Ferramenta de Especificação Gráfica de Máquinas de Estados Finitas para o Ambiente de Teste baseado em Injeção de Falhas por Software (ATIFS)

Peterson Costa Barbalho de Melo
Matrícula: 200027782
Engenharia de Computação
Universidade Federal do Rio Grande do Norte
Bolsista CNPq/PIBIC

Orientador
Romualdo Alves Pereira Júnior

Natal, maio de 2003

Resumo

Neste trabalho apresenta-se a modelagem e desenvolvimento de uma ferramenta para a especificação gráfica de máquinas de estados finitas e o armazenamento e configuração de seus atributos relacionados, os quais foram definidos com base na Linguagem de Especificação de Protocolos (LEP) do projeto ATIFS, um projeto de Pesquisa e Desenvolvimento do INPE/ETE, do grupo de software, fruto de uma cooperação entre a Divisão de Desenvolvimento de Sistema de Solo - DSS e o Instituto de Computação da UNICAMP, desde 1995.

Esta ferramenta tem como intuito principal a geração de arquivos em LEP contendo a especificação da máquina de estados modelada.

Além da definição e representação da máquina de estados e seus atributos de forma a contemplar todo o escopo da LEP, também desenvolveu-se uma interface amigável e que permite uma maior organização e agilidade na modelagem.

Sumário

1. Objetivos

1.1. Objetivos Gerais

O presente trabalho objetiva fornecer subsídios para uma utilização mais eficiente do Ambiente de Testes baseado em Injeção de Falhas por Software - ATIFS e para isto pretende-se modificar o método de geração de testes de um de seus componentes, a ferramenta CONDADO, pois um dos pontos desse método consiste na transformação da especificação da máquina de estados finita estendida para a LEP o que é feito de forma não automatizada pelo usuário.

1.2. Objetivos Específicos

Com o intuito de automatizar e facilitar a transformação da especificação da máquina de estados finita estendida para LEP, o que corresponde a primeira etapa do método de geração de testes da ferramenta CONDADO, desenvolveu-se uma ferramenta computacional que possibilita ao usuário modelar graficamente e de maneira intuitiva uma máquina de estados finita estendida. Adicionalmente, pode-se configurar os seus atributos e então, a partir dessa modelagem, gerar automaticamente a especificação em LEP a qual servirá como entrada para o "analisador", que é a segunda etapa do método de geração de testes.

2. Introdução

2.1. ATIFS

O ATIFS é um ambiente integrado composto por vários subsistemas e é baseado em injeção de falhas por software. Este ambiente além de dar suporte para as atividades de geração, execução e análise dos resultados durante o processo de teste, também possibilita várias facilidades as quais permitem a comunicação entre seus subsistemas, além de uma interface uniforme para o usuário. O ATIFS provê ainda duas camadas: uma para o gerenciamento de objetos de configuração e a outra para o gerenciamento e armazenamento dos dados.

Os subsistemas que compõem o ATIFS são os seguintes:

2.1.1. Desenvolvimento dos testes

O subsistema de desenvolvimento dos testes compreende as funções de geração de testes e definição das falhas, onde esta ultima tem o intuito de ajudar o usuário na definição das classes de falhas, seus atributos e o método utilizado para injetá-las.

2.1.2. Geração do script

Neste subsistema o usuário descreve informações tais como os casos de teste a serem aplicados, os dados que eles utilizam, as classes de falhas a serem injetadas, o momento da injeção, a duração do experimento, etc; e estas informações são utilizadas para a definição da seqüência de teste a serem realizados.

2.1.3. Controle de execução

Este subsistema controla a execução dos testes e monitora o sistema alvo, onde a ativação e interação com o sistema em teste é feito baseado no script gerado anteriormente e os dados observados durante os testes são coletados.

2.1.4. Tratamento dos resultados

Neste ponto, o subsistema em questão faz a análise do comportamento observado e obtém as estatísticas, ambos baseados nos dados coletados durante os testes.

2.2. CONDADO

A CONDADO é uma ferramenta a qual está inserida no subsistema de desenvolvimento de testes do ATIFS e ela tem por objetivo a geração de testes combinando controle e dados dos parâmetros de interações do protocolo e para isto utiliza três técnicas de caixa preta: teste de transição de estados, teste de sintaxe e teste de domínio.

Para possibilitar o método de geração de teste, uma linguagem para a especificação de protocolos foi descrita e chamada de Linguagem de Especificação de Protocolos (LEP) permitindo então que a CONDADO realize a transformação da especificação do protocolo em uma especificação de teste e em seguida gere os casos de teste utilizando as técnicas de caixa preta citadas anteriormente.

O método de geração de teste consiste no seguinte: primeiramente, definem-se os requisitos do protocolo representados por uma máquina de estados finita estendida expressa em LEP, que serve como entrada para o analisador. O analisador, por sua vez, monta a tabela de símbolos, o código intermediário da especificação e gera a matriz de transição contendo a parte de controle do protocolo. Com base na matriz de transição gerada pelo analisador, o verificador de propriedades analisa a especificação com o objetivo de definir quais as propriedades que esta possui. Tanto o analisador como o verificador de propriedades apenas oferecem serviços a CONDADO a qual baseada na tabela de símbolos, código intermediário e propriedades da especificação utiliza seus dois componentes principais para gerar os casos de teste. Os componentes principais da CONDADO consistem no seguinte:

2.2.1. Conversor

Este componente baseia-se na especificação em LEP, obtida através dos serviços do analisador, e na satisfação dos requisitos da especificação para transformar os dados e transições em uma especificação de teste cujo formato é baseado em Cláusulas de Horn, que são interpretadas em Prolog.

2.2.2. Gerador

De posse das Cláusulas de Horn fornecidas pelo Conversor e das restrições definidas pelo usuário, este componente utiliza um conjunto de técnicas de caixa preta implementadas em Prolog para gerar seqüências de teste as quais englobam controle e dados.

A CONDADO permite ainda testes completos ou seletivos, onde nos testes seletivos o usuário define propósitos de teste como por exemplo quais transições serão testadas ou quantas vezes serão executas. Além disso, o usuário pode utilizar as restrições para auxiliar na geração dos casos de teste onde, não havendo restrição, todos os casos de teste para as técnicas implementadas pela CONDADO serão geradas.

2.3. Linguagem de Especificação de Protocolos (LEP)

Com o objetivo de se obter uma notação única que pudesse representar tanto a parte de controle quanto a parte de dados de um protocolo de comunicação representado na forma de uma máquina de estado finita estendida foi desenvolvida a Linguagem para Especificação de Protocolos (LEP). A LEP é baseada na linguagem utilizada pela ferramenta TAG no que diz respeito à especificação da parte de controle, já para a parte de dados tomou-se como base a notação ASN.1 a qual é utilizada para descrição de estrutura de dados das interações dos protocolos.

A LEP pode ser dividida em cinco partes principais, as quais são descritas a seguir:

2.3.1. Seções principais da LEP

2.3.1.1. Definição das variáveis

Esta seção é iniciada pela palavra reservada VARIABLES e em seguida são colocados os nomes das variáveis e seus tipos os quais pode ser os seguintes: INTEGER, REAL, OCTETSTRING, BITSTRING e ENNUMERATED. Por exemplo, a definição da variável v1 do tipo real corresponde ao seguinte:

```
VARIABLES:
    v1 : real;
```

2.3.1.2. Definição dos Estados

Nesta seção é feita a declaração de todos os estados presentes na máquina de estados finita estendida que representa o protocolo de comunicação. Ela é iniciada pela palavra reservada STATES e em seguida são colocados os nomes dos estados onde o estado inicial tem o seu nome precedido pelo símbolo #. Por exemplo, para representar dois estados da máquina, sendo *idle* o estado inicial, usa-se a seguinte notação:

```
STATES:
    #idle;
    waitconec;
```

2.3.1.3. Definição das interações

Uma interação pode ser classificada com entrada e/ou saída, porém, em LEP, as entradas e as saídas são especificadas em seções separadas onde as entradas são precedidas pela palavra reservada INPUTS, enquanto as saídas, pela palavra reservada OUTPUTS. Caso uma interação seja classificada como entrada e saída simultaneamente ela deve ser discriminada nas duas seções. Como exemplo de especificação de interações pode-se fornecer o seguinte:

```
INPUTS:
    SENDrequest;
```

DISrequest;

OUTPUTS:
CR;
DISrequest;

2.3.1.4. Definição dos dados

A LEP possibilita ainda a representação de dados estruturados através dos seguintes tipos estruturados: SEQUENCE, SET e CHOICE, onde SEQUENCE é utilizado para o agrupamento de campos dos mais variados tipos e a ordem da colocação desses campos é considerada, já o SET também é utilizado para agrupar campos de tipos diferentes porém a ordem desses campos não interessa e CHOICE permite a escolha de um campo dentre um conjunto de campos definidos os quais podem ser dos mais variados tipos. Para SEQUENCE e SET existe ainda a possibilidade de se definir um seqüência de valores semelhante a um vetor e isto é feito colocando-se o termo OF seguido do tamanho do vetor entre colchetes após o tipo estruturado, ou seja, SEQUENCE OF [10] ou SET OF [10]. Os campos que compõem esses tipos estruturados podem assumir os mesmos tipos que as variáveis da LEP, isto é, INTEGER, REAL, OCTETSTRING, BITSTRING e ENNUMERATED.

A declaração dos tipos estruturados inicia-se com a palavra reservada DATA. Em seguida são colocados nesta ordem, o identificador do tipo estruturado, o símbolo "::=", o tipo estruturado, o símbolo "{", o identificador do campo, o tipo do campo, os valores possíveis do campos, onde esses três ultimos blocos se repetem para quantos campos existirem e o bloco da cada campo é separado por vírgula do outro e finalmente encerra-se a declaração desses tipo estruturado com o símbolo "};" e esta declaração pode se repetir para quantos tipos estruturados houverem. Por exemplo, a seguir existe a declaração de dois tipos estruturados:

```
DATA:
    DATArequest ::= SEQUENCE {    SDU          OCTETSTRING    5,
                                  Number       ENNUMERATED    2 | 4 | 8,
                                  Blockbound   INTEGER        3..15 };

    Tpcadastro ::= SET OF [10]  {  nome        OCTETSTRING    20,
                                   endereco    OCTETSTRING    30,
                                   salario     REAL           133..844};
```

2.3.1.5. Definição das transições

Após todas as declarações, a parte dinâmica do protocolo é especificada, ou seja, as transições com estado de origem e de destino; entradas pela camada superior (representada por "U") ou pela camada inferior (representada por "L"); condições de disparo; ações a serem executadas; faltas; e, finalmente, as saídas que pode podem vir da camada superior ou inferior.

A especificação das transições é iniciada pela palavra reservada TRANSITIONS e a partir daí para cada transição os campos são colocados na seguinte ordem: identificador da transição precedido pelo símbolo "*", identificador do estado de origem precedido pelo símbolo ">", identificador das entradas precedido pelo símbolo "?U" se vier da camada superior ou "?L" se vier da camada inferior, as condições entre colchetes, as ações entre chaves, as faltas precedidas pelo símbolo "@", identificador das saídas precedido pelo símbolo "!U" se for para a camada superior ou "!L" se for para a camada inferior e finalmente o identificador do estado de destino precedido pelo símbolo "<" e finalizado com ponto-e-vírgula. Como exemplo da especificação de uma transição podemos citar a seguinte:

```
TRANSITIONS:
        *t1
        >idle
                ?U.SENDrequest
                [ counter <= 10 ] { counter := counter + 1 }
                !U.CR
        <waitconec;
```

2.3.2. Gramática formal para a LEP

2.3.2.1. Palavras reservadas

- VARIABLES
- STATES
- INPUTS
- OUTPUTS
- DATA
- TRANSITIONS
- END

2.3.2.2. Símbolos especiais

- : segue palavra reservada
- // precede comentários
- = atribuição
- ; encerra atribuição
- ::= precede tipo estruturado
- {} delimita ações
- [] delimita condições e tamanho dos tipos estruturados
- ? representa a ação de enviar
- ! representa a ação de receber
- L representa o testador inferior
- U representa o testador superior
- > precede o nome do estado de origem
- < precede o nome do estado de destino

- # precede o estado inicial
- @ precede uma falha
- * precede a identificação da transição

2.3.2.3. Descrição formal da gramática para a LEP

<FSM_espec>	::=	[<variables_defs>]<states_defs><inputs_defs> <outputs_defs>[<data_defs>]<transitions_defs>END‘.’
<variables_defs>	::=	VARIABLES‘:’<variable_def>‘;’{<variable_def>‘;’}
<states_defs>	::=	STATES’:’<state_def>’;’{<state_def>’;’}
<inputs_defs>	::=	INPUTS’:’<input_def>’;’{<input_def>’;’}
<outputs_defs>	::=	OUTPUTS’:’<output_def>’;’{<output_def>’;’}
<data_defs>	::=	DATA’:’<data_def>’;’{<data_def>’;’}
<transitions_defs>	::=	TRANSITIONS’:’<transition_def>’;’{<transition_def>’;’}
<variable_def>	::=	<variable_name>’:’<type>
<state_def>	::=	[‘#’]<state_name>
<input_def>	::=	<input_name>
<output_def>	::=	<output_def>
<data_def>	::=	<input_name>’::=’<structured_type>[<size>]’{’ <data_name><type><data_value> {‘,’<data_name><type><data_value>}‘}’
<transition_def>	::=	‘*’<transition_name>’>’<state_name><transition_cont>’<’ <state_name>
<transition_cont>	::=	<input>[<condition>][<action>][<fault>]{<output>}| [<input>]<condition>[<action>][<fault>]{<output>}
<structured_type>	::=	SEQUENCE | SET | CHOICE | SEQUENCE OF | SET OF
<input>	::=	‘?’(‘U’|’L’)<input_name>
<output>	::=	‘?’(‘U’|’L’)<output_name>
<fault>	::=	‘@’<fault_name>
<condition>	::=	‘[’boolean_exp‘]’
<action>	::=	‘{’program_statements‘}’
<data_value>	::=	<value>[‘..’<value>] | <id>{‘|’<id>}
<type>	::=	INTEGER | REAL | OCTETSTRING | BITSTRING | ENNUMERATED
<size>	::=	‘[’<value>‘]’
<value>	::=	digit{digit}
<id>	::=	(letter | digit) { letter | digit }
<transition_name>	::=	<name>
<variable_name>	::=	<name>
<state_name>	::=	‘a’.. ‘z’{ letter | digit }
<input_name>	::=	<name>
<output_name>	::=	<name>
<data_name>	::=	<name>
<fault_name>	::=	<name>
<name>	::=	letter{ letter | digit }

3. Metodologia

3.1. Busca por trabalhos já desenvolvidos

Inicialmente, antes de se desenvolver uma ferramenta integralmente decidiu-se procurar por trabalhos já realizados e de código aberto que pudessem ser aproveitados, tendo em vista que isso levaria a uma economia de tempo considerável bem como o aproveitamento de vários recursos desenvolvidos, utilizados e supostamente testados restando como trabalho a ser feito apenas as adaptações necessárias para englobar o sistema no escopo da LEP e o desenvolvimento de novas funcionalidades específicas. O direcionamento dessa busca se deu da seguinte forma: primeiramente escolheu-se a internet como principal fonte de procura já que a rede mundial de computadores representa um canal rápido e fácil de acesso a inúmeras universidades e desenvolvedores de todo o mundo. A partir daí, o foco da busca consistiu em uma ferramenta de modelagem e simulação gráfica de máquinas de estados finitas as quais tivessem o seu código fonte livre para alteração, possibilitando então as adaptações desejadas.

Como resultado da pesquisa descrita anteriormente foram encontradas inúmeras ferramentas, porém, apenas algumas delas tinham seu código livre e todas, além de serem extremamente específicas, possuíam recursos de edição gráfica limitados e não muito intuitivos. No entanto, o ponto principal considerado foi a escassa documentação sobre o código fonte, o que seria de extrema importância para o seu entendimento e possibilitaria as modificações necessárias. Contudo, as ferramentas observadas serviram como boa referência na organização da interface, representações gráficas utilizadas, recursos e comportamentos entre outras.

Considerando os resultados obtidos e expostos acima decidiu-se desenvolver integralmente a ferramenta foco deste trabalho, ganhando com isso uma maior flexibilidade quanto à escolha das funcionalidades a serem implementadas, segurança na codificação e conhecimento geral do comportamento e requisitos da mesma.

3.2. Ferramentas computacionais

Para a escolha da linguagem de programação e plataforma utilizada no desenvolvimento da ferramenta levou-se em consideração o conhecimento prévio do desenvolvedor, recursos suportados e portabilidade, logo, escolheu-se o Borland Delphi 5.0® como ambiente de desenvolvimento, já que este é um ambiente de desenvolvimento rápido (RAD) e possui diversos recursos os quais podem ser amplamente utilizados durante a codificação. Já o sistema operacional escolhido foi o Windows XP® tendo em vista sua ampla utilização e conseqüente portabilidade.

3.3. Definição dos requisitos

Como ponto inicial para a definição dos requisitos partiu-se da funcionalidade primordial da ferramenta a ser implementada, ou seja, a geração da especificação em LEP

de uma máquina de estados finita estendida e também do fato que esta especificação teria como fonte uma representação gráfica. Adicionaram-se então recursos básicos de editores gráficos em geral além de outros que auxiliam na organização da especificação, resultando então nos requisitos a seguir:

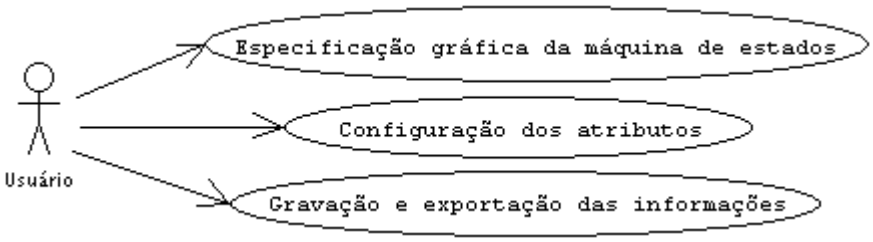
- A ferramenta a ser desenvolvida deve permitir a modelagem gráfica de uma máquina de estados estendida com representações correspondentes a todos os seus elementos sendo eles: estados e seus identificadores, transições e seus identificadores, indicativo de estado inicial e indicativo de estado final;
- Pelo menos as operações de inclusão, movimentação e remoção devem ser possíveis para os elementos da máquina de estados finita estendida;
- Deve ser possível o armazenamento da imagem que representa a especificação gráfica em arquivo de formatos conhecidos (ex: bmp, jpg);
- Recursos padrões de editores gráficos em geral devem ser permitidos como: desfazer, refazer, copiar, colar, recortar e selecionar;
- Recursos para organização da representação gráfica como alinhamento e coloração dos elementos e tratamento de camadas de desenho devem ser implementados;
- A ferramenta deve possibilitar o armazenamento e configuração de todo e qualquer atributo suportado pela LEP referente aos elementos da máquina de estados finita estendida;
- Deve ser possível o armazenamento da especificação corrente em arquivo tanto da representação gráfica quanto dos atributos, bem como seu carregamento permitindo manipulações posteriores;
- Deve ser possível armazenar em arquivo a especificação em LEP da máquina de estados finita estendida especificada;

3.4. Modelagem da ferramenta

Para se efetuar a modelagem dos aspectos fundamentais da ferramenta escolheu-se a Unified Modeling Language (UML), pois trata-se de uma linguagem de modelagem amplamente adotada e universal, independente de projeto, extremamente rica no que diz respeito a representação dos aspectos envolvidos no desenvolvimento de sistemas.

3.4.1. Diagrama de casos de uso

O seguinte diagrama de casos de uso representa o ator Usuário e suas ações possíveis:

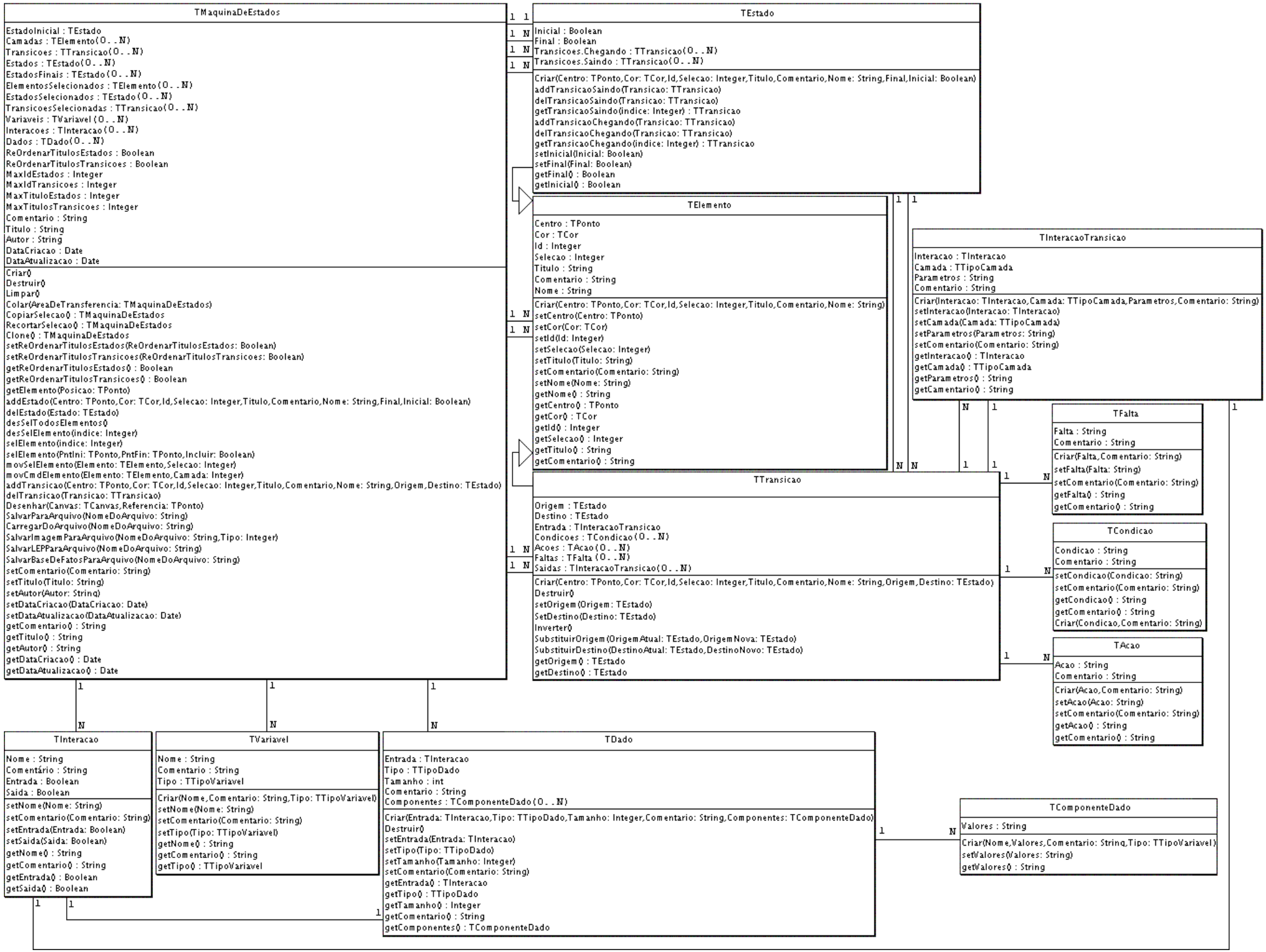


3.4.2. Diagrama de classes

A elaboração do diagrama de classes levou em consideração todos os elementos que podem compor uma máquina de estados finita estendida bem como os atributos necessários para se ter uma representação gráfica deles e todo e qualquer atributo possível de ser representado em LEP para esses elementos. Abaixo segue uma breve descrição de cada classe envolvida no diagrama.

Classe	Descrição
TMáquinaDeEstados	Esta é a classe principal a qual gerencia todos os objetos que compõem a máquina de estados e principalmente as informações gráficas associadas a eles.
TElemento	Considerando que podemos ter estados e transições na máquina de estados e que estes possuem coisas em comum principalmente no que diz respeito as informações gráficas como posicionamento, cor, seleção, entre outra, então criou-se a classe elemento a qual serve de superclasse para as classes que representam os estados e as transições.
TEstado	Consiste numa subclasse de TElemento que contém os atributos e métodos referentes aos estados que compõem a máquina de estados.
TTransicao	Consiste numa subclasse de TElemento que contém os atributos e métodos referentes às transições que compõem a máquina de estados.
TVariavel	Essa classe representa as variáveis associadas a LEP
TInteracao	Consiste na representação das interações utilizadas na LEP
TDado	Representa os dados estruturados que podem ser cadastrados
TComponenteDado	Representa os campos dos dados estruturados cadastrados
TInteracaoTransicao	Consiste nas interações que podem ser associadas a uma transição que compõe a máquina de estados podendo ser entradas ou saídas
TCondicao	Representa a condição a qual pode ser associada com um transição da máquina de estados
TAcao	Representa a ação a qual pode ser associada com um transição da máquina de estados
TFalta	Representa a falta a qual pode ser associada com um transição da máquina de estados

O diagrama de classes é exposto a seguir:


TMaquinaDeEstados
EstadoInicial : TEstado
Camadas : TElemento(0..N)
Transicoes : TTransicao(0..N)
Estados : TEstado(0..N)
EstadosFinais : TEstado(0..N)
ElementosSelecionados : TElemento(0..N)
EstadosSelecionados : TEstado(0..N)
TransicoesSelecionadas : TTransicao(0..N)
Variaveis : TVariavel(0..N)
Interacoes : TInteracao(0..N)
Dados : TDado(0..N)
ReOrdenarTitulosEstados : Boolean
ReOrdenarTitulosTransicoes : Boolean
MaxIdEstados : Integer
MaxIdTransicoes : Integer
MaxTituloEstados : Integer
MaxTitulosTransicoes : Integer
Comentario : String
Titulo : String
Autor : String
DataCriacao : Date
DataAtualizacao : Date
Criar()
Destruir()
Limpar()
Colar(AreaDeTransferencia: TMaquinaDeEstados)
CopiarSelecao() : TMaquinaDeEstados
RecortarSelecao() : TMaquinaDeEstados
Clone() : TMaquinaDeEstados
setReOrdenarTitulosEstados(ReOrdenarTitulosEstados: Boolean)
setReOrdenarTitulosTransicoes(ReOrdenarTitulosTransicoes: Boolean)
getReOrdenarTitulosEstados() : Boolean
getReOrdenarTitulosTransicoes() : Boolean
getElemento(Posicao: TPonto)
addEstado(Centro: TPonto,Cor: TCor,Id,Selecao: Integer,Titulo,Comentario,Nome: String,Final,Inicial: Boolean)
delEstado(Estado: TEstado)
desSelTodosElementos()
desSelElemento(indice: Integer)
selElemento(indice: Integer)
selElemento(PntIni: TPonto,PntFin: TPonto,Incluir: Boolean)
movSelElemento(Elemento: TElemento,Selecao: Integer)
movCmdElemento(Elemento: TElemento,Camada: Integer)
addTransicao(Centro: TPonto,Cor: TCor,Id,Selecao: Integer,Titulo,Comentario,Nome: String,Origem,Destino: TEstado)
delTransicao(Transicao: TTransicao)
Desenhar(Canvas: TCanvas,Referencia: TPonto)
SalvarParaArquivo(NomeDoArquivo: String)
CarregarDoArquivo(NomeDoArquivo: String)
SalvarImagemParaArquivo(NomeDoArquivo: String,Tipo: Integer)
SalvarLEPParaArquivo(NomeDoArquivo: String)
SalvarBaseDeFatosParaArquivo(NomeDoArquivo: String)
setComentario(Comentario: String)
setTitulo(Titulo: String)
setAutor(Autor: String)
setDataCriacao(DataCriacao: Date)
setDataAtualizacao(DataAtualizacao: Date)
getComentario() : String
getTitulo() : String
getAutor() : String
getDataCriacao() : Date
getDataAtualizacao() : Date
TEstado
Inicial : Boolean
Final : Boolean
Transicoes.Chegando : TTransicao(0..N)
Transicoes.Saindo : TTransicao(0..N)
Criar(Centro: TPonto,Cor: TCor,Id,Selecao: Integer,Titulo,Comentario,Nome: String,Final,Inicial: Boolean)
addTransicaoSaindo(Transicao: TTransicao)
delTransicaoSaindo(Transicao: TTransicao)
getTransicaoSaindo(indice: Integer) : TTransicao
addTransicaoChegando(Transicao: TTransicao)
delTransicaoChegando(Transicao: TTransicao)
getTransicaoChegando(indice: Integer) : TTransicao
setInicial(Inicial: Boolean)
setFinal(Final: Boolean)
getFinal() : Boolean
getInicial() : Boolean
TElemento
Centro : TPonto
Cor : TCor
Id : Integer
Selecao : Integer
Titulo : String
Comentario : String
Nome : String
Criar(Centro: TPonto,Cor: TCor,Id,Selecao: Integer,Titulo,Comentario,Nome: String)
setCentro(Centro: TPonto)
setCor(Cor: TCor)
setId(Id: Integer)
setSelecao(Selecao: Integer)
setTitulo(Titulo: String)
setComentario(Comentario: String)
setNome(Nome: String)
getNome() : String
getCentro() : TPonto
getCor() : TCor
getId() : Integer
getSelecao() : Integer
getTitulo() : String
getComentario() : String
TInteracaoTransicao
Interacao : TInteracao
Camada : TTipoCamada
Parametros : String
Comentario : String
Criar(Interacao: TInteracao,Camada: TTipoCamada,Parametros,Comentario: String)
setInteracao(Interacao: TInteracao)
setCamada(Camada: TTipoCamada)
setParametros(Parametros: String)
setComentario(Comentario: String)
getInteracao() : TInteracao
getCamada() : TTipoCamada
getParametros() : String
getComentario() : String
TFalta
Falta : String
Comentario : String
Criar(Falta,Comentario: String)
setFalta(Falta: String)
setComentario(Comentario: String)
getFalta() : String
getComentario() : String
TCondicao
Condicao : String
Comentario : String
setCondicao(Condicao: String)
setComentario(Comentario: String)
getCondicao() : String
getComentario() : String
Criar(Condicao,Comentario: String)
TAcao
Acao : String
Comentario : String
Criar(Acao,Comentario: String)
setAcao(Acao: String)
setComentario(Comentario: String)
getAcao() : String
getComentario() : String
TTransicao
Origem : TEstado
Destino : TEstado
Entrada : TInteracaoTransicao
Condicoes : TCondicao(0..N)
Acoes : TAcao(0..N)
Faltas : TFalta(0..N)
Saidas : TInteracaoTransicao(0..N)
Criar(Centro: TPonto,Cor: TCor,Id,Selecao: Integer,Titulo,Comentario,Nome: String,Origem,Destino: TEstado)
Destruir()
setOrigem(Origem: TEstado)
SetDestino(Destino: TEstado)
Inverter()
SubstituirOrigem(OrigemAtual: TEstado,OrigemNova: TEstado)
SubstituirDestino(DestinoAtual: TEstado,DestinoNovo: TEstado)
getOrigem() : TEstado
getDestino() : TEstado
TInteracao
Nome : String
Comentário : String
Entrada : Boolean
Saida : Boolean
setNome(Nome: String)
setComentario(Comentario: String)
setEntrada(Entrada: Boolean)
setSaida(Saida: Boolean)
getNome() : String
getComentario() : String
getEntrada() : Boolean
getSaida() : Boolean
TVariavel
Nome : String
Comentario : String
Tipo : TTipoVariavel
Criar(Nome,Comentario: String,Tipo: TTipoVariavel)
setNome(Nome: String)
setComentario(Comentario: String)
setTipo(Tipo: TTipoVariavel)
getNome() : String
getComentario() : String
getTipo() : TTipoVariavel
TDado
Entrada : TInteracao
Tipo : TTipoDado
Tamanho : int
Comentario : String
Componentes : TComponenteDado(0..N)
Criar(Entrada: TInteracao,Tipo: TTipoDado,Tamanho: Integer,Comentario: String,Componentes: TComponenteDado)
Destruir()
setEntrada(Entrada: TInteracao)
setTipo(Tipo: TTipoDado)
setTamanho(Tamanho: Integer)
setComentario(Comentario: String)
getEntrada() : TInteracao
getTipo() : TTipoDado
getTamanho() : Integer
getComentario() : String
getComponentes() : TComponenteDado
TComponenteDado
Valores : String
Criar(Nome,Valores,Comentario: String,Tipo: TTipoVariavel)
setValores(Valores: String)
getValores() : String

4. Implementação

Modelados os aspectos fundamentais da ferramenta partiu-se para a fase de implementação a qual pode ser dividida em duas grandes etapas, onde a primeira consistiu no desenvolvimento das funcionalidades gráficas e a segunda no desenvolvimento das funcionalidades relacionadas aos dados, isto é, ao armazenamento e configuração dos atributos dos elementos da máquina de estados.

4.1. Definições Iniciais

Antes de tudo nomeou-se a ferramenta de "Modelador de Máquina de Estados" ou simplesmente "MME" e é desta forma como ela será referenciada de agora em diante. Além do nome também definiu-se o padrão da interface com o usuário como sendo Simple Document Interface (SDI) e poderia ser dividida em quatro blocos distintos descritos abaixo:

1. Menu: consiste num menu padrão no qual devem ser colocadas todas as funcionalidades desenvolvidas exceto as específicas de alguns componentes;

2. Barra de Ferramentas: consiste numa área a qual contém apenas botões com o intuito de propiciar um acesso rápido às funcionalidades cujo uso é mais freqüente principalmente no que diz respeito à edição gráfica;

3. Área de Desenho: é a área de disponível para a colocação e manipulação de elementos gráficos os quais modela visualmente a máquina de estados;

4. Área dos Atributos: região reservada para a exibição dos atributos do elemento atualmente selecionado na representação gráfica ou mesmo os atributos gerais da máquina de estados.

A figura 1 a seguir ilustra a estrutura descrita anteriormente:

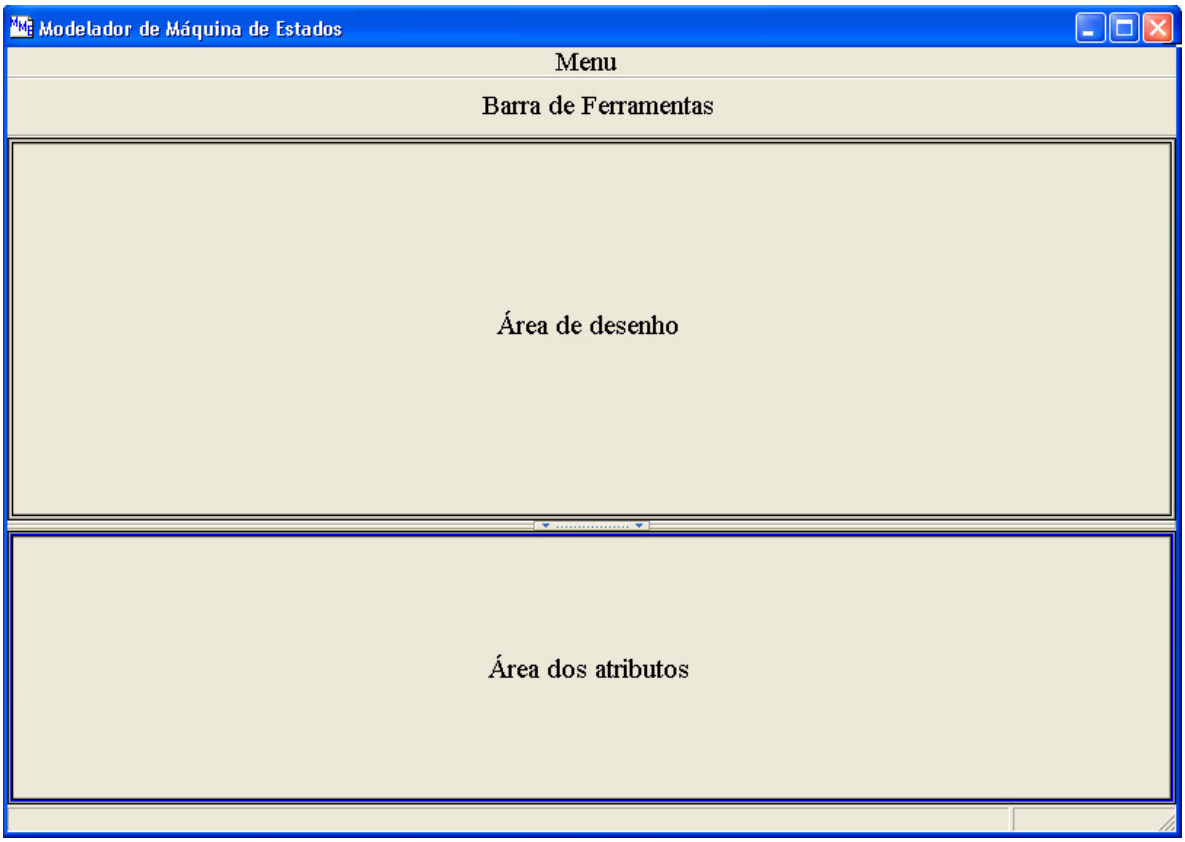


figura 1 – divisão de blocos da interface do MME

Tendo em vista também que o desenvolvimento se deu de forma incremental fez-se necessário a definição de uma estrutura de versões para o acompanhamento do desenvolvimento da MME por parte dos utilizadores e a estrutura escolhida segue o seguinte formato: MME v. 9.9.9 é composta por três números separados por pontos onde o primeiro número indica quantas versões completas foram desenvolvidas, o segundo indica quantas modificações ocorreram no que diz respeitos a parte de atributos da máquina de estados e o terceiro indica quantas modificações ocorreram na parte de edição gráfica da máquina de estados.

Visando informar a versão corrente do MME de maneira apresentável para o usuário decidiu-se criar uma tela de apresentação inicial a qual aparece antes do início da execução do MME que pode ser visualizada durante alguns segundos e contém não só a versão da MME, mas também as informações básicas do projeto.

A figura 2 a seguir ilustra a tela de apresentação criada.

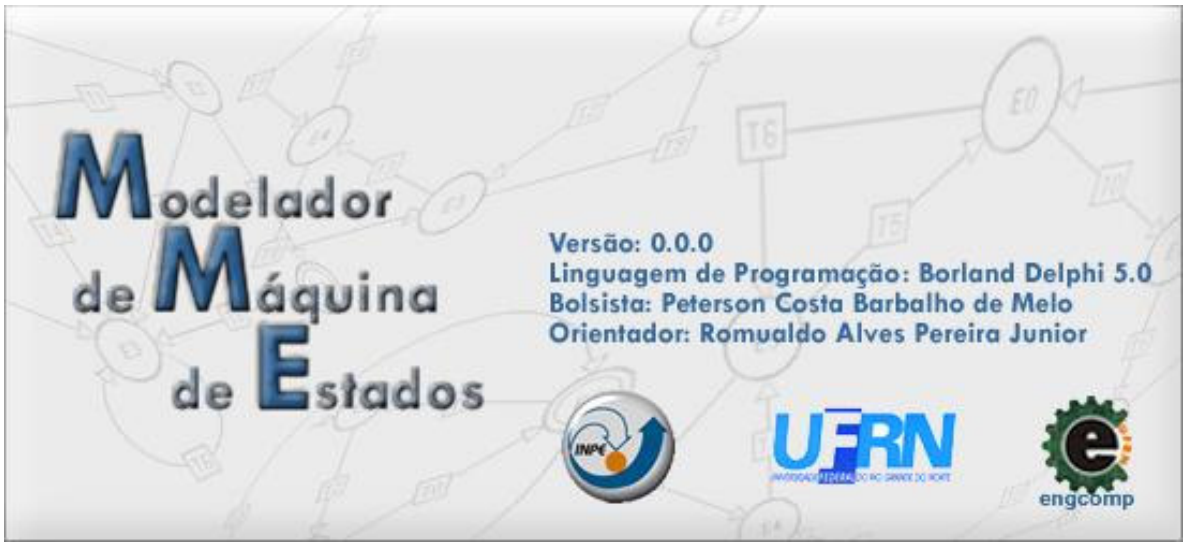


figura 2 – Tela de apresentação do MME

4.2. Desenvolvimento das funcionalidades de edição gráfica

Como dito anteriormente, na primeira etapa de desenvolvimento do MME, o foco do trabalho se voltou para o desenvolvimento de todo o ferramental referente à edição gráfica e recursos associados.

Primeiramente, definiu-se a estrutura de dados que iria armazenar as informações referentes à máquina de estados e seus elementos onde esta definição foi baseada no diagrama de classes sendo escolhido então a orientação a objetos como paradigma. Em seguida foram levantadas as funcionalidades referentes à edição gráfica da máquina de estados para que estas fossem implementadas, as quais são descritas nos próximos tópicos.

4.2.1. Inserção dos elementos que compõem a máquina de estados

A funcionalidade principal desta etapa consistiu na adição de elementos que compõem a máquina de estados, isto é, estados e transições, dentro da área de desenho, porém a forma como estes elementos são inseridos é diferente tendo em vista que a natureza deles é diferente.

Para a inclusão de estados definiu-se que se poderia habilitar ou desabilitar um modo de inserção e esta operação deveria ser feita tanto na barra de ferramentas quanto no menu. Quando ela estivesse habilitada, a cada click do mouse em um ponto da área de desenho, um novo estado seria colocado lá, desde que este ponto não fizesse parte de outro estado, pois caso isto acontecesse o modo de inserção seria desabilitado e este estado ficaria selecionado. Deve-se ressaltar ainda que o primeiro estado inserido sempre começa como estado inicial. Para ilustrar esta operação seguem as figuras 3 e 4.

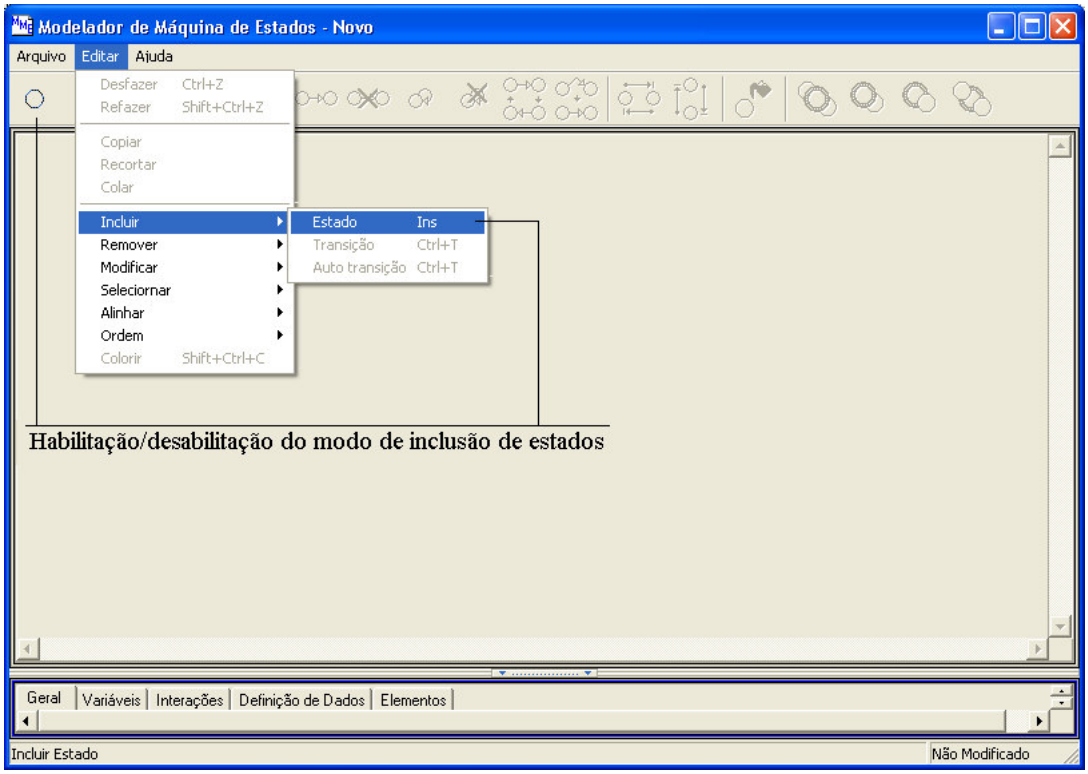


Figura 3 – habilitação / desabilitação do modo de inclusão

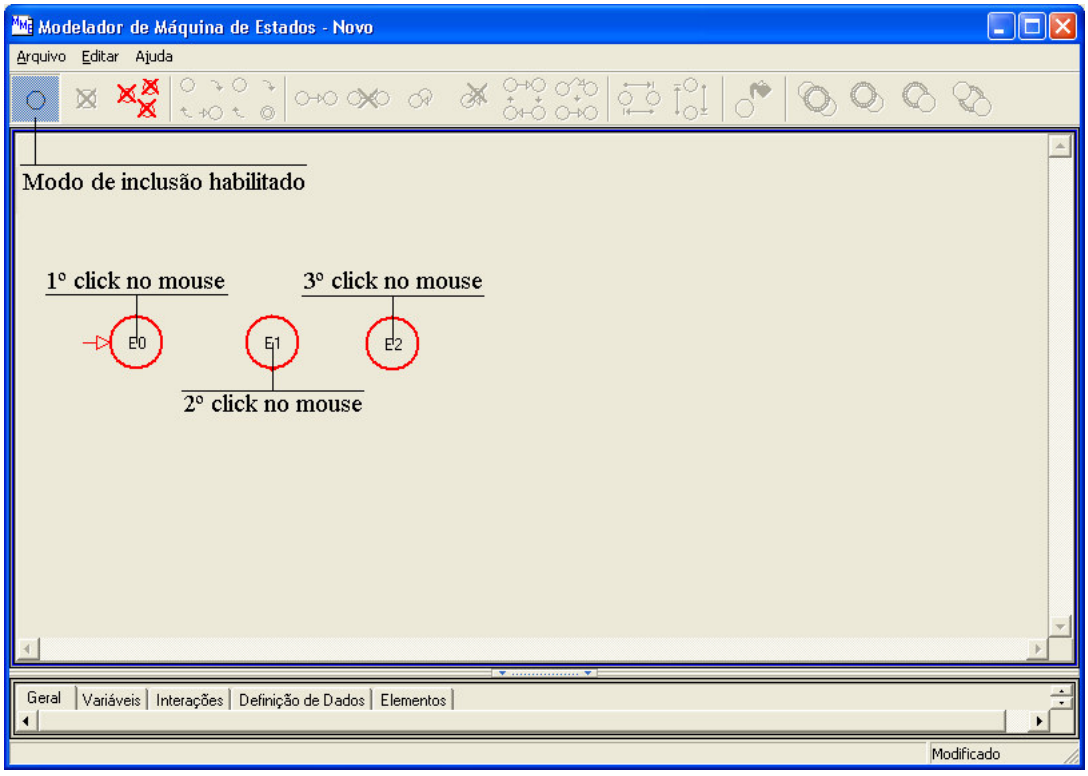


Figura 4 – inclusão de estados

Já a inclusão de transições possui um procedimento diferente tendo em vista que necessita-se da escolha em um estado de origem e um estado de destino, podendo ser ainda uma auto-transição onde o estado de origem é o mesmo do estado de destino. Portanto, para ficar mais claro para o usuário da ferramenta decidiu-se separar essas duas operações como indica a figura 5 abaixo:

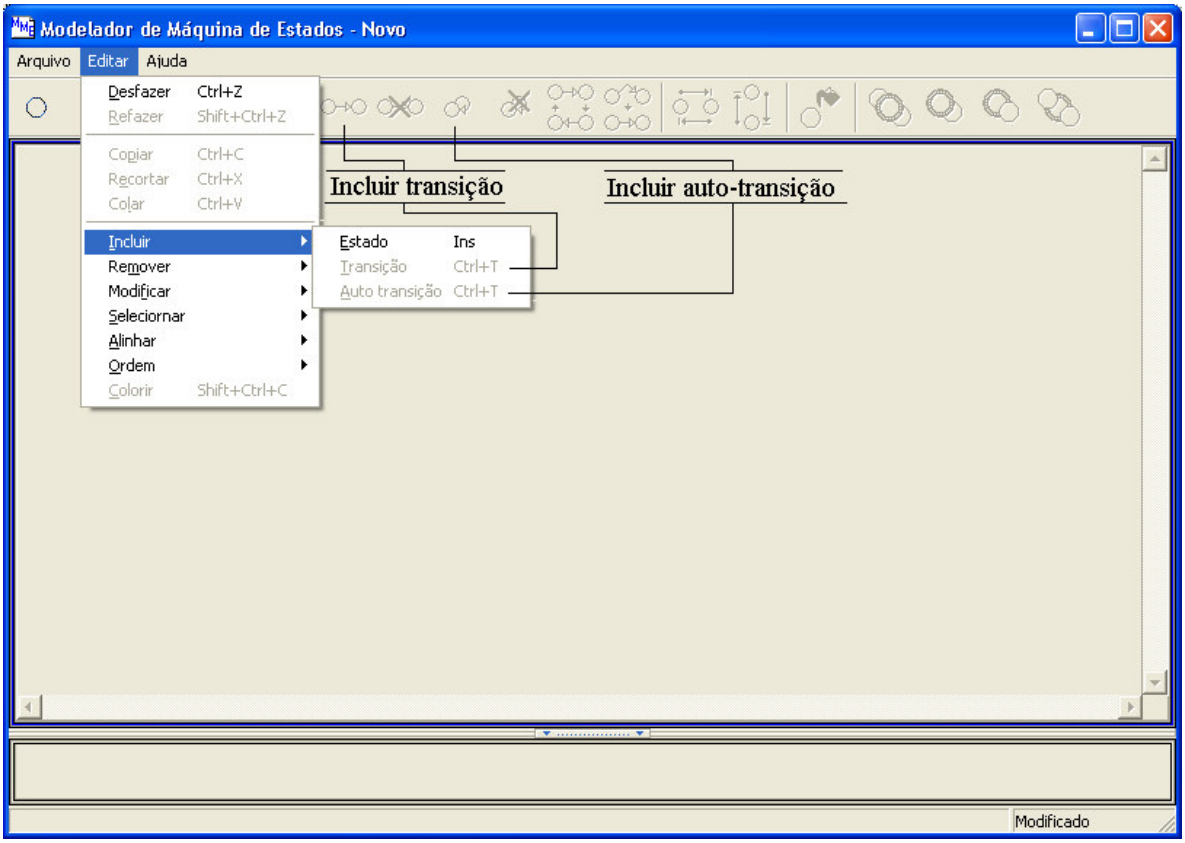


Figura 5 – inclusão de transições

Como os tipos das transições são diferentes, o processo para inseri-las também é distinto. Para inserir-se uma auto-transição basta selecionar um estado e pressionar o botão ou clicar no menu (local indicado na figura 5), que a auto-transição será inserida. No outro caso, primeiramente seleciona-se o estado de origem e em seguida seleciona-se o estado de destino. Assim, ao pressionar o botão ou clicar no menu (local indicado na figura 5) a transição será inserida (vide figura 6).

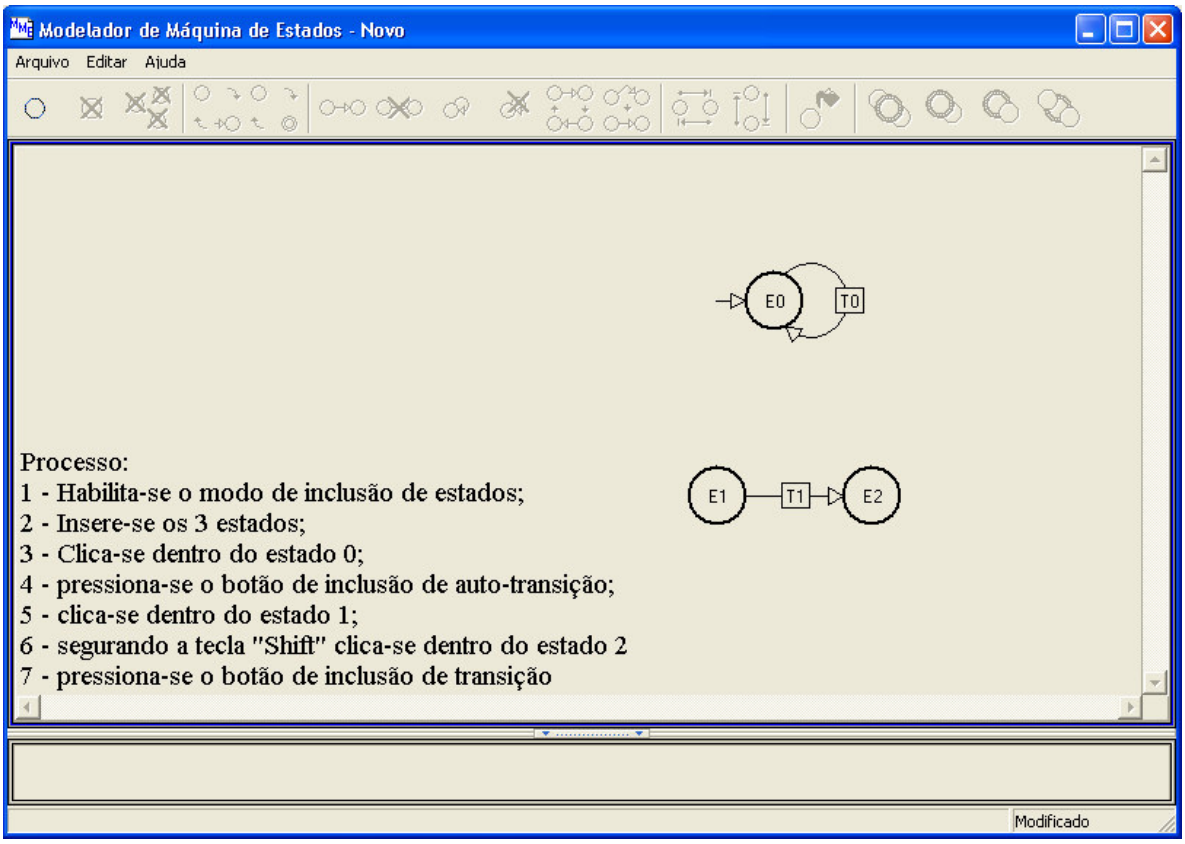


Figura 6 – processo de inclusão de transições

4.2.2. Seleção dos elementos que compõem a máquina de estados

Outra funcionalidade essencial e que é utilizada previamente em várias outras é a seleção dos elementos que compõem a máquina de estados. Definiu-se que esta seleção seria ordenada, isto é, dependendo da ordem em que um elemento fosse selecionado ele teria um grau de seleção maior ou menor e o objeto como maior grau de seleção é chamado de “elemento de referência” sendo distinguido dos demais pelo fato de sua seleção ser indicada por azul claro e os outros por azul escuro. Pode-se ainda modificar o elemento de referência clicando-se com o mouse sobre um objeto selecionado que se deseja tornar o novo elemento de referência.

A seleção pode ser por elemento ou por área; incremental ou não. A seleção por elemento consiste no clique dentro do elemento, já a seleção por área consiste na definição de uma região onde qualquer elemento que tiver suas coordenadas de centro dentro desta região ficará selecionado e a definição desta região se dá com o pressionamento do botão do mouse num ponto livre da área de desenho arrastando-se o mouse para outro ponto da área de desenho e liberando-se o botão do mouse. Assim, a região selecionada fica entre esses dois pontos. Na seleção não incremental, a cada nova seleção os objetos selecionados anteriormente não ficam mais selecionados, já na seleção incremental, a qual é ativada pelo pressionamento da tecla “Shift”, as seleções antigas permanecem quando as novas ocorrem.

Como ilustração do processo de seleção tem-se a figura 7 a seguir:

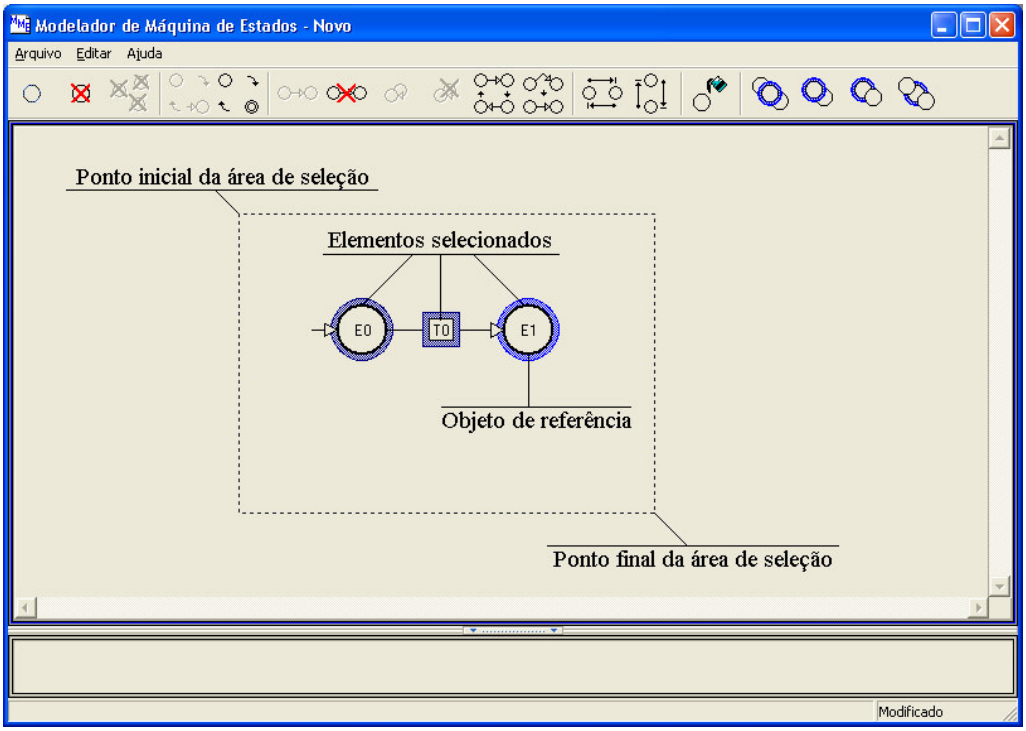


Figura 7 – fundamentos da seleção de elementos

Como recursos auxiliares foram introduzidos no menu a possibilidade de seleção de todos os elementos em geral, todos os estados ou todas as transições bem como a opção de desfazer a seleção corrente. O local de disparo destas opções está exposto na figura 8 abaixo:

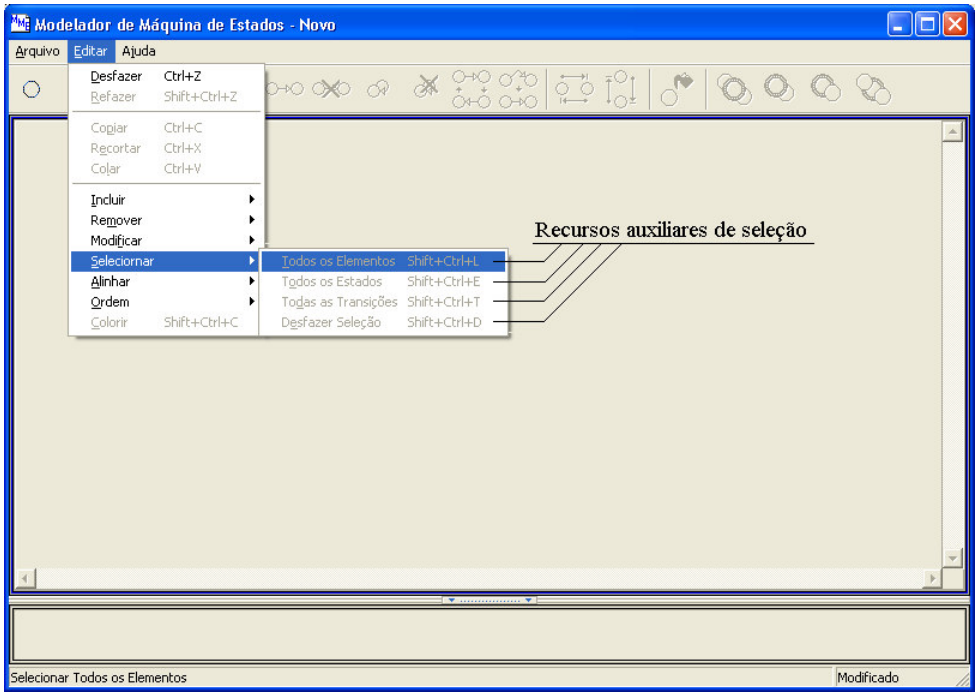


figura 8 – recursos auxiliares de seleção

4.2.3. Exclusão dos elementos que compõem a máquina de estados

Este recurso, por se bastante utilizado, está presente tanto no menu como na barra de ferramentas. O seu processo é bastante intuitivo e pode ocorrer de duas formas: a primeira delas consiste apenas na seleção dos elementos e pressionamento da tecla "DEL" sendo então excluídos todos os elementos selecionados. A segunda forma consiste na utilização dos itens da barra de ferramentas ou do menu, os quais são exclusivos para cada tipo de elemento, ou seja, existe um botão e item do menu para excluir cada tipo de elemento dentre os selecionados. Além destes recursos padrões foi introduzido mais um que consiste na exclusão de todos os estados isolados, ou seja, que não possui nenhuma transição chegando ou saindo dele. Deve-se ressaltar que quando se exclui um estado, todas as transições que tem ele como origem ou destino também são excluídas. Para se indicar a localização destes recursos seguem as figura 9 e 10:

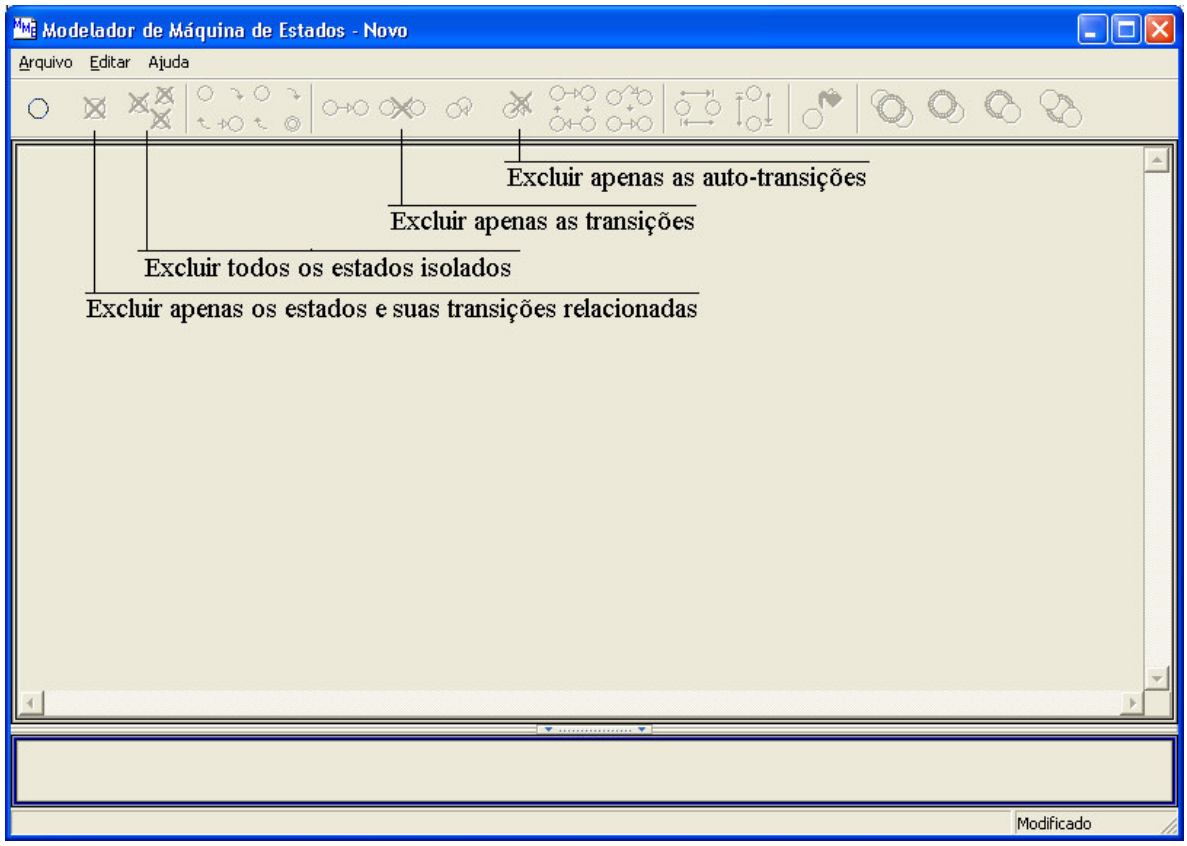


Figura 9 – botões da barra de ferramentas para exclusão dos elementos

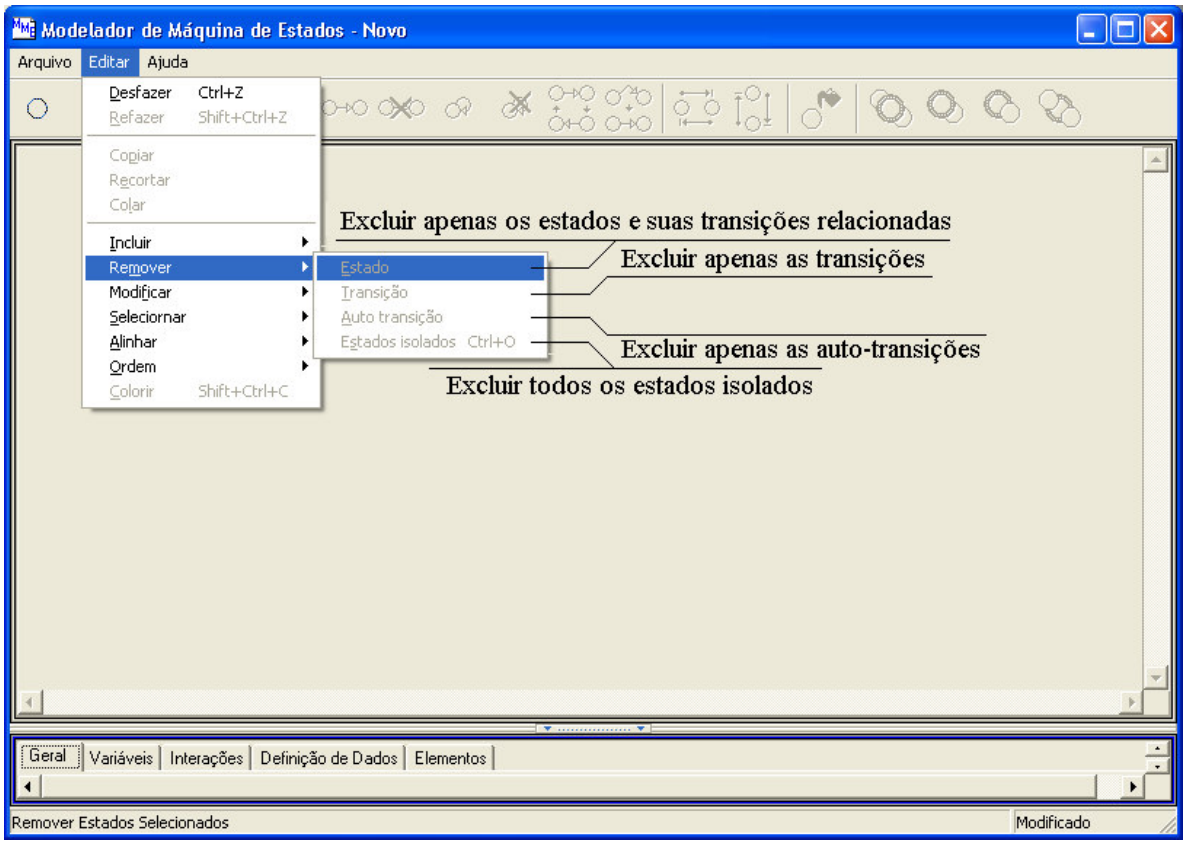


Figura 10 – itens de menu para exclusão dos elementos

4.2.4. Indicação dos estados iniciais e finais

Os estados que compõem a máquina de estados podem ser iniciais e finais e esses atributos devem possuir uma indicação gráfica além de poderem ser alterados a qualquer instante. Logo, fez-se necessário a definição da representação gráfica desses atributos bem como de locais onde eles pudessem ser alterados.

A representação gráfica adotada consiste na representação padrão, ou seja, o estado inicial tem uma seta apontando para ele e os estados finais têm uma borda com dupla circunferência. Tendo em vista que essas funcionalidades são bastante utilizadas no decorrer da modelagem da máquina de estados elas foram colocadas na barra de ferramentas e no menu.

As figuras 11 e 12 a seguir mostram a representação gráfica adotada e indicam os locais onde os atributos podem ser alterados.

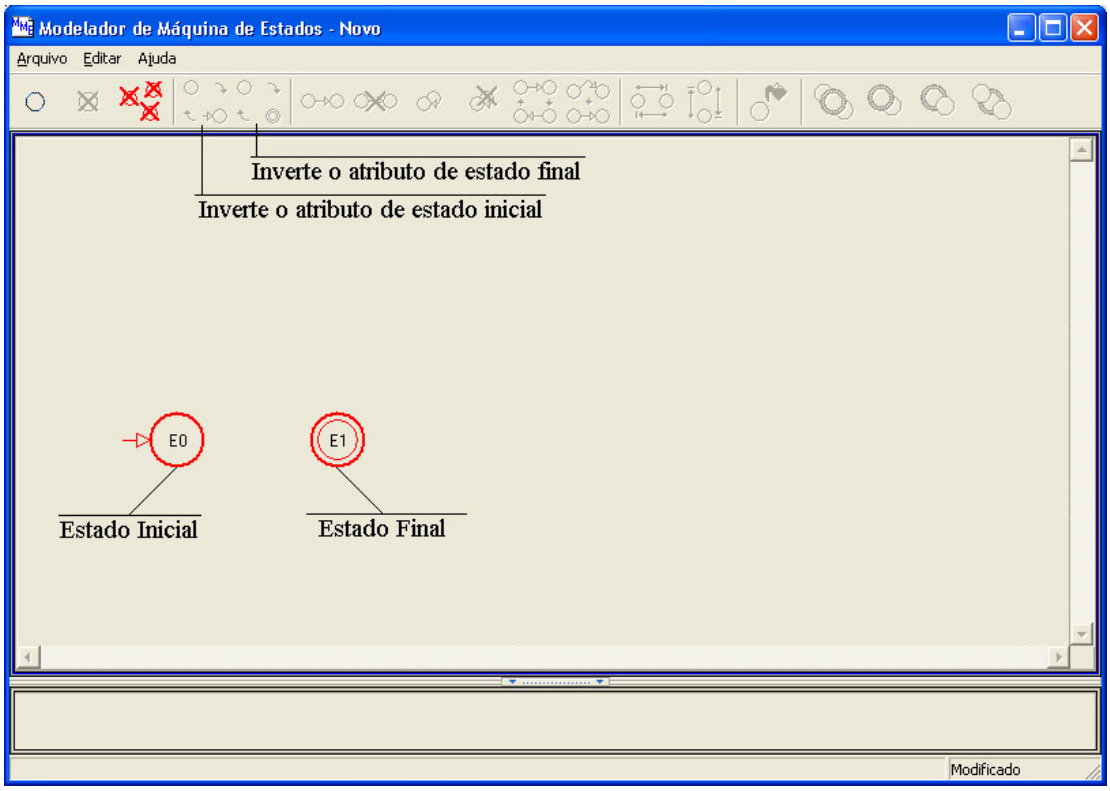


Figura 11 – representação gráfica do estado iniciais e finais

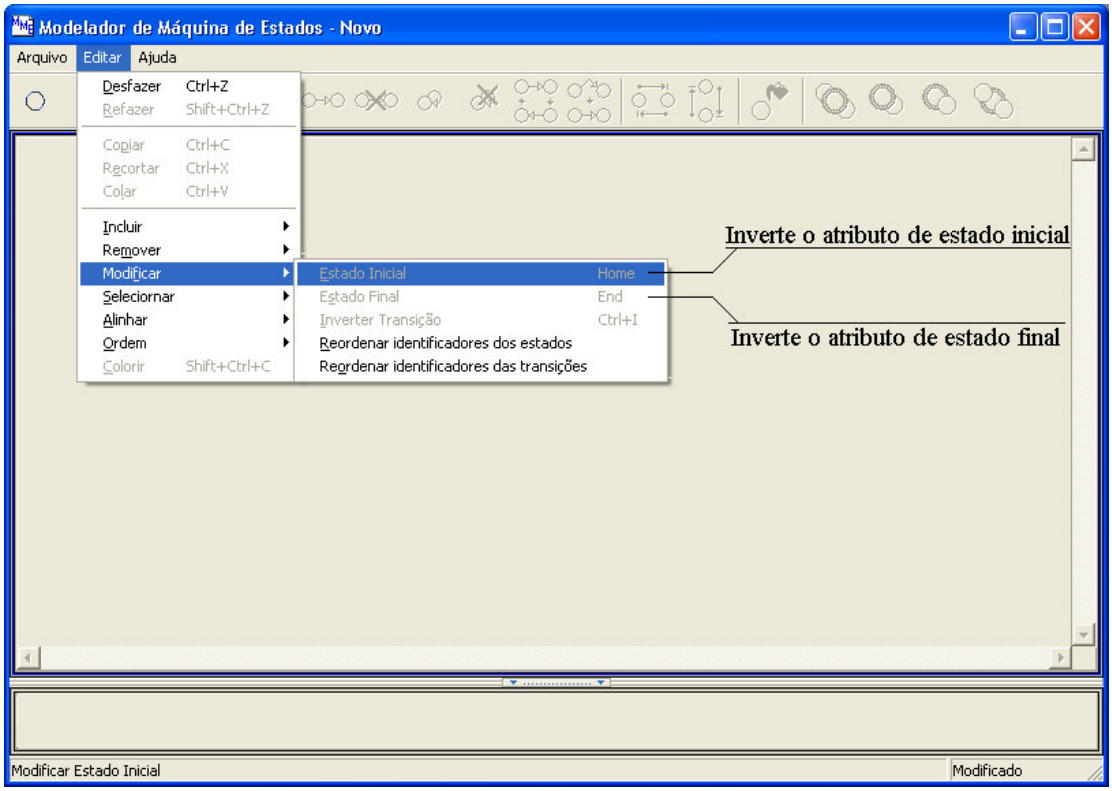


Figura 12 – Itens do menu para inversão do atributos de inicial e final dos estados

4.2.5. Funcionalidades Organizacionais

Visando propiciar ao usuário funcionalidades que auxiliem na organização da especificação da máquina de estados foram introduzidos recursos de uso de cores nos elementos, alinhamento tanto de elementos quanto de seleção e tratamento de camadas de desenho. Essas funcionalidades são descritas com maiores detalhes a seguir.

4.2.5.1. Uso de cores nos elementos

Permite-se ao usuário atribuir cores aos elementos que compõem a máquina de estados. Pode-se utilizar essa estrutura de classificação por critérios pessoais. Esta funcionalidade está presente tanto na barra de ferramentas quanto no menu e seu processo consiste na seleção dos elementos que se deseja colorir, pressionando-se o botão da barra de ferramentas ou o item do menu e finalmente escolhendo-se a cor desejada na paleta de cores, ficando então todos os objetos selecionados coloridos com a cor escolhida. Para ilustrar o processo segue a figura 13.

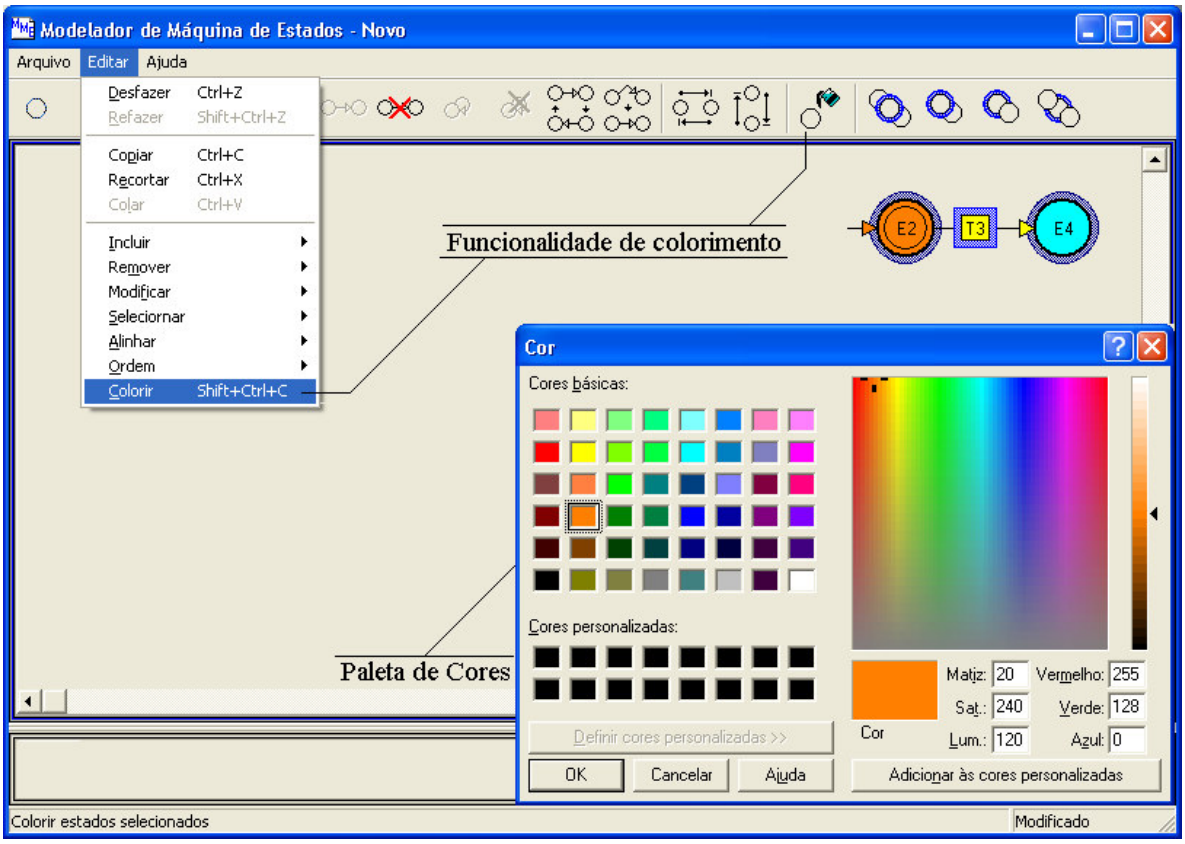


Figura 13 – Processo de colorimento dos elementos selecionados

4.2.5.2. Alinhamento de elementos

Permite-se ao usuário também alinhar os elementos selecionados em relação ao elemento de referência e este alinhamento pode ser horizontal e vertical. Para se alinhar os elementos, primeiramente eles são selecionados e em seguida escolhe-se o elemento de referência. Daí, pressiona-se no botão da barra de ferramentas ou no menu. Caso o alinhamento seja vertical, os centros de todos os elementos selecionados terão suas ordenadas iguais a do elemento de referência. Já, se o alinhamento for horizontal, os centros de todos os elementos selecionados terão suas abscissas iguais a do elemento de referência. Para exemplificar segue as figuras 14 e 15.

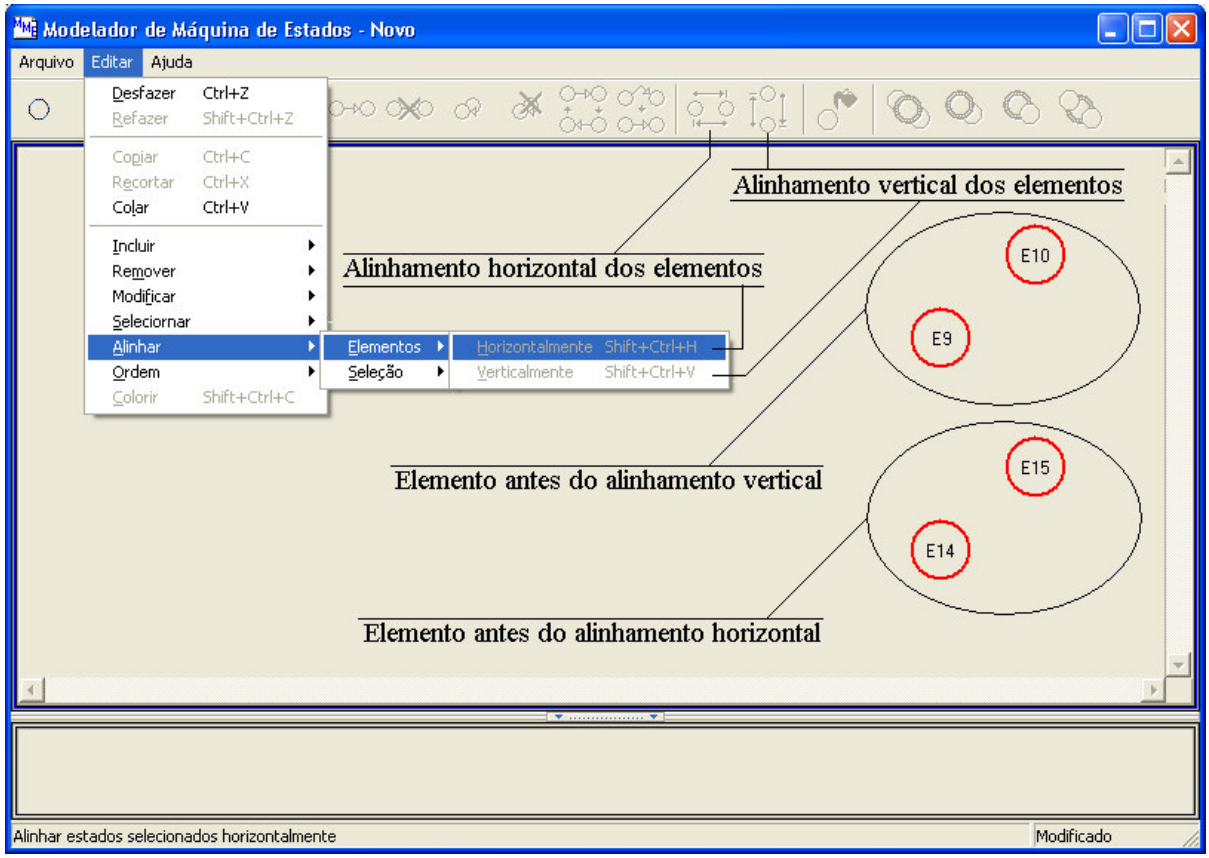


Figura 14 – Localização das funcionalidades de alinhamento de elementos

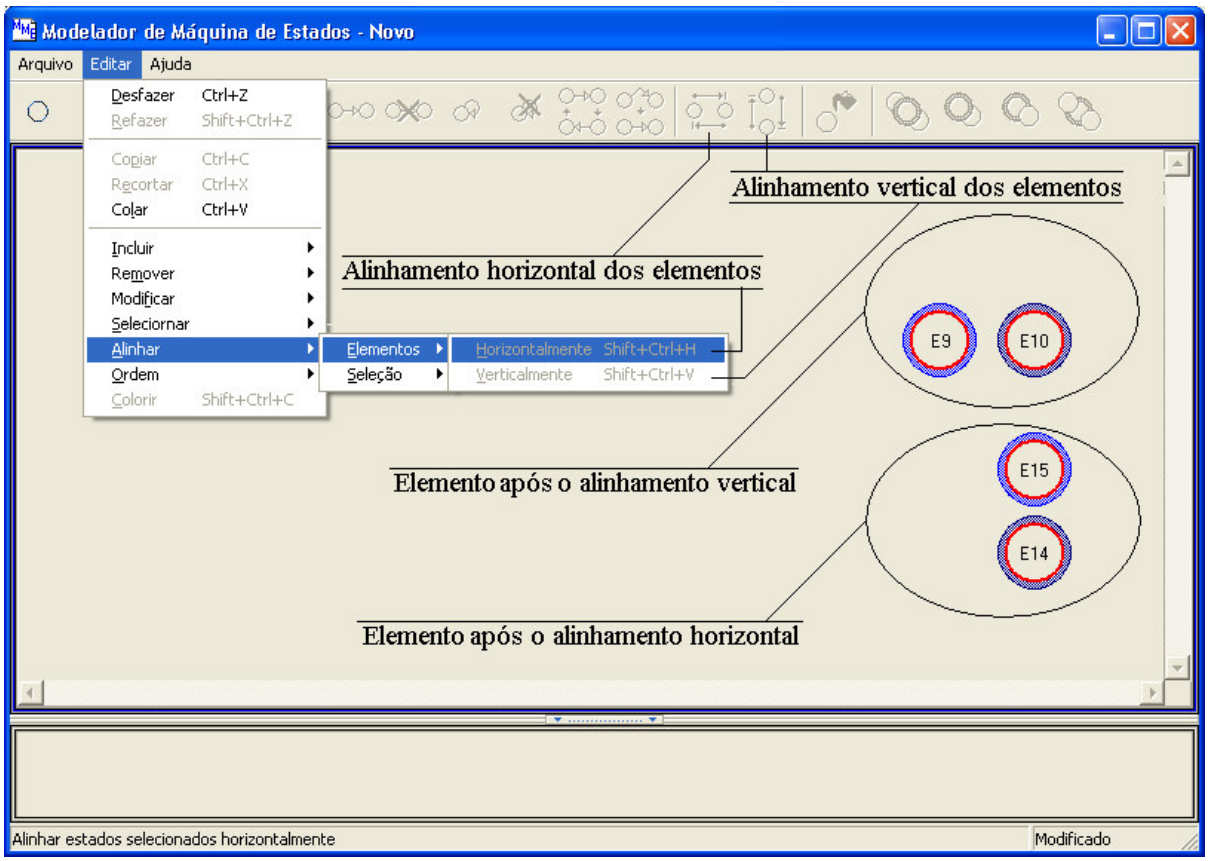


Figura 15 – Localização das funcionalidades de alinhamento de elementos

4.2.5.3. Alinhamento de seleção

O alinhamento de seleção consiste no posicionamento em um determinado local da área de desenho de um certo conjunto de elementos selecionados. O locais possíveis para este tipo de alinhamento são esquerda, horizontalmente no centro, direita, superior, verticalmente no centro e inferior.

A localização desta funcionalidade é ilustrada pela figura 16 a seguir.

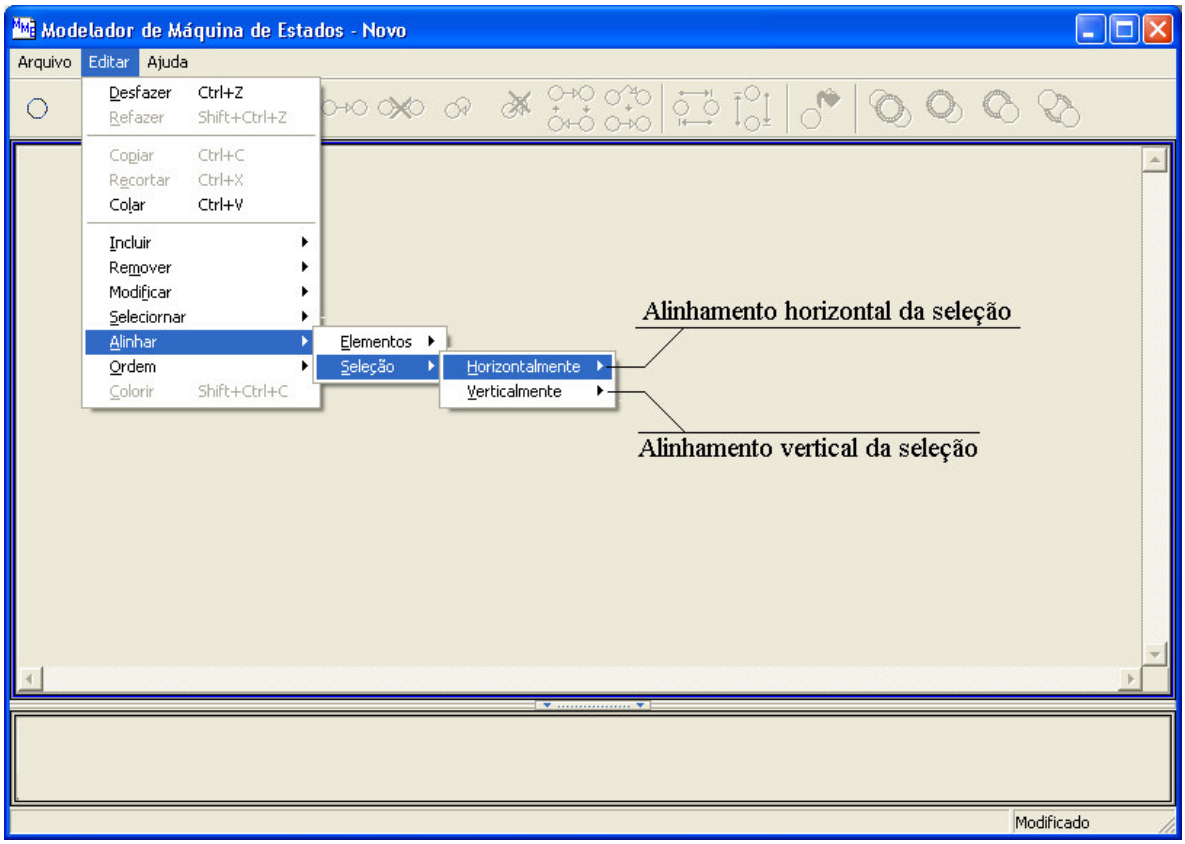


Figura 16 – Localização das funcionalidades de alinhamento de seleção

4.2.5.4. Tratamento de camadas de desenho

Este recurso define a ordem que a MME vai utilizar para desenhar os elementos que compõem a máquina de estados e conseqüentemente que elemento vai ficar acima ou abaixo dos demais, onde quanto mais na frente for a camada do elemento mais acima ele fica. A figura 17 a seguir indica a localização de onde se pode alterar a camada dos elementos selecionados.

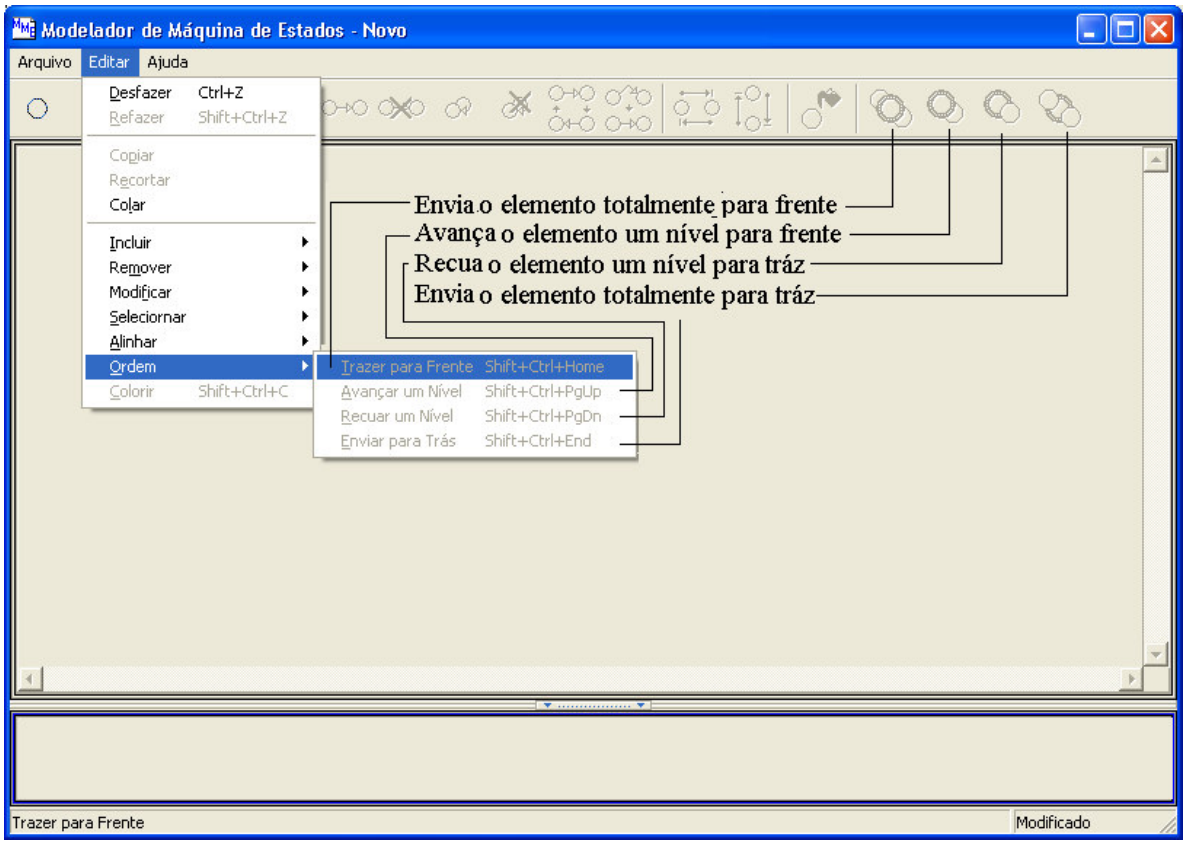


Figura 17 – Localização das funcionalidades de tratamento de camadas

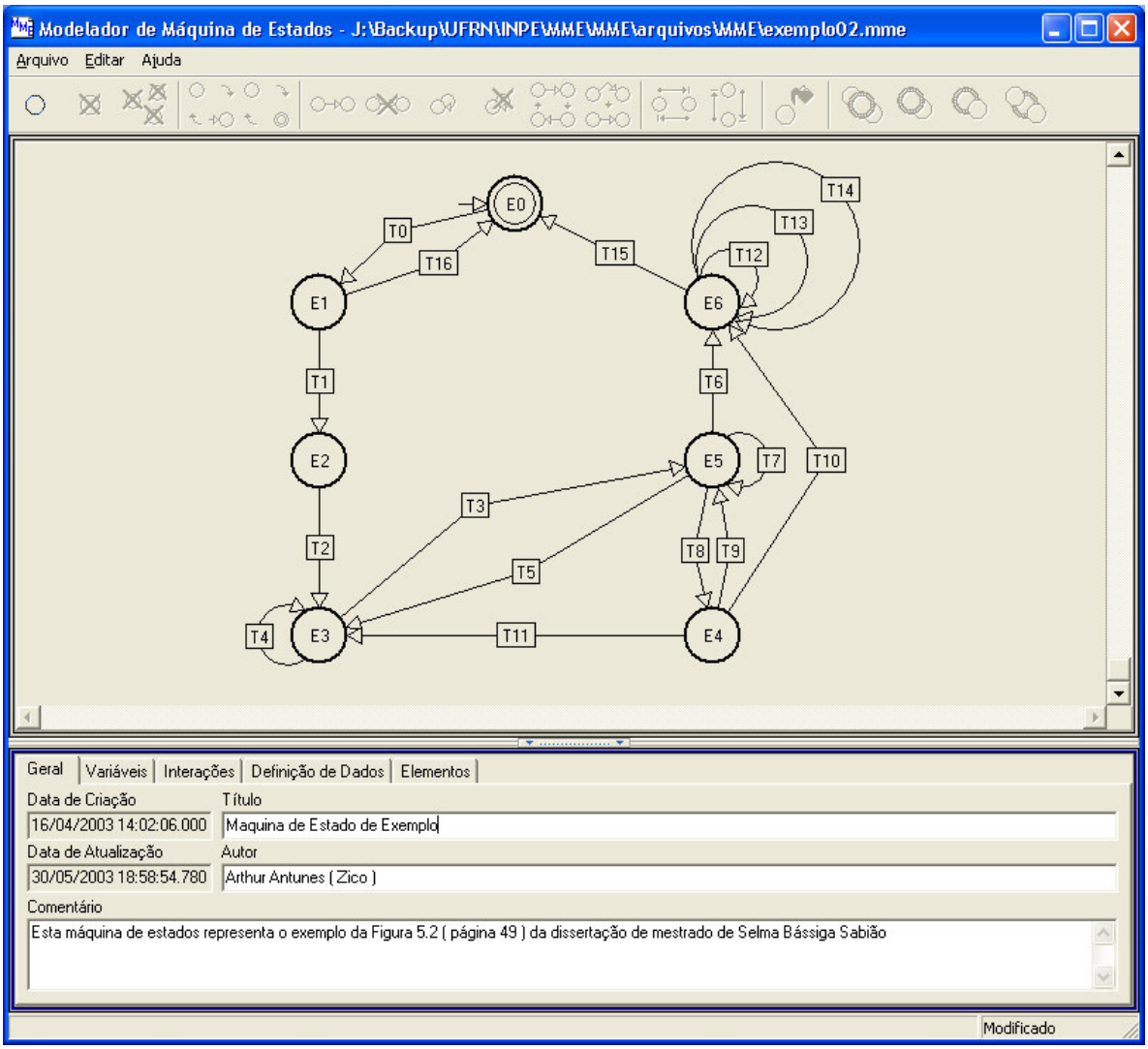
4.3. Desenvolvimento do tratamento dos atributos

Desenvolvidas as funcionalidades referentes à modelagem gráfica partiu-se para o tratamento dos atributos e isto inclui armazenamento e configuração dos mesmos. Primeiramente, definiu-se que os atributos a serem exibidos na área reservada a eles seriam os do elemento selecionado no momento, onde se não tivesse nenhum elemento selecionado ou mais de um, então seriam exibidos os atributos gerais da máquina de estados.

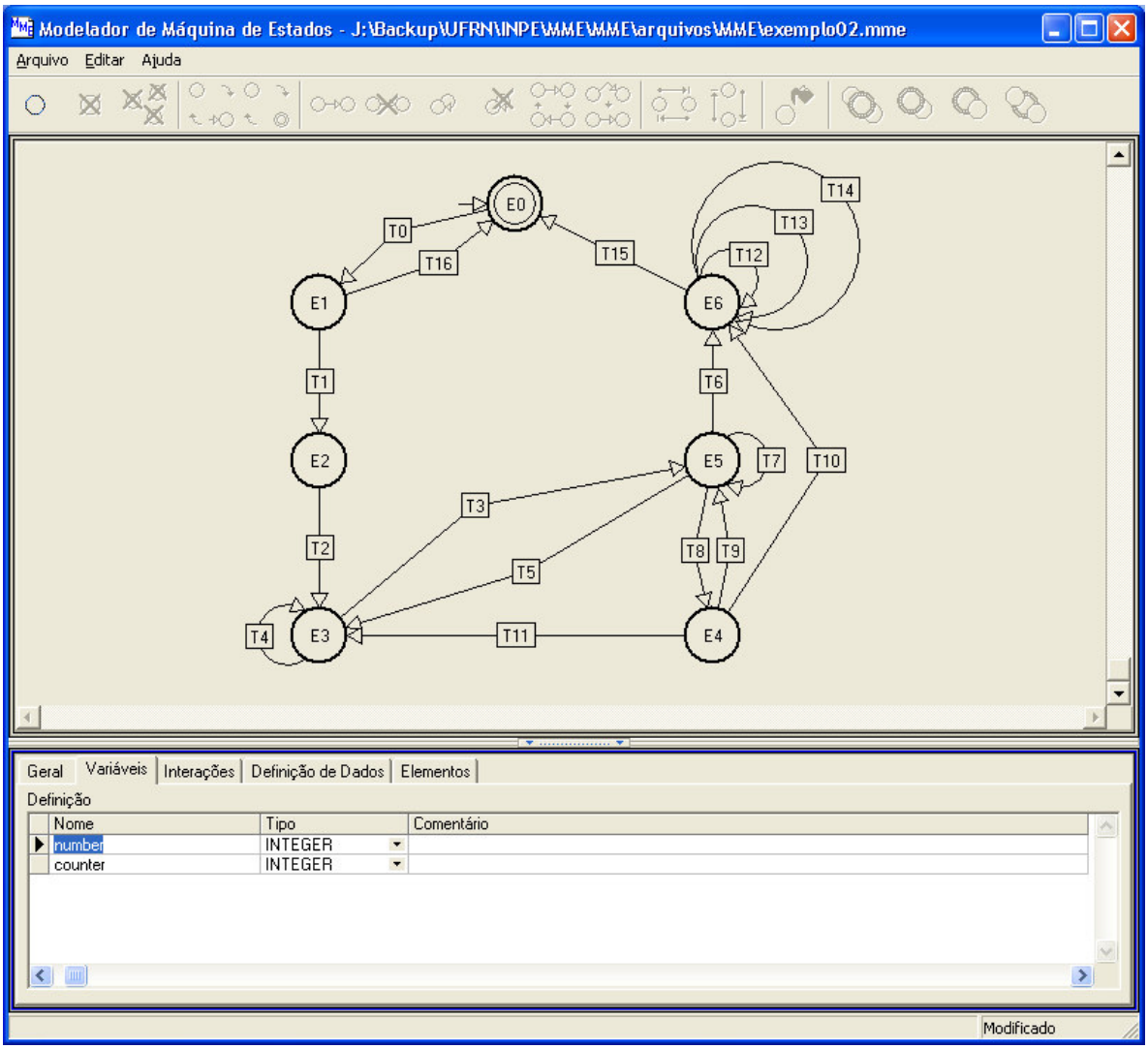
Tendo em vista que se podem dividir os atributos em três categorias: atributos da máquina de estados, atributos do estado e atributos da transição, então fez-se necessária a definição de três formatos diferentes de exibição, sendo um para cada categoria desta, onde o usuário pudesse também alterar os seus valores.

4.3.1. Interfaces ligadas aos atributos da máquina de estados

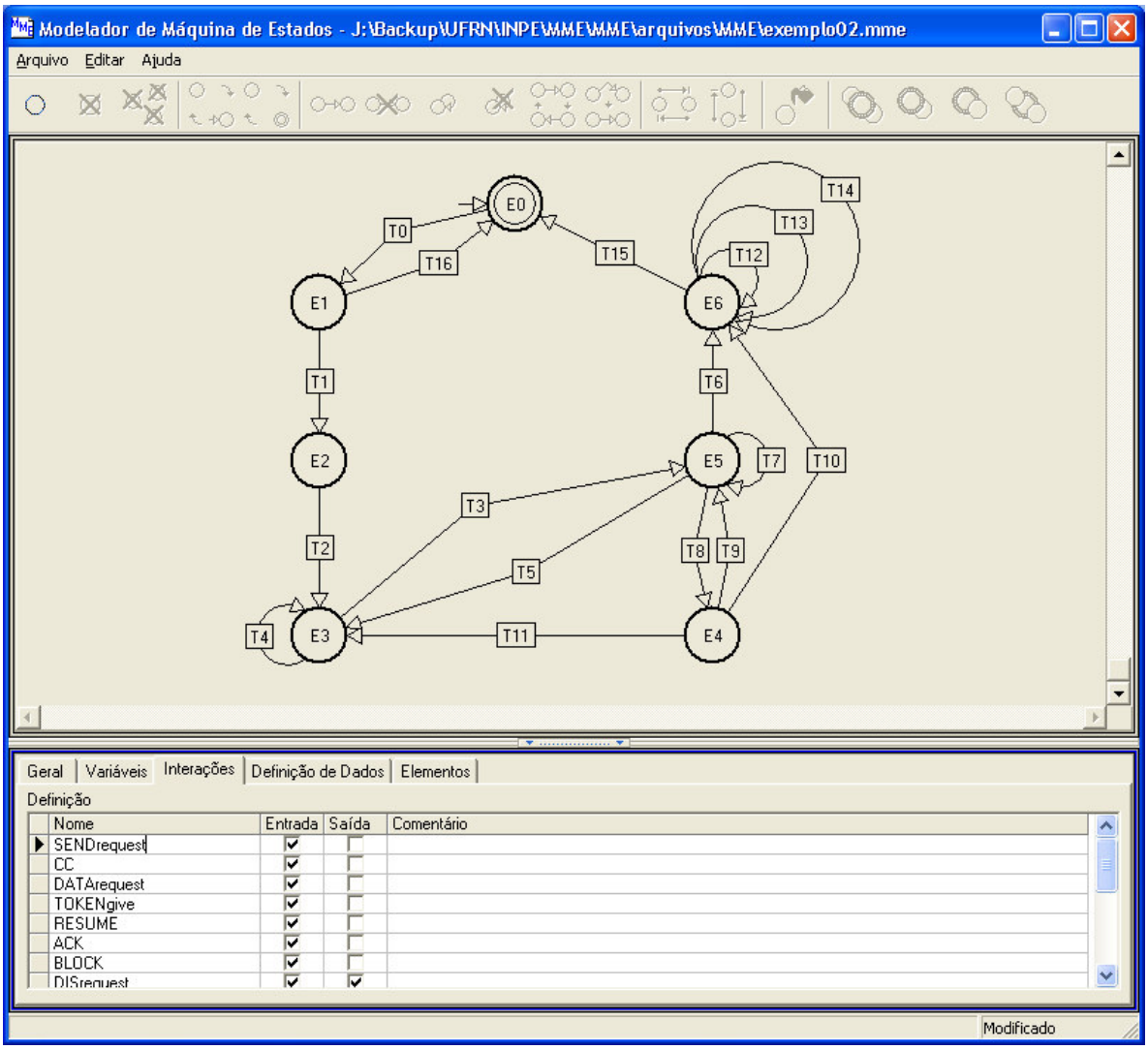
4.3.1.1. Atributos gerais da máquina de estados



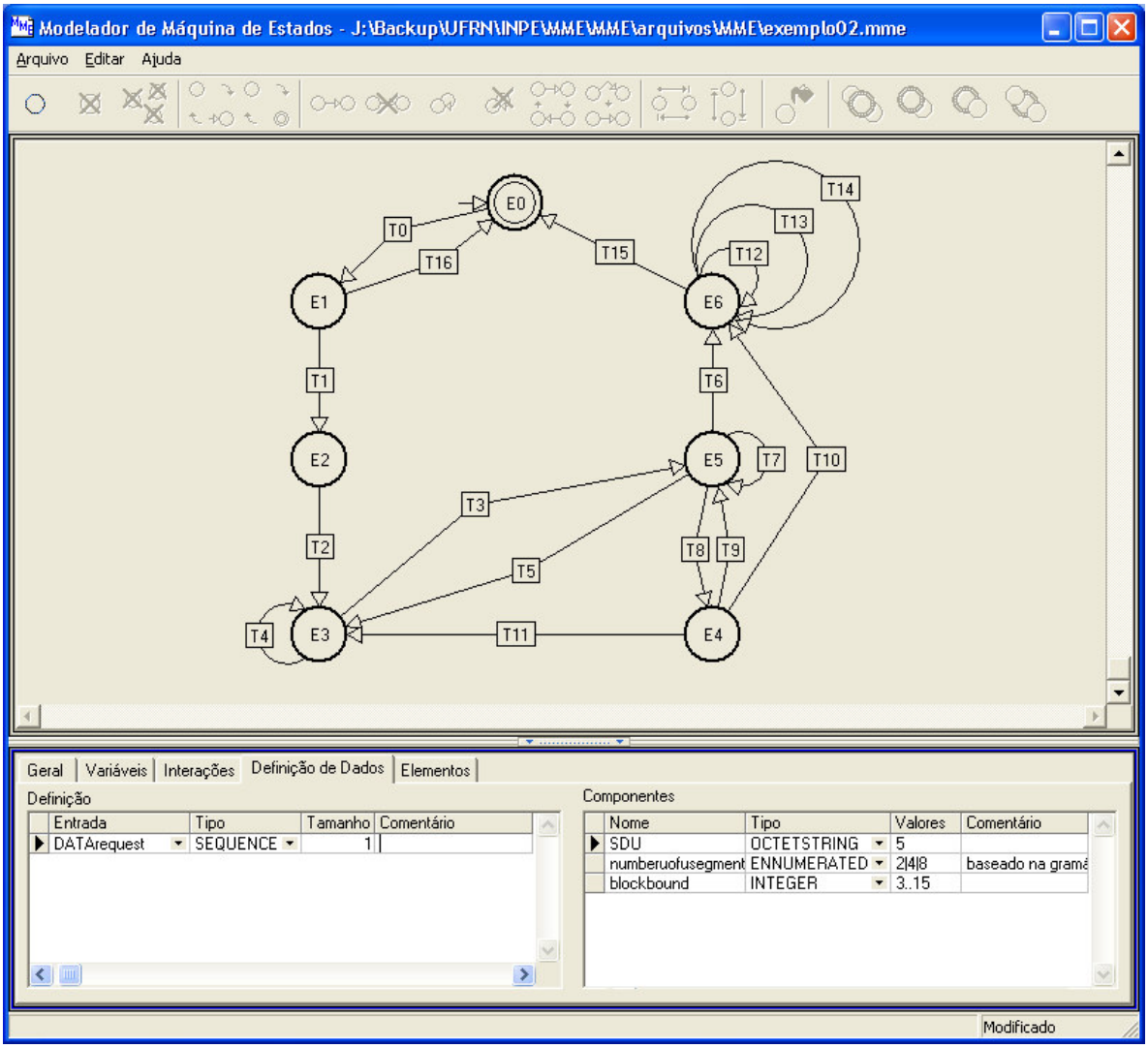
4.3.1.2. Atributos das variáveis da máquina de estados



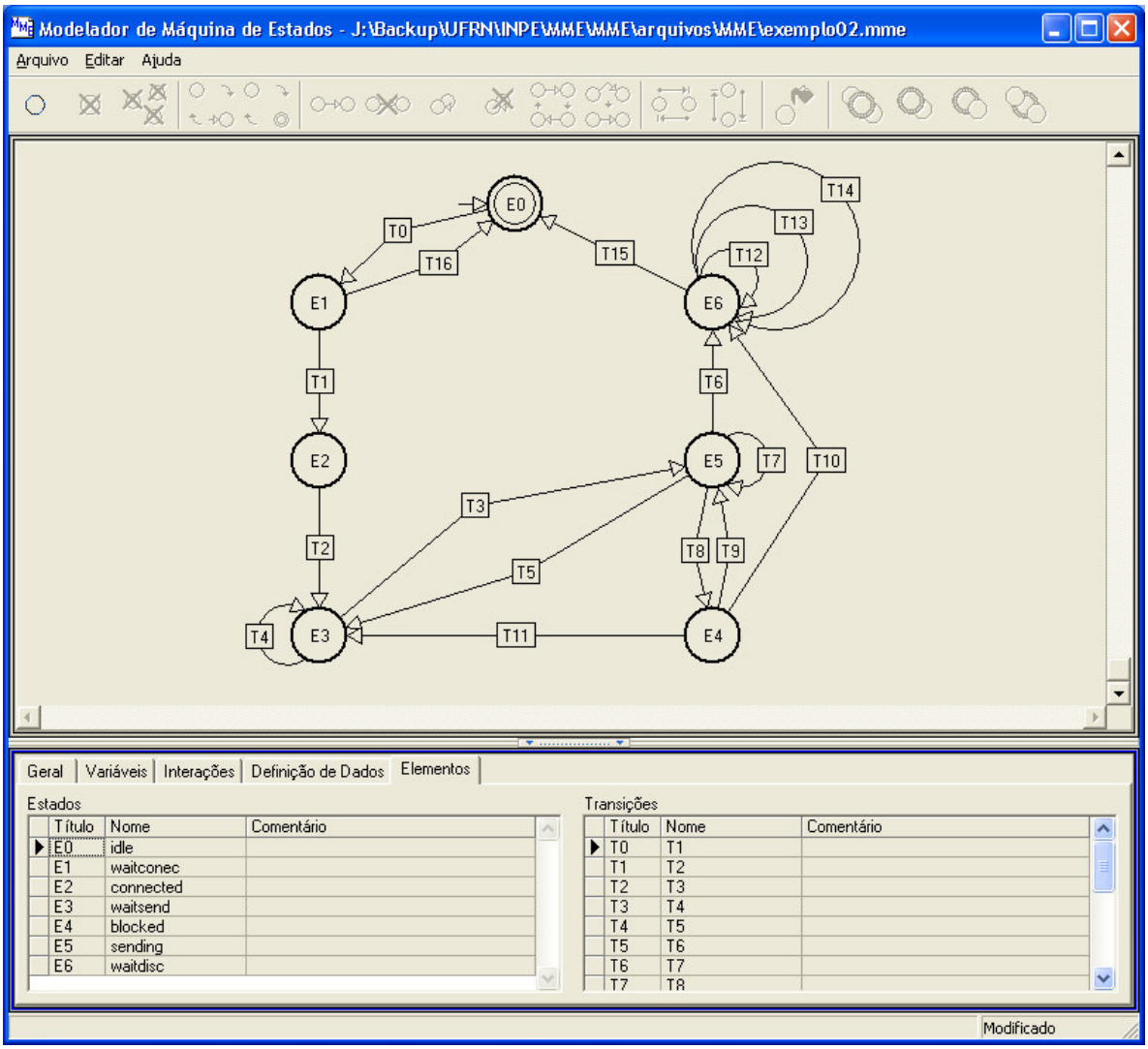
4.3.1.3. Atributos das interações da máquina de estados



4.3.1.4. Atributos dos dados estruturados da máquina de estados

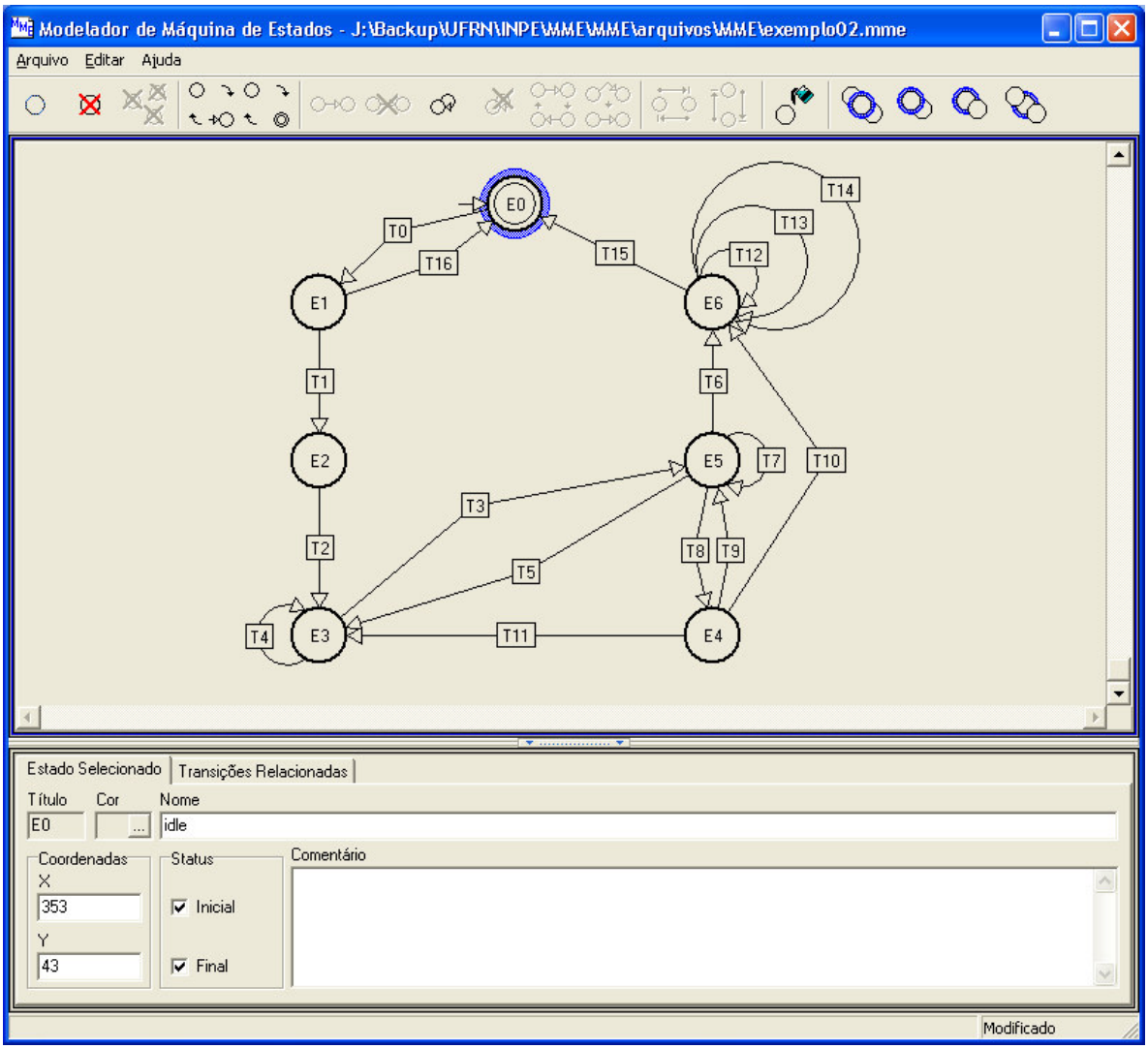


4.3.1.5. Informações gerais sobre os elementos da máquina de estados

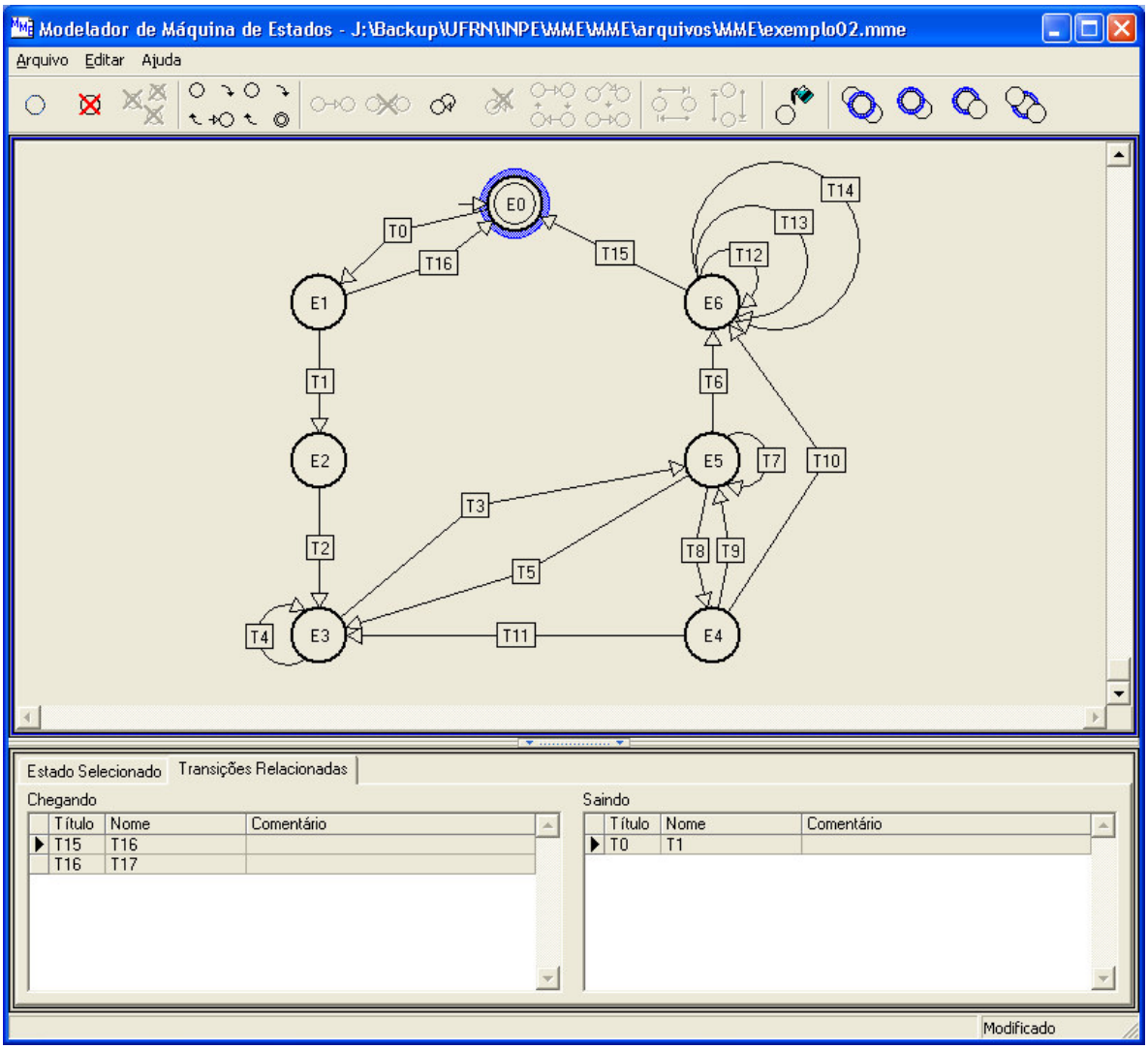


4.3.2. Interfaces ligadas aos atributos do estado

4.3.2.1. Atributos gerais do estado

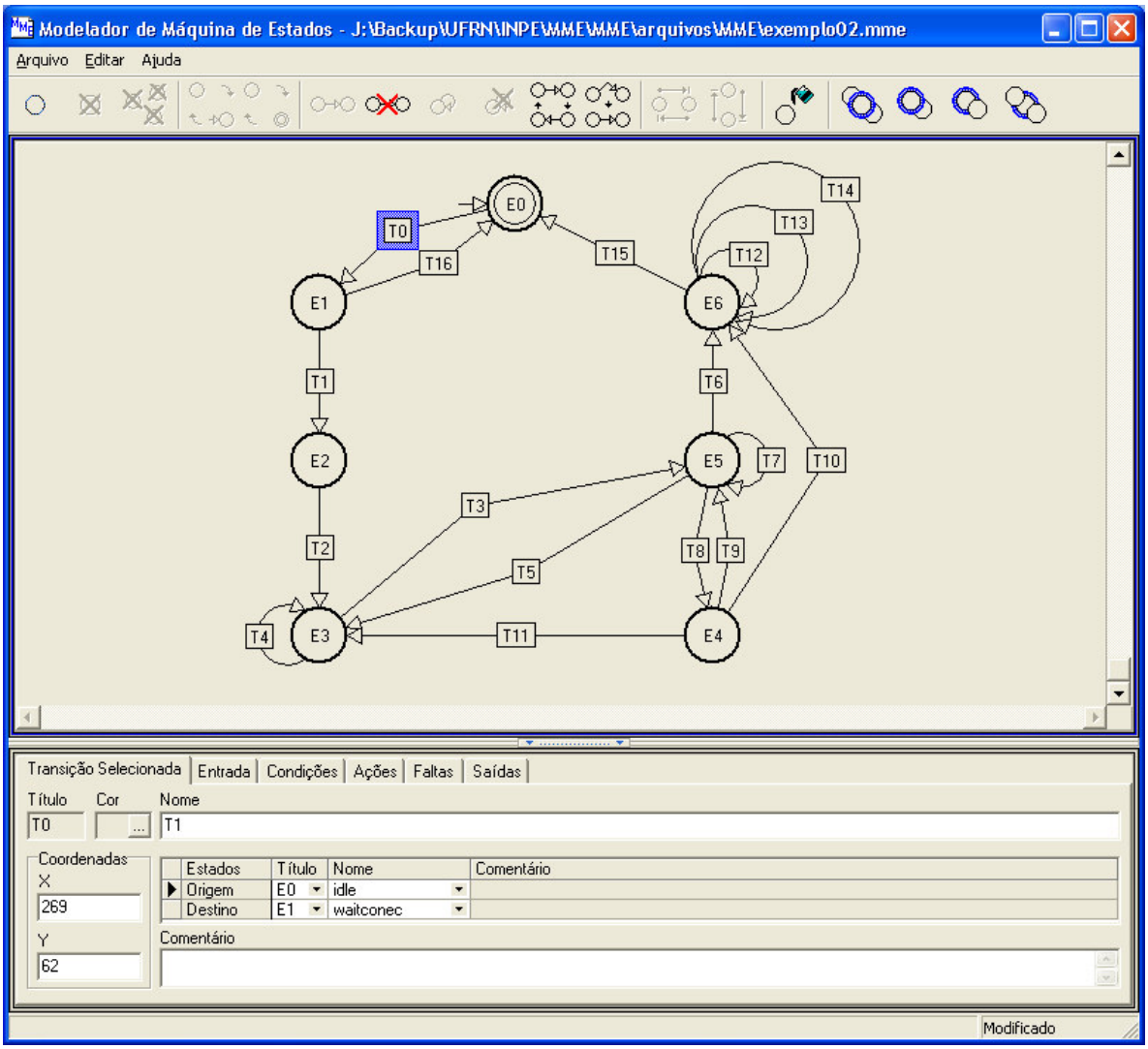


4.3.2.2. Informações gerais sobre as transições relacionadas ao estado

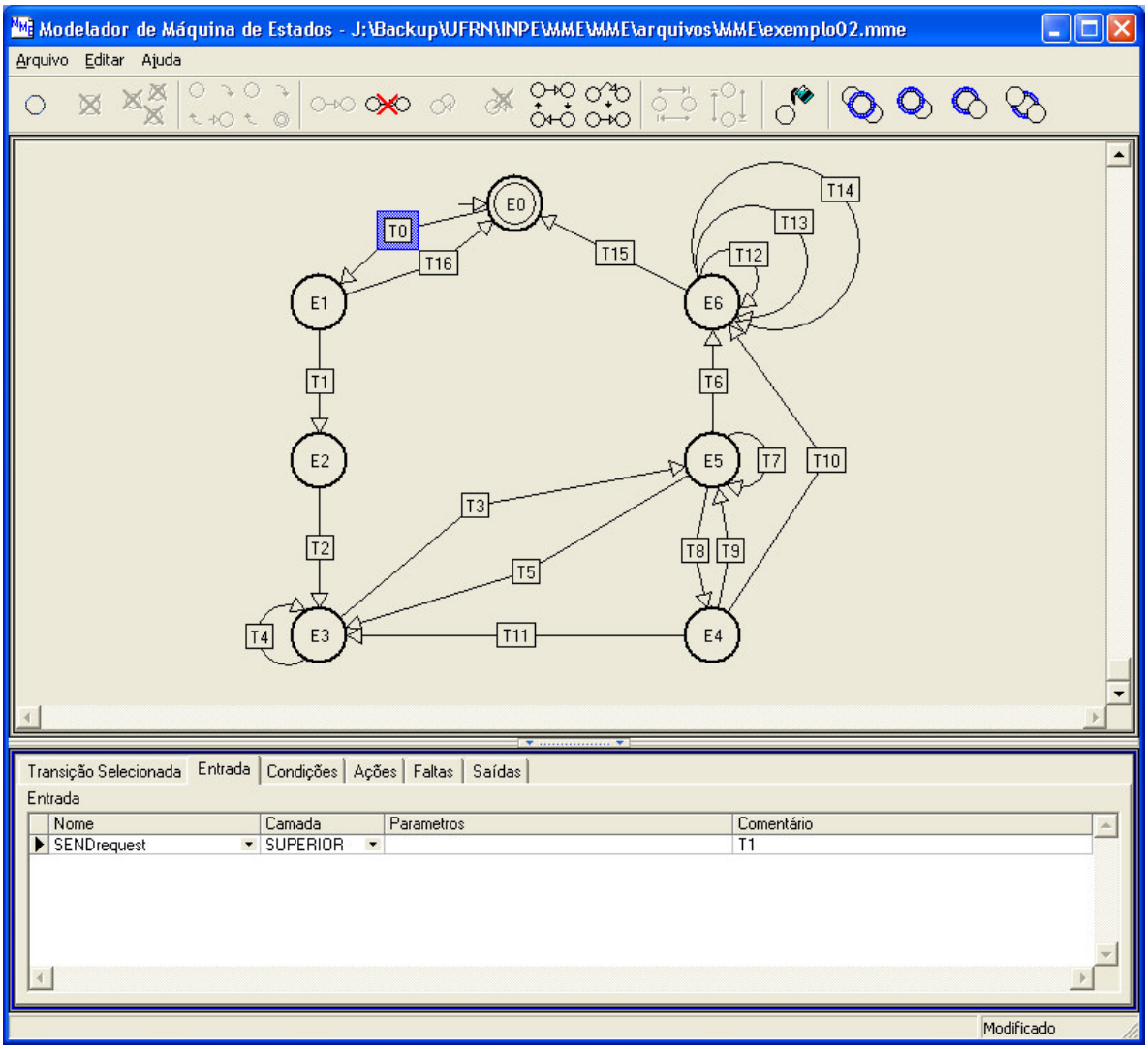


4.3.3. Interfaces ligadas aos atributos da transição

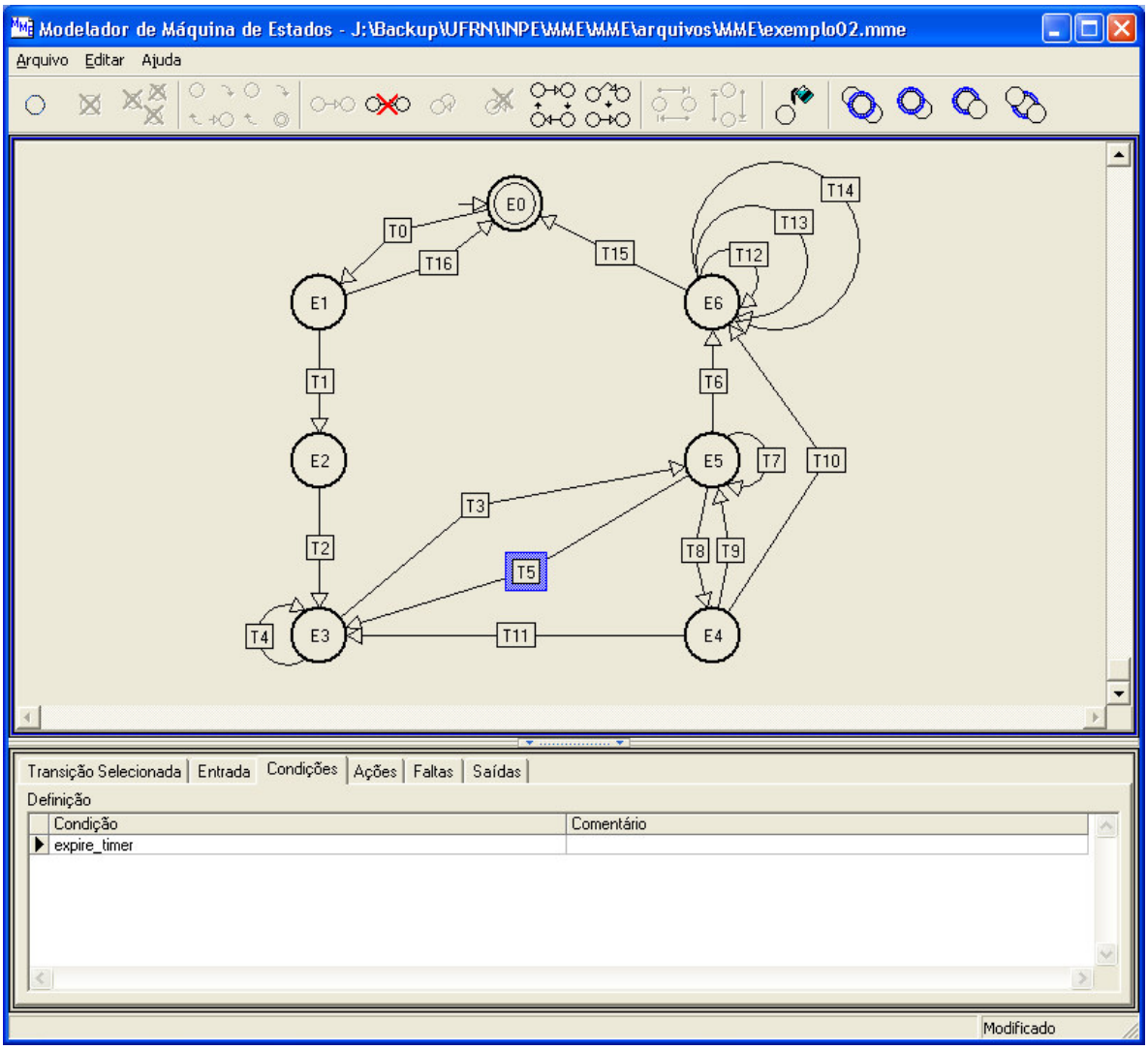
4.3.3.1. Atributos gerais da transição



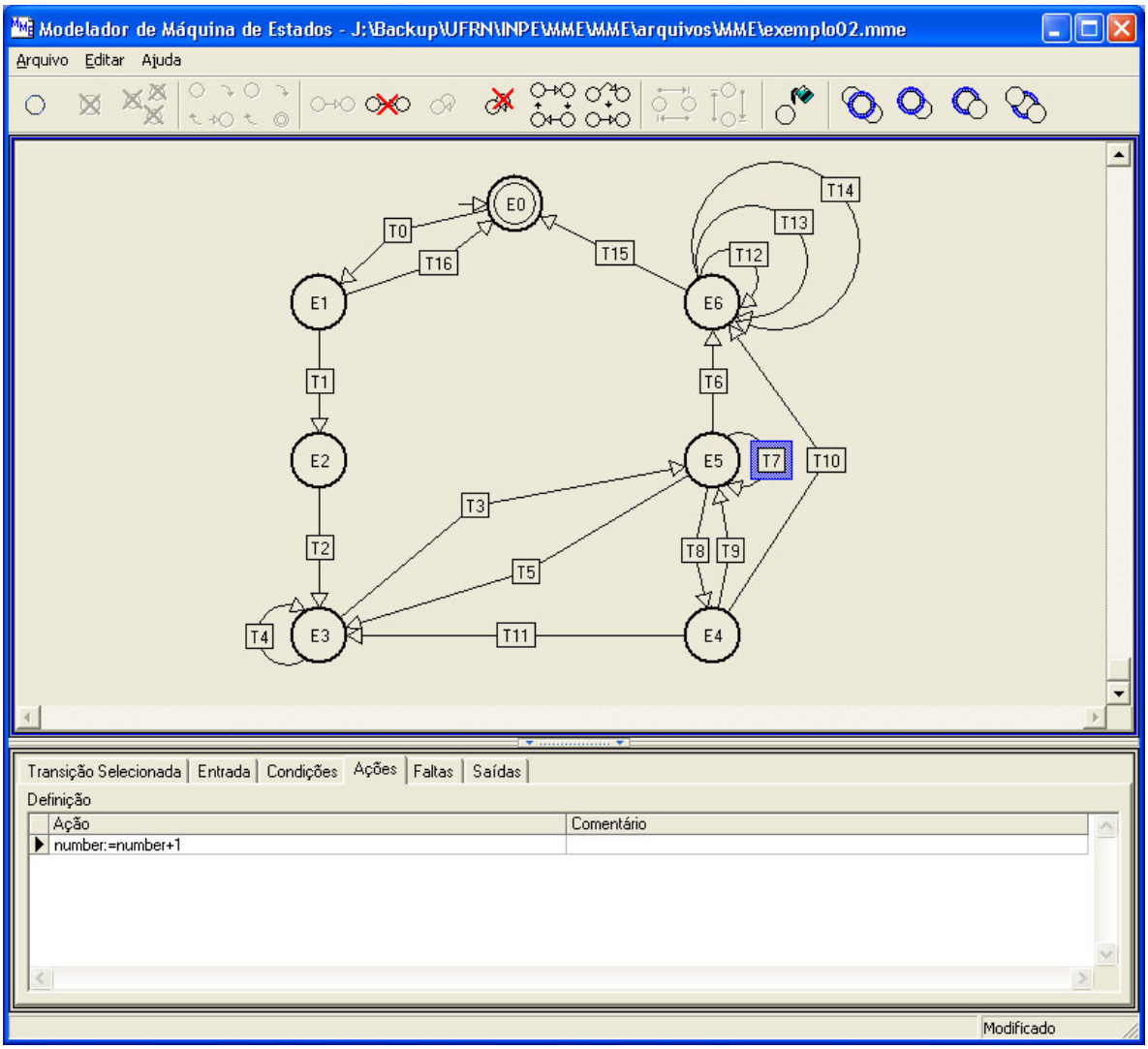
4.3.3.2. Atributos das entradas relacionadas à transição



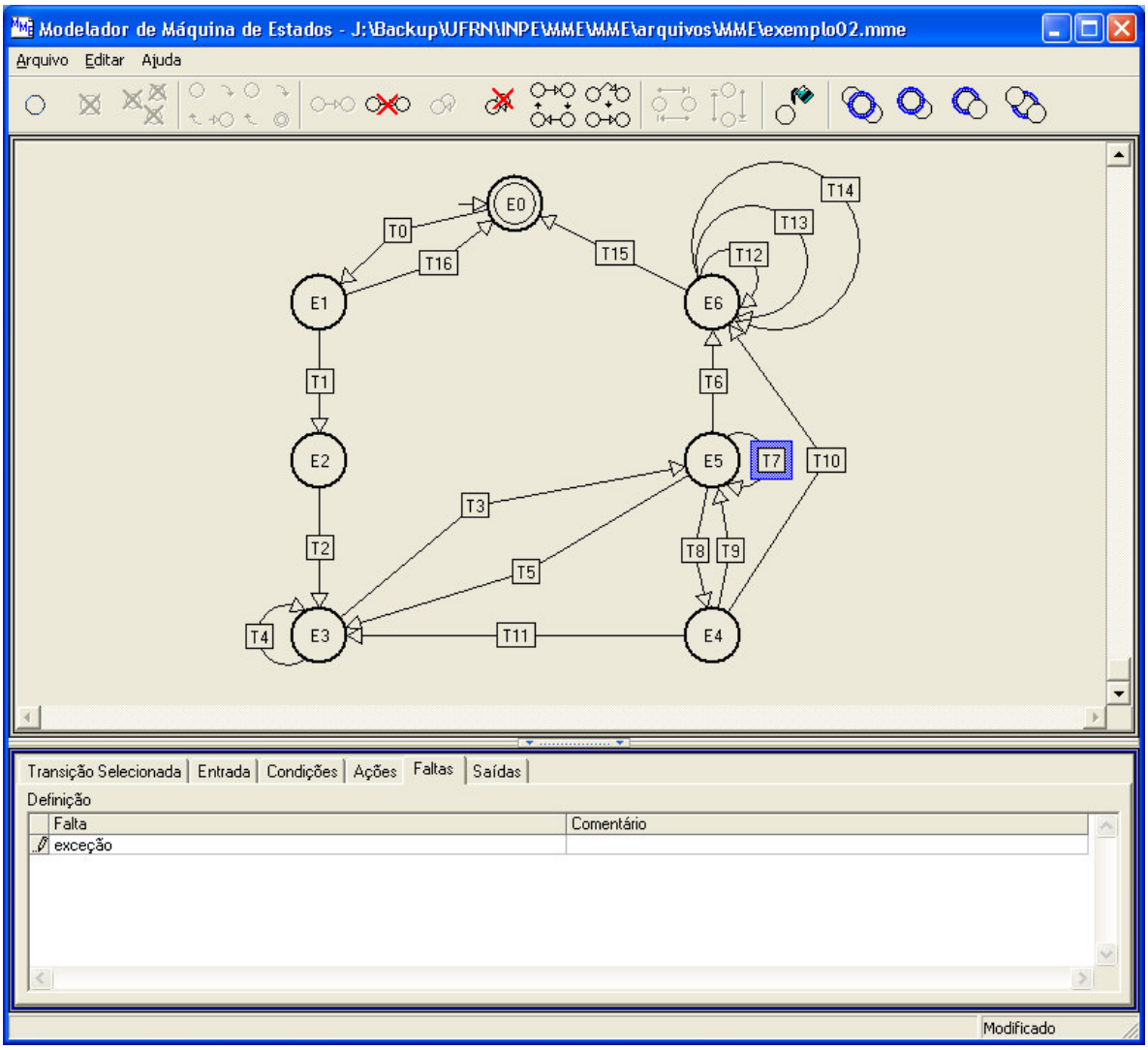
4.3.3.3. Atributos das condições relacionadas à transição



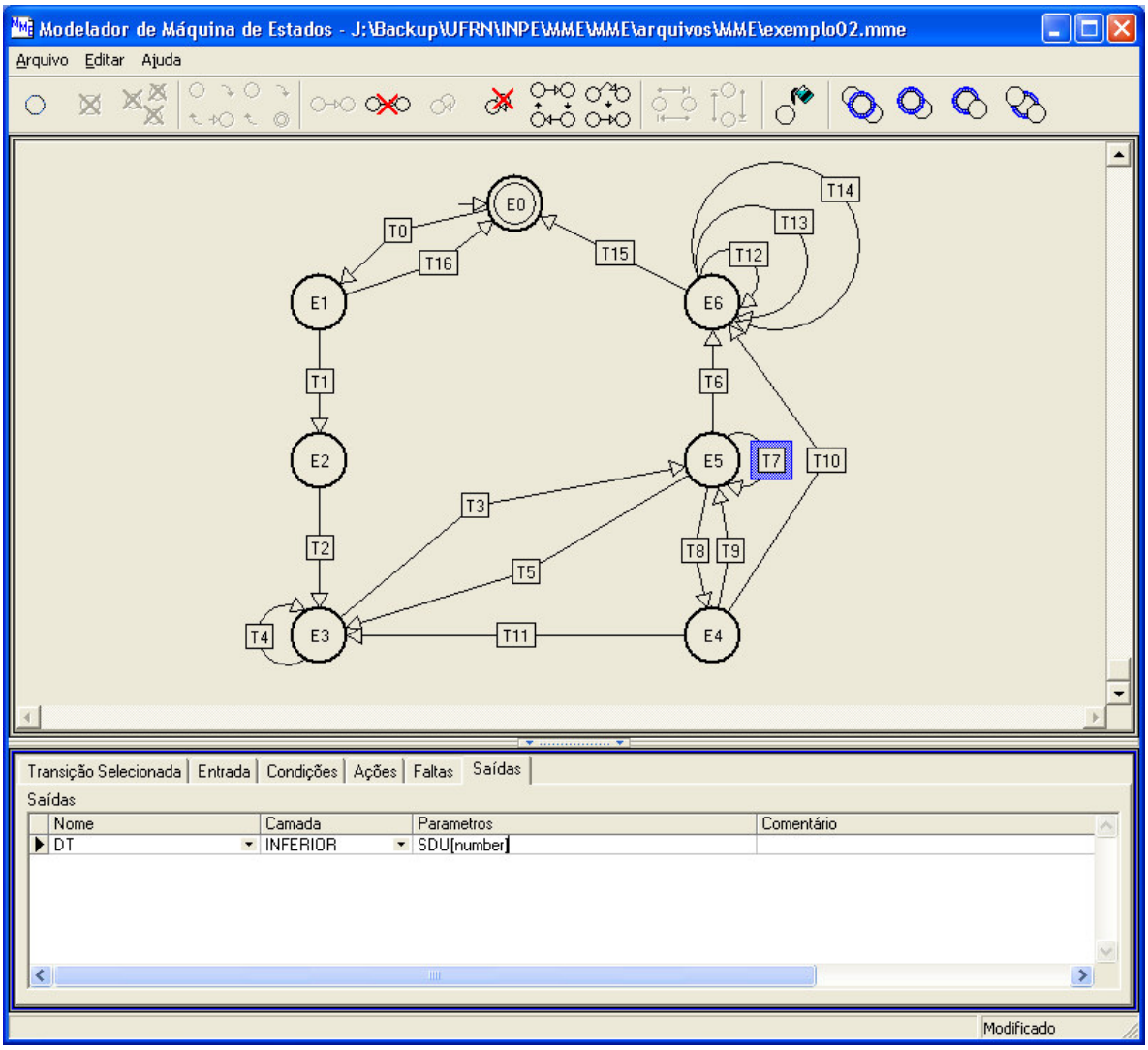
4.3.3.4. Atributos das ações relacionadas à transição



4.3.3.5. Atributos das faltas relacionadas à transição



4.3.3.6. Atributos das saídas relacionadas à transição



4.3.4. Armazenamento dos dados

Para se guardar os dados de uma especificação de forma persistente foram adicionadas as funcionalidades de gravação e carregamento de uma especificação, sendo possível então as operações de criar uma nova especificação, abrir uma especificação existente, salvar a especificação corrente e salvar a especificação corrente com outro nome. Os locais onde essas funcionalidades foram introduzidas são apresentados na figura 18 a seguir.

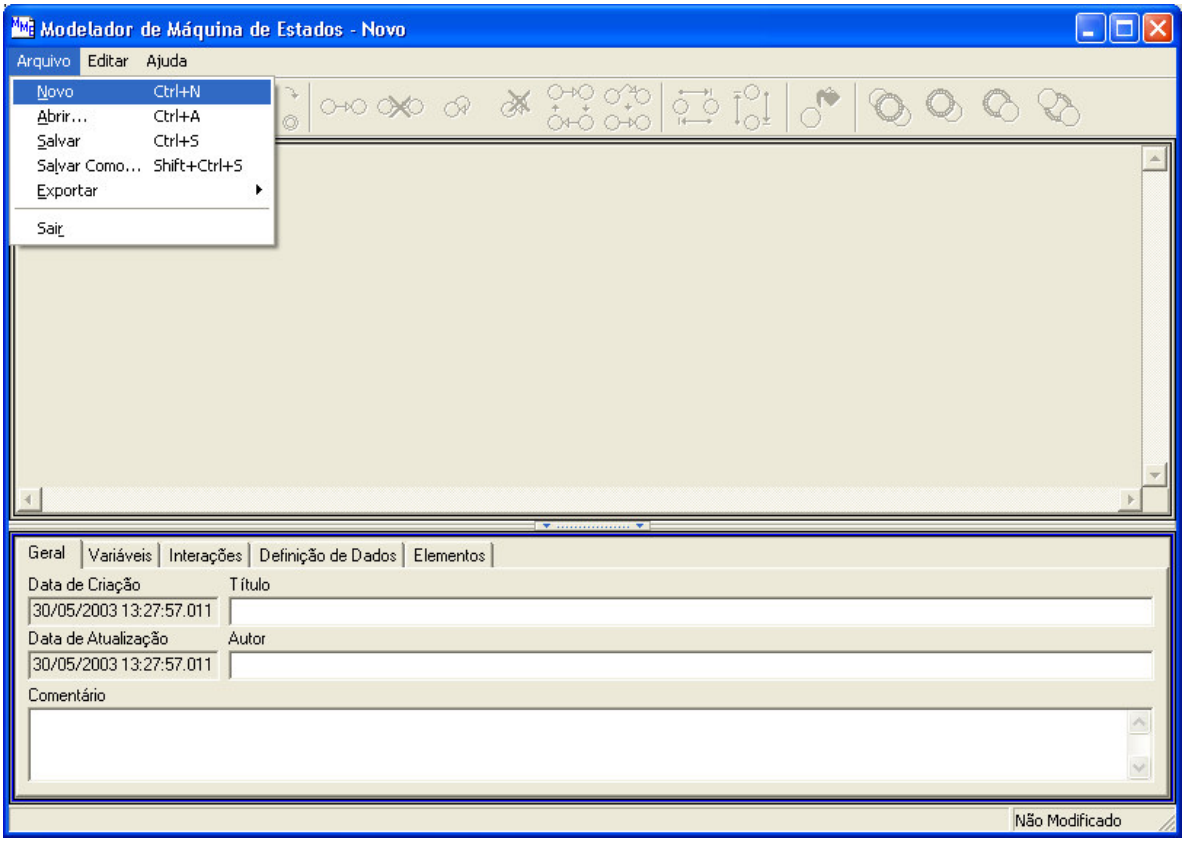


Figura 18 – Localização das funcionalidades de armazenamento e recuperação dos dados

4.3.5. Exportação dos dados

Neste ponto foi desenvolvida a funcionalidade principal do trabalho que é a exportação da especificação corrente para LEP, além disso, resolveu-se acrescentar a exportação para base de fatos que corresponde às Cláusulas de Horn e para imagem onde esta última contém apenas o desenho da máquina de estados.

A figura 19 a seguir indica onde estas funcionalidades foram colocadas.

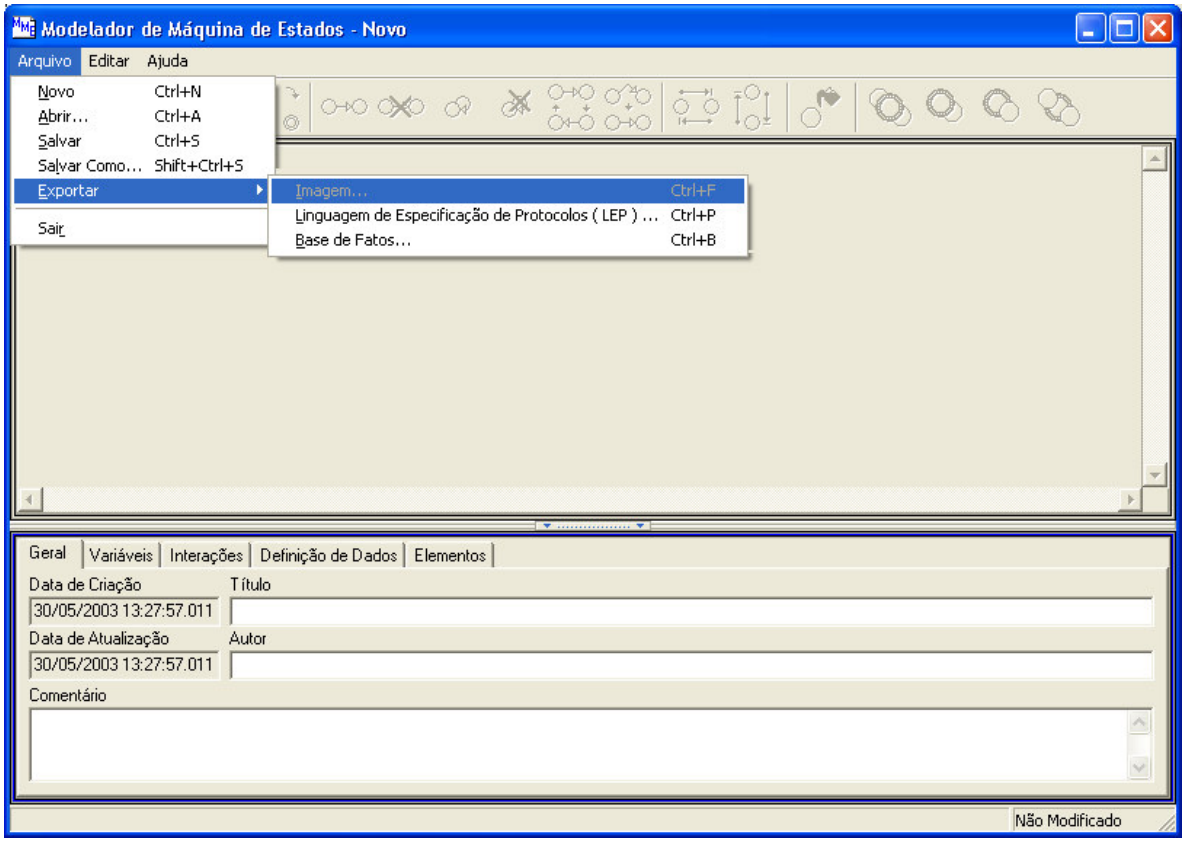


Figura 19 – Localização das funcionalidades de exportação dos dados

5. Exemplo de geração da LEP pela MME

Como exemplo de geração da LEP vamos considerar a máquina de estados exposta na figura 20 a seguir:

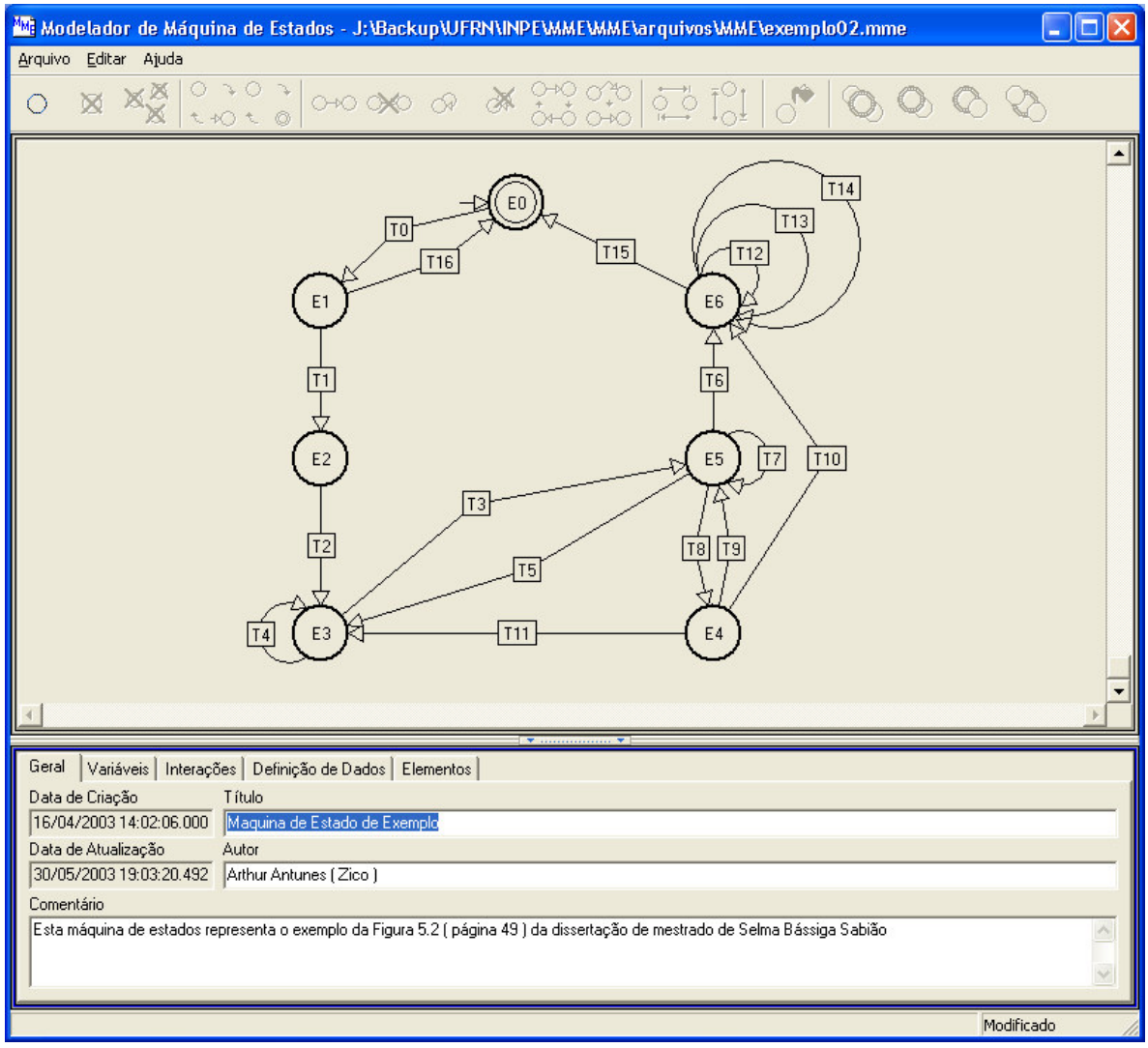


Figura 20 – Exemplo de máquina de estados

A especificação em LEP gerada pelo MME para a máquina de estados exposta anteriormente é a seguinte:

```
VARIABLES:

  number: INTEGER;
  counter: INTEGER;

STATES:

  #idle;
  waitconec;
  connected;
  waitsend;
  blocked;
  sending;
  waitdisc;

INPUTS:

  SENDrequest;
  CC;
  DATArequest;
  TOKENgive;
  RESUME;
  ACK;
  BLOCK;
  DISrequest;

OUTPUTS:

  DISrequest;
  CR;
  SENDconfirm;
  DT;
  TOKENrelease;
  MONITORuCOMPLETE;
  TOKENuRELEASE;
  MONITORuINCOMPLETE;
  DISindication;

DATA:

  DATArequest ::= SEQUENCE { SDU                 OCTETSTRING  5,
                             numberuofusegment   ENNUMERATED  2|4|8,
                             blockbound          INTEGER      3..15
                           };

TRANSITIONS:

  *T1
    >idle
      ?U.SENDrequest
      !L.CR
    <waitconec;

  *T2
```

```
  >waitconec
    ?L.CC
    !U.SENDconfirm
  <connected;

*T3
  >connected
    ?U.DATArequest( SDU, numberuofusegment, blockbound )
    {counter:=0}
    {nomber:=0}
  <waitsend;

*T4
  >waitsend
    ?L.TOKENgive
    {start_timer}
    {number:=number+1}
    !L.DT(SDU[number])
  <sending;

*T5
  >waitsend
    ?L.RESUME
  <waitsend;

*T6
  >sending
    [expire_timer]
    !L.TOKENrelease
  <waitsend;

*T7
  >sending
    ?L.ACK
    [number=numberuofusegment]
    !U.MONITORuCOMPLETE(counter)
    !L.TOKENrelease
    !L.DISrequest
  <waitdisc;

*T8
  >sending
    ?L.ACK
    [number<numberuofusegment]
    [not expire_timer]
    {number:=number+1}
    !L.DT(SDU[number])
  <sending;

*T9
  >sending
    ?L.BLOCK
    [not expire_timer]
    {counter:=counter+1}
  <blocked;

*T10
  >blocked
```

```
      ?L.RESUME
      [not expire_timer]
      [counter<=blockbound]
    <sending;

  *T11
    >blocked
      [counter>blockbound]
      !L.TOKENrelease
      !U.MONITORuINCOMPLETE(number)
      !L.DISrequest
    <waitdisc;

  *T12
    >blocked
      [counter<=blockbound]
      !L.TOKENrelease
    <waitsend;

  *T13
    >waitdisc
      ?L.RESUME
    <waitdisc;

  *T14
    >waitdisc
      ?L.BLOCK
    <waitdisc;

  *T15
    >waitdisc
      ?L.ACK
    <waitdisc;

  *T16
    >waitdisc
      ?L.DISrequest
      !U.DISindication
    <idle;

  *T17
    >waitconec
      ?L.DISrequest
      !U.DISindication
    <idle;

END.
```

6. Conclusões

O objetivo do trabalho foi satisfatoriamente atingido, porém, tendo em vista que a modelagem teve como base a especificação da LEP e principalmente a sua gramática, alguns pontos ficaram livres, isto é, a critério dos usuários pois na definição da gramática não existia uma especificação exata para eles.

Essa ferramenta é de fundamental importância, já que ela foi desenvolvida para permitir uma utilização fácil e intuitiva proporcionando que qualquer usuário com conhecimento mínimo de computação consiga utilizá-la facilmente, além de libertá-los do conhecimento obrigatório da estrutura da LEP pois a especificação nesta linguagem é construída de maneira automática.

7. Trabalhos Futuros

Apesar do objetivo ter sido plenamente atingido, ou seja, a ferramenta oferece as funcionalidades que se desejava para a modelagem da máquina de estados finita estendida, ela pode ser utilizada como base para expansões cuja essência seja a mesma do trabalho realizado, como, por exemplo, a modelagem de redes de Petri, entre outras.

Outro aspecto a ser considerado diz respeito à portabilidade da ferramenta a qual está restrita ao ambiente Windows. Portanto, seria bastante interessante torná-la portável para outras plataformas ou até mesmo desenvolvê-la para executar na internet o que quebraria quase que totalmente essas barreiras.

8. Bibliografia

CHRISTIN, N. **A deterministic finite automata applet simulator**. Disponível em: http://www.cs.virginia.edu/~nc2y/dfa/index.html. Acesso em: 06/09/2002.

TAN, Q. M. **A test generation tool for specifications in the form of state machines**. [paper]. Montreal: Université de Montreal: 1999.

AMBRÓSIO, A. M. **ATIFS – Ambiente de teste baseado em injeção de falhas por software**. [relatório]. São José dos Campos: INPE, 1999.

MARTINS, E. **ConData**: a tool for automating specification-based test case generation for communication systems. [paper]. Campinas: UNICAMP, 2000.

CANTÚ, M. **Dominando o Delphi 5 "A Bíblia"**. São Paulo: Makron Books, 2000.

Møller, A. **Finite-State Automata for Java™**. Disponível em: http://www.brics.dk/~amoeller/automaton/. Acesso em: 06/09/2002.

KRÖNERT, K.; DALLMANN, U. **FSM editor and simulator**. Disponível em: http://tech-www.informatik.uni-hamburg.de/applets/java-fsm/index.html. Acesso em: 03/09/2002.

ACIÓLY, B. M.; BEDREGAL, B. R. C.; LYRA, A. **Introdução à Teoria das Linguagens Formais, dos Autômatos e da Computabilidade**. Natal: Edições UnP, 2002.

RUMBAUGH, J. **Modelagem e projetos baseados em objetos**. Rio de Janeiro: Campus, 1994.

MERZ, S. **State Machine Simulator**. Disponível em: http://siskin.pst.informatik.uni-muenchen.de/~merz/software/fsm/Simulator.htm. Acesso em: 03/09/2002.

CHAPMAN, M. **The Finite State Machine Explorer**. Disponível em: www.belgarath.demon.co.uk/java/fsme.html. Acesso em: 06/09/2002.

BRAINERD, W. S. **Theory of computation**. Nova York: John Wiley & Sons, 1974.

SABIÃO, S. B. **Um método para geração de testes baseado em máquina finita de estado estendida combinando técnicas de teste de caixa preta**. Campinas, 1998. Dissertação - Instituto de Computação da Universidade Estadual de Campinas.

Booch, G.; Rumbaugh, J.; Jacobson, I. **UML Guia do Usuário**. Rio de Janeiro: Campus, 2000.

LARMAN, C. **Utilizando UML e padrões**: uma introdução à análise e ao projeto orientado a objetos. Porto Alegre: Bookman, 2000.