maneiras e poder trabalhar sobre esses arranjos é fundamental em momentos de decisão, no fechamento de projetos, ou mesmo no treinamento de um profissional que poderá enxergar a dinâmica da manufatura em seus diferentes ângulos de aplicação, e assim aprimorar seus conceitos.

A diversificação de aplicações é favorecida pela possibilidade de editar e adicionar objetos ao modelador da maquete. O código fonte é implementado usando programação orientada a objetos, que facilita a manipulação das propriedades dos elementos já presentes no sistema e permite a disponibilidade de recursos padrões para elaboração de novos integrantes no ambiente gráfico.

O fato de ser independente de fabricante e equipamento torna esse sistema ainda mais vantajoso, porque o usuário pode optar entre os equipamentos de diferentes fornecedores, combinando suas peculiaridades e montando o perfil adequado diante dos requisitos de sua aplicação.

Na modelagem dos objetos a precisão é incondicional, e são considerados ainda os agentes do meio: ação da gravidade e colisão; fatores inexistentes nos demais *softwares*.

Criar um sistema mecânico automatizado sem controle lógico de seus dispositivos, estabelecendo apenas tarefas de cunho repetitivo não permite a previsão e resolução de situações adversas; como na ocorrência de falha em um sensor, em que o sistema pode simplesmente estacionar ou provocar algum tipo de acidente, caso não haja uma alternativa lógica para ação do elemento de campo que depende da informação correta do suposto sensor defeituoso. A lógica dos comandos é fundamental, pois é quem dita as ações dos objetos na maquete e pode mudar completamente um processo de fabricação.

Isso evidencia a relevância de haver um simulador de lógica de controle acoplado ao Sistema de Manufatura Virtual. Diante dessa necessidade, um recurso importante presente nesse sistema de manipulação de processos, não encontrado nos demais simuladores, é um ambiente para programar os dispositivos controláveis. É ele quem permite testar as lógicas de controle que devem manipular os elementos de campo na produção antes de serem efetivamente praticadas.

Por isso é grande a responsabilidade incutida no Ambiente de Programação e Integração para Manufatura Virtual (APIMV), pois associado à realidade da simulação implementada, torna-se uma ferramenta extremamente prática e decisiva.

Também é um ponto positivo a padronização desse ambiente de programação, pois ao respeitar a norma que rege os regulamentos para as linguagens industriais, oferece a vantagem do código de controle não precisar de

conversão quando posto em prática na execução do projeto, tornando direta a elaboração do controle.

Outro ponto favorável à aplicabilidade desse sistema é a instalação e interligação do conjunto. O desenvolvimento do sistema de manufatura virtual está sendo feito sobre uma plataforma .NET, portanto dispõe de toda a praticidade, versatilidade e portabilidade que o Framework .Net oferece para sistemas distribuídos, o que lhe permite se compor numa estrutura modular e garante a interoperabilidade entre seus programas. A mobilidade e integração entre as partes do sistema, dispostas em múltiplas arquiteturas de base, fornece opções de disposição ao sistema, dependendo de sua concepção de uso: centralizada ou distribuída, usando o protocolo TCP/IP.

VIII RESULTADOS

O projeto está todo idealizado. Das maquetes, têm-se modelos já implementados, outros em desenvolvimento ou em testes. Do ambiente de programação, projeto em questão nesse trabalho, o programador Ladder está concluído com suas funções básicas, como fora proposto.

O APIMV foi desenvolvido com base na Norma IEC61131-3, portanto dispõe de uma fácil programação aos usuários que já dominam os elementos lógicos previstos nessa norma. Aos que não têm tanta familiaridade com programação industrial também não é complicado, pois o ambiente é bastante simples, um *software* intuitivo e claro.

Foram realizados testes e simulações do ambiente de programação para aferição de sua funcionalidade. Os resultados foram satisfatórios e atenderam às expectativas. As respostas foram coerentes, não ocorreram falhas e as limitações impostas pela Norma IEC61131-3 [1] foram respeitadas.

Os primeiros testes de lógica foram realizados sem a comunicação com maquetes virtuais. Observou-se o comportamento *online* de muitas lógicas para aferição dos elementos de programação. A resposta foi satisfatória.

Para ilustrar os testes realizados no ambiente de programação sem o *link* com qualquer maquete virtual, segue a simulação do item VIII.1 feita com dois contatos, relacionados a duas entradas, e uma bobina, associada a um ponto de saída.

Após testar o funcionamento individual do ambiente de programação, foram realizados testes de comunicação apenas de *link* com a maquete virtual. As informações fornecidas pela maquete foram captadas com sucesso na tabela de leitura e os dados copiados na tabela de escrita foram devidamente lidos pela maquete. A dinâmica automática da comunicação, que consiste na interpretação correta dos comandos do programa ladder para a maquete, não foi avaliada devido às intervenções e ajustes na padronização que o módulo de comunicação ainda vinha sofrendo.

Para exemplificar e mostrar os resultados, encontra-se no item VIII.2 uma simulação feita para uma maquete.

Outros testes podem ser apreciados no APÊNDICE C, o APÊNDICE D ou na prática.

VIII.1 Simulação sem maquete

O Diagrama Ladder (DL) dessa simulação é formado por um contato normalmente aberto em série com um contato normalmente fechado acionando uma bobina comum.

A simulação consistiu no acionamento dos pontos de entrada e na observação da resposta do diagrama ao estímulo aplicado. Tabelas geradas durante a simulação (Tabela 16 à Tabela 20) mostram os resultados obtidos apresentando a imagem gráfica do DL, que retrata a situação em que se encontrava nas etapas da simulação, o estado de cada elemento, resultante de seu comportamento diante do valor da variável referenciada por ele, e o conteúdo das tabelas de comunicação entre o ambiente de programação e o simulador de *I/O* (o formato das tabelas de leitura e escrita foi abordado no item IV.2).

Tabela 16 – Diagrama Ladder offline e tabelas de comunicação.

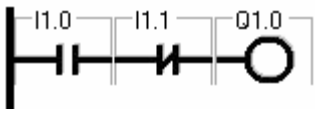


Diagrama Ladder (off-line)

Tabela de Leitura					**Estado do Elemento**	**Tabela de Escrita**
I/O	***Nome***	***Comentário***	***Valor Inicial***	***Valor Final***		
E	I1.0	Entrada 1.0	false	false	false	-
E	I1.1	Entrada 1.1	false	false	false	-
S	Q1.0	Saída 1.0	false	false	false	false

A Tabela 16 mostra o DL em estado *offline* (programa parado), por isso todos os estados de ativação (valores inicial e final) de todas as variáveis de entrada e saída estão apresentados como *false*, ou seja, pontos de *I/O* desativados (não energizados). Inclusive o contato normalmente fechado (NF) não tem o estado do elemento inverso ao da variável associada a ele nessa tabela.

Tabela 17 – Diagrama Ladder em estado inicial e tabelas de comunicação.

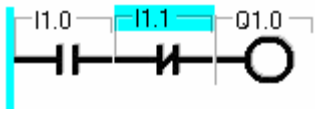


Diagrama Ladder (momento inicial)

Tabela de Leitura					**Estado do Elemento**	**Tabela de Escrita**
I/O	***Nome***	***Comentário***	***Valor Inicial***	***Valor Final***		
E	I1.0	Entrada 1.0	false	false	false	-
E	I1.1	Entrada 1.1	false	false	***true***	-
S	Q1.0	Saída 1.0	false	false	false	false

A Tabela 17 ilustra o início da execução do programa. O momento inicial desse DL tem como característica todos os pontos de *I/O* desativados, com valor final igual ao valor inicial (*false*). Como o estado de cada elemento depende de seu comportamento diante do estado de ativação assumido (valor final) por sua variável associada naquela etapa, o segundo contato, por ser um elemento normalmente fechado, demonstra comportamento inverso ao da variável referenciada por ele, o que leva seu estado a permanecer ativo (*true*) enquanto a variável estiver desativada, ou seja, valor final de I1.1 igual a *false* implica estado do elemento NF igual a *true*. Como não houve liberação do fluxo de energia até a saída Q1.0, esta também tem estado *false*.

Tabela 18 - Diagrama Ladder no estágio 1 e tabelas de comunicação.

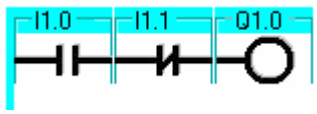


Diagrama Ladder (estágio1)

Tabela de Leitura					**Estado do Elemento**	**Tabela de Escrita**
I/O	***Nome***	***Comentário***	***Valor Inicial***	***Valor Final***		
E	I1.0	Entrada 1.0	false	***true***	***true***	-
E	I1.1	Entrada 1.1	false	false	***true***	-
S	Q1.0	Saída 1.0	false	false	***true***	***true***

Na Tabela 18, o acionamento da variável I1.0 (valor final igual a *true*) associada ao elemento normalmente aberto (NA), cujo estado acompanha o estado de ativação de sua variável, fez com que o fluxo de energia fosse liberado até a saída, que foi também ativada, já que o contato NF não foi alterado nessa etapa.

Tabela 19- Diagrama Ladder no estágio 2 e tabelas de comunicação.

Diagrama Ladder (estágio2)

Tabela de Leitura					**Estado do Elemento**	**Tabela de Escrita**
I/O	***Nome***	***Comentário***	***Valor Inicial***	***Valor Final***		
E	I1.0	Entrada 1.0	false	***true***	***true***	-
E	I1.1	Entrada 1.1	false	***true***	***false***	-
S	Q1.0	Saída 1.0	false	***true***	***false***	***false***

O estágio 2 (Tabela 19) teve o fluxo de energia bloqueado, pois nessa circunstância a variável I1.1 foi ativada (valor final igual a *true*) levando o estado do elemento NF a *false*, o que provocou o desligamento da bobina (estado *false*).

Tabela 20- Diagrama Ladder no estágio final e tabelas de comunicação.

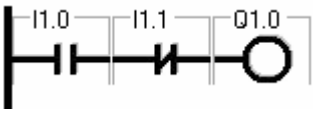


Diagrama Ladder (estágio final)

Tabela de Leitura					**Estado do Elemento**	**Tabela de Escrita**
I/O	***Nome***	***Comentário***	***Valor Inicial***	***Valor Final***		
E	I1.0	Entrada 1.0	false	***false***	***false***	-
E	I1.1	Entrada 1.1	false	***true***	false	-
S	Q1.0	Saída 1.0	false	***false***	false	false

Nesse último estágio da simulação, o elemento NA retomou seu estado original (*false*) com a desabilitação da variável I1.0 e o elemento NF permaneceu com seu estado em *false*, uma vez que o valor de sua variável foi mantido em *true*. Sem alimentação, a bobina permaneceu desligada (estado do elemento igual a *false*).

VIII.2 Simulação com maquete

Uma lógica foi implementada e testada apenas com o simulador do ambiente de programação e com o modo manual da maquete sob comandos direcionais isolados, pois o módulo de comunicação entre o ambiente de programação e a maquete virtual ainda passava por alguns ajustes e intervenções no padrão de reconhecimento e interpretação dos dados trocados entre os dois *softwares*. Isso fazia com que a resposta da maquete aos comandos da lógica de controle fosse comprometida. Portanto testou-se com sucesso o vínculo com a maquete, onde se confirmou um *link* estabelecido e transferência de alguns dados, suficiente para avaliação de resultados dessa primeira fase do projeto e analisou-se a aplicação, que deveria ser posta em prática na maquete, avaliando o comportamento da lógica via simulador. A nova versão do ambiente de programação, junto ao módulo de comunicação concluído, irá possibilitar a implementação completa do exemplo desenvolvido.

A aplicação elaborada para ilustração é bem simples. Criada para a maquete virtual de uma mesa de processamento de peças da Festo (Figura 31) . Foi desenvolvido um programa de controle em Diagrama Ladder para comandar as operações.

Figura 31 – Ilustração da maquete virtual da mesa da FESTO.

A mesa consiste de uma base giratória indexada de quatro posições, ferramentas de processamento de peças, sensores e botões de comando. Cada posição representa uma estação de processamento (E1 a E4) onde uma ferramenta trabalha a peça ou um robô a manipula.

O cenário onde a mesa é disposta pode ser escolhido entre diversos ambientes, conforme o tema da aplicação. Como essa simulação foi realizada apenas para efeito de teste e para ilustrar o projeto, escolheu-se um

fundo aleatório, como pode ser visto na Figura 31 que tem como *background* um oceano.

A presença de peça nas estações é informada por sensor, ativado (estado de ativação igual a *true*) quando detecta um objeto e o posicionamento das ferramentas também é informado por sensores.

Como agentes desse processo, tem-se:

- Botão de start (BS);
- Motor da base (MB);
- Acionamento de avanço de ferramenta (A1, A2, A3, A4);
- Acionamento de recuo de ferramenta (R1, R2, R3, R4);
- Sensores de presença de peça (SP1, SP2, SP3, SP4);
- Sensores de ferramenta em cima (SC1, SC2, SC3, SC4);
- Sensores de ferramenta em baixo (SB1, SB2, SB3, SB4);

Supondo a seguinte condição inicial:

- Base parada (indexada)

  Motor da base: MB = *false*

- Ferramentas recuadas (estado de repouso)

  Sensores em baixo: SB1 = SB2 = SB3 = SB4 = *false*

  Sensores em cima: SC1 = SC2 = SC3 = SC4 = *true*

  Avanço de ferramenta: A1 = A2 = A3 = A4 = *false*

  Recuo de ferramenta: R1 = R2 = R3 = R4 = *false*

- Peça posicionada na primeira estação (E1)

  Sensores de peça: SP1 = *true*

  SP2 = SP3 = SP4 = *false*

A aplicação consiste em:

- Confirmar presença de peça na estação (SP1);
- Acionar avanço de ferramenta (A1 = *true*);
- Verificar sensor em baixo (SB1);
- Parar ferramenta (A1 = *false*);

- Contar tempo de processo

  Tempo de Processo: TP1 = TP2 = TP3 = TP4 = 3 segundos

  Tempo de Processo concluído: TPok1, TPok2, TPok3, TPok4

- Acionar recuo de ferramenta (R1 = *true*);
- Verificar sensor em cima (SC1);
- Parar ferramenta (R1 = *false*);
- Acionar giro da base (MB = *true*);
- Repetir os procedimentos anteriores para as próximas estações.
- Considerar troca de peça a cada ciclo completo (quatro estações)

Foi testada a lógica para uma estação (um ciclo de procedimentos) com auxílio do simulador de *I/O*, que atendeu bem ao teste, já que o objetivo desse simulador é justamente testar pequenas lógicas ou trechos curtos de programas. O número de pontos de *I/O* disponíveis no simulador foi suficiente e a dinâmica de acionamentos foi bem sucedida. As variáveis da aplicação foram relacionadas às do simulador, como mostra a Tabela 21. Nessa tabela pode ser vista a correspondência entre as variáveis disponíveis no simulador (primeira coluna) e as que foram definidas na aplicação (segunda coluna).

Tabela 21 – Relação de variáveis para uso do simulador.

ENTRADAS			**SAÍDAS**		
simulador	**aplicação**		**simulador**	**aplicação**	
DI0	BS	Botão de start	DO0	MB	Motor da base
DI1	SP1	Sensor de presença de peça na estação 1	DO1	A1	Avanço de ferramenta
DI2	SC1	Sensor de ferramenta em cima	DO2	R1	Recuo de ferramenta
DI3	SB1	Sensor de ferramenta em baixo	DO3	TPok1	Tempo de Processo concluído
DI4	SP2	Sensor de presença de peça na estação 2			
AI0	TP1	tempo de processo			

Há diferentes formas de se programar um mesmo controle. Como demonstração disso, foram elaboradas duas lógicas diferentes para essa aplicação, com intuito de apresentar duas técnicas possíveis e distintas para se obter um mesmo resultado. Os Diagramas Ladder usados nos testes estão na Figura 32.

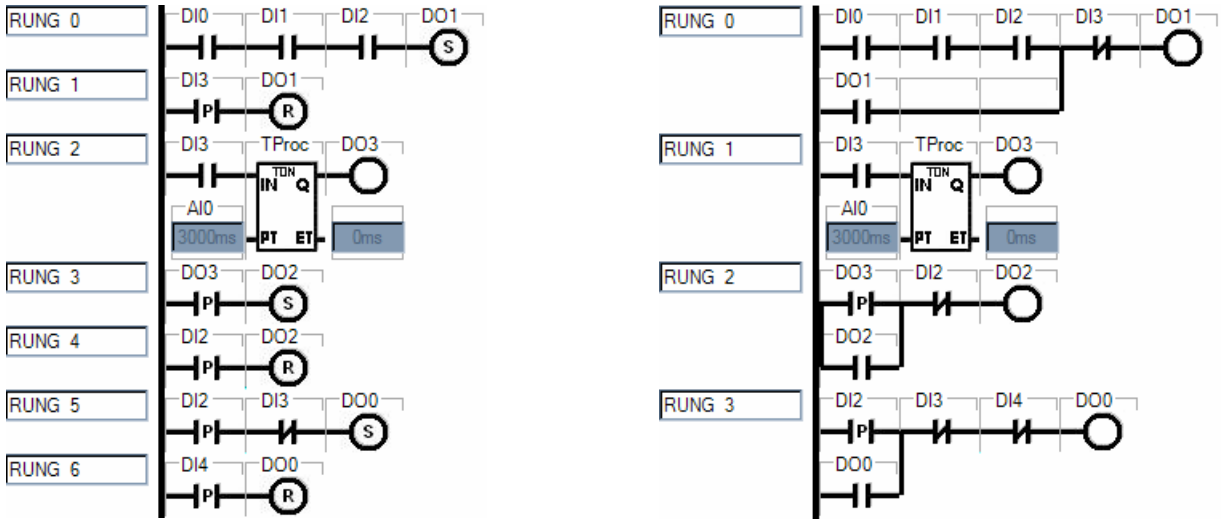


Figura 32 – Diagramas Ladder para uma estação usando o simulador

(a)com bobinas retentivas (b)com selo

Testada a lógica para uma estação, basta repeti-la para as demais estações e substituir as variáveis para associar o Diagrama Ladder à maquete da mesa. A análise dos diagramas ladder usados nesse teste encontram-se no APÊNDICE D.

Nota: Nessa aplicação não foram consideradas condições de falhas e ignorou-se qualquer interrupção de funcionamento por acionamento de emergência.

IX CONCLUSÃO

O mercado industrial está cada vez mais exigente. Não basta satisfazer o cliente, é preciso surpreendê-lo. Com isso a tecnologia empenhada nas atividades dos profissionais da indústria, estimula o aperfeiçoamento das técnicas, aprimora ferramentas e impõe qualidade.

Um sistema de manufatura virtual é de suma importância para auxiliar as atividades dos especialistas tanto da produção quanto da gestão empresarial. É um forte colaborador no meio administrativo. Significa um ganho substancial na formalização de propostas e aprovação de projetos, uma vez que projetistas e gerentes não estarão discutindo idéias calçadas somente em embasamentos teóricos. No meio acadêmico é instrumento perfeito de formação e qualificação, pois pode retratar o panorama da indústria ao trazer a realidade dos processos de fabricação para dentro das instituições.

É um recurso que permite ilustrar projetos, simular situações, inventar, criar, reprojetar, e com isso levar a uma análise da aplicação, desde seu funcionamento até a viabilidade de implementação, o que implica em criar uma idéia mais palpável de possibilidades e limitações, além de uma visão mais ampla de economia, qualidade, eficiência, metodologia e soluções inovadoras. Para empresas, pode culminar inclusive em reconhecimento no mercado.

Pode ser destinado tanto a aplicações de caráter acadêmico para formação e treinamento de profissionais, quanto ao aprimoramento de projetos de manufatura, ao planejamento da produção, ou mesmo à implantação de processos, treinando operadores e pessoal de manutenção.

Além de versátil, o *software* precisa ser flexível, não se prender muito ao meio. Da forma como está sendo desenvolvido, o Sistema de Manufatura Virtual apresenta grande flexibilidade de arquitetura. Permite integração entre múltiplas plataformas em se tratando de um sistema distribuído.

Software dessa natureza, para fins genéricos de simulação, é importante que seja aberto, padronizado, independente de fabricante ou equipamento, como o projeto citado nesse trabalho.

O APIMV, como parte integrante desse projeto maior, que é o Sistema de Manufatura Virtual, foi desenvolvido diante de muita análise e avaliação de diversos aplicativos para programação de controle, como o Isagraf [24], PCWorx [20] e outros. Suas características foram surgindo contrastantes às limitações apresentadas por esses *softwares* similares a ele, e desvinculado da arquitetura de *hardware*, podendo ser instalado em qualquer plataforma. Sua varredura de execução é rápida, empregando um algoritmo eficaz. E possui um módulo simulador de entradas e saídas, além de se comunicar com uma maquete virtual.

Esse projeto abriu frente para outros trabalhos e novas pesquisas. Já está sendo implementado um outro módulo integrante do sistema de manufatura virtual para compor mais uma opção de linguagem industrial a ser disponibilizada pelo ambiente de programação. Esse *software* deve acrescentar no sistema a linguagem SFC (*Sequential Function Chart*) e deve ser facilmente integrado ao ambiente de programação devido à forma como foi concebido. Assim, muito em breve o sistema disponibilizará duas linguagens de programação. Em andamento, estará sendo providenciada a implantação das demais linguagens.

A idéia fundamental de se desenvolver um sistema desse porte é deter a tecnologia dessas ferramentas e incentivar novas pesquisas, criações e inovações tecnológicas, tendo em vista ser este um trabalho inovador para o mercado.

Como trabalho futuro, poderia ser aderido ao ambiente de programação um aplicativo para aferição e diagnóstico interativo para elaboração de relatórios, descrevendo os pontos críticos e sensíveis do processo simulado, apresentando as possíveis causas e sujestões para soluções. Isso deve facilitar ainda mais o trabalho dos projetistas e analistas, pois seria um recurso inteligente dando apoio às decisões nas modificações e melhorias do sistema. Outra idéia é a extensão desse projeto para estabelecer comunicação com o meio externo (dispositivos de *hardware*), que implicaria o desenvolvimento de drivers para reconhecimento dos equipamentos e uma evolução da metodologia usada no presente projeto.

X REFERÊNCIAS BIBLIOGRÁFICAS

[1] International Electrotechnical Commission – IEC. Programmable Controllers – Programming Languages. Final Draft – IEC 61131-3*, 2nd Ed., 2001.*

[2] Miyagi, P.E.. Controle Programável – Fundamentos do Controle de Sistemas a Eventos Discretos, Editora Edgard Blücher LTDA, 1996.

[3] Honório, L.M., Souza, L.E., Rocha, L.C., Desenvolvimento e Simulação de Dispositivo Robótico Virtual, Congresso Nacional de Tecnologia da Informação e Comunicação, Florianópolis, SUCESU 2004.

[4] Heng, P.A., Cheng, C.Y., Wong, T.T., Xu, Y., Chui, Y.P., Chan, K.M., Tso, S.K., IEEE – A Virtual-Reality Training System for Knee Arthroscopic Surgery, IEEE Transactions on Information Technology in Biomedicine, vol. 8, No. 2, 2004.

[5] Grant, H., Lai, C.K., Simulation Modeling with Artificial Reality Technology (Samrt): An Integration of Cirtual Reality and Simulation Modeling, Winter Simulation Conference, ed. Medeiros, D.J., Watson, E.F., Carson, J.S., Manivannan, M.S.,1998.

[6] Willis, P., Virtual Physics for Virtual Reality, Theoty and Pratice of Computer Graphics, IEEE, 2004

[7] Bogdan, S., Kovacic, Z., Rocak, N.S., Birgmajer, B., A Matrix Approach to an FMS Control Design – Virtual Modeling to a Pratical Implementation, IEEE Robotics & Automation Magazine, 2004.

[8] Festo Didactic GmbH & Co., COSIMIR Professional, www.festo-didactic.com,2006.

[9] Lima, Edwin. "C# .net - Guia do Desenvolvedor", Campus, 2004.

[10] Ralha, C.P.S., Segredos do Visual Studio .Net, Digerati Books, 1 ed, 2004.

[11] Settimi, M.M, Toledo, L.F., Paparelli, R., Martins, M., Souza, I.M., Silva, J.A.P., Lesões por Esforços Repetitivos (LER) e Distúrbios OsteoMusculares Relacionados ao Trabalho (DORT), Centro de Estudos em Saúde e Trabalho – CEST, 2000.

[12] Oliveira, J.C, Yu, S.J., Georganas, N.D., Synchronized World Embedding in Virtual Environments, IEEE Computer Graphics and Applications, 2004.

[13] Johnsson,A., Wall, J., Broman, G., A virtual machine concept for real-time simulation of machine tool dynamics, School of Engineering, Blekinge Institute of Technology, 2005.

[14] Gutiérrez,M., Vexo, F., Thalmann, D., The Mobile Animator: Interactive Character Animation in Collaborative Virtual Environments, Virtual Reality Lab, Swiss Federal Institute of Technology Lausanne, IEEE Virtual Reality, 2004.

[15] Bogdan, S., Lewis, F.L., Kovaèiæ, Z., Gürel, A., Stajdohar, M., New Matrix Formulation for Supervisory Controller Design in Practical Flexible Manufacturing System, IEEE, 1999.

[16] Kovaeic, Z., Bogdan, S., Smolicroeak, N., Birgmajer, B., Teaching Flexible Manufacturing Systems by Using Design and Simulation Program Tools, University of Zagreb, Faculty of Electrical Engineering and Computing Unska 3, IEEE, 2003.

[17] Mayr, H., Virtual Automation Environments - Design, Modeling,Visualization, Simulation, Marcel Dcliker, 2002.

[18] Wrigh, J.R., SWOT Analysis of COSIMIR® Software, Rixan Associates, Department of Industry & Technology, Millersville University of Pennsylvania, Janeiro/2005

[19] Combs, V., Product Manager, BNET Think Tank, Case studies, webcasts and resources, Louisville, Kentucky, http://industries.bnet.com, 2006.

[20] Phoenix Contact, IBS PC WORX 2.0 - Automation Software for INTERBUS Configuration, Parameterization and Diagnostics and IEC 61131-3 Programming, Data Sheet 5955A, www.phoenixcontact.com.br, 2006.

[21] Braga, C.M.P., Norma IEC 1131-3 - Padronização em Programação de Controle Industrial, DELT / UFMG, 2005.

[22] Siemens Automation and Drives, STEP 7 Professional - Programming and configuring according to IEC 61131-3, Siemens AG, www.automation.siemens.com/simatic/industriesoftware, 2006.

[23] Braga, C.M.P., Norma IEC 61131-3 para Programação de Controladores, CEAI/UFMG, 2004.

[24] ICS Triplex ISaGRAF Inc., General Specifications ISaGRAF (+Enhancements) – Installation and Licensing, Workbench and Control Run-Time, I/O Communications, OPC Data Access, Trending, Alarms and Events., ISaGRAF Product Specification, 2003.

[25] WEG Automação, PC12 Design Center, V2.0, www.weg.com.br, 2003.

[26]Schneider Electric, Telemecanique , Twido Soft, V3.1, www.schneider-electric.com.br, 2002.

[27] Silva, L.D., Perkusich, A., Uso de Realidade Virtual para Validação de Modelos de Sistemas Flexíveis de Manufatura, Departamento de Engenharia Elétrica, Universidade Federal de Campina Grande, VI Simpósio Brasileiro de Automação Inteligente, Bauru, 2003.

[28] Barbosa, A.L., SimuPLC - Simulador de Controlador Lógico Programável, Universidade Federal de Goiás, V3.2.0, www.andrebarbosa.eti.br, 2006.

[29] Dias, W., Ambiente de Simulação para Manufatura Virtual - Dissertação de Mestrado, Universidade Federal de Itajubá, 2006.

[30] Arantes, J.F.R., Dias, W., Souza, L.E., Honório, L.M., Ambiente de Programação e Integração para Manufatura Virtual - Artigo, VII SBAI/ II IEEE LARS, São Luís, 2005.

APÊNDICE A

CONFIGURAÇÃO DOS ELEMENTOS DE PROGRAMAÇÃO

Tabela 22 – Parâmetros que definem os elementos de programação.

Tipo	Parâmetro		Definição
string	label		nome da variável
	categoria		categoria do elemento(CONTATO/BOBINA/BLOCO/FIO_H/FIO_V)
	tipo		tipo do elemento: CONTATO: NA/NF/P/N BOBINA: X/NEG/P/N/S/R BLOCO: TEMPORIZADOR/CONTADOR
	atr		tipo de variável: I : Entrada O: Saída L: local interna COUNTER: Contagem TIMER: contagem de tempo
	td		tipo de dado (INT/REAL/TIMER/BOOL/BYTE/WORD)
	vi		valor inicial
	va		valor atual
	texto		comentários sobre a variável
	Figura		imagem .bmp associada
bool	estado		estado da variável (combinados: estado de ativação e alimentação)
	troca		detector de mudança de estado
point	pin	X	ponto de conexão à esquerda na horizontal
		Y	ponto de conexão à esquerda na vertical
	pout	X	ponto de conexão à direita na horizontal
		Y	ponto de conexão à direita na horizontal
int	id		index para busca na tabela correspondente
	x		coordenada de tela na horizontal
	y		coordenada de tela na vertical
GroupBox	gb	Name	nome da representação gráfica
		Text	texto da representação gráfica (aparece na tela)
		Width	largura da representação gráfica
		Height	altura da representação gráfica
		Left	posicionamento horizontal da representação gráfica
		Top	posicionamento vertical da representação gráfica

APÊNDICE B

COMPORTAMENTO DOS ELEMENTOS DE PROGRAMAÇÃO

Os elementos de programação do Diagrama Ladder são usados como chaves que estabelecem continuidade lógica nas linhas de comando, ou seja, exercem a tarefa de bloquear ou liberar o fluxo de energia ao longo de uma linha, como os relés eletromecânicos em circuitos de painéis elétricos nas indústrias, que ao terem suas bobinas energizadas liberam ou não a passagem de corrente elétrica. Alguns realizam operações e dependem da variação em suas entradas, como os temporizadores e contadores.

O comportamento de cada elemento é representado nesse apêndice através de curvas que variam no tempo num período aleatório. O estado das variáveis digitais (vide Tabela 6) é representado por dois níveis na curva:

- Baixo: que indica ausência de habilitação; também entendido como *false* (condição falsa) ou, na simbologia matemática, 0 (zero);
- Alto: que indica presença de habilitação; também entendido como *true* (condição verdadeira) ou, na simbologia matemática, 1 (um);

Por convenção, serão usados os termos: *edge* positivo e *edge* negativo, para representar a transição de nível baixo para alto (0 para 1) e de nível alto para baixo (1 para 0) respectivamente.

Variáveis analógicas inspiram valores dentro de uma determinada faixa e são representadas por diversos níveis ao longo do tempo, que podem ser ilustrados na curva como degraus ou rampa.

Para analisar o comportamento dos elementos do tipo contato, são necessárias três curvas: de alimentação, de ativação da variável e do estado assumido pelo elemento ao combinar as duas curvas anteriores. Na análise da bobina, é preciso somente duas curvas: de alimentação (P) e de resposta do elemento (estado assumido - R). Para os blocos de funções há necessidade de se analisar a curva característica de cada porta de entrada e de saída, isso varia de acordo com a categoria; por exemplo: o comportamento do temporizador TON (Tabela 23) pode ser compreendido através da combinação de três curvas, ao passo que o funcionamento do contador CTU (Tabela 24) requer quatro.

A seguir são apresentadas as análises gráficas do funcionamento de cada elemento disponível nessa primeira fase do Ambiente de Programação e Integração para Manufatura Virtual.

CONTATOS

Nota: Identificação dos pontos de análise dos contatos:
P_{j-1} = alimentação do elemento (energia à esquerda);
S_j = estado de ativação da variável associada;
R_j = estado assumido pelo elemento (liberação/bloqueio de energia).

As formas de onda representadas a seguir (Figura 33 à Figura 36), ilustram a resposta (R_j) de cada tipo de contato ao estímulo (P_{j-1} ou alimentação) combinado com o estado de ativação (S_j) do ponto associado. Nos gráficos são destacados os períodos em que a resposta permanece ativa, isto é, os trechos de curvas em que o estado de ativação do elemento encontra-se em nível alto (1), para que seja evidente sob quais condições o elemento de programação estará habilitado.

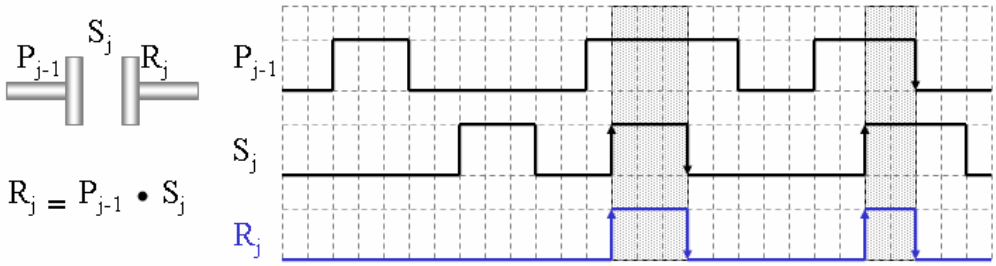


Figura 33- Resposta da Instrução Contato NA.

A Figura 33 apresenta as curvas de comportamento de um contato normalmente aberto (NA). Esse tipo de contato tem seu estado de ativação (R_j) naturalmente em 0 (nível baixo). O estado do elemento só é alterado para 1 (nível alto) na ocasião em que tanto a alimentação (P_{j-1}) quanto o estado da variável (S_j) apresentam-se em 1. Se pelo menos uma dessas condições não for mais satisfeita, o estado de ativação do elemento também volta a 0.

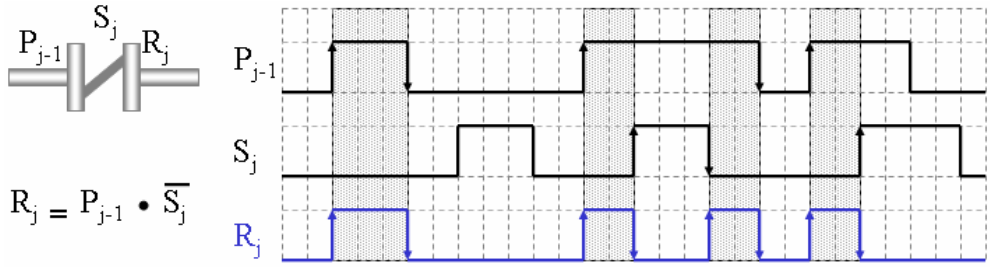


Figura 34- Resposta da Instrução Contato NF.

A Figura 34 apresenta as curvas de comportamento de um contato normalmente fechado (NF). Esse tipo de contato tem seu estado de ativação (R_j) naturalmente em 1 (nível alto), ou seja, quando a variável está inativa (estado igual a 0) o estado do contato é igual a 1, ou seja, seu estado de ativação é contrário ao estado de ativação da variável (S_j) associada a ele. Seu estado só é alterado para 0 (nível baixo) na ocasião em que há alimentação (P_{j-1}) e a variável apresenta-se em 1. Se pelo menos uma dessas condições não for satisfeita o estado de ativação do elemento volta a 1.

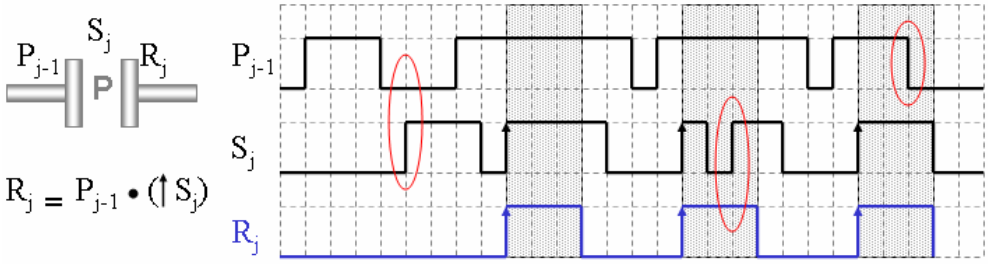


Figura 35 - Resposta da Instrução Contato P.

Na Figura 35 são apresentadas as curvas de comportamento de um contato tipo P, que capta *edge* positivo no estado da variável associada (S_j). Observando o gráfico, percebe-se que, na presença de alimentação (P_{j-1}), quando ocorre a transição de nível baixo para alto no estado de ativação da variável o elemento é habilitado mantendo seu estado (R_j) ativo por um tempo determinado (pulso). O estado do elemento, quando assume nível alto, permanece nessa condição mesmo que haja transição em P de nível alto para baixo ou na ocorrência de variação no estado da variável. A transição positiva do estado da variável é ignorada quando não há alimentação.

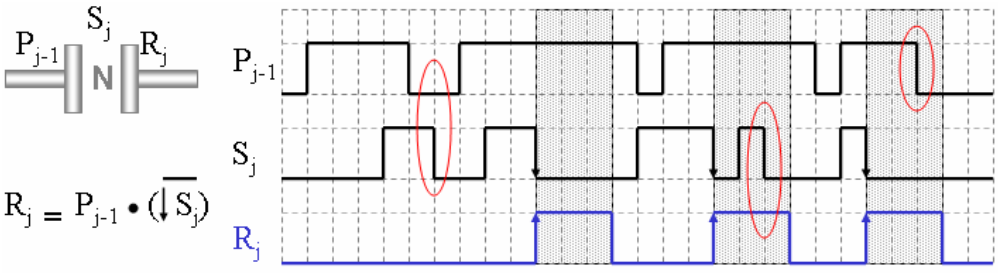


Figura 36 - Resposta da Instrução Contato N.

Na Figura 36 são apresentadas as curvas de comportamento de um contato tipo N, que capta *edge* negativo no estado da variável associada (S_j). Seu funcionamento é análogo ao do contato de tipo P, difere apenas por reconhecer transição de nível alto para nível baixo, enquanto o P reconhece transição de nível baixo para alto.

BOBINAS

Nota: Identificação dos pontos de análise das bobinas:
P_{j-1} = alimentação do elemento (energia à esquerda);
Q = estado assumido pelo elemento (libera/bloqueia fluxo de energia);
ativa estado da variável associada;

As formas de onda representadas a seguir (Figura 37 à Figura 41), ilustram a resposta (Q) de cada instrução do tipo bobina ao estímulo (P_{j-1} ou alimentação). Nos gráficos são destacados os períodos em que elemento permanece ativo, isto é, os trechos de curvas em que o estado de ativação mantém-se em nível alto (1), para que seja evidente sob quais condições o elemento de programação estará habilitado e acionará a variável associada.

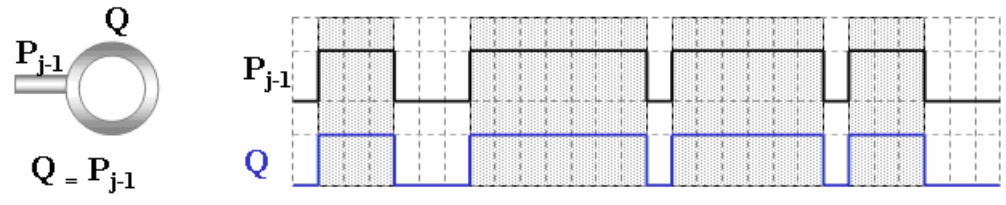


Figura 37 – Resposta da Instrução Bobina.

A Figura 37 mostra as curvas de comportamento de uma bobina comum, cujo acionamento (Q) acompanha as variações da alimentação (P_{j-1}), isto é, o elemento terá estado de ativação em nível alto quando houver energia chegando até ele.

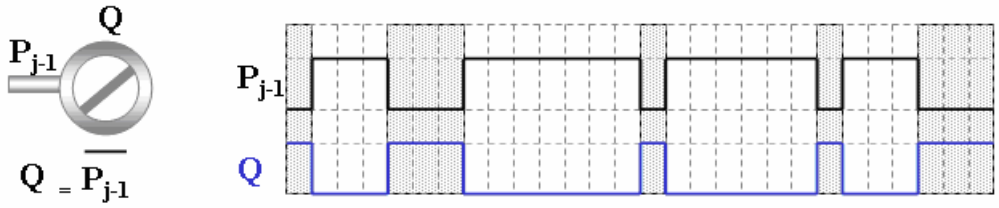


Figura 38- Resposta da Instrução Bobina Inversa (NEG).

A Figura 38 mostra as curvas de comportamento de uma bobina inversa, cujo acionamento (Q) é contrário às variações da alimentação (P_{j-1}), isto é, o elemento terá estado de ativação em nível alto quando não houver energia chegando até ele.

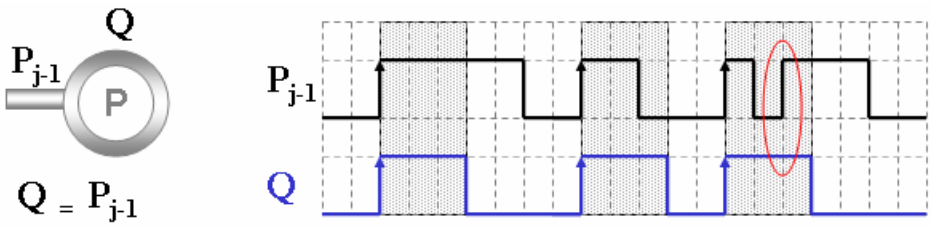


Figura 39 - Resposta da Instrução Bobina P.

Na Figura 39 podem ser conferidas as curvas de comportamento de uma bobina do tipo P, que capta *edge* positivo na alimentação (P_{j-1}). Observando o gráfico, percebe-se que quando ocorre a transição de nível baixo para alto na alimentação, o elemento é habilitado mantendo seu estado (Q) ativo por um tempo determinado (pulso). O estado do elemento quando assume nível alto, permanece nessa condição mesmo na ocorrência de variação da alimentação, mantendo acionada a variável associada.

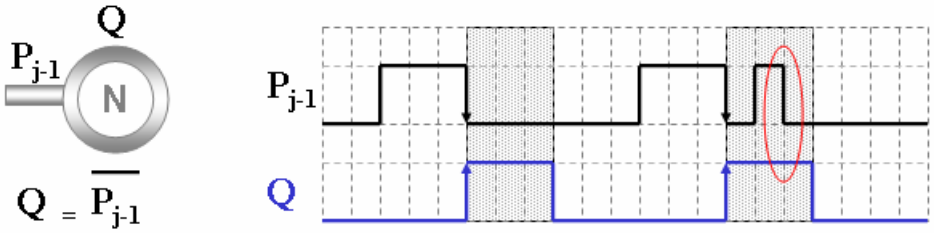


Figura 40 - Resposta da Instrução Bobina N

Na Figura 40 são apresentadas as curvas de comportamento de uma bobina do tipo N, que capta edge negativo na alimentação (P_{j-1}). Seu funcionamento é análogo ao da bobina de tipo P, difere apenas por reconhecer transição de nível alto para nível baixo, ao contrário dele.

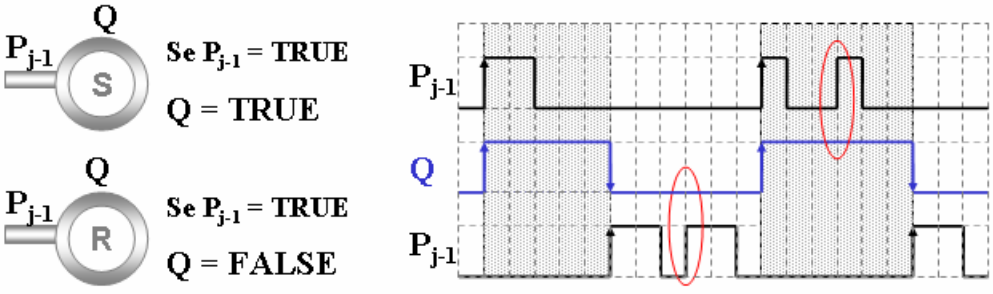


Figura 41 - Resposta das Instruções Bobina S e Bobina R.

A Figura 41 mostra as curvas de comportamento de dois tipos de bobinas: *set* (S) e *reset* (R), ambas retentivas e associadas a uma mesma variável. Como pode ser conferido nas curvas, o estado dos elementos é alterado na transição positiva da alimentação (P_{j-1}) e é mantido. São elementos complementares, isto é, o funcionamento de S depende do funcionamento de R e vice-versa.

A bobina S assume nível alto quanto ocorre um *edge* positivo na sua alimentação e mantém esse estado até que a bobina R seja acionada. A bobina R sendo acionada, ou seja, na ocorrência de *edge* positivo na sua alimentação (P_{j-1}), o estado assumido por ela e pela bobina S é nível baixo desativando a variável associada. O estado resultante da bobina R é mantido até que o estado do elemento complementar S seja acionado, levando o estado de ambas as bobinas (S e R) para nível alto.

Após a mudança de estado, transições na alimentação da bobina que causou a troca de nível são ignoradas. No caso da bobina S, quando uma transição positiva (nível baixo pra alto) na sua alimentação provoca estado igual a 1, levando a bobina R a assumir o mesmo estado, as variações de alimentação não terão efeito. O estado de ambas só será alterado para 0 quando a alimentação da bobina R sofrer uma transição positiva.

BLOCOS DE FUNÇÃO

As tabelas seguintes (Tabela 23 à Tabela 26), descrevem com formas de onda o comportamento de cada tipo de bloco implementado nesse aplicativo. Dois campos iniciais das tabelas definem como identificar o elemento: o primeiro campo (á esquerda) apresenta a imagem gráfica do elemento, com indicação do tipo de dado associado a cada porta de entrada ou saída do bloco, e o segundo (á direita) identifica os tipos de operações disponíveis para a categoria.

A indicação de instantes (t_N) facilita a correspondência entre as curvas, permitindo associar o comportamento do bloco e sua resposta no momento de cada ocorrência de evento (mudança de estado). Essa relação entre os sinais é o que conduz a analogia do funcionamento de cada bloco de função.

Nos gráficos, as áreas destacadas realçam os períodos em que o bloco encontra-se habilitado, isto é, apto a realizar a operação de sua competência. Por exemplo, no caso dos comparadores, enquanto há alimentação em sua entrada IN a comparação é realizada e a saída Q acionada sempre que a condição for verdadeira (vide Tabela 25).

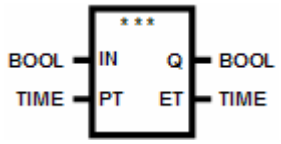
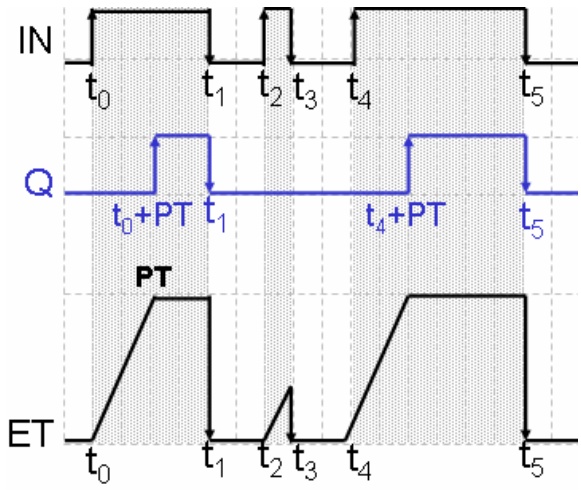
A amplitude das curvas varia conforme sua finalidade. Em caso de representação de variáveis digitais, o nível tem dois patamares: 0 e 1, representando o valores booleanos (*false* = nível baixo e *true* = nível alto, respectivamente). Em se tratando de outros tipos, a curva poderá exibir rampas (curvas ET na Tabela 23), simbolizando um incremento contínuo de valores, ou patamares (curvas CV na Tabela 24 indicando valores assumidos.

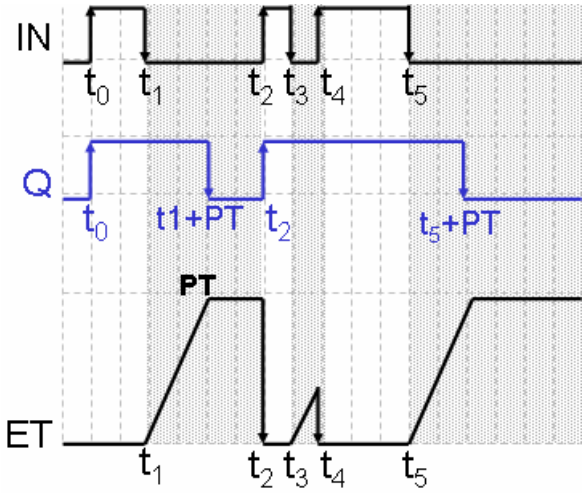
A nomenclatura das portas de entrada e saída dos blocos é diferente para cada tipo de função, identificando o dado associado a cada porta conforme operação que executam, mas todos acionam uma saída digital de nome Q (saída que libera a passagem de energia). A curva Q mostra a resposta do bloco na porta de saída em função da alimentação e da operação que rege seu comportamento, ou seja, a variação na saída Q representa o bloqueio ou liberação do fluxo de energia através do bloco estabelecendo ou não continuidade lógica na linha de comando. Portanto a identificação desse ponto de análise é:

- Q = estado assumido pelo elemento (libera/bloqueia fluxo de energia) que ativa o estado da variável associada.

Os demais pontos de análise têm sua identificação junto à explanação da tabela de cada categoria.

Tabela 23 – Resposta dos Temporizadores TON e TOF.

<table>
<tr><th colspan="2">TEMPORIZADORES</th></tr>
<tr><th>Forma Gráfica</th><th>Descrição</th></tr>
<tr><td> 
</td><td>*** deve ser substituído por:
• TON (On-delay)
• TOF (Off-delay)</td></tr>
<tr><th colspan="2">Formas de Onda</th></tr>
<tr><td> 

TON</td><td> 

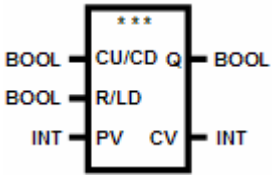
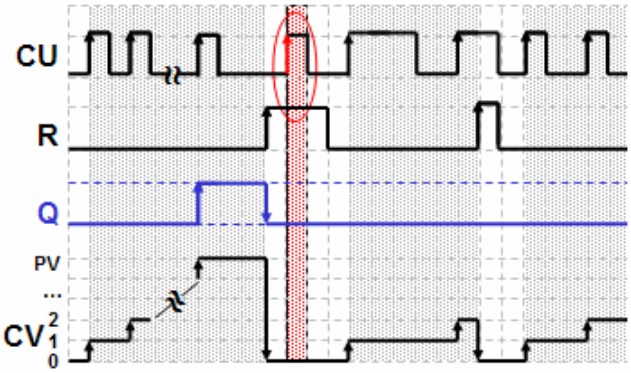
TOF</td></tr>
</table>

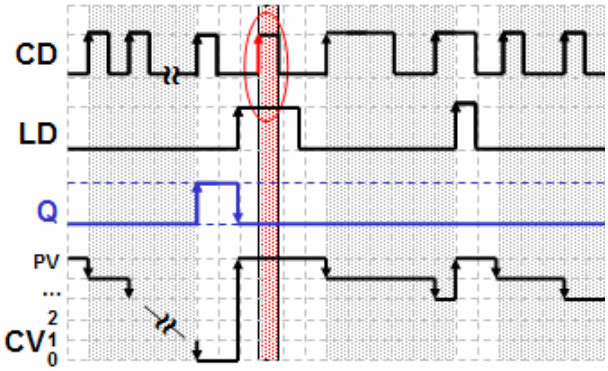
Na Tabela 23, a curva IN representa variações na porta de entrada que habilita o bloco, enquanto a curva ET mostra a contagem de tempo com incremento contínuo de zero até o valor de parada, que pode ser o limite estipulado por PT ou qualquer outro valor menor que ele quando ocorrer interrupção da contagem.

Observando o gráfico do TON, nota-se que no momento em que ocorre a transição de 0 para 1 (*edge* positivo) na entrada IN, o contador interno do bloco é disparado. No final da contagem, quando atinge o tempo máximo especificado na entrada PT, a saída Q é acionada (ocorre uma transição de 0 para 1). O contador mantém o valor máximo até que haja uma transição na entrada IN de 1 para 0 desabilitando o bloco, instante em que a saída Q retorna a 0 e o contador de tempo também é zerado. Se houver variação na alimentação durante a contagem de tempo, o temporizador é interrompido e retorna a zero.

Ao fazer a comparação entre as curvas do TOF, nota-se que no momento em que ocorre a transição de 0 para 1 (*edge* positivo) na entrada IN a saída Q é acionada (ocorre uma transição de 0 para 1). O contador de tempo só é disparado quando a alimentação retorna a 0 e ao terminar a contagem desabilita a saída Q. Se houver variação na alimentação durante a contagem de tempo, o temporizador é interrompido e retorna a zero.

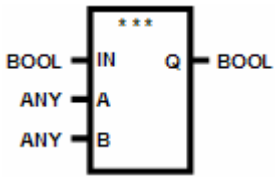
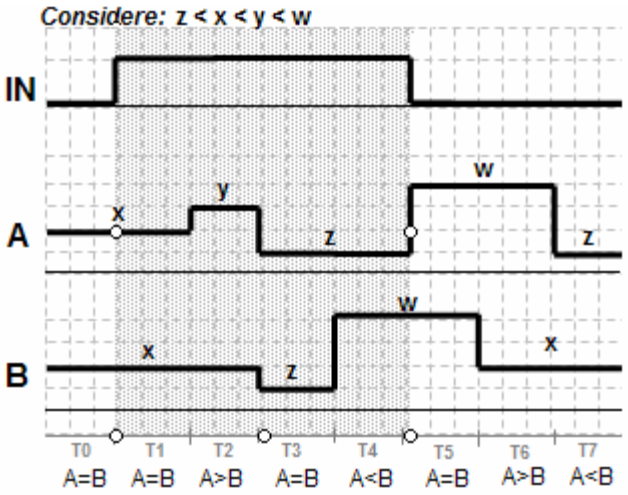
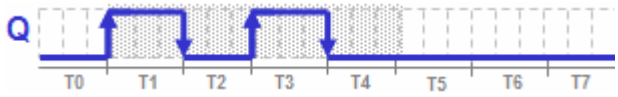
Tabela 24 – Comportamento dos Contadores CTU e CTD.

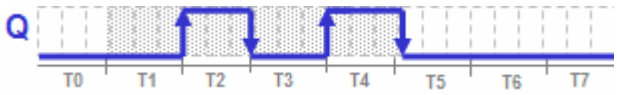
<table>
<tr><th colspan="2">CONTADORES</th></tr>
<tr><th>Forma Gráfica</th><th>Descrição</th></tr>
<tr><td> 
</td><td>*** deve ser substituído por:
CTU (contador up)
CTD (contador down)</td></tr>
<tr><th colspan="2">Formas de Onda</th></tr>
<tr><td> 

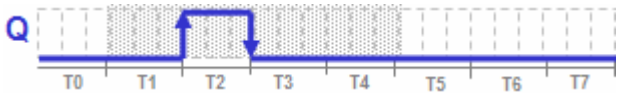
CTU</td><td> 

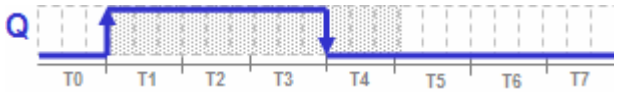
CTD</td></tr>
</table>

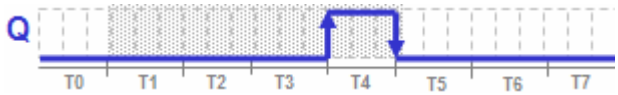
Tabela 25 - Comportamento dos Comparadores.

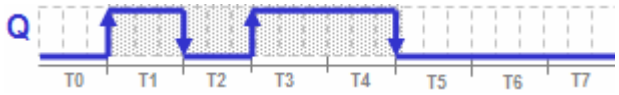
<table>
<tr><th colspan="2">COMPARADORES</th></tr>
<tr><th>Forma Gráfica</th><th>Descrição</th></tr>
<tr><td> 
</td><td>*** deve ser substituído por:
EQ (A igual a B)
NE (A diferente de B)
GT (A maior que B)
GE (A maior ou igual a B)
LT (A menor que B)
LE (A menor ou igual a B)</td></tr>
<tr><th colspan="2">Formas de Onda</th></tr>
<tr><td colspan="2"> 
</td></tr>
<tr><td> 

EQ</td><td> 

NE</td></tr>
<tr><td> 

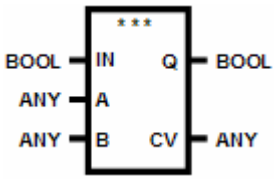
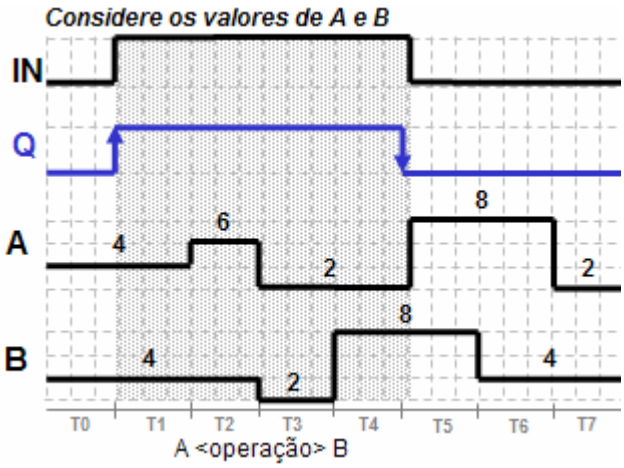
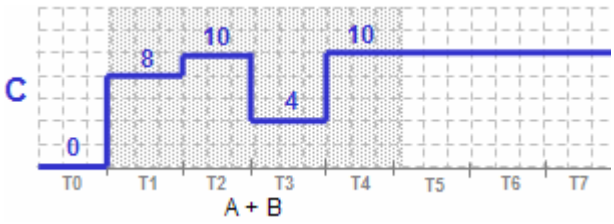
GT</td><td> 

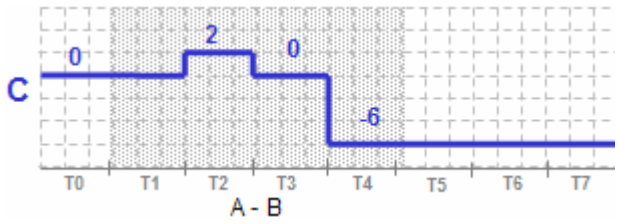
GE</td></tr>
<tr><td> 

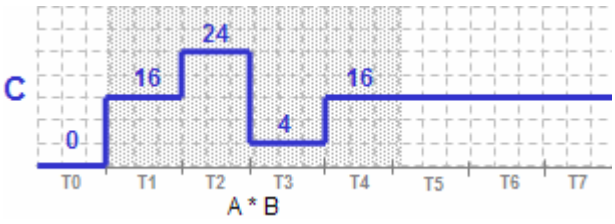
LT</td><td> 

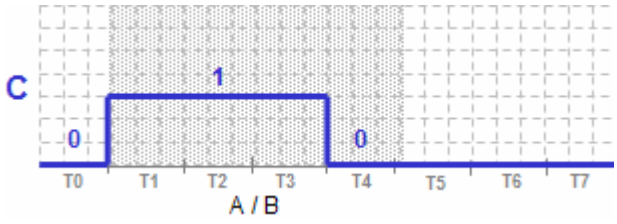
LE</td></tr>
</table>

Tabela 26 – Comportamento dos Operadores Aritméticos

<table>
<tr><th colspan="2">OPERADORES ARITMÉTICOS</th></tr>
<tr><th>Forma Gráfica</th><th>Descrição</th></tr>
<tr><td> 
</td><td>*** deve ser substituído por:
ADD
SUB
MUL
DIV</td></tr>
<tr><th colspan="2">Formas de Onda</th></tr>
<tr><td colspan="2"> 
</td></tr>
<tr><td> 

ADD</td><td> 

SUB</td></tr>
<tr><td> 

MUL</td><td> 

DIV</td></tr>
</table>

APÊNDICE C

SIMULAÇÕES E RESULTADOS

Nota: o estado de ativação de um elemento nas simulações desse apêndice usam representações nas formas: booleana, com TRUE ou FALSE e gráfica, com as cores azul ou branca. Assim, quando o elemento estiver ativo (estado=TRUE) assumirá cor azul e no simulador sua variável será ticada, caso contrário (estado=FALSE) sua cor será branca e não haverá marcação na variável.

Para aferição do funcionamento dos elementos de programação foram testadas lógicas básicas, bastante simples, de modo que ficasse evidente o comportamento de cada componente na linha de comando. A seguir são apresentados alguns testes realizados.

Simulação 1 - Ligação série

Diagrama:

Na Figura 42 é apresentada uma *rung* composta por: um contato normalmente aberto (DI0), um contato normalmente fechado (DI1) e uma bobina comum (DO0), dispostos numa ligação em série.

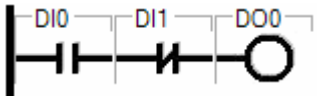


Figura 42 – Diagrama Ladder da Simulação 1.

Pontos *I/O* do diagrama:

A Tabela 27 mostra a listagem dos pontos de entrada e saída usados do simulador indicando os tipos de elemento aos quais foram associados.

Tabela 27 – Pontos de I/O da Simulação 1.

Entrada (E) / Saída (S)	***Variável***	***Comentário***
E	DI0	Entrada associada a contato NA
E	DI1	Entrada associada a contato NF
S	DO0	Saída associada a bobina comum

Estado Inicial:

Na Tabela 28 é relatado o estado inicial da *rung* e na Figura 43 é apresentada a condição gráfica da linha de comando nessa situação.

Tabela 28 - Estado Inicial da Simulação 1.

Variável	***Valor Inicial***	***Definição***
DI0	false	Entrada desabilitada
DI1	false	Entrada desabilitada
DO0	false	Saída desabilitada

Figura 43 – Estado inicial do programa da Simulação 1.

Acionamentos:

A Tabela 29 mostra a seqüência de acionamentos nas variáveis de entrada (DI0 e DI1) e a resposta da lógica na variável de saída (DO0).

Tabela 29 – Descrição e resultados dos acionamentos da Simulação 1.

Estado Anterior	***Estado Seguinte***	***Variável***	***Valor Inicial***	***Valor Final***	***Resultados***
Estado Inicial	Estado 1	DI0	false	true	Entrada habilitada
		DI1	false	false	Entrada desabilitada
		DO0	false	true	Saída acionada
Estado 1	Estado 2	DI0	true	true	Entrada desabilitada
		DI1	false	true	Entrada desabilitada
		DO0	true	false	Saída acionada
Estado 2	Estado 3	DI0	false	false	Entrada desabilitada
		DI1	false	true	Entrada habilitada
		DO0	true	false	Saída desacionada

A Figura 44 representa de forma gráfica o ciclo de acionamentos conforme a Tabela 29:

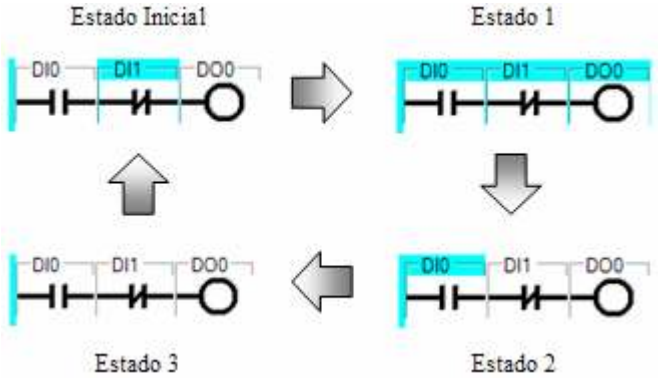


Figura 44 – Ciclo de acionamentos da Simulação 1.

Simulação 2 – Ligações série e paralela

Diagrama:

Na Figura 45 é apresentada uma *rung* composta por: um contato normalmente aberto (DI0) em paralelo com outro contato normalmente aberto (DO0), seguidos de um contato normalmente fechado (DI1) em série com uma bobina comum (DO0).

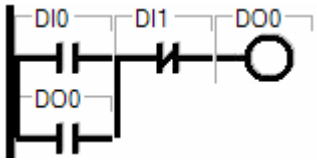


Figura 45 – Diagrama Ladder da Simulação 2.

Pontos *I/O* do diagrama:

A Tabela 30 mostra a listagem dos pontos de entrada e saída usados do simulador indicando os tipos de elemento aos quais foram associados.

Tabela 30 – Pontos de I/O da Simulação 2.

I/O	***Variável***	***Comentário***
E	DI0	Entrada associada a contato NA
E	DI1	Entrada associada a contato NF
S	DO0	Saída associada a contato NA
S	DO0	Saída associada a bobina comum

Estado Inicial:

Na Tabela 31 é relatado o estado inicial da *rung* e na Figura 46 é apresentada a condição gráfica da linha de comando nessa situação.

Tabela 31 - Estado Inicial da Simulação 2.

Variável	***Valor Inicial***	***Definição***
DI0	false	Entrada desabilitada
DI1	false	Entrada desabilitada
DO0	false	Saída desabilitada
DO0	false	Saída desabilitada

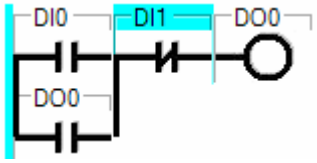


Figura 46 – Estado inicial do programa da Simulação 2.

Acionamentos:

A Tabela 32 mostra a seqüência de acionamentos nas variáveis de entrada (DI0 e DI1) e a resposta da lógica na variável de saída (DO0).

Tabela 32 – Descrição e resultados dos acionamentos da Simulação 2.

Estado Anterior	***Estado Seguinte***	***Variável***	***Valor Inicial***	***Valor Final***	***Resultados***
Estado Inicial	Estado 1	DI0	false	true	Entrada habilitada
		DI1	false	false	Entrada desabilitada
		DO0	false	true	Saída acionada
Estado 1	Estado 2	DI0	true	false	Entrada desabilitada
		DI1	false	false	Entrada desabilitada
		DO0	true	true	Saída acionada
Estado 2	Estado 3	DI0	false	false	Entrada desabilitada
		DI1	false	true	Entrada habilitada
		DO0	true	false	Saída desacionada

A Figura 47 representa de forma gráfica o ciclo de acionamentos conforme a Tabela 32.

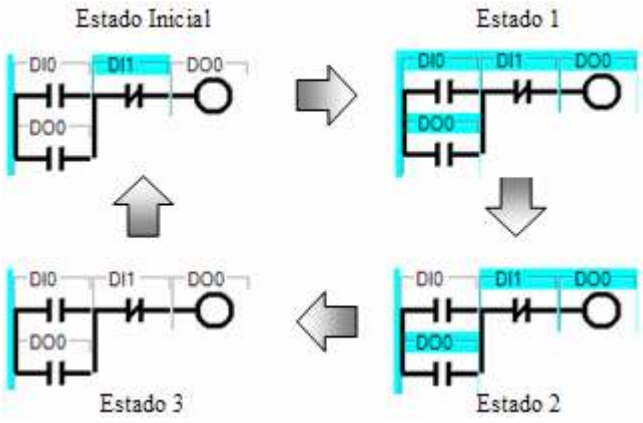


Figura 47 – Ciclo de acionamentos da da Simulação 2.

Simulação 3 – Temporizador TON

Diagrama:

Na Figura 48 é apresentada uma *rung* composta por: um contato normalmente aberto (DI0) em série com um temporizdor do tipo TON (TON1) seguido de uma bobina comum (DO0). O tempo limite do temporizador (PT) é 1s (mil milissegundos).

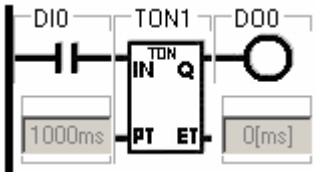


Figura 48 – Diagrama Ladder da Simulação 3.

Pontos *I/O* do diagrama:

A Tabela 33 mostra a listagem dos pontos de entrada e saída usados do simulador indicando os tipos de elemento aos quais foram associados.

Tabela 33 – Pontos de I/O da Simulação 3.

Entrada (E) / Saída (S)	***Variável***	***Comentário***
E	DI0	Entrada associada a contato NA
S	DO0	Saída associada a bobina comum

Estado Inicial:

Na Tabela 34 é relatado o estado inicial da *rung* e na Figura 55 é apresentada a condição gráfica da linha de comando nessa situação.

Tabela 34 - Estado Inicial da Simulação 3.

Variável	***Valor Inicial***	***Definição***
DI0	false	Entrada desabilitada
TON1	0 [ms]	Bloco desabilitado
DO0	false	Saída desabilitada

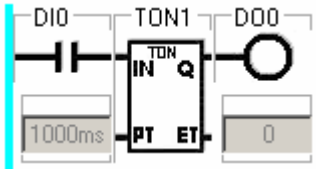


Figura 49 – Estado inicial do programa da Simulação 3.

Acionamentos:

A Tabela 35 mostra o acionamento da variável de entrada (DI0), o acionamento do temporizador e a resposta da lógica na variável de saída (DO0). Quando o tempo está sendo contado, o valor na tabela é apresentado como um valor (ET) entre 0 e o valor limite determinado por PT.

Tabela 35 – Descrição e resultados dos acionamentos da Simulação 3.

Estado Anterior	***Estado Seguinte***	***Variável***	***Valor Inicial***	***Valor Final***	***Resultados***
Estado Inicial	Estado 1	DI0	false	true	Entrada habilitada
		TON1	0 [ms]	0<ET<PT [ms]	Temporizador contando
		DO0	false	false	Saída desabilitada
Estado 1	Estado 2	DI0	true	true	Entrada desabilitada
		TON1	0<ET<PT [ms]	ET = PT	Fim da contagem de tempo
		DO0	false	true	Saída acionada

A Figura 50 representa de forma gráfica o ciclo de acionamentos conforme a Tabela 35.

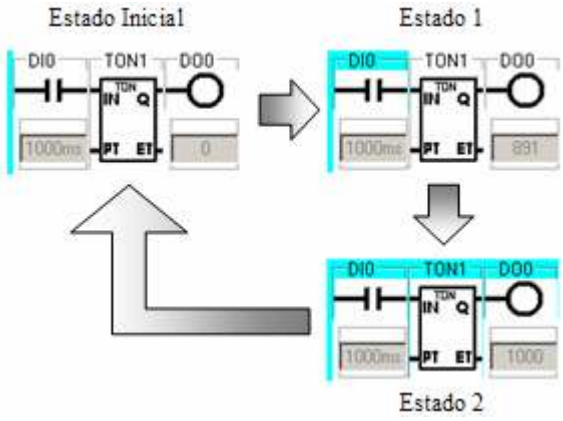


Figura 50 – Ciclo de acionamentos da Simulação 3.

Simulação 4 – Contador CTU

Diagrama:

Na Figura 51 é apresentada uma *rung* composta por: um contato normalmente aberto (DI0) acionando a entrada de pulsos do contador, um contato normalmente aberto (DI1) acionando a entrada de *reset* do bloco, uma variável analógica (AI0) na entrada PV definindo limite de contagem igual a 2 (dois) e uma bobina comum (DO1) sendo acionada pelo bloco.

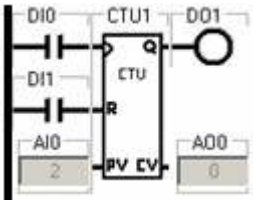


Figura 51 – Diagrama Ladder da Simulação 4.

Pontos *I/O* do diagrama:

A Tabela 36 mostra a listagem das variáveis envolvidas na simulação.

Tabela 36 – Pontos de I/O da Simulação 4.

Entrada (E) / Saída (S)	***Variável***	***Comentário***
E	DI0	Entrada associada a contato NA
E	DI1	Entrada associada a contato NA
E	AI0	Entrada associada a entrada de valor do bloco
E	AO0	Saída associada a saída de valor do bloco
S	DO1	Saída associada a bobina comum

Estado Inicial:

Na Tabela 37 é relatado o estado inicial da *rung* e na Figura 52 é apresentada a condição gráfica da linha de comando nessa situação.

Tabela 37 - Estado Inicial da Simulação 4.

Variável	***Valor Inicial***	***Definição***
DI0	false	Entrada desabilitada
DI1	false	Entrada desabilitada
AI0	2	Entrada com valor inicial
AO0	0	Saída com valor inicial
CTU1	0	Bloco desabilitado
DO1	false	Saída desabilitada

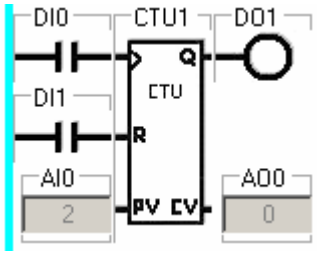


Figura 52 – Estado inicial do programa da Simulação 4.

Acionamentos:

A Tabela 38 mostra os acionamentos das variáveis, a contagem do bloco e o acionamento da saída. A variação de DI0 do estado inicial para o estado 1, e do estado 2 para o 3, causa incremento na contagem. Quando a contagem alcança o valor máximo previsto em PV no estado 3, a saída é acionada. No estado 5, o pulso em R (entrada de *reset* do bloco), faz com que o contador seja reinicializado (retorno a zero).

Tabela 38 – Descrição e resultados dos acionamentos da Simulação 4.

Estado Anterior	***Estado Seguinte***	***Variável***	***Valor Inicial***	***Valor Final***	***Resultados***
Estado Inicial	Estado 1	DI0	false	true	Entrada habilitada
		DI1	false	false	Entrada desabilitada
		AI0	2	2	Entrada com valor inicial
		AO0	0	1	Saída incrementada (CV<PV)
		DO1	false	false	Saída desabilitada
Estado 1	Estado 2	DI0	true	false	Entrada desabilitada
		DI1	false	false	Entrada desabilitada
		AI0	2	2	Entrada com valor inicial
		AO0	1	1	Saída mantida (CV<PV)
		DO1	false	false	Saída desabilitada
Estado 2	Estado 3	DI0	false	true	Entrada habilitada
		DI1	false	false	Entrada desabilitada
		AI0	2	2	Entrada com valor inicial
		AO0	1	2	Saída incrementada (CV=PV)
		DO1	false	true	Saída habilitada
Estado 3	Estado 4	DI0	true	false	Entrada desabilitada
		DI1	false	false	Entrada desabilitada
		AI0	2	2	Entrada com valor inicial
		AO0	2	2	Saída mantida (CV=PV)
		DO1	false	true	Saída habilitada
Estado 4	Estado 5	DI0	false	false	Entrada desabilitada
		DI1	false	true	Entrada habilitada
		AI0	2	2	Entrada com valor inicial
		AO0	2	0	Saída zerada (valor inicial)
		DO1	true	false	Saída habilitada

A Figura 53 representa de forma gráfica o ciclo de acionamentos conforme a Tabela 38.

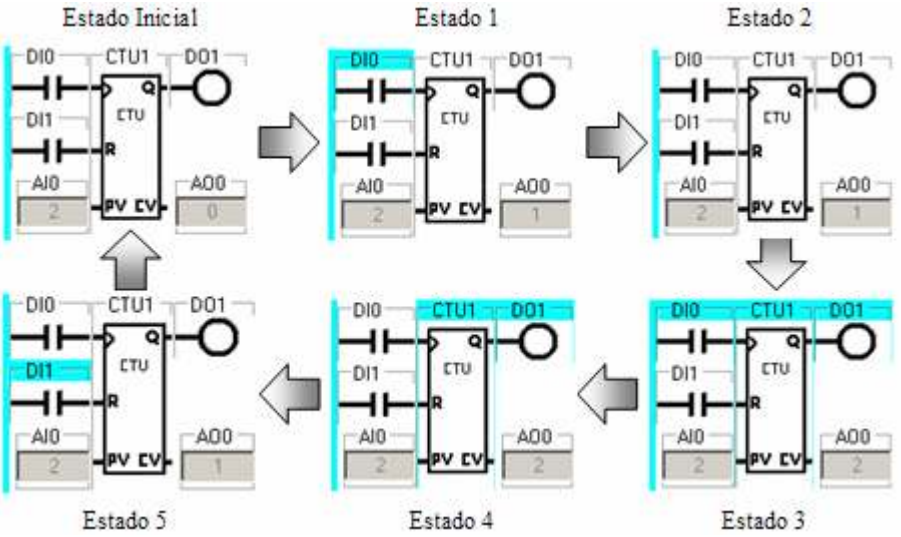


Figura 53 – Ciclo de acionamentos da Simulação 4.

APÊNDICE D

SIMULAÇÃO COM MAQUETE VIRTUAL

A aplicação usada como exemplo para ilustrar a comunicação entre o ambiente de programação e uma maquete virtual é o funcionamento de uma mesa da Festo, que processa peças em sua base giratória indexada para quatro estações de trabalho. A peça é posicionada na base onde permanece durante todo o ciclo do processo, isto é, a peça é fixa e a base gira para conduzi-la de uma estação para outra. Há quatro posições para fixar peças na superfície da base, de modo que podem ser processadas até quatro peças simultaneamente, uma em cada estação. A Figura 54 mostra uma imagem da maquete virtual que representa essa mesa.

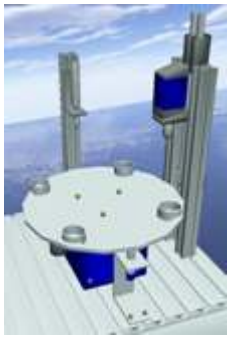


Figura 54 – Ilustração da mesa giratória e estações de processamento.

Para demonstrar a funcionalidade do projeto, foi considerada a operação de apenas uma estação da mesa, o que foi suficiente. Portanto, o programa de controle foi desenvolvido direcionado apenas para uma estação, em que uma ferramenta desce por uma haste até a base, processa a peça e retorna, liberando a peça para a próxima estação. O intertravamento entre as estações e o gerenciamento da aplicação com todos os seus aspectos de segurança não foram minuciados no exemplo. Os procedimentos abordados foram somente os básicos e consistem no seguinte:

- Se houver peça na estação (DI1=1) e a ferramenta estiver recuada (DI2=1), acionar avanço da ferramenta (DO1=1), mediante comando de início de processo através do botão de start (DI0 = *edge* positivo);
- Quando a ferramenta atingir seu fim de curso (DI3=1), desativar o avanço de ferramenta (DO1=0), e contar tempo de processamento da peça (AI0 = tempo máximo);
- No final da contagem de tempo (DO3=1), ativar o retorno da ferramenta (DO2=1);
- Quando a ferramenta retornar (DI2=1), desativar o recuo (DO2=0) e acionar o giro da base (DO0=1).
- Quando o index da base for concluído (DI4=1) o comando de giro é desativado (DO0=0).

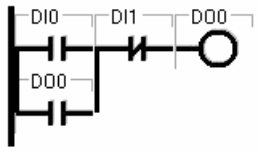
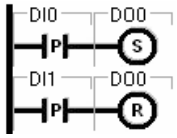
A seguir será apresentada a análise do programa de controle e os resultados da simulação.

ANÁLISE DA LÓGICA

O controle do funcionamento da maquete foi implementado de duas formas. O primeiro Diagrama Ladder usou bobinas retentivas e o segundo usou bobinas comuns. São técnicas distintas, mas que causam mesmo efeito lógico.

Como as bobinas retentivas mantêm o estado de ativação da saída, é preciso uma linha de comando para acioná-la e outra linha de comando para desativá-la. O mesmo efeito é obtido com bobinas comuns, utilizando-se um recurso de programação denominado Selo, que consiste numa maneira de se manter acionada uma saída usando um contato relacionado à própria bobina em paralelo com o(s) elemento(s) que a acionaram. A Tabela 39 mostra um exemplo de implementação identificando essas duas técnicas. Nessa tabela, note na figura (i) que a entrada DI0 aciona DO0, uma vez que DI1 esteja desativada mantendo a continuidade lógica da linha de comando, e o contato relacionado a DO0 em paralelo com DI0 mantém o acionamento. Portanto, assim como a bobina Set, apresentada na figura (ii), basta um pulso para que a saída seja habilitada. Analogamente acontece com o comando de desabilitar a saída. Ocorrendo uma transição na entrada DI1, a saída DO0 cai a zero em ambas as situações, pois na linha implementada com o selo, DI1 interrompe a continuidade lógica abrindo o contato normalmente fechado, enquanto que na situação (ii) a saída é desativada pela bobina *Reset* que força seu estado para nível baixo.

Tabela 39 – Exemplo de lógica ladder usando: (i) selo e (ii) retenção.

	
(i)	(ii)

Na Tabela 39 pode-se perceber que uma mesma saída pode ser acionada tanto por uma bobina simples quanto por bobinas retentivas (*Set* e *Reset*). Observe que nos dois casos são usados quatro elementos de programação e a mesma quantidade de linhas do diagrama. A atuação dos comandos é a mesma, ambos acionam a saída DO0 quando a entrada DI0 estiver ativa e a entrada DI1 estiver desabilitada, e desabilitam-na quando a entrada DI1 for acionada. Isso mostra que uma aplicação pode ser programada de diversas maneiras para executar a mesma operação sem que o resultado lógico seja modificado.

A seguir são demonstradas as duas simulações de entradas e saídas, realizadas para confirmação da lógica de controle, e mostradas as imagens ilustrativas da resposta da maquete aos comandos do diagrama quando vinculada ao ambiente de programação.

CONFIRMAÇÃO DA LÓGICA DE CONTROLE

Nota: o estado de ativação de um elemento é representado graficamente por cor, assim, quando o elemento estiver ativo (estado=1) assumirá cor azul e no simulador a variável correspondente será ticada, caso contrário (estado=0) sua cor será branca e não haverá marcação na variável.

O teste da programação foi realizado no simulador. Na Figura 55 são apresentados os dois diagramas que deveriam usados para testar a comunicação do ambiente de programação com a maquete virtual. Pode-se notar que o diagrama que usa bobinas retentivas é composto por sete linhas de comando (*rungs*) ao passo que o diagrama com bobinas comuns utiliza apenas quatro. Mas ambos empregam a mesma quantidade de elementos de programação (vinte) e mesmo efeito lógico, como pode ser confirmado pela tela do simulador nas figuras que mostram os resultados da simulação (Figura 55 à Figura 64).

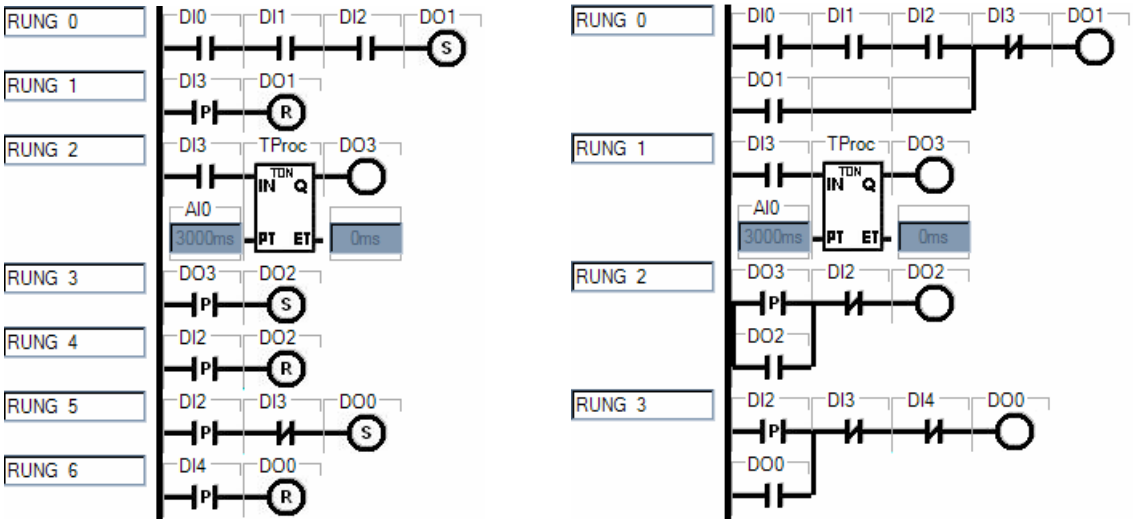


Figura 55 – Diagramas Ladder para mesa da Festo usando bobinas retentivas.

Observando a Figura 55, na primeira imagem tem-se o diagrama com bobinas retentivas. Nesse diagrama, a primeira linha de comando (*rung*) é responsável pelo procedimento inicial, em que a descida da ferramenta (DO1) é acionada; a segunda e a terceira são comandadas pelo fim de curso (DI3) da ferramenta, e atuam sobre a parada da ferramenta e sobre a contagem do tempo de processo respectivamente; a quarta e a quinta comandam a subida da ferramenta no término do processo e sua parada no ponto de repouso; a sexta *rung* aciona o giro da base e, por fim, a sétima pára a base posicionando a peça a próxima estação. Já no outro diagrama, a primeira linha de comando (*rung*) é responsável pelos dois primeiros procedimentos, em que a descida da ferramenta (DO1) é acionada e pára no fim de curso (DI3); a segunda aciona o tempo de processamento da peça; a terceira comanda a subida da ferramenta no término do processo e sua parada no ponto de repouso; e a quarta *rung* aciona o giro da base e sua parada na próxima estação.

Os estágios do teste usando o simulador são apresentados nas figuras seguintes (Figura 56 à Figura 64). Cada uma dessas figuras traz os dois diagramas e a tela do simulador caracterizando o instante atual do teste.

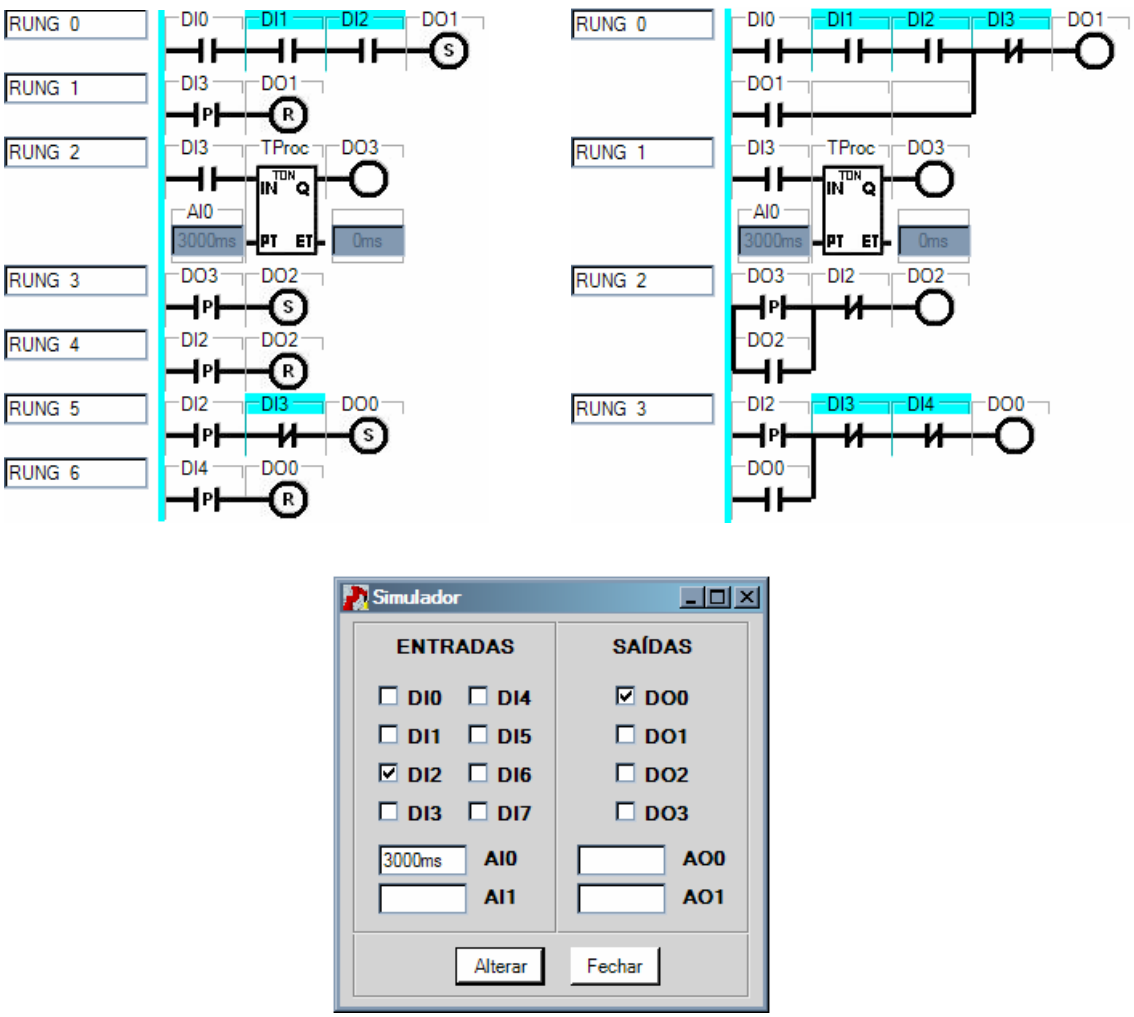


Figura 56 – Simulação - condição inicial.

A Figura 56 mostra a condição inicial do processo, situação em que o sistema está pronto para iniciar sua operação. Na tela do simulador podem ser vistas as entradas digitais DI1 e DI2 acionadas, indicando presença de peça na estação 1 e ferramenta recuada respectivamente, e a entrada analógica AI0, que é associada ao temporizador (na entrada PT do bloco), com valor 3000ms para determinar o tempo de trabalho da ferramenta.

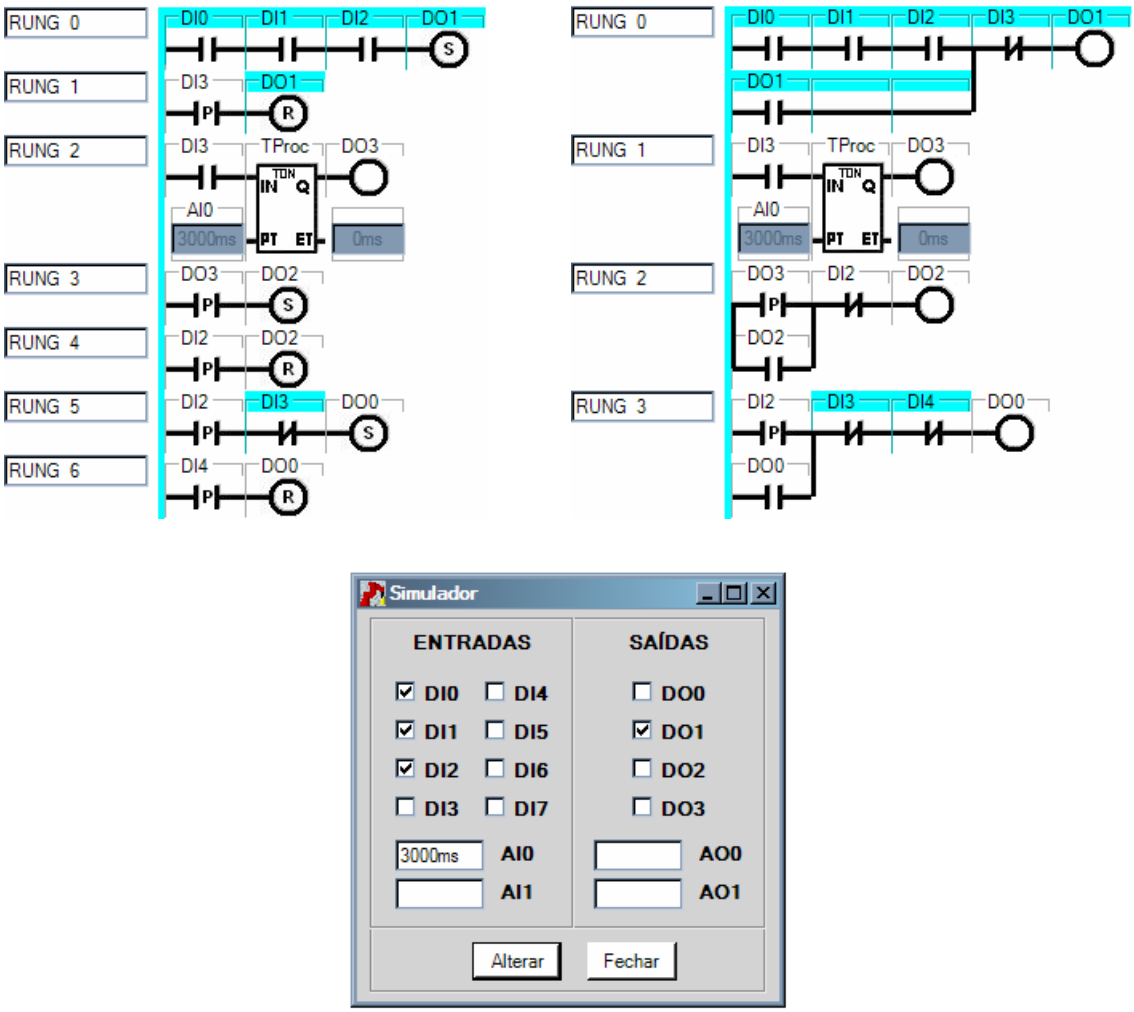


Figura 57 - Simulação – start do processo.

Na Figura 57 a entrada DI0 está ativa, o que estabelece continuidade lógica na primeira linha e dá início ao processo ativando a saída DO1.

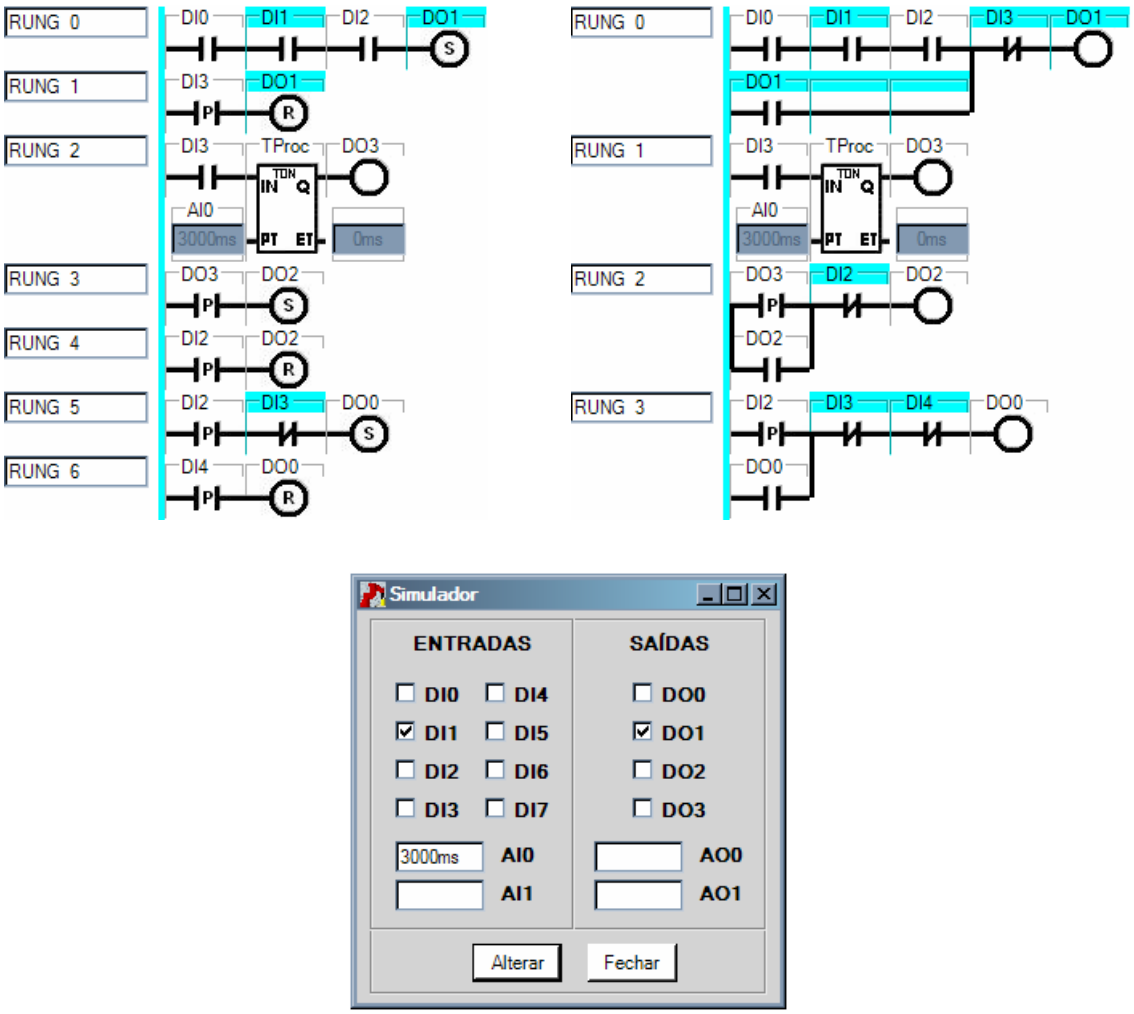


Figura 58 - Simulação – ferramenta descendo.

A Figura 58 mostra a entrada DI2 desabilitada, o que representa movimento de descida da ferramenta, visto que o comando de avanço está acionado.

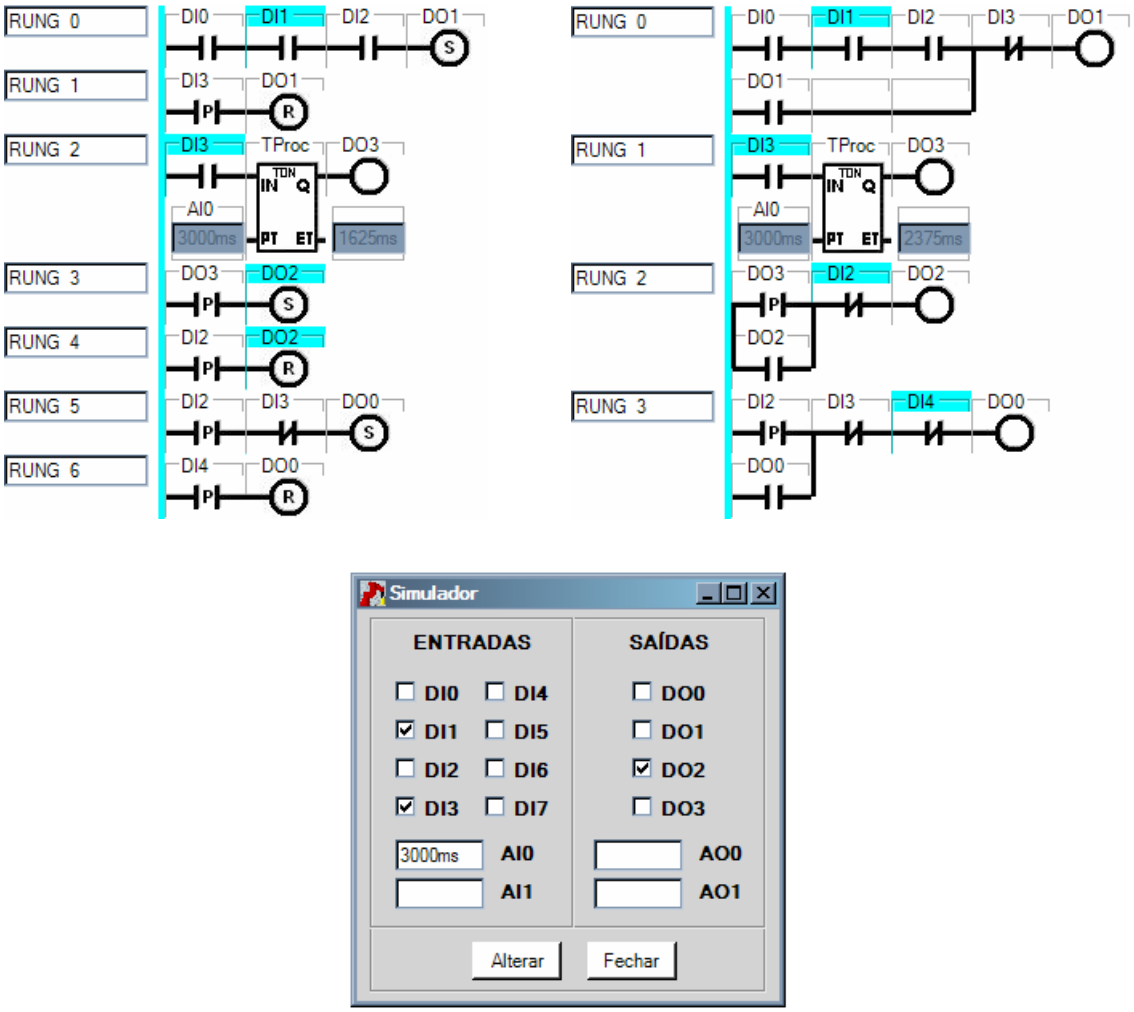


Figura 59 - Simulação – processando a peça.

A situação apresentada pela Figura 59 mostra a entrada DI3 ativa, simbolizando a ferramenta em baixo, e o temporizador contando o período de processamento da peça. Nesse caso não há variável associada à saída ET, ou seja, na tela do simulador não foi mostrado o incremento de tempo, o que pode ser acompanhado no próprio bloco. Observe que na saída ET do temporizador há um valor diferente de zero, o que significa que no momento da captura da imagem do diagrama, a contagem de tempo estava correndo.

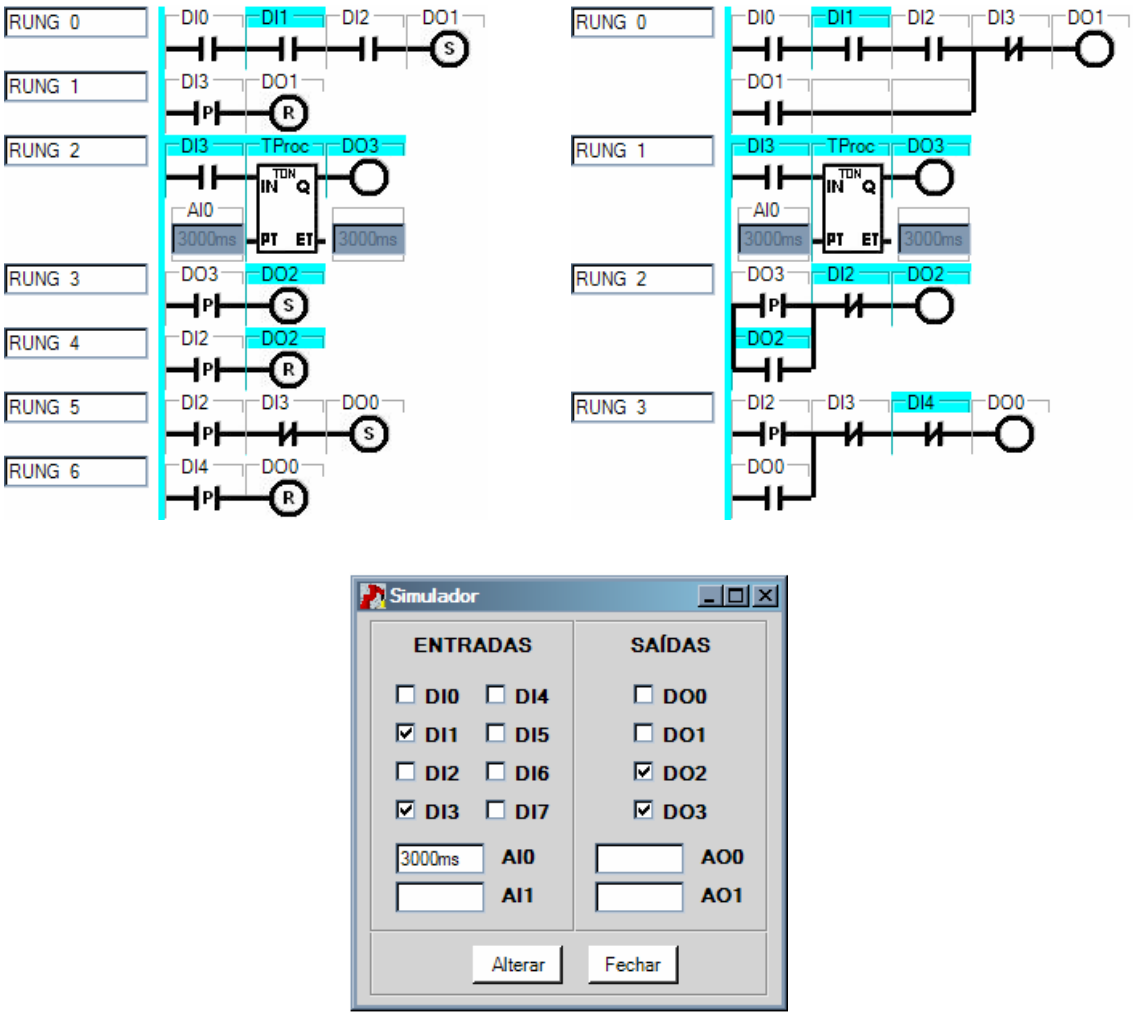


Figura 60 – Simulação – final de processamento da peça.

A Figura 60 representa a situação em que o temporizador finaliza sua contagem (DO3) e aciona a subida da ferramenta (DO2).

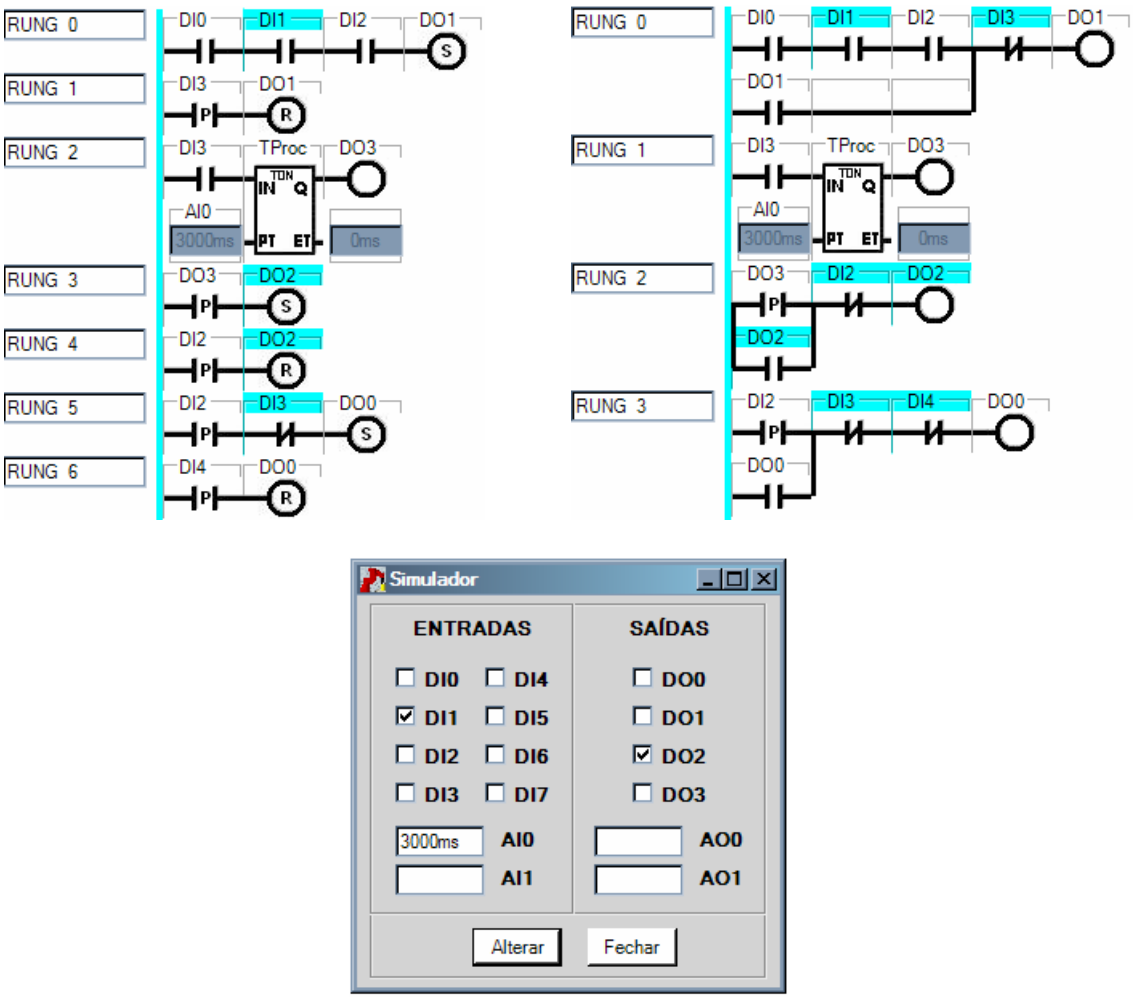


Figura 61 – Simulação – ferramenta subindo.

Na Figura 61 pode-se notar que a ferramenta está em movimento de subida, já que o comando de recuo está acionado e o sensor de baixo (DI3) não está mais ativo. Na ausência de DI3, o temporizador também volta a zero.

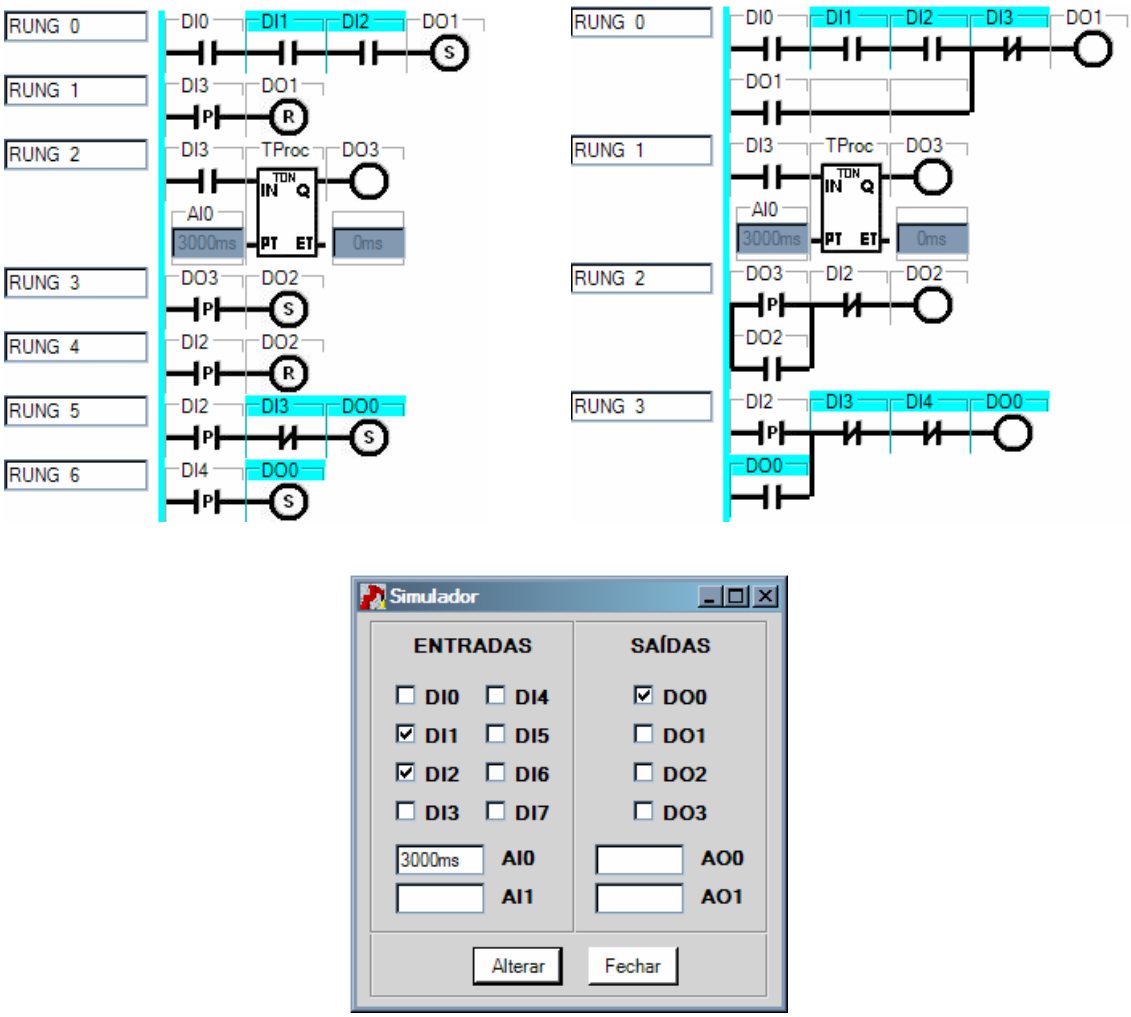


Figura 62 – Simulação – processo concluído.

A Figura 62 mostra o momento em que a ferramenta atinge o sensor de cima, desativando o comando de subida (DO2) e ativando o giro da base para conduzir a peça até a próxima estação.

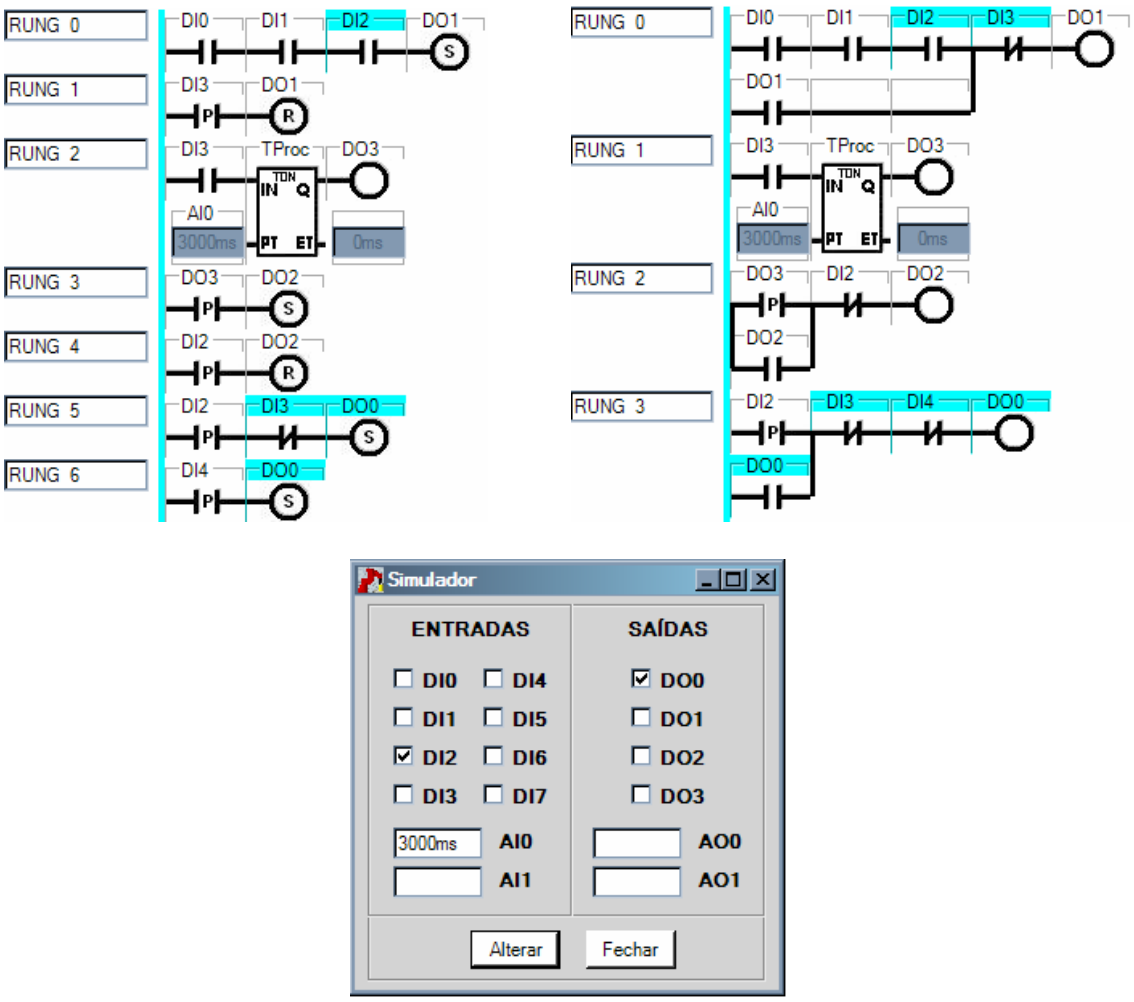


Figura 63 - Simulação – base girando.

Observe na Figura 63 que a peça não se encontra mais na primeira estação (DI1=0), pois está em movimento (DO0=1).

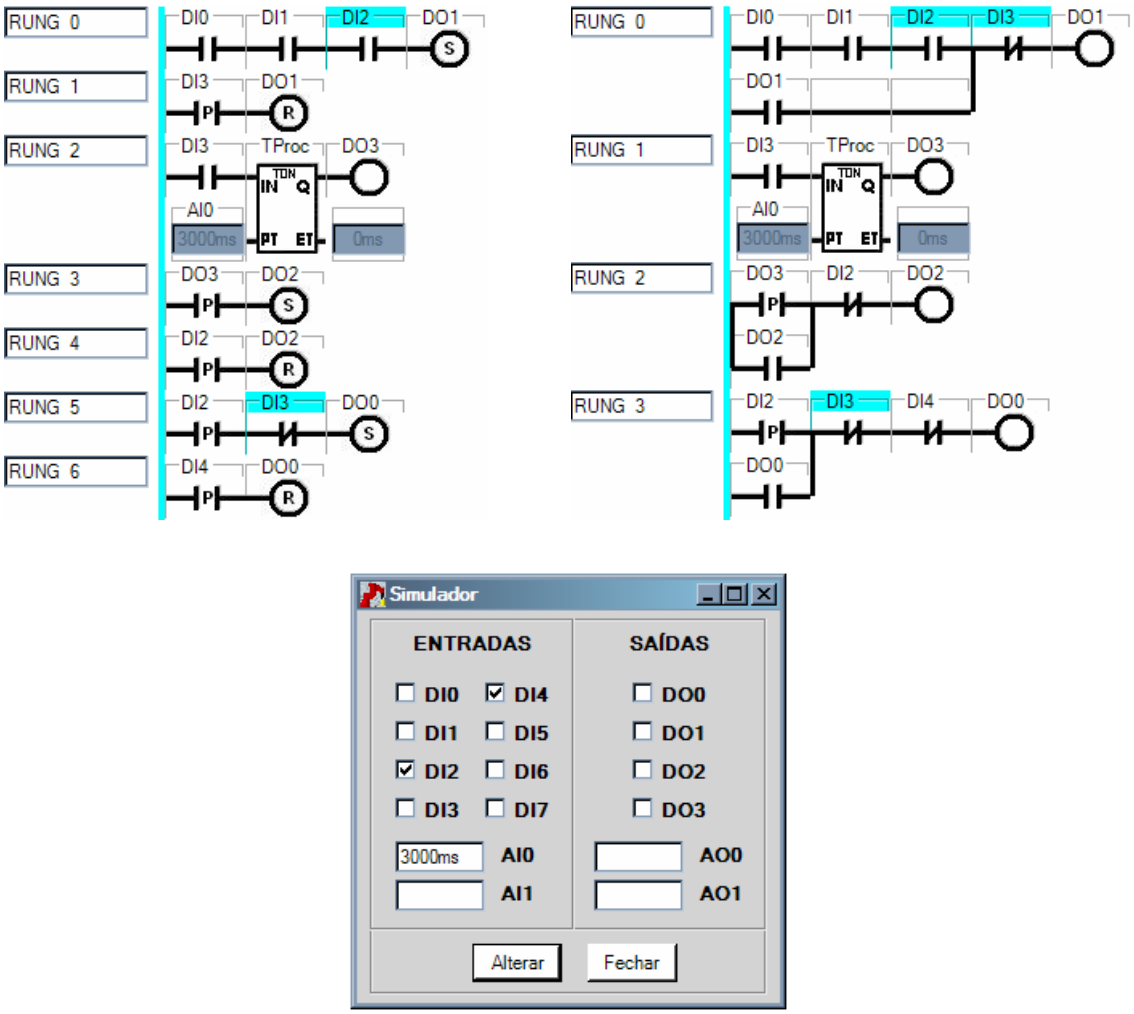


Figura 64 - Simulação – peça na próxima estação.

Por fim, a Figura 64 mostra a chegada da peça na próxima estação (DI4=1) e a parada da base (DO0=0).

RESPOSTA DA MAQUETE

Após confirmar a lógica de controle usando o simulador de entradas e saídas, caso fosse estabelecido o vínculo com a maquete, as imagens resultantes em resposta aos comandos do programa deveriam ser (Figura 65):

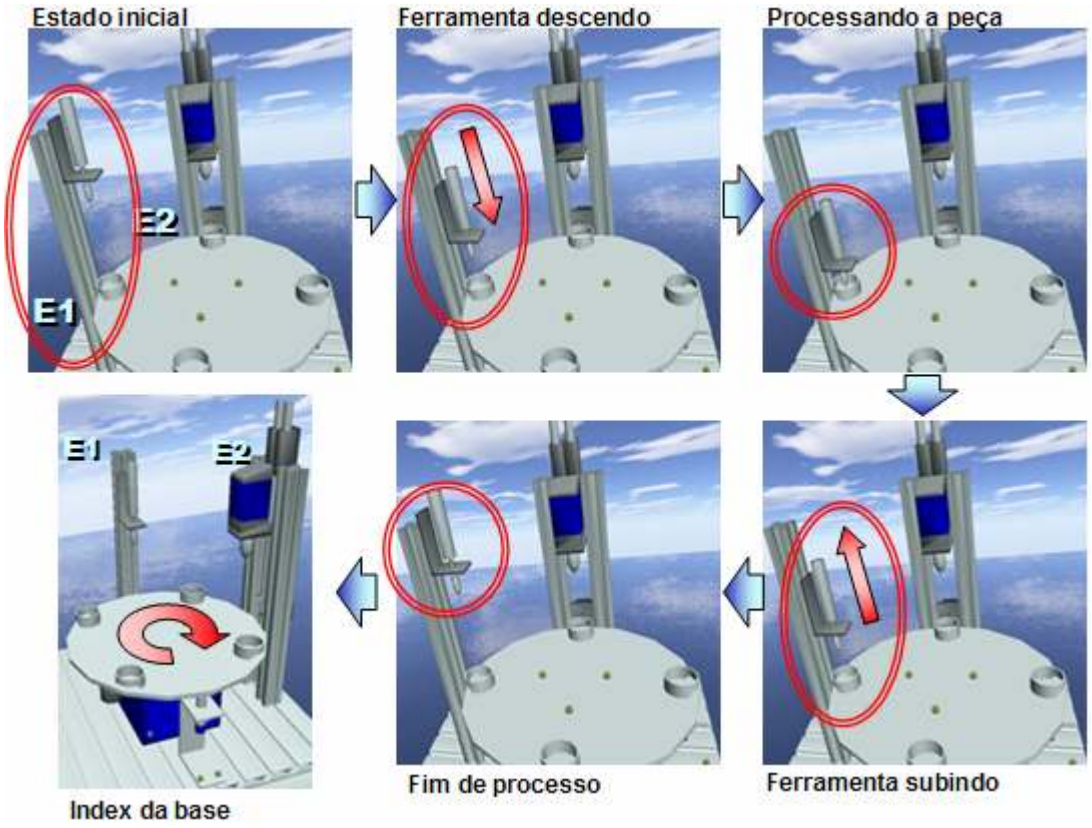


Figura 65 – Simulação – resposta da maquete.

Na Figura 65 pode ser vista a imagem da maquete em cada etapa da simulação. Representa um ciclo completo da estação usada nesse exemplo. A primeira imagem mostra a situação inicial do processo, quando a ferramenta encontra-se estacionada na parte superior da haste (na estação 1 – E1). Na imagem seguinte, a ferramenta é apresentada num instante do seu deslocamento durante a descida até a base. Na próxima imagem, a ferramenta está totalmente avançada para processar a peça. Em seguida, é apresentada a ferramenta retornando para o topo da haste. A imagem seguinte mostra a ferramenta novamente em situação de repouso, porém agora tendo concluído sua operação e liberando a peça para a estação posterior (E2). Finalmente, a última imagem mostra um instante do movimento rotativo da base conduzindo a peça até a próxima estação.

Nota: essas imagens foram geradas com operação manual da maquete.

APÊNDICE E

DESCRIÇÃO DO *SOFTWARE*

O software é composto por 10 classes e 14 forms. Suas principais partes encontram-se descritas nas tabelas a seguir (Tabela 40 à Tabela 56).

CLASSES

Para definir e identificar as classes, segue uma série de tabelas (Tabela 40 à Tabela 47) que apresentam a definição de cada classe com seus atributos e métodos, descrevendo cada um deles.

Tabela 40 – Descrição da classe: contatos.

contatos	
Determina os parâmetros que caracterizam o elemento de programação.	
Atributos :	• categoria: classificação do elemento (contato,bobina,bloco, fio) • tipo: tipo de elemento dentro da categoria (NA,R, TON, Fio_V, etc.) • pin: conexão à esquerda dentro do diagrama; • pout: conexão à direita dentro do diagrama; • estado: resultado da combinação entre alimentação e condição atual do elemento; • troca: flag de troca de valor na execução do programa; • x: posição horizontal na tela; • y: posição vertical na tela; • figura: imagem associada ao elemento conforme seu tipo; • gb: célula que conte a identificação gráfica do componente; • label: nome dado ao elemento; • atr: direção da atuação (I = entrada / O = saída / L = local); • id: index para busca na tabela correspondente; • td: tipo de dado (BOOL/ INT/TIMER/COUNTER); • vi: valor inicial; • va: valor atual; • texto:comentários;
Métodos:	• Elemento: define particularidades gráficas e físicas do elemento; • Var: define a identificação do elemento e valores.

Tabela 41 - Descrição da classe: comunicação.

comunicação	
Estabelece o vínculo entre o programa e a maquete virtual ou manipula arquivos padrões de simulação de *I/O*.	
Atributos :	• dadosmv: lê dados da maquete para distribuir nas tabelas; • sr: capta conteúdo do arquivo de entrada do simulador; • saídas: aponta para o arquivo de escrita para transmitir os comandos ao simulador; • Linha: capta um linha por vez para distribuir os dados nas tabelas; • entradas: tabela que armazena os valores de entrada do simulador; • estados: tabela que armazena os dados de saída a serem passados para a maquete; • ti: aponta para o arquivo de entrada do simulador para atualizar com os dados da tela.
Métodos:	• LeTabela: faz a leitura de estados da maquete ou do simulador; • GravaOUT: transmite os comandos para a maquete ou para o simulador; • GravaTI: atualiza arquivo de entrada do simulador.

Tabela 42 - Descrição da classe: compilação.

compilação	
Responsável pelo funcionamento do algoritmo e varredura de execução.	
Atributos :	• Fsim: chamada do simulador; • com: chamada da classe de comunicação; • grav: chamada da classe de gravação; • clock: gera pulsos em milissegundos;
Métodos:	• nlLista: retorna o número de linhas de uma ArrayList; • ncLista: retorna o número de colunas de uma ArrayList; • matrizP: monta a matriz P do algoritmo; • matrizS: monta a matriz S do algoritmo; • matrizLAB: monta a matriz Lab do algoritmo; • SimuladorAtualiza: atualiza arquivo de entrada com valores da janela de simulação; • Varredura: cria ciclo de execução do algoritmo; • CorOnline: atualiza cores dos elementos gráficos na execução do programa; • Temporizadores: encontra blocos temporizadores e os aciona para rodarem em paralelo; • ResetOut: reinicializa saídas; • AtualizaIN: atualiza valores de entrada na lista de elementos; • AtualizaOUT: atualiza valores de saída e variáveis internas na tabela de saída; • CorOriginal: Volta a cor normal dos elementos; • ReinicializaEl: atribui o valor inicial a todos os elementos de programação

Tabela 43 - Descrição da classe: gravação.

gravação	
Trabalha com leitura e escrita de arquivos	
Atributos:	• Edit: chamada da classe edição.
Métodos:	• Grava: realiza gravação do diagrama no local apontado pelo usuário; • Abre: resgata programa do local apontado pelo usuário;

Tabela 44 - Descrição da classe: g.

g	
Determina atributos gerais usados em todo o programa.	
Atributos:	• ex: execução; • ed: edição; • si: simulação; • ti: tabela de entrada disponível; • ListaR: informações das rungs: linha inicial, quantidade de linhas, posição do primeiro elemento, quantidade de elementos, posição do primeiro texto, quantidade de textos; • ListaER: armazena os elementos da rung durante a edição; • ListaVR: armazena as conexões verticais da rung durante a edição; • ListaTR: armazena os textos da rung durante a edição; • ListaBR: armazena os blocos de função da rung durante a edição; • ListaEl: armazena todos os elementos do diagrama; • ListaV: armazena todas as conexões verticais do diagrama; • ListaT: armazena todos os textos do diagrama; • ListaB: armazena todos os blocos de função do diagrama; • S: combinação de estados de ativação dos contatos; • R: estados de ativação após o contato; • P: estados de ativação resultantes; • c: especifica hierarquia de pastas; • a: determina o nome do arquivo; • i: localização das imagens e informações; • entradas: guarda os estados das entradas do simulador para atualizar o arquivo de simulação de *I/O*; • TIlb(Olb/Llb): guardam nomes das variáveis disponíveis para o programa; • TIst(Ost/Lst): guardam os status das variáveis disponíveis para o programa; • TIvi(Ovi/Lvi): guardam os valores iniciais das variáveis disponíveis para o programa; • TItd(Otd/Ltd): guardam os tipos de dados das variáveis disponíveis para o programa; • TItx(Otx/Ltx): guardam os textos das variáveis disponíveis para o programa; •
Métodos:	• f: flags para sinalizar a situação em que o programa se encontra; • l: listas de armazenamento e controle dos objetos de programação; • m: matrizes de compilação; • p: caminhos padrões que definem fonte e destino de informações ; • t: tabelas de organização dos parâmetros de funcionamento dos elementos de entrada, de saída e internos (locais);

Tabela 45 - Descrição da classe: rungs.

rungs	
Define estrutura de armazenamento dos parâmetros de cada rung.	
Atributos:	• E: posição do primeiro elemento na lista de elementos e quantidade de elementos; • T: posição do primeiro texto na lista de textos, quantidade de textos; • L: linha inicial da rung e quantidade de linhas que a compõe.
Métodos:	• Composição: Determina três tipos de informação sobre a rung: elemento, texto e linha.

Tabela 46 - Descrição da classe: blocoTimer.

blocoTimer	
Faz o gerenciamento dos temporizadores, que funcionam em paralelo ao programa principal.	
Atributos:	• IN: determina o status de habilitação do temporizador; • Q: assume o status da saída do temporizador; • TIPO: identifica o tipo do temporizador; • UN: multiplicador de unidade (base); • PT: limite de contagem de tempo; • el: representa o bloco de função; • pt: representa a variável associada de preset; • et:: representa a variável associada de contagem.
Métodos:	• CriaTimer: cria uma thread para o temporizador; • INPUT: administra a habilitação do temporizador; • OUTPUT: controla a resposta do temporizador; • TimerRUN: mantém a thread ativa e atualizando o temporizador.

Tabela 47 - Descrição da classe: edição

edição	
Realiza todos os cálculos e serviços de busca na parametrização e fixação dos elementos no diagrama.	
Atributos:	• elemento: reúne os parâmetros do elemento de programação que está em configuração; • painel: espaço de edição; • AlturaMax: tamanho da célula na vertical.
Métodos:	• LinhaRung: retorna a posição (linha) da rung e seu número de linhas; • RungAtualGb: retorna a posição (linha) da rung a que pertence um elemento gráfico; • RungAtual: retorna a posição (linha) da rung a que pertence um texto; • PxRung: retorna número de identificação e a linha onde deve ser posicionada a próxima rung; • RungFinal: retorna número de identificação e linha inicial da última rung; • IndexListaEl: calcula index de elemento na lista geral de elementos; • posiçãoOK: verifica se o elemento pode ser fixado em determinada posição; • CélulaOcupada: verifica se não há elemento numa determinada posição; • Ponto_xy: calcula o ponto de fixação do elemento na tela; • CriaGBcomPB: cria uma célula gráfica com figura; • CriaGBcomTB: cria uma célula gráfica com texto; • CriaPB: cria figura e configura seus parâmetros; • CriaGB: cria uma célula gráfica e configura seus parâmetros; • Conexão: calcula os ponstos de conexão do elemento; • PontoLigação: calcula o ponto (xy) de referência da tela a ser usado para fixar o elemento; • BuscaPin: retorna o ponto de conexão do elemento no diagrama; • OrdenaLista:coloca os elementos em ordem crescente usando como referência o ponto de conexão; • FixaImagem (Var/Texto): cria o elemento e o posiciona na tela; • ConfereCategoria: verifica se o elemento pertence a uma determinada categoria; • FioH (V): cria, parametriza e desenha fio de conexão no diagrama; • CancelaSeleçãoGB(LB/PB/TB): cancela a seleção de um elemento marcado na tela; • LimpaTela: exclui todos os objetos da tela e limpa a lista de elementos; • LimpaTabelas: exclui todos os itens das tabelas principais de organização do programa; • ApagaElemento (VarBloco/FioV/TB/PB): elimina elemento da tela e da lista de elementos e atualiza as tabelas de organização do programa; • LimpaTabelaLocal: atualiza tabelas de organização do programa excluindo item da posição indexada pelo elemento que foi eliminado da lista de elementos; • Atualiza_id: atualiza o indexador dos elementos que estão dispostos após a posição de onde foi retirado um elemento.

	• AtributosVar: atribui os parâmetros da variável declarada ao elemento; • AtributosTabela: extrai das tabelas parâmetros de um determinado elemento; • ProcuraVar: busca nas tabelas uma determinada variável; • InterpretaBase: identifica multiplicador usado pelos temporizadores na conversão de unidade para milissegundos; • ReconheceSintaxeTIMER: reconhece a unidade de tempo de uma variável; • BuscaVarAss: verifica existência de uma variável indicada para ser associada a um bloco; • Listagem: apresenta a lista de variáveis disponíveis e coerentes com a categoria do elemento em edição para escolha da que será relacionada a ele; • ListaVarAss: apresenta a lista de variáveis disponíveis e coerentes com a categoria do bloco de função que está sendo editdo, para escolha da que será associada a ele destinada à especificação ou ao armazenamento de valores; • ListaTipo: mostra a lista de variáveis de um determinado tipo; • LimpaVarAss: reinicializa variáveis auxiliares de parametrização das variáveis associadas a um bloco de função;

FORMS

Para definir e identificar os forms usados nesse projeto, segue uma série de tabelas assim como a definição das classes (Tabela 48 à Tabela 56). É apresentada a definição de cada *form* com seus atributos e métodos, descrevendo cada um deles.

Tabela 48 – Descrição do form: fmAbertura.

fmAbertura	
Faz a apresentação do APIMV e dá acesso ao aplicativo.	
Atributos:	• btEntrar: botão de acesso ao APIMV; • Fproj: chamada da tela de projetos;
Métodos:	• btEntrar_Click: chama a tela de projetos quando o botão é pressionado.

Tabela 49 – Descrição do form: fmTree.

fmTree	
Tela onde a aplicação aparece em formato de árvore, mostrando as definições lógicas e físicas do sistema. Dá acesso ao editor de programa.	
Atributos:	• Fprog: referência ao *form* fmPrograma para abrir a tela de programação; • com: referência à classe comunicação para estabelecer vínculo com maquete virtual ou habilitar o simulador de entradas e saídas; • btMqt: botão para especificar vínculo com maquete virtual; • btProjetos: botão para acessar a tela de programação; • btSair: botão para sair da aplicação;
Métodos:	• fmProjetos_Load**:** inicializações da tela e das referências a diretórios para busca ou armazenamento de arquivos; • btMqt_Click: permite especificar a localização da maquete (nome ou IP da máquina); • btProjetos_Click: realiza a leitura dos dados da maquete ou do simulador de *I/O* e faz a chamada da tela de programação; • btSair_Click: fecha a aplicação.

Tabela 50 – Descrição do form: fmProjetos.

fmProjetos	
Tela acionada no início do projeto. Define a aplicação e carrega os dados de entrada e saída. Responsável por estabelecer o vínculo com uma maquete virtual ou acionar modo de simulação de *I/O*.	
Atributos:	• caminhoTab: armazena o local de localização da maquete (fonte de dados); • Fprog: chamada da tela de programação; • com: chamada da classe de comunicação; • BrowserTab: abre uma janela para escolha do diretório; • brMqt: botão que determinar a localização da maquete; • btProjetos: botão que dá acesso ao editor; • btSair: finaliza aplicação e fecha o APIMV. •
Métodos:	• fmProjetos_Load: ao abrir a tela de Projetos é feita uma leitura de dados de *I/O* armazenados num arquivo padrão. Isso garante disponibilidade de pontos a serem associados aos elementos de programação mesmo que não se escolha uma maquete para efetuar o *link*. • brProjetos_Click: chama a tela de estrutura. • btMqt_Click: abre uma janela contendo um campo de inserção de texto para indicação da máquina onde está alocada a maquete virtual; • btSair_Click: fecha aplicativo.

Tabela 51 – Descrição do form: fmPrograma.

fmPrograma	
Tela onde fica disposto o diagrama completo, mostrando todas as rungs em seqüência, e permite a execução do programa.	
Atributos:	• Fdiag: referência ao *form* fmDiag para abrir a tela de edição de linhas de comando; • Fsim: referência ao *form* fmSimulador para abrir a tela de simulação de entradas e saídas; • Ftree: referência ao *form* fmTree para retornar à tela de estrutura da aplicação; • edit: referência à classe edição, que concentra as operações gráficas e os cálculos que definem as características e propriedades do elementos; • compila: referência à classe compilação para execução do algorítimo de interpretação lógica; • grav : referência à gravação para trabalhar com os arquivos de armazenamento de programas e com os auxiliares do simulador; • divisão: representa a linha principal (de alimentação) do diagrama; • r: componente do tipo rungs.Composição que concentra dados da rung que está em edição; • el: componente do tipo contato.Elemento, que guarda todos os parâmetros de configuração do elemento de programação que está sendo editado. • pbSelecionado: armazena figura selecionada da tela; • gbSelecionado: armazena um grupo selecionado da tela; • tbSelecionado: armazena um texto selecionado da tela; • lbSelecionado: armazena um label selecionado da tela; • LarguraMax: limita o tamanho horizontal das células da grade de edição; • AlturaMax: limita o tamanho vertical das células da grade de edição; • MargemMax: determina a margem reservada para o nome das rungs; • btRungIns: botão que aciona inserção de linha de commando ao diagrama; • btSai: fecha a tela de programa; • btRodar_Click: aciona execução da lógica e desativa operações de edição; • btParar_Click: interrompe execução da lógica e reabilita operações de edição; • btSalvar_Click: aciona gravação do diagrama em arquivo; • btAbrir_Click: aciona abertura de diagrama armazenado em arquivo;

	• btLimpar: botão que limpa a tela do diagrama; • btSimular: botão que aciona o simulador de entardas e saídas;
Métodos:	• fmPrograma_Load: inicializações da tela; • btRungIns_Click: chama a tela de edição, calcula os dados da nova rung e insere os novos elementos no diagrama; • CalcDesloc: calcula posição da rung no diagrama; • AtualizaR: atualiza dados da rung com o posicionamento calculado para localização no diagrama; • Redesenha: reproduz na tela a rung editada; • Desenha: desenha elemento na tela; • GeraDiagrama: reproduz na tela um diagrama resgatado de arquivo; • fmPrograma_Closed: fecha a janela retornando à tela de estrutura da aplicação; • btLimpar_Click: apaga todo o diagrama da tela; • btSai_Click: aciona fechamento de tela; • btAbrir_Click: aciona regate de diagrama armazenado em arquivo; • btSalvar_Click: aciona gravação do diagrama em arquivo; • btRodar_Click: aciona execução do programa e desativa operações de edição; • btParar_Click: interrompe execução do programa e reabilita operações de edição; • OnlineDes: inibe botões de edição; • OnlineHab: reabilita botões de edição; • btSimular_Click: abre a tela do simulador.

Tabela 52 – Descrição do form: fmDiag.

fmDiag	
Editor de lógicas para confecção do diagrama, onde são editadas e testadas as linhas de comando.	
Atributos:	• r: componente do tipo rungs.Composição que concentra dados da rung que está em edição; • tbR: armazena o nome da rung em edição; • PR: variável tipo point que guarda o número da rung e a linha onde se posiciona no diagrama; • elemento: componente do tipo contato.Elemento, que guarda todos os parâmetros de configuração do elemento de programação que está sendo editado. • ElVarAssI1~O2: componente do tipo contato.Elemento, que guarda todos os parâmetros de configuração das variáveis associadas ao elemento que está sendo editado; • edit: referência à classe edição, que concentra as operações gráficas e os cálculos que definem as características e propriedades do elementos; • compila: referência à classe compilação para execução do algorítimo de interpretação lógica; • grav : referência à gravação para trabalhar com os arquivos de armazenamento de programas e com os auxiliares do simulador; • Fprog: referência ao *form* fmPrograma, que é a tela de chamada ao editor de lógicas; • Fedit: referência ao *form* fmEdit para definição dos parâmetros dos elementos; • Fbloco: referência ao *form* fmBloco para escolha do grupo de funções ao qual um bloco de função pertencerá; • Fdecl: referência ao *form* fmDeclara para declaração de variáveis; • Fcoment: referência ao *form* fmComent para inserir textos no diagrama; • FsimD: referência ao *form* fmSimulador para abrir a tela de simulação de entradas e saídas; • caminho: determina o diretório padrão onde devem ser buscados os arquivos; • arquivo: especifica o arquivo que se deseja manipular o gravar; • LarguraMax: limita o tamanho horizontal das células da grade de edição; • AlturaMax: limita o tamanho vertical das células da grade de edição; • MargemMax: determina a margem reservada para o nome das rungs; • divisão: representa a linha principal (de alimentação) do diagrama; • movendo: flag que indica movimento do elemento na tela; • pbSelecionado: armazena figura selecionada da tela; • gbSelecionado: armazena um grupo selecionado da tela; • tbSelecionado: armazena um texto selecionado da tela; • lbSelecionado: armazena um label selecionado da tela; • gbSel_1 e gbSel_2: auxiliares na seleção de elementos para determinadas operações, como ligação

	em paralelo que depende da marcação de dois elementos. • ImagemBotao: assume figura referente ao botão selecionado; • tb: armazena o texto em edição que deve ser adicionado ao diagrama. • BotaoAdicionar: botão que insere um contato na rung; • BotaoBobina: botão que insere uma bobina na rung; • BotaoBloco: botão que insere um bloco na rung; • BotaoDelete: botão que apaga o elemento selecionado; • BotaoTexto: botão que insere um comentário na rung; • Button BotaoV: botão que liga elementos selecionado com linhas verticais; • timer1: contador de tempo para funcioanmento dos blocos temporizadores; • btRodar: botão que comanda o início de execução da lógica implementada; • btParar: botão que interrompe a execução da lógica; • btLimpar: botão que limpa a tela de edição; • btAbrir: botão que permite a abertura de uma lógica armazenada em arquivo; • btSalvar: botão que permite gravar em arquivo a lógica que editada; • BotaoH: botão que liga elementos selecionados com linhas horizontais; • btFechar: botão que permite sair da tela de edição confirmando a lógica editada e retorna à tela de programação onde encontra-se o diagrama completo já com a nova lógica inserida. • panel1: espaço de edição onde são dispostos os elementos gráficos; • btCancel: botão que permite sair da tela de edição ignorando a lógica editada e retorna à tela de programação onde encontra-se o diagrama completo; • btSimular: botão que aciona o simulador de entardas e saídas; • btChange: botão que permite editar um elemento selecionado;
Métodos:	• Form2_Load: inicializações da tela; • Clicado: retorna o elemento selecionado e sua posição na tela; • CancelaSelecao: cancela seleção do elemento; • BotaoDelete_Click: exclui objeto selecionado; • btChange_Click: edita um elemento selecionado; • BotaoAdicionar_Click: aciona inserção de contato e mostra uma imagem genérica referente à categoria do elemento; • BotaoBobina_Click: aciona inserção de bobina e mostra uma imagem genérica referente à categoria do elemento; • BotaoBloco_Click: aciona inserção de bloco e mostra uma imagem genérica referente à categoria do elemento; • BotaoH_Click: aciona inserção de ligações horizontais para compor conexão em série; • DesenhaH: desenha linhas horizontais estabelecendo a ligação horizontal entre os elementos; • BotaoV_Click: aciona inserção de ligações verticais para compor conexão paralela; • DesenhaV: desenha linhas verticais estabelecendo a ligação vertical entre os elementos; • BotaoTexto_Click: aciona inserção de comentário; • btLimpar_Click: apaga todos os elementos e comntários do espaço de edição; • ResetaRung: ignora dados da rung que foi apagada; • ImagemBotao_MouseDown: aciona flag de movimento quando a imagem referente ao botão, que definiu o elemento a ser adicionado, é selecionada; • ImagemBotao_MouseMove: capta a posição do mouse quando a imagem referente ao botão, que definiu o elemento a ser adicionado, é movimentada na tela; • ImagemBotao_MouseUp: fixa na tela a imagem referente ao botão, que definiu o elemento a ser adicionado, onde o mouse a soltar e cria o elemento gráfico; • VarBloco: cria na tela a variável associada o bloco de função e atualiza tabelas com seus dados; • Desenha: desenha elemento na tela; • tbTexto_MouseDown: aciona flag de movimento quando o campo de inserção de texto é selecionado; • tbTexto_MouseMove: capta a posição do mouse quando o campo de inserção de texto é movimentado na tela; • tbTexto_MouseUp: fixa na tela o campo de inserção de texto onde o mouse o soltar; • FioVFrente: traz as linhas verticais para a frente da tela de modo que fiquem totalmente visíveis; • btRodar_Click: aciona execução da lógica e desativa operações de edição; • btParar_Click: interrompe execução da lógica e reabilita operações de edição; • OnlineDes: inibe botões de edição; • OnlineHab: reabilita botões de edição; • btSalvar_Click: aciona gravação de lógica em arquivo; • btAbrir_Click: aciona abertura de lógica armazenada em arquivo;

	• Redesenha: reproduz na tela a lógica resgatada de arquivo; • fmDiag_Closed: fecha a tela de edição e restaura a de programação; • btFechar_Click: aciona fechamento de tela confirmando alterações; • btCancel_Click: aciona fechamento de tela ignorando alterações; • btSimular_Click: abre a tela do simulador.

Tabela 53 – Descrição do form: fmEdit.

fmEdit	
Tela onde são definidas as particularidades do elemento de acordo com sua categoria e definições do programador.	
Atributos:	• var: componente do tipo contatos.Var que armazena dados do elemento na declaração de variável; • elemento: componente do tipo contato.Elemento, que guarda todos os parâmetros de configuração do elemento de programação que está sendo editado; • ElVarAssI1~O2: componente do tipo contato.Elemento, que guarda todos os parâmetros de configuração das variáveis associadas ao elemento que está sendo editado; • caminho: determina o diretório padrão onde devem ser buscados os arquivos; • arquivo: especifica o arquivo que se deseja manipular o gravar; • k: flag de busca de variáveis • Fdecl: referência ao *form* fmDeclara para declaração de variáveis; • edit: referência à classe edição, que concentra as operações gráficas e os cálculos que definem as características e propriedades do elementos; • cbLabel: campo de indicação do nome a ser atribuído ao elemento; • rbTipo1~6: botões de seleção associados ao tipo de elemento. As opções variam conforme categoria; • cbVI1~cbVO2: campos para definir variáveis associadas ao elemento em edição; disponíveis somente para os blocos de função; • tbVI1 e tbVI2: campos para editar valores a serem atribuídos às variáveis associadas de entrada; disponíveis somente para os blocos de função; • btOK: botão de saída e confirmação de dados;
Métodos:	• Form2_Load: inicializações da tela; • ClearAll: limpa todos os campos da tela; • HabTipo: habilita todos os botões de seleção de tipo; • DesabTipo: desabilita todos os botões de seleção de tipo; • RstSelTipo: desmarca todos os botões de seleção de tipo; • ShowTipo: mostra todos os botões de seleção de tipo; • HideTipo: esconde todos os botões de seleção de tipo; • DisableVarAssociadaAll: desabilita todos os campos referentes a variáveis associadas; • HideVarAssociadaAll: esconde todos os campos referentes a variáveis associadas; • ShowVarAssIN1~OUT2: mostra especificamente os campos referentes à variável associada VarAssIN1~OUT2; • EnableVarAssIN1~OUT2: habilita especificamente os campos referentes à variável associada VarAssIN1~OUT2; • rbTipo1_MouseUp~rbTipo6_MouseUp: escolha do tipo de elemento; • btOK_Click: confirma a parametrização do elemento e fecha a janela atualizando a representação do elemento na tela; • btCancel_Click: fecha a janela de parametrização ignorando os dados e elimina a imagem padrão apresentada na tela durante a configuração do elemento; • ConfereDadosValidos: verifica se os valores atribuídos às variáveis associadas são coerentes com a operação do bloco de função; • AtualizaViVAss: atualiza tabelas com os novos valores atribuídos às variáveis associadas; • DeclaraVarAss: aciona declaração de variável para variável associada;

Tabela 54 – Descrição do form: fmSimulador.

fmSimulador	
Tela acionada durante a execução (RUN) do programa. Disponibiliza campos de seleção para ativar/desativar pontos de *I/O* fictícios, a fim de testar uma lógica de controle implementada.	
Atributos:	• elemento: componente do tipo contato.Elemento, que guarda todos os parâmetros de configuração do elemento de programação que está sendo editado. Nesse caso o parâmetro em questão é a categoria. • btOK: botão para confirmar alterações feitas na tela; • btCancel: botão para fechar a janela do simulador.
Métodos:	• btSimP: botão de chamada dessa tela a partir da tela de programação; • btSimD:botão de chamada dessa tela a partir da tela de edição; • fmSimulador_Load: apresenta estado inicial do sistema através dos campos na tela; • btOK_Click: confirma mudanças de estado e valores; • btCancel_Click: fecha tela de simulação.

Tabela 55 – Descrição do form: fmBloco.

fmBloco	
Tela acionada durante a edição do programa e é específica para blocos de função. Permite escolher a categoria à qual o elemento irá pertencer.	
Atributos:	• elemento: componente do tipo contato.Elemento, que guarda todos os parâmetros de configuração do elemento de programação que está sendo editado. Nesse caso o parâmetro em questão é a categoria. • btOK: botão para confirmar opção; • btCancel: botão para cancelar parametrização de bloco.
Métodos:	• fmBloco_Load: lista as categorias de blocos de função disponíveis no ambiente de programação; • btOK: confirma opção da categoria do bloco; • btCancel: interrompe a parametrização fechando a janela de seleção de bloco.

Tabela 56 – Descrição do form: fmComent.

fmComent	
Tela acionada durante a edição do programa. Disponibiliza campo de texto para que um comentário seja adicionado ao diagrama.	
Atributos:	• elemento: componente do tipo contato.Elemento, que guarda todos os parâmetros de configuração do elemento de programação que está sendo editado. Nesse caso o parâmetro em questão é a categoria. • btOK: botão para confirmar opção; • btCancel: botão para cancelar parametrização de bloco.
Métodos:	• btOK: confirma inserção do texto; • btCancel: cancela edição de comentário.