UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Desenvolvimento de uma Ferramenta de Programação baseada em SFC

Ambiente de Programação e Integração para Manufatura Virtual

Luís Phillipe Ferreira Machado

Itajubá, fevereiro de 2008

2008	**Luís Phillipe Ferreira Machado**	**Dissertação de Mestrado**

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Luís Phillipe Ferreira Machado

Desenvolvimento de uma Ferramenta de Programação baseada em SFC

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica como parte dos requisitos para obtenção do Título de Mestre em Ciências em Engenharia Elétrica

Área de Concentração: Automação e Sistemas Elétricos Industriais

Orientador: Luiz Edival de Souza

fevereiro de 2008
Itajubá - MG

AGRADECIMENTOS

Primeiramente a minha família, em especial meus pais, meu pai Wagner e minha mãe Helenice, que sempre me incentivaram, e foram responsáveis pela minha formação pessoal e profissional, me dando apoio em todos os aspectos, seja no pessoal, com seu amor e afeto, como no financeiro, me mantendo na faculdade e agora nos meses finais do mestrado, e a minha irmã Francini, pela sua enorme amizade, e apoio nos momentos difíceis.

A minha namorada Camila, que sempre soube me ouvir, me deu o apoio, e também sempre me incentivou.

Aos amigos, companheiros de república, com os quais moro há anos, e que já se tornaram meus irmãos.

Aos membros do CRTI, que de alguma forma contribuíram para a realização deste trabalho, em especial ao Welinton, que no decorrer do desenvolvimento do projeto, sempre me ajudou e me "socorreu" nos momentos de dúvida na programação deste *software*, e à Janaína, que me ajudou nas fases iniciais da implementação do programa.

Aos meus professores, Luiz Edival de Souza, meu orientador, por ter me dado a oportunidade, por ter me concedido uma bolsa de estudos, e por toda a orientação, ajuda e incentivo, e ao meu co-orientador, Leonardo de Mello Honório, pela ajuda e pelas dicas extremamente pertinentes em relação à implementação do *software*.

"Hoje levantei cedo pensando no que tenho a fazer antes que o relógio marque meia noite. É minha função escolher que tipo de dia vou ter hoje. Posso reclamar porque está chovendo ou agradecer às águas por lavarem a poluição. Posso ficar triste por não ter dinheiro ou me sentir encorajado para administrar minhas finanças, evitando o desperdício. Posso reclamar sobre minha saúde ou dar graças por estar vivo. Posso me queixar dos meus pais por não terem me dado tudo o que eu queria ou posso ser grato por ter nascido. Posso reclamar por ter que ir trabalhar ou agradecer por ter trabalho. Posso sentir tédio com o trabalho doméstico ou agradecer a Deus. Posso lamentar decepções com Amigos ou me entusiasmar com a possibilidade de fazer novas amizades. Se as coisas não saíram como planejei posso ficar feliz por ter hoje para recomeçar. O dia está na minha frente esperando para ser o que eu quiser. E aqui estou eu, o escultor que pode dar forma.

Tudo depende só de mim."

Charles Chaplin

RESUMO

Atualmente o mercado é caracterizado por uma forte competição e a busca por meios de tornar processos mais ágeis, seguros, confiáveis e obviamente, a um custo reduzido, se tornou um importante foco dentro das empresas. Uma solução encontrada foi a utilização de ambientes de simulação chamados Manufaturas Virtuais. A idéia é usar este ambiente como uma ferramenta para a identificação de erros de projetos, e determinar a funcionalidade e aplicabilidade de alguma manufatura, dentro do microcomputador, antes de realizá-la no mundo real. Este trabalho apresenta o desenvolvimento de um *software* de automação, onde o usuário poderá programar lógicas de controle, usando a linguagem SFC (*Sequential Function Chart*), padronizada pela norma IEC61131-3. Ele está inserido em um projeto do Grupo de Automação e Tecnologias da Informação, da Universidade Federal de Itajubá, com o apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), denominado APIMV, ou Ambiente de Programação e Integração para Manufatura Virtual, que é um ambiente, que oferece a possibilidade do uso, além do SFC, das demais linguagens também padronizadas pela norma citada, *Ladder*, *Instruction List*, *Function Block* e *Structured Text*, para a criação de lógicas que irão controlar os processos de maquetes virtuais, integradas ao ambiente de programação. O trabalho mostra também como foi criado a lógica usada para se fazer a compilação e execução de um programa criado pelo usuário, e os resultados obtidos.

ABSTRACT

Nowadays, the market is characterized by a strong competition, and the search for ways to make processes faster, safe, trustworthy, and obviously to a reduced cost, has become an important focus of the companies. A found solution was the use of simulation environments called Virtual Manufactures. The idea is to use this environment as a tool to identify projects errors, to determine the functionality and applicability of a manufacture, in the computer, before doing it in the real world. This work presents the development of a *software* for automation, where the user will be able to program, using the SFC language (Sequential Function Chart), standardized by IEC61131-3 Norm. It is inserted in a project of the Automation and Information Technologies Group, from Federal University of Itajubá, with the support of the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), called APMIV, or Environment of Programming and Integration for Virtual Manufacture, that is an environment, that offers the possibility of use, not only SFC, but the other languages, also standardized by the norm cited, Ladder, Instruction List, Function Block and Structured Text, for the creation of logics that will control the processes of virtual mockups, integrated to the programming environment. The work also shows the creation of the logic used for the compilation and execution of the program created by the user, and the results.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE ABREVIAÇÕES

APIMV: Ambiente de Programação e Integração para Manufatura Virtual.

SFC: Sequential Function Chart.

PLC: Programmable Logic Controller.

IEC: International Electrotechnical Commission.

UNIFEI: Universidade Federal de Itajubá.

FBD: Function Block Diagram.

ST: Structured Text.

IL: Instruction List.

LD: Ladder.

Grafcet: Graphe Fonctionnel de Command Etape Transition.

POU: Program Organization Units.

I/O: Inputs/Outputs.

CPU: Central Processing Unit.

OPC: Ole for Process Control.

1. INTRODUÇÃO

Nos últimos tempos, a automação tem sido um grande destaque no meio industrial, devido às grandes facilidades que ela trás. Através dela, o usuário é capaz de maximizar com precisão e qualidade seu processo produtivo, controlando, por exemplo, variáveis diversas e podendo gerenciar toda sua cadeia produtiva à distância. E com a forte competitividade do mercado atual, é bastante cômodo, e também lucrativo, para uma empresa, ter seus processos automatizados, com uso reduzido de mão-de-obra.

O parque tecnológico brasileiro também vem se tornando cada vez mais atuante em diversas áreas e a automação de manufatura é um importante foco dos estudiosos da área. Com isso, cursos de graduação relacionados são instituídos e assim, alunos podem ter acesso a equipamentos de automação, com os quais eles adquirem os conhecimentos teóricos e também são treinados a usá-los. No entanto, esta é uma área de equipamentos caros, sendo que muitos deles necessitam ser importados, ou seja, envolve um alto custo, o que limita uma universidade de poder contar com vários aparelhos, resultando, assim, numa conseqüente queda na qualidade do ensino.

Para suprir essas limitações dos recursos de *hardware*, os microcomputadores vêm sendo utilizados como uma alternativa, provendo aos usuários o treinamento necessário, através de ambientes de simulação, ou realidade virtual. São criadas então, as chamadas manufaturas virtuais, uma nova proposta que está sendo aplicada por empresas a fim de revolucionar os seus processos.

A idéia é criar um ambiente integrado, composto de diversos sistemas e ferramentas de *software*, para gerar um novo método de desenvolvimento de produtos e através do uso dos microcomputadores, dispositivos audiovisuais e sensores, poder simular uma manufatura, com o objetivo de prever, com maior segurança, possíveis erros de projeto e ineficiências na funcionalidade e

manufaturabilidade do produto. Dentro deste contexto, vem sendo desenvolvido pelo Grupo de Automação e Tecnologias da Informação, da Universidade Federal de Itajubá, um ambiente de manufatura virtual, chamado APIMV, Ambiente de Programação e Integração para Manufatura Virtual [16]. É um ambiente bastante simples de ser programado, e tem como base a norma IEC61131-3 [1], que padroniza as linguagens usadas na automação industrial. O objetivo é tornar o programa aberto e oferecer a possibilidade de programação nas cinco linguagens padronizadas por essa norma, *Ladder*, *Structured Text*, *Function Block*, *Instruction List* e *Sequential Function Chart* (SFC). Através deste ambiente, o usuário pode desenvolver aplicações, efetuar a programação e também monitorar suas aplicações, através de uma interface amigável e intuitiva.

O foco deste trabalho é o desenvolvimento do ambiente de programação, a ser usado no APIMV, no qual o usuário poderá programar utilizando a linguagem SFC, e integração entre a lógica de controle e a maquete. A norma IEC61131-3 define o SFC como uma ferramenta de programação essencial para sistemas de controle. Sua estrutura clara permite uma visão global do programa de controle e é, portanto, uma das partes mais importantes dessa norma.

Treze anos se passaram desde a sua publicação [2], mas muitos dos seus conceitos ainda são ignorados por uma grande quantidade de pessoas envolvidas com automação industrial, e durante anos a linguagem *Ladder* foi a única utilizada para programar PLCs. Mas o uso da linguagem SFC facilita a vida do usuário, pois trás algumas vantagens, como por exemplo, uma sintaxe reduzida e maior facilidade na identificação de erros no projeto. É uma linguagem bastante apropriada para ser utilizada nas etapas iniciais e nas fases mais cruciais do desenvolvimento do *software*.

Para se testar a funcionalidade do sistema em questão, foi criada uma maquete virtual de uma mesa giratória Festo e também algumas aplicações em SFC para efetuar o controle desta mesa.

1.1. OBJETIVO

O objetivo desta dissertação consiste na criação e desenvolvimento de uma ferramenta de programação, onde o usuário poderá criar aplicações na linguagem SFC, padronizada pela norma IEC61131-3, de modo que possa efetuar edição e manipulação dos elementos gráficos na tela. Esta ferramenta deverá ser integrada com uma Maquete Virtual do seguinte modo: quando o programa está sendo executado, é feita a leitura de status de todos os sensores desta maquete. A aplicação criada é executada, considerando os valores recebidos, e os resultados são atualizados e enviados à maquete, que por sua vez, se comporta de acordo com a lógica de controle criada.

Espera-se que a ferramenta traga facilidades ao usuário, oferecendo um ambiente amigável de programação, onde a manipulação e edição dos elementos sejam simples e fáceis.

Em suma, a idéia é possibilitar, com este sistema, a identificação de possíveis falhas, a verificação da funcionalidade de uma manufatura no mundo virtual, antes que seu modelo real seja utilizado, e também ser aplicado em universidades como uma ferramenta para o treinamento dos alunos.

1.2. A REALIDADE VIRTUAL NA AUTOMAÇÃO

A realidade virtual vem sendo muito utilizada nos mais variados campos do conhecimento. Um fator importante para este fato é o estudo envolvendo a realidade virtual nas universidades, e sua respectiva divulgação nas indústrias [3], e o acesso cada vez maior das pessoas à informática.

Existem várias definições de realidade virtual, como por exemplo, uma interface homem-máquina [4], que simula um ambiente real e permite que os usuários interajam neste ambiente. Há também a definição de realidade virtual como o uso da alta tecnologia [5] para convencer o usuário de que ele está em outra realidade, um novo meio de “estar” e “tocar” em informações. É um modelo feito no microcomputador, baseado num modelo real, para se determinar seu comportamento, suas reações, em variadas situações.

Uma interface de realidade virtual deve possibilitar que o usuário navegue neste mundo não-real e interaja com a aplicação em tempo real, causando um sentimento de imersão, ou seja, que o mesmo faz parte daquele mundo.

No setor da energia, por exemplo, a realidade virtual é usada para reduzir custos [6], e aumentar a produtividade em projetos, manutenção de plantas, subestações e linhas de transmissão. Em empresas dos setores automotivo e aeroespacial, a aceitação dessa tecnologia é cada vez mais extensa, a *Boeing*, por exemplo, desenvolveu o modelo 777 [7] virtualmente antes de fazê-lo no mundo real.

A realidade virtual, aplicada a uma manufatura, deve ser feita de maneira bastante completa, pois o objetivo é prever potenciais problemas e ineficiências na funcionalidade e manufaturabilidade de um produto. Para isso, as características do sistema devem ser compreendidas, e tem que se avaliar tempos de ciclo, comportamento de sensores e atuadores, eventos e reações.

Há no mercado algumas ferramentas que possibilitam manipulações de objetos em realidade virtual, algumas em 2D e outras em 3D, com recursos que auxiliam na avaliação de um projeto:

- *Vericut*, da *CGTech* [8], que simula o processo de remoção de um determinado material e detecta colisão de ferramentas, interferências, condições de corte inadequadas;
- *UltraArc* da *Delmia* [9], que é usada em programação de robôs, para desenvolvimento, programação e otimização de aplicações em células de manufatura;

- *Flexman* [10], *software* de modelagem 3D de manufatura flexível.

O *Vericut* é uma ferramenta que detecta erros, possíveis colisões ou áreas de ineficiência, e através dele, pode-se modelar e simular uma máquina. Permite visualizar todas as funções da máquina virtualmente, simulando o chão de fábrica de uma empresa. No entanto, não possibilita uma programação usando linguagens industriais padronizadas.

O *UltraArc* provê simulação gráfica 3D para robótica. Pode gerar automaticamente movimentos de robôs. As células de manufatura são desenvolvidas usando bibliotecas de construção, tabelas de posicionamento, pistolas de soldagem e outros equipamentos relacionados, mas oferece a mesma carência do Vericut.

Outra ferramenta é o *Flexman*, para desenvolvimento e simulação de sistemas de manufatura flexíveis. Através dele, pode-se verificar várias configurações de chão de fábrica [11], as estratégias de trabalho, para alcançar um nível de flexibilidade da manufatura necessário para se garantir uma competitividade com indústrias do segmento. Permite ao usuário criar um sistema de controle reconfigurável dinamicamente com base nas informações dos sensores. Oferece uma interface gráfica para se criar simulações. Também não dá condições de se desenvolver uma lógica de controle usando uma linguagem industrial padronizada.

A grande maioria dos *softwares* desenvolvidos para esse tipo de controle e simulação é dedicada a equipamentos. Por exemplo, o *PCWorx*, voltado para equipamentos da *Phoenix Contact* [12], o *Straton*, ferramenta de automação voltada para o sistema *WAGO-I/O*[1] [13], e o *Step 7*, da *Siemens* [14].

[1] Sistema de barramento industrial modular para indústria, automação de acabamento e processamento. Possibilita livre combinação de módulos digitais, analógicos e módulos mais complexos.

Há também uma ferramenta, o *Isagraf* [15], um ambiente de desenvolvimento, independente de *hardware*, baseado no padrão internacional para linguagens de programação de controle e automação, IEC61131-3, oferecendo programação nas cinco linguagens padronizadas. Porém, não está integrado a um sistema de manufatura virtual.

A manufatura virtual desenvolvida pelo Grupo de Automação e Tecnologias da Informação é vinculada a um ambiente de programação, onde o usuário poderá desenvolver suas aplicações nas linguagens industriais [16], e assim, efetuar o controle da maquete virtual. É independente de equipamento ou fabricante. Provê uma programação simples, também baseada na norma IEC61131-3, com uma interface intuitiva. E também apresenta um baixo custo, o que torna viável sua utilização em universidades e empresas.

1.3. O CONTEÚDO DA DISSERTAÇÃO

Este trabalho está organizado em dez capítulos e três apêndices. O capítulo dois descreve manufatura virtual e maquete virtual; o capítulo três aborda a linguagem SFC, e a norma IEC61131-3; o capítulo quatro enfoca rapidamente a linguagem e a ferramenta utilizadas para desenvolver-se o código do programa, bem como o método utilizado; o capítulo cinco aborda o ambiente de programação, lógica de criação do grafo de elementos, lógica de compilação, e comunicação com a maquete; no capítulo seis, são mostradas algumas telas do programa; o capítulo sete explana a aplicabilidade do sistema criado; no capítulo oito, têm-se os resultados obtidos com algumas aplicações realizadas e finalmente a conclusão do trabalho é feita no capítulo nove. Nos anexos, há uma descrição mais detalhada do programa, seu diagrama de classes conceitual, três exemplos da utilização do ambiente, mostrando com mais detalhes a maquete virtual usada nas simulações, e diagramas contendo a situação passada, presente e futura do APIMV.

2. MANUFATURA VIRTUAL

O termo manufatura virtual passou a ser usado na década de 90, inicialmente como resultado de uma ação do governo dos Estados Unidos, que foi desenvolver a capacidade de confirmar a confiabilidade, a manufaturabilidade e a possibilidade de novos sistemas de armas antes de realizá-los no mundo real.

É um modelo virtual, que pode ser usado em uma grande variedade de contextos de sistemas de manufaturas, que utiliza modelos computacionais, dispositivos audiovisuais e sensores para simular ou projetar o ambiente da manufatura e assim auxiliar no projeto de um determinado produto, prevendo problemas, falhas, ineficiências na funcionalidade, evitando, assim, prejuízos reais. É uma integração [17] de um ambiente sintético de manufatura, ou seja, com objetos, atividades e processos simulados, capaz de realçar todos os níveis de controle e decisões do sistema.

Apresenta algumas vantagens:

- Capacidade de avaliar com mais detalhes um número maior de alternativas, melhorando assim a qualidade de um projeto;
- Os componentes físicos e conceituais de um sistema podem ser representados através de entidades de *software*, que emulam sua estrutura e função;
- Redução nos ciclos e custos [18] de desenvolvimento, sendo que não é necessário assim fazer um grande número de protótipos físicos;
- É uma excelente maneira de se estabelecer a credibilidade para o modelo simulado;
- É mais bem aplicado [19] como treinamento para especialização de profissionais e estudantes, que usar sistemas reais, ou seja, estes usuários apresentam melhor desempenho quando treinados num sistema virtual;
- Possibilidade de mudanças mais [20] efetivas no sistema de manufatura;
- Análises do sistema simulado mais seguras e precisas;

- Maior segurança na integração dos equipamentos, pois permite testes e modificações no projeto para eliminar falhas e pontos de risco.

O ambiente que vem sendo desenvolvido na UNIFEI apresenta basicamente, duas partes principais, a maquete virtual, uma representação gráfica de determinados objetos que fazem parte da manufatura real; e o ambiente de programação, onde o usuário poderá escolher a linguagem e definir sua lógica de controle da maquete. As duas partes brevemente descritas são integradas e a troca de informações entre elas é ilustrada na tela do computador, com animações [16], equivalentes ao comportamento de cada elemento da manufatura real.

2.1. MAQUETE VIRTUAL

A maquete virtual é um avançado recurso de informática, que integra técnicas de gerenciamento de dados [21], e computação gráfica.

É uma representação tridimensional feita em computador, de uma determinada maquete real, onde poderá haver uma interação do usuário. Esta maquete é feita de modo que seu comportamento fique exatamente, ou o mais próximo possível, do comportamento de seu modelo real, e através dela, pode-se simular diversas situações que talvez, no mundo real, fosse arriscado ou perigoso. Provê uma maior versatilidade, interatividade, permite modificar um projeto futuro e ainda interagir com este, alterando suas características.

A Figura 1 mostra dois exemplos de maquetes virtuais desenvolvidas por professores e alunos da Universidade Federal de Itajubá.

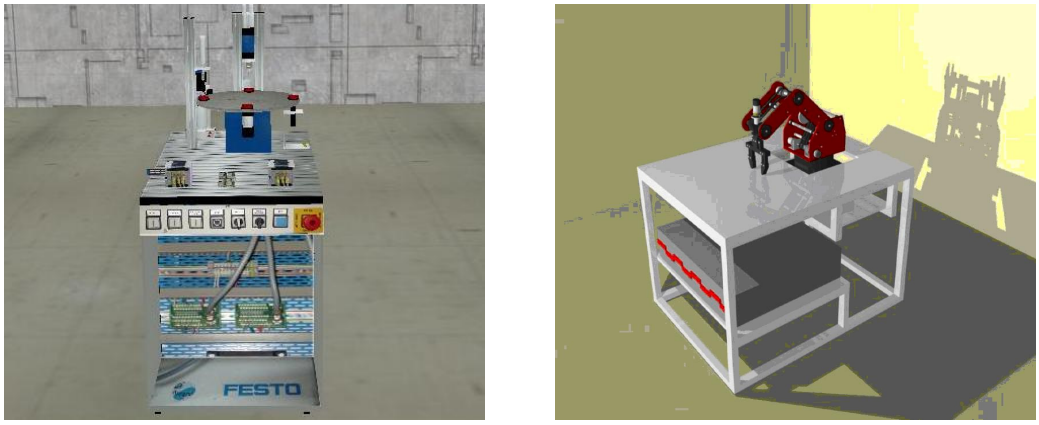


Figura 1 - Maquetes virtuais

Dentro do meio industrial, a maquete é uma excelente maneira de se representar a forma e o comportamento de equipamentos de campo, como sensores, atuadores e controladores.

É também uma ferramenta de treinamento para estudantes e profissionais, com vantagens significativas: redução de custos, pois equipamentos para automação de uma manufatura são caros, podendo chegar a milhares de dólares; o usuário pode interagir com a maquete e não apenas observar, o que atrai mais quem está aprendendo; e o tempo de aprendizado também diminui.

Basicamente, as maquetes virtuais são desenvolvidas e criadas em quatro etapas [22]. Primeiramente, são feitos a escolha e minuciosos estudos do componente a ser representado virtualmente. Tal modelo é cotado com extrema precisão [23], são tiradas fotos de cada detalhe para que a representação virtual seja a mais real possível. É feita então a modelagem deste componente. Depois, a dinâmica de movimento também é tratada e modelada com detalhes, após análises de otimização do desempenho do modelo virtual. Finalmente observa-se o comportamento e relacionamento do componente com o mundo virtual. Pode-se utilizar um *software* de projetos em 3D para gerar as maquetes. Cria-se então, para o usuário, o efeito de mundo tridimensional, que inclui objetos interativos, e uma forte sensação de presença tridimensional. A Figura 2 representa o elevado nível de detalhamento do modelo real.

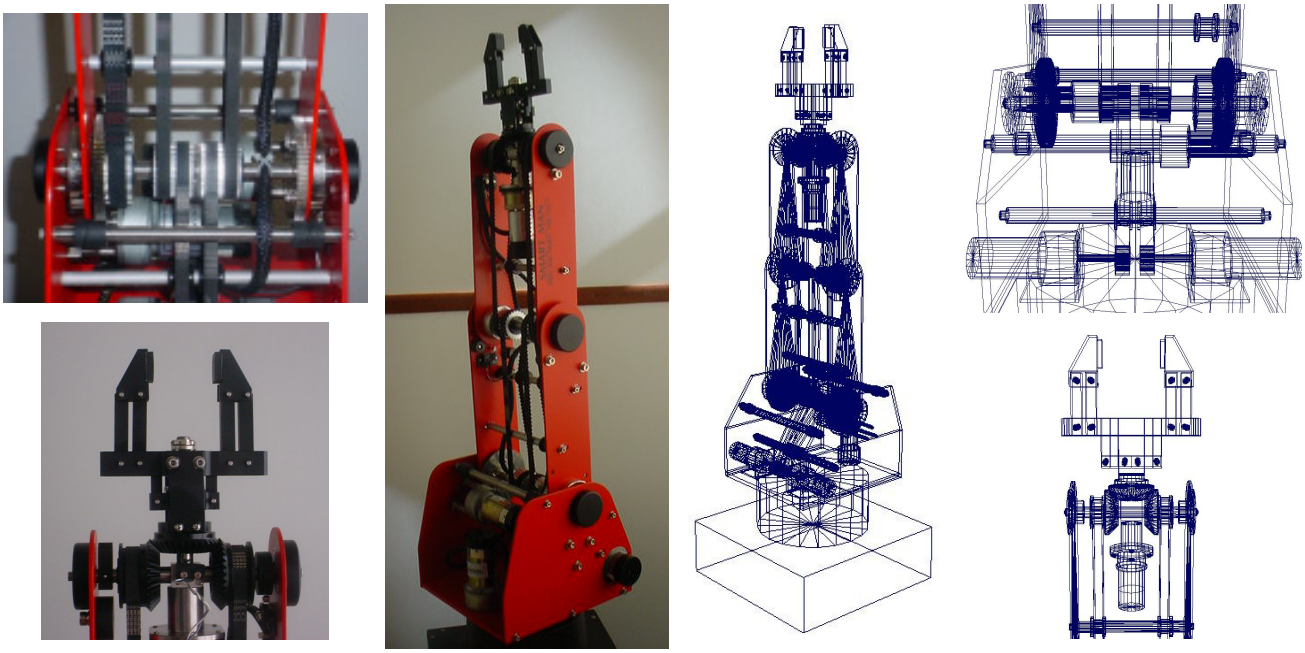


Figura 2 - Fotografias e representação gráfica do braço mecânico[22]

A Figura 3 mostra o modelo real do braço mecânico SCORBOT-ER V, e seu modelo virtual.

Figura 3 - Modelo real e sua maquete virtual[22]

3. A LINGUAGEM SFC

Sequential Function Chart, ou Diagrama Seqüencial de Funções, é uma linguagem gráfica que descreve o comportamento de um sistema através de uma seqüência de eventos ou processos [24]. É uma ferramenta essencial para sistemas de controle. As definições para o SFC [25] na norma IEC61131-3 [26] são baseadas no padrão francês IEC60848 [27], publicado em 1988.

Basicamente, a linguagem SFC é composta de passos e transições [28]. Os passos são representados graficamente por caixas retangulares, e representam os estados particulares do sistema a ser controlado. Sempre há um passo inicial, que obviamente é o primeiro passo a ser ativado. A cada passo está associada uma ação, que é executada quando o mesmo é ativado.

Os passos são conectados por transições, que são representadas graficamente por uma linha cortada. A cada transição é associada uma condição. Quando a condição é satisfeita, o passo anterior a ela é desativado e o seguinte se torna ativo.

As transições podem também ser diretamente descritas usando as linguagens FBD, ST ou LD, ou descritas em associação a diagramas usando qualquer uma das quatro demais linguagens. Neste caso, a definição é associada ao nome da transição.

Um fator importante a ser mencionado é que o SFC mostra os estados principais de um sistema, todas as possíveis mudanças de estado, e as razões para que estas mudanças ocorram. Ela pode ser usada no nível mais alto, para mostrar as principais fases de um processo e também em níveis mais baixos.

A Figura 4 mostra um exemplo básico do diagrama seqüencial de funções.

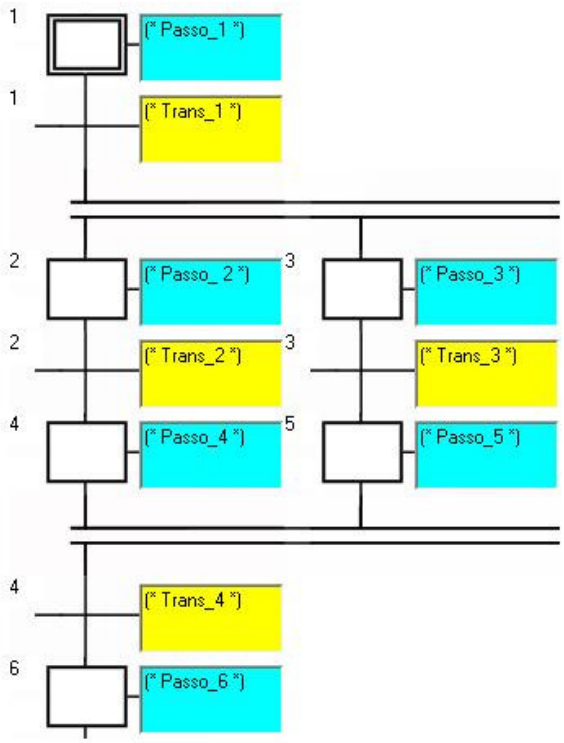


Figura 4 - Diagrama seqüencial de funções

3.1. NORMA IEC61131-3

Com o decorrer da história, avanços no campo tecnológico sempre estiveram diretamente ligados aos processos industriais. E uma maneira de se fazer este progresso se tornar mais rápido e eficaz é a utilização de padrões e normas. Os diversos progressos surgidos durante a Revolução Industrial só começaram a acelerar e a se tornar evidentes após o desenvolvimento de padrões. Henry Ford, por exemplo, só começou a obter sucesso nas suas linhas de montagem automobilísticas após realizar uma padronização de medidas das peças usadas nessas linhas.

No início dos anos 80, algumas normas surgiram para a definição da programação de PLCs, como a norma francesa NFC-03-190, para definição da linguagem GRAFCET [29].

Neste mesmo período, uma organização internacional, *International Electrotechnical Commission* [29], a IEC, começou a definir uma norma para os PLCs, levando em consideração vários aspectos relacionados, como sua instalação, a parte de *hardware*, testes, documentação e programação. Criou-se então um grupo para discutir e escrever a norma, chamado de W7, ou *Working Group 7*. Este grupo foi dividido em forças-tarefas, cada uma responsável por uma parte, e a força-tarefa três desenvolveu o padrão das linguagens, chamado IEC1131, que a partir de 1998 passou a ser chamada de IEC61131. Em 1993, a norma referente à parte três foi publicada (IEC61131-3).

3.1.1. Características da Norma IEC61131-3

É uma parte da norma IEC61131, destinada ao modelo de *software* e às linguagens de programação [30]. Cada vez mais os usuários têm percebido seus benefícios [31], embora ainda haja muitos erros de interpretação. Ela serve para decompor as soluções dos problemas de automação em partes distintas com interfaces formais e claras, ou seja, divide o programa em pequenas partes gerenciáveis.

Esta estrutura de decomposição dos problemas é construída em camadas, cada uma com características e funções distintas, sendo que as camadas mais baixas são mais ligadas ao controle do processo e são controladas pelas camadas superiores.

Os elementos inferiores nesta hierarquia de modelo de *software* são chamados de POUs (*Program Organization Units*), e são divididas em Programas, Blocos de Funções, e Funções.

O Programa é a organização lógica dos elementos das linguagens e as construções a serem usadas para o processamento do sinal necessário ao controle de algum processo ou máquina, ou seja, é mais usado em soluções mais específicas.

Blocos de Função são as unidades que produzem os valores a serem executados. Podem ser criadas instâncias dos blocos, às quais estarão associadas às variáveis de saída e também às variáveis internas.

Já as Funções produzem um único valor de saída e podem ser chamadas como lógicas em linguagens textuais, ou como operandos em expressões algébricas. Não se pode guardar qualquer informação de estado nas funções, sendo que elas não podem conter dados internos. As funções são mais apropriadas para a resolução de problemas combinatoriais, sem a necessidade de variáveis internas no registro de dados.

Acima, na hierarquia, há os recursos ou *resources*, que podem ser vistos como uma máquina virtual, que oferece as características necessárias para executar os programas. Provê acessos a pontos de *I/*O [29], e permite o processamento de programas baseados na norma. Dentro desses *resources*, podem haver uma ou mais tarefas ou *tasks*, que irão comandar a execução dos programas e blocos de funções.

A camada mais alta é a *configuration,* ou camada de configuração. Contém todos os elementos de *software* a serem processados no PLC, os *resources*, as tarefas, os Blocos de Funções, etc.

Tem-se na Figura 5 o modelo de *software* da norma IEC61131-3.

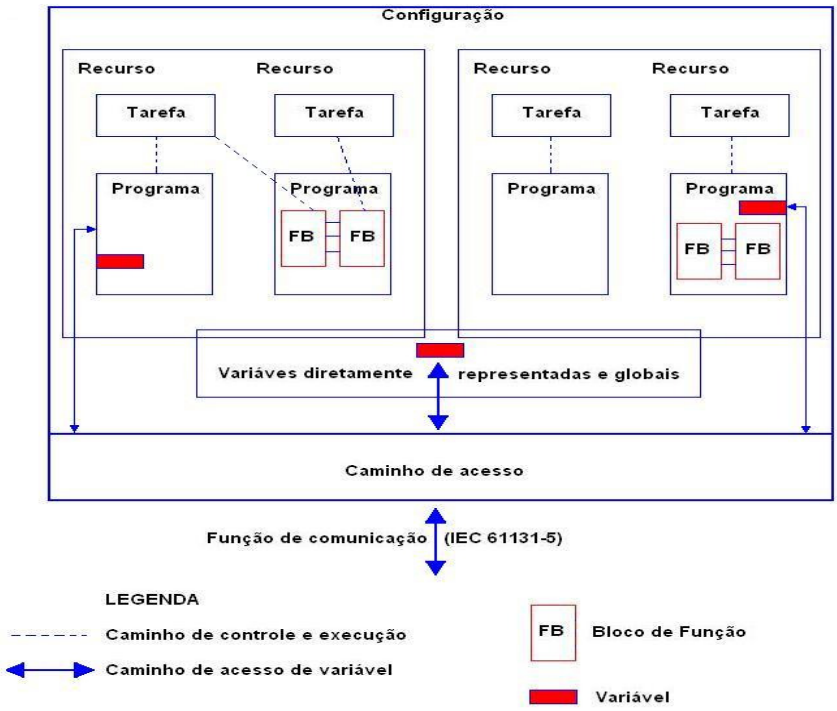


Figura 5 - Modelo de software da norma IEC61131-3

Resumidamente, os principais conceitos e características apresentados pela norma IEC61131-3 são:

- Tipos de dados;
- Base de dados com declaração de variáveis e alocação dinâmica;
- Estruturação, modularização, portabilidade, e reutilização de *software*;
- Orientação de objetos;
- Processamento multitarefa;
- Cinco linguagens de programação.

3.1.2. Linguagens de Programação

Foram criadas cinco linguagens, a partir da racionalização de diferentes linguagens já existentes, e dos vários dialetos do *Ladder*. Cada uma delas pode ser usada para descrever partes do programa [32]. Assim, o desenvolvedor do projeto poderá escolher qual a linguagem mais apropriada para um determinado tipo de aplicação. As linguagens são:

- Diagrama *Ladder* – LD;
- Lista de Instruções – IL;
- Diagramas de Blocos Funcionais – FBD;
- Texto Estruturado – ST;
- Diagrama Seqüencial de Funções – SFC.

A linguagem *Ladder* é uma linguagem gráfica que utiliza símbolos de relés [33]. Foi a primeira linguagem destinada especificamente à programação de PLCs. Lista de Instruções é uma linguagem de baixo nível, uma seqüência de instruções bem parecida com *Assembley*. Diagrama de Blocos Funcionais nada mais é que uma linguagem gráfica, onde blocos de funções se comunicam e processam sinais e fluxo de dados. A linguagem Texto Estruturado é textual e de alto nível, bem semelhante ao Pascal. E o SFC, foco deste trabalho, é uma linguagem gráfica que permite definir o comportamento seqüencial de um sistema, através de eventos específicos e ou intervalos de tempos.

A norma define a sintaxe das suas linguagens textuais, formalmente e sem ambigüidades, bem como a sua semântica.

Pode-se citar como vantagens do uso dos conceitos da norma os seguintes pontos:

- É possível executar diferentes partes do programa com prioridades e taxas distintas;

- Com a utilização do modelo de *software* e das POUs, os programas são construídos de maneira mais estruturada;
- A interface do PLC detecta quando o programador tenta atribuir um dado de formato errado a uma variável;
- Através da linguagem SFC, os comportamentos seqüenciais complexos podem ser descritos de forma mais clara;
- Flexibilidade na seleção da linguagem, de acordo com o problema e o projetista.

No entanto, há algumas desvantagens, como por exemplo, os formatos dos arquivos que armazenam os programas dos PLCs não terem sido padronizados, o que torna difícil a utilização do código de um fabricante, no PLC de outro fabricante.

3.2. CICLO DE VARREDURA

Figura 6 - Ciclo de Varredura

Programas são criados e inseridos na memória de uma *CPU* (Unidade Central de Processamento), em forma de operações. E na execução destes programas, o PLC realiza os chamados ciclos de varreduras, que como na Figura 6, consiste em três etapas:

- leitura das entradas externas;
- execução da lógica programada;
- atualização das saídas externas.

Na leitura dos dados de entrada, o processador recebe os estados dos dispositivos conectados e guarda suas informações em uma tabela imagem das entradas. A lógica de controle inserida pelo usuário é executada e os resultados lógicos são armazenados em uma tabela imagem das saídas. Finalmente, as saídas externas são efetivamente atualizadas. Em seguida, um novo ciclo é iniciado e isto ocorre enquanto um determinado programa estiver sendo executado.

3.3. SEQÜÊNCIAS CONVERGENTES, DIVERGENTES E SIMULTÂNEAS

Nem sempre os passos são ativados numa mesma seqüência. É possível que haja caminhos divergentes, convergentes ou simultâneos.

3.3.1. Ou Divergente

Uma seqüência “ou divergente” se dá quando há duas transições após um passo. Geralmente, as transições são verificadas da esquerda para direita. Na Figura 7 há um exemplo dessa divergência.

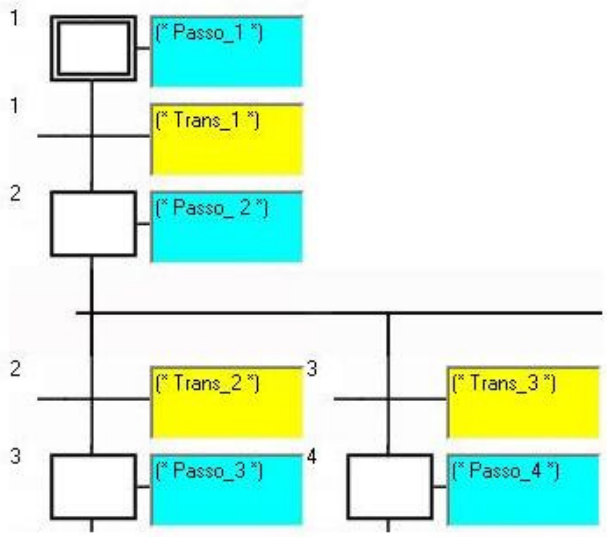


Figura 7 - Ou divergente

Quando *Passo_2* estiver ativo, as duas transições seguintes são testadas, primeiramente *Trans_2*. Se a condição referente a ela não for satisfeita, verifica-se a condição de *Trans_3*. Se for satisfeita, *Passo_4* se torna ativo e o passo anterior *Passo_2* fica inativo. Se em ambas as transições as condições não forem satisfeitas, *Passo_2* fica ativo até que uma das duas transições seguintes seja satisfeita.

Se primeiramente *Trans_2* for satisfeita, *Passo_3* se torna ativo e *Passo_2* fica inativo.

3.3.2. Ou Convergente

A seqüência “ou convergente” se dá quando duas transições com passos distintos de entrada apresentam um passo apenas de saída, como na Figura 8.

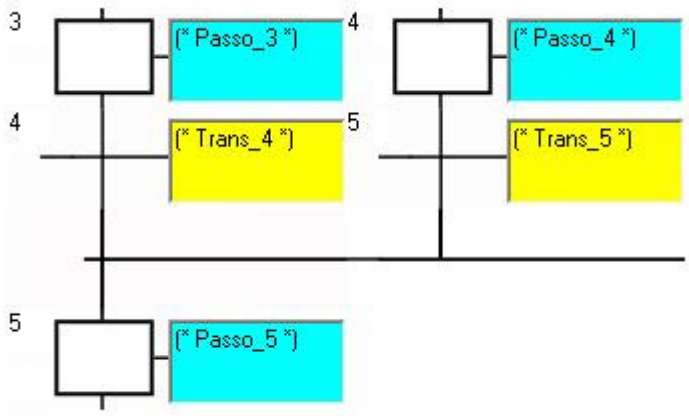


Figura 8 - Ou convergente

Para que *Passo_5* se torne ativo a condição de *Trans_4* ou *Trans_5* deve ser satisfeita, não obrigatoriamente ambas. Se *Trans_4* for satisfeita, *Passo_3* fica inativo e o passo seguinte é ativado. O mesmo ocorre para *Trans_5*, ou seja, caso sua condição seja satisfeita, *Passo_4* é desativado e *Passo_5* passa a ficar ativo.

3.3.3. E Divergente

Ocorre quando após uma transição há dois ou mais passos simultâneos.

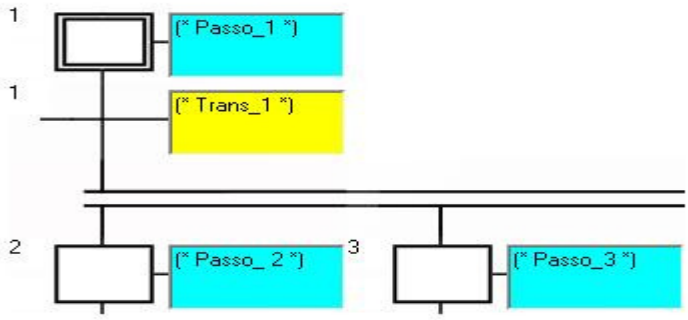


Figura 9 - E divergente

Passo_1 estando ativo, as ações referentes a ele são executadas e as condições de *Trans_1* são verificadas em seguida. Caso sejam satisfeitas, *Passo_1* fica inativo e *Passo_2* e *Passo_3* ficam ativos.

3.3.4. E Convergente

Esta seqüência se dá quando uma transição é precedida por dois ou mais passos simultâneos, como na Figura 10.

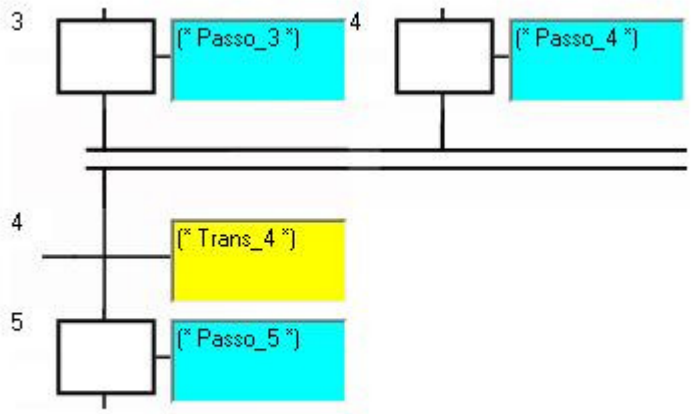


Figura 10 - E convergente

Para que *Trans_4* possa ser verificada, *Passo_3* e *Passo_4* devem estar ativos, ou seja, todos os estados anteriores à transição que sofre a convergência devem estar ativos para que ela seja verificada.

3.3.5. Simultaneidade

Ocorre quando passos precedem ou sucedem uma transição, sendo que para que esta transição seja verificada, todos os passos anteriores devem ter sido

ativados, o que resulta na conseqüente ativação dos passos posteriores caso a referida condição seja verdadeira. Pode-se observar isto na Figura 11.

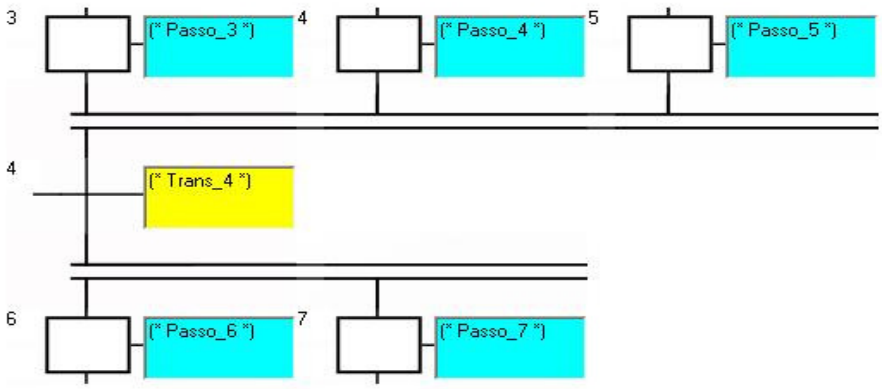


Figura 11 – Simultaneidade

3.4. AÇÕES BOOLEANAS

As ações booleanas [34] são utilizadas para atribuir um valor a uma variável booleana com a ativação de um determinado passo. A variável booleana pode ser de saída ou interna.

Há três tipos básicos de ações booleanas:

- Ação N, atribui o sinal de ativação do passo à variável, ou seja, enquanto o passo está ativo, o valor *TRUE* é atribuído à variável;
- Ação /, ou ação de negação, dá à variável a negação do sinal de ativação do passo;
- Ação S, assinala o valor *TRUE* à variável, quando o sinal de ativação do passo se torna *TRUE*;
- Ação R, "dá um *reset"* na variável, ou seja, atribui o valor *FALSE* a ela, quando o sinal de ativação do passo se torna *TRUE*.

Isso pode ser melhor observado na Figura 12.

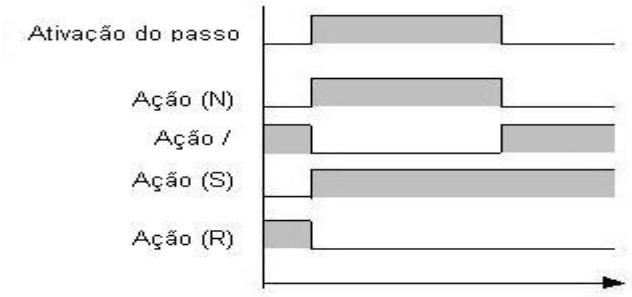


Figura 12 - Ações booleanas

3.5. AÇÕES *NON-STORED*

É uma lista de instruções em *Structured Text* ou *Instruction List* que são executadas [35] a cada ciclo de varredura durante todo o período de ativação do passo. Há por exemplo a ação *Action (N)*, que se comporta como na Figura 13.

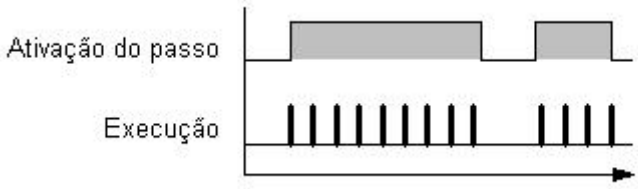


Figura 13 - Non-stored action

Na Figura 13, por exemplo, na primeira ativação do passo são realizados nove ciclos de varredura, onde a ação ou ações referentes aquele passo são executadas a cada ciclo.

3.6. AÇÕES PULSO

Uma ação pulso é uma lista de instruções em *Structured Text* ou *Instruction List* que são executadas apenas na ativação do passo, como um pulso. Observa-se na Figura 14.

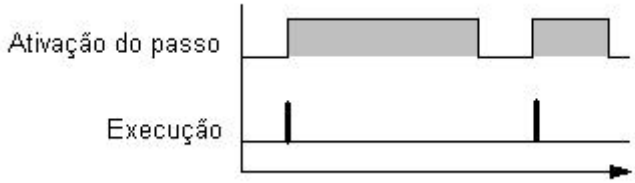


Figura 14 - Ação pulso

3.7. *JUMP*

O *jump* é um elemento usado na linguagem SFC para ligar uma transição a um passo qualquer, ou um passo a uma transição qualquer. Isto pode ser observado na Figura 15, onde no caso A, o *jump* liga T2 ao passo inicial PI1 e no caso B conecta P3 à transição T1.

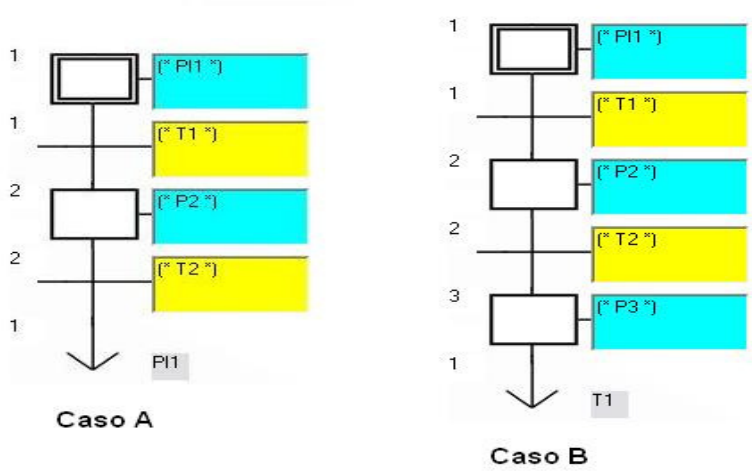


Figura 15 - Casos do elemento jump

4. MATERIAIS E MÉTODOS

O programa foi desenvolvido na linguagem *C-Sharp*, com o uso da ferramenta *Microsoft Visual Studio .Net 2005*, usando os conceitos de orientação a objetos.

Primeiramente, esta ferramenta de desenvolvimento foi escolhida devido a uma séria de vantagens que a mesma trás, como por exemplo, a orientação a objetos. Mas sem dúvida a maior vantagem não está na linguagem, mas sim na plataforma. O *.Net Framework* fornece um ambiente de execução completamente seguro. Como o código não é compilado, mas sim interpretado, o *Framework* tem controle total sobre a execução, desde prevenção de ações maliciosas até o controle de erros que poderiam travar o sistema. E uma aplicação desenvolvida no *.Net* pode ser executada em qualquer plataforma, bastando que para isto o *Framework* compatível esteja instalado.

Já o *C-Sharp* é a mais nova linguagem [36] que a Microsoft criou a partir do zero unindo o que existe de melhor em todas as tecnologias no mercado e adicionou um pouco de facilidades e recursos.

E finalmente a orientação a objetos [37] trás também vários benefícios como:

- Maior facilidade para reutilização de código e por conseqüência, do projeto;
- Possibilidade de o desenvolvedor trabalhar em um nível mais elevado de abstração;
- Utilização de um único padrão conceitual durante todo o processo de criação de *software*;
- Maior adequação à arquitetura cliente/servidor;
- Maior facilidade de comunicação com os usuários e com outros profissionais de informática;
- Ciclo de vida mais longo para os sistemas;
- Desenvolvimento acelerado de sistemas;

- Possibilidade de se construir sistema muito mais complexos, pela incorporação de funções prontas;
- Menor custo para desenvolvimento e manutenção de sistemas.

5. AMBIENTE DE PROGRAMAÇÃO DESENVOLVIDO

Dentro do propósito de se criar uma manufatura virtual, um fator de suma importância é o ambiente de programação, pois é onde o usuário irá criar as lógicas de controle de uma determinada maquete virtual, podendo testar variadas situações antes de aplicá-las no mundo real e observar o comportamento da maquete de acordo com suas "instruções".

Este capítulo trata justamente do ambiente de programação desenvolvido para a linguagem SFC, foco deste trabalho, mostrando inicialmente o sistema em que ele está inserido, o APIMV, ou Ambiente de Programação e Integração para Manufatura Virtual, e posteriormente suas características, seus modos de funcionamento, comunicação com a maquete virtual e o desenvolvimento da lógica para a linguagem em questão.

5.1. AMBIENTE DE PROGRAMAÇÃO E INTEGRAÇÃO PARA MANUFATURA VIRTUAL

Este ambiente, também chamado APIMV, provê uma programação simples [16], independente de equipamentos ou fabricantes; para aplicações reais, não há a necessidade de se efetuar conversões de código para linguagens de mercado; e oferece uma interface amigável e de rápida assimilação.

Está sendo desenvolvido no *Microsoft Visual Studio .Net 2005*, na linguagem de programação *C-Sharp*, basicamente usando os conceitos de orientação a objetos

É baseado na norma IEC61131-3 [26], que dá os padrões para linguagens usadas na automação industrial, justamente com o objetivo de ser um ambiente

aberto e programável em qualquer uma das cinco de linguagens padronizadas pela norma citada.

A sua comunicação com a maquete virtual é feita através do protocolo TCP/IP, sendo que as maquetes são desenvolvidas em *DirectX*[2].

As entradas e saídas são lidas por transferência de tabelas de dados, gravadas em arquivos *xml*[3]. O ambiente de programação lê essas tabelas de *I/O,* fato que ocorre constantemente durante o ciclo de varredura dessa comunicação. A lógica especificada pelo usuário é aplicada, obtendo os dados de entrada e atuando nas saídas. Os resultados são atualizados do APIMV para a maquete.

5.2. FUNCIONAMENTO DO AMBIENTE DE PROGRAMAÇÃO USANDO A LINGUAGEM SFC

O ambiente de programação em linguagem SFC está inserido no APIMV descrito no tópico anterior.

Há basicamente dois modos de funcionamento, modo simulação e modo conectado à maquete.

No modo simulação, o usuário cria suas variáveis internas, de entrada e de saída no dicionário, podendo "conectar" no máximo dezesseis entradas digitais, dezesseis saídas digitais, dezesseis entradas analógicas e dezesseis saídas analógicas. Este número de entradas e saídas foi definido devido à implementação usada no desenvolvimento do *software .*

[2] É uma interface de programação onde se pode interagir com elementos gráficos, placas de som e dispositivos de entrada.

[3] *Extensible Markup Language* (linguagem extensível de formatação), padrão usado para marcação de documentos que contêm informações estruturadas. Veio para superar as limitações do HTML, padrão das páginas da internet.

As entradas digitais são representadas por botões com *leds* que se acendem quando algum deles é acionado. As saídas digitais são representadas por *leds*, que também se acendem.

Já as entradas analógicas são *textbox* onde o usuário entra com um determinado valor e aperta seu botão equivalente. Finalmente as saídas mostram o valor, também através de um *textbox*. Neste modo não há conexão com a maquete virtual.

No caso do ambiente conectado à maquete, o usuário carrega as tabelas com as variáveis provenientes da mesma, e para cada passo e transição o usuário cria sua lógica, definindo ações e condições para tais variáveis.

Caso o usuário, ao escrever sua lógica dentro de um passo ou transição, use uma variável que não esteja na lista provida pela maquete, ou use alguma sintaxe equivocada, ele é notificado deste erro.

Se ele montar o diagrama SFC de maneira errada, como por exemplo, colocar um passo após outro passo, ou uma transição seguida de transição, caso o botão para simular, ou o botão para executar o programa sejam acionados, este erro é acusado e listado na tela, não permitindo assim a simulação ou execução até que todos os erros sejam corrigidos.

O arquivo com as informações de uma determinada aplicação, como por exemplo, a disposição dos elementos na tela, os dados de cada um destes elementos, as tabelas de variáveis utilizadas, caminho para comunicação com a maquete virtual, pode ficar alocado em qualquer local do computador, pronto para ser carregado no ambiente.

5.3. ENTENDENDO A EXECUÇÃO DA LÓGICA SFC

O diagrama SFC básico, como já mostrado anteriormente na Figura 4 do capítulo três, é composto de passos e transições, podendo haver divergências e convergências. Os elementos básicos usados no ambiente de programação são os seguintes:

- Passo Inicial;
- Transição;
- Passo;
- E Inicial Divergente;
- E Divergente;
- E Convergente;
- E Inicial Convergente;
- Ou Inicial Divergente;
- Ou Divergente;
- Ou Convergente;
- Ou Inicial Convergente;
- E Horizontal;
- Ou Horizontal;
- Linha Vertical;
- *Jump*.

Passo Inicial, Transição, Passo e *Jump* equivalem obviamente aos elementos de mesmo nome.

O E Inicial Divergente é o primeiro elemento usado quando vai se fazer uma divergência com lógica E. O E Divergente é usado sempre a direita do E Divergente Inicial, são os demais elementos da divergência, podendo ser usados quantos elementos o usuário desejar.

O E Inicial Convergente é o primeiro da convergência com lógica E, e o E Convergente também é usado a sua direita, como o E Divergente. O Ou Inicial Divergente é similar ao E Inicial Divergente, só que para a lógica OU, o mesmo acontecendo para os demais elementos Ou Divergente, Ou Inicial Convergente e Ou Convergente.

E têm-se também os fios de conexão. O E Horizontal é uma linha usada tanto nas divergências com lógica E, como nas convergências, apenas para se alongar horizontalmente o diagrama. O mesmo ocorre com o Ou Horizontal, usado nas divergências e convergências com lógica OU. E a Linha Vertical é usada para se estender o diagrama verticalmente.

Quando um programa é executado, tanto em modo conectado à maquete, quanto em modo simulação, o passo inicial inicialmente fica da cor azul, indicando que o mesmo está ativo naquele momento. Assim, a lógica definida pelo usuário para aquele estado é executada. O programa checa então a condição imposta na transição seguinte. Se for satisfeita, o passo inicial se torna inativo e o estado seguinte é ativado, ocorrendo o mesmo com ele, ou seja, a execução de sua lógica interna. E assim se desenrola todo o diagrama SFC.

Na Tabela 1, pode-se observar todos os elementos citados acima.

Tabela 1 - Elementos SFC

ELEMENTOS SFC		
 PASSO INICIAL	 PASSO	 TRANSIÇÃO
 E INICIAL DIVERGENTE	 E DIVERGENTE	 E CONVERGENTE
 E INICIAL CONVERGENTE	 OU INICIAL DIVERGENTE	 OU DIVERGENTE
 OU CONVERGENTE	 OU INICIAL CONVERGENTE	 E HORIZONTAL
OU HORIZONTAL	LINHA VERTICAL	 JUMP

A lógica SFC é uma lógica seqüencial. Quando a execução é iniciada, a primeira ação ocorrida é a ativação do passo inicial. Quando o mesmo é ativado, as ações associadas a este estado serão executadas. Por exemplo, na Figura 16, o usuário determinou que quando o passo inicial é ativado, há uma ação S na variável *q_down*, ou seja, seu estado passará para TRUE.

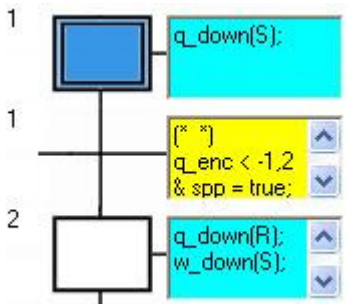


Figura 16 - Lógica SFC (Passo inicial ativo)

Ao interpretar a ação executada, dada a ativação daquele passo, o estado daquela variável é atualizado e enviado à maquete. No caso, a ação referente à *q_down* se tornar TRUE é executada na maquete virtual.

A tabela de variáveis é constantemente lida pelo ambiente de programação, como já dito anteriormente, há a execução da lógica, e finalmente, o envio da tabela atualizada para a maquete. Esta leitura e escrita constantes, ou seja, o ciclo de varredura, são feitas repetidas vezes, em partes, com o auxílio de uma *thread*[4].

Quando o estado fica ativo, a transição ou caso houvesse uma seqüência divergente, as transições seguintes, também são checadas. No caso da Figura 16, a condição estabelecida é *q_enc < -1,2* e *spp = true*. O ambiente fica recebendo as informações referentes às variáveis *q_enc* e *spp*, e verifica se esta condição imposta será satisfeita.

Caso seja satisfeita, o passo inicial se torna inativo, e o passo 2 é ativado (Figura 17), e novamente, as ações referentes a este novo estado são executadas. As variáveis *q_down* e *w_down* terão seus estados alterados. No exemplo, *q_down* sofre a ação R, ou seja, se seu estado era TRUE, agora se torna FALSE, e *w_down* sofre a ação S, que muda seu estado para TRUE.

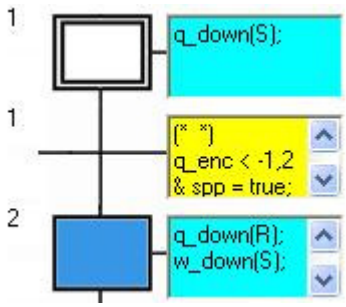


Figura 17 - Lógica SFC (Passo 2 ativo)

[4] Fluxo de controle seqüencial isolado dentro de um programa. Como um programa seqüencial qualquer, uma *thread* tem começo, fim e uma seqüência de comandos, mas não roda sozinha, e sim "dentro" de um programa. Permite que um programa simples possa executar várias tarefas ao mesmo tempo, independentemente umas das outras.

Novamente, a tabela com os dados atualizados é passada à maquete. E assim ocorre durante toda a execução da lógica SFC.

5.4. MANIPULAÇÃO DOS ELEMENTOS

Um diagrama de elementos SFC é montado do seguinte modo: o usuário dá um clique simples no botão referente ao elemento que quer usar no diagrama. O espaço em que os elementos ficam disponíveis internamente é visto como uma grande matriz. A Figura 18 mostra o que acontece quando o usuário clica num dos botões, o elemento sempre vai pra célula de coordenada (0,0).

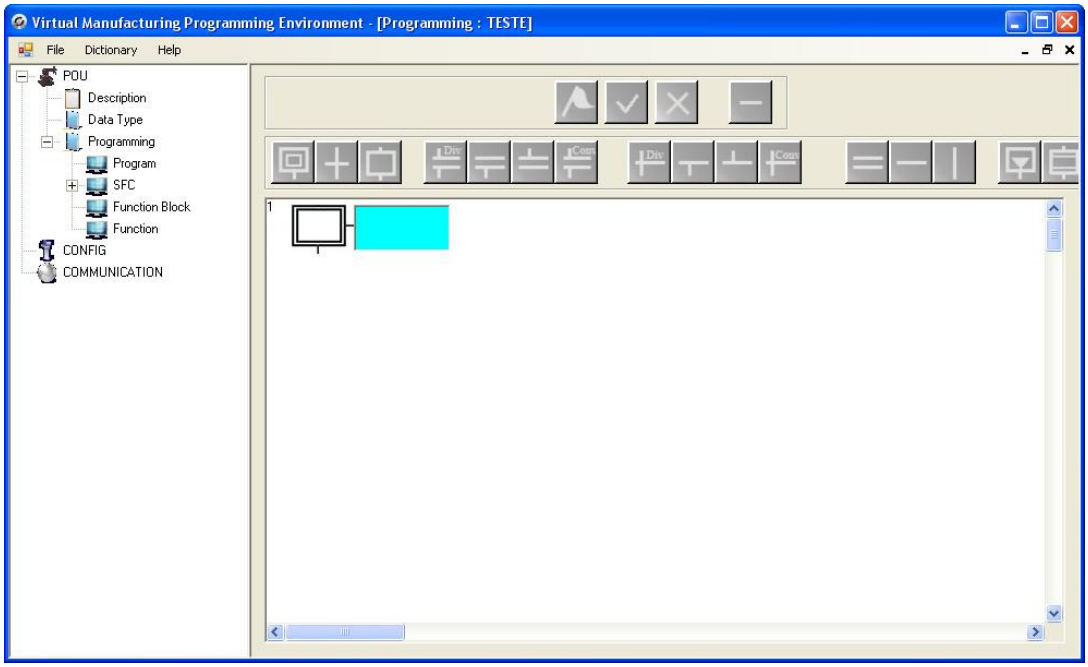


Figura 18 - Inserindo um elemento

Internamente, o programa é apresentado na Figura 19.

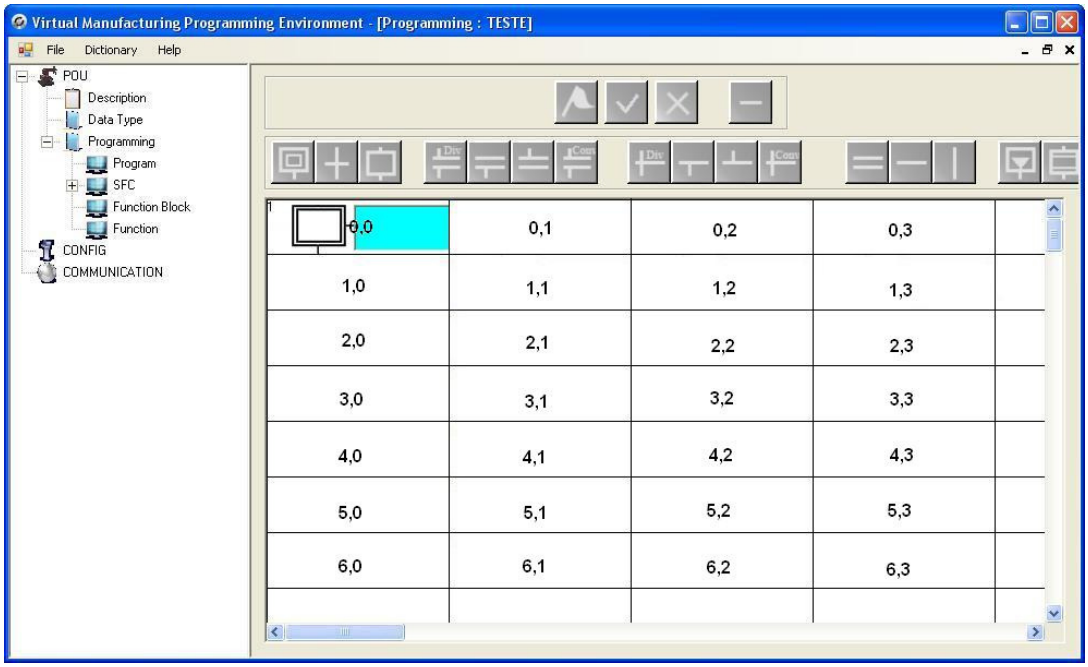


Figura 19 - Matriz de elementos

Inicialmente, há dois modos de movimentar o elemento. Ou o usuário efetua um clique simples e arrasta o elemento até o ponto desejado, ou ele clica neste ponto, e o elemento irá diretamente da posição (0,0) até a posição escolhida.

Já posicionado, este elemento só poderá ser deslocado através do clique e arrasto.

5.4.1. Lógica de Fixação do Elemento

O elemento SFC sempre é posicionado corretamente nas células mostradas na Figura 19, no momento em que ele é deslocado pelo usuário, ou seja, o programa calcula automaticamente a posição correta e o fixa nesta posição.

Todos os elementos obviamente têm o mesmo tamanho (Figura 20).

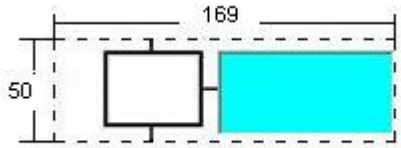


Figura 20 – Modelo de elemento SFC

Caso o elemento esteja na posição (230, 70), por exemplo, como na Figura 21, o programa divide as duas coordenadas respectivamente por 169 e 50, chegando a dois pontos (X, Y):

$$X = 230 / 169 = 1,36$$

$$Y = 70 / 50 = 1,4$$

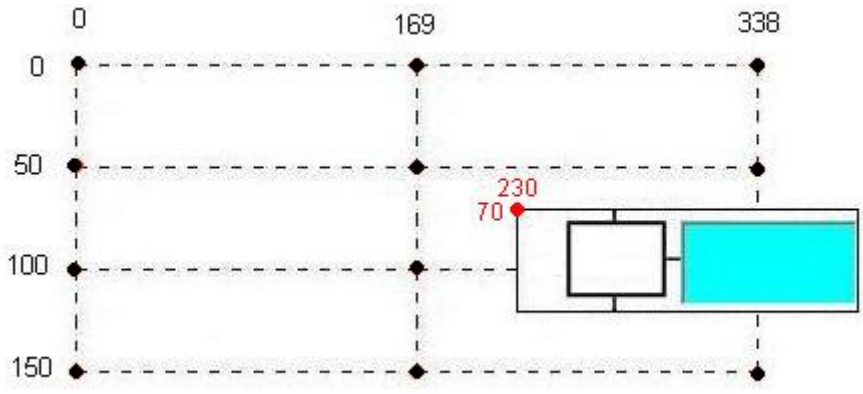


Figura 21 - Fixação do elemento antes da correção

Logo, X = 1 e Y = 1, pois recebem apenas a parte inteira da resposta. Finalmente, há a multiplicação de X e Y por 169 e 50, o que ocasiona na fixação correta do elemento na tela. Pode-se observar na Figura 22.

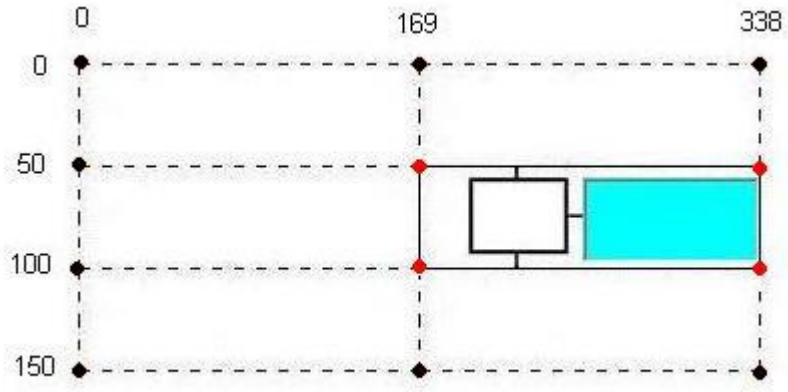


Figura 22 - Fixação do elemento após correção

Assim, o elemento permanece fixado na posição correta, e ainda sabem-se suas coordenadas x e y, no caso (1,1).

Se o usuário soltar o elemento numa posição onde já existe outro elemento, o programa “joga” o elemento que está sendo posicionado, para o próximo espaço abaixo disponível.

5.4.2. Comportamento dos Elementos na Execução do Programa

Antes de iniciar a execução do programa, todos os elementos apresentam a cor branca. Dada a execução, inicialmente o passo inicial é ativado e passa da cor branca para a azul, como na Figura 23.

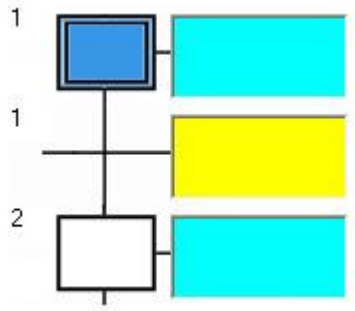


Figura 23 - Passo inicial ativo

Neste momento, como já explicado anteriormente neste capítulo, todas as ações associadas a este estado ou passo são executadas.

Quando satisfeita a transição seguinte ao passo inicial, este é desativado e o novo estado ativo é o passo 2 (Figura 24).

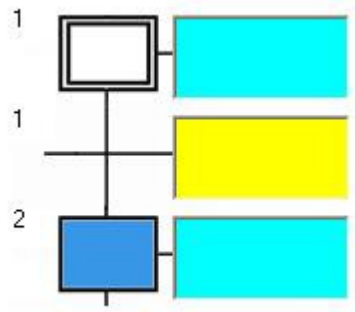


Figura 24 - Passo inicial inativo e passo 2 ativo

Os únicos elementos que alternam suas cores, no momento da execução, são o passo inicial e os demais passos.

5.5. GRAVAÇÃO DO PROGRAMA

Uma aplicação dentro do ambiente de programação pode ser gravada no computador, e também carregada, através de um arquivo de extensão .SFC. Todas as informações referentes a esta aplicação são salvas num arquivo *xml*.

O arquivo .SFC apresenta o formato descrito na Tabela 2.

Tabela 2 - Formato do arquivo .SFC

ATRIBUTOS	DESCRIÇÃO	TIPO
connected	Informa se há conexão com a maquete	bool
ConnectionString	Endereço do servidor da maquete	string
Description	Descrição da aplicação em questão	
MyClient	Cliente da conexão	Client
MySFCPss	Nome da aplicação, contadores e elementos	clsSFCCollections
MySFCTables	Nome e tabelas para modo simulação	ClsSFCTables
MySFCRealTables	Nome e tabelas para modo conectado a maquete	ClsSFCRealTables

Dentro do atributo *MySFCPss* há os campos apresentados na Tabela 3.

Tabela 3 - Atributos de MySFCPss

ATRIBUTOS	DESCRIÇÃO	TIPO
Name	Nome da aplicação	string
cj	Contador de jump	int
cms	Contador de macro step	
cmsf	Contador de macro step final	
cmsi	Contador de macro step inicial	
cp	Contador de passo	
cpi	Contador de passo inicial	
ct	Contador de transição	
sElementDatas	Lista de elementos SFC	clsSFC_Element

sElementDatas é uma lista que contém todos os elementos SFC que estão dispostos na tela, sendo que cada elemento com seus atributos (Tabela 4).

Tabela 4 - Atributos dos elementos SFC

ATRIBUTOS	DESCRIÇÃO	TIPO
nlinha	Coordenada vertical	int
ncoluna	Coordenada horizontal	
colreferencia	Coluna de referência para montagem da árvore	
sfcimageindex	Índice da figura usada no elemento	
id	Identidade do elemento	
contadorpi	Contador de passo inicial	
contadorp	Contador de passo	
contadort	Contador de transição	
contadormsi	Contador de macro step inicial	
contadorms	Contador de macro step	
contadormsf	Contador de macro step final	
contadorj	Contador de jump	
contadorgeral	Contador geral de elementos	
contdivconv	Contador de convergências	
contelediv	Contador de divergências	
jumplinha	Coordenada vertical do elemento apontada pelo jump	
jumpcoluna	Coordenada horizontal deste elemento	
idconv	Identidade do elemento que converge	
iddedivergencia	Identidade do elemento que diverge	
sfctooltipText	Tooltip para os elementos	string
sfcname	Nome do elemento	
sfcnick	Apelido do elemento	
tipo	Tipo do elemento	
elemento	Indica se o elemento é PI,P,T entre outros	
nomeapelido	Apelido do elemento	
sfclogica	Indica a lógica escolhida para o elemento	
texto	Texto que fica na caixa à direita do elemento	
jump	Elemento apontado pelo jump	
apontajump	Elemento apontado pelo jump	
daumjump	Indica se há ou não um jump	bool
estaconvergindoe	Indica se está havendo convergência E	
estaconvergindoou	Indica se está havendo convergência OU	
estadivergindoe	Indica se está havendo divergência E	
estadivergindoou	Indica se está havendo divergência OU	
estaconvergindo	Indica se está havendo convergência	
estadivergindo	Indica se está havendo divergência	
primeiroadivergir	Indica o primeiro elemento a divergir	
eoprimeiro	Indica se é o primeiro elemento a divergir	
ActionN	Indica se há ação N	
apontapara	Lista de elementos apontados por um jump	List<string>
eapontadopor	Lista de elementos que estão apontando	
estrepetidos	Lista de estados repetidos na montagem da árvore	List<clsSFCState>
transrepetidos	Lista de transições repetidas na montagem da árvore	List<clsSFCTrans>
LisnatN	Lista de elementos onde há ação N	ArrayList

O atributo *MySFCTables* contém os atributos listados na Tabela 5. É usado para o modo simulação.

Tabela 5 - Atributos de MySFCTables

ATRIBUTOS	DESCRIÇÃO	TIPO
FileName	Nome do arquivo de gravação	string
InputTable	Lista de variáveis de entrada	List<clsSFCTable>
OutputTable	Lista de variáveis de saída	
InnerTable	Lista de variáveis internas	

As três listas citadas na Tabela 5, *InputTable*, *OutputTable* e *InnerTable* são as listas de variáveis e cada variável apresenta os campos da Tabela 6.

Tabela 6 - Atributos das variáveis no modo simulação

ATRIBUTOS	DESCRIÇÃO	TIPO
status	Status da variável	object
ivalue	Valor inicial da variável	
ação	Informa a ação sobre aquela variável	string
datatype	Tipo da variável	
direction	Direção da variável	
text	Comentário sobre a variável	
categoria	Categoria da variável	
connected	Informa se a variável está conectada	
index_var	Índice da variável	
conflito	Indica se já há variáveis com o mesmo nome	bool

O atributo *MySFCRealTables* contem os mesmo campos vistos na Tabela 5, mas é usado para o modo conectado à maquete. As variáveis, neste caso, apresentam os atributos da Tabela 7.

Tabela 7 - Atributos das variáveis no modo conectado à maquete

ATRIBUTOS	DESCRIÇÃO	TIPO
labels	Nome da variável	string
status	Status da variável	
ivalue	Valor inicial da variável	
datatype	Tipo da variável	
direction	Direção da variável	
text	Comentário sobre a variável	

Este arquivo .SFC poderá também ser carregado no programa, que utiliza todas as informações disponíveis e o interpreta de modo que a aplicação seja aberta na tela exatamente do mesmo modo que foi salva.

Caso não haja o preenchimento de alguns campos, a aplicação ainda pode ser salva.

5.6. LÓGICA DE EXECUÇÃO

Na execução, estão envolvidas, basicamente, quatro classes:

- Classe *clsSFCGrafo*, ou classe grafo;
- Classe *clsSFCState*, ou classe estado;
- Classe *clsSFCTrans*, ou classe transição;
- Classe *clsSFC_Element*, ou classe elemento.

Na classe *clsSFCGrafo*, é onde o grafo de elementos começará a ser montado pra ser corretamente percorrido e o programa bem executado.

A classe *clsSFC_*Element é a classe com todos os atributos dos elementos SFC que estão dispostos na tela.

A classe *clsSFCState*, ou classe estado, contém três atributos principais, o elemento SFC, do tipo *clsSFC_Element*, sua transição ou suas transições seguintes e sua ou suas transições anteriores.

clsSFCTrans também contém um atributo elemento, bem como passo(s) seguinte(s), e passo(s) anterior(es) .

Pode-se tomar com exemplo o diagrama da Figura 25.

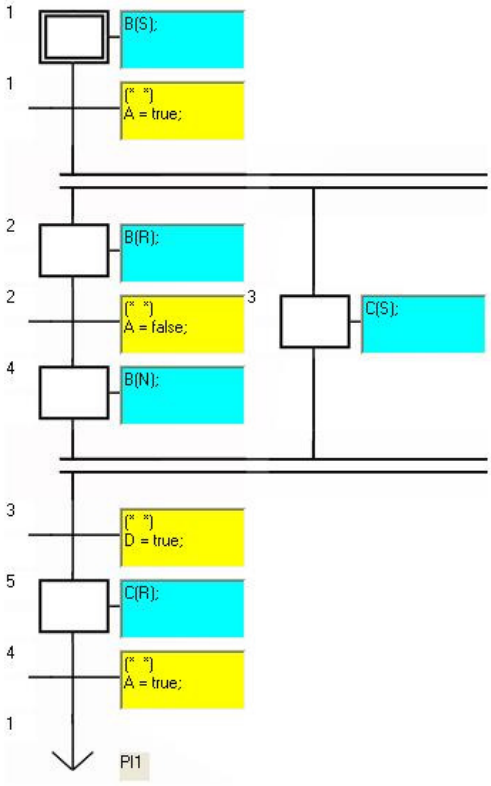


Figura 25 - Diagrama SFC - Montagem do grafo de elementos

À medida que o usuário adiciona elementos SFC na tela, estes elementos vão sendo também adicionados numa lista de elementos. E esta lista é ordenada pela coordenada Y de cada elemento, sendo o primeiro elemento aquele que apresenta o menor Y, no caso da Figura 25, P1. Caso haja mais de um elemento numa mesma linha, o que está mais a esquerda é adicionado primeiro na referida lista. Há também a criação de uma segunda lista, clone da primeira, que se mantém inalterada, para que quando ocorra um caso de *jump,* por exemplo, a localização do elemento apontado possa ser pesquisada nesta segunda lista.

Na classe *clsSFCGrafo*, inicializamos uma instância da classe *clsSFCState* passando como parâmetro a lista de elementos, e tem-se que o estado inicial ou E_0 é o primeiro elemento da lista. O trecho do código a seguir que mostra esta situação:

```
E0 = new clsSFCState(lista_el);
E0.element = ((clsSFC_Element)lista_el[0]);
```

Este elemento é então retirado da lista, e dentro da classe *clsSFCState*, um método é chamado, para encontrar o próximo elemento:

```
lista_el.RemoveAt(0);
```

Caso o próximo elemento da lista seja uma linha vertical, por exemplo, este elemento também é retirado. Quando encontra T1, o mesmo ocorre, só que na classe *clsSFCState*, e uma lista de transições é criada:

```
t = new clsSFCTrans(lista_el);
t.element = ((clsSFC_Element)lista_el[0]);
transicao = new List<clsSFCTrans>();
transicao.Add(t);

lista_el.RemoveAt(0);
```

No exemplo da Figura 25, o próximo elemento da lista é um E divergente inicial. Neste caso, o programa fica sabendo que há uma divergência. Então, pula os demais elementos daquela linha e vai para o primeiro da linha seguinte, P2:

```
s = new clsSFCState(lista_el);
s.element = ((clsSFC_Element)lista_el[0]);
next_state = new List<clsSFCState>();
next_state.Add(s);

lista_el.RemoveAt(0);
```

Repete-se o procedimento para T2 e P4, e chega-se ao E convergente inicial. Fica guardada então a informação sobre o passo anterior à convergência (P4).

O próximo elemento é T3. O procedimento para transições ocorre novamente, bem como para P5 (procedimento para estados) e T4.

E assim é feito até o último elemento da primeira coluna, o *jump*. Quando se encontra um *jump*, têm-se informações do elemento para o qual ele está apontando

(P1), como o tipo, nome e suas coordenadas. Faz-se então uma busca na lista original e repetem-se os procedimentos descritos acima.

O programa retorna então até o ponto E divergente inicial e passa a analisar a partir do E divergente a sua direita. No exemplo, o próximo elemento encontrado é o P3. Quando se chega ao E convergente, o programa sabe que a transição seguinte é a mesma do último passo (P4) antes do E convergente inicial, pois esta informação foi armazenada.

O resultado dessa "jornada" por todo o diagrama mostrado na Figura 26.

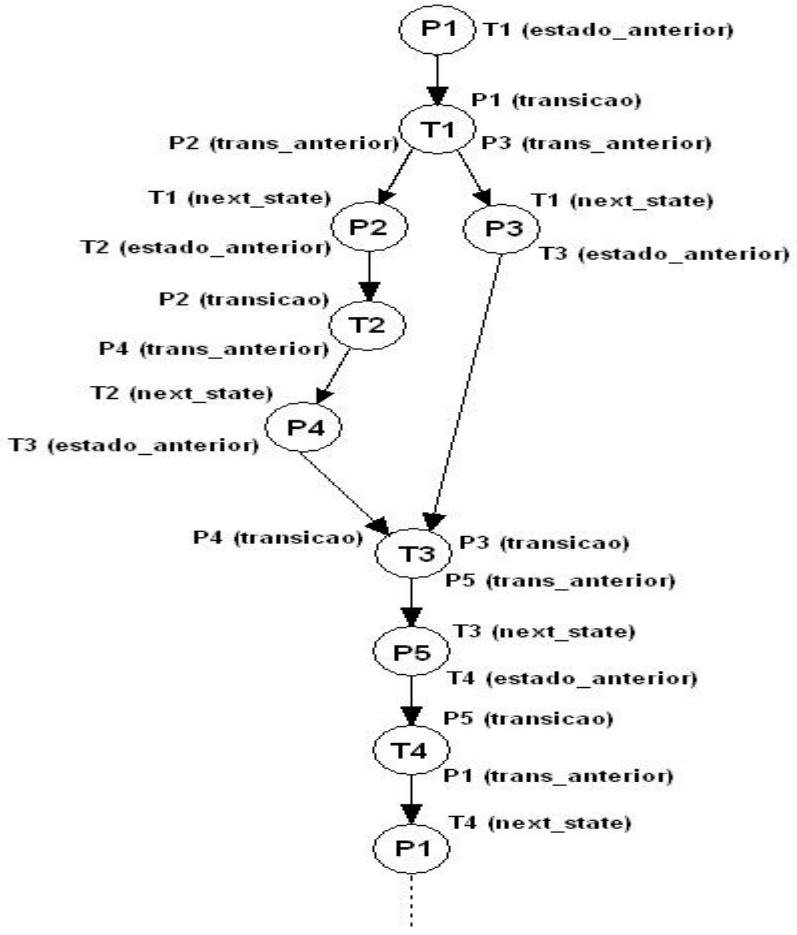


Figura 26 - Grafo de elementos

Portanto, o programa terá informações sobre cada passo e cada transição do diagrama. Neste exemplo, são as seguintes informações:

- P1 é estado anterior de T1;
- T1 é transição seguinte de P1;
- T1 é transição anterior de P2 e P3;
- P2 e P3 são estados seguintes de T1;
- P2 é estado anterior de T2;
- P3 é estado anterior de T3;
- T2 é transição seguinte de P2;
- T2 é transição anterior de P4;
- P4 é estado seguinte de T2;
- P4 é estado anterior de T3;
- T3 é transição seguinte de P3 e P4;
- T3 é transição anterior de P5;
- P5 é estado seguinte de T3;
- P5 é estado anterior de T4;
- T4 é transição seguinte de P5;
- T4 é transição anterior de P1;
- P1 é estado seguinte de T4;

E assim fecha-se o ciclo. O caminho percorrido pelo programa para montar o grafo de elementos está explicitado na Figura 27.

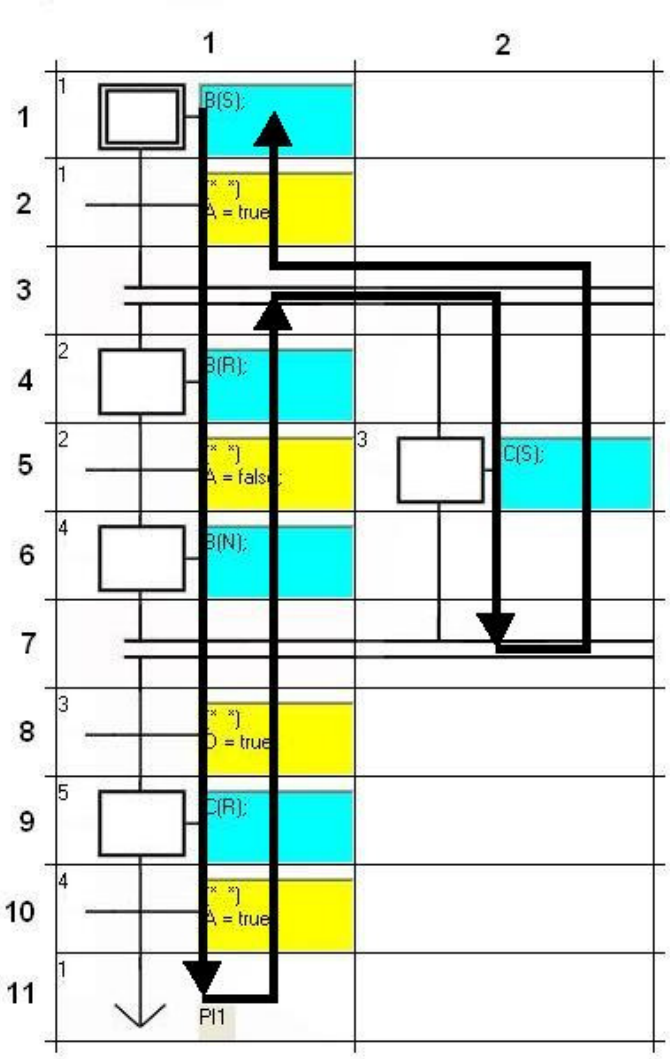


Figura 27 - Caminho percorrido para montagem do grafo de elementos

Começa então a execução do programa, ou seja, entra em ação uma classe para analisar a lógica imposta pelo usuário para P1, no caso, a variável de saída B sofre a ação S. Chama-se então a classe que cuida da parte das ações que os estados irão sofrer. As tabelas que irão ser enviadas à maquete, no caso do modo conectado à maquete, serão atualizadas e enviadas, e a cor do elemento P1 muda para azul, quando ativado. Quando desativado, volta para a cor branca, e P2 e P3 se tornam azuis, e assim sucessivamente.

A compilação da lógica para as transições e estados será tratada posteriormente, e também a conexão do ambiente com a maquete virtual.

5.7. COMUNICAÇÃO COM A MAQUETE

Como já afirmado anteriormente, há dois modos de funcionamento, simulação e conectado à maquete. No segundo modo, o ambiente de programação conecta-se à maquete por meio de um *driver* de protocolo TCP/IP [16], o que permite acesso direto a um endereço local ou remoto.

A localização da maquete é informada pelo usuário, no ambiente de programação, pelo nome ou endereço IP da máquina.

A arquitetura usada é a Cliente-Servidor, sendo que o ambiente de programação é o cliente, e a maquete faz o papel do servidor (Figura 28).

Nesta conexão, estão envolvidas tabelas de dados. O ambiente recebe da maquete uma tabela de sensores, uma tabela de ações, tabelas de variáveis de entrada, saída e variáveis internas.

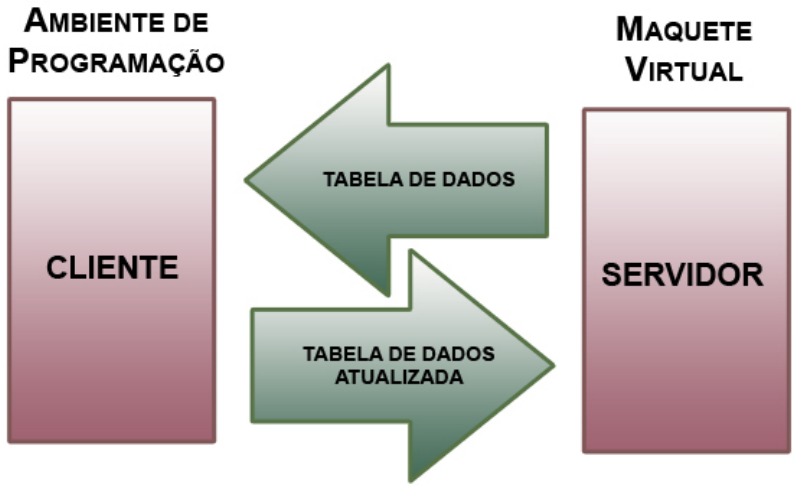


Figura 28 - Conexão entre maquete e ambiente de programação

O ambiente de programação recebe um pacote de dados. Um exemplo de tabelas que são enviadas pela maquete virtual pode ser observado nas Tabelas 8 e 9.

Tabela 8 - Tabela de sensores enviada ao ambiente de programação

SENSORS		
ATRIBUTOS	**STATUS**	**TIPO**
activate	false	bool
AllFace	false	
toExport	true	
hasAction	false	
Name	SFC1F	string
Element1	“furad_fura_festo”	
Element2	“sensorfuradeira”	
s1	null	Sensor
s2	null	

Tabela 9 - Tabela de ações enviada ao ambiente de programação

RACTIONSSTATE		
ATRIBUTOS	**STATUS**	**TIPO**
isDown	false	bool
isUp	false	
isOn	false	
action	“q”	string
Delta	0.0	float
delta	0.0	
oldValue	0.0	
value	0.0	
Value	0.0	

Os dados destas tabelas são organizados numa tabela com variáveis de entrada, saída e variáveis internas, como nas Tabelas 10 e 11, que mostram as entradas e saídas. Suas descrições já foram feitas na Tabela 7 da seção 5.5.

Tabela 10 - Variáveis de entrada

INPUTTABLE		
ATRIBUTOS	**STATUS**	**TIPO**
labels	q_enc	string
status	False	
ivalue	00,000	
datatype	FLOAT	
direction	0	
text	W encoder	

Tabela 11 - Variáveis de saída

OUTPUTTABLE		
ATRIBUTOS	**STATUS**	**TIPO**
labels	q_up	string
status	False	
ivalue	00,000	
datatype	BOOL	
direction	1	
text	Furadeira para cima	

Resumindo, o ambiente recebe as tabelas *Sensor*s e *RActionState* da maquete. Lê os dados de *Sensors* e atua nos dados de *RActionState* e envia estes dados já atualizados de *RActionState* de volta à maquete virtual.

5.8. COMPILADOR DA LÓGICA PARA ESTADOS E TRANSIÇÕES

Dentro de cada passo e transição, o usuário escolhe, através de qualquer uma das cinco linguagens padronizadas pela Norma IEC61131-3, as ações para os passos e as condições para as transições. No caso deste trabalho, o compilador foi desenvolvido para a linguagem Texto Estruturado.

Na Figura 29 tem-se um exemplo simples que mostra as ações e condições para uma aplicação em SFC.

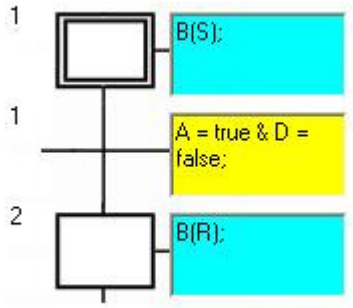


Figura 29 - Exemplo de ações e condições a serem compiladas

O programa, ao ativar o passo 1, irá verificar a ação associada a este estado. No caso, a variável de saída B sofre a ação *set*, ou seja, seu status passará de *false* para *true.*

O compilador criado funciona do seguinte modo: são criados inicialmente os chamados *tokens*, ou tipos de elementos de uma expressão:

- Inicio;
- End;
- Comparador;
- Assign;
- Operador;
- OperadorBooleano;
- Operando;
- AbreParentese;
- FechaParentese;
- PontoEVirgula;
- Separador;
- SeparadorFor;
- PassoFor;
- ActionBoolN;
- ActionBoolS;
- ActionBoolR;
- PulseAction;

- NonStoredAction;
- ParenteseEspecial.

Há um atributo para cada elemento SFC chamado *sfclogica*, que é uma *string*, onde fica armazenada a lógica, no exemplo “B(S);”. Desta *string*, o programa constrói uma expressão pós-fixa, analisando cada caractere.

Inicialmente é inserido no final da *string* o caractere “#”, para indicar o fim da mesma. O primeiro caractere lido é o “B”, e o mesmo é inserido numa pilha de dados, e retirado da expressão original. O programa verifica então que B é um operando e irá checar se este é valido, analisando se ele pertence à tabela de variáveis de saída. Caso não fosse válido, uma mensagem de erro seria mostrada na tela.

O próximo caractere é “(“. Quando encontrado um caractere deste tipo, o programa analisa o caractere seguinte. Se for o equivalente a uma ação, S, R ou N, o programa junta “(“ com “S”, no caso e verifica o seguinte. Sendo “)”, é inserido na pilha o termo “(S)”.

O caractere restante é o “;”. Este não é inserido na pilha, e finalmente “#” é lido, indicando que a expressão chegou ao fim.

Temos então uma pilha de termos que será analisada e executada.

Deve-se ater à lógica da transição, para um melhor entendimento. A *string* deste elemento é:

```
“A = true & D = false;”
```

O programa internamente substitui “&” por “AND” e chega à seguinte expressão:

```
expression = “A = true AND D = false;#”;
```

Construir-se-á então a pilha de dados a partir de *expression*. O primeiro caractere é “A”, e como afirmado anteriormente, o programa vê que “A” é um operando e checa sua validade, agora na tabela de variáveis de entrada, e o insere na pilha (Figura 30).

Figura 30 - Pilha de dados

Agora, a expressão é:

```
expression = “= true AND D = false;#”;
```

O caractere seguinte é “=”. É do tipo Comparador, e por isso é inserido numa pilha auxiliar, a pilha de operadores (Figura 31).

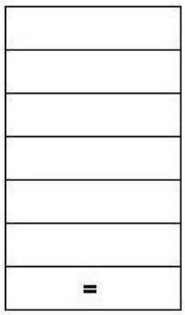


Figura 31 - Pilha de operadores

Tem-se então:

```
expression = “true AND D = false;#”;
```

“t” é o caractere da vez, e para o programa, ele é um operando. Assim, parte para o próximo até encontrar um espaço em branco, formando o termo “*true*”, que é inserido na pilha de dados (Figura 32).

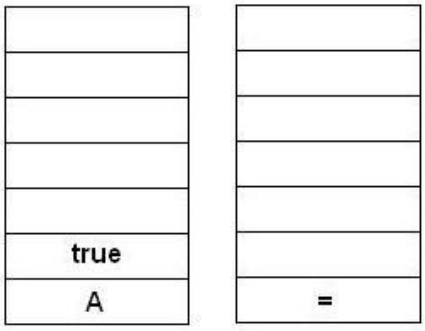


Figura 32 - Pilhas de dados e de operadores

Seguindo, tem-se:

```
expression = "AND D = false;#";
```

O caractere "A" é verificado, e como é considerado um operando, vai-se novamente até chegar-se a uma lacuna, resultando no termo "AND". É do tipo OperadorBooleano.

O programa checa agora a pilha de operadores, que já contém "=" e compara a prioridade deste termo com "AND". Há duas verificações de prioridades, a antes e a depois. A antes é a do elemento que já está na pilha de operadores e a depois, do elemento em questão. Por ter maior prioridade antes, o termo "=" é retirado da pilha de operadores e inserido na pilha de dados, e "AND" é colocado na pilha de operadores, como mostra a Figura 33.

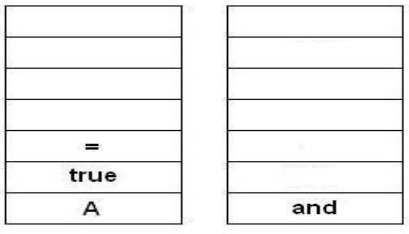


Figura 33 - Pilhas de dados e de operadores

Neste momento, a expressão base já está do seguinte modo:

```
expression = "D = false;#";
```

O mesmo que aconteceu com o termo “A”, acontece para “D”. Portanto, a pilha de dados fica como na Figura 34.

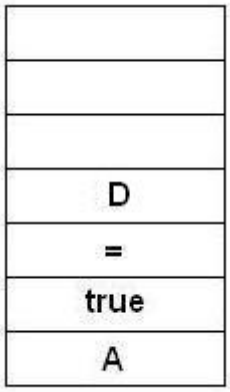


Figura 34 - Pilha de dados

A expressão fica:

```
expression = “= false;#”;
```

Tem-se então “=” novamente. Agora a prioridade antes do elemento “AND”, que está na pilha de operadores é igual à prioridade depois do termo “=”. Por esta razão, “=” vai para esta pilha, ficando (Figura 35).

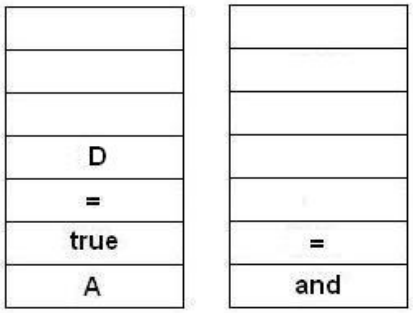


Figura 35 - Pilhas de dados e de operadores

Neste momento:

```
expression = “false;#”;
```

O procedimento para o termo “*true*” ocorre para “*false*”, resultando na pilha da Figura 36.

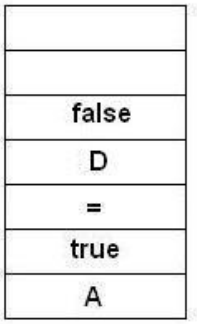


Figura 36 - Pilha de dados

Chega-se a:

```
expression = ";#";
```

Como o próximo termo é do tipo PontoEVirgula, o programa desempilha a pilha de operadores e empilha estes dados na pilha de dados. Da pilha de operadores da Figura 35, chega-se à seguinte pilha de dados da Figura 37.

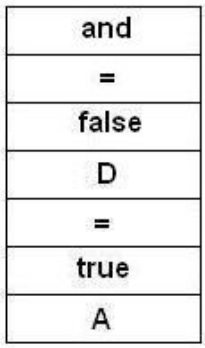


Figura 37 - Pilha final de dados

Quando se chega ao termo final “#”, a pilha de dados é desempilhada e, por conseguinte, empilhada numa pilha chamada *Expressao* (Figura 38), a qual será usada para se executar a lógica.

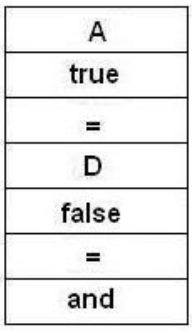


Figura 38 - Pilha Expressao

O programa vai então para um método chamado *AplicaTrans()*, onde cada elemento da pilha é analisado.

O primeiro elemento é o operando A. Ele é desempilhado e inserido numa pilha chamada *Variaveis*, e o seu valor, no caso *false* é inserido numa pilha *Operandos*. E há uma quarta pilha *Condicoes*, onde é inserido sempre o termo *true*. Isto pode ser observado na Figura 39.

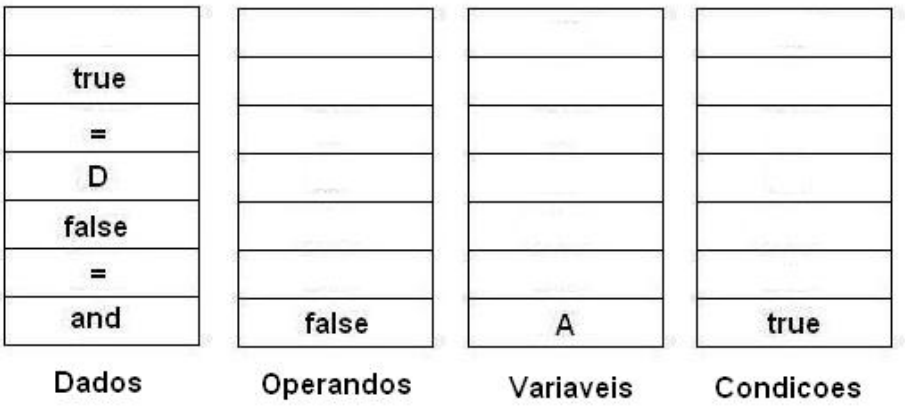


Figura 39 – Pilhas

Posteriormente, o programa analisa o elemento "*true*" da pilha de dados. Também é um operando, e as pilhas ficam como na Figura 40.

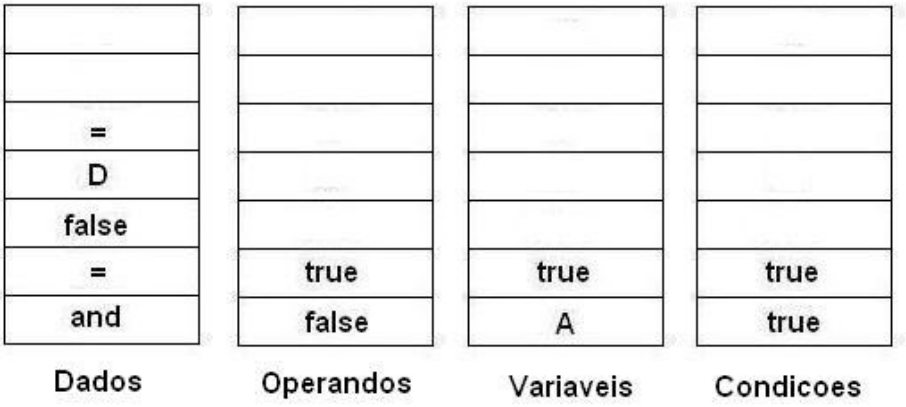


Figura 40 – Pilhas

Como o termo seguinte é "=", o topo de *Variaveis* é desempilhado, inicialmente. E os dois termos de *Operandos* também são desempilhados, comparando-se um termo com o outro, pois estamos dentro de uma transição. No caso *true* é diferente de *false.* Logo, a condição não é satisfeita momentaneamente, e é empilhado por isso o termo *false* em *Operandos*, como também se empilha *false* em *Variaveis*. Tem-se então as pilhas da Figura 41.

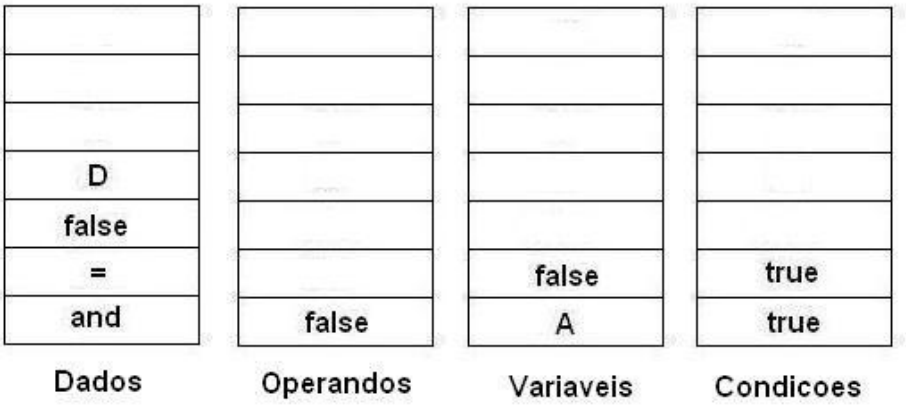


Figura 41 – Pilhas

Neste momento, o elemento D é desempilhado. Ele é um operando e seu status é *false*. As pilhas ficam então como na Figura 42.

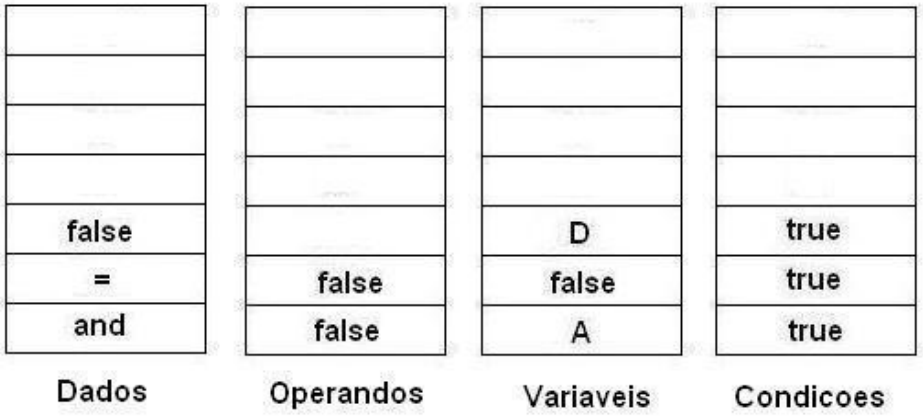


Figura 42 – Pilhas

O próximo termo analisado é *false*. Ele sai da pilha de dados e vai para as pilhas *Operandos* e *Variaveis* (Figura 43).

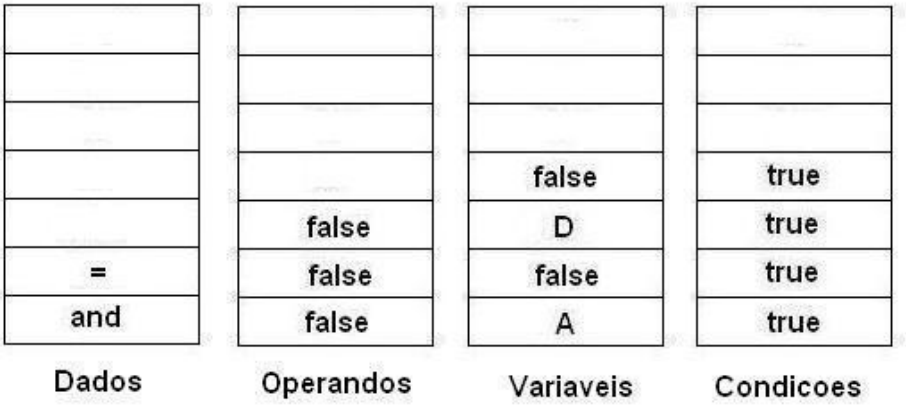


Figura 43 – Pilhas

Tem-se novamente um termo “=”. O topo de *Variaveis* mais uma vez é desempilhado, e compara-se o topo de *Operandos* com o termo seguinte, desempilhando-nos, no caso *false* com *false*. Como são iguais, *true* é empilhado nas duas pilhas em questão, chegando às pilhas da Figura 44.

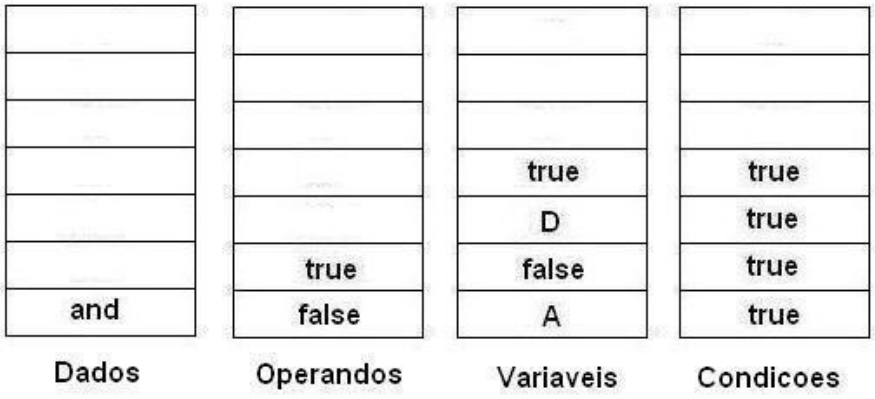


Figura 44 – Pilhas

Finalmente, tem-se o termo AND. O mesmo é desempilhado, e o programa compara os dois termos da pilha *Operandos*, o primeiro com o topo de *Condicoes*, e o segundo com o segundo termo de *Condicoes.* Ao mesmo tempo em que os dois termos de *Operandos* são desempilhados, os dois primeiros termos de *Variaveis* também o são.

Neste caso, como o há um termo *false* de *Operandos*, a condição da transição não foi satisfeita, pois é uma lógica E, ou seja, os dois termos teriam que ser *true*.

Todo o procedimento descrito acima é refeito até que a condição seja satisfeita, e o programa possa assim desativar o passo 1 e ir para o passo 2.

E assim é feita a compilação das lógicas escolhidas para os elementos num diagrama SFC.

5.9. MODOS DE FUNCIONAMENTO

Como já foi afirmado, o ambiente de programação em linguagem SFC trabalha em dois modos, conectado à maquete virtual e em modo de simulação.

5.9.1. Modo Simulação

Neste modo, o ambiente não está conectado à maquete. O usuário monta seu diagrama SFC e as variáveis de entrada, saída e internas são também criadas, dentro do dicionário, podendo ser até sessenta e quatro pontos, dezesseis entradas digitais, dezesseis entradas analógicas, dezesseis saídas analógicas e dezesseis saídas digitais.

Abaixo, pode-se ver o dicionário para a criação das variáveis (Figura 45).

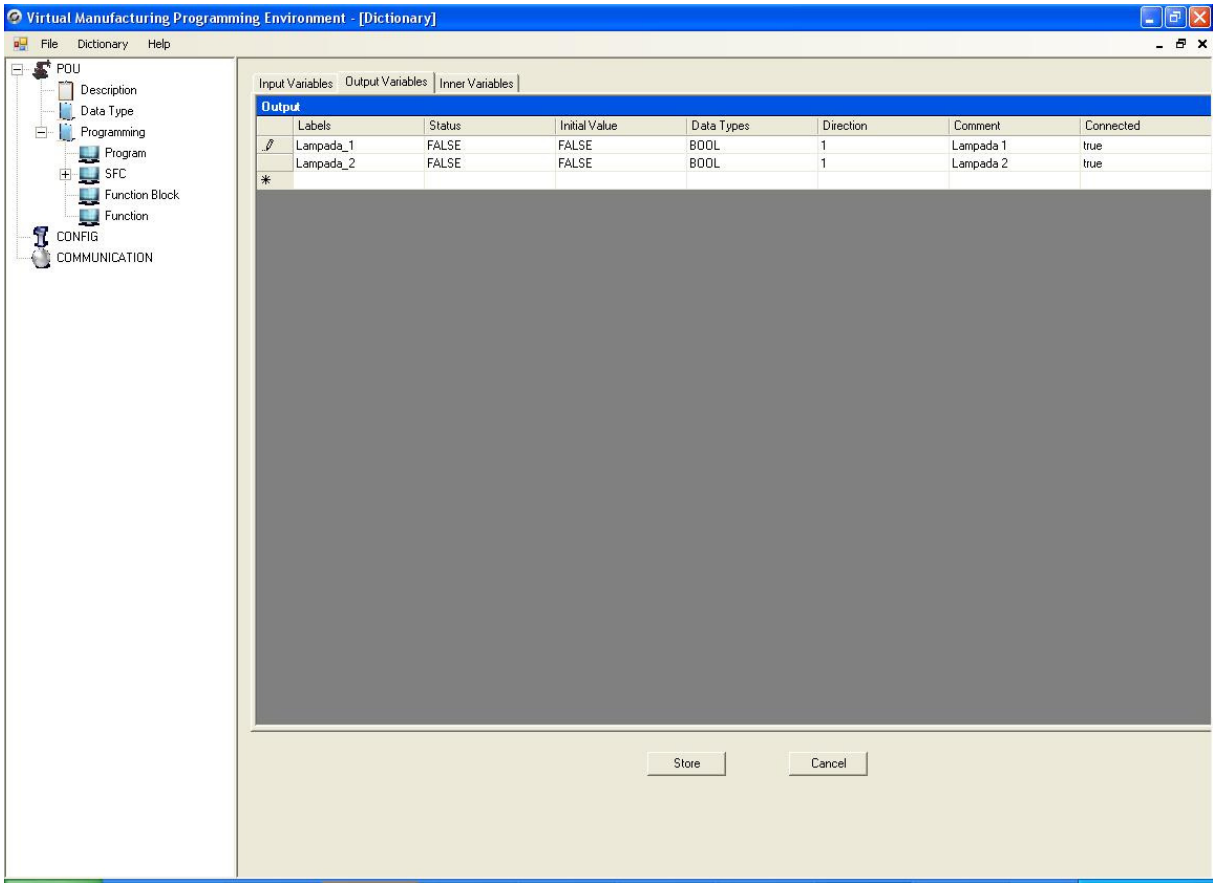


Figura 45 - Dicionário - modo simulação

No campo *Connected* é definido se a variável será conectada, e assim, se vai fazer parte da execução daquela aplicação.

Na Figura 46, são listadas as entradas e saídas conectadas, no caso, as digitais

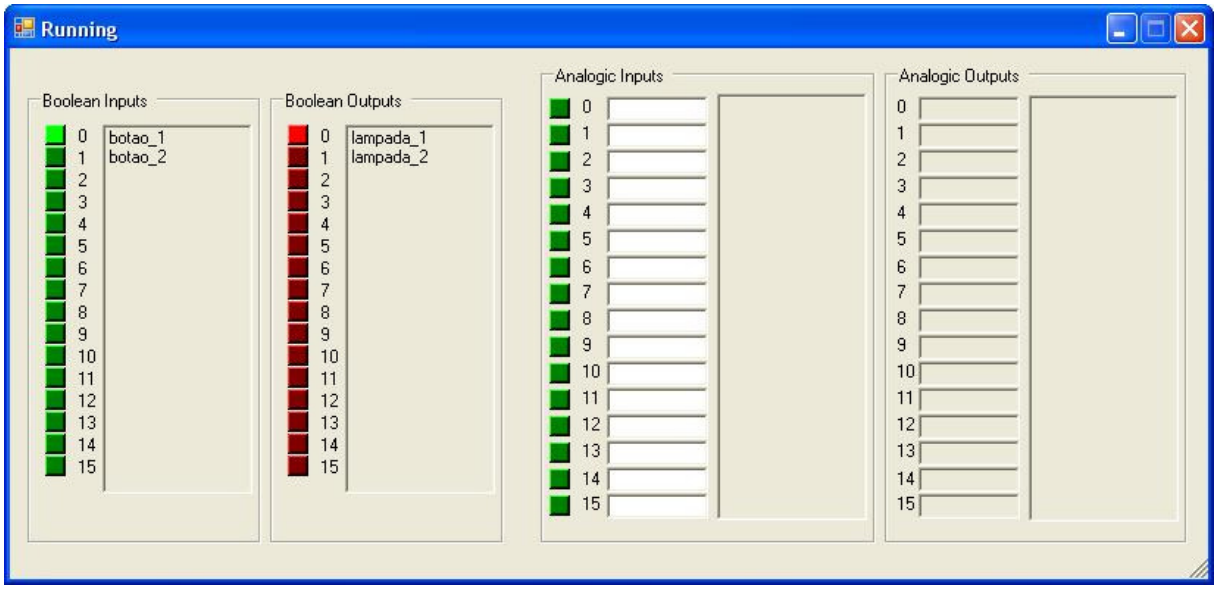


Figura 46 - Entradas e saídas digitais conectadas

Este modo de funcionamento é um recurso importante, pois permite ao usuário testar sua lógica antes da aplicação final, conectada à maquete, ou até mesmo de uma aplicação real.

Na Figura 47 foram adicionadas entradas e saídas analógicas. O usuário entra com um valor qualquer e vê os resultados nas saídas.

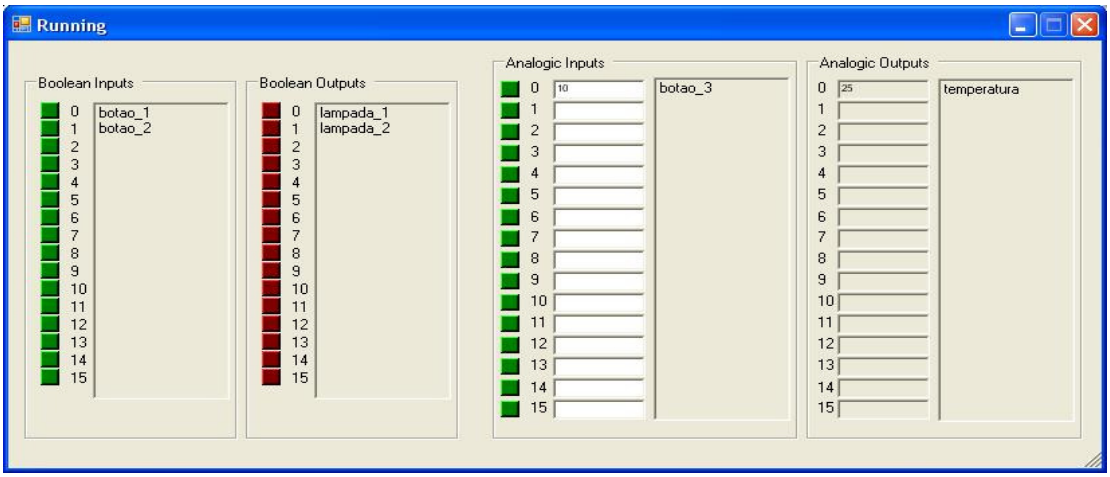


Figura 47 - Entradas e saídas digitais e analógicas conectadas

5.9.2. Modo Conectado à Maquete

No modo conectado à maquete, o usuário entra com o endereço do servidor, ou seja, da maquete virtual, e recebe as tabelas de variáveis, as quais, dada a execução do programa, serão manipuladas e enviadas de volta à maquete.

No momento da execução, o usuário não vê a tabela de variáveis, como no modo simulação, sendo que todas as variáveis estão já conectadas. Aparece na tela a maquete sofrendo as ações estabelecidas pelo usuário e os estados do diagrama sendo ativados e desativados. Abaixo, tem-se a Figura 48, que mostra o ambiente de programação funcionando integrado à maquete, e atuando na mesma.

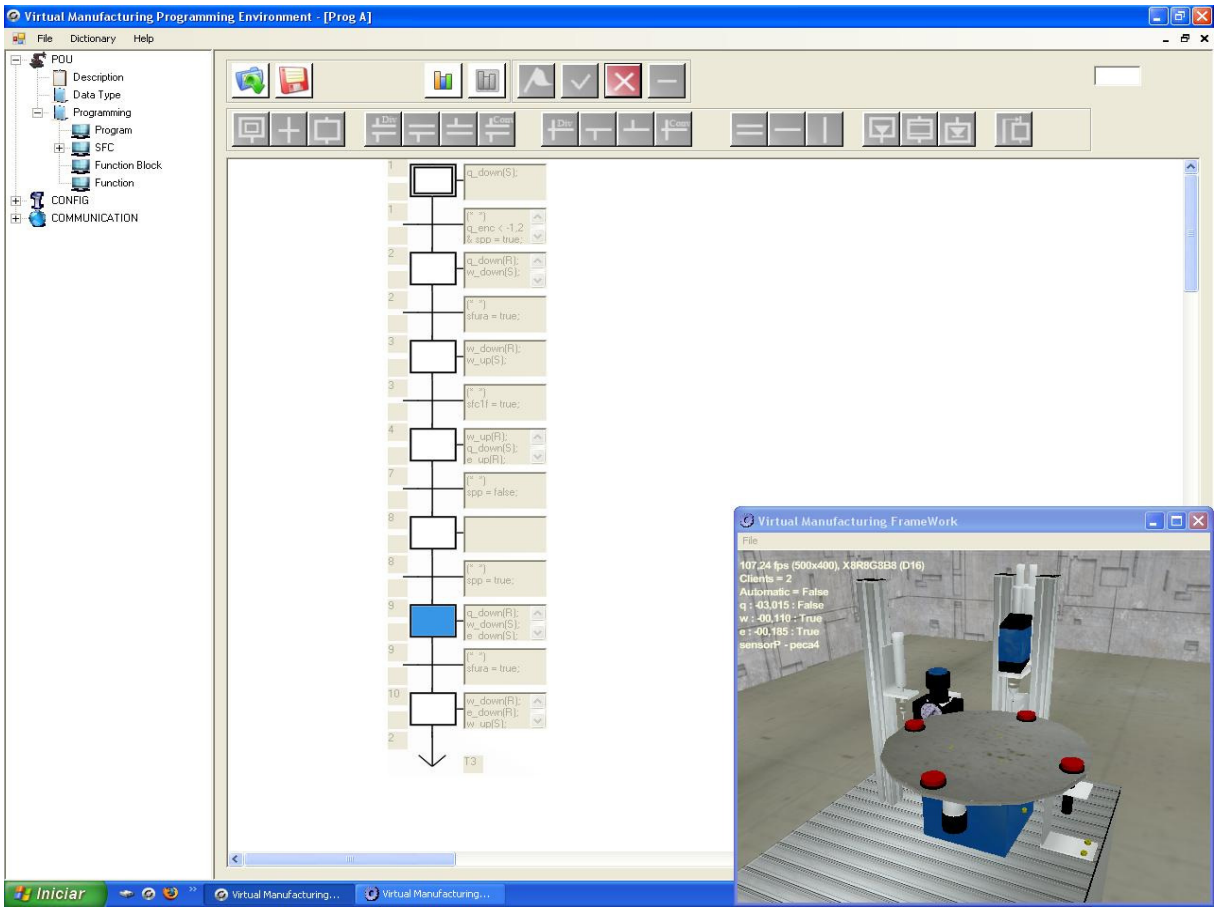


Figura 48 - Modo conectado à maquete

No anexo deste trabalho, serão mostrados alguns exemplos de aplicações, e este modo poderá ser melhor visualizado.

6. TELAS DO PROGRAMA

O programa conta com as seguintes telas:

- Tela principal: tela principal do programa, onde o usuário escolhe, por exemplo, a linguagem a ser usada;
- Dicionário: usuário insere variáveis de entrada, saída e interna, para o modo simulação;
- Descrição: onde é feita uma pequena descrição daquela aplicação;
- *Data Type*: o usuário pode visualizar as variáveis de entrada e saída vindas da maquete e também inserir alguma variável interna;
- Tela de programação: o usuário monta nesta tela o diagrama SFC;
- Edição da transição: usuário insere dados para a transição;
- Edição do passo: onde se coloca as informações para um passo;
- *Jump*: usuário escolhe o alvo de um *jump*;
- *Running*: o usuário atua nas entradas e observa o comportamento das saídas para o modo simulação;
- Comunicação: o usuário entra com o link de comunicação com a maquete.
- Erros: indica os erros na disposição dos elementos SFC na tela de programação.

Há também algumas telas pequenas, de aviso, como por exemplo, quando o usuário escreve uma lógica com a sintaxe errada, ou quando ocorre erro de comunicação com a maquete.

6.1. TELA PRINCIPAL

É a tela principal do programa, organizada em forma de menus e árvore, destinada à descrição, programação e comunicação.

Na parte dos menus, o usuário pode abrir ou salvar uma aplicação, bem como iniciar uma nova, e há também como entrar com suas preferências, parte esta a ser implementada por outros desenvolvedores. Há também o dicionário, que será mostrado ainda neste tópico, e o *help* do ambiente, também ainda não implementado.

Na árvore, o usuário pode entrar com uma breve descrição de sua aplicação. Pode abrir o *Data Type*, também a ser mostrado neste tópico, bem como escolher a linguagem de programação, *Ladder* ou SFC, por exemplo.

Há o *Config*, em que o usuário carrega o arquivo com as tabelas de variáveis provenientes da maquete, e *Communication*, no qual se entra com a *string* de comunicação com a maquete. A Figura 49 mostra a tela em questão.

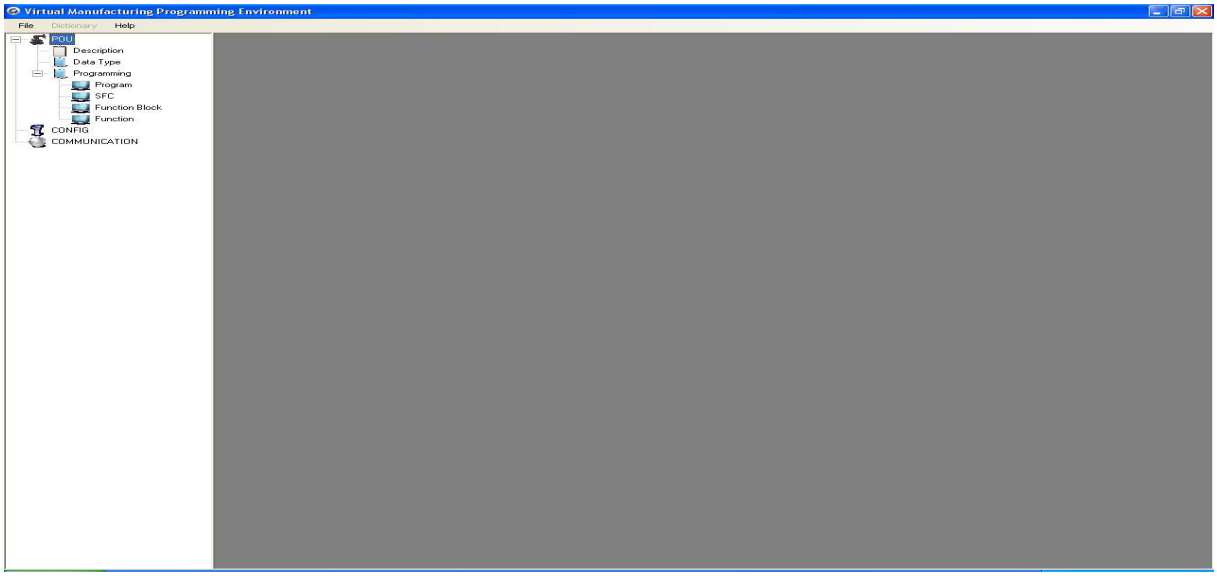


Figura 49 - Tela principal

6.2. TELA DICIONÁRIO

Esta parte do programa foi desenvolvida para o usuário entrar com as variáveis de entrada, saída e variáveis internas, que serão usadas no modo simulação, já descrito anteriormente.

Há três tabelas, *Input Variables, Output Variables* e *Inner Variables*. Cada tabela, com os seguintes campos:

- *Labels*: nome da variável;
- *Status*: status da variável, indica sendo, por exemplo, uma variável booleana, se a mesma está *true* ou *false*;
- *Initial Value*: valor inicial da variável;
- *Data Types*: usuário entra com o tipo da variável (*Float* ou *Bool*)*;*
- *Direction*: direção da variável, usado mais na lógica *Ladder;*
- *Comment*: comentário pertinente referente à variável*;*
- *Connected*: usuário entra com *true* ou *false*, indicando se a variável estará ou não conectada, para sua utilização no momento da execução de uma aplicação*;*

E há o botão *Store*, para gravar os dados inseridos e o botão *cancel*, que fecha a tela sem armazenar estes dados. Para o funcionamento correto do programa, no momento da sua execução, todos os campos devem ser preenchidos como mostra a Figura 50.

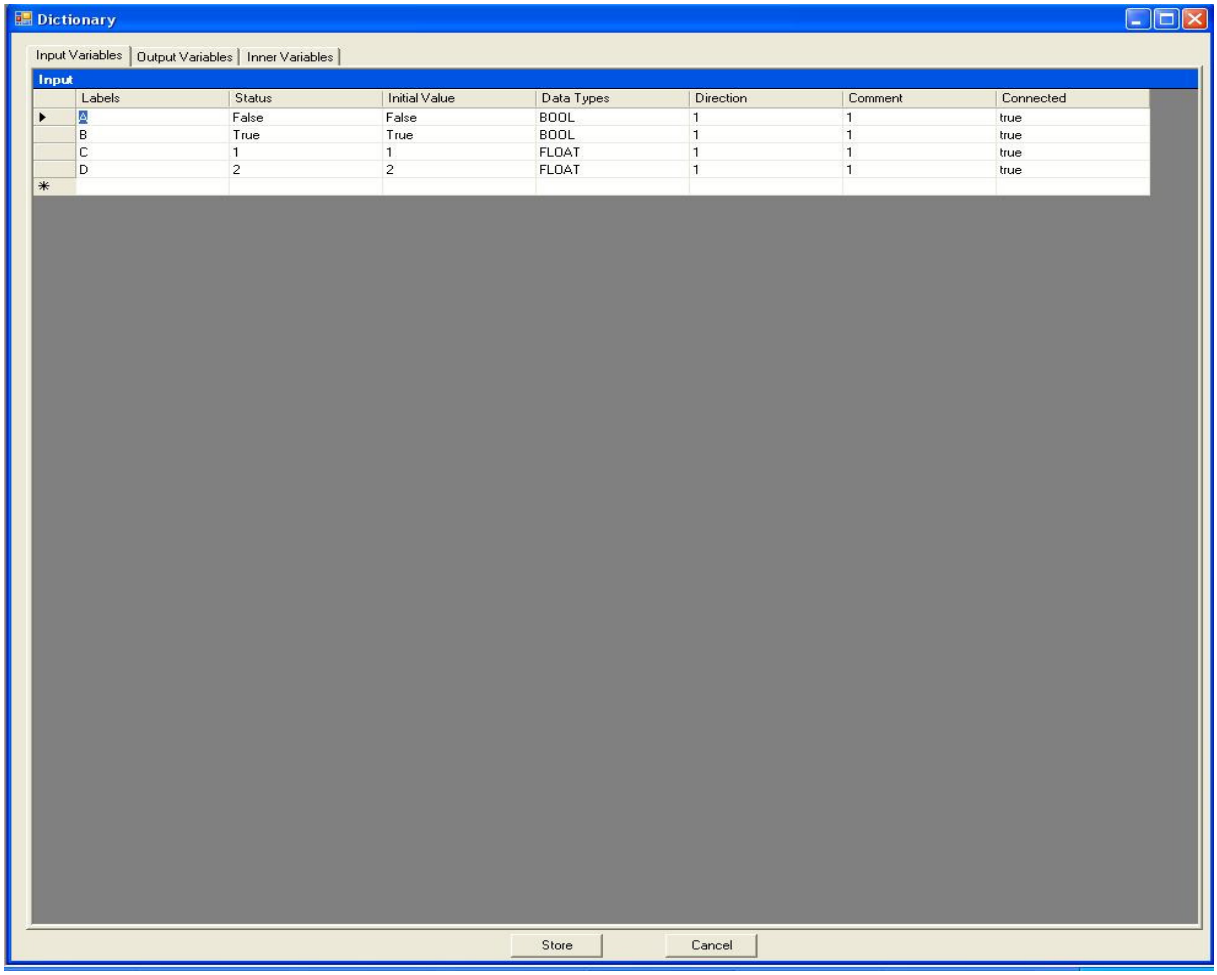


Figura 50 – Tela do dicionário

6.3. TELA DESCRIÇÃO

Esta tela serve para se fazer uma breve descrição de uma determinada aplicação, como por exemplo, mostrado na Figura 51.

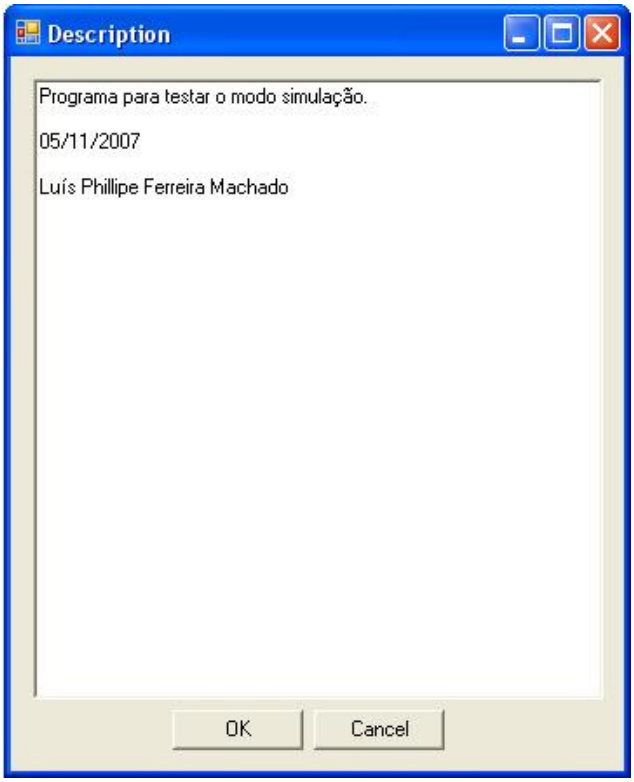


Figura 51 – Tela descrição

6.4. TELA *DATA TYPE*

Aqui o usuário tem acesso às tabelas de entrada, saídas e variáveis internas provenientes da maquete virtual, ou carregadas em *Config*. O usuário pode também inserir variáveis internas. Os campos são os mesmos do dicionário, não havendo apenas o campo *Connected*.

Pode-se observar o *Data Type* na Figura 52.

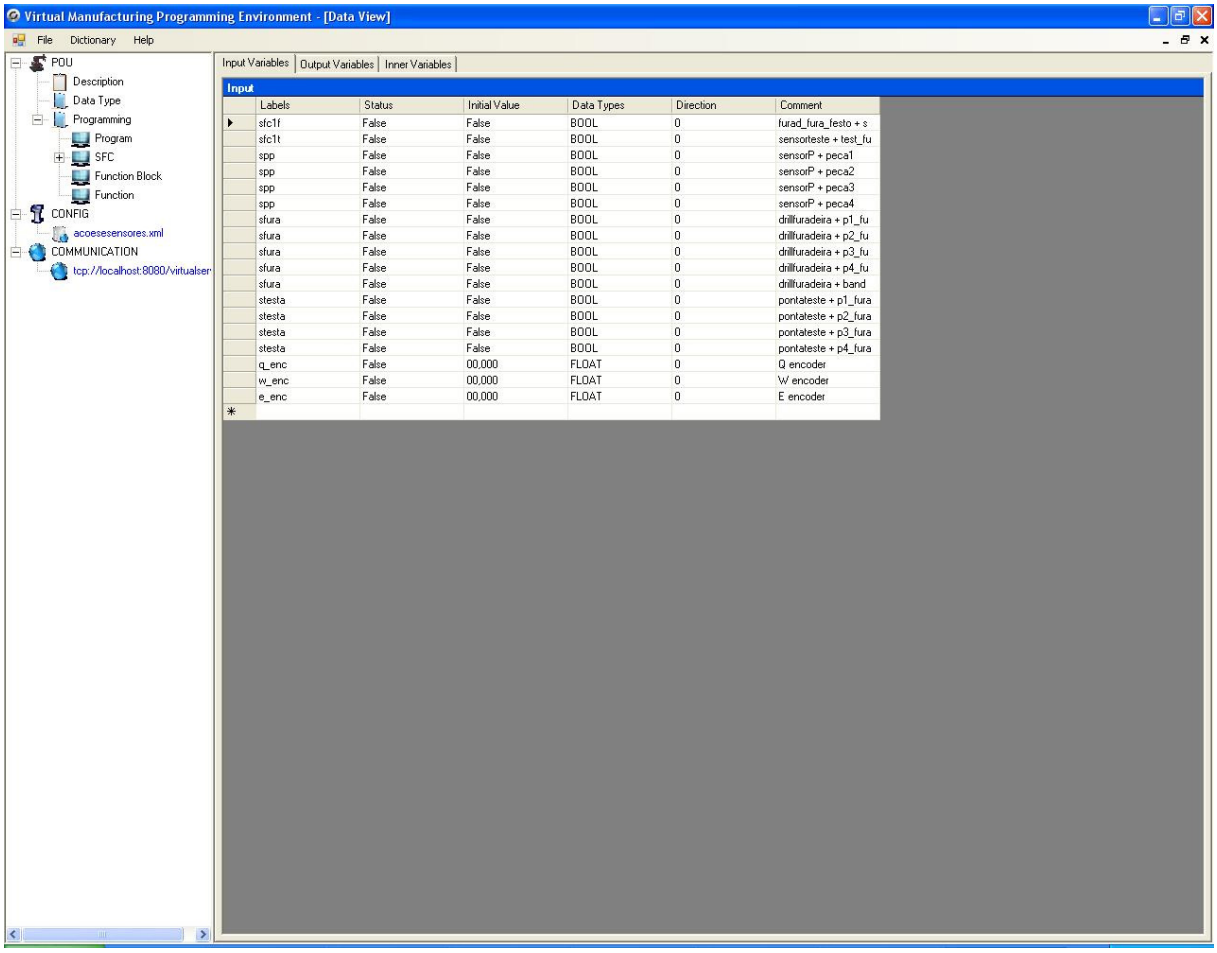


Figura 52 – Tela Data Type

6.5. TELA DE PROGRAMAÇÃO

É a parte do *software*, em que o usuário programa a aplicação em SFC, como mostra a Figura 53. A tela em branco serve para o usuário manipular os elementos, inseri-los, editá-los ou apagá-los. E há também os botões para abrir, salvar, dicionário, *data type*, botões para criação de elementos, de execução, e remoção, bem como um mostrador de coordenadas do elemento que está selecionado num dado momento.

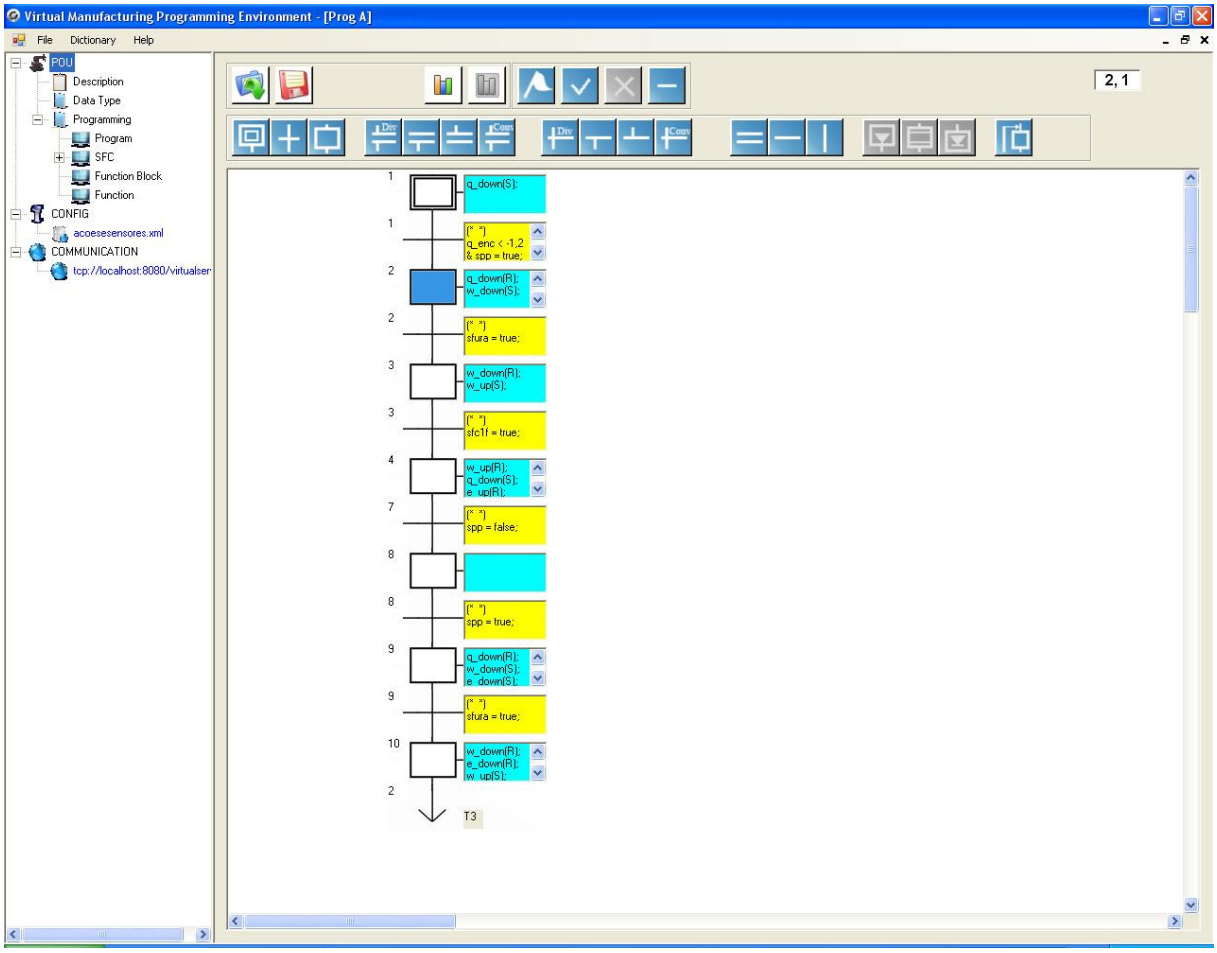
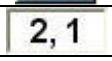


Figura 53 - Tela de programação

A Tabela 12 mostra cada botão, e sua respectiva descrição.

Tabela 12 - Elementos da tela de programação

BOTÃO	DESCRIÇÃO
	Abrir aplicação
	Salvar aplicação
	Abre o dicionário
	Abre o *Data Type*
	Roda a aplicação no modo conectado à maquete
	Roda a aplicação no modo simulação
	Encerra a execução de uma determinada aplicação

	Deleta um elemento SFC
	Cria passo inicial
	Cria transição
	Cria passo
	Cria E inicial divergente
	Cria E divergente
	Cria E convergente
	Cria E inicial convergente
	Cria OU inicial divergente
	Cria OU divergente
	Cria OU convergente
	Cria OU inicial convergente
	Cria linha E
	Cria linha OU
	Cria linha vertical
	Cria *jump*
	Coordenada de um elemento selecionado

6.6. TELA DE EDIÇÃO DA TRANSIÇÃO

Nesta tela, o usuário edita o elemento transição, quando dá um duplo clique no elemento. Há seis abas com as listas de variáveis de entrada, saída e variáveis internas presentes naquela aplicação. As abas que não apresentam *Simulation* em suas denominações, são usadas nas aplicações no modo conectado à maquete, e as demais no modo simulação.

Caso o usuário marque alguma variável, no campo *Variable Data*, aparecem as suas informações. No campo *Name*, pode-se inserir o nome daquela transição e em *Logic*, a sua lógica de controle (Figura 54).

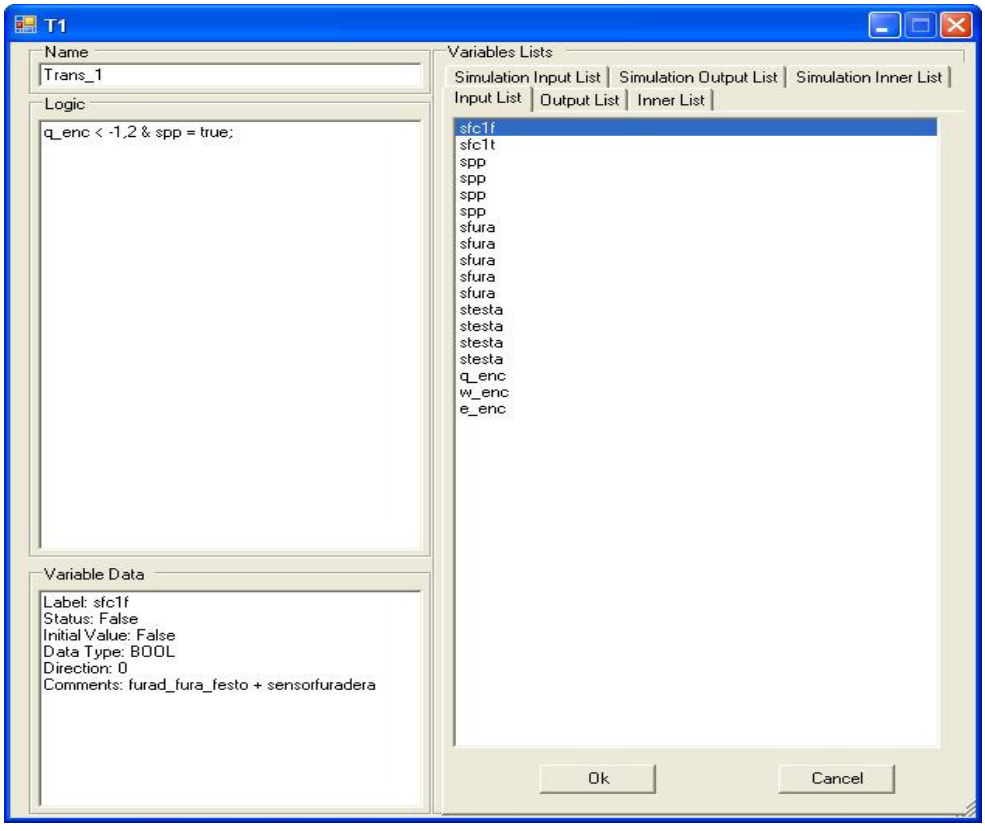


Figura 54 – Tela de edição da transição

6.7. TELA DE EDIÇÃO DO PASSO

Na tela de edição do passo, há, como na edição da transição, as seis abas com as tabelas de variáveis e também o campo *Variable Data*, mostrando as informações pertinentes da variável selecionada. Também é aberta quando um elemento passo sofre um duplo clique.

No campo *Actions*, o usuário marca *Pulse Action* ou *Non stored Action*, já descritas nas seções 3.4 e 3.5 do Capítulo 3, caso queira usá-las na lógica de controle para aquele passo.

Há também o campo *Name*, para se colocar o nome do estado, e *Logic*, para o usuário entrar com a lógica de controle (Figura 55).

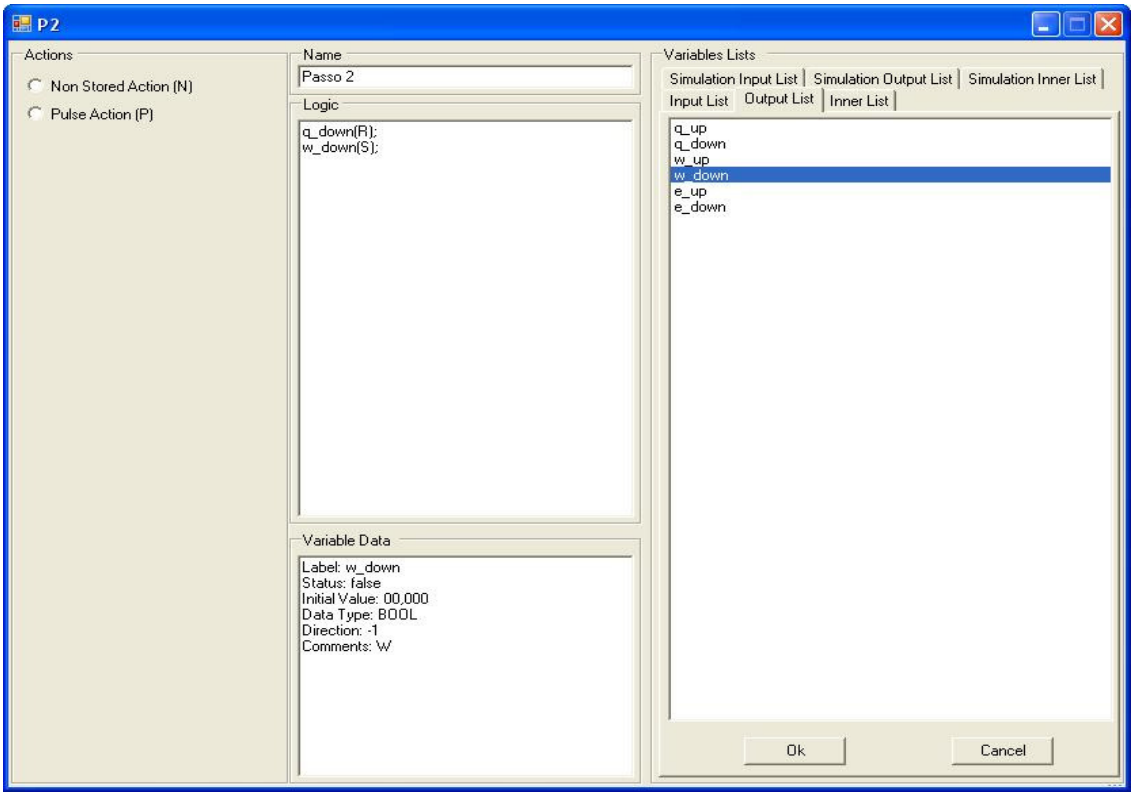


Figura 55 - Tela de edição do passo

6.8. TELA DE JUMP

O usuário executa um clique duplo no elemento *jump* e abre a seguinte tela (Figura 56):

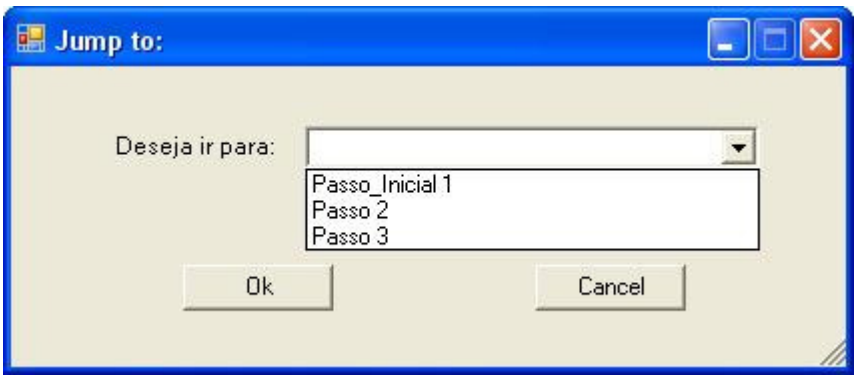


Figura 56 - Tela de jump

Através desta tela, o usuário escolhe para qual passo ou transição quer que o diagrama se dirija naquele ponto.

6.9. TELA *RUNNING*

Esta tela aparece quando o usuário executa o programa no modo simulação. Nela são mostradas as variáveis de entrada e saídas digitais, bem como as analógicas.

Nas entradas, há botões a esquerda dos nomes das variáveis. No caso das entradas digitais, o usuário clica num botão para ligá-lo ou desligá-lo, e *leds* vermelhos acendem ou apagam, representando as saídas digitais.

E na parte analógica, nas entradas, há um campo para o usuário entrar com um valor numérico, e a esquerda um botão para aplicar este valor. E nas saídas são mostrados os valores obtidos.

Isto é demonstrado na Figura 57:

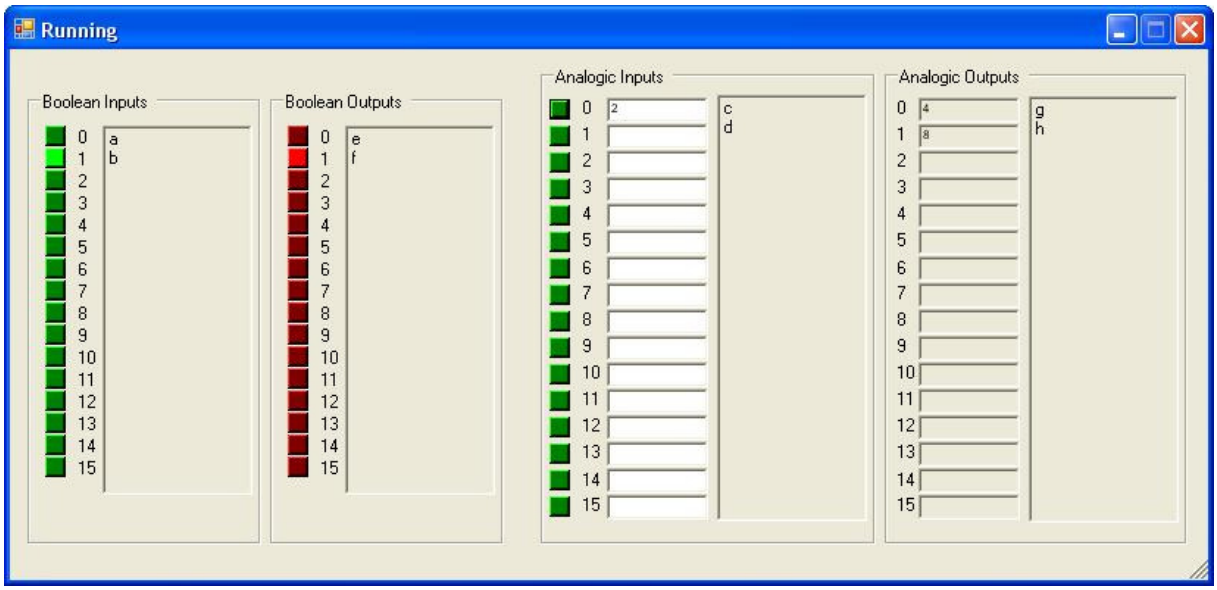


Figura 57 - Tela Running

6.10. TELA ENDEREÇO DE COMUNICAÇÃO

Aqui o usuário apenas digita o endereço do servidor, ou maquete virtual, para integrar o ambiente de programação com a maquete, como na Figura 58.

Figura 58 - Tela endereço de comunicação

Caso o usuário queira usar uma aplicação conectada à maquete, ele deve digitar na tela de comunicação a seguinte *string*: *tcp://localhost:8080/virtualserver*

6.11. TELA DE ERROS

Finalmente, a tela de erros, onde são listados todos os erros cometidos pelo usuário ao montar um diagrama SFC, como por exemplo, colocar um passo após o outro.

Esta tela sempre aparece quando o usuário executa o programa, em quaisquer dos dois modos. Caso não haja erros, o programa prossegue. No entanto, havendo elementos postos erroneamente, os erros são listados e o programa pára, só sendo executado novamente quando todos os erros tenham sido corrigidos.

A Figura 59 demonstra a ocorrência de erros em uma aplicação.

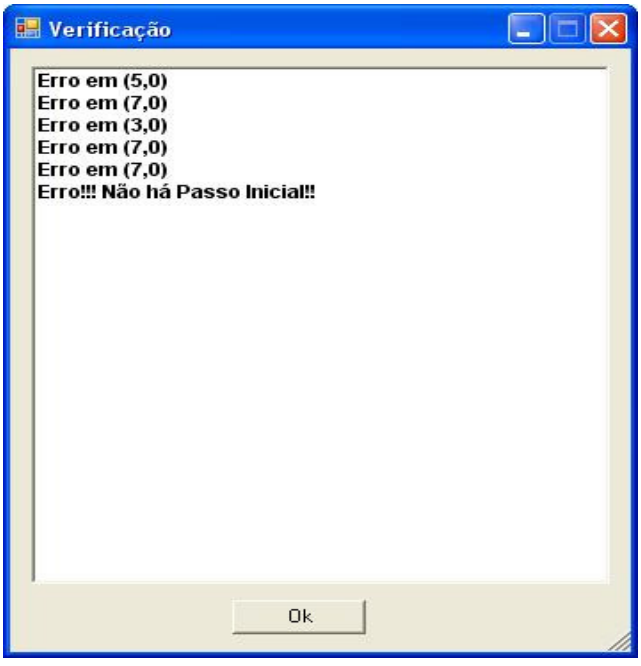


Figura 59 - Tela de erros

7. APLICABILIDADE DO SISTEMA

Em todos os processos envolvendo equipamentos, maquinários, o risco que envolve ensaios ou testes de algum determinado processo, é a possível ocorrência de perdas ou danos nestes equipamentos, sobretudo no meio industrial, onde os custos normalmente são elevados.

E no meio acadêmico, no campo da automação industrial ou engenharia elétrica e mecânica, por exemplo, é interessante que os estudantes tenham acesso a estes equipamentos, e realizem ensaios em laboratórios, para um aprendizado de maior qualidade. Mas as universidades que desejam oferecer este tipo de ensinamento, na maioria das vezes, esbarram nos altos custos dos equipamentos, e também em possíveis erros dos próprios alunos, que por vezes, os danificam.

Um dos propósitos de se desenvolver uma manufatura virtual é transportar o conhecimento e a visão intuitiva que o usuário possui sobre o mundo físico, para o mundo virtual, e dar a ele a liberdade de poder testar uma manufatura, por exemplo, de variados modos, e assim, prever seu comportamento e suas reações, evitando potencialmente, possíveis imprevistos.

É então desenvolvida uma maquete virtual dos equipamentos presentes no processo, retratando-os minuciosamente. Esta representação virtual deve ser um modelo que espelhe fielmente as características dinâmicas e físicas do seu modelo real, e fornecer o máximo de informações possíveis sobre o mesmo.

Para que o usuário tenha condições de efetuar o controle destes processos simulados, é necessário uma ferramenta que possa ser integrada à maquete, e atuar sobre ela. E este ambiente deve oferecer uma interface intuitiva e de fácil compreensão, e prover algum modo de o usuário criar lógicas de controle.

No mundo da automação industrial, como já foi mostrado em tópicos anteriores, existem linguagens de programação para controle, padronizadas pela

norma IEC61131-3 [26], e o ideal seria que este ambiente de programação oferecesse o controle, através destas linguagens, dando maior flexibilidade ao sistema e maior liberdade ao usuário.

E atualmente, no mercado industrial, existem muitas ferramentas de simulação, mas em sua grande minoria, longe de uma configuração similar à descrita acima, sendo limitados, dedicados a apenas um ou outro fabricante.

Como citado anteriormente, existem ferramentas de simulação em 3D, como o *Vericut*, que detecta erros, possíveis colisões ou áreas de ineficiência, e através dele, pode-se modelar e simular uma máquina, mas que, no entanto, não possibilita uma programação usando linguagens industriais padronizadas.

Ou o *Flexman*, para desenvolvimento e simulação de sistemas de manufatura flexíveis [11], mas que não permite edição e programação de novos elementos gráficos, para definir formas [16], comportamento e dinâmica, e são dedicados a fabricantes, e também não provêem possibilidade do desenvolvimento de uma lógica de controle usando as linguagens industriais padronizadas.

O sistema desenvolvido pela Universidade Federal de Itajubá, além de apresentar todas as vantagens descritas acima, tem um custo relativamente baixo, e pode ser usado em treinamentos de profissionais, bem como uma ferramenta de ensaios para os alunos nas universidades.

No que diz respeito ao ambiente de programação baseado na linguagem SFC, foco deste trabalho, pode-se afirmar que esta linguagem, dentre as cinco padronizadas, é a que melhor descreve o comportamento de um sistema, pois pode-se observar no momento da execução, toda a seqüência de eventos. Esta linguagem nos informa os estados de um sistema, suas mudanças, e os motivos para que elas ocorram.

Outras vantagens são a sintaxe reduzida, com poucos tipos de elementos; uma maior facilidade na identificação de erros de projeto; o tempo de desenvolvimento de um projeto é reduzido significativamente; as discussões entre

as pessoas envolvidas no projeto se tornam mais simples, devido a esta redução de elementos, e fácil visualização da seqüência de eventos de um determinado processo; e finalmente, apresentam uma notação mais compacta que a lógica *Ladder*.

Para se construir um ambiente tridimensional que retrate fielmente o mundo real, algumas etapas devem ser rigorosamente seguidas. O equipamento a ser simulado, deve ser precisamente detalhado; após isso, é feita sua modelagem em 3D; faz-se então um estudo e uma análise completa da sua dinâmica de movimentos; e estuda-se seu comportamento e relacionamento com o mundo virtual. E o ambiente criado pelos desenvolvedores da universidade segue estas etapas e usa tecnologias aplicadas a jogos, retratando com bastante fidelidade, todos os elementos, suas imagens e comportamentos, e levando em consideração fatores como colisões, e ações da força de gravidade.

Além disso, o ambiente não depende de fabricante ou equipamento, e prioriza fornecer ao usuário uma interface simples, intuitiva, e a liberdade de poder criar suas aplicações de diversos modos e prever assim o comportamento daquele sistema representado virtualmente, passando a idéia, de que aquele comportamento específico é exatamente o seu comportamento real.

Todo o código fonte do sistema foi e está sendo desenvolvido, utilizando os conceitos da orientação a objetos, o que facilita inserções futuras de novos componentes de programa.

O fato de usar linguagens padronizadas, trás a vantagem de o código de controle não precisar ser convertido para a execução de um projeto qualquer.

E finalmente, o ambiente é feito no *Microsoft Visual Studio .Net 2005*, o que provê várias vantagens, como uma maior portabilidade e flexibilidade para sistemas distribuídos, o gerenciamento da memória é feito pelo sistema operacional e não pelos programas, e seu executável é independente de plataforma, bastando apenas que haja um sistema de *runtime* que possa compilar o programa.

8. RESULTADOS

O ambiente de programação para a linguagem SFC, foco deste trabalho, foi testado tanto no modo simulação, como no modo conectado à maquete. Em ambos os casos, várias configurações de diagramas foram feitas, e os resultados foram sempre positivos e satisfatórios.

Foram testados diagramas com divergência E. Neste caso, para que ocorra a sua convergência, todos os estados anteriores a ela devem estar ativos, não podendo ir adiante, caso um estado esteja desativado. E o que se presenciou foi o comportamento correto de acordo com esta parte da lógica.

Testou-se também a divergência OU. Neste caso, a restrição que ocorre para o caso anterior não existe. E a seqüência de ativação dos estados também foi feita de maneira correta.

Outro teste executado foi verificar se os *jumps* inseridos no diagrama davam a seqüência correta da dinâmica do programa. E os resultados foram precisos e corretos.

No que diz respeito ao compilador para as lógicas impostas pelo usuário para as transições e passos, quando o mesmo entrava com alguma variável errada, ou cometia erros de sintaxe, o programa indicou corretamente estes erros, informando-os ao usuário, e o compilador se comportou bem.

Com relação ao modo conectado à maquete virtual, as informações fornecidas por ela foram transmitidas e recebidas com sucesso pelo ambiente de programação, e os dados atualizados e enviados à maquete também foram captados corretamente.

Abaixo serão dados dois exemplos de aplicações simples, a primeira no modo simulação, e a segunda no modo conectado à maquete.

O primeiro exemplo conta com o seguinte diagrama (Figura 60):

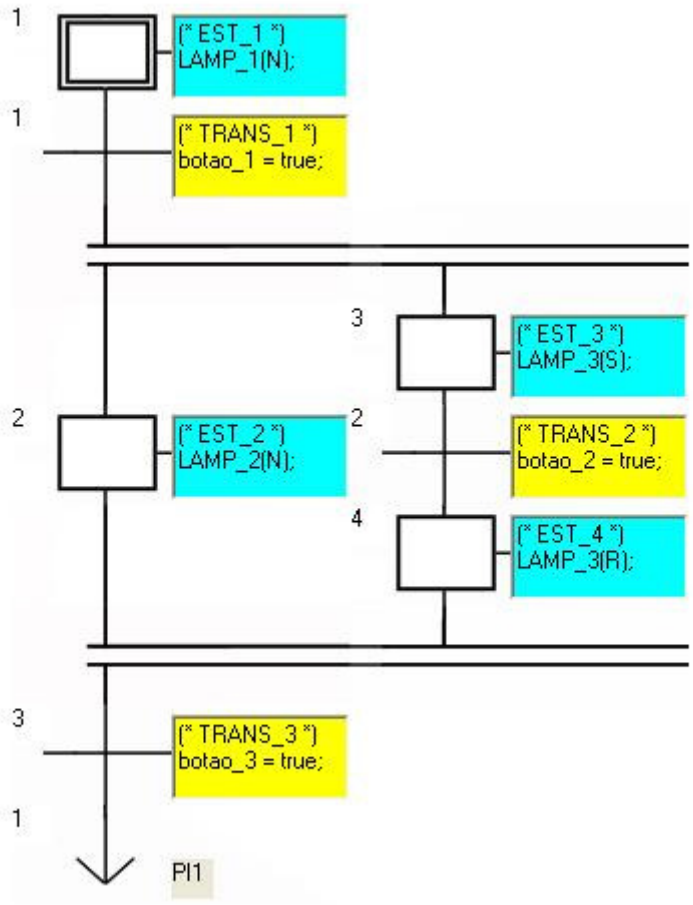


Figura 60 - Diagrama do exemplo 1

Pode-se observar que é um diagrama que aciona lâmpadas através de botões, ou seja, as lâmpadas são as saídas do sistema, e os botões atuam como as entradas. As Tabelas 13 e 14 mostram as variáveis utilizadas no exemplo.

Tabela 13 - Tabela de entradas para o exemplo 1

INPUTTABLE						
LABELS	STATUS	INITIAL VALUE	DATA TYPE	DIRECTION	COMMENT	CONNECTED
botao_1	false	false	BOOL	1	Botoeira 1	true
botao_2	false	false	BOOL	1	Botoeira 2	true
botao_3	false	false	BOOL	1	Botoeira 3	true

Tabela 14 - Tabela de saídas para o exemplo 1

OUTPUTTABLE						
LABELS	STATUS	INITIAL VALUE	DATA TYPE	DIRECTION	COMMENT	CONNECTED
lamp_1	false	false	BOOL	1	Lâmpada 1	true
lamp_2	false	false	BOOL	1	Lâmpada 2	true
lamp_3	false	false	BOOL	1	Lâmpada 3	true

Dentro de cada passo e transição, há as ações e condições do sistema. Por estas tabelas, podemos verificar que todas as lâmpadas e botões estão conectados ao ambiente de programação, e estão inicialmente no status *false*.

Ao se executar o programa, inicialmente o usuário é informado que o diagrama não contém erros (Figura 61), e assim o programa pode ser executado.

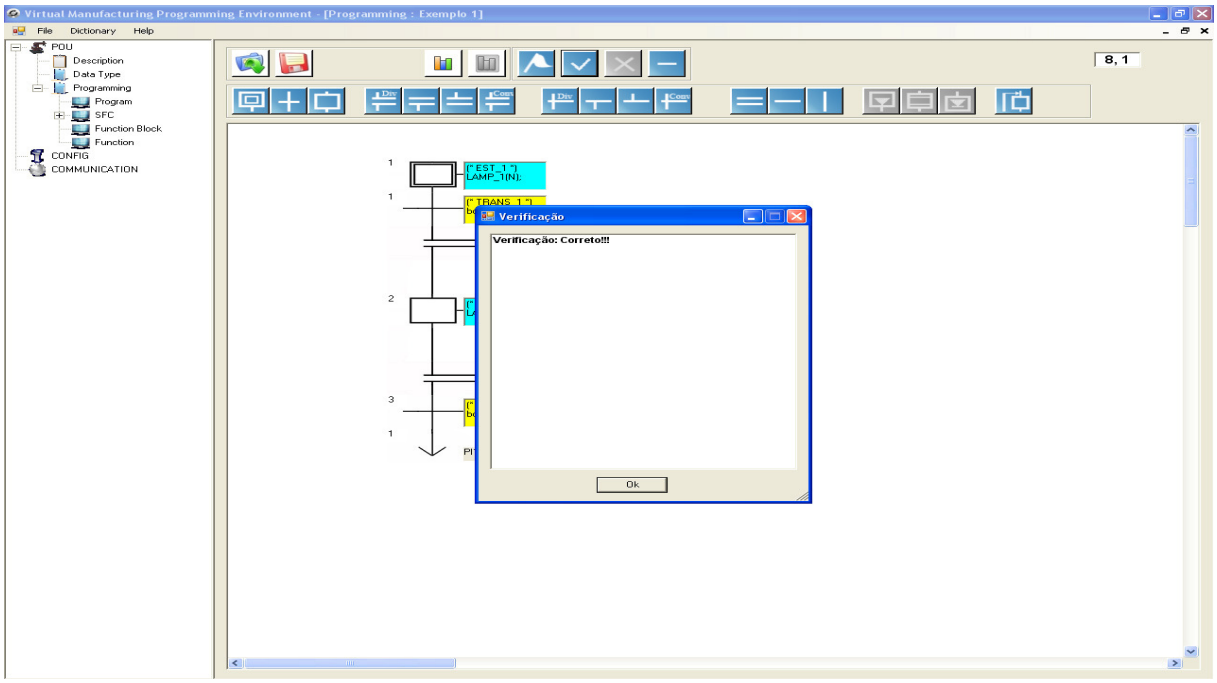


Figura 61 - Verificação de erros

Após esta verificação, a tela *Running* é aberta, e todas as saídas e entradas conectadas serão mostradas (Figura 62).

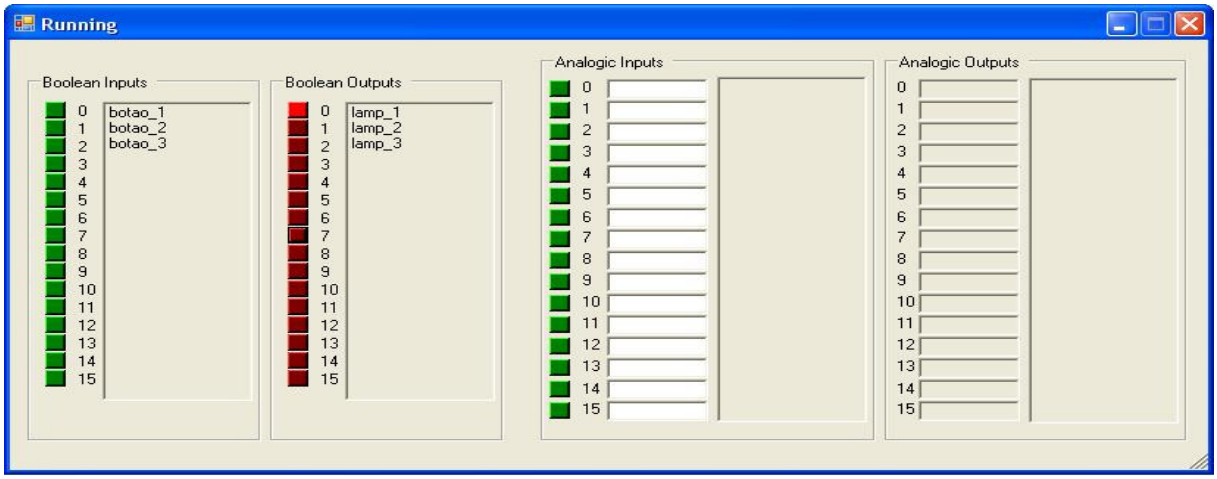


Figura 62 - Entradas e saídas digitais do exemplo 1

O programa neste momento se comporta do seguinte modo (Figura 63):

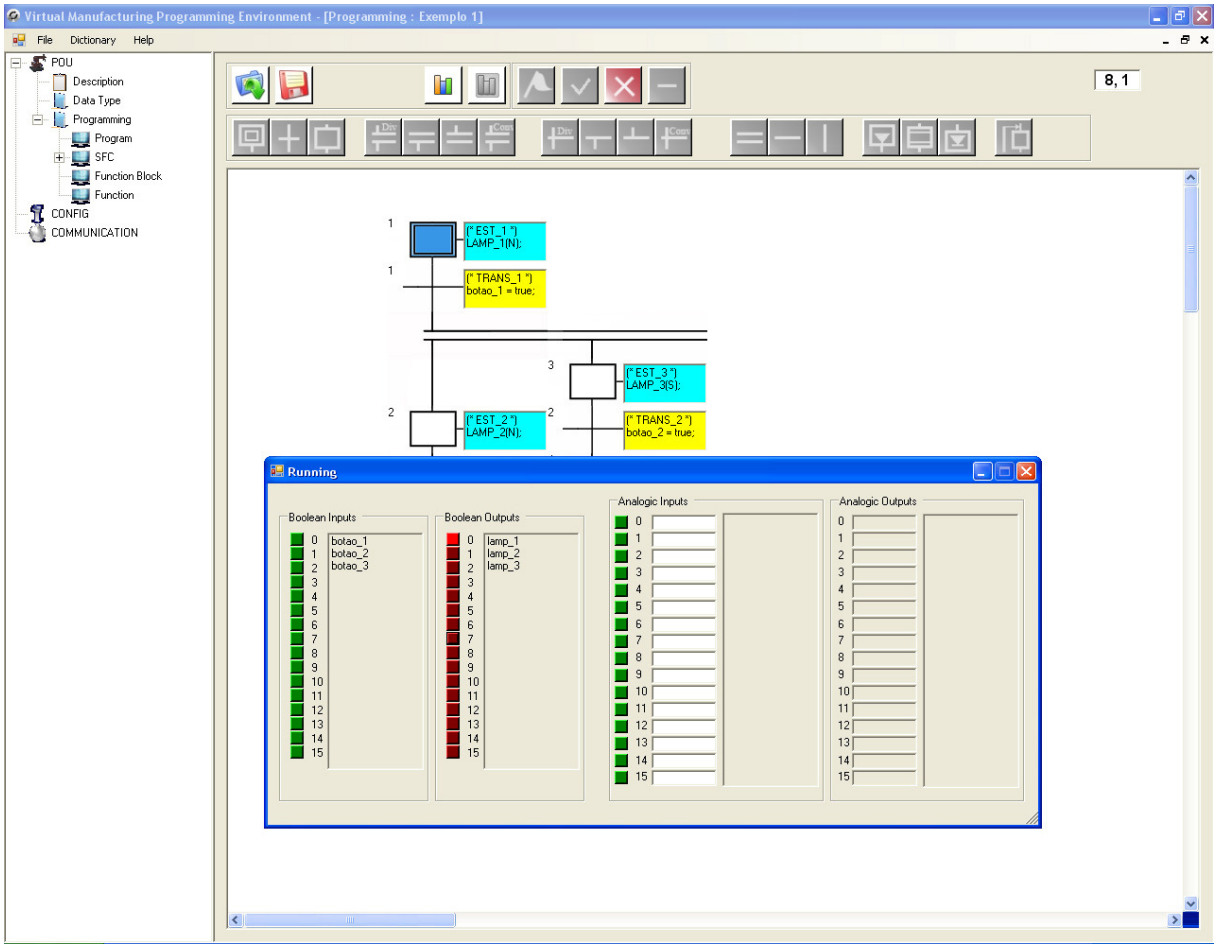


Figura 63 - Passo 1 ativo

O passo inicial é ativado, como mostra a figura acima. A ação associada a este estado é:

```
lamp_1(N);
```

Isto significa que a saída *lamp_1* sofre uma ação N quando o estado inicial fica ativo. Como já explicado anteriormente, quando uma variável sofre esta ação, ela fica ativa pelo período que o passo corrente estiver ativo. Quando este passo é desativado, o mesmo ocorre com a variável que sofre esta ação. Logo, pode-se verificar na tela *Running* que *lamp_1* ficou acesa.

A sua transição TRANS_1 tem como condição o acionamento da botoeira 1. Se esta for acionada, o passo inicial fica inativo e os passos 2 e 3 são ativados.

Observa-se na Figura 64, o momento em que o botão 1 é pressionado.

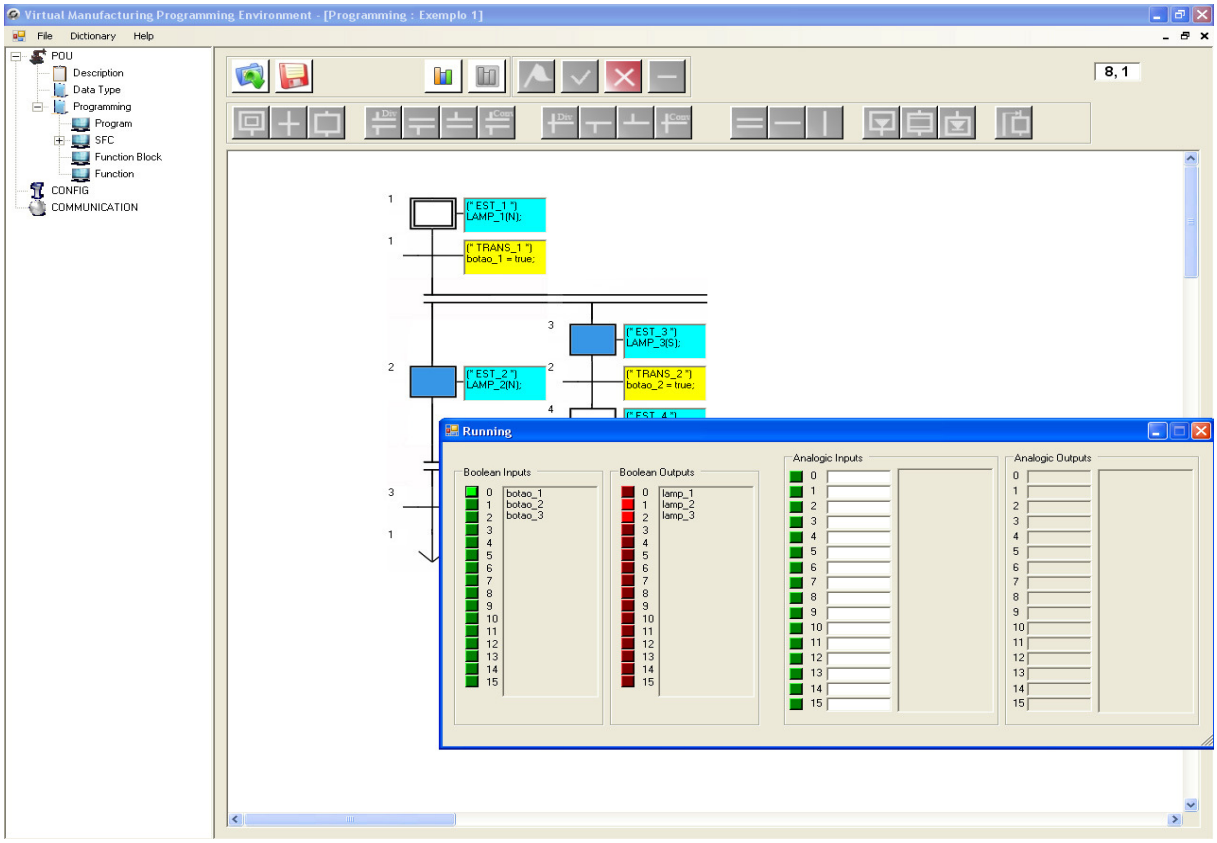


Figura 64 - Passos 2 e 3 ativos

Verifica-se que os passos 2 e 3 foram ativos. Quando o passo inicial estava ativo, *lamp_1* estava acesa. Como sofreu a ação N, com a desativação deste passo, ela apagou.

Para o passo 2, ou EST_2, a ação correspondente é:

```
lamp_2(N);
```

E para EST_3:

```
lamp_3(S);
```

Isto quer dizer que a lâmpada 2, também como a lâmpada 1, sofreu a ação N, e a lâmpada 3 a ação S. Esta ação seta a variável para *true* e esta variável só tem seu estado modificado quando sofre uma ação R ou ação *reset*.

Pela Figura 64, pode-se observar que *botao_1* está acionado e as lâmpadas 2 e 3 foram acesas, exatamente como estabelecido no diagrama SFC.

Está havendo neste caso, uma divergência E, e a transição seguinte a sua convergência tem como condição o acionamento do botão 3. Como já explicitado, para a lógica E, o sistema apenas converge quando todos os estados anteriores a ela estiverem ativos, ou seja, para o sistema chegar à transição 3, EST_2 e EST_4 devem estar ativos, fato este que não ocorre no momento, pois os estados ativos são EST_2 e EST_3.

Pressiona-se então o botão 3 para se confirmar esta regra, e verifica-se na Figura 65.

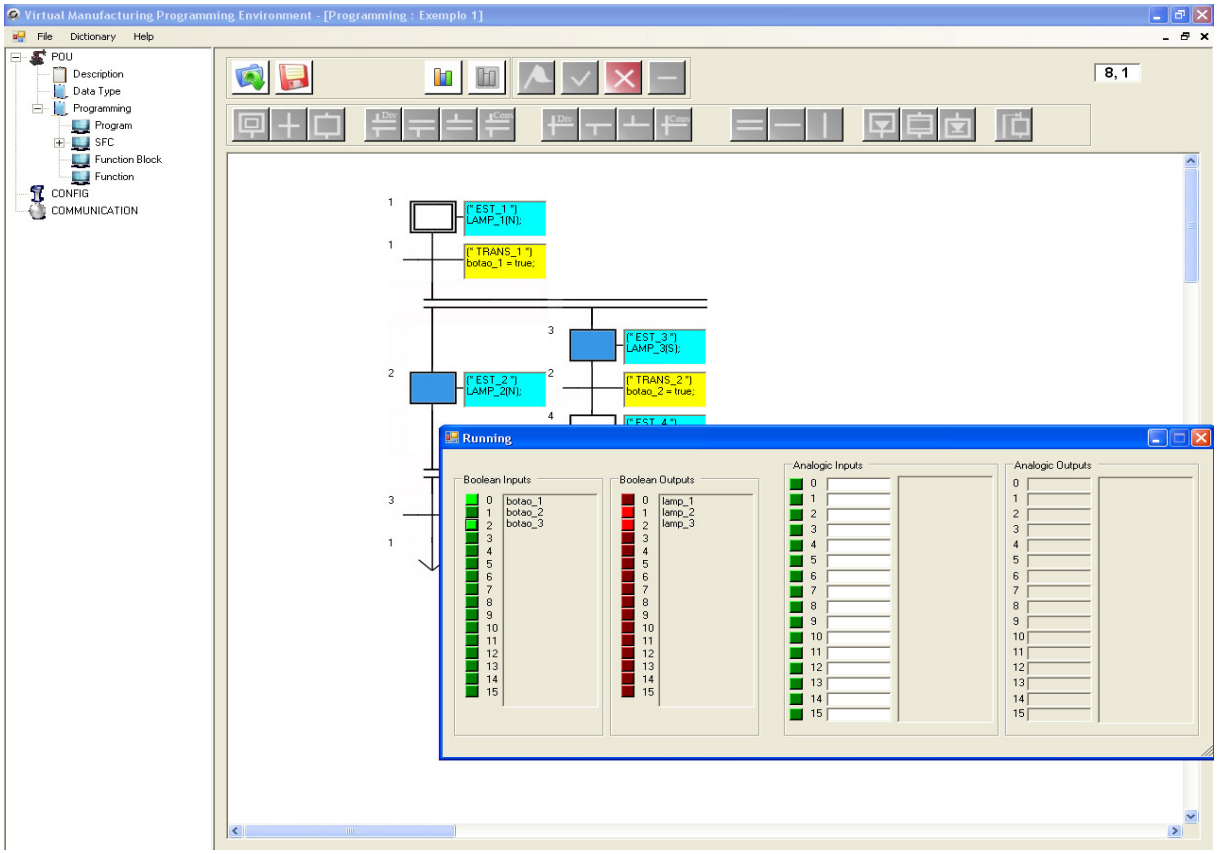


Figura 65 - Regra do E divergente

A figura anterior mostra que o botão 3 foi pressionado. No entanto, o sistema se manteve no estado em que estava na Figura 64.

Para o passo 3, ele será desativado caso, de acordo com sua transição, TRANS_2, o botão 2 seja pressionado. Logo (Figura 66):

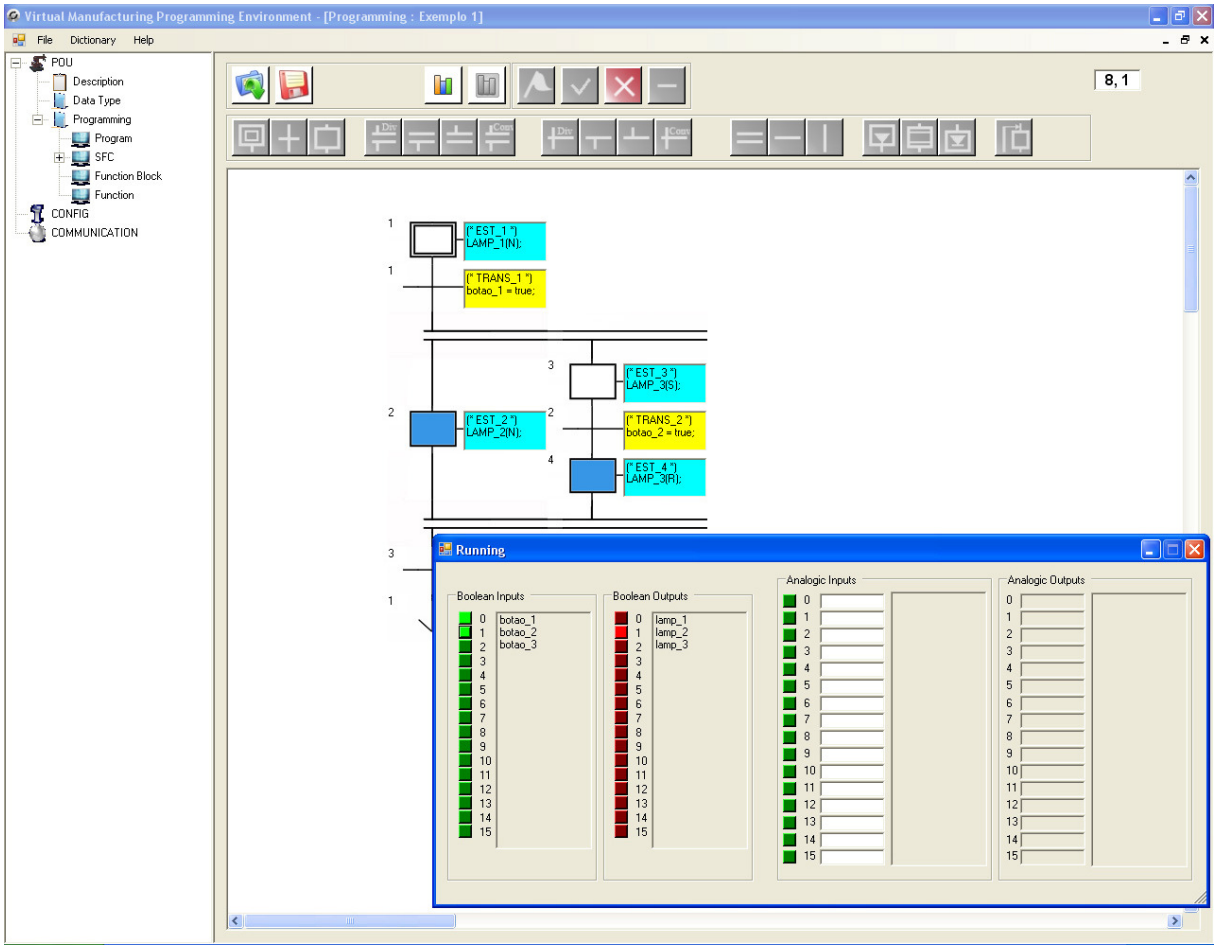


Figura 66 - Passos 2 e 4 ativos

Como *lamp_3* havia sofrido uma ação S no estado 2, e por isso havia acendido, agora no passo 4, ela sofre uma ação R, e logo, foi desligada. O botão 3 foi pressionado novamente, para ficar *false,* para se poder observar passo a passo esta aplicação, pois como agora, estão ativos os passos 2 e 3, e como a transição destes 2 passos diz que se o botão 3 estiver ligado, o programa pula para o passo inicial, ele foi desligado. E o mesmo vai ocorrer com o botão 1, que também será desligado a seguir.

O próxima etapa é pressionar o botão 3, pois a condição da transição TRANS_3 é este acionamento. Pressionado este botão, chega-se ao seguinte estado (Figura 67):

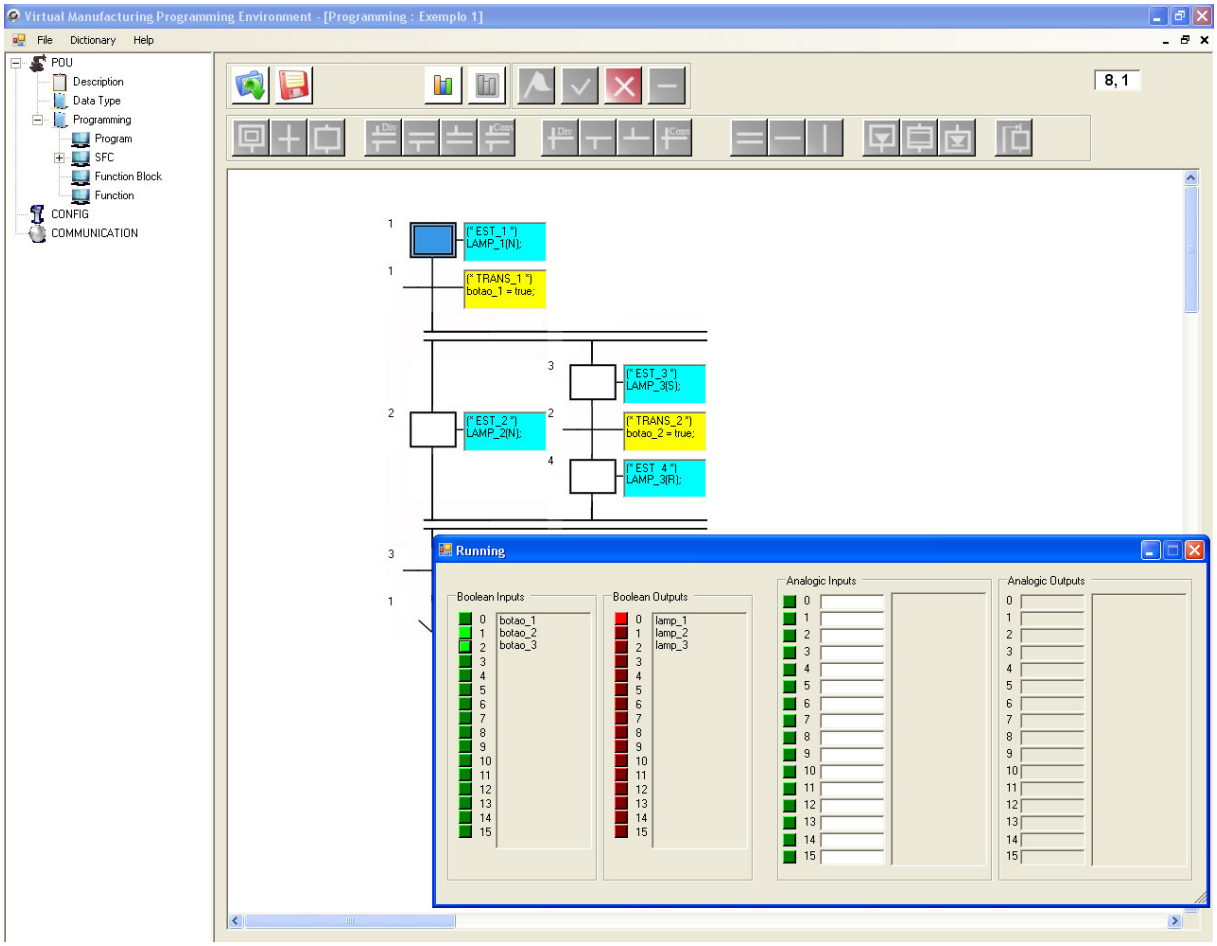


Figura 67 - Passo inicial ativo

Verifica-se que a lâmpada 2 apagou, e acendeu neste momento a lâmpada 1. Isto porque a lâmpada 2 havia sofrido a ação N, e como o estado 2 foi desativado, a lâmpada se apagou. O programa, após TRANS_3, sofre um *jump* para o passo inicial. Logo, o mesmo ficou ativo, e novamente a ação associada a ele foi executada, ou seja, o estado de *lamp_1* passou para *true*.

Logo, chega-se a conclusão, para este modo de simulação, que o sistema funciona de maneira correta, e se comportando de acordo com a lógica do usuário. No anexo deste trabalho, será mostrado mais um exemplo neste modo, para saídas e entradas digitais, bem como para variáveis analógicas.

Agora, o segundo exemplo será feito no modo conectado à maquete virtual. O diagrama proposto é simples, apenas para demonstrar-se o funcionamento correto do programa. A maquete simula uma mesa giratória Festo, com quatro peças, que

são furadas e testadas, de acordo com o acionamento de certos sensores. No anexo desta dissertação, há um exemplo mais complexo que mostra com detalhes esta maquete virtual.

A Figura 68 mostra o diagrama SFC para este exemplo:

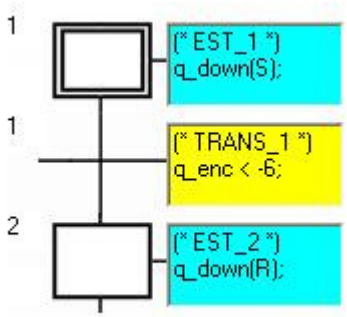


Figura 68 - Diagrama do exemplo 2

Observa-se agora as Tabelas de dados 15 e 16, que serão utilizadas, provenientes da maquete:

Tabela 15 - Tabela de entradas da maquete

Input Variables | Output Variables | Inner Variables

Input

	Labels	Status	Initial Value	Data Types	Direction	Comment
	sfc1f	False	False	BOOL	0	furad_fura_festo + s
	sfc1t	False	False	BOOL	0	sensorteste + test_fu
	spp	False	False	BOOL	0	sensorP + peca1
	spp	False	False	BOOL	0	sensorP + peca2
	spp	False	False	BOOL	0	sensorP + peca3
	spp	False	False	BOOL	0	sensorP + peca4
	sfura	False	False	BOOL	0	drillfuradeira + p1_fu
	sfura	False	False	BOOL	0	drillfuradeira + p2_fu
	sfura	False	False	BOOL	0	drillfuradeira + p3_fu
	sfura	False	False	BOOL	0	drillfuradeira + p4_fu
	sfura	False	False	BOOL	0	drillfuradeira + band
	stesta	False	False	BOOL	0	pontateste + p1_fura
	stesta	False	False	BOOL	0	pontateste + p2_fura
	stesta	False	False	BOOL	0	pontateste + p3_fura
	stesta	False	False	BOOL	0	pontateste + p4_fura
	q_enc	False	00,000	FLOAT	0	Q encoder
▶	w_enc	False	00,000	FLOAT	0	W encoder
	e_enc	False	00,000	FLOAT	0	E encoder
✳						

Tabela 16 - Tabela de saídas da maquete

Input Variables | Output Variables | Inner Variables

Output

	Labels	Status	Initial Value	Data Types	Direction	Comment
▶	q_up	False	00,000	BOOL	1	Q
	q_down	False	00,000	BOOL	-1	Q
	w_up	False	00,000	BOOL	1	W
	w_down	False	00,000	BOOL	-1	W
	e_up	False	00,000	BOOL	1	E
	e_down	False	00,000	BOOL	-1	E
*						

Ao se iniciar a execução do programa, o passo inicial é executado. A ação associada a ele é a seguinte:

```
q_down(S);
```

Isto implica que a variável *q_down* sofrerá a ação S. A tabela é atualizada e enviada à maquete. Quando *q_down* é ativada, significa que a mesa deve começar a girar. A Figura 69 mostra a mesa antes de iniciarmos a execução. E na Figura 70 já se pode observá-la após o início da execução.

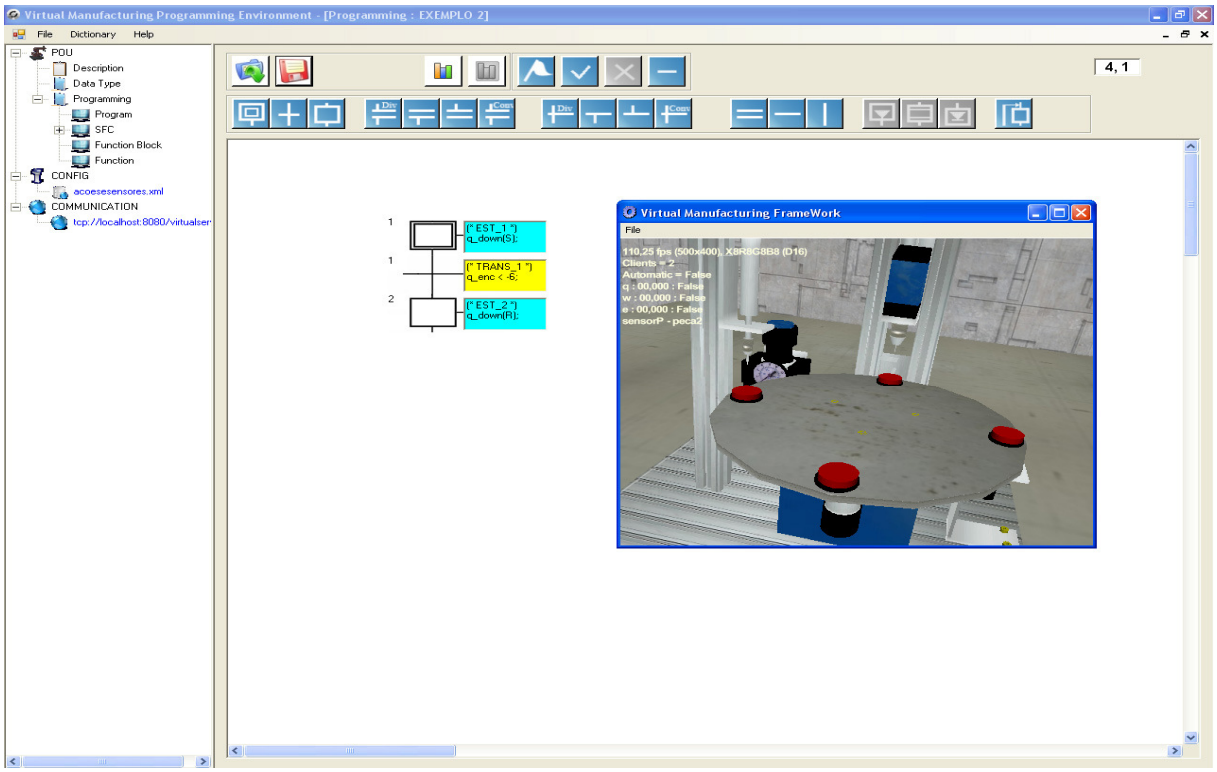


Figura 69 - Mesa antes da ativação do passo inicial

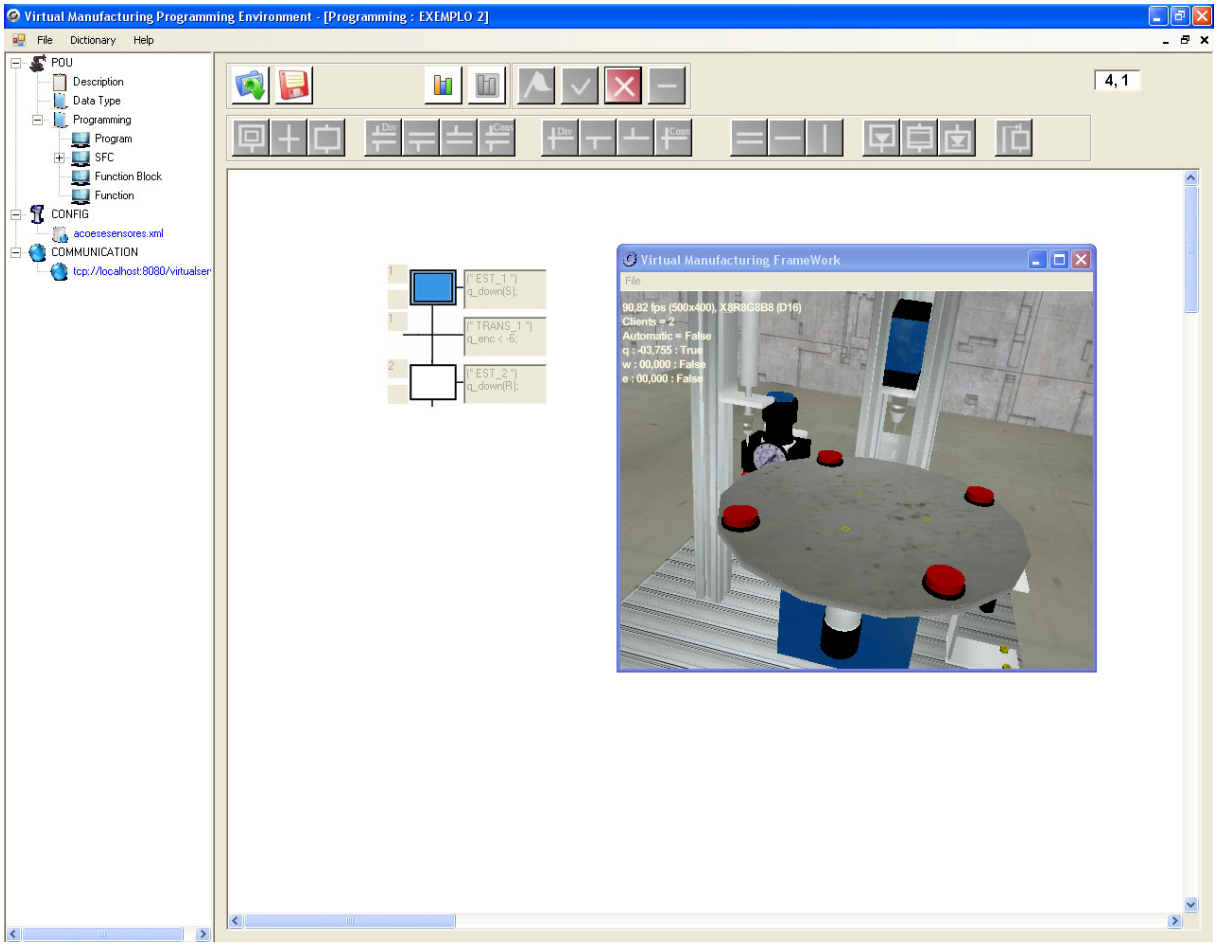


Figura 70 - Mesa após a ativação do passo inicial

Pode-se verificar pela Figura 70, que a mesa girou realmente.

A transição tem a seguinte condição:

```
q_enc < -6;
```

Significa que quando o valor do *encoder*, proveniente da maquete virtual, for menor que -6, o passo inicial deve ser desativado e o passo 2 fica ativo. A ação associada ao passo 2 dá um *reset* em *q_down*. Portanto, quando isto ocorre, a mesa para de girar.

Verifica-se na Figura 71:

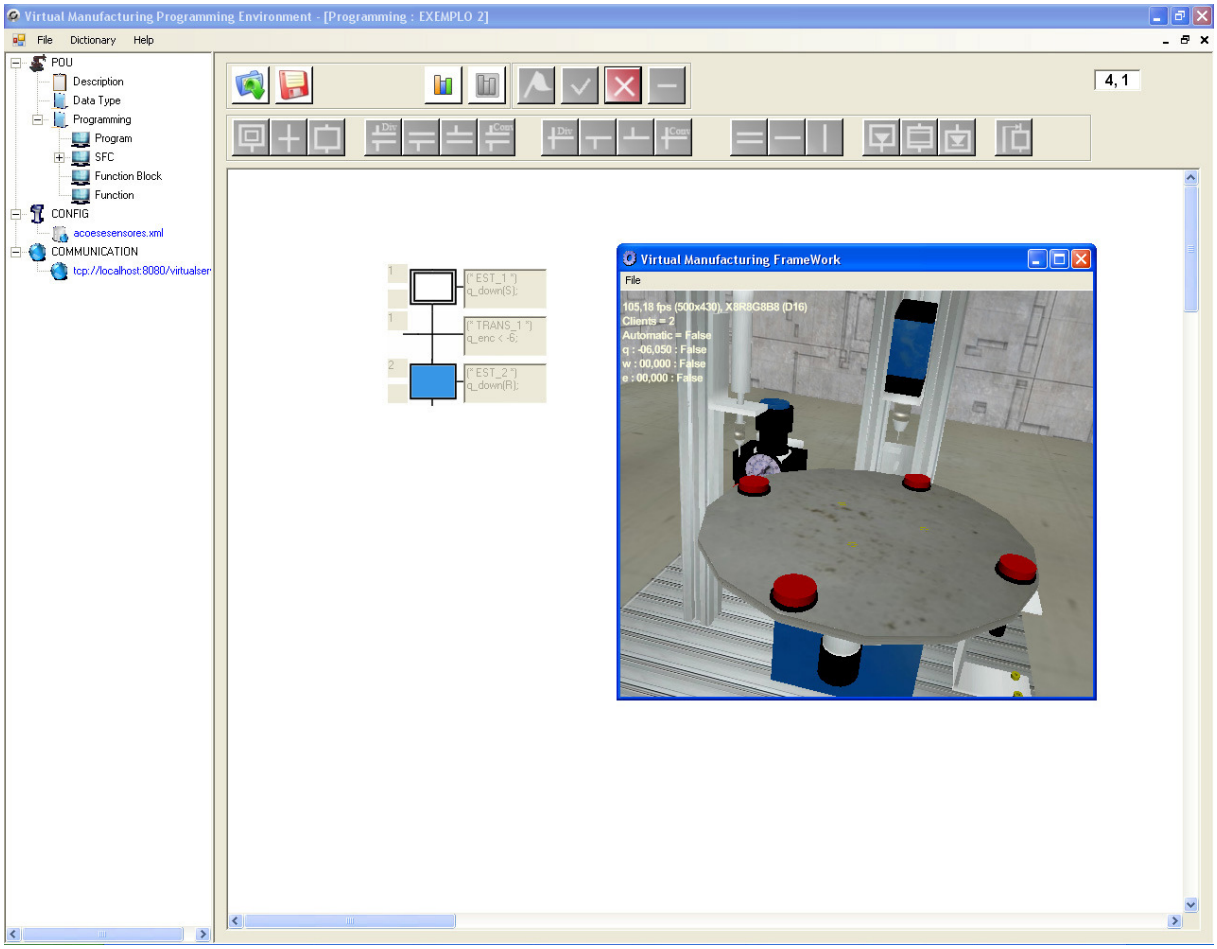


Figura 71 - Passo 2 ativo

Neste instante, quando o passo 2 se torna ativo, a maquete recebe os dados atualizados e pára de girar. Observa-se na Figura 72 que o valor do *encoder* (q) está -06,050.

Figura 72 - Valor do encoder

Pode-se dizer então, para este pequeno exemplo, que o sistema também funciona perfeitamente conectado à maquete virtual, recebendo seus dados, e os retornando atualizados a ela, atuando de maneira correta sobre a mesma.

Logo, os resultados obtidos, para ambos os modos de funcionamento, foram satisfatórios, produzindo respostas eficientes e coerentes, não havendo falhas.

9. CONCLUSÃO E PERSPECTIVAS FUTURAS

Como dito anteriormente, o ambiente de programação, foco deste trabalho de dissertação, está inserido no projeto do APMIV, que foi desenvolvido e criado para que as necessidades de segurança, flexibilidade e custo reduzido, fossem supridas, e a fim de possibilitar ao usuário o controle da manufatura, com os benefícios já descritos em tópicos anteriores. E também independe de arquiteturas de *hardware* e pode ser empregado em qualquer plataforma.

A linguagem mais difundida no meio industrial é a linguagem *Ladder*, linguagem esta, que já está disponível no ambiente desenvolvido. Porém, a linguagem SFC é uma linguagem seqüencial e gráfica que trás vantagens, como uma sintaxe mais reduzida, com poucos tipos de elementos. Melhor descreve o comportamento de um sistema, pois se pode observar no momento da execução, toda a seqüência de eventos e todos os estados do sistema, suas mudanças e os motivos para que elas ocorram.

Através do SFC, pode-se também identificar as falhas de projeto com maior facilidade. Apresenta uma notação mais compacta e o tempo de desenvolvimento de um projeto é reduzido significativamente.

O sistema foi desenvolvido com o propósito de possibilitar ao usuário a programação e execução do programa em modo de simulação, para que o mesmo possa testar sua lógica antes de conectar o sistema à maquete virtual, utilizando para isso dos elementos SFC e das variáveis criadas por ele, e também a execução integrada à maquete, onde o usuário possa carregar uma lista de variáveis, atuar sobre elas através de lógicas de controle, e consequentemente, atuar na maquete.

Em ambos os casos, testes foram realizados, situações foram simuladas e os resultados obtidos foram bastante satisfatórios, ou seja, o desenvolvimento proposto foi executado.

E neste desenvolvimento e implementação do ambiente, algumas dificuldades foram encontradas, principalmente no que diz respeito à lógica para a montagem do grafo de elementos SFC. Alguns modos foram testados, não se chegando a resultados satisfatórios, ou seja, tal grafo não traduzia fielmente a disposição dos elementos na tela.

No entanto, após algumas tentativas, uma lógica coerente foi encontrada. E os resultados foram obtidos de maneira correta.

Outra dificuldade foi em relação ao compilador da lógica de controle. Depois de algumas tentativas, acabou se chegando à conclusão que este devia ser desenvolvido de modo que se chegasse a uma expressão pós-fixa, contendo os termos da lógica criada pelo usuário, e através desta expressão, o compilador entraria em ação. Os pontos propostos a serem desenvolvidos foram criados, e os resultados também obtidos de maneira correta.

Após o trabalho desenvolvido, alguns pontos deverão ser implementados no futuro. A idéia dos alunos e professores envolvidos no projeto é desenvolver outros módulos integrantes ao sistema de manufatura virtual, e adicionar ao sistema as demais linguagens industriais padronizadas pela Norma IEC61131-3, o que incentiva novas pesquisas e futuros aprimoramentos e inovações.

Dentro da linguagem SFC, existem os chamados *macro steps*, que nada mais são que passos, que quando ativados, chamam um outro diagrama SFC paralelo, que é executado. Estes passos não foram criados. Também não foram implementadas variáveis de tempo, e no compilador para as lógicas dos estados e transições, ainda não foi criado o interpretador das expressões *FOR*, *WHILE*, *CASE* e *REPEAT*. Portanto, como trabalho futuro, os *macro steps* poderiam ser inseridos no ambiente de programação para linguagem SFC, bem como a implementação da parte das variáveis e do compilador. E também como extensão do mesmo, uma conexão do ambiente com dispositivos de *hardware*, usando para isto a comunicação OPC (*Ole for Process Control*), ou seja, um conjunto de protocolos

padronizados, pela fundação OPC, para troca de dados entre aplicações de automação e controle, e dispositivos de campo.

10. REFERÊNCIAS BIBLIOGRÁFICAS

[1] ARANTES, J. F. R.; DIAS, W.; SOUZA, L. E.; HONÓRIO, L. M. Ambiente de Programação e Integração para Manufatura Virtual. Itajubá, 2007.

[2] FAUSTINO, M. R. Norma IEC61131-3: *Aspectos Históricos, Técnicos e um Exemplo de Aplicação*. 2005. 136 f. Dissertação (Mestrado em Engenharia) – Escola Politécnica da Universidade de São Paulo, São Paulo, 2005.

[3] CURZEL, J. L.; HOUNSELL, M. S.; LEAL, A. B. Uso da Realidade Virtual para Ensino de Automação da Manufatura, *International Conference on Engineering and Computer Education*, São Paulo, mar. 2007.

[4] LATTA, J. N.; OBERG, D. J. A Conceptual Virtual Reality Model, jan. 1994.

[5] BELL, J. T.; FOGLER, S. Recent Developments in Virtual Reality Based Education, *The American Society for Engineering Education Annual Conference Proceedings*, Washington, DC, jun. 1996.

[6] BURIOL, T. M.; SCHEER, S. Integração de Modelagem Tridimensional, Visualização Científica e Realidade Virtual com Aplicação em Subestações de Energia Elétrica, *Espaço Energia*, n.6, abr. 2006.

[7] PORTO, A. J. V.; SOUZA, M. C. F.; RAVELLI, C. A.; BATOCCHIO, A. Manufatura Virtual: Conceituação e Desafios, *Gestão & Produção*, v. 9. n.3, p. 297-312, dez. 2002.

[8] Vericut. Disponível em <http://www.cgtech.com/usa/index.php>. Acesso em: 10 out. 2007.

[9] UltraArc. Disponível em <www.delmia.com/gallery/pdf/DELMIA_UltraArc.pdf>. Acesso em: 11 out. 2007.

[10] BOGDAN, S., LEWIS, F. L., KOVAÉIAE Z., GÜREL, A., STAJDOHAR, M., New Matrix Formulation for Supervisory Controller Design in Practical Flexible Manufacturing System, IEEE, 1999

[11] KOVACIÉ, S. B.; REICHENBACH, T.; SMOLIÉ-ROCAK, N.; BOGDAN, S.; BIRGMAJER, B. FlexMan – A Computer-integrated Tool for Design and Simulation of Flexible Manufacturing Systems, Zagreb, 1999.

[12] IBS PC WORXS 2.0 Data Sheet 5955A, Phoenix Contact, set. 1999.

[13] STRATON IEC61131-3 Integrated Environment Development. Disponível em: <http://www.copalp.com/english/index.htm>. Acesso em: 15 out. 2007.

[14] Siemens Automation and Drives, STEP 7 Professional - Programming and configuring according to IEC61131-3, Siemens AG 2006. Disponível em: <www.automation.siemens.com/simatic/industriesoftware>. Acesso em: 15 out. 2007.

[15] ISaGRAF IEC61131-3 Open Control. Disponível em <http://www.sixnetio.com/html_files/products_and_groups/isagraf.htm>. Acesso em: 15 out. 2007.

[16] ARANTES, J. F. R. *Ambiente de Programação e Integração para Manufatura Virtual 'APIMV'.* 2007. 123 f. Dissertação (Mestrado em Automação e Sistemas Elétricos Industriais) - Centro de Referência em Tecnologias da Informação, Universidade Federal de Itajubá, Itajubá. 2007.

[17] NETTO, A. V. *Prototipação de um Torno CNC Utilizando Realidade Virtual.* 1998. 135 f. Dissertação (Mestrado em Engenharia) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 1998.

[18] CAMANHO, A.; CAMANHO, R. Vantagens e Aplicações da Simulação de Estampagem. ESI Software, São Paulo, out. 2001.

[19] KOZAK, J. J.; HANCOCK, P. A.; ARTHUR, E. J.; CHRYSLER, S. T. Transfer of Training From Virtual Reality. In: Journal Ergonomics, Volume 36, pages 777 – 78, Minneapolis. *Resumos…*Minneapolis, 1993.

[20] HITCHCOCK, M. F.; BAKER, A. D.; BRINK, J. R. The Role of Hybrid Systems Theory in Virtual Manufacturing. 1994.

[21] LIMA, F. R.; COSENZA, C. A. N.; LIMA, P. R.; RHEINGANTZ, P. A. Maquete Virtual Aplicada em Diagnóstico de Adequação: Edifício Corporativo do INPI / RJ – Brasil. *SIGraDi Biobio 2001*, Rio de Janeiro, p. 291-294, 2001.

[22] DIAS, W.; Ambiente de Simulação para Manufatura Virtual - Dissertação de Mestrado, Universidade Federal de Itajubá, jun. 2006.

[23] HONÓRIO, L. M.; SOUZA, L. E.; ROCHA, L. C. Desenvolvimento e Simulação de Dispositivo Robótico Virtual. Itajubá, abr. 2005.

[24] BAUER, N.; HUUCK, R.; LUKOSCHUS, B.; ENGELL, S. A Unifying Semantics for Sequential Function Charts.

[25] Sequential Function Chart Language. Disponível em: <http://kernow.curtin.edu.au/www/plc/6SFC.HTM>. Acesso em: 17 ago. 2006.

[26] International Electrotechnical Commission – IEC. Programmable Controllers – Programming Languages. Final Draft – IEC61131-3, 2nd Ed. Jan. 2001.

[27] Short Presentation of Graficet. *Automatique Productique Informatique Industrielle, Hermes Editions,* França, v. *27.* Disponível em: < http://www.lurpa.ens-cachan.fr/grafcet/generalites/presentation_uk.html>. Acesso em: 17 ago. 2006.

[28] MORAES, C. C.; CASTRUCCI, P. L. Engenharia de Automação Industrial. Rio de Janeiro: LTC Editora, 2001. p. 66-73.

[29] Norma IEC61131-3. Disponível em < www.iec61131.com.br>. Acesso em: 19 ago. 2006.

[30] TORRES, B. S.; SANTOS, D. G.; FONSECA, M. O. Implementação de Estratégias de Controle Multimalha Utilizando a Norma IEC61131-3 e Ferramentas de PIMS.

[31] WAL, E. V. Os Erros Mais Comuns na Interpretação da Norma IEC61131-3, *PLCOpen.*

[32] BRUGGINK, M. Programming PLCs Using Sequential Function Chart. Nijmegen, jan. 1999.

[33] PEREIRA, S. L. Controladores Lógicos Programáveis, São Paulo, 2003.

[34] SFC AND ST PROGRAMMING LANGUAGES. *Manual da Allen-Bradley.* 2005. 137 p.

[35] ISAGRAF LANGUAGE REFERENCE. *Ajuda do IsaGraf.* 1999.

[36] Linguagem C-Sharp. Disponível em <www.microsoft.com/brasil/msdn/tecnologias/vsnet/visualstudio_basico.aspx> Acesso em: 03 abr. 2008.

[37] Orientação a Objetos. Disponível em <www.infotem.hpg.ig.com.br/tem_progr_orobj.htm#Vantagens>. Acesso em: 03 abr. 2008.

APÊNDICE A – CLASSES E FORMS DO PROGRAMA

O programa contém 6 *forms* e 20 classes. A seguir, serão mostradas tabelas contendo seus principais métodos e atributos.

CLASSES

Tabela 17 - Descrição da classe clsSFC

clsSFC	
Classe onde são listados os elementos SFC ordenadamente por linha e coluna, e a qual contém uma classe extra MatrixCheck, que serve para que a matriz de identidades dos elementos seja montada.	
ATRIBUTOS	**nlinhas:** Índice da linha do elemento; **ncolunas:** Índice da coluna do elemento; **menorl:** Menor índice da linha da matriz de elementos; **maiorl:** Maior índice da linha da matriz de elementos; **menorc:** Menor índice da coluna da matriz de elementos; **maiorc:** Maior índice da coluna da matriz de elementos; **xlmin:** Menor índice da linha da matriz de elementos; **xlmax:** Maior índice da linha da matriz de elementos; **xcmin:** Menor índice da coluna da matriz de elementos; **xcmax:** Maior índice da coluna da matriz de elementos; **ElementDatas:** Lista do tipo clsSFC_Element dos elementos SFC; **listaorganizadaSFC:** Lista já ordenada por linhas e colunas, dos elementos SFC;
MÉTODOS	**Add:** Adiciona os elementos numa lista; **rodar:** Inicia a execução conectado à maquete; **rodarSim:** Inicia a execução no modo simulação;

Tabela 18 - Descrição da classe clsSFC_Element

clsSFC_Element	
Classe que contém todos os atributos de um elemento SFC.	
ATRIBUTOS	**nlinha:** Indica a linha que está o elemento; **ncoluna:** Indica a coluna que está o elemento; **colreferencia:** Indica a coluna de referência para a montagem da árvore; **sfcimageindex:** Índice da imagem usada no elemento; **sfctooltipText:** *Tooltip* para os elementos; **sfcname:** Nome do elemento; **sfcnick:** Apelido do elemento; **tipo:** Tipo do elemento; **elemento:** Indica se o elemento é PI, P, T, J entre outros; **id:** Identidade do elemento, um número que cada elemento contém; **sfclogica:** Indica a lógica escolhida para o elemento; **texto:** Indica o texto da caixa do elemento; **contadorpi:** Contador de passo inicial; **contadorp:** Contador de passo; **contadort:** Contador transição; **contadormsi:** Contador *macro step* inicial; **contadorms:** Contador de *macro step*; **contadormsf:** Contador de *macro step* final; **contadorj:** Contador de *jump*; **contadorgeral:** Contador geral de elementos; **contdivconv:** Contador de convergência; **conteledivi:** Contador de divergência; **jump:** Elemento apontado pelo *jump*; **jumplinha:** Linha do elemento apontado pelo *jump*; **jumpcoluna:** Coluna do elemento apontado pelo jump; **daumjump:** Indica se está havendo ou não um jump; **estaconvergindoe:** Indica se está havendo convergência E; **estaconvergindoou:** Indica se está havendo convergência OU; **estadivergindoe:** Indica se está havendo divergência E; **estadivergindoou:** Indica se está havendo divergência OU; **estaconvergindo:** Indica se está havendo convergência; **estadivergindo:** Indica se está havendo divergência; **primeiroadivergir:** Indica o primeiro elemento a divergir; **idconv:** Indica o ID do elemento que está convergindo; **eoprimeiro:** Indica se é o primeiro elemento a divergir; **iddedivergencia:** Indica o ID do elemento que está divergindo; **apontapara:** Lista de elementos apontados por um determinado elemento; **eapontadopor:** Lista de elementos que estão apontando para outro elemento; **estrepetidos:** Lista de estados repetidos na montagem da árvore; **transrepetidos:** Lista de transições repetidas na montagem da árvore; **LisnatN:** Lista de elementos onde há ação N; **ActionN:** Indica se há ação N;

Tabela 19 - Descrição da classe clsSFCAcharErros

clsSFCAcharErros	
Classe que verifica se os elementos estão dispostos corretamente na tela e cria a matriz com os ID's destes elementos.	
ATRIBUTOS	**SFC:** Matriz de ID para os elementos; **erros:** Chama o *form* que lista os erros; **valornovamatriz:** Id's do elemento SFC a ser colocado na matriz SFC; **tempassoinicial:** Indica se há ou não passo inicial;
MÉTODOS	**SetSFCValue:** Monta a matriz com o ID dos elementos; **Verificacao:** Verifica os erros;

Tabela 20 - Descrição da classe clsSFCLogic

clsSFCLogic	
Classe que inicia a montagem do grafo de elementos.	
ATRIBUTOS	**est:** Lista de estados pra se inicializar; **trans:** Lista de transições pra se inicializar; **Vector:** Lista contendo os elementos SFC dispostos no *panel*;
MÉTODOS	**rodandoSim:** Chama a montagem da árvore de execução para o modo simulação; **rodando:** Chama a montagem da árvore de execução para o modo conectado;

Tabela 21 - Descrição da classe clsSFCAction

clsSFCAction	
Classe que toma as ações escolhidas pelo usuário para os estados.	
ATRIBUTOS	**dtables:** Tabela com as variáveis de simulação; **dRealTables:** Tabela com as variáveis vindas da maquete virtual; **name:** Nome das ações**;** **analisador:** Variável analisadora da lógica; **analisadortemp:** Variável analisadora da lógica; **mysensores:** Tabela de sensors; **myactions:** Tabela de ações;
MÉTODOS	**TomarAcaoNInv:** Executa ação N ou inversa para as variáveis de simulação; **TomarAcaoNInvReal:** Executa ação N ou inversa para as variáveis da maquete; **TomarAcaoO:** Liga ou desliga as saídas digitais de simulação; **TomarAcaoAnalogic:** Executa ações nas variáveis analógicas de simulação;

Tabela 22 - Descrição da classe clsSFCTable

ClsSFCTable	
Classe com os atributos de cada variável usada no modo simulação.	
ATRIBUTOS	**label:** Nome da variável; **status:** Status da variável; **ivalue:** Valor inicial da variável; **acao:** Informa a ação sobre aquela variável; **datatype:** Tipo da variável; **direction:** Direção da variável; **text:** Comentário sobre a variável; **categoria:** Categoria da variável; **connected:** Informa se a variável está conectada para o modo simulação; **index_var:** Índice da variável;

Tabela 23 - Descrição da classe clsSFCTables

ClsSFCTables	
Classe com as tabelas de variáveis usadas no modo simulação.	
ATRIBUTOS	**FileName:** Nome da aplicação; **InputTable:** Lista de variáveis de entrada; **OutputTable:** Lista de variáveis de saída; **InnerTable:** Lista de variáveis internas;
MÉTODOS	**clearall:** Inicializa as três tabelas acima;

Tabela 24 - Descrição da classe clsSFCAnalisador

clsSFCAnalisador	
Classe que analisa a sintaxe das lógicas de controle inseridas para cada estado e transição..	
ATRIBUTOS	**datatables:** Tabela com as variáveis de simulação; **dataRealTables:** Tabela com as variáveis vindas da maquete virtual; **Expressao:** Pilha com a lógica no modo expressão pós-fixa; **operadores:** Pilha de operadores; **marcadores:** Pilha de *tokens*; **resultado:** Pilha de resultados ao se executar a lógica; **pilha:** Pilha temporária com a expressão da lógica; **pilhateste:** Pilha temporária com a expressão da lógica; **termos:** Lista com cada termo da expressão da lógica; **variaveis:** Lista com as variáveis nas quais serão aplicados os resultados da lógica; **expressionteste:** Expressão clone da expressão da lógica; **condtransicao:** Condição da transição; **condicao:** Seta verdadeiro ou falso para variáveis digitais usadas na lógica; **erros:** Testa se tem erro na expressão; **not:** Testa se tem NOT na expressão; **checou:** Teste dos operandos; **elem:** Elemento SFC; **name:** Nome da ação; **pode:** *Flag* de montagem das sub-expressões; **mysensores:** Tabela de sensores; **myactions:** Tabela de ações;
MÉTODOS	**TestaParenteses:** Testa a disposição dos parênteses na expressão; **ChecaBAction:** Verifica se tem (R), (S) ou (N); **LeProximoToken:** Divide os tokens; **TestaDivOuNeg:** Verifica se é divisão ou inverso; **TestaSubOuNeg:** Verifica se é subtração ou número negativo; **TermosSintaxe:** Divide termo por termo da lógica feita pelo usuário; **TestaValidadeOperando:** Testa a validade do operando no modo simulação; **TestaValidadeOperandoReal:** Testa a validade do operando no modo conectado; **TestaSintaxe:** Indica os erros de sintaxe; **ConstroiString:** Empilha a expressão na forma pós-fixa; **SubstituirExpressaoStep:** Faz substituições de caracteres para lógica no *step*; **SubstituirExpressaoTrans:** Faz o mesmo, mas para transição; **TipoOperando:** Dá o tipo do operando em modo simulação; **TipoOperandoReal:** Dá o tipo do operando em modo conectado; **AplicaTrans:** Aplica a lógica na transição; **AplicaStep:** Aplica a lógica no passo; **UpdateStatus:** Aplica o resultado da lógica no modo simulação; **UpdateStatusReal:** Aplica o resultado da lógica no modo conectado à maquete; **AvaliaEntrada:** Verifica o valor dos operandos no modo simulação; **ReadInnerTable:** Lê a tabela de variáveis internas; **ReadDigitalServer:** Lê os dados do servidor; **ReadDigitalInput:** Lê as entradas do servidor; **ReadDigitalOutput: :** Lê as saídas do servidor; **WriteDigitalOutput:** Escreve nas saídas; **PrioridadeAntes:** Define as prioridades dos operadores; **PrioridadeDepois:** Define as prioridades dos operadores; **RetornaTipoToken:** Retorna o tipo do *token*;

Tabela 25 - Descrição da classe clsSFCSerializer_SFC

clsSFCSerializer_SFC	
Classe usada para salvar os elementos e seus contadores.	
ATRIBUTOS	**cpi:** Contador de passo inicial; **ct:** Contador de transição; **cp:** Contador de passo; **cj:** Contador de *jump*; **cmsi:** Contador de *macro step* inicial; **cms:** Contador de *macro step*; **cmsf:** Contador de *macro step final*; **x:** Variável do tipo *form* de programação; **sElementDatas:** Lista de elementos SFC;
MÉTODOS	**Iguala:** Iguala seus atributos aos atributos reais para serem salvos;

Tabela 26 - Descrição da classe clsSFCPrograms

clsSFCPrograms	
Classe usada para se adicionar os elementos SFC a serem salvos bem como o nome da aplicação.	
ATRIBUTOS	**Name:** Nome da aplicação; **SFCL:** Lista do tipo clsSFCSerializer_SFC;
MÉTODOS	**Add:** Adiciona os elementos em SFCL;

Tabela 27 - Descrição da classe clsSFCCollections

clsSFCCollections	
Classe com todos os dados de uma determinada aplicação.	
ATRIBUTOS	**mySFCPs2:** Lista do tipo clsSFCPrograms;
MÉTODOS	**Add:** Adiciona dados na lista mySFCPs2;

Tabela 28 - Descrição da classe clsSFCComparadorElementos

clsSFCComparadorElementos	
Classe que compara os elementos adicionados na lista e os ordena por linha e coluna.	
MÉTODOS	**Compare:** Compara os elementos para serem ordenados;

Tabela 29 - Descrição da classe clsSFCSerializer

clsSFCSerializer	
Classe para salvar e carregar os projetos em XML.	
ATRIBUTOS	**mySFCPss:** Instância da classe clsFSCCollections com as tabelas de variáveis, nome e demais dados de uma aplicação; **mySFCTables:** Instância da classe clsSFCTables que contém as listas de variáveis para simulação; **mySFCRealTables:** Instância da classe ClsSFCRealTables() que contém as listas de variáveis da maquete virtual; **ConnectionString:** *String* com o link de conexão; **Description:** *String* com a descrição da aplicação; **connected:** Indica se está ou não conectado; **MyClient:** Cliente da conexão;
MÉTODOS	**Connect:** Recebe a *string* e chama outro método Connect() para tentar a conexão; **Connect:** Tenta conectar e retorna *false* ou *true*; **Save:** Método para salvar a aplicação; **Load:** Método para carregar a aplicação;

Tabela 30 - Descrição da classe clsSFCRealTable

clsSFCRealTable	
Classe com os atributos de cada variável usada no modo conectado à maquete virtual.	
ATRIBUTOS	**labels:** Nome da variável; **status:** Status da variável; **ivalue:** Valor inicial da variável; **datatype:** Tipo da variável; **direction:** Direção da variável; **text:** Comentário sobre a variável;
MÉTODOS	**reset:** Leva o valor do status da variável para o seu valor inicial;

Tabela 31 - Descrição da classe clsSFCRealTables

ClsSFCRealTables	
Classe com as tabelas de variáveis usadas no modo conectado à maquete.	
ATRIBUTOS	**FileName:** Nome da aplicação; **InputTable:** Lista de variáveis de entrada; **OutputTable:** Lista de variáveis de saída; **InnerTable:** Lista de variáveis internas;
MÉTODOS	**clearall:** Inicializa as três tabelas acima; **resetall:** Leva valor do status da variável para o seu valor inicial; **LoadfromActions:** Carrega as tabelas provenientes da maquete;

Tabela 32 - Descrição da classe clsSFCGrafo

ClsSFCGrafo	
Classe que monta o grafo a partir do Estado Inicial E0	
ATRIBUTOS	**estadoatual:** Lista com o estado atual e inicial; **ListaEst:** Lista de estados; **ListaTrans:** Lista de transições; **iniciando:** Lista para se inicializar o estado e a transição; **E0:** Estado Inicial; **CrtState:** Variável do tipo estado para ser inicializada; **state:** Estado qualquer; **stateJ:** Estado indicado por um *jump*; **CrtTrans:** Variável do tipo transição para ser inicializada; **transicao:** Transição qualquer; **transicaoJ:** Transição indicada por um jump; **count:** Contador da lista de elementos; **inicio:** Indica se a árvore está começando a ser montada; **Iniciando:** Indica se a árvore está começando a ser montada; **datatables:** Tabelas de variáveis no modo simulação; **dataRealTables:** Tabelas de variáveis no modo conectado à maquete; **acaoinicial:** Ação do estado inicial; **mysensores:** Lista de sensors; **myactions:** Lista de ações;
MÉTODOS	**Populate:** Inicializa as variáveis presentes na montagem do grafo e ativa o estado inicial; **UpdateStatus:** Atualiza variáveis a serem enviadas à maquete;

Tabela 33 - Descrição da classe clsSFCRunning

ClsSFCRunning	
Classe onde é aplicada a lógica de execução do programa em SFC.	
ATRIBUTOS	**estadoanterior:** Lista com os estados anteriores da árvore; **temp:** Lista temporária com os estados seguintes da árvore; **temp2:** Lista temporária com os estados seguintes da árvore; **te:** Verifica se há mais de um estado seguinte; **conttempdetemnp:** Contador da lista *temp*; **conttemp2:** Contador da lista temp2; **cqt:** Contador das transições seguintes de um estado; **qt:** Contador das transições seguintes de um estado; **podeconver:** Indica se está havendo ou não uma convergência na árvore; **condition:** Dá a condição da transição; **ad:** Informa se um estado está sendo ativado ou desativado; **an:** Variável analisadora da sintaxe da lógica; **el:** Elemento SFC; **acao:** Variável do tipo clsSFCAction;
MÉTODOS	**Rodando:** Executa a lógica de execução; **verificatrans:** Verifica a condição da transição; **UpdateStatus:** Atualiza variáveis a serem enviadas à maquete;

Tabela 34 - Descrição da classe clsSFCState

ClsSFCState	
Classe Estado.	
ATRIBUTOS	**transicao:** Lista das transições seguintes a um estado; **transicaojump:** Lista de transições para as quais poderá ocorrer um *jump*; **transicoes_anteriores:** Lista de transições anteriores a um estado; **transicaojumps:** Lista de transições que sofreram *jump*; **transicoesrepetidas:** Lista de transições repetidas para não precisar percorrer a mesma transição duas vezes na montagem da árvore; **ListaGeralTransicoes:** Lista geral de transições; **linha:** Lista com as linhas dos elementos; **coluna:** Lista com as colunas dos elementos; **colunaconv:** Lista com as colunas dos elementos que estão convergindo; **nometransicaojump:** Lista com os nomes das transições que sofreram *jump*; **SVect:** Lista de elementos; **SVectInicial:** Lista inicial de elementos; **SVectAux:** Lista auxiliar de elementos; **OuIDSVect:** Lista auxiliar de elementos; **OuIDVectfixo:** Lista auxiliar de elementos; **ListaGeral:** Lista geral de elementos; **ListaContVert:** Lista que indica em qual elemento está quando se retorna na montagem da pilha; **LinhaConv:** Lista que indica a linha que tem convergência; **LCO:** Lista de elementos que estão abaixo de uma convergência OU; **conttransicaojump:** Contador; **contransicaojumps:** Contador; **contLinhaConv:** Contador; **contListaVertical:** Contador; **countcolunaconv:** Contador; **contLGeral:** Contador; **linhaconv:** Linha do elemento que converge; **linhadiv:** Linha do elemento que diverge; **contadordivconve:** Contador; **contadordivconvou:** Contador; **contadorvertical:** Contador; **referencia:** Coluna de referência para uma divergência; **contapontapara:** Contador; **counter:** Contador; **countlinha:** Contador; **countcol:** Contador; **acao:** Variável do tipo clsSFCAction; **SSFCElement:** Elemento SFC; **EElement:** Elemento SFC; **t:** Transição; **rodar:** Variável do tipo clsSFCRunning; **nome:** Nome do elemento; **apontado:** Nome do elemento apontado por um *jump*; **marcadoree:** Marcador de E; **ativo:** Elemento ativou ou não; **temoudiv:** Indica se tem divergência OU; **temeconv:** Indica se tem convergência E; **converfim:** Indica se está convergindo; **marcadorou:** Marcador de OU; **markroumove:** Marcador de OU; **oudentroe:** Indica se tem Ou dentro de E; **markcoluna:** Marcador de coluna; **divfim:** Indica se está divergindo; **ida:** Indica se a árvore está descendo ou subindo;

	oudivinicial: Indica se está na divergência OU inicial; **marcadorLCE:** Marcador da lista LCE; **marcadorLCO:** Marcador da lista LCO; **aponta:** Indica se o *jump* está apontando pra algum elemento; **convergiramtodose:** Indica se os elementos convergem em E; **proxtransconverge:** Indica se a próxima transição converge; **proxpassodiverge:** Indica se o próximo passo diverge; **vairepetir:** Indica se aquele elemento já faz parte da árvore que está sendo montada; **vairepetirsim:** Indica se aquele elemento já faz parte da árvore que está sendo montada; **temjump:** Indica se há *jump***:** **analisadortemp:** Variável analisadora; **datables:** Tabelas de variáveis no modo simulação; **dataRealTables:** Tabelas de variáveis no modo conectado à maquete;
MÉTODOS	**estadoseguinte:** Execução da lógica de execução; **UpdateStatus:** Atualiza variáveis a serem enviadas à maquete; **CompareListasE:** Verifica listas para ver se a árvore está subindo ou descendo; **AddSObj:** Monta árvore de elementos; **AddSEDlObj:** Monta árvore de elementos dentro de um E Divergente; **AddSOuDlObj:** Monta árvore de elementos dentro de um OU Divergente;

Tabela 35 - Descrição da classe clsSFCTrans

ClsSFCTrans	
Classe Transição.	
ATRIBUTOS	**next_state:** Lista dos próximos estados a uma transição; **Next_jump:** Lista de estados para as quais poderá ocorrer um *jump*; **Next_jumps:** Lista de estados para as quais poderá ocorrer um *jump*; **estados_anteriores:** Lista de estados anteriores a uma transição; **estadosrepetidos:** Lista de estados repetidos para não precisar percorrer o mesmo estado duas vezes na montagem da árvore; **linha:** Lista com as linhas dos elementos; **ListaContDiver:** Lista com o número de elementos que estão dentro de uma divergência; **ListaGeralEstados:** Lista de todos os estados; **nomenext_jump:** Lista com os nomes dos estados que sofreram jump; **TVect:** Lista de elementos; **TVectInicial:** Lista inicial de elementos; **TVectaux:** Lista auxiliar de elementos; **EIDTVect:** Lista auxiliar de elementos; **EIDVectfixo:** Lista auxiliar de elementos; **ListaConvergencia:** Lista de elementos que estão convergindo; **LCE:** Lista de elementos que estão abaixo de uma convergência E; **LGeralCOnv:** Lista geral de elementos que convergem; **TSFCElement:** Elemento SFC; **EElement:** Elemento SFC; **s:** Estado; **nome:** Nome do elemento; **apontado: Nome do elemento apontado por um *jump*;** **contNext_jump:** Contador; **contNext_jumps:** Contador; **counter:** Contador; **countlinha:** Contador; **countcol:** Contador; **contLEfixo:** Contador; **contLGeralConv:** Contador; **contadordivconvou:** Contador;

	contListaContDiver: Contador; **contdiver:** Contador; **contadoriddedivergencia:** Contador; **contapontapara:** Contador; **temediv:** Indica se tem divergência E; **marcadore:** Marcador de E; **markremove:** Marcador; **marcadorouou:** Marcador de OU; **edentroou:** Indica se há E dentro de OU; **bateucoluna:** Indica se a coluna de um elemento coincidiu com a coluna de referência de uma divergência; **edivinicial:** Indica se é divergência E inicial; **temouconv:** Indica se tem convergência OU; **ida:** Indica se a árvore está descendo ou subindo; **aponta:** Indica se o *jump* está apontando pra algum elemento; **proxpassoconverge:** Indica se o próximo passo converge; **proxtransdiverge:** Indica se a próxima transição diverge; **convergiramtodosou:** Indica se os elementos convergem em OU; **temjump:** Indica se há jump; **eoprimeiromesmo:** Indica se é o primeiro elemento a divergir; **CONDICAO:** Condição da transição;
MÉTODOS	**UpdateStatus:** Atualiza variáveis a serem enviadas à maquete; **VerificarCondicao:** Verifica a condição da transição; **CompareListasOu:** Verifica listas para ver se a árvore está subindo ou descendo; **AddTObj:** Monta árvore de elementos; **AddTEDlObj:** Monta árvore de elementos dentro de um E Divergente; **AddTOuDlObj:** Monta árvore de elementos dentro de um OU Divergente;

Tabela 36 - Descrição da classe ucSFCElement

ucSFCElement	
Classe base de um elemento SFC.	
ATRIBUTOS	**myElement:** Instância da classe clsSFC_Element; **uu:** Variável do tipo fmSFCRunTimePrg; **p:** Variável do tipo Panel**;**
PROPRIEDADES	**SetImageIndex:** Seta o índice das imagens para os elementos SFC; **SetText:** Seta o texto dos *tooltips* para os elementos SFC; **SetName:** Seta o nome do elemento SFC; **SetLogica:** Seta a lógica de cada elemento SFC; **SetTextos:** Seta as *strings* dos elementos SFC como nome, apelido, lógica. **SetNomeApelido:** Seta o nome do elemento SFC dado pelo usuário; **SetCount:** Seta os contadores de cada elemento SFC; **SetJumpLinha:** Seta a linha do elemento escolhido no *jump*; **SetJumpColuna:** Seta a coluna do elemento escolhido no *jump*; **SetApontaJump:** Seta o nome do elemento escolhido no *jump*;
MÉTODOS	**SetPicture:** Desenha a figura do elemento SFC; **SetElement:** Seta os elementos SFC a serem carregados no *panel* com todos os seus atributos;

FORMS

Tabela 37 - Descrição do form fmSFCRunTimePrg

fmSFCRunTimePrg	
Form para montagem do diagrama SFC.	
ATRIBUTOS	**flag:** Flag usado nos eventos *MouseDown*, *MouseMove* e *MouseUp* num elemento; **click:** Flag usado nos eventos *MouseDown*, *MouseMove* e *MouseUp* num elemento; **clicks:** Flag usado nos eventos *MouseDown*, *MouseMove* e *MouseUp* num elemento; **down:** Flag usado nos eventos *MouseDown*, *MouseMove* e *MouseUp* num elemento; **pode:** Flag usado nos eventos *MouseDown, MouseMove* e *MouseUp* num elemento; **elementoselecionado:** Indica se um elemento está selecionado; **doubleclick:** Indica se há um *DoubleClick* no elemento; **erro:** Indica se o *jump* aponta para algum elemento; **SFCFORM:** Indica se o *form* SFC está ativo; **tr:** Indica se antes de um *jump* há transição ou passo; **criouelemento:** Indica se um elemento foi ou não criado; **ocupado:** Indica se há já existe um elemento na posição em que se queira soltar um outro elemento; **rod:** Indica se a aplicação está rodando; **iniciou:** indica se iniciou a execução da aplicação; **estevocupado:** Indica se há já existe um elemento na posição em que se queira soltar um outro elemento; **SFCElement:** Elemento SFC; **sfcelement:** Elemento SFC; **Element:** Elemento SFC; **RunProg:** *Thread* para rodar a aplicação; **px:** Variável de posicionamento dos elementos quando há eventos *MouseDown*, *MouseMove* e *MouseUp*; **py:** Variável de posicionamento dos elementos quando há eventos *MouseDown*, *MouseMove* e *MouseUp*; **cpi:** Contador de passo inicial; **ct:** Contador de transição; **cp:** Contador de passo; **cj:** Contador de *jump*; **cmsi:** Contador de macro *step* inicial; **cms:** Contador de macro *step*; **cmsf:** Contador de macro *step* final; **bobjeto:** Indica se o elemento está sendo movido, editado ou deletado; **dif_Y:** Posição X da barra de rolagem; **dif_X:** Posição Y da barra de rolagem; **x:** Variável do tipo clsSFC; **jump:** *Jump*; **erros:** Erros; **estado:** Editando um estado; **transicao:** Editando uma transição; **dataTables:** Tabelas com variáveis de simulação; **dataRealTables:** Tabelas com variáveis da maquete; **frun:** Rodando; **MyClient:** Cliente; **mysensores:** Lista de sensores; **myactions:** Lista de ações; **Programs:** Variável do tipo clsSFCProggrams; **rodar:** Variável do tipo clsSFCRunning; **myserialp:** Variável usada para salvar ou abrir um projeto; **fprincipal:** *Form* principal;

	achararro: Indica se houve ou não erro na disposição dos elementos na tela; **pp:** *Form* RunTime; **podefechartela:** Indica se pode ou não fechar o *form* RunTime; **tree:** Variável do tipo clsSFCTree; **logic:** Variável do tipo clsSFCLogic; **po:** *PictureBox*; **globaltemp:** Usado no posicionamento dos elementos; **axx:** Usado no posicionamento dos elementos; **ayy:** Usado no posicionamento dos elementos; **caxx:** Usado no posicionamento dos elementos; **cayy:** Usado no posicionamento dos elementos; **axxtemp:** Usado no posicionamento dos elementos; **ayytemp:** Usado no posicionamento dos elementos; **xxtemp:** Usado no posicionamento dos elementos; **yytemp:** Usado no posicionamento dos elementos; **pyy:** Usado no posicionamento dos elementos; **pxx:** Usado no posicionamento dos elementos; **ppi:** Indica se é passo inicial ou passo; **nomeele:** Nome do elemento usado no *jump*; **SIMULAR:** Indica se está no modo simulação; **RODAR:** Indica se a aplicação está rodando;
MÉTODOS	**UpdateProgramPanel:** Carrega os elementos SFC na tela; **deleteToolStripMenuItem_Click:** Deleta um elemento; **AddSFCElementinPanel:** Carrega os elementos salvos no panel; **fmSFCRunTimePrg_Closing:** Fechando o *form*; **AntesdeSalvar:** Adiciona elementos em listas para ser salvo; **SFCCreateNewElement:** Cria os elementos SFC; **EventoDoubleClick:** Evento *DoubleClick*; **VerificaOcupado:** Verifica se há um elemento onde o usuário soltar o elemento clicado; **StopProgram:** Encerra a aplicação; **Execute:** Executa a aplicação; **UpdateStatus:** Atualiza as variáveis para serem enviadas à maquete; **ReadDigitalServer:** Lê as variáveis vindas da maquete; **BotaoVerificar_Click:** Verifica se há erros; **Run:** Roda uma aplicação; **ProcessSFCLogic:** Executa a lógica de execução; **ResetSim:** Inicializa as variáveis; **antjump:** Verifica se antes de um *jump* há transição ou estado; **listaelementos:** Lista os elementos a serem listados no *jump*; **editSFCElements:** Editando os elementos com *double_click*;

Tabela 38 - Descrição do form fmSFCDictionary

fmSFCDictionary	
Form dicionário.	
ATRIBUTOS	**connected:** Indica se a variável está ou não conectada; **myTserial:** Variável do tipo clsSFCSerializer usada pra abrir ou salvar um projeto; **myTTTserial:** Variável do tipo clsSFCSerializer usada pra abrir ou salvar um projeto; **inpData:** Lista com os atributos de cada variável; **tabindex:** Indica qual lista de variáveis está sendo editada no dicionário; **index_var:** Indica o índice da lista de variáveis que está sendo editada;
MÉTODOS	**FillTables:** Preenche as tabelas; **fill:** Preenche as tabelas; **update:** Atualiza as tabelas;

Tabela 39 - Descrição do form fmSFCRunning

fmSFCRunning	
Form que mostra as entradas e saídas no momento da execução do programa em modo simulação.	
ATRIBUTOS	**dataTables:** Listas com as variáveis de simulação;
MÉTODOS	**Preencher:** Preenche *form* com entradas e saídas; **PreencherSaidas:** Preenche com as saídas; **TomarAcaoO:** Executa ação inicial nas saídas digitais; **PreencherEntradas:** Preenche com as entradas; **TomaAcaoInp:** Toma a ação da entrada; **TomaAcaoAnalogicInp:** Toma a ação para variáveis analógicas;

Tabela 40 - Descrição do form fmSFCEstado

fmSFCEstado	
Form de edição de um passo.	
ATRIBUTOS	**dataTables:** Tabelas com variáveis de simulação; **dataRealTables:** Tabelas com variáveis da maquete; **listnumber:** Indica em qual lista está uma determinada variável; **pt:** Panel; **el:** Elemento SFC; **es:** *Form* Runtime; **na:** Variável usada para analisar a lógica escolhida; **ell:** Elemento SFC;
MÉTODOS	**btOk_Click:** Grava as informações dadas pelo usuário para o estado; **bt_outputOK_Click:** Grava as informações dadas pelo usuário para o estado; **bt_okInner_Click:** Grava as informações dadas pelo usuário para o estado; **gravae:** Grava as informações dadas pelo usuário para o estado; **VariableData:** Mostra os atributos da variável que foi clicada; **acaoN_Click:** Executa ação N; **acaoP_Click:** Executa ação P; **button2_Click:** Grava as informações dadas pelo usuário para o estado;

Tabela 41 - Descrição do form fmSFCTransicao

fmSFCTransicao	
Form de edição de uma transição.	
ATRIBUTOS	**pt:** Panel; **el:** Elemento SFC; **Current:** Componente SFC; **listnumber:** Indica em qual lista está uma determinada variável; **tr:** *Form* RunTime; **na:** Variável usada para analisar a lógica escolhida; **dataTables:** Tabelas com variáveis de simulação; **dataRealTables:** Tabelas com variáveis da maquete; **ell:** Elemento SFC;
MÉTODOS	**btOk_Click:** Grava as informações dadas pelo usuário para a transição; **bt_outputOK_Click:** Grava as informações dadas pelo usuário para a transição; **bt_okInner_Click:** Grava as informações dadas pelo usuário para a transição; **gravat:** Grava as informações dadas pelo usuário para a transição; **VariableData:** Mostra os atributos da variável que foi clicada; **button2_Click:** Grava as informações dadas pelo usuário para o estado;

Tabela 42 - Descrição do form fmSFCJump

fmSFCJump	
Form jump.	
ATRIBUTOS	**rr:** *Form* RunTime; **nome:** Nome do elemento que está sendo indicado pelo *jump*; **índice:** Índice do elemento que está sendo indicado pelo *jump*; **el:** Elemento que está sendo apontado; **eljump:** Elemento que aponta;
MÉTODOS	**jumpaponta:** Indica para qual elemento SFC o *jump* está apontando;

APÊNDICE B – SIMULAÇÕES E RESULTADOS

Foram feitas algumas aplicações simples para o modo simulação e também no modo conectado à maquete, para se verificar o comportamento do sistema. A seguir, serão apresentados três exemplos destas aplicações, uma com variáveis analógicas e digitais, no primeiro modo, e duas com o sistema integrado à maquete virtual.

Esta maquete representa uma mesa giratória Festo. Nela, há quatro suportes para peças, podendo ser fixadas até quatro peças ao mesmo tempo, como mostra a Figura 73.

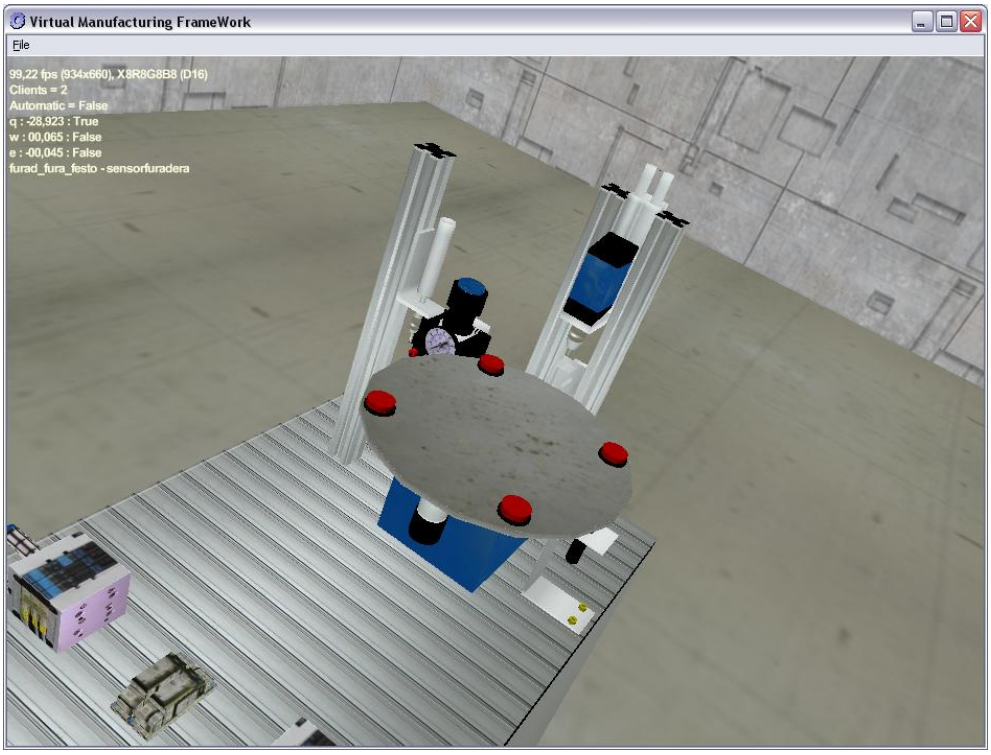


Figura 73 - Mesa giratória Festo

A mesa gira, e com isso as peças passam por três pontos: um sensor de peça, uma furadeira e um ponto de teste de furo.

EXEMPLO 1 – MODO SIMULAÇÃO

Inicialmente o usuário deve entrar na pasta *SOFTWARE*. Dentro dela há a pasta *Ambiente de Programação*, a qual contem o arquivo executável do programa, *VirtualProgramming*. A seguinte tela se abre (Figura 74):

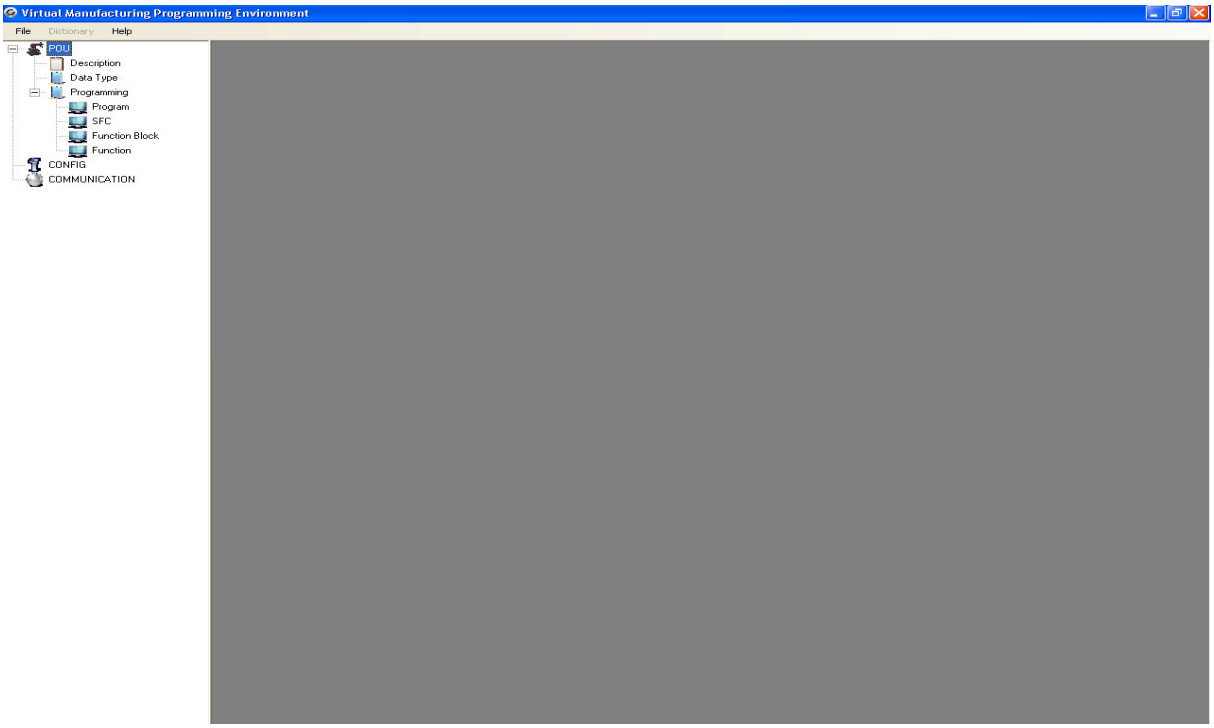


Figura 74 - Tela principal

Para se iniciar uma aplicação, o usuário deve dar um clique simples em SFC com o botão esquerdo do mouse, e após isso, com o botão direito, escolher *Add*. Isto pode ser observado na Figura 75.

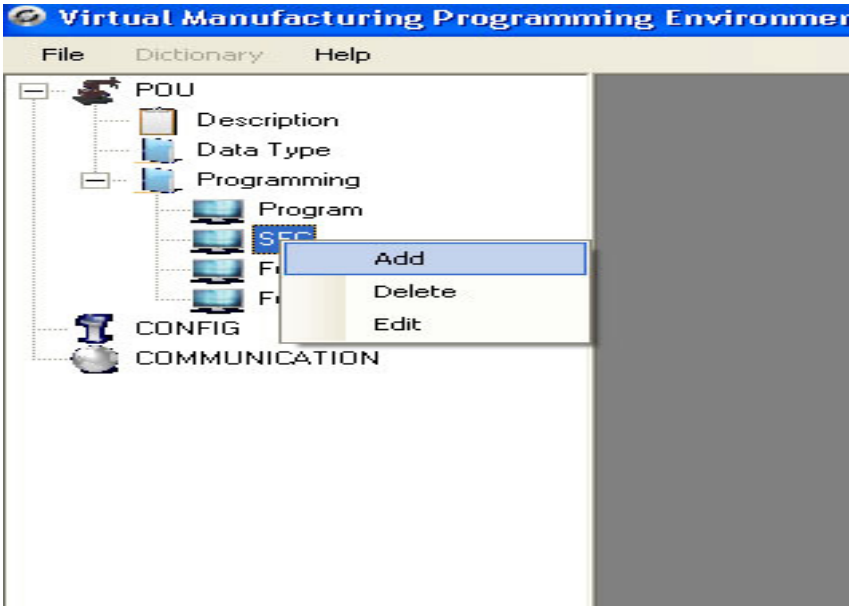


Figura 75 - Adicionando uma nova aplicação em SFC

A tela da Figura 76 irá aparecer, ou seja, o usuário deve entrar com o nome da aplicação:

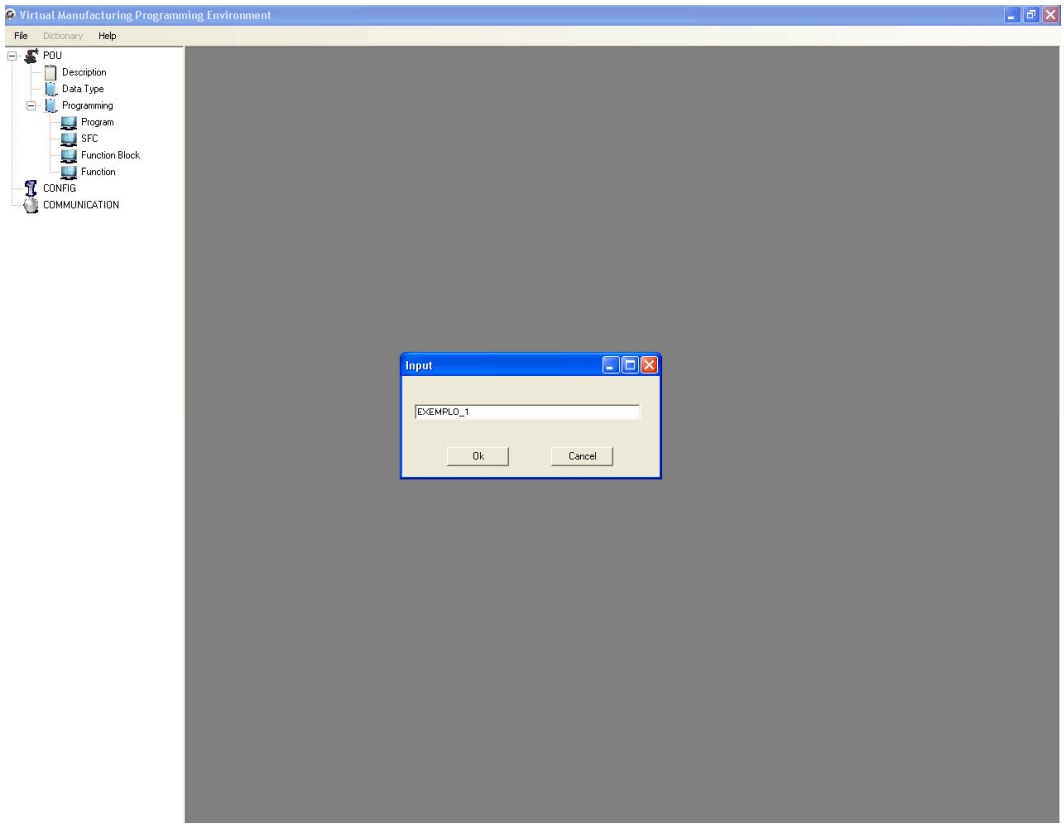


Figura 76 - Entrando com o nome da aplicação

Após isso, abre-se a tela de programação (Figura 77):

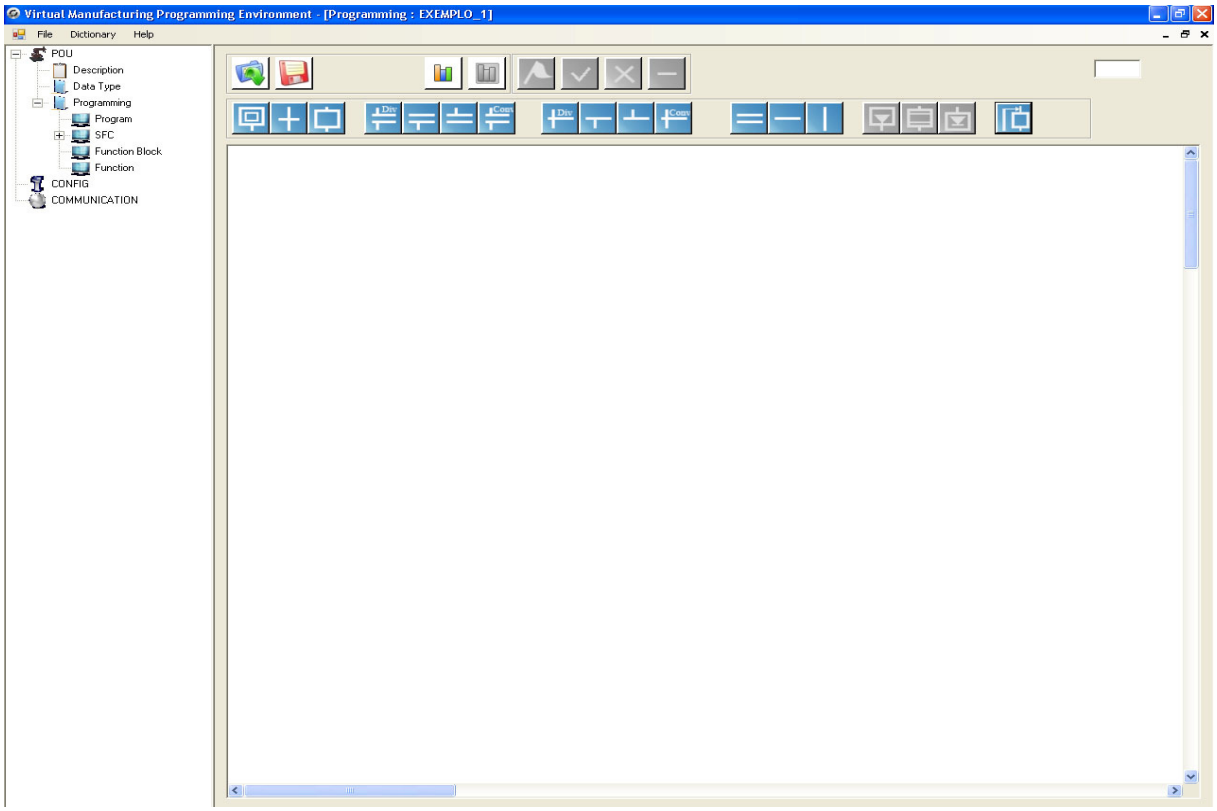
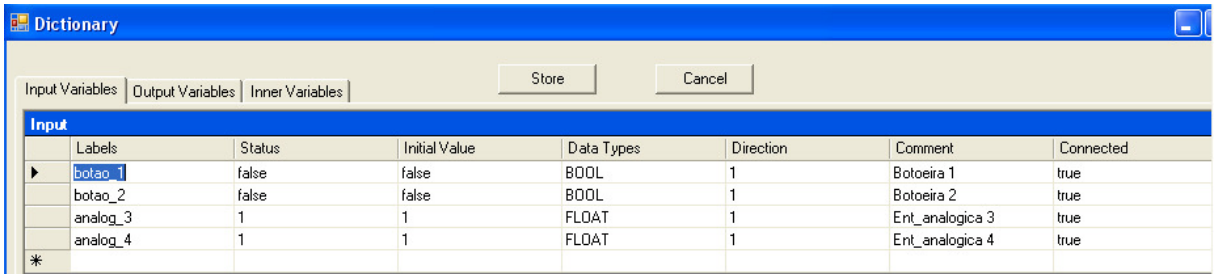


Figura 77 - Tela de Programação

Na criação de uma determinada aplicação, no modo simulação, devem ser criadas variáveis de entrada e saída. Para isto, o usuário deve entrar no dicionário. O usuário posiciona o mouse na célula inicial, e através do teclado entra com os atributos de uma variável. Todos os campos devem ser preenchidos, como nas duas tabelas a seguir. E para a gravação dos dados, o botão *Store* deve ser pressionado.

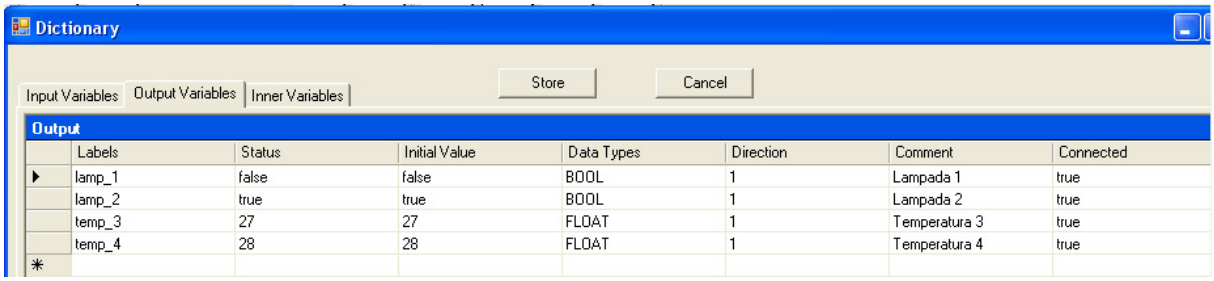
Este exemplo envolve 8 variáveis, sendo 4 entradas (2 analógicas e 2 digitais) e 4 saídas (2 analógicas e 2 digitais), listadas a seguir (Tabelas 43 e 44):

Tabela 43 - Variáveis de entrada



	Labels	Status	Initial Value	Data Types	Direction	Comment	Connected
▶	botao_1	false	false	BOOL	1	Botoeira 1	true
	botao_2	false	false	BOOL	1	Botoeira 2	true
	analog_3	1	1	FLOAT	1	Ent_analogica 3	true
	analog_4	1	1	FLOAT	1	Ent_analogica 4	true
*							

Tabela 44 - Variáveis de saída



	Labels	Status	Initial Value	Data Types	Direction	Comment	Connected
▶	lamp_1	false	false	BOOL	1	Lampada 1	true
	lamp_2	true	true	BOOL	1	Lampada 2	true
	temp_3	27	27	FLOAT	1	Temperatura 3	true
	temp_4	28	28	FLOAT	1	Temperatura 4	true
*							

No que diz respeito à montagem do diagrama SFC, o usuário clica nos botões referentes aos elementos que ele deseja utilizar. Por exemplo, para inserir o primeiro elemento, *Passo_Inicial*, o seguinte procedimento deve ser realizado: o botão Passo Inicial deve ser clicado, e aparecerá então o elemento Passo Inicial na célula de coordenada (0,0), como na Figura 78:

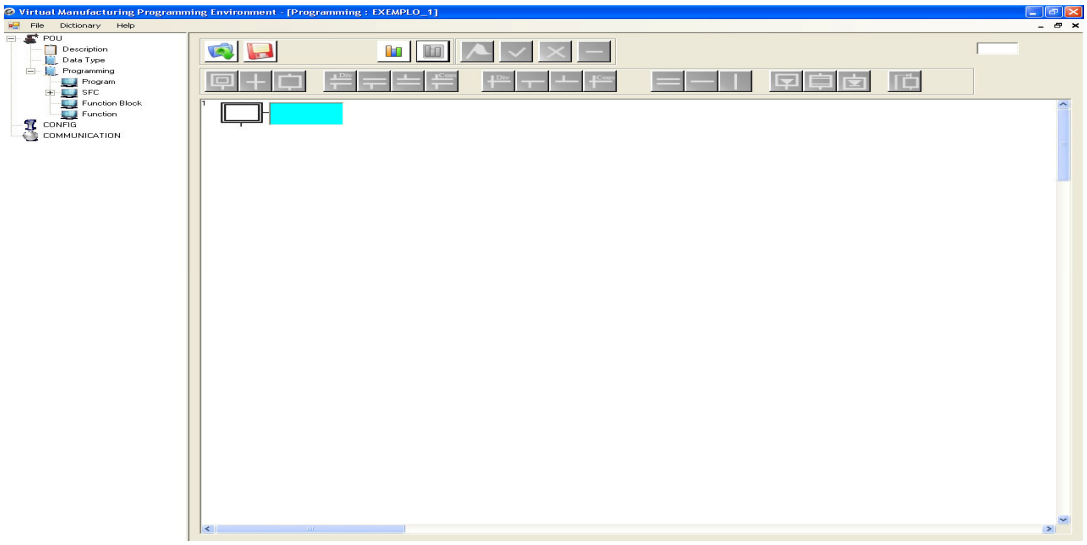


Figura 78 - Inserindo um elemento

Neste momento, o usuário deve arrastar o elemento até uma posição desejada ou clicar nesta posição, para que o elemento seja posicionado de modo correto. E assim ocorre com os demais elementos.

Para a inserção das lógicas de controle para os passos e transições, deve se clicar no elemento. Para o passo, ocorre o seguinte (Figura 79):

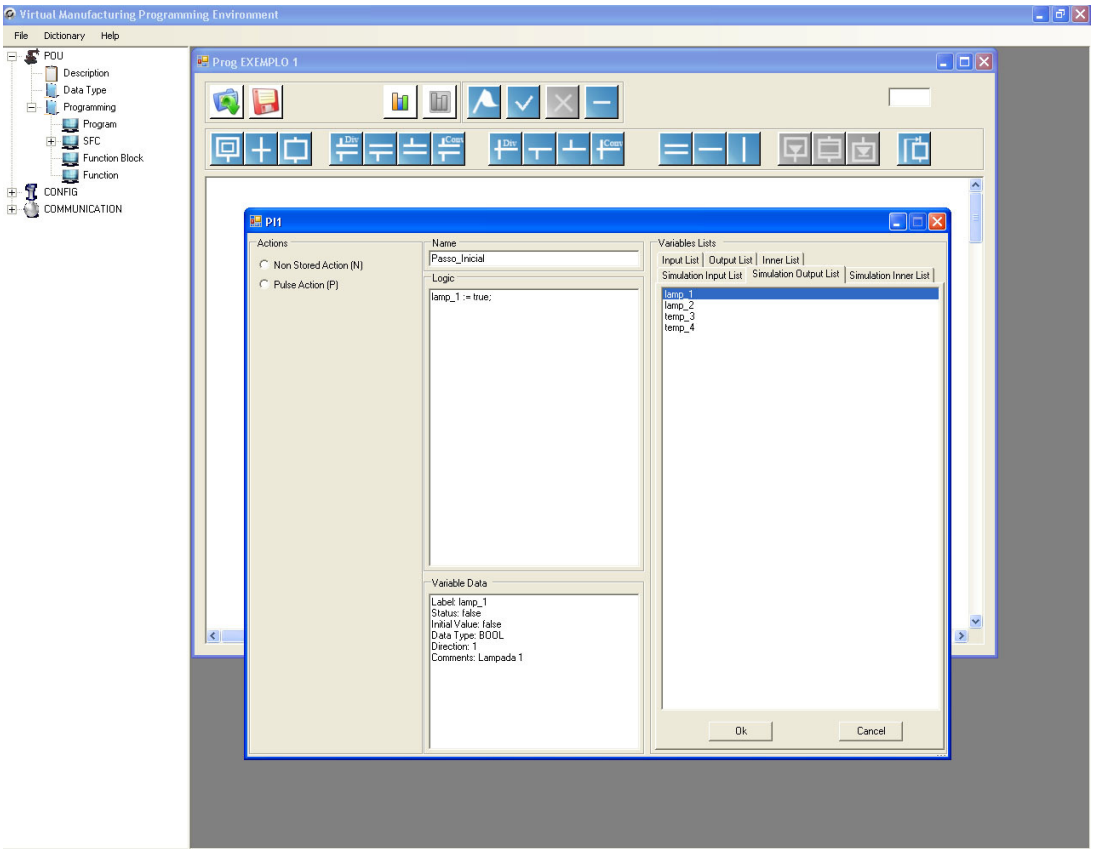


Figura 79 - Editando um passo

Nas abas *Simulation*, são listadas as variáveis criadas. No campo *Name,* o usuário entra com o nome do elemento. E no campo *Logic*, é inserida a lógica. Após isso o mesmo clica em OK. E o mesmo deve ser feito para os demais elementos.

O diagrama SFC do exemplo é o seguinte (Figura 80):

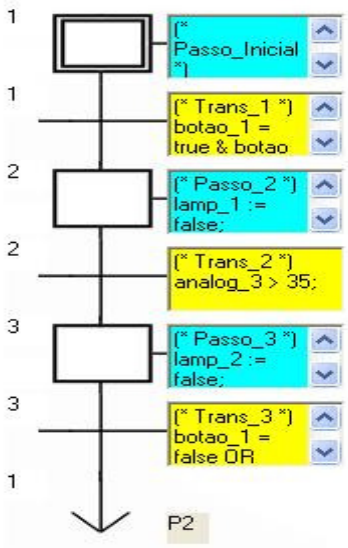


Figura 80 - Diagrama SFC do Exemplo 1

Para o caso do elemento *Jump*, o usuário clica nele e uma tela aparece mostrando os possíveis elementos que o mesmo pode indicar. No exemplo, está indicando para o passo *P2* (Figura 81).

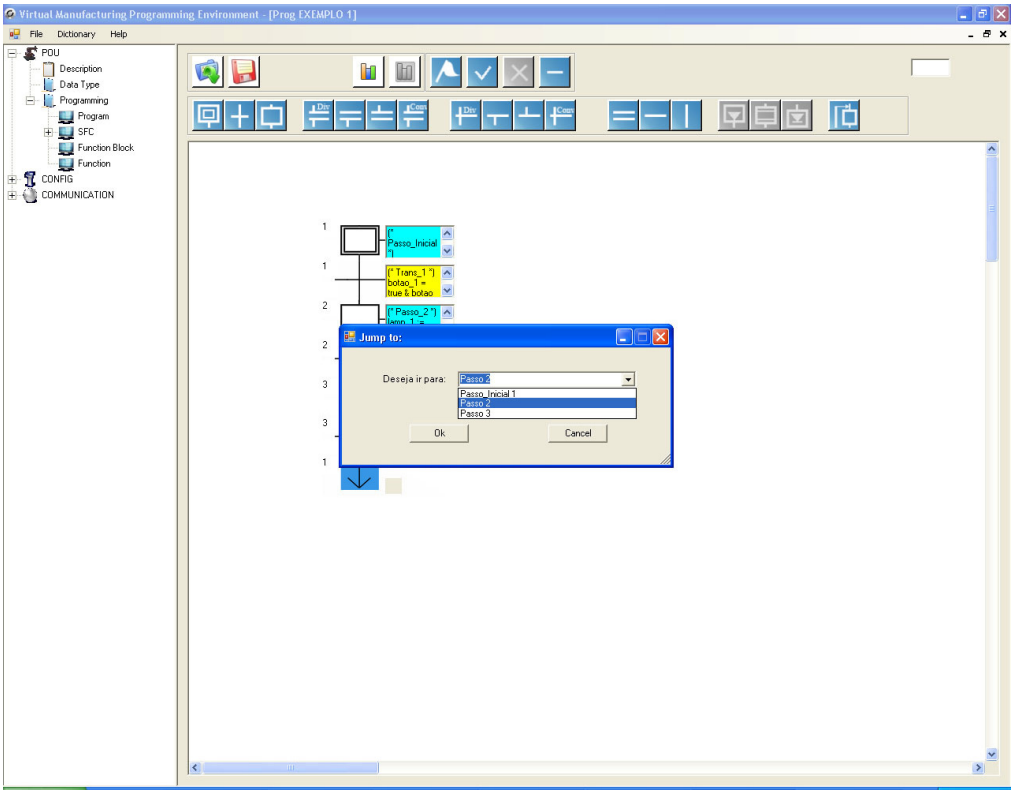


Figura 81 - Escolhendo o elemento apontado pelo jump

A lógica para cada estado e transição pode ser vista na Tabela 45:

Tabela 45 - Elementos e lógicas

ELEMENTO	LÓGICA
Passo_Inicial	lamp_1 := true;
Trans_1	botao_1 = true & botao_2 = true;
Passo_2	lamp_1 := false; lamp_2 := true; temp_4 := 31;
Trans_2	analog_3 > 35;
Passo_3	lamp_2 := false; temp_3 := 25;
Trans_3	botao_1 = false OR botao_2 = false;

Para se iniciar a execução desta aplicação, o usuário clica no botão *Rodar com Variáveis de Simulação*, e aparece assim, a tela de verificação de erro, como na Figura 82. Após isso, o *Passo_Inicial* é ativado.

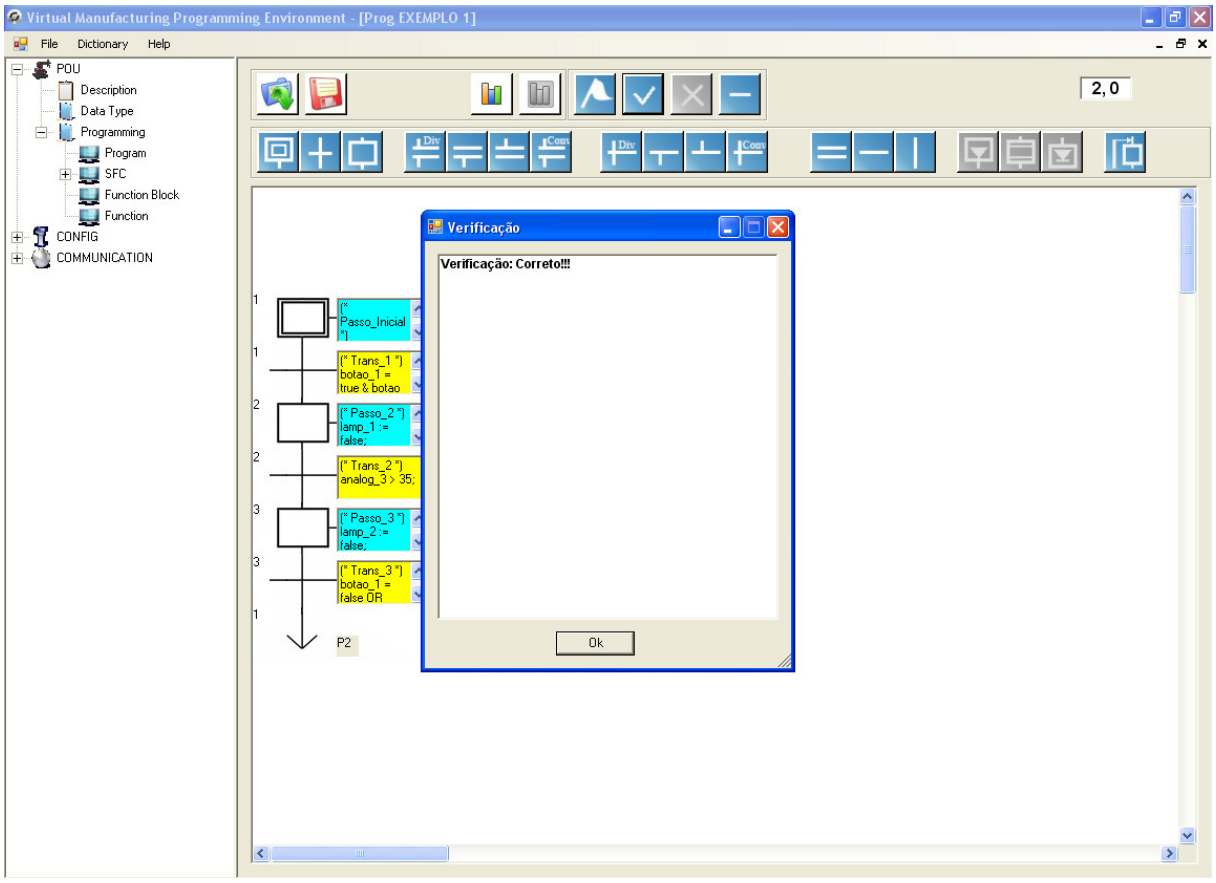


Figura 82 - Iniciando execução do programa

Com a ativação do primeiro estado, a lógica que está contida nele é aplicada, ou seja, *lamp_1* é acionada. E como o estado inicial de *lamp_2* é *true*, logo, ao abrir a tela *Running*, ambas as lâmpadas estarão ligadas. Pode-se observar na Figura 83.

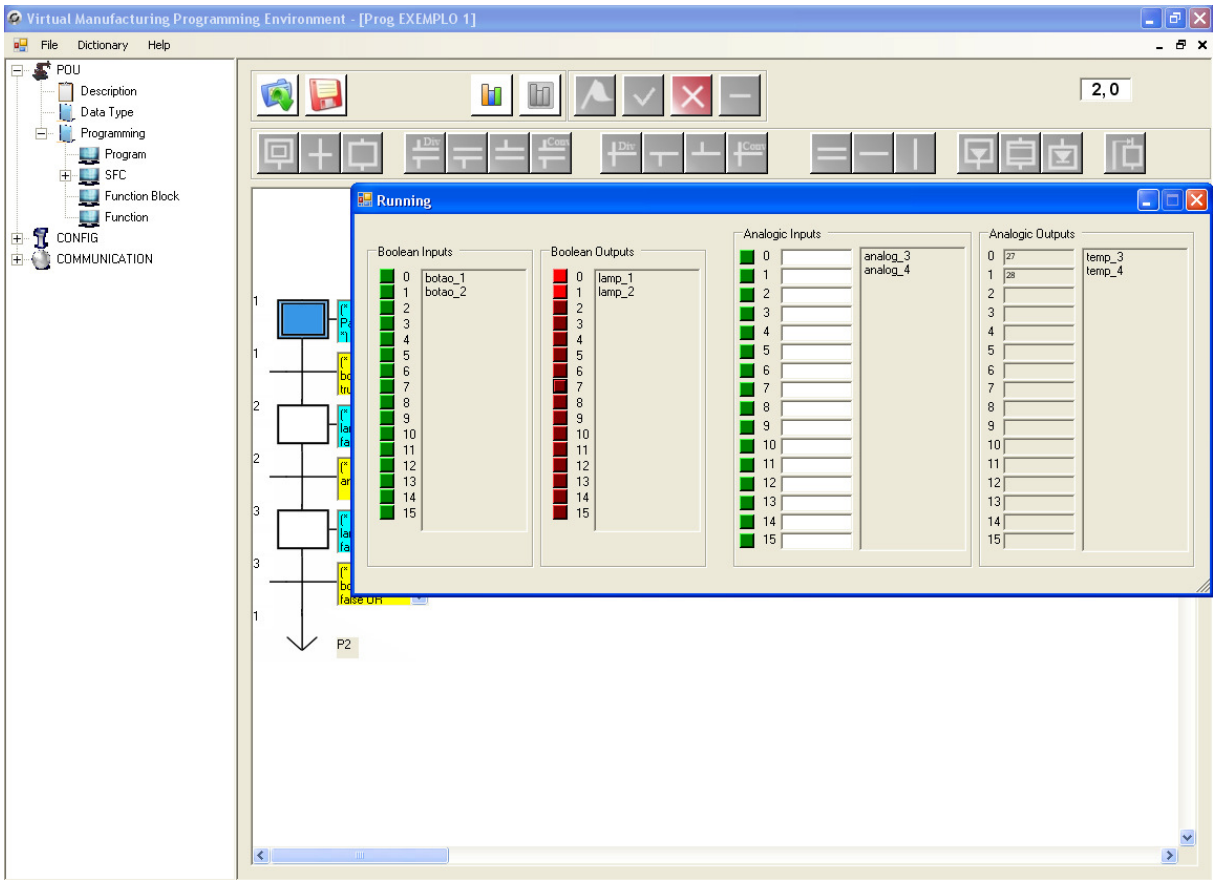


Figura 83 – Passo_Inicial ativo

O próximo passo é o *Passo_2*. Para ele ser ativado, e consequentemente o *Passo_Inicial* ficar inativo, a condição de *Trans_1* deve ser satisfeita. Esta condição diz que para que isto ocorra, *botao_1* e *botao_2* devem ser acionados.

Verifica-se que ao se acionar apenas um dos dois botões, *Passo_2* ainda fica inativo (Figura 84):

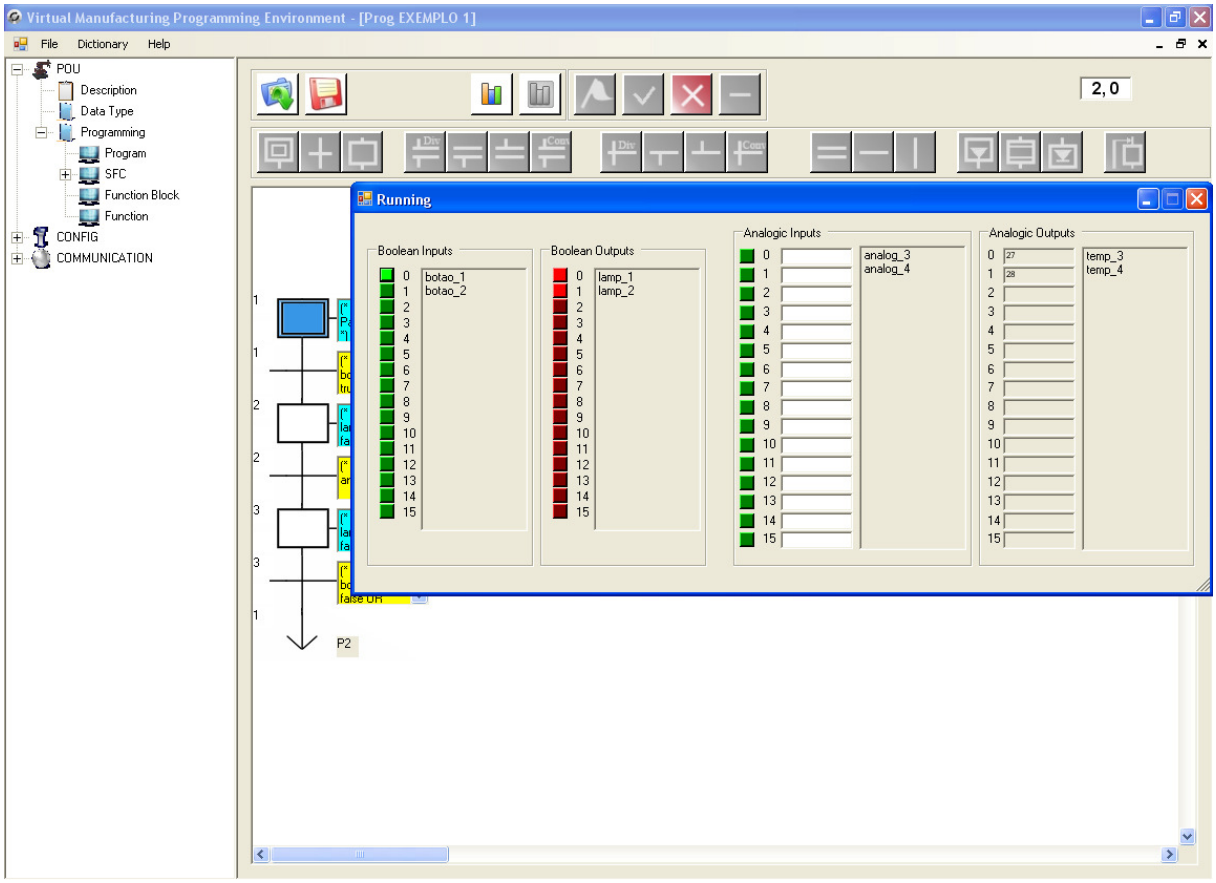


Figura 84 – Passo_Inicial ainda ativo

Ao se acionar os dois botões, a condição foi satisfeita e o *Passo_2* ficou ativo. A lógica inserida nele é a seguinte:

```
lamp_1 := false; lamp_2 := true; temp_4 := 31;
```

A Figura 85 mostra as saídas após a aplicação desta lógica:

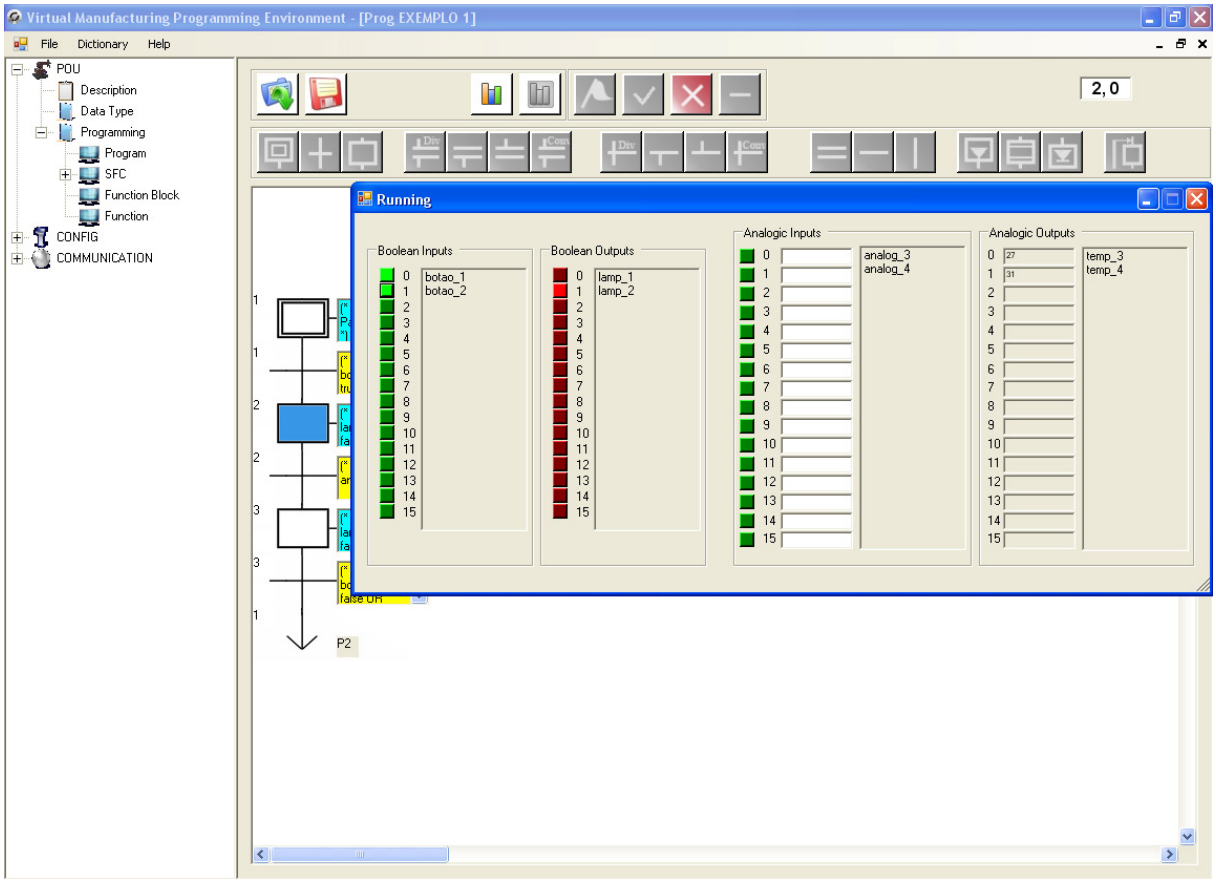


Figura 85 – Passo_2 ativo

Para que *Passo_3* seja ativado, a seguinte condição deve ser satisfeita:

```
analog_3 > 35;
```

Caso o usuário agora atue na entrada analógica, inserindo um valor maior que 35, *Trans_3* é satisfeita.

Pode-se verificar na Figura 86 o sistema após o valor de *analog_3* ser maior que 35:

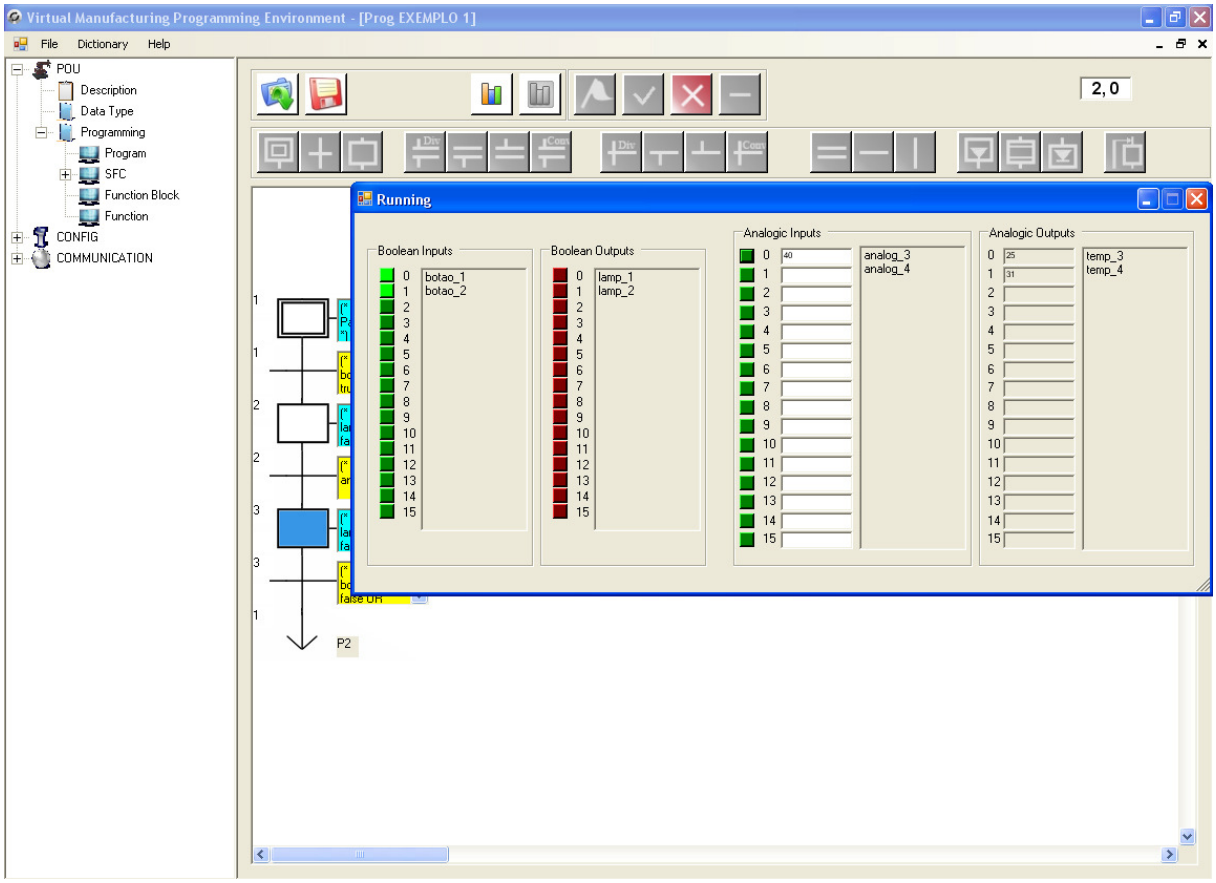


Figura 86 - Passo_3 ativo

O *Passo_3* agora ficou ativo, e a lógica associada a ele foi executada:

```
lamp_2 := false; temp_3 := 25;
```

A próxima transição apresenta a seguinte condição:

```
botao_1 = false OR botao_2 = false;
```

Que significa que caso sejam desativados o *botao_1* ou *botao_2*, ocorrerá um *jump* do *Passo_3* para o *Passo_2*. Verifica-se isto na Figura 87.

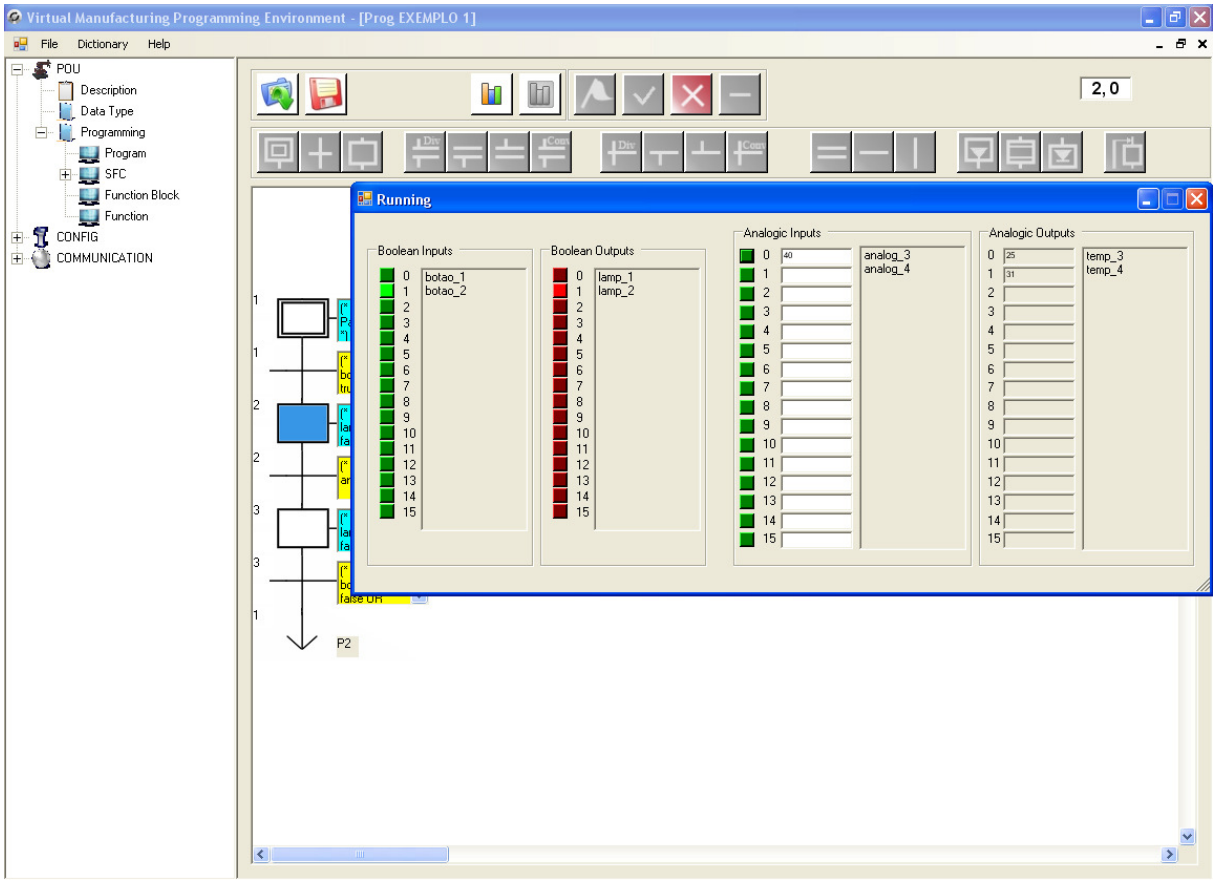


Figura 87 - Passo_2 ativo após o jump

Como a condição de *Trans_2* é que *analog_3* seja maior que 35, e o valor desta variável está em 40, o programa ficou alternando entre *Passo_2* e *Passo_3*, pois a condição de *Trans_3* também está sendo satisfeita.

Finalmente, para se encerrar a aplicação, fecha-se a tela *Runnig*, e o botão *Stop* deve ser acionado. Para se salvar o programa, o usuário deve clicar em *Arquivo* e *Salvar*, ou no próprio botão *Salvar*.

EXEMPLO 2 – MODO CONECTADO À MAQUETE

Este exemplo conta com dois estados e uma transição. É um exemplo simples para demonstrar a interação entre o ambiente de programação e a maquete virtual.

Inicialmente o usuário deve abrir o *software*, como descrito para o exemplo anterior. Porém neste caso, ele deve abrir também a maquete virtual. O arquivo da maquete está na pasta *SOFTWARE*. Dentro dela há a pasta *Maquete*, que contém tal arquivo, denominado *VirtualManufacturing.* Deste modo a maquete é aberta.

O usuário deve dar um clique simples na tela da maquete e usar o botão central do mouse para a aproximação da mesma.

Já no ambiente de programação, o procedimento inicial é o mesmo do exemplo anterior. O usuário abre a tela de programação. Mas neste modo, após isso, o usuário deve integrar o ambiente à maquete. E para isto, inicialmente, o mesmo deve entrar com a *string* de comunicação: *tcp://localhost:8080/virtualserver*

O usuário clica em *Communication* com o botão direito do mouse e entra com esta *string*, clicando em *Add*.

Isto pode ser observado nas Figuras 88 e 89.

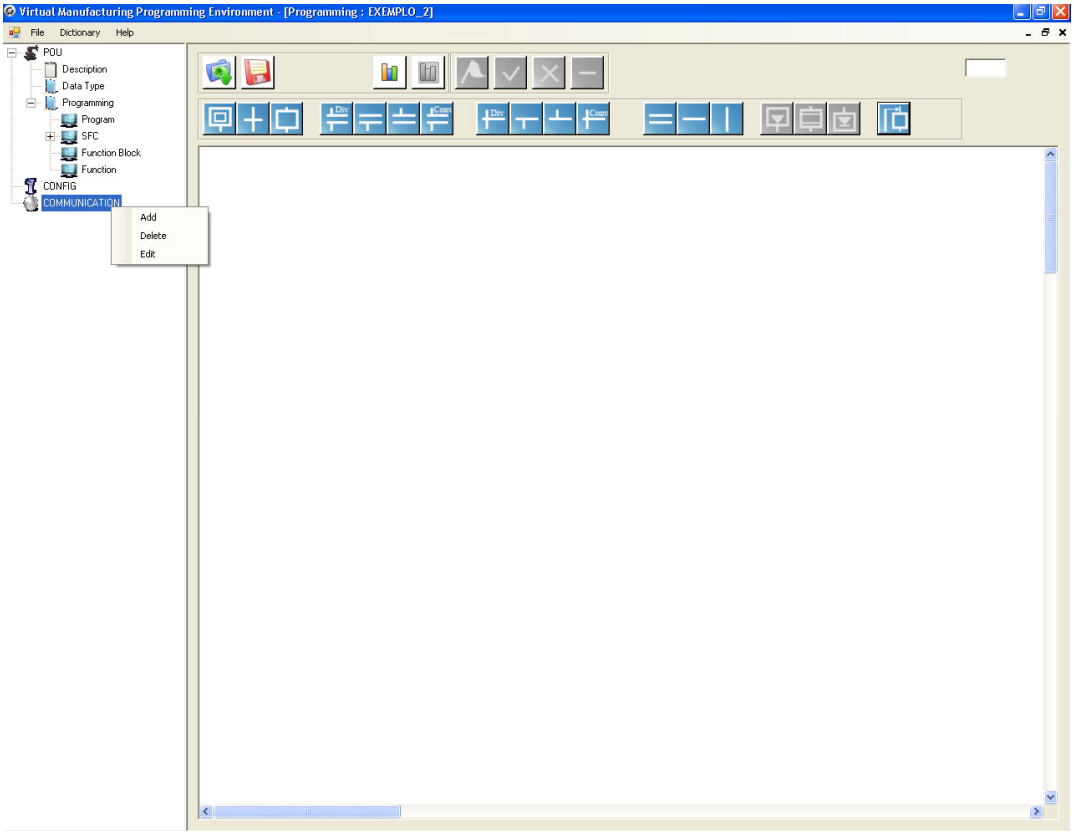


Figura 88 - Entrando com a string de comunicação

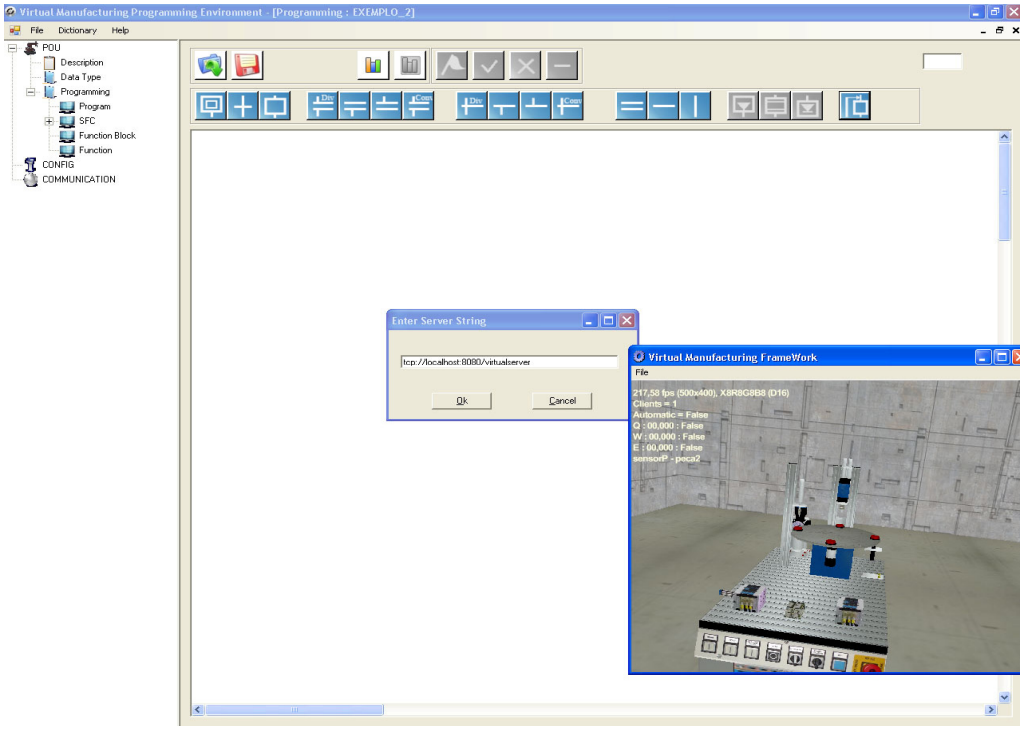


Figura 89 - String de comunicação

Ao se clicar em OK, deve-se clicar também na tela da maquete. Caso a *string* tenha sido digitada de maneira correta, haverá um acréscimo em Cliente, na tela da maquete, e o endereço ficará da cor azul na tela do ambiente, como na Figura 90.

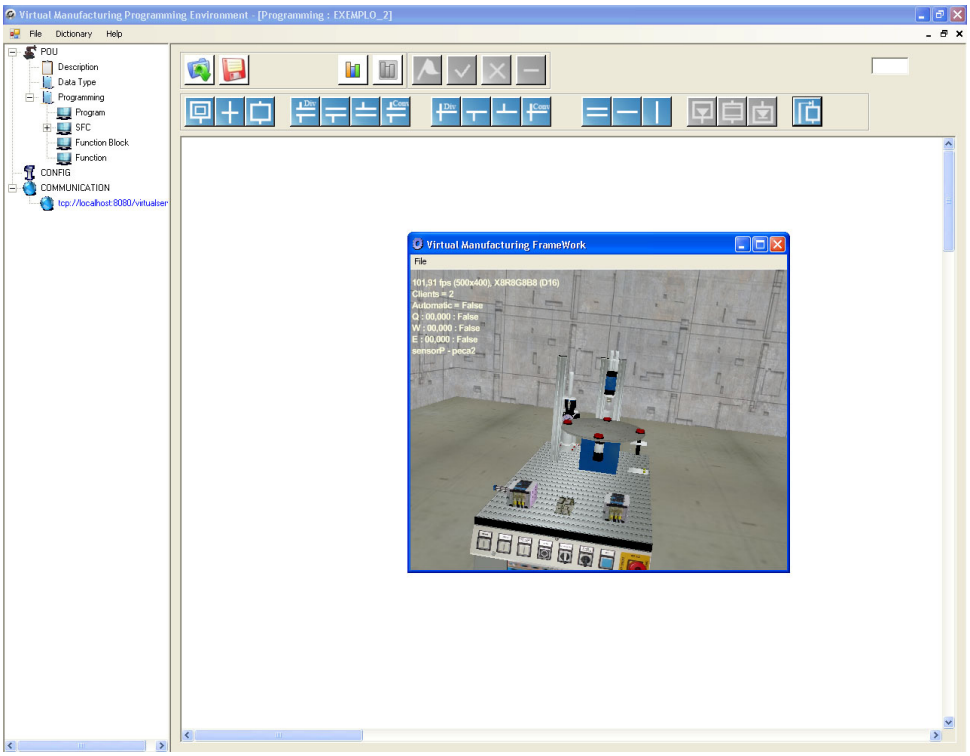


Figura 90 - Comunicação estabelecida

E para o usuário carregar a tabela com as variáveis da maquete, ele deve efetuar o mesmo procedimento, porém clicando em *Config* e depois em *Add.* O arquivo a ser carregado se chama *acoesesensores.xml* e se encontra na pasta *Media*, dentro da pasta *SOFTWARE*.

Para visualizar a tabela, o usuário deve clicar em *Data Type*. As Tabelas 46 e 47 são carregadas.

Tabela 46 - Variáveis de entrada vindas da maquete

Input						
	Labels	Status	Initial Value	Data Types	Direction	Comment
▶	sfc1f	False	False	BOOL	0	furad_fura_festo + s
	sfc1t	False	False	BOOL	0	sensorteste + test_fu
	spp	False	False	BOOL	0	sensorP + peca1
	spp	False	False	BOOL	0	sensorP + peca2
	spp	False	False	BOOL	0	sensorP + peca3
	spp	False	False	BOOL	0	sensorP + peca4
	sfura	False	False	BOOL	0	drillfuradeira + p1_fu
	sfura	False	False	BOOL	0	drillfuradeira + p2_fu
	sfura	False	False	BOOL	0	drillfuradeira + p3_fu
	sfura	False	False	BOOL	0	drillfuradeira + p4_fu
	sfura	False	False	BOOL	0	drillfuradeira + band
	stesta	False	False	BOOL	0	pontateste + p1_fura
	stesta	False	False	BOOL	0	pontateste + p2_fura
	stesta	False	False	BOOL	0	pontateste + p3_fura
	stesta	False	False	BOOL	0	pontateste + p4_fura
	q_enc	False	00,000	FLOAT	0	Q encoder
	w_enc	False	00,000	FLOAT	0	W encoder
	e_enc	False	00,000	FLOAT	0	E encoder
*						

Tabela 47 - Variáveis de saída vindas da maquete

Output						
	Labels	Status	Initial Value	Data Types	Direction	Comment
▶	q_up	False	00,000	BOOL	1	Q
	q_down	false	00,000	BOOL	-1	Q
	w_up	False	00,000	BOOL	1	W
	w_down	False	00,000	BOOL	-1	W
	e_up	False	00,000	BOOL	1	E
	e_down	False	00,000	BOOL	-1	E
*						

Os procedimentos para a montagem do diagrama e edição dos elementos são os mesmo do exemplo anterior, diferenciando-se apenas no momento da execução da aplicação. Pois neste caso o botão que deve ser acionado é *Rodar*.

O diagrama SFC do exemplo é o seguinte (Figura 91):

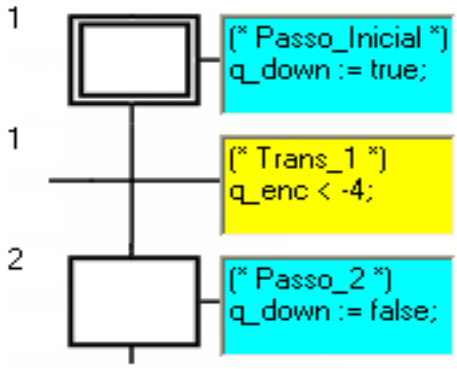


Figura 91 - Diagrama SFC do Exemplo 2

Iniciando a aplicação, o *Passo_Inicial* é ativado. O estado de *q_down* fica *true*. As tabelas são atualizadas e enviadas à maquete, e a mesa começa a girar. A Figura 92 mostra a mesa no estado inicial, e na Figura 93, observa-se a mesa após o acionamento do estado 1.

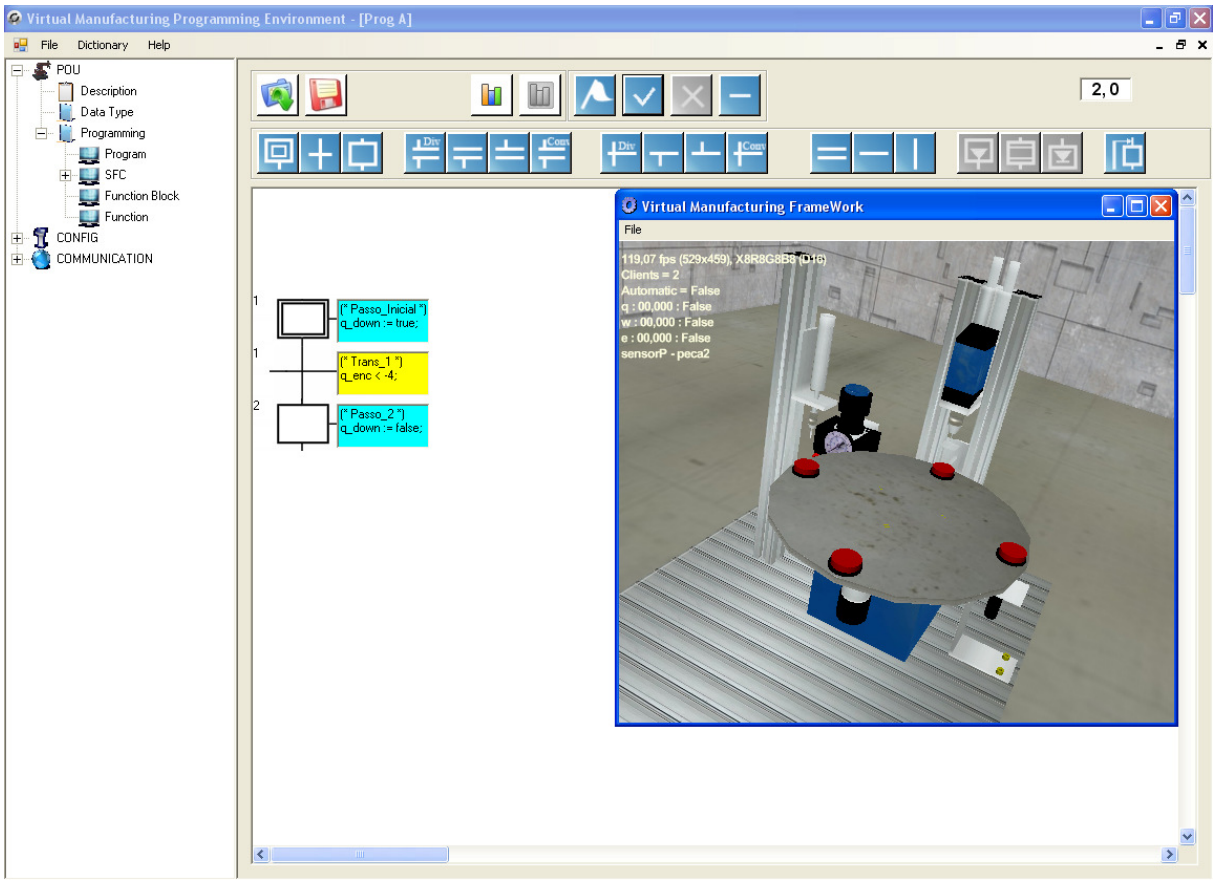


Figura 92 - Mesa no estado inicial

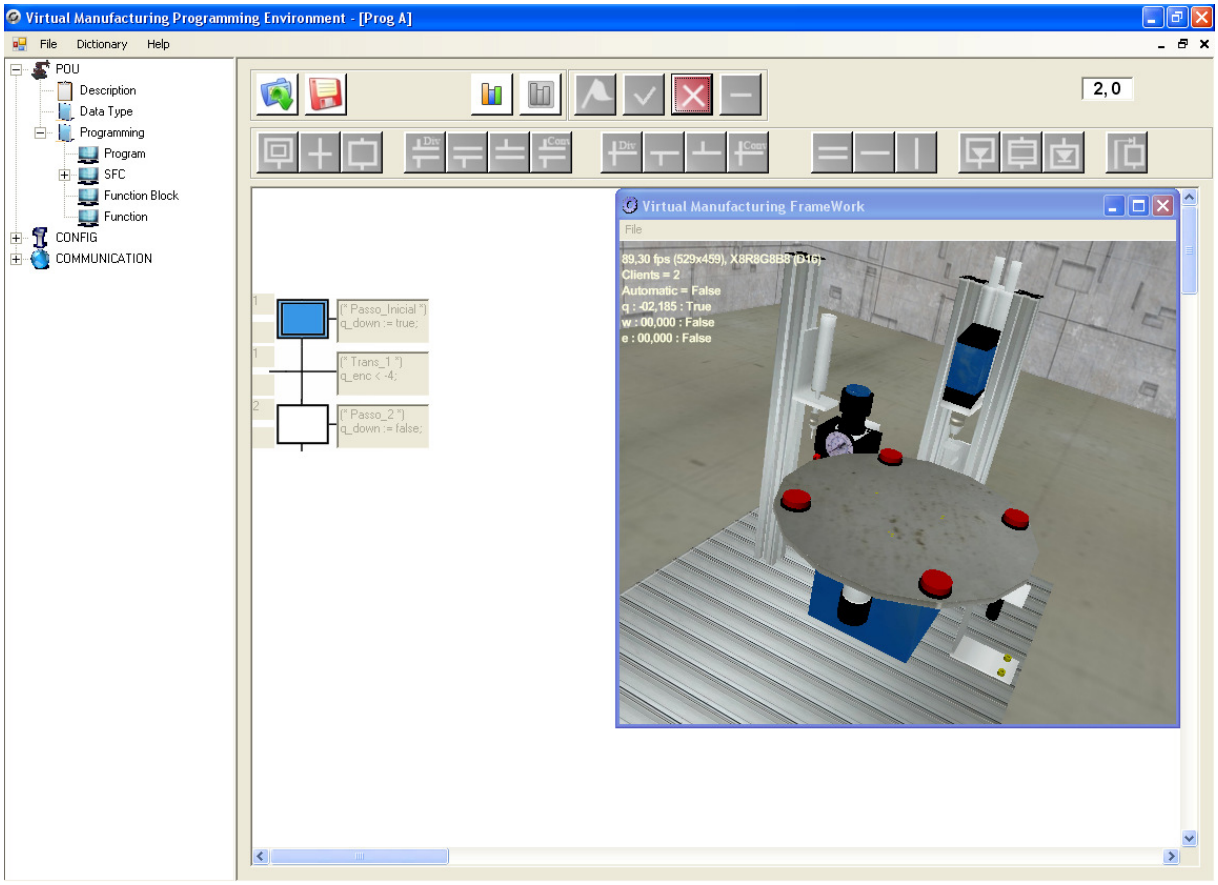


Figura 93 - Passo_Inicial ativo

A transição seguinte tem associada a ela a condição:

```
q_enc < -4;
```

A mesa no estado inicial apresenta o valor de *q_enc* igual a 0. Quando começa a girar, *q_enc* decresce. Logo, quando seu valor for menor que -4, a condição de *Trans_1* é satisfeita, e o *Passo_2* é ativado.

A ação associada ao *Passo_2* faz parar a mesa (Figura 94):

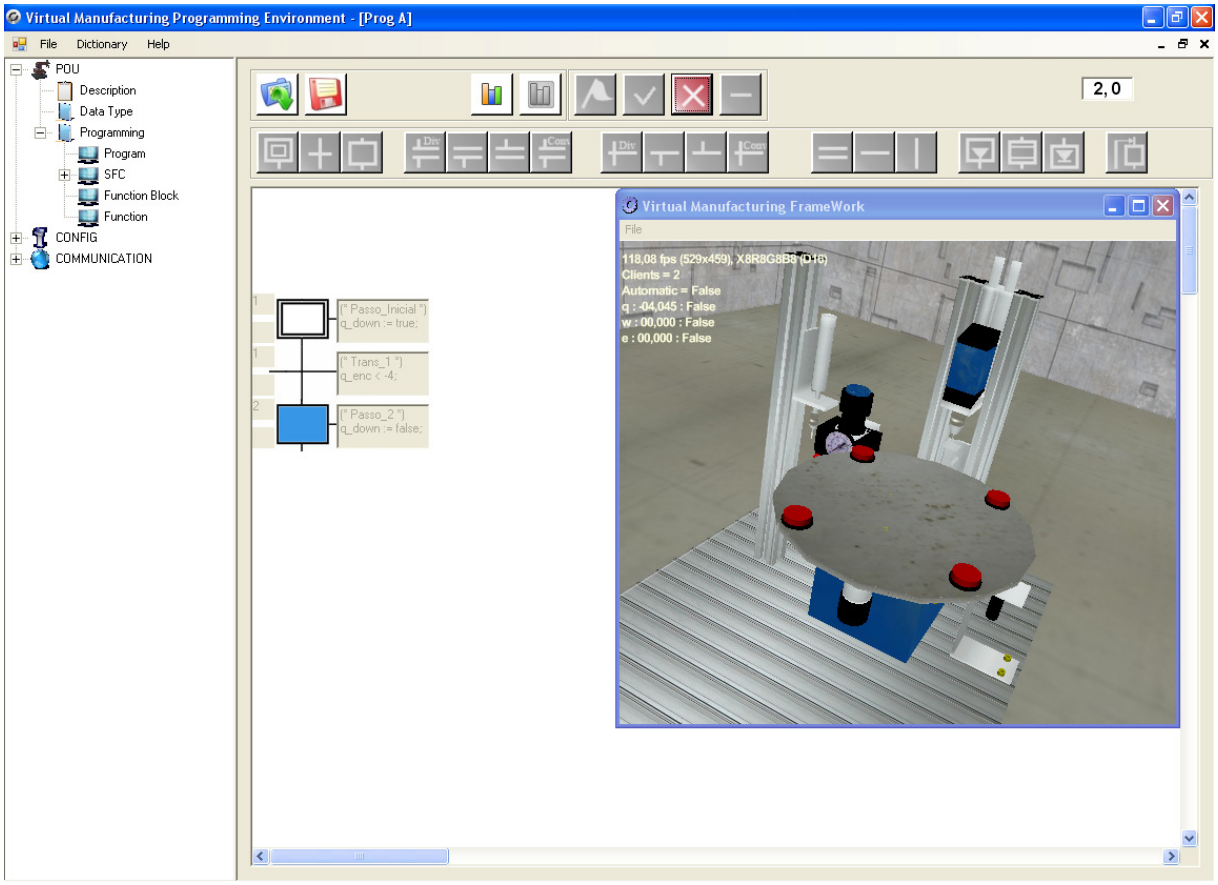


Figura 94 - Passo_2 ativo

Para se encerrar a aplicação, basta clicar no botão Stop. E para fechar a maquete, deve se pressionar a tecla esc. O procedimento de salvar é o mesmo do exemplo anterior.

EXEMPLO 3 – MODO CONECTADO À MAQUETE USANDO AÇÕES (S) E (R)

O exemplo 3 mostra um diagrama um pouco menos simples que o do exemplo anterior, e nele será mostrada a aplicação das ações (S) e (R) na maquete virtual.

O diagrama SFC em questão é o seguinte (Figura 95):

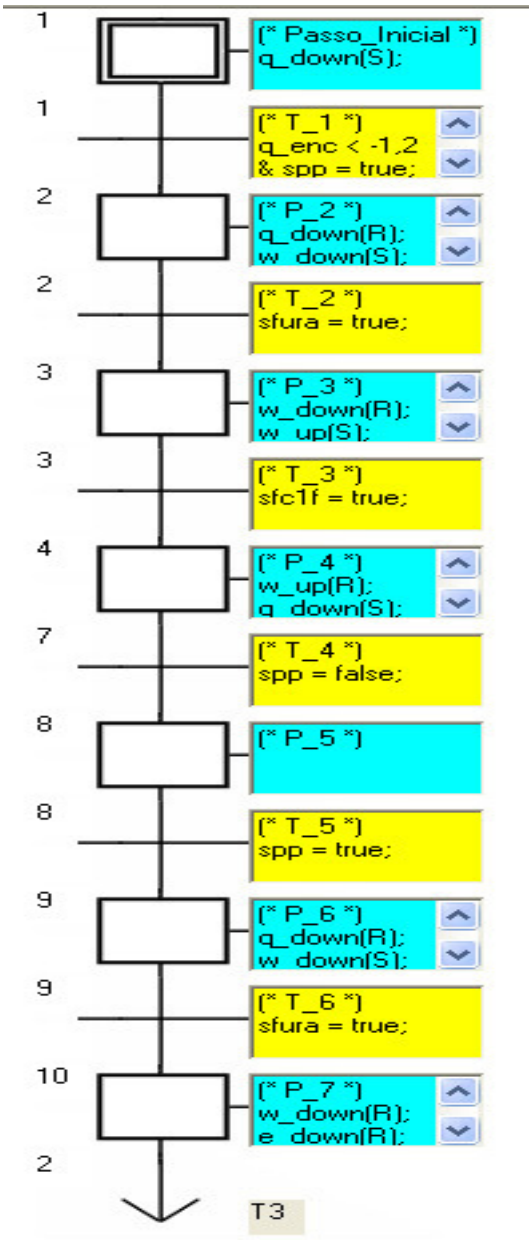


Figura 95 - Diagrama SFC do Exemplo 3

A Tabela 48 mostra todos os estados e transições com suas respectivas lógicas de execução.

Tabela 48 - Elementos SFC e suas respectivas lógicas

ELEMENTO	LÓGICA
Passo_Inicial	q_down(S);
T_1	q_enc < -1,2 & spp = true;
P_2	q_down(R); w_down(S);
T_2	sfura = true;
P_3	w_down(R); w_up(S);
T_3	sfc1f = true;
P_4	w_up(R); q_down(S); e_up(R);
T_4	spp = false;
P_5	
T_5	spp = true;
P_6	q_down(R); w_down(S); e_down(S);
T_6	sfura = true;
P_7	w_down(R); e_down(R); w_up(S); e_up(S);

As variáveis usadas aqui são as mesmas do Exemplo 2, bem como o estado inicial da mesa.

Quando o programa é iniciado, *Passo_Inicial* é ativado e sua ação é executada. Logo a variável *q_down* sofre uma ação *set*, ou seja, seu status passa para *true*, o que faz com que a mesa comece a girar (Figura 96).

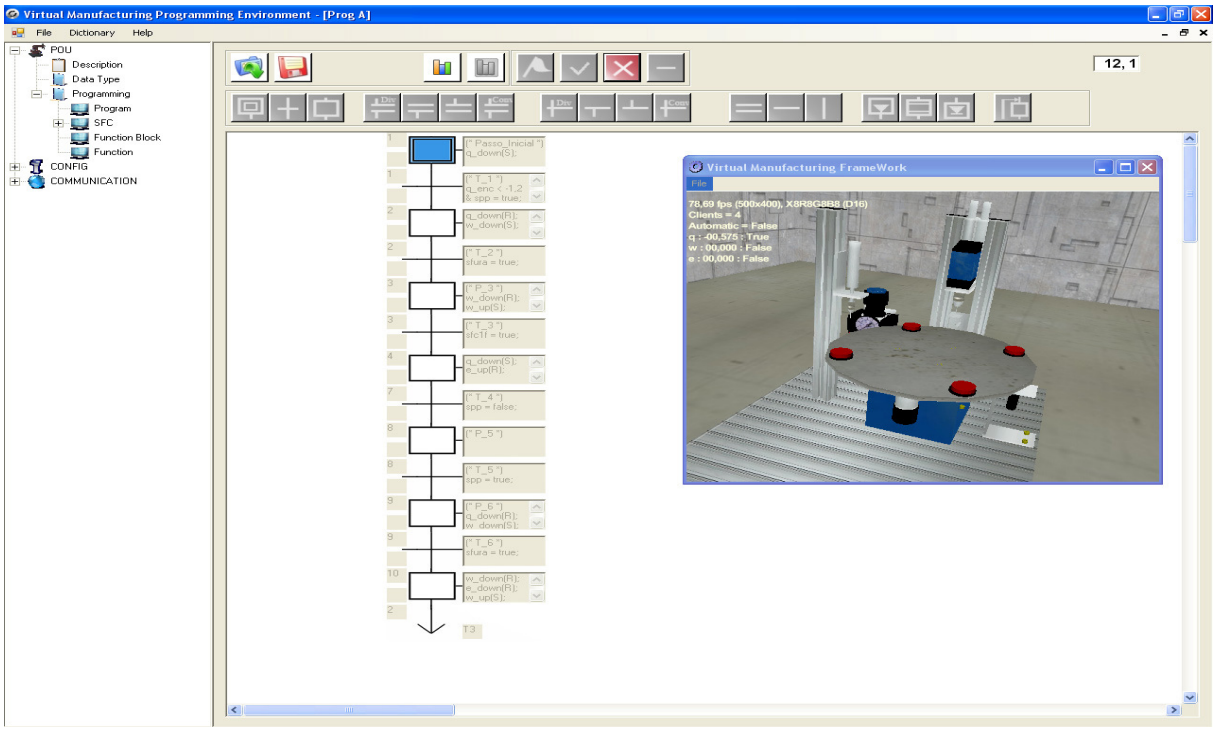


Figura 96 - Passo_Inicial ativo

A condição para que *T_1* seja satisfeita é que o valor de *q_enc* seja menor que -1,2, e que o status do sensor de presença *spp* passe para *true*. Ocorrendo isto, *Passo_Inicial* fica inativo e *P_2* é agora ativado.

Quando *P_2* fica ativo, *q_down* sofre uma ação de *reset*, que faz parar a mesa, e *w_down* sofre a ação *set*. Isto significa que a furadeira desce. Observa-se isto na Figura 97.

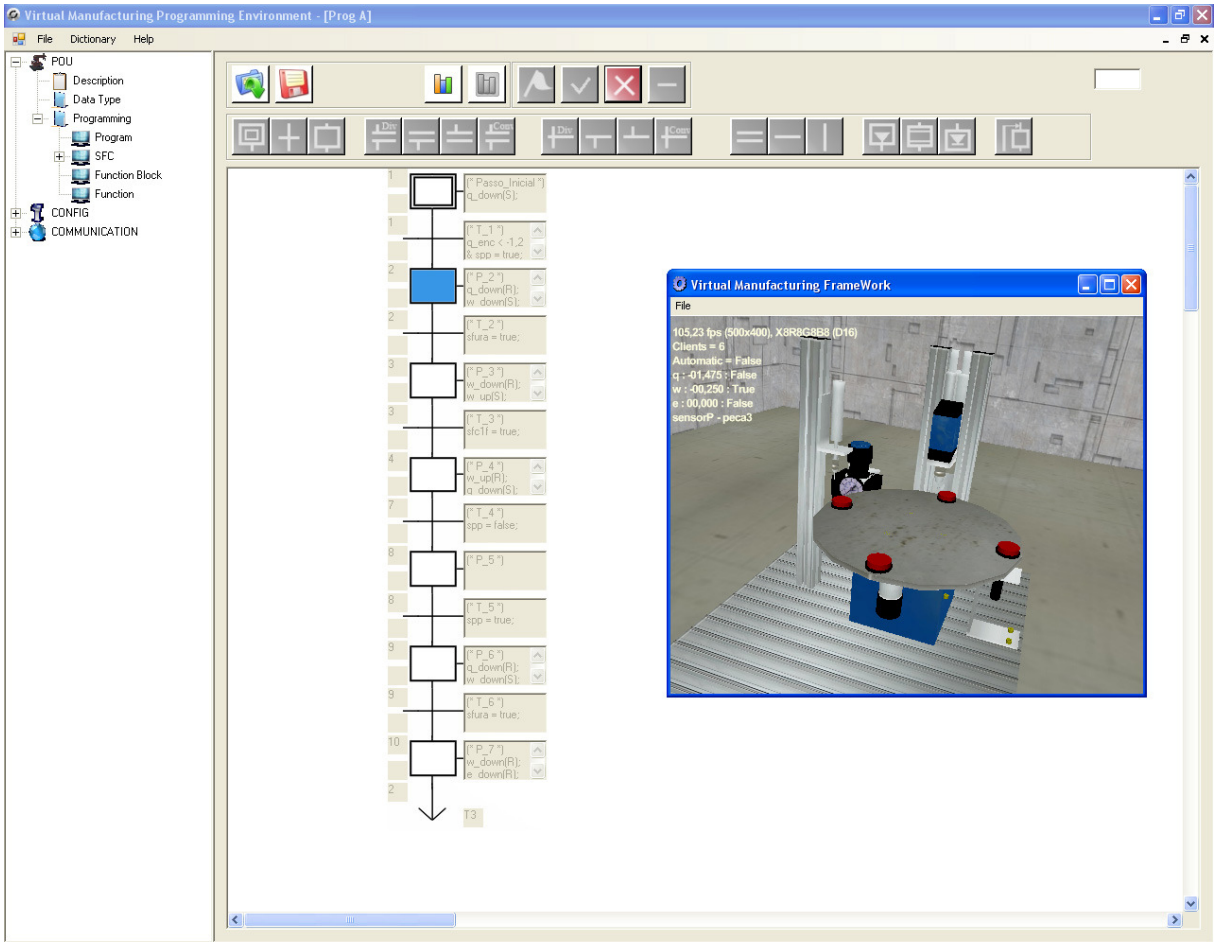


Figura 97 - P_2 ativo

Quando a furadeira desce e atinge a peça, há um sensor que também é ativado, o *sfura*. E a condição associada à *T_2* é satisfeita justamente quando este passa para *true*.

Ocorrendo isto, *P_3* se torna ativo. A ação associada a este estado dá um *reset* em *w_down* e seta a variável *w_up*, isto é, a furadeira para de descer e começa a subir, como na Figura 98.

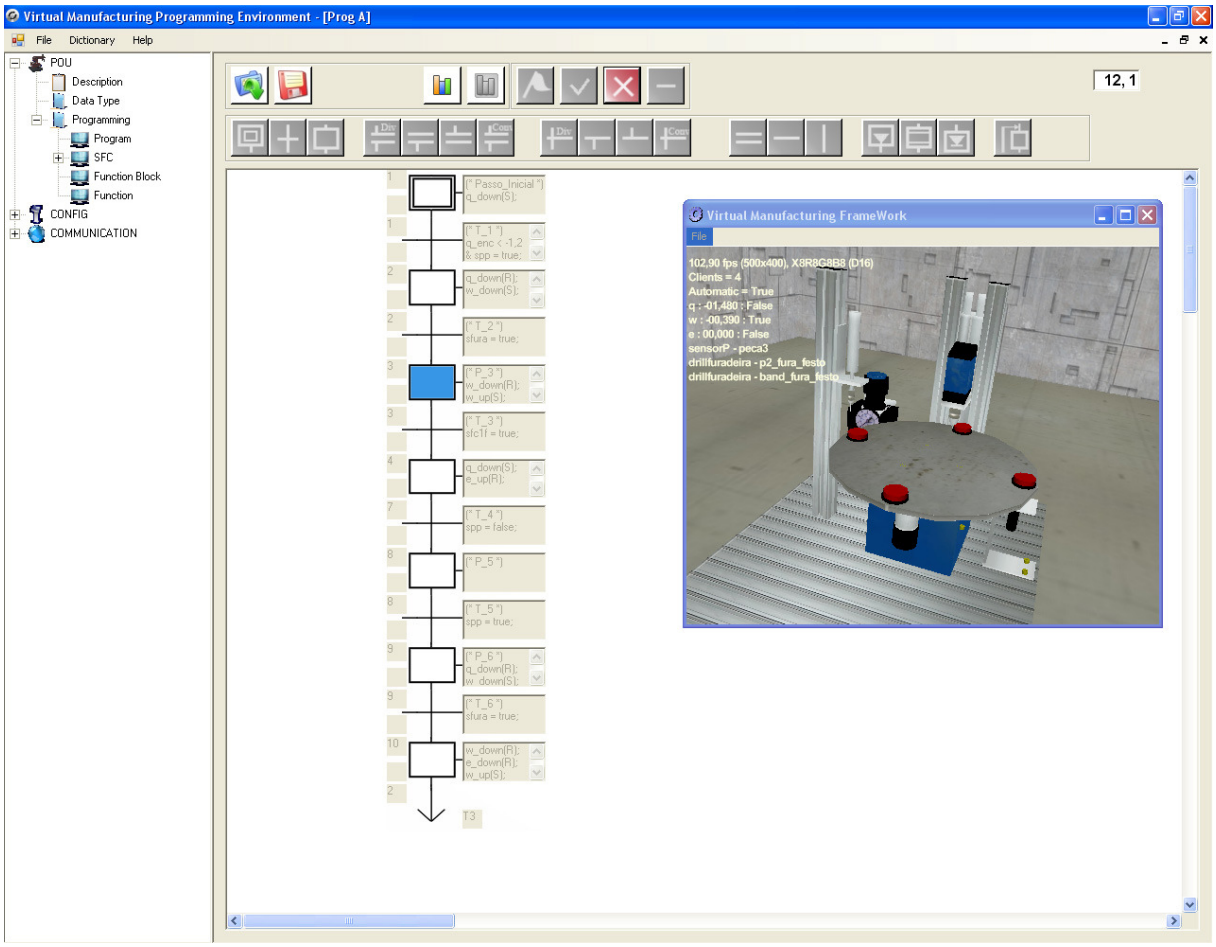


Figura 98 - P_3 ativo

A condição imposta a *T_3* é que o sensor *sfc1f* seja ativado, ou seja, quando a furadeira subir até seu limite máximo. Quando isto ocorre, *P_4* fica ativo, como na Figura 99.

Neste estado, *w_up* sofre ação (R), parando assim de subir, e a mesa volta a girar com a ação (S) em *q_down*. A variável *e_up* sofre uma ação (R), devido ao *jump*, que leva *P_7* de volta a *P_4*.

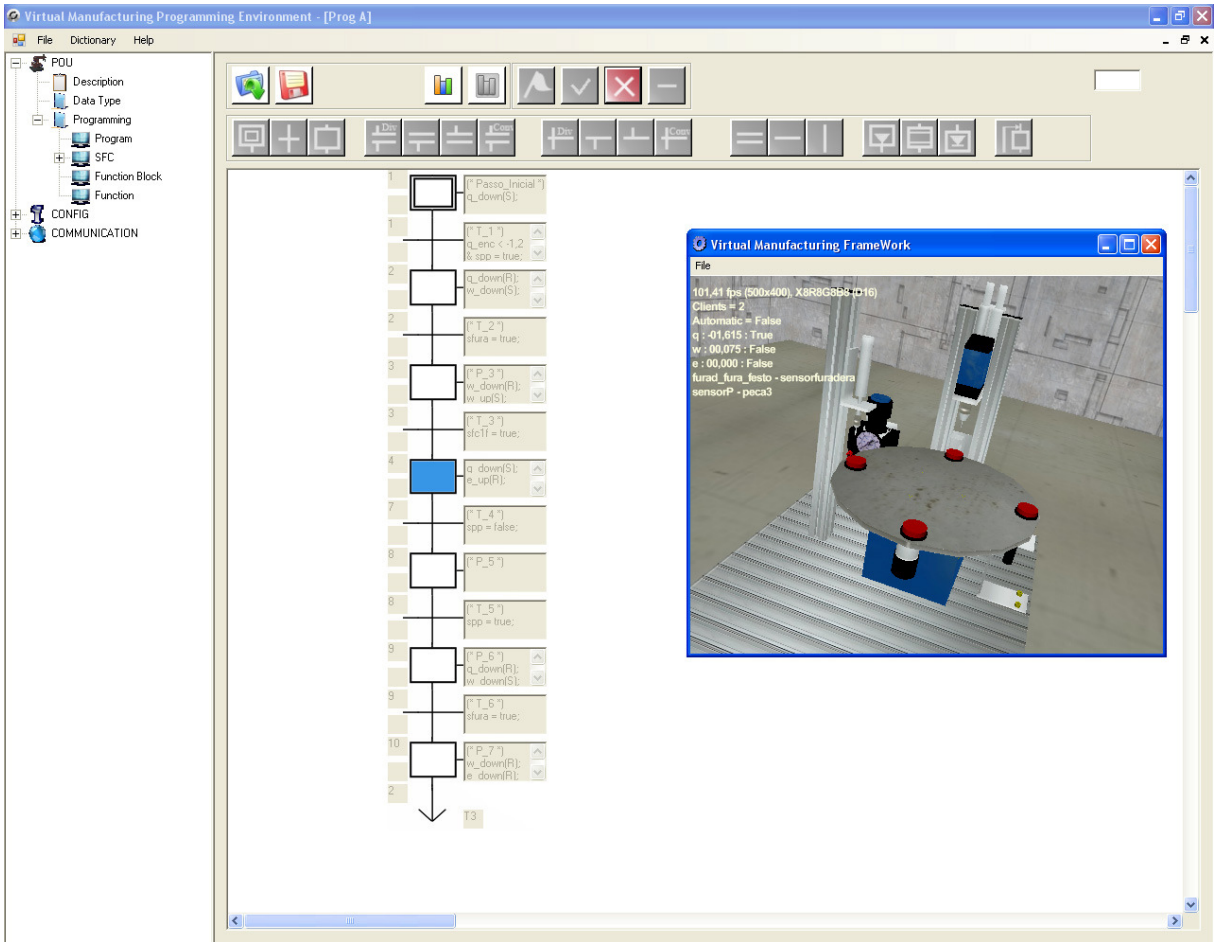


Figura 99 - P_4 ativo

Neste momento, a condição de *T_4* é satisfeita quando o sensor de presença de peças fica inativo, o que resulta na ativação de *P_5*. Não está associada a este estado nenhuma ação ou lógica, pois agora a mesa apenas fica girando. Observa-se a Figura 100.

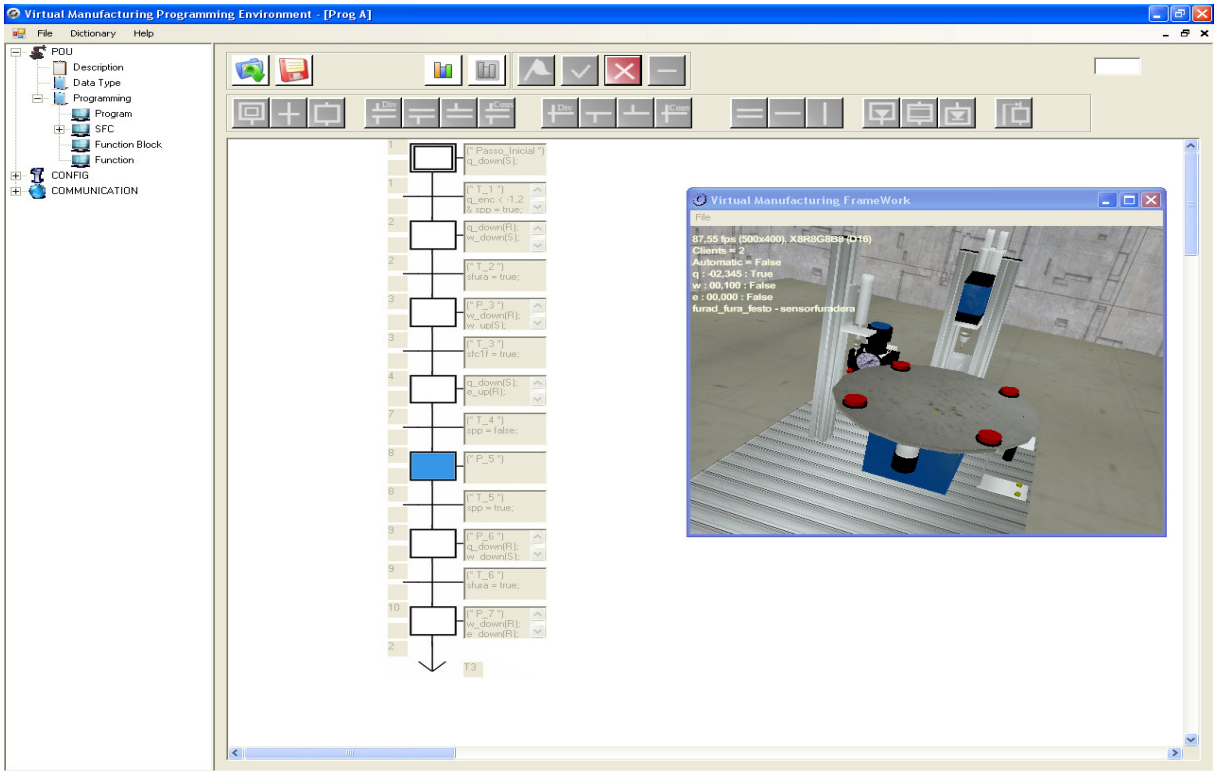


Figura 100 - P_5 ativo

Para *T_5* ser satisfeita, o sensor de presença de peças deve ser ativado. Assim, *P_6* é acionado, e *q_down* sofre a ação (R), o que faz parar a mesa novamente; *w_down* sofre uma ação *set*, levando a furadeira a descer; e *e_down* sofre também uma ação (S), que resulta no acionamento do ponto de teste de furo. Observa-se na Figura 101.

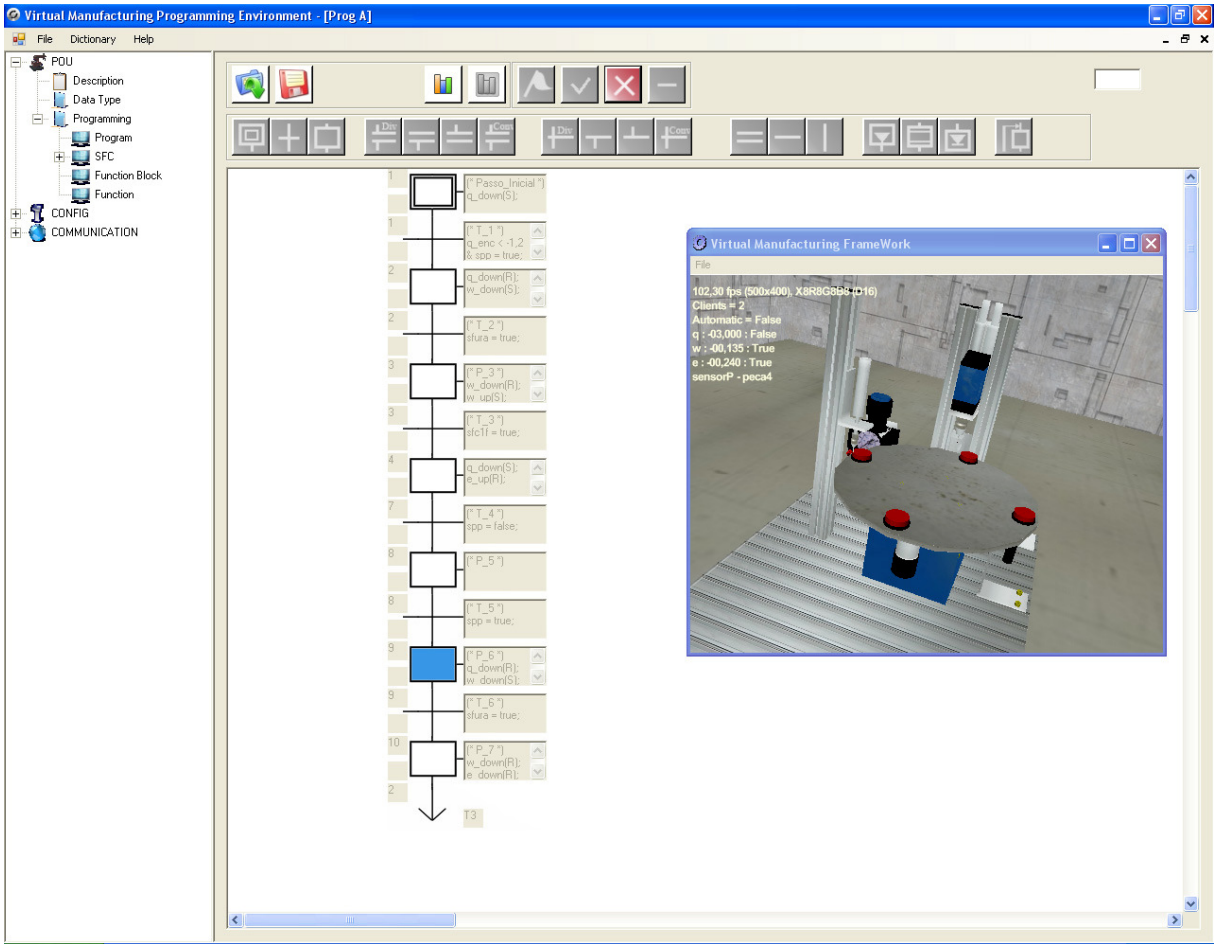


Figura 101 - P_6 ativo

Agora, a condição associada à *T_6* é o acionamento do sensor *sfura*, ou seja, quando a furadeira atingir novamente uma peça. Sendo satisfeita, *P_7* se torna ativo, e *w_down* sofre um *reset*, parando a furadeira; *w_up* sofre um *set*, levando a furadeira a subir; *e_down* sofre uma ação (R), ou seja, o ponto de teste de furo pára, e finalmente *e_up* sofre uma ação (S), fazendo o ponto de teste começar a subir (Figura 102):

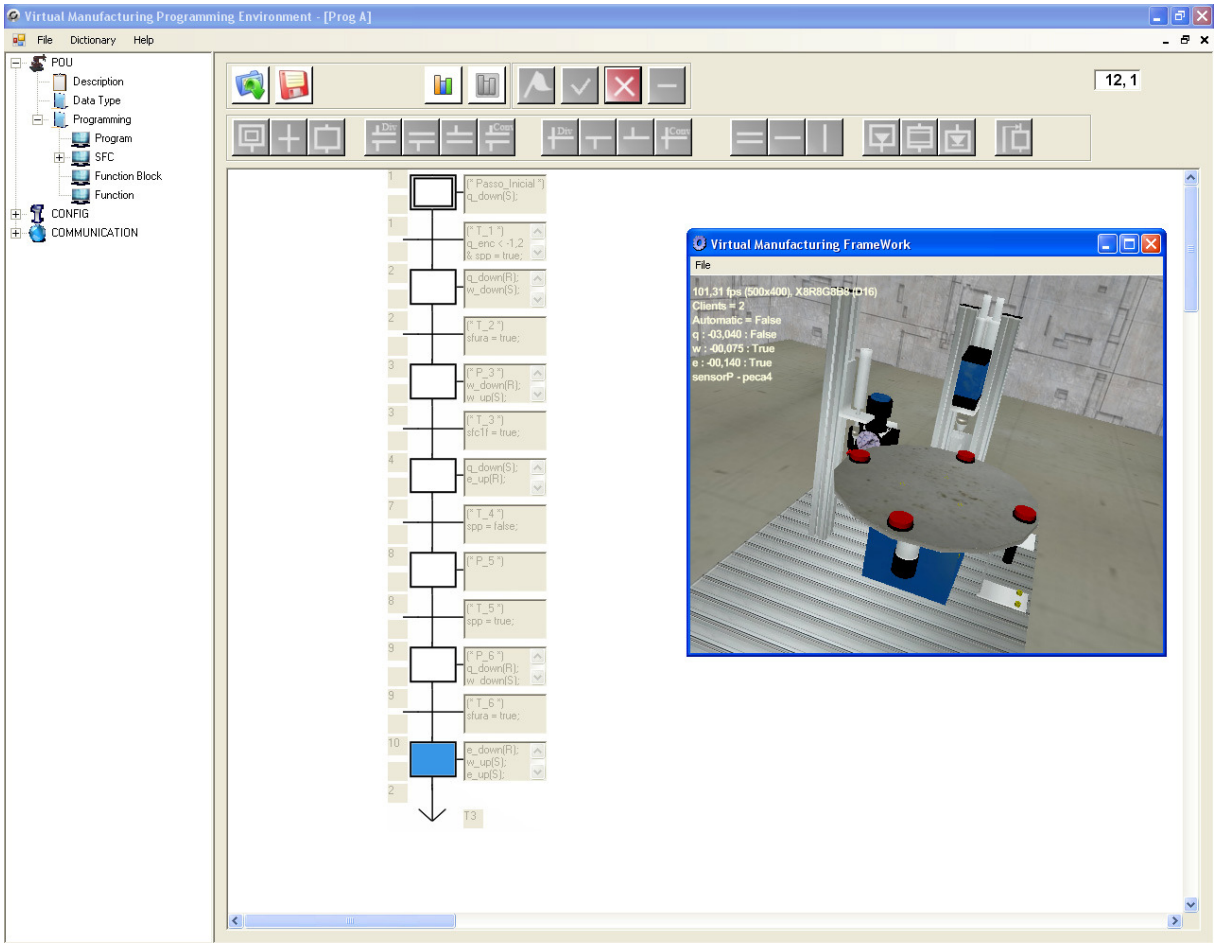


Figura 102 - P_7 ativo

Acontece neste instante um *jump* do passo *P_7* para a transição *T_3*. Como já visto anteriormente, a condição de *T_3* é que o sensor *sfc1f* seja ativado, ou seja, quando a furadeira subir até seu limite máximo.

Ocorrendo isto, *P_4* torna-se novamente ativo. Observa-se na Figura 103.

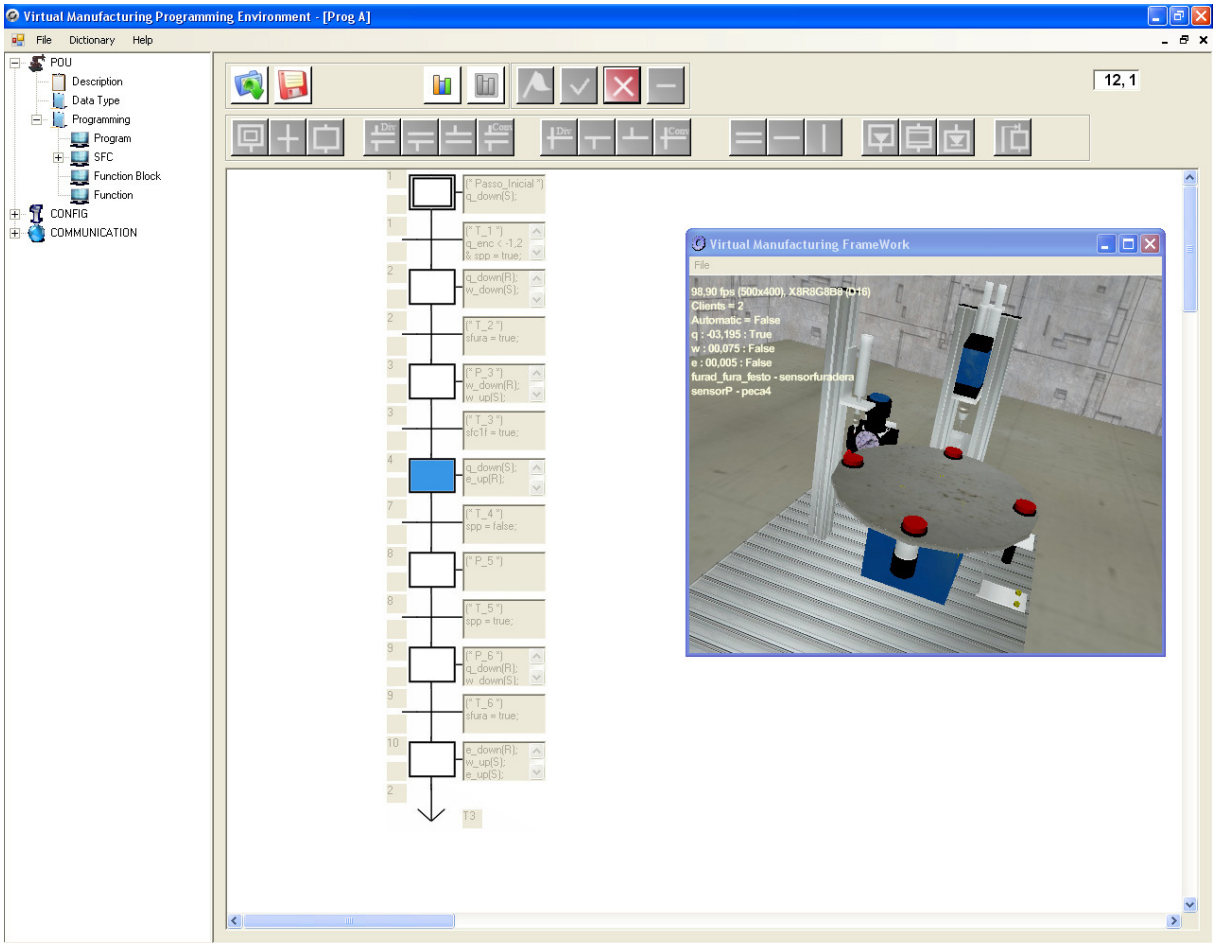


Figura 103 - P_4 novamente ativo

E assim, o sistema funciona de modo que a mesa gire, e todas as peças sejam furadas e testadas de maneira correta.

Em todos os exemplos explicitados neste anexo, os resultados foram satisfatórios e precisos.

APÊNDICE C – DIAGRAMA DE CLASSES CONCEITUAL

Abaixo será mostrado o diagrama de classes conceitual do programa, que foi desenvolvido usando os conceitos de orientação a objeto. O diagrama de classes é composto basicamente de classes que se relacionam (Figura 104). Modela os aspectos estáticos do sistema.

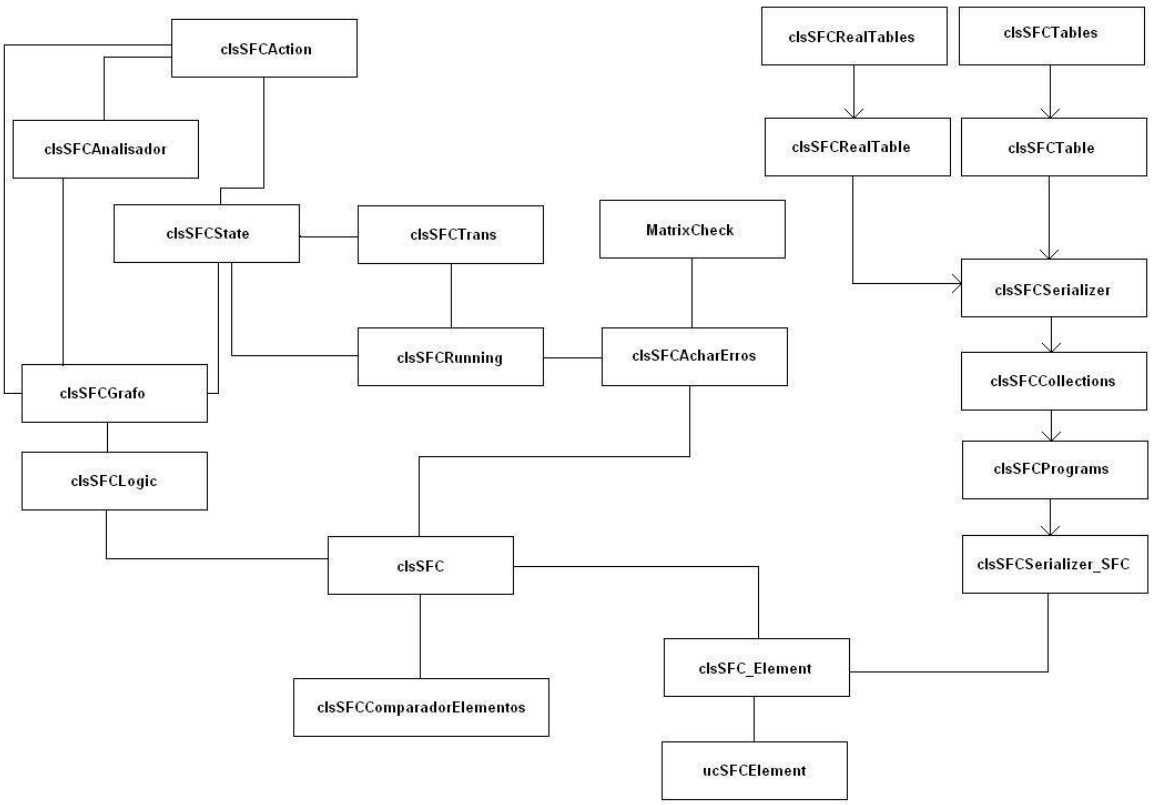


Figura 104 - Diagrama de classes conceitual

Basicamente, há relações de associação e dependência entre as classes presentes no programa.

APÊNDICE D – PASSADO, PRESENTE E FUTURO DO APIMV

Neste apêndice, serão apresentados três diagramas. O primeiro, da Figura 105, mostra a situação passada do APIMV, ou seja, como este ambiente se encontrava antes do desenvolvimento deste trabalho de mestrado. A Figura 106 mostra como o sistema ficou após a conclusão deste trabalho. E finalmente na Figura 107 observa-se como o APIMV estará num futuro próximo.

Figura 105 - APIMV antes do desenvolvimento deste trabalho

Figura 106 - Situação atual do APIMV, após a conclusão deste trabalho

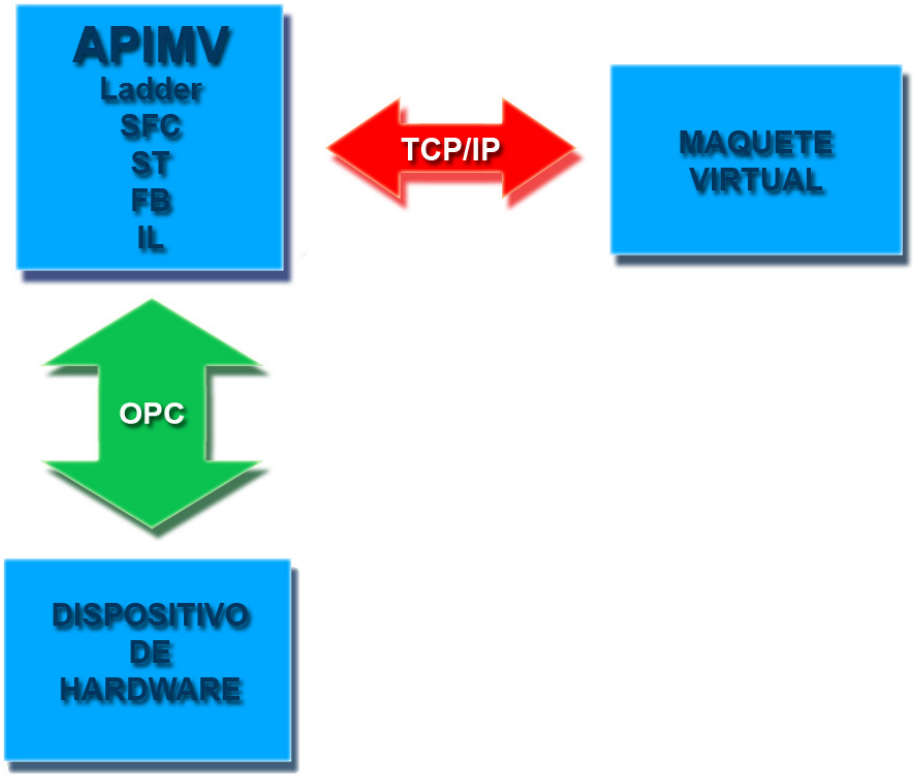


Figura 107 - Futuro do APIMV