UNIVERSIDADE ESTADUAL DE CAMPINAS
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

"PROTR - UM PROTOTIPADOR PARA SISTEMAS DE TEMPO REAL".

DISSERTAÇÃO SUBMETIDA À UNIVERSIDADE ESTADUAL DE CAMPINAS COMO PARTE DOS REQUISITOS PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA

ALUNA : GLAUCIA DANTAS FRANCO AZEVEDO
ORIENTADOR : MARIO JINO

CAMPINAS, OUTUBRO DE 1993.

Dedico ao
Helio, meu marido,
pelo incentivo, compreensão e
colaboração, que tanto me motivaram
para conclusão deste trabalho

e

Felipe, nosso filho,
pelas alegrias do dia-a-dia.

Agradecimentos.

Esta tese é um marco em minha vivência junto à FCTI - Fundação Centro Tecnológico para Informática. Desta forma, seria redundante falar da importância desta Instituição em minha formação profissional, mas é inequívoco o crescimento que me proporcionou no âmbito da pesquisa.

Agradeço à FCTI pela oportunidade da realização deste trabalho e, em especial às pessoas do IA - Instituto de Automação que apoiaram e viabilizaram o desenvolvimento deste trabalho.

Os meus agradecimentos ao Prof. Mario Jino, pelo seu acompanhamento, orientação objetiva, sugestões, discussões, críticas e confiança durante todas as fases de elaboração deste trabalho, desde sua definição até a apresentação final.

Ao Fabio Nauras Akhras, pelo incentivo, amizade e participação na fase de definição deste trabalho, sem cujo apoio inicial, não teria sido realizado.

Uma referência especial é devida ao Rubens Campos Machado, Chefe de Departamento de Controle de Processos, e Paulo Cesar Berardi, Chefe da Divisão de Sistemas Dedicados, pela confiança depositada e apoio dado ao desenvolvimento da tese.

Agradeço em especial ao Helio, meu marido, pelo seu estímulo permanente, paciência e confiança que depositou em mim, além da carinhosa convivência destes anos, sem os quais esta tese não teria sido realizada. Foi, na prática, uma espécie de co-orientador; que com amizade, incentivo e dedicação apoiou e auxiliou efetivamente na realização deste trabalho.

RESUMO

Sistemas de Tempo real constituem uma área em que a utilização de computadores cresceu significativamente na última década, paralelamente ao desenvolvimento de técnicas e à proposição de abordagens de engenharia de software tendo por finalidade a produção de software de uma maneira efetiva e econômica. Aplicações de Tempo Real possuem características distintas das de outros tipos de aplicações e que impõem técnicas e modelos adequados para o desenvolvimento de software.

O objetivo deste trabalho é apresentar o desenvolvimento de um Prototipador (ProTR) para auxiliar no projeto de Sistemas de Tempo Real, baseado na metodologia "Desenvolvimento Estruturado para Sistemas de Tempo Real", introduzida por Ward e Mellor.

ABSTRACT

Real Time Systems is an application area where digital computer usage has grown significantly in the last decade, in parallel with the development of techniques and the proposition of software engineering approaches aiming at an effective and economical way of producing software. Real Time applications present features distinct from other applications and demand suitable techniques and models for software development.

The objective of this work is to present a Prototyping system (ProTR), designed to aid the project of Real Time Systems, based on one of the most well-known methodology, "Structured Development for Real-time Systems", due to Ward & Mellor.

ÍNDICE

LISTA DE FIGURAS

página

LISTA DE TABELAS

página

Capítulo 1. INTRODUÇÃO.

A demanda na utilização de sistemas computacionais cresceu significativamente na última década. Nessa expansão, os Sistemas de Tempo Real (STR) representam uma parcela importante das aplicações, variando desde sistemas muito simples até aqueles altamente complexos. Exemplos de STR incluem o controle de experimentos em laboratórios, eletrônica embarcada em automóveis, controle de usinas nucleares, controle de processos e sistema de controle de vôo.

STRs complexos possuem um alto custo de produção e sua funcionalidade é analisada através de testes altamente controlados, ou com custosas simulações.

Nesse contexto, áreas como especificação de sistemas, tolerância a falhas, sistemas operacionais, arquitetura de computadores, banco de dados, comunicação entre processadores, escalonadores e linguagens de programação, tornaram-se alvos de atenção por parte dos pesquisadores com o objetivo de sedimentar as bases necessárias ao desenvolvimento de STRs [STA88].

Na área de especificação de sistemas, uma das contribuições mais difundidas é a metodologia de Paul T. Ward e Stephen J. Mellor [FAL88], que permite expressar as características de ambientes de tempo real como **estímulo** e **resposta a eventos**, permitindo uma melhor visualização da operação do sistema. Esta metodologia representa uma extensão ao clássico método de especificação proposto por Yourdon-DeMarco [DEM78] para tratar Sistemas de Tempo Real, sendo atualmente um dos métodos mais difundidos para ferramentas "CASE" de Tempo Real.

Nos últimos anos, o interesse por parte dos usuários em ferramentas CASE tem se desviado dos recursos gráficos das ferramentas e direcionado para elementos como : especificações executáveis, particionamento automático, prototipação rápida e geração automática de código.

Particularmente, as técnicas de prototipação têm se firmado como um valioso auxílio ao desenvolvimento de sistemas computacionais [DEA83], minimizando o tempo de desenvolvimento e gerando produtos mais confiáveis.

Boar [BOA83] define prototipação como uma estratégia específica para efetuar a definição de requisitos, onde as necessidades dos usuários são extraídas, apresentadas e sucessivamente refinadas, construindo-se rapidamente um modelo executável do sistema alvo em seu ambiente de trabalho.

A principal proposta desta abordagem é construir um Protótipo com a finalidade de acentuar os aspectos críticos do sistema e envolver o usuário nas fases iniciais do desenvolvimento, permitindo que sejam realizadas extensões e verificadas inconsistências e ambiguidades.

Alavi [ALA84] sugere alguns atributos de protótipos de sistemas de informação :

1) Um protótipo é inicialmente incompleto. É construído com a intenção de ser modificado e refinado.

2) Um protótipo é um sistema real, que é avaliado em um modo operacional.

3) A proposta do protótipo é auxiliar no entendimento e aprendizagem por parte do usuário e projetista e servir como um veículo para testes.

4) Em resposta às necessidades do usuário, um protótipo deve ser construído o mais rápido e economicamente possível.

Este trabalho tem por objetivo aplicar técnicas de prototipação à metodologia de desenvolvimento de software de Ward & Mellor, denominada "Desenvolvimento Estruturado para Sistemas de Tempo Real" [WAR86] [WAR85], fornecendo ao projetista uma visão mais concreta do sistema proposto, antes da fase de implementação.

Como resultado, é possível realizar análise dinâmica da consistência do sistema além de fornecer um caminho para a geração automática de código executável.

Uma questão pode ser levantada quanto à utilização da metodologia Ward & Mellor. Neste trabalho, inúmeras linhas de atuação, com relação a Sistemas de Tempo Real e Prototipação, são apresentadas; em princípio, qualquer uma seria apropriada para ser utilizada como base do nosso desenvolvimento. A justificativa pela utilização da Metodologia de Ward & Mellor está baseada nos seguintes aspectos :

- **Alcance** : Um dos princípios fundamentais na realização de um trabalho é o desejo de que ele tenha o maior alcance possível e seja, de alguma forma, útil no contexto onde se insere. A metodologia Ward-Mellor tem sido largamente utilizada pelos analistas, no âmbito de Sistemas de Tempo Real, estando disponível em inúmeras ferramentas CASE [FAL88], até mesmo em CASE nacional [PCC90]. Dessa forma, sua utilização como metodologia base para a prototipação representa um fator importante no alcance obtido pelo ProTR.

- **Apelo Gráfico** : A possibilidade de utilizar os recursos gráficos presentes na metodologia para representar a dinâmica de execução do protótipo (como sugerido por Ward & Mellor), que utiliza um esquema baseado em movimentação de marcas sobre os fluxos, é atraente pela facilidade de acompanhamento da execução do protótipo.

- **Interface com o Usuário** : Um dos objetivos fundamentais das técnicas de prototipação é permitir uma maior interação com o usuário. A utilização de uma metodologia bem aceita no mercado funciona como catalisadora dessa interação.

No próximo capítulo são apresentados conceitos de prototipação segundo diversos autores, bem como seu impacto no ciclo de vida do software e propostas de classificação.

No Capítulo 3 são apresentadas algumas das principais técnicas de prototipação existentes atualmente. Exemplos de utilização são apresentados dando ênfase a sistemas semelhantes ao proposto neste trabalho.

No Capítulo 4 apresenta-se uma breve introdução à metodologia de Ward & Mellor. Esta introdução não é completa, sendo recomendável a leitura de [WAR85] para possíveis usuários.

No Capítulo 5 a ferramenta ProTR [AZE91b] [AZE91d] é introduzida, descrevendo seu objetivos, forma de utilização e exemplos.

No Capítulo 6 descreve-se o desenvolvimento da ferramenta ProTR, ou seja, detalhes da implementação e estrutura do software.

No Capítulo 7 o trabalho é concluído sendo realçadas as principais características do sistema e planos para extensões à ferramenta obtida.

No Anexo A é apresentado um exemplo de aplicação do ProTR para visualização das fases de execução do sistema.

No Anexo B são apresentadas as funções do Núcleo de Tempo Real NProTR, acessíveis ao usuário.

A ferramenta ProTR é descrita em uma série de documentos (Especificação ProTR, Especificação do Software do Núcleo de Tempo Real NProTR, Manual de Usuário do ProTR), cujo volume torna inviável a sua inclusão neste trabalho; os Anexos C, D e E contêm somente os grafos sintáticos do "Esquema de Transformações", do "Dicionário de Dados" da metodologia, e do "Cenário", respectivamente.

Capítulo 2. PROTOTIPAÇÃO.

2.1. Conceituação de Prototipação.

O ciclo de vida convencional do software tem se mostrado inadequado em algumas áreas mais complexas [DEA83]. A Figura 2.1. apresenta o ciclo de vida tradicional do software, dividido em 2 estágios : desenvolvimento e operação & manutenção [CVR84].

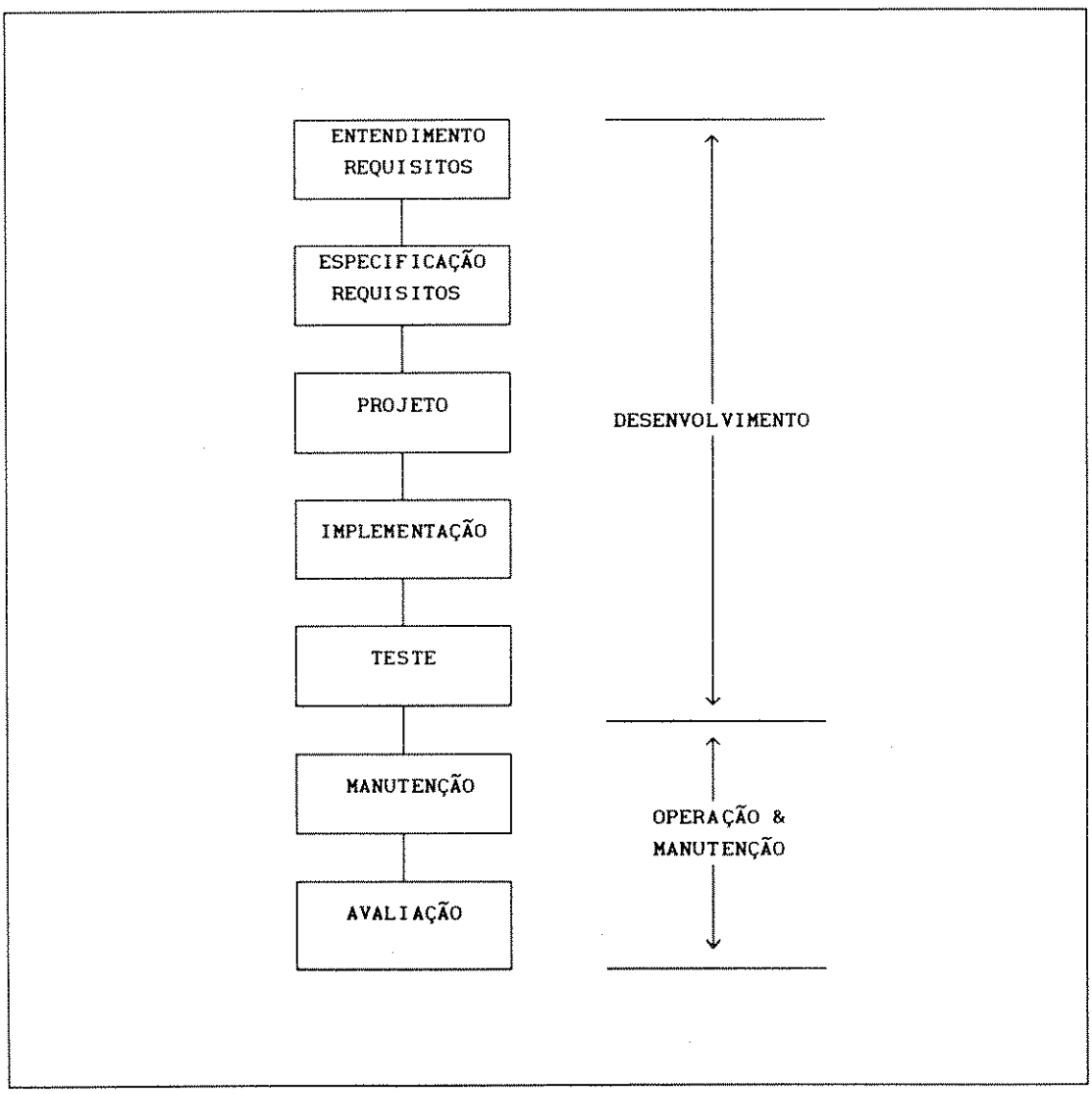


Figura 2.1. Ciclo de Vida do Software.

Na análise de requisitos são levantadas as necessidades do usuário, que são funcionalmente agrupadas e analisadas na fase de projeto. Uma vez que o projeto esteja concluído, é realizada a implementação que direciona às fases de testes e aceitação.

Este modelo requer tempo na fase de desenvolvimento de especificações e permite pouco "feedback" por parte do usuário até o estágio de codificação e testes.

Nesta abordagem o usuário é passivo, tendo participação apenas na fase inicial, quando o analista investiga o problema. Qualquer equívoco no estágio de análise é convertido em erros no final do sistema [DEA83], causando dificuldades nas fases de implementação e/ou manutenção, onerando o custo total do sistema.

A passividade do usuário está relacionada com sua dificuldade em especificar requisitos no nível de detalhes que o projetista necessita. Na maioria das vezes, os sistemas falham muito mais devido a especificação inadequada do que em razão do projeto técnico pobre [ALA86].

O ideal seria que o usuário tivesse participação durante as fases de especificação e implementação do sistema alvo. Prototipação tem sido apontada como um dos caminhos alternativos para evitar alguns inconvenientes do modelo convencional de ciclo de vida do software. O principal inconveniente do ciclo de vida tradicional está associado ao isolamento do usuário durante as fases de projeto e implementação. Após sua participação na geração de requisitos, o usuário só é novamente consultado na fase de testes quando o produto está concluído e nem sempre corresponde às suas expectativas.

O uso da estratégia de prototipação, durante as fase de análise e projeto, torna o desenvolvimento do sistema um processo dinâmico [DEA83]; o usuário e o analista estão em contínua comunicação. O usuário avalia a proposta

através da utilização do protótipo, verificando se a solução apresentada se adapta às suas necessidades e à sua forma de trabalho [STU87].

Prototipação pode ser definida como uma técnica que reconhece que o desenvolvimento do sistema é um processo interativo [GIL86] [HER89]. Permite mostrar ao usuário o que será desenvolvido; a proposta é extrair do usuário a informação requerida e, em seguida, permitir a sua visualização em um contexto real.

A apresentação do protótipo constitui um momento essencial no ciclo de desenvolvimento. É neste momento que avaliam-se as premissas originais e as características da solução proposta [STU87].

A metodologia de desenvolvimento por prototipação difere em grande parte da convencional em suas atividades e resultados produzidos nas fases individuais. Embora a distinção entre as fases seja mantida, a análise e especificação do problema se sobrepõem e, projeto, implementação e teste se misturam [POM91].

O protótipo pode ser considerado como a base para o desenvolvimento do resto do sistema. Inicialmente é construído mais direcionado à funcionalidade do que à lógica; após a aprovação, é adicionada a lógica restante conservando a sua funcionalidade. Esta forma de prototipação reduz a redundância no esforço do desenvolvimento [GIL87].

Prototipação oferece um caminho que resulta em um modelo executável do software a partir do qual os requisitos podem ser refinados. Para conduzir prototipação apropriadamente, técnicas e ferramentas especiais são requeridas [PRE88].

É importante salientar a diferença entre o protótipo e o sistema final. O protótipo usualmente não atende todos os requisitos do usuário, implementa somente os aspectos mais importantes, permitindo intervenções por parte do usuário antes que o sistema final seja projetado [CVR84].

Os requisitos do sistema são acordados interativamente na estratégia de prototipação rápida através da operação de protótipos executáveis. O projetista constrói um protótipo baseado nos requisitos e examina a execução do protótipo juntamente com o cliente. A partir desta avaliação, os requisitos são ajustados e o protótipo é modificado até atender as necessidades do sistema alvo.

O principal benefício de prototipação consiste em poder-se modificar o comportamento do protótipo com menor esforço do que o requerido para modificar o produto final. A Figura 2.2. ilustra este processo apresentando uma forma de integrar prototipação ao ciclo de vida tradicional [ALA86].

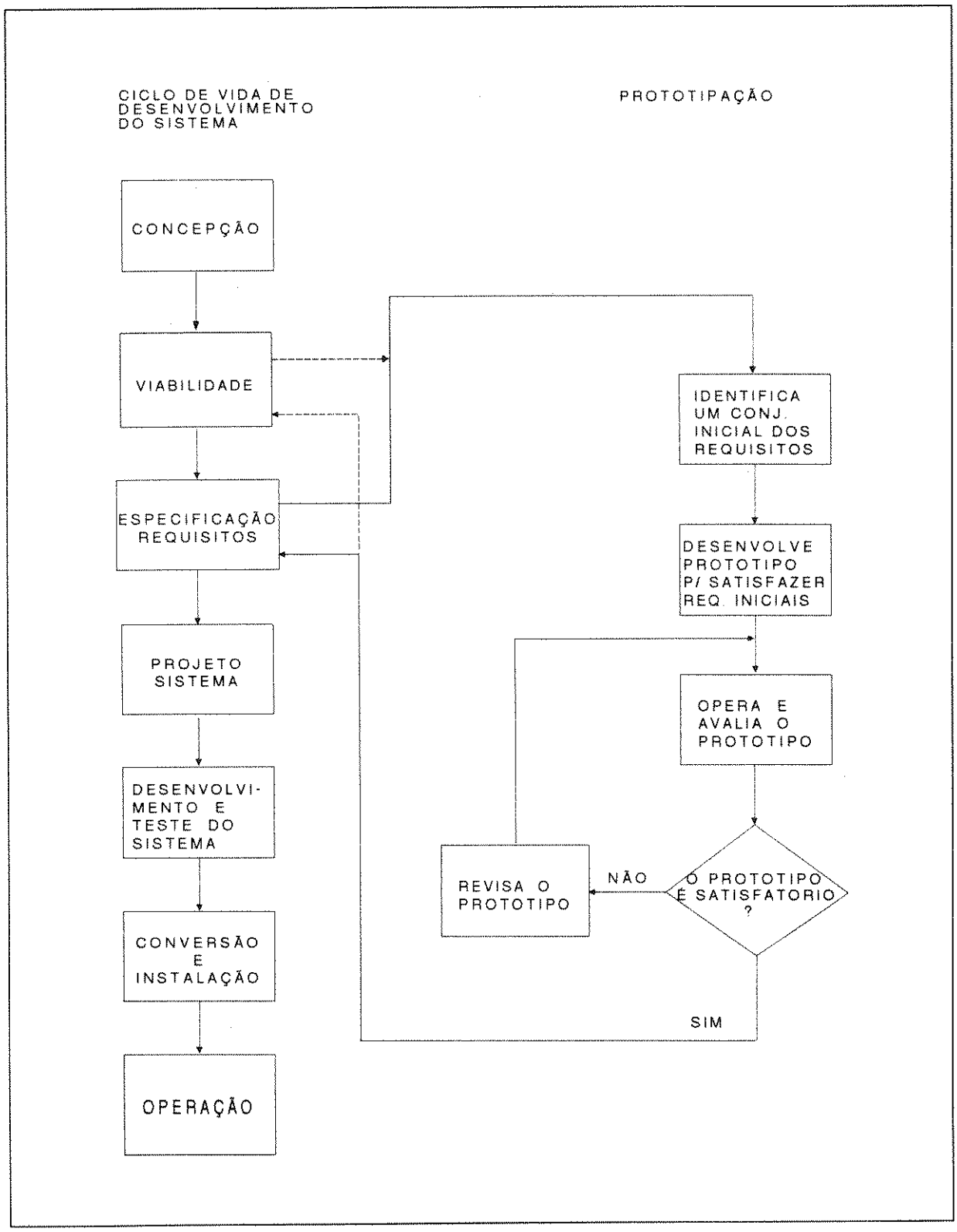


Figura 2.2. Integração de Prototipação no Ciclo de Vida Tradicional.

Dearnley e Mayhew [DEA83] advertem quanto à importância de não confundir os termos "protótipo" e "modelo". Embora apareçam no dicionário como sinônimos, no contexto da aplicação apresentam uma diferença considerável. O modelo serve como uma representação pictórica do produto desejado, enquanto o protótipo é um sistema que funciona, podendo se transformar no produto final.

Finalmente, vamos utilizar a definição de Dearnley e Mayhew para ressaltar uma "Prototipação Satisfatória" [MAY90] : "Prototipação é o processo de construir e avaliar modelos de trabalho de um sistema com a finalidade de APRENDER sobre certos aspectos do sistema requerido e/ou sua solução potencial".

O componente principal desta definição recai sobre a palavra APRENDER. Existem muitos aspectos do sistema que podem ser clarificados com o uso de prototipação, muito a ser APRENDIDO, como :

- descobrir os requisitos do sistema.
- experimentar com projetos alternativos.
- avaliar o efeito na organização.
- avaliar o hardware disponível.
- estimar níveis de desempenho.

Considerando-se este aspecto de aprendizagem, pode-se acrescentar a opinião apresentada em [CER87] : Prototipação é uma excelente técnica para promover o processo de aprendizagem mútua entre analistas e usuários.

Existem dois tipos principais de modelos de ciclo de vida para protótipos. No primeiro, o protótipo é construído como um sistema "Throw-away" [VAC86]; ou seja, após o usuário ficar satisfeito, o protótipo é descartado quando, então, é iniciada a implementação do sistema final. O protótipo do tipo "Throw-away" consiste de representações, modelos de partes específicas do

sistema funcional. Neste caso, as ferramentas (e a linguagem) com o qual o protótipo é desenvolvido, em muitos casos, difere das que farão parte do sistema final. Logo, somente o conceito funcional e as especificações formais são preservadas [KEU82].

O segundo modelo de ciclo de vida se refere ao protótipo que se comporta como um sistema inicial que será refinado até o definitivo. Algumas vezes o protótipo, ou porções dele, podem ser incorporadas ao sistema final [KLI86]. Essa estratégia considera o desenvolvimento de sistemas como um caminho iterativo; ou seja, após diversos passos intermediários e sucessivos refinamentos, é concluído o sistema final. São considerados protótipos "add-on", pois são continuamente modificados e ampliados [KEU82].

2.2. Impacto de Prototipação.

Um protótipo é construído para ilustrar a viabilidade de novas idéias ou projetos. O protótipo frequentemente começa com especificações incompletas, mas os erros existentes são reparados por rápidas alterações locais. A experiência obtida na construção de um protótipo resultará em um produto melhor com um custo total inferior [GEH82].

Em um experimento de desenvolvimento de software realizado com algumas equipes utilizando o caminho de Especificação e outras, o de Prototipação, foram verificados os seguintes resultados [BOE84] :

1. Prototipação gerou produto com desempenho equivalente, mas com 40% a menos de código e 45% a menos de esforço.

2. O produto prototipado apresentou funcionalidade inferior, mas superioridade quanto ao uso e facilidade de aprendizagem.

3. A especificação produziu projetos mais coerentes e software mais fácil de integrar.

Podemos observar outras conclusões obtidas no desenvolvimento de um sistema de informação em uma Indústria Química [ALA84] :

Em geral os usuários do sistema prototipado apresentaram uma avaliação mais favorável e demonstraram maior satisfação com o sistema de informação, alegando que o sistema apresentou alta precisão e melhor desempenho. Em termos de tempo de resposta e facilidade de uso, nenhuma diferença foi observada.

Os usuários demonstraram satisfação quanto ao nível de participação e perceberam menos conflitos com os projetistas. Estes últimos, por sua vez, observaram um alto grau de alteração nas especificações do usuário durante o processo do projeto e sentiram maior dificuldade em gerenciar e controlar o

processo do projeto.

Os experimentos anteriores descrevem os resultados da utilização de protótipos no sistema como um todo. Gilhooley [GIL86] apresenta o impacto de prototipação, isoladamente, nas diversas fases de desenvolvimento - Figura 2.3.

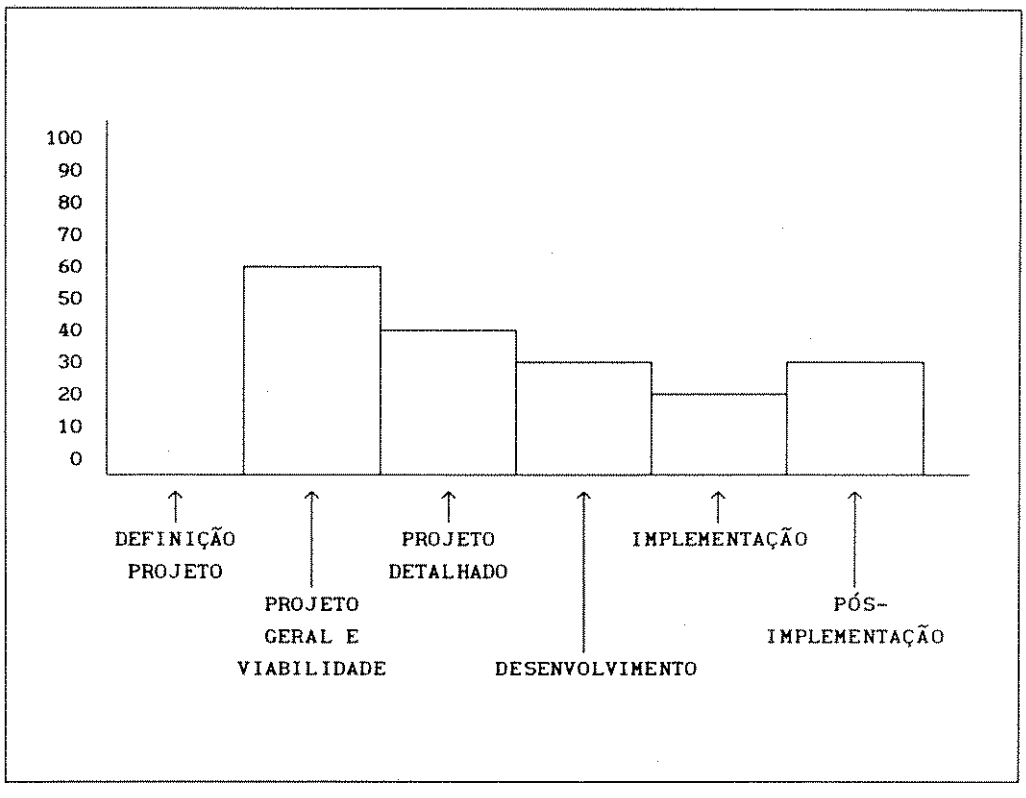


Figura 2.3. Impacto de Prototipação nas Fases de Desenvolvimento.

A Figura 2.3 representa, segundo Gilhooley, a porcentagem de envolvimento com técnicas de prototipação associada com cada fase do ciclo de vida. As porcentagens não se referem a uma medida específica mas são significativas para mostrar a importância relativa de prototipação em cada fase. Os próximos parágrafos justificam as porcentagens apresentadas ressaltando aspectos de aplicação de técnicas de prototipação.

- **Definição de Projeto (0%)** - envolve expressão de uma idéia. Como para a maioria dos projetos, a fase de definição do projeto é completada através do desenvolvimento de uma estratégia de informação, é pouco provável que o protótipo seja desenvolvido neste estágio.

- **Projeto Geral e Viabilidade (60%)** - nesta fase os projetistas definem e documentam as necessidades do usuário. Prototipação é aplicável somente para apresentar a solução do sistema ao usuário. Uma das vantagens é que o usuário pode interagir com o sistema a ser construído além de executar testes com o protótipo.

- **Projeto Detalhado (40%)** - nesta fase, o projeto geral é expandido ao ponto onde programação detalhada pode começar. Padrões envolvendo dados, rotinas comuns e interação entre módulos são finalizados. O protótipo auxilia a realçar as áreas em que estes padrões são requeridos. Neste contexto, prototipação pode contribuir na especificação de telas, consultas e relatórios a serem produzidos pelo aplicativo. Serve também como uma ferramenta de documentação da lógica de programas.

- **Desenvolvimento (30%)** - ocorre produção e teste de todos os aspectos do sistema. Até este ponto, o protótipo tem sido desenvolvido principalmente para funcionalidade. O protótipo providencia a base para codificar dentro de cada programa.

- **Implementação (20%)** - nesta fase ocorre o treinamento do usuário. Com prototipação, este treinamento pode começar a qualquer tempo após o Projeto Geral e Fase de Viabilidade.

- **Pós-implementação (30%)** - o objetivo é verificar se o sistema satisfaz os requisitos do usuário; o processo é simplificado com o Protótipo, que já foi aprovado pelo usuário.

2.3. Características desejáveis num ambiente de Prototipação.

Como pode ser observado nos itens anteriores, prototipação está muito mais associada a uma determinação existente entre o usuário e analista com o intuito de refinar os requisitos, do que a um conjunto de ferramentas complexas e custosas para sua implementação.

Apesar disso, a existência de um ambiente apropriado para o desenvolvimento de protótipos é altamente desejável. A seguir, são apresentadas algumas características que tais ambientes devem possuir para alcançar seus objetivos.

- **Rapidez na geração do protótipo** - Uma das premissas básicas em prototipação é a geração rápida do protótipo a um custo reduzido.

- **Facilidade da implementação de alterações** - Alterações são inevitáveis na utilização de protótipos. As ferramentas de prototipação disponíveis devem garantir que tais alterações possam ser rapidamente refletidas nos requisitos para a geração de um novo protótipo a ser novamente analisado pelo usuário.

- **Interface amigável** - O protótipo deve ter uma interface amigável, compreendendo preferencialmente uma interface gráfica que facilite a interação do usuário e a visualização do comportamento do sistema. Os relatórios e documentações emitidos pela ferramenta devem ser acessíveis ao usuário; ou seja, devem fazer uso de termos comuns ao ambiente definitivo do sistema.

- **Geração automática de código** - A geração automática de código, possibilitando sucessivos refinamentos até atingir um estágio bem próximo ao da implementação, é uma característica interessante pois minimiza o esforço necessário durante as fases posteriores ao do refinamento de requisitos.

- **Ambiente Integrado** - Existência de um ambiente integrado que garanta facilidades para edição das entradas do sistema, geração do protótipo, execução e geração de relatórios.

- **Protótipo Executável** - O protótipo deve ser passível de execução para permitir uma validação e análise das funções do sistema previamente definidas.

Estas caraterísticas se tornam mais críticas em Sistemas de Tempo Real (STR). Devido às rígidas restrições presentes em STR como precisão, segurança e confiabilidade [LUQ93], sua representação através de protótipos necessita uma atenção especial. O paradigma de implementação de STR leva à existência de um conjunto de tarefas que interagem e possuem restrições de tempo que devem ser cumpridas. Para coordenar a execução destas tarefas são utilizados Núcleos de Tempo Real. A construção de uma ferramenta para prototipar STR deve, portanto, contemplar os seguintes elementos :

- Presença de um Núcleo de Tempo Real similar ao do sistema definitivo que permita a interrupção da execução e análise do estado das tarefas.

- Criar mecanismos que permitam um acompanhamento preciso do instante da ocorrência de cada evento no sistema. Por exemplo, a utilização de um relógio de tempo real presente no núcleo.

- Permitir que as entradas (estímulos externos) fornecidas ao protótipo possam ser consumidas em instantes bem determinados de forma análoga aos eventos em STR.

Descrição detalhada de cada uma destas características, bem como suas implicações na implementação do prototipador desenvolvido neste trabalho, são apresentados nos próximos capítulos.

2.4. Vantagens e Desvantagens de Prototipação.

O principal objetivo de sistemas protótipos é assegurar o desenvolvimento eficiente de um sistema que cumpra todos os requisitos do usuário.

O sucesso da prototipação rápida depende da aplicação. Aplicações apropriadas são aquelas associadas a processos dinâmicos com extensivo uso de diálogos ou que requerem capacidade de tempo real. Por outro lado, as aplicações de orientação "batch", extremamente grandes, ou que envolvem pouca ou nenhuma interface não são boas candidatas à utilização de prototipação [KLI86].

Prototipação apresenta inúmeras **vantagens** quando usada corretamente :

- **Nível de Especificação** - Prototipação elimina a redundância de produção de vários níveis de especificações antes que qualquer desenvolvimento comece. As facilidades oferecidas por linguagens de 4ª geração e a flexibilidade em uma base de dados relacional permitem que o protótipo seja desenvolvido rapidamente.

- **Volume de Documentação** - Ao invés de produzir muita documentação para descrever como o sistema executará, prototipação permite que o usuário examine o sistema e ganhe experiência com o protótipo.

- **Processo de Revisão** - O protótipo permite que os usuários revisem e critiquem à medida que o sistema está sendo construído, o que provê uma facilidade de teste e validação dos requisitos. As alterações podem ser feitas fácil e rapidamente.

- **Manutenção do sistema** - Com linguagens de 4ª geração que requerem poucos procedimentos para desempenhar seleção e manipulação de dados e com a flexibilidade oferecida por uma Base de Dados relacional, a manutenção do sistema é simplificada.

- **Melhor comunicação** - um protótipo operacional elimina os problemas de comunicação entre o usuário e projetista e resulta em interpretações mais precisas da especificação do sistema. Possibilita a criação de um sentimento de grupo, um espírito de equipe entre usuário e analista [ALA86].

- **Redução de riscos de projeto** - prototipação reduz os riscos do projeto testando-se viabilidade técnica, econômica e organizacional do sistema [ALA86].

- **Redução de tempo** - o uso de prototipação conduz a uma redução no tempo humano gasto através de uma maior utilização computacional, permitindo uma redução do tempo gasto nas fases de análise e projeto.

- **Ferramenta de Aprendizagem** - um usuário com pouco contato com computadores pode ganhar uma boa experiência com um protótipo. Por outro lado, o analista aprende muito sobre o que o usuário necessita e deseja, além de aprender como projetar e desenvolver um sistema que atenda aos requisitos [DEA83].

- **Prototipação conduz ao certo** - auxilia a assegurar que o núcleo de um sistema está correto, isto é, apresenta os resultados desejados, antes de empregar todos os recursos no desenvolvimento do sistema completo [ALA84].

Algumas **desvantagens** também são apontadas e requerem um estudo cuidadoso para tentar minimizá-las.

- **Protótipos são difíceis de gerenciar e controlar** - devido à novidade e natureza de prototipação, existe uma falta de "know-how" para planejar, gerenciar e controlar sua utilização. Planejamento e controle são complexos devido : à forma de evolução do sistema, ao número de revisões

no protótipo e ao desconhecimento dos requisitos do usuário. A ausência de planejamento ou controle pode levar a uma redução da disciplina necessária para um gerenciamento apropriado, ou seja, atividades de teste ou documentação podem ser conduzidas superficialmente [ALA84].

- **Dificuldade de prototipar sistemas de informação grandes** - não está claro como um sistema de grande porte seria dividido para a proposta de prototipação. Na maioria dos casos, tempo e restrições de recursos de projeto determinam as limitações.

- **Potencial consumo de tempo e dinheiro** - um protótipo inapropriado deixaria a equipe de desenvolvimento sem observar o comportamento do sistema. Isto não deve desencorajar o uso de protótipo mas, sim, ensejar uma verificação quanto às questões de tempo de prototipação e como alcançar um balanço de simplicidade e realidade.

- **Gerenciamento de Tarefas** - O envolvimento dos usuários deve ser constante e consistente. Mesmo que os usuários se distanciem dos objetivos finais, o projetista deve manter o projeto no "caminho" correto sem, contudo, inibir a criatividade do usuário. Ao projetista compete a difícil tarefa de monitorar o processo de prototipação, mantendo um bom relacionamento com os usuários e, ao mesmo tempo, direcionando-os ao longo do processo da prototipação.

De forma resumida, prototipação é uma ferramenta poderosa para desenvolvimento de sistemas devido a sua alta produtividade, envolvimento do usuário e flexibilidade. Entretanto, pesquisas ainda necessitam ser feitas para torná-la uma técnica mais abrangente.

2.5. Classificação de Protótipos.

A partir dos conceitos de prototipação, foram criadas "Classificações de Protótipos" baseadas nos componentes envolvidos no processo de prototipação.

Devido ao grande número de pesquisadores envolvidos com o problema, inúmeras propostas têm sido apresentadas como a "solução" para a classificação de protótipos. Os próximos itens apresentam as classificações mais relevantes.

a) Classificação de Floyd.

Floyd [MAY87] distingue três classes de prototipação, quanto à finalidade :

- **Exploratória** - Importante nos estágios iniciais do desenvolvimento do software; tem a finalidade de facilitar a comunicação entre analista e usuário. O principal objetivo é auxiliar na especificação dos requisitos do sistema alvo.

- **Experimental** - Consiste da construção do protótipo de uma solução proposta para um problema particular. O protótipo pode conter todas as funções do sistema alvo, mas podem ser testadas apenas algumas funções de interesse particular.

- **Evolutiva** - Ocorre a evolução gradual do sistema com finalidade de atender os requisitos. Inúmeras versões de um mesmo sistema são geradas até que o sistema definitivo seja obtido.

b) Classificação de Law.

Law [MAY87] acrescentou dois tipos à classificação sugerida por Floyd:

- **Desempenho** - Também chamada de **Síntese**, é utilizada para indicar se o sistema será capaz de atender ao seu ambiente operacional. Auxilia a detectar combinações incompatíveis de hardware e software. É considerado um caso especial de prototipação Experimental.

- **Organizacional** - O protótipo é avaliado no local de trabalho do usuário, num dia normal de atividades, tendo como objetivo verificar se os requisitos do usuário serão atendidos. Este método também é considerado um caso particular do Experimental. Recomenda-se a sua utilização em projetos que implantarão o sistema em diversos locais e/ou quando o sistema é muito grande, com centenas de terminais.

c) Classificação de Mayhew/Dearnley.

Mayhew e Dearnley [MAY87] sugerem uma classificação denominada **Alternativa**, com fundamento nos componentes envolvidos no processo de prototipação : prototipador, usuário, software e hardware.

A Figura 2.4. mostra a representação gráfica das interações entre estes quatro componentes básicos de prototipação. Cada vértice da pirâmide representa um dos 4 principais elementos e as linhas de união representam os relacionamentos entre eles. A partir desta pirâmide, interpreta-se a classificação proposta por Law.

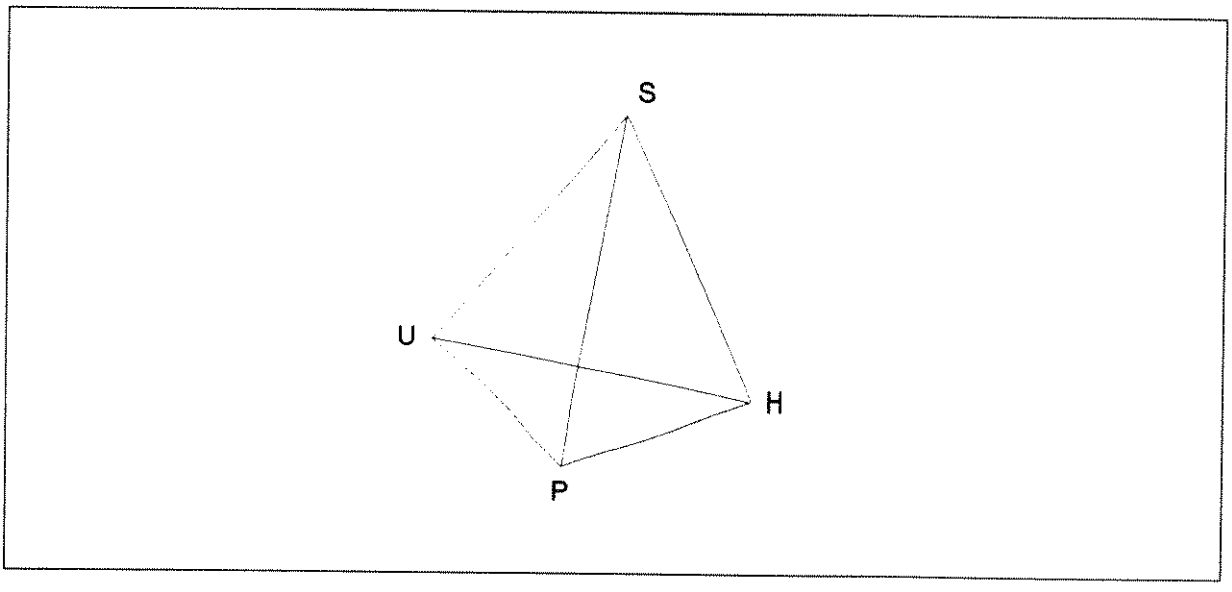


Figura 2.4. Componentes principais de Prototipação.

- **Exploratória** - Envolve o prototipador (P), usuário (U) e os componentes do software (S); concentra-se na comunicação entre prototipador e o usuário final.

- **Experimental** - possibilita determinar se a solução proposta é adequada ao problema particular. Os elementos envolvidos são prototipador (P), software (S) e hardware (H).

  A prototipação **Experimental** apresenta três subtipos, de acordo com a interação dos elementos considerada mais importante :

  . **Experimental** - O protótipo é construído para validar alguns elementos do projeto; os componentes envolvidos são prototipador (P) e software (S).

  . **Desempenho** - Se concentra na interação entre software (S) e hardware (H), como descrito na classificação de Law.

. **Hardware** - O protótipo é construído para auxiliar na seleção do hardware, envolvendo o prototipador (P) e hardware (H).

- **Organizacional** - Usado no ambiente destino, com a finalidade de clarificar os requisitos e implicações. Os três participantes principais são usuário (U), software (S) e hardware (H). De acordo com a análise de interação entre os componentes, identifica-se subtipos da prototipação Organizacional.

  . **Ergonômica** - Concentra-se na interação entre o usuário (U) e hardware (H); útil na avaliação da conveniência do hardware.

  . **Funcional** - Fundamenta-se na interação entre o usuário (U) e software (S); usado para examinar as facilidades providenciadas pelo sistema.

- **Evolutiva** - Os principais componentes envolvidos são software (S), usuário (U) e hardware (H). Note que os mesmos componentes do tipo Organizacional estão presentes; entretanto, o autor afirma que a Prototipação Evolutiva é simplesmente uma extensão da Organizacional com um período de tempo maior entre as versões.

d) Classificação de Canning.

Uma outra classificação a ser considerada, é a apresentada por Canning [MEL90], que aborda a prototipação quanto à sua finalidade.

- **Identificação de Necessidades do Usuário** - Corresponde à prototipação Exploratória de Floyd. O objetivo é reduzir os problemas de comunicação entre usuário e analista. O protótipo aprovado serve de "documentação viva" das necessidades reais de seu usuário.

- **Validação da Funcionalidade** - Visa testar a potencialidade de um sistema, em termos de componentes internos : utilitários, programas aplicativos e arquivos físicos utilizados pelo sistema. Compreende os aspectos de implementação de um software, em termos de capacidade de relacionamento, acoplamento, coesão e desempenho.

- **Construção de um sistema definitivo ou final** - ocorre quando parte ou o protótipo total é utilizado como o sistema final.

e) Classificação de Lee.

Lee [MEL90] classifica os protótipos de acordo com os seus métodos.

- **Descartável** - utilizado para especificar as necessidades e requisitos do usuário. É considerado um excelente método de descoberta e documentação lógica do sistema : entradas, saídas e transações básicas.

- **Fundamental** - é uma extensão do descartável; pelo processo Evolutivo torna-se o produto final. O protótipo, além de contribuir para a descoberta do "mundo real" do sistema, deve servir de base ou fundamento para a construção do sistema definitivo.

f) Classificação de Hekmatpour/Ince.

Hekmatpour e Ince [STE90] classificam prototipação em 3 categorias :

- **"Throw-away"** - usado para identificar requisitos, é "descartado" após a avaliação pelos projetistas e usuários. É criado para mostrar o comportamento do sistema ao usuário.

- **Incremental** - o projeto do sistema é especificado, as sub-tarefas são identificadas, e os protótipos são construídos, avaliados e modificados de acordo com o "feedback" do usuário. É importante salientar que o projeto permanece inalterado.

- **Evolutiva** - permite que o sistema evolua, ocorrendo uma alteração dinâmica. O protótipo inicial eventualmente evoluirá até o produto final a partir de detalhamento dos requisitos e implementação. Inicialmente se concentra nos requisitos funcionais; os requisitos de desempenho e interface são incorporados posteriormente [BAL90].

Uma classificação **alternativa** é apresentada considerando a proposta do protótipo [STE90] :

. **Funcional** - se concentra na função do sistema.
. **Interface** - está preocupado com a interface do usuário.

É importante salientar que um protótipo pode abranger mais do que um dos tipos aqui descritos. O fundamental é que o analista seja capaz de distinguir os objetivos de cada um destes tipos de protótipos.

Um dos resultados concretos deste trabalho consiste na ferramenta para prototipar sistemas de tempo real denominada ProTR. O prototipador ProTR segue a linha **Evolutiva** pois tem como objetivo gerar um código executável bem próximo do produto final à partir de refinamentos sucessivos realizados sobre as versões anteriores de protótipos gerados. No momento em que analista e usuários estiverem satisfeitos, o produto resultante do esforço de prototipação pode ser utilizado como base para o desenvolvimento do sistema definitivo.

Capítulo 3. TÉCNICAS E FERRAMENTAS DE PROTOTIPAÇÃO.

Prototipação é muito mais que o uso de um conjunto de ferramentas para reduzir o tempo de desenvolvimento do sistema. Poderia ser definida como a criação de algum sistema funcional ou subsistema durante as primeiras fases do processo de desenvolvimento, tendo como objetivo básico criar uma oportunidade para que o analista e o usuário possam exercitar uma versão (geralmente limitada) de uma solução proposta, antes de comprometer recursos com uma implementação completa do sistema.

Esta definição permite concluir que prototipação é muito mais uma abordagem filosófica aplicada ao ciclo de desenvolvimento do que um conjunto de ferramentas que devem ser empregadas. Entretanto, a utilização de técnicas e ferramentas apropriadas permitem acelerar o processo de prototipação.

Neste capítulo serão apresentadas técnicas apropriadas para ambientes de prototipação bem como ferramentas associadas à aplicação de cada técnica.

3.1. Técnicas de Prototipação.

Podemos realçar as áreas durante o processo de projeto, nas quais alguma forma de prototipação pode ser aplicada [LIP86] :

1. Validação do modelo de dados lógicos.
2. Validação do modelo de processos.
3. Apresentação de exemplos de telas "on line".
4. Criação de exemplos de relatórios.
5. Criação de programas "stub" durante o desenvolvimento estruturado de programas.
6. Estabelecimento de um modelo para a "prova de conceitos".
7. Simulação de "carga" em redes de comunicação.

8. Tradução de modelos de dados lógicos em modelos físicos e simulação de acessos à base de dados.

Estas áreas podem ser agrupadas em três grupos básicos, cada um deles descrevendo uma filosofia que orienta a aplicação de técnicas de prototipação.

3.1.1. Estimativas e Medidas de Desempenho.

Neste grupo estão presentes os itens 7 e 8; a utilização de técnicas de estimativa de desempenho ocorre, geralmente, durante dois estágios do processo de desenvolvimento :

a) Na fase de projeto é utilizado para verificar se o projeto exibe alguma característica que possa indicar uma falha básica no atendimento aos requisitos fundamentais de desempenho.
   Os protótipos desenvolvidos por estas técnicas são na realidade modelos matemáticos dos componentes do sistema. Como um modelo matemático é de uso restrito como sistema definitivo, normalmente o analista o descarta após alcançar uma melhor compreensão dos requisitos do sistema.

b) Após uma porção significativa do sistema ter sido "codificada", é gerada uma "carga" artificial que alimenta o sistema, tornando possível analisar seu comportamento.
   Os protótipos desenvolvidos por estas técnicas não são protótipos do sistema mas sim, simulações daquela porção do ambiente sendo desenvolvida fora do sistema. Estas simulações permitem que alguma forma de "carga" seja aplicada ao sistema para análise e validação dos requisitos. É necessário reconhecer que "cargas" podem ser colocadas contra processos criados por prototipação, mas as "cargas" propriamente ditas não podem ser classificadas como protótipos.

Como pode ser visto, a aplicação destas duas técnicas se complementam, sendo que a primeira estima o desempenho projetado do sistema e a segunda verifica o desempenho atual.

3.1.2. Produção de Código.

Na produção de código estão presentes as técnicas que auxiliam o projeto, desenvolvimento e codificação de "programas" através de linguagem "procedimental" ou "não-procedimental" (linguagem de 4ª geração) (itens 3, 4 e 5 acima citados).

As técnicas presentes nesta categoria podem ser agrupadas em :

- Utilização do conceito de refinamento passo-a-passo na criação de código procedimental e não-procedimental.

- Criação de telas para validação pelo usuário.

- Substituição de código procedimental (COBOL, PL/1, etc) por código não-procedimental (Mantis, Enable, ADS/O).

a) Refinamento Passo-a-Passo.

Refinamento passo-a-passo é uma técnica que permite que os detalhes do desenvolvimento do programa sejam analisados ao final da implementação. Isto permite ao programador concentrar-se em todos os fluxos de controle e projeto do processo, ou seja, nos problemas conceituais (lógicos), antes de dispender recursos no desenvolvimento de detalhes de código. Entretanto, deve-se utilizar recursos para desenvolver programas "stubs" que permitem que esta técnica seja aplicada.

b) Telas e Formatação de Relatórios.

A criação de telas para validação pelo usuário compreendem geração de "hard copy", mapas de tela e fluxos de transação (isto é, qual tela é ativada quando a tecla **Fi** é pressionada). As ferramentas que fornecem este recurso normalmente não oferecem qualquer aspecto funcional às telas apresentadas, ou seja, os usuários simplesmente concordam com o formato da tela e que as informações ali apresentadas estão corretas.

Trabalhando com os usuários, o projetista compreende melhor o problema e otimiza as interfaces já construídas, produzindo um conjunto de interfaces similares às do sistema final.

Inúmeros trabalhos têm explorado essa linha [ALA86], [RYA85], [JON86], [MAS82], [BEN89], atingindo resultados promissores, principalmente quando aplicados em sistemas comerciais.

c) Linguagens Não-Procedimentais (4ª geração).

Pode-se construir um protótipo de alto nível, utilizando uma linguagem não-procedimental de 4ª geração, baseada na premissa de "O que fazer" ao invés de "Como fazer". Uma vez obtido o protótipo, é possível que seja necessário realizar um trabalho de reprojeto em uma linguagem de 3ª geração antes da implementação [KLI86].

As linguagens de 4ª geração apresentam produtividade em média 10 vezes maior que linguagens de programação de 3ª geração [GIL86]. Linguagens de 4ª geração são usualmente linguagens interpretativas [GIL87], ou seja, o processador interpreta cada procedimento da linguagem fonte na execução, ao invés de passar pela fase de compilação antes da execução, como ocorre com linguagens de 3ª geração. As linguagens interpretativas de 4ª geração apresentam flexibilidade quando alterações são necessárias.

Tipicamente, as linguagens de 4ª geração requerem poucas declarações para desempenhar seleção múltipla, manipulação e apresentação de dados.

A incorporação de uma base de dados relacional e um dicionário de dados em sistemas escritos nestas linguagens de programação simplificam a manutenção. A base de dados e o dicionário de dados provêm independência dos dados e suportam a sua manipulação em contextos diferentes do da aplicação em desenvolvimento [GIL85].

As características mais importantes que uma linguagem de programação deve possuir com a finalidade de ser empregada em prototipação são [POM91] :

- interpretação ao invés de compilação.
- conceitos de abstração que permitam a construção de componentes reusáveis.
- incorporadas ao ambiente de desenvolvimento.

Como exemplo de protótipo utilizando linguagem de 4ª geração, podemos citar a linguagem SPEC [BER90], cuja finalidade é expressar as especificações da interface para sistemas distribuídos com restrições de tempo real.

A substituição de código procedimental por não-procedimental pode acontecer em uma das seguintes formas :

. Substituição completa, tal que não existe intenção de usar código procedimental e o não-procedimental serve como o processo de produção.

. O uso de linguagem não-procedimental com a finalidade de verificar o projeto do sistema, traduzindo-o em código procedimental antes da fase de produção do sistema (isto é, cria-se um protótipo descartável).

. Uso de linguagem não-procedimental para criar o processo e possível uso da mesma para produção, a menos que fortes argumentos revelem que o código não-procedimental deveria ser convertido para código procedimental.

3.1.3. Validação de Projeto.

Prototipação é útil em duas áreas diferentes de validação de projeto: o desenvolvimento de um modelo de "validação do conceito" do sistema, e a validação de modelos de dados (itens 1, 2 e 6 citados acima).

Um modelo de "validação do conceito" é usado para mostrar se o projeto proposto é viável. Seu enfoque é similar ao das simulações matemáticas descritas em "Estimativas e Medidas de Desempenho", exceto que estes modelos simulam porções da interface do usuário do sistema de modo que eles possam ser validados. Tais protótipos são normalmente criados na fase de projeto e, portanto, não são mantidos. Esta é uma área em que protótipo "Throw-away" é útil.

A validação de um modelo de dados por prototipação é feita indiretamente. O modelo de dados é implementado com um Gerenciador de Banco de Dados que providencia a capacidade para a implementação de um modelo de dados em relações físicas. Consultas são então realizadas para testar a estrutura do modelo para verificar se contém os dados requeridos.

3.2. Ferramentas de Prototipação.

A variedade de ferramentas que podem ser usadas em ambientes de prototipação tem sido limitada, mas está aumentando rapidamente em resposta à demanda na área. As ferramentas podem ser agrupadas como produtos que fornecem auxílio em uma das diversas áreas :

- Sistema de Gerenciamento de Base de Dados.
- Modelamento de Dados e Processos.
- Simulação estatística.

3.2.1. Sistema de Gerenciamento de Base de Dados.

O ambiente de prototipação exige um Gerenciador de Base de Dados que é flexível sob dois aspectos distintos. Deve suportar um modelo de dados lógico versátil; contudo, deve exibir um desempenho suficiente para ser eficiente em um ambiente de produção. Deve ser fácil de usar de modo que estas estruturas possam ser modificadas rápida e facilmente em resposta a uma alteração no projeto. Tais Gerenciadores de Banco de Dados estão começando a aparecer; como exemplo ORACLE da Oracle, DB2 da IBM, e CA-Universe da CA.

3.2.2. Modelamento de Dados e Processos.

Modelagem de Processo tem sido usada há muitos anos e diversas ferramentas têm sido criadas; como exemplo, gerador automático de fluxogramas. Entretanto, o ambiente de prototipação requer que os dados e ferramentas de modelagem de processo estejam integrados. Este requisito tem sido parcialmente atendido por diversos pacotes (tais como AUTO-MATE da LBMS, PRISM da Deltacom e Excelerator da Intech).

Para modelagem de dados, pode-se citar : Design Manager da MPS, ADAM da Database Design Group. Contudo, existe um longo caminho a percorrer antes que surja um pacote completo em que modelagem de processo, modelagem de dados, dicionário e Gerenciador de Banco de dados estejam integrados.

3.2.3. Simulação Estatística.

Concentra-se em ferramentas que podem simular um projeto ou podem ser usadas para a geração e prototipação de saídas de um sistema. Exemplos de ferramentas com estas características : GPSS da IBM, Simula do Instituto SAS.

Uma ferramenta que tem sido pesquisada por Stavely [STA82] refere-se a projetos que descrevem componentes do sistema na forma de "modelos de estado-finito". No modelo de estado finito, um componente é modelado em termos de um número finito de classes de equivalência de estados. Este modelo apresenta um comportamento similar ao da eventual implementação.

Outras técnicas de modelagem consideradas são métodos Probabilísticos e Estocásticos para descrever aspectos quantitativos do sistema como temporização, desempenho, recursos, etc.

Este modelo contendo semânticas bem definidas e detalhes operacionais, pode ser "interpretado" para produzir um "Projeto Executável", que pode ser visto como um "feedback" do projeto proposto. Este tipo de protótipo poderia ser útil no desenvolvimento de sistemas concorrentes e sistemas de tempo real.

A maior dificuldade é que o projeto apresenta apenas um modelo da organização interna em um vocabulário voltado ao projetista. Como solução, deveria ser providenciado um protótipo com uma interface acessível ao usuário, permitindo a utilização do "Projeto Executável" como uma técnica para prototipação de um sistema completo.

3.3. Sistemas de Prototipação.

As ferramentas de prototipação são comumente aplicadas em conjunto, de forma a produzir um sistema unificado para prototipação. Neste contexto é comum encontrar na literatura sistemas completos que implementam desde a modelagem dos processos/dados até a geração de código executável.

Os próximos itens exploram esses sistemas, selecionando aqueles que de alguma forma podem ser aplicados a ambientes de tempo real.

3.3.1. Prototipação de Requisitos Executáveis.

Através da análise de requisitos podemos chegar a um protótipo do sistema. Naturalmente, os requisitos devem estar formalmente descritos para se alcançar o protótipo. A forma de atingir tal formalismo é através de uma linguagem de especificação. Davis [DAV82] apresenta uma ferramenta denominada "Feature Simulator" que "executa" especificações de requisitos para sistemas de tempo real e habilita a interação do usuário com o produto simulado na fase de especificação de requisitos.

"Feature Simulator" foi desenvolvida como parte do "Sistema de Processamento de Requisitos" (RPS), que é um conjunto de ferramentas com a proposta de automatizar a tarefa de especificação de requisitos.

O sistema RPS (Figura 3.1.) é composto dos seguintes elementos :

- **Processador de Linguagem de Requisitos (RLP)** : examina os requisitos e os traduz para um modelo de máquina de estados finitos armazenado em uma Base de Dados.

- **Ferramenta de Definição de Linguagem (LDT)** : esta ferramenta é usada uma vez para cada nova aplicação permitindo definição das características sintáticas e semânticas da linguagem e o vocabulário específico da aplicação.

- **Ferramenta Padronizadora do Projeto (PST)** : permite estabelecer formatos de documentos.

- **Simulador de Característica** : realiza a simulação da especificação.

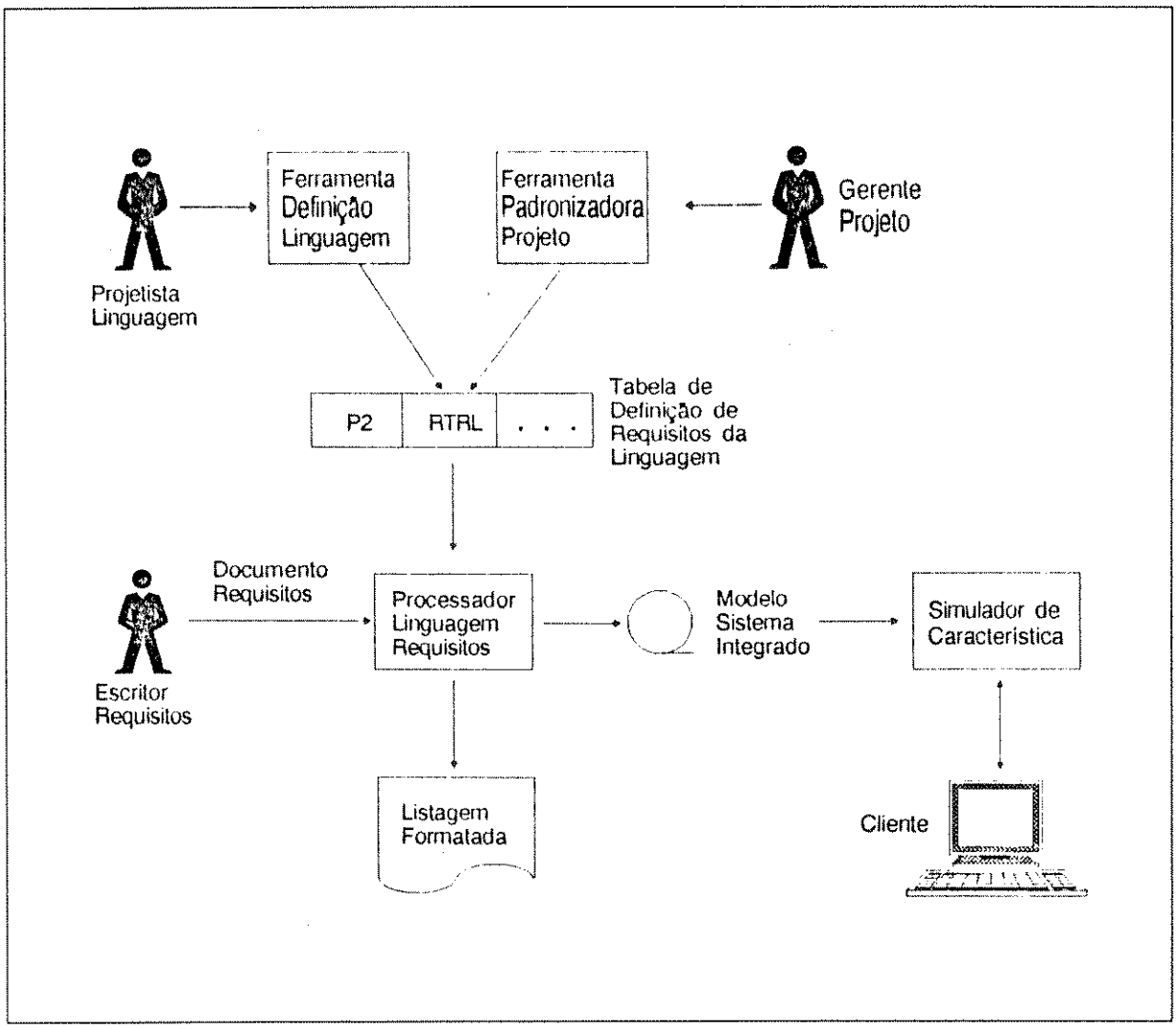


Figura 3.1 : Sistema de Processamento de Requisitos (RPS).

3.3.2. Prototipação utilizando Estrutura de Transição de Estados.

Lee [LEE85] apresenta uma forma de especificação executável como método para prototipação rápida, usando a estrutura de transição de estados. Esta estrutura é uma generalização dos modelos de Rede de Petri. As especificações são escritas em Prolog.

Com a estrutura de transição de estados, o comportamento do sistema é especificado por regras contendo pré e pós-condições para cada transição.

O sistema de transição de estados pode ser definido pela quádrupla **<Estados,Entradas,F,S0>**, que consiste de um conjunto de estados, um conjunto de entradas, um conjunto de funções de transição **F : (Estado x Entrada) ---> Estado** e um estado inicial **S0**.

Cada transição tem um nome, um conjunto de pré-condições que habilitam a transição, um conjunto de pós-condições que influirão nas transições seguintes.

Para qualquer tipo de sistema de transição de estado, a especificação é executada a partir da "ativação" de transições. O Predicado Prolog **"fire" (Estado Corrente S1, Transição T, Próximo Estado S2)**, é interpretado como "no estado corrente" **S1**, transição **T** está habilitada para ativar, resultando no Estado **S2**.

A estrutura de Transição de Estados é usada para dar suporte a "Prototipação algorítmica", produzindo rápidas implementações, que iriam requerer quantidade significativa de programação se escrita em outras linguagens.

3.3.3. Prototipação utilizando Redes de Petri de Alto Nível.

Bruno e Marchetto [BRU85] [BRU86] apresentam uma técnica para prototipação onde extensões de Rede de Petri, denominadas "Process Translatable Nets" - "PROT NETS", provêm estrutura para integrar as principais fases de desenvolvimento do software : formalização de especificações; avaliação, simulação e validação do modelo; tradução automática para estruturas de programa.

A tradução automática para estruturas de programa facilita a prototipação rápida do sistema : o esqueleto do programa derivado da rede pode ser refinado com a finalidade de obter um protótipo para demonstração ao usuário.

Rede de Petri é um poderoso e rigoroso formalismo para descrever sistemas distribuídos, onde os aspectos de concorrência e sincronização entre subsistemas são fundamentais.

A principal característica destas Redes é que suas notações e semânticas permitem que a implementação de processos e suas sincronizações sejam geradas automaticamente, o que determina "Process-Translatable Petri Nets" - PROT NETS.

PROT NETS apoiam o caminho operacional para especificação, ou seja, produzem o modelo executável do sistema e facilitam a tarefa de modelagem, pois as sincronizações são especificadas usando uma notação gráfica na forma estruturada top-down, e o comportamento de cada processo pode ser extraído da Rede como uma visão separada do sistema. As Redes obtidas podem ser traduzidas em estruturas de programa ADA considerando processos e sincronizações.

A Figura 3.2 apresenta o ciclo de prototipação rápida baseada em PROT NETS. Os ítens a seguir realçam pontos importantes nesse ciclo.

- **Especificação** : "PROT NETS" apresenta uma especificação operacional do sistema baseado nos conceitos de "processo" e "sincronização".

- **Validação ou Análise Estrutural** : As Redes podem ser analisadas para determinar propriedades tais como : Limitação, Segurança, "Liveness", etc.

- **Avaliação de Desempenho** : Pode-se verificar aspectos críticos do sistema introduzindo "Temporização" na Rede.

- **Prototipação** : Um esqueleto de programa pode ser derivado automaticamente a partir da "PROT NET". A funcionalidade prevista do sistema pode ser examinada no Protótipo, evitando erros estruturais em estágios subsequentes de implementação. Este esqueleto do programa contém definições de processos e de suas sincronizações escritas em linguagem de programação. As estruturas de programa são a base para obtenção do Protótipo; introduz-se algum tipo de animação para estabelecer ligação entre a natureza abstrata da Rede de Petri e a aplicação modelada.

- **Implementação** : O protótipo obtido pode ser usado como uma aproximação da Implementação final.

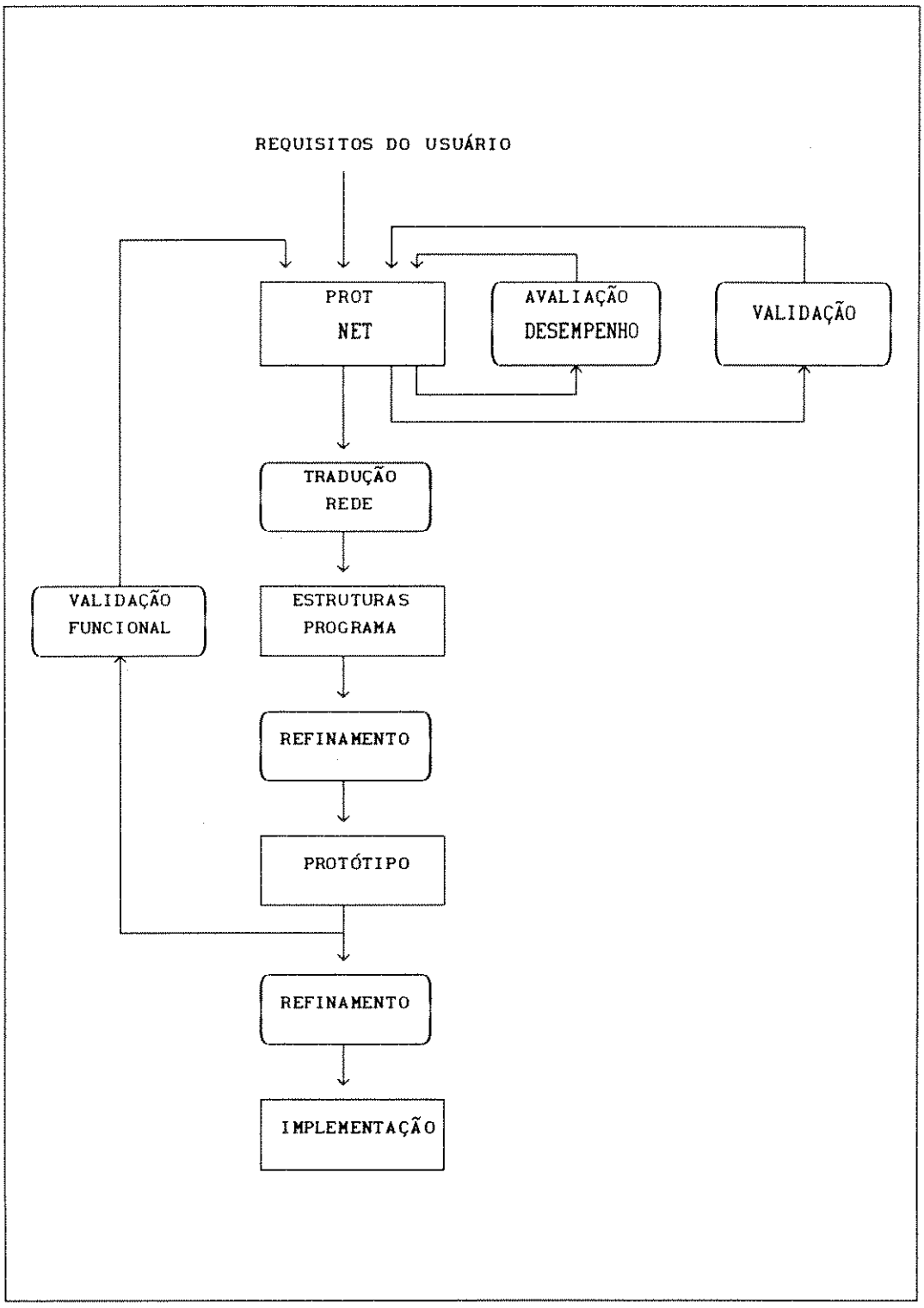


Figura 3.2 : Ciclo de Prototipação baseada em PROT NETS

3.3.4. Prototipação com PSDL - Linguagem de Descrição de Sistema Protótipo.

Esta técnica de prototipação segue a linha de construção de protótipo para sistema de tempo real que combina um modelo computacional para sistemas de tempo real com uma linguagem de prototipação PSDL - Linguagem de Descrição de Sistema Protótipo [LUQ88] [LUQ88a] [LUQ88b] [LUQ88c] [LUQ89] [LUQ90] [LUQ91].

Nesta linha, o ciclo de vida de software tradicional é substituído por um ciclo de vida com duas fases : prototipação rápida e geração automática de programas. Geração completamente automática de programas a partir de especificações de alto nível não é viável atualmente, mas geração automática de protótipos é possível.

Os benefícios de prototipação dependem da habilidade de modificar o comportamento do protótipo com menos esforço que o necessário para modificar o sistema definitivo. "Computer-aided prototyping" e "Object-based prototyping" provêm soluções para este problema. "Computer-aided prototyping" provê auxílio mecânico e "Object-based prototyping" provê simplicidade conceitual para tratar as complexas estruturas necessárias em prototipação automática.

"Computer-aided prototyping" melhora a eficiência do desenvolvimento introduzindo ferramentas de software que auxiliam o projetista a construir e executar o protótipo.

"Object-based prototyping" é baseada na abstração de dados. Objetos servem como uma base para projeto e implementação; provêm componentes para reuso de código, execução paralela e controle de versão.

A Figura 3.3 ilustra os principais passos do ciclo de prototipação :

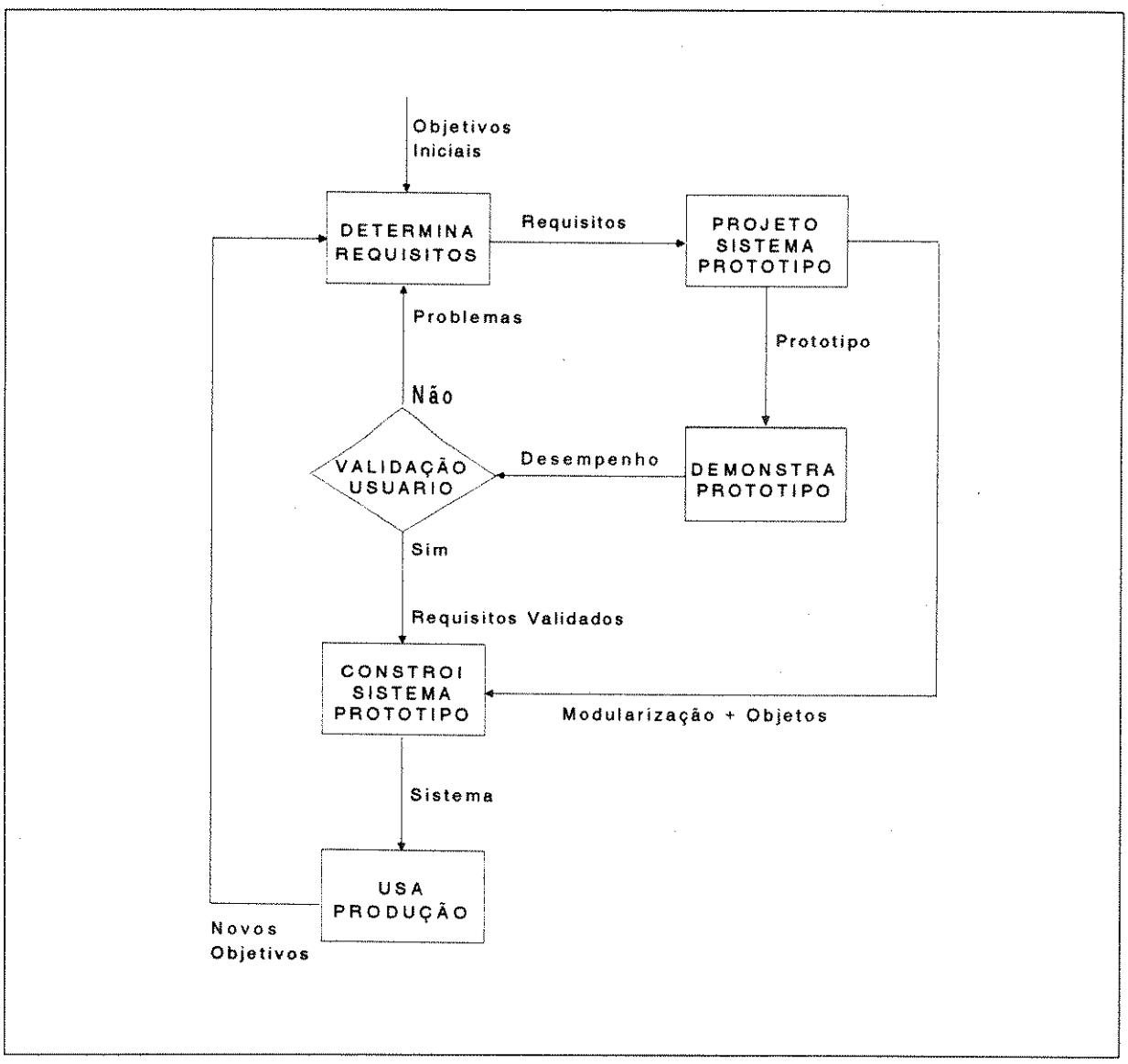


Figura 3.3. Ciclo de Prototipação.

PSDL foi projetada para servir como uma linguagem de prototipação executável a nível de projeto ou especificação, e tem características que atendem sistemas de tempo real.

PSDL e seu método de prototipação foram projetados para uso em um ambiente contendo um sistema de gerenciamento de componentes reusáveis, editor de sintaxe dirigida com capacidades gráficas, base de dados do projeto e um sistema de suporte à decisão.

A linguagem PSDL provê uma notação gráfica para diagrama de fluxo de dados com extensões de controle não-procedimental e restrições de temporização. Uma forma sintática descreve estas restrições e outros atributos para especificar um protótipo.

A descrição resultante é isenta de detalhes a nível de programação, em contraste aos protótipos construídos com linguagem de programação.

Um conjunto integrado de ferramentas de software, "Computer-aided prototyping system - CAPS", é utilizado para auxiliar na prototipação de sistemas de software complexos. Os principais componentes de CAPS são :

- Linguagem de Descrição de Sistema Protótipo - PSDL.

- Interfaces gráficas para acelerar a entrada e evitar erros de sintaxe.

- Um sistema de suporte à execução para acompanhar o comportamento do protótipo e efetuar análises estáticas.

- Um sistema para gerenciar componentes de software reusável e dados.

- Uma base de software para armazenar componentes reusáveis.

- Uma base de dados para armazenar o projeto protótipo.

PSDL provê abstrações adequadas para descrever sistemas grandes e restrições de tempo real. O projeto do protótipo é baseado em funções abstratas, dados abstratos e controle abstrato. Esta visão de alto nível enfatiza a configuração total a cada nível, sem se aprofundar em detalhes a nível de implementação.

A especificação de um protótipo escrito em PSDL provê uma documentação formal para o sistema, que é um projeto estruturado hierarquicamente com especificações de todos os componentes e interconexões entre eles.

Os principais atributos que um protótipo deve demonstrar ao usuário usualmente aparecem em um subsistema crítico; logo, deve-se criar um esqueleto do sistema com o qual o subsistema prototipado deve interagir.

Este método de prototipação associado ao PSDL resulta em um protótipo hierarquicamente estruturado. Desta maneira, provê-se uma "especificação caixa-preta" para cada componente, em adição à implementação.

Acredita-se que simulação do ambiente de software é parte essencial de prototipação rápida e que qualquer linguagem para prototipação de sistemas em tempo real deve suportar a construção de tais simulações. O protótipo é avaliado executando-o através de um pré-processador PSDL criado com um exemplo de dados de entrada usando o interpretador e depurador PSDL nessa operação.

3.3.5. Protótipos de Sistema de Gerenciamento de Informação em "ADA".

O método estruturado [BUR86] faz uso de diagramas "data-flow" que consiste de uma metodologia eficiente na fase de especificação do sistema. Este diagrama tem como suporte de documentação a ferramenta "Dicionário de Dados", que permite descrever em detalhes os elementos do DFD - Diagrama de Fluxos de Dados, metodologia de Especificação de Sistemas, proposta por Yourdon-DeMarco [DEM78].

A exatidão do DFD, bem como a consistência do dicionário de dados, podem ser verificados por diversas técnicas sendo uma delas a utilização da linguagem de descrição do problema (PSL) que, além de consistência, gera diversos relatórios sobre o DFD.

O DFD consiste de uma valiosa representação para o usuário final pois conserva a concorrência do sistema de informação. Devido a esta concorrência, a abordagem tradicional de transformação de DFD em programas sequenciais incorre em consumo de tempo, despesas, estando propenso a erros.

A linguagem de programação ADA, com ênfase em modularidade e interfaces de módulos bem definidas, é indicada como apropriada para o desenvolvimento de prototipação dos sistemas de gerenciamento de informação.

O protótipo é um modelo de trabalho usado na fase de análise para verificação dos requisitos do sistema. Também auxilia na exploração das diversas soluções para problemas técnicos e nas descobertas de erros de projeto no sistema proposto.

A técnica utiliza os seguintes passos para a obtenção do protótipo:

1- Constrói definições de dados no dicionário de dados.

2- Codifica a tarefa de controle do usuário.

3- A partir do DFD, instancia todos os fluxos de dados.

4- Constrói as tarefas de controle de arquivos.

5- Constrói uma tarefa gerente para todos os processos de sistema de Gerenciamento de Informação.

6- Constrói e testa transformações apropriadas para cada processo.

Seguindo este procedimento, um protótipo pode ser construído rapidamente. É importante acrescentar que este protótipo pode ser transformado no sistema de informação desejado, através da inclusão de transformações.

Burns e Kirkham concluem que o DFD tem-se mostrado eficiente na modelagem de sistemas de gerenciamento de informação apresentando seus requisitos e estrutura. A linguagem ADA tem demonstrado ser consistente para a implementação de protótipos, complementando a estratégia da análise estruturada.

3.3.6. Executando Especificações de Análise Estruturada para Tempo Real "Teamwork/ES".

A proposta do sistema "Teamwork/ES (Teamwork/Executable Specification)" consiste em permitir a execução de especificações de análise estruturada para tempo real em uma maneira gráfica interativa e intuitiva [BLU88].

Consiste basicamente de uma ferramenta CASE, que suporta as estratégias de análise estruturada para tempo real de Ward-Mellor [WAR86].

"Teamwork/ES" desempenha somente a execução simbólica, isto é, as marcas não possuem valores de dados, e a atividade é baseada na estrutura e interação entre os diversos objetos do diagrama. Este tipo de execução concentra-se no comportamento do sistema, não associa valores aos fluxos. Este sistema também pode desempenhar detecção de "deadlock" e teste de alcançabilidade dinâmica, permitindo ao analista estabelecer recursos potenciais de comportamento no sistema modelado.

A especificação pode ser executada em qualquer nível do Diagrama de Fluxo de Dados (DFD) hierárquico. Portanto, modelos incompletos podem ser avaliados dinamicamente antes do sistema total ter sido modelado.

Os objetos que participam da execução são separados dos que estão na tela. Nem todos os objetos abertos na tela necessitam participar da execução, e nem todos os envolvidos na execução necessitam estar na tela. Por exemplo, o modelo total pode ser executado, mas somente o diagrama-contexto, observado (execução caixa-preta).

O painel de controle da execução conduz a interação entre o usuário e a execução do sistema (Figura 3.4).

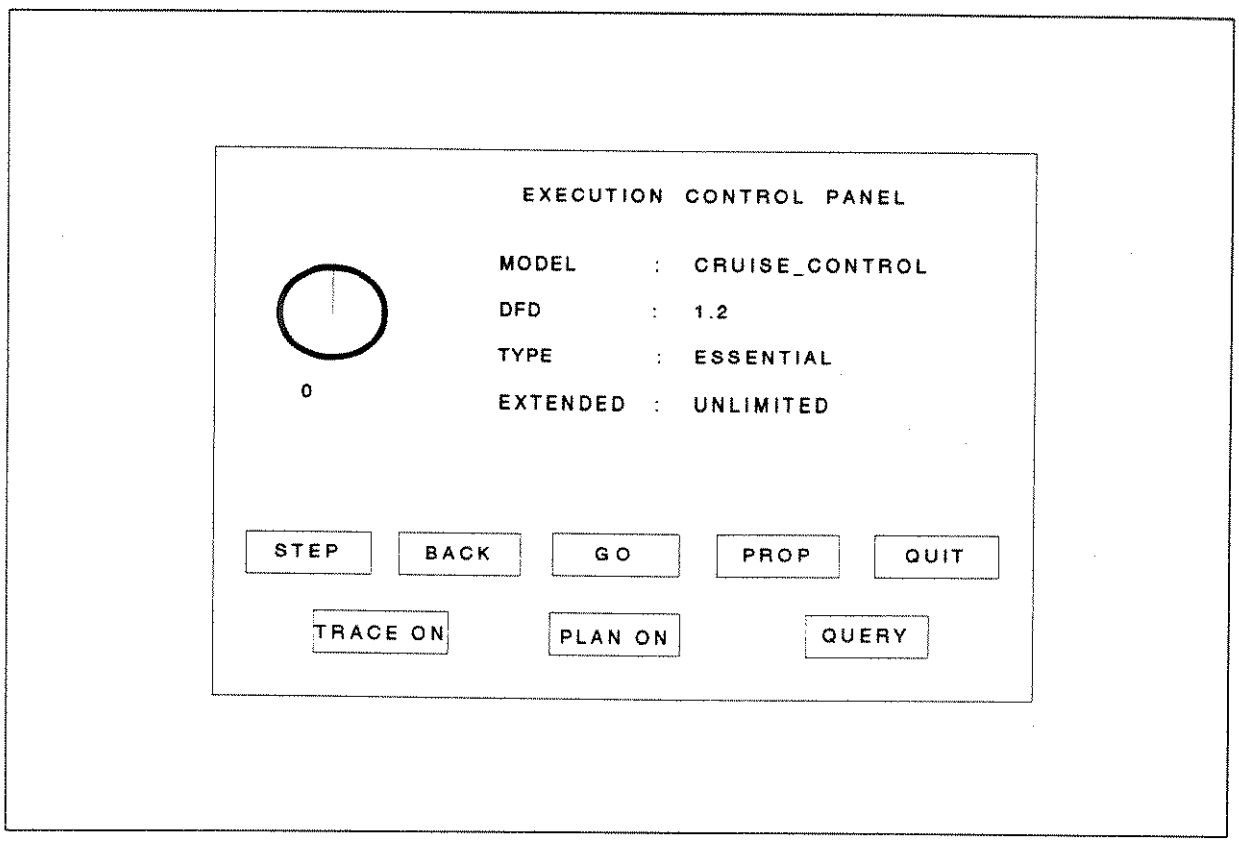


Figura 3.4. Painel de Controle da Execução.

Existe o modo de execução em "batch", isto é, sem intervenção do usuário. Isto é realizado com um **plano de execução** que consiste de um arquivo com um conjunto de vetores de entrada que especificam quando e onde as marcas serão colocadas ou removidas. Esta forma de execução é importante em modelos grandes que requerem casos de teste complexos, ou um número grande deles.

Um benefício deste sistema é que o esforço de prototipação rápida também é direcionado na criação da especificação do sistema. Entretanto, duas restrições podem ser levantadas em relação à sua utilização [COO89].

- Os relatórios são muito técnicos para serem avaliados pelo cliente, demandando um esforço extra para esclarecer as propriedades da especificação ao cliente.

- Como os fluxos não possuem dados na simulação, o sistema não pode determinar, quando da ativação de uma transformação de dados, quais eventos de saída mutuamente exclusivos devem ser ativados.

O sistema ProTR implementado neste trabalho não possui estas restrições apresentando, desta forma, um resultado mais real na simulação do sistema desejado.

3.3.7. Ferramenta Gráfica para Prototipação de Sistemas de Tempo Real.

A proposta da ferramenta é auxiliar na prototipação de sistemas de tempo real. Permite que o projetista modele e execute o sistema utilizando a metodologia de Ward & Mellor - Desenvolvimento Estruturado para Sistemas de Tempo Real [COO90]. A ferramenta implementa a estratégia de execução funcional em que os dados são passados e as especificações dos processos são executadas.

A execução do protótipo baseia-se nas regras da metodologia de Ward & Mellor; pode ser visualizada a partir da posição das marcas. O posicionamento de uma marca em um elemento representa uma atividade potencial. Uma marca em um fluxo indica que o fluxo está transportando dado.

As transformações de dados e certos terminadores podem ter descrições de processo associadas. Para esta finalidade, existe o Editor de Processo, cuja sintaxe é baseada na da linguagem Smalltalk com algumas alterações.

A Figura 3.5 apresenta uma transformação de dado com a descrição de seu processo correspondente.

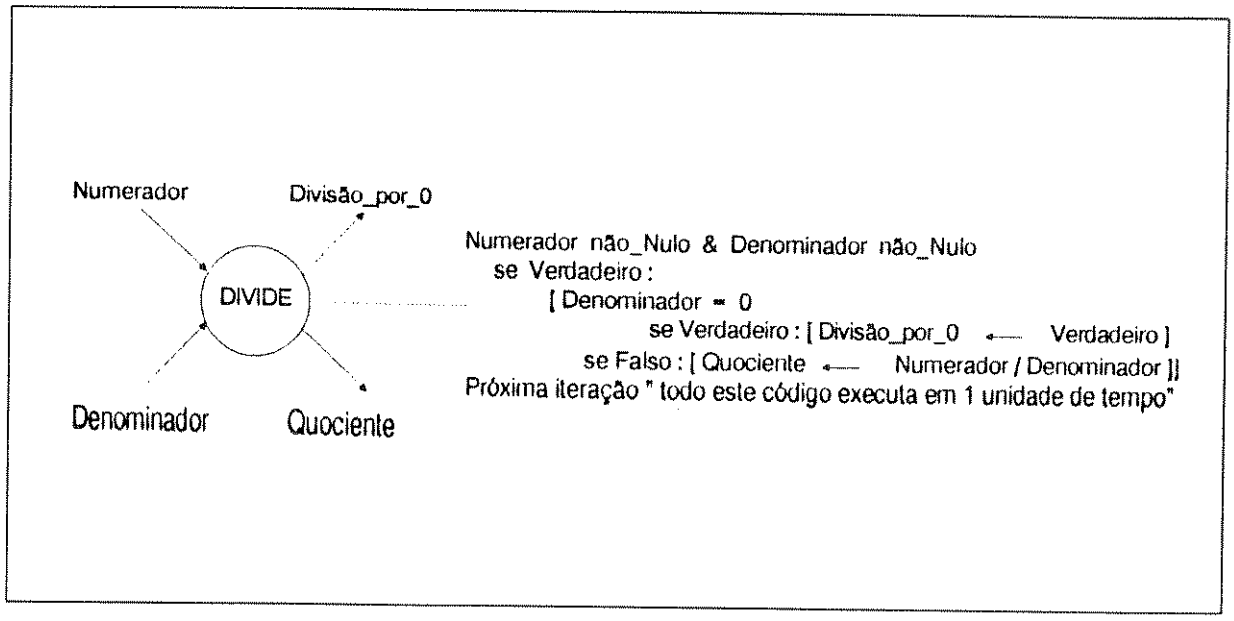


Figura 3.5. Transformação de dado e sua descrição de processo.

A natureza gráfica desta feramenta facilita o projeto rápido de sistema e as características de simulação proporcionam um "feedback" ao projetista com relação a funcionalidade e coordenação do sistema.

Neste protótipo, as descrições das transformações não são utilizadas diretamente na implementação do sistema definitivo; apesar disso, a ferramenta tem sido utilizada em sistemas de médio porte com sucesso.

3.3.8. Prototipação utilizando Telas Pictóricas.

Um grupo da Loughborough University of Technology tem usado animação computacional para apresentar a interação do sistema com o ambiente [COO89]. Esta comunicação entre o sistema e o cliente, realizada por uma abordagem de fácil compreensão, é considerada um aspecto "chave" para uma prototipação rápida bem sucedida.

O ambiente do sistema é animado com a utilização do pacote de simulação "SIMSCRIPT II.5" tendo como objetivo definir sequências padrão de operação e identificar a presença de condições inseguras.

O sistema usado para apresentar a ferramenta é parte de uma instalação química usada para comprimir Nitrogênio ou Hidrogênio com objetivo de utilizá-los em reações químicas subsequentes. Na animação deste exemplo, pode-se acompanhar a execução observando o comportamento a cada instante através dos esquemas gráficos. Por exemplo, se um compressor está em execução sem isolamento e com válvulas de ventilação fechadas, pode explodir. Este processo permite que resultados como inicialização da instalação, execução normal e emergência sejam explorados.

A Figura 3.6 mostra telas de animação permitindo a visualização do sistema em estados **ativado** e **falha segura**.

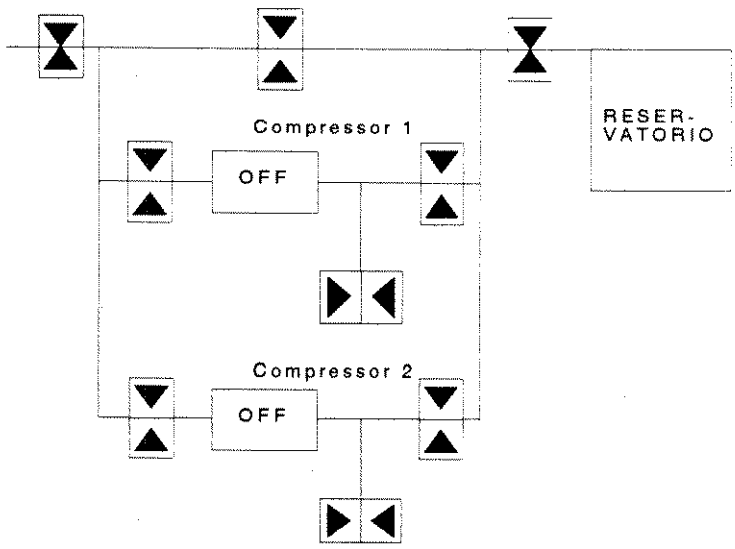


Figura 3.6 a) Estado de "Falha Segura"

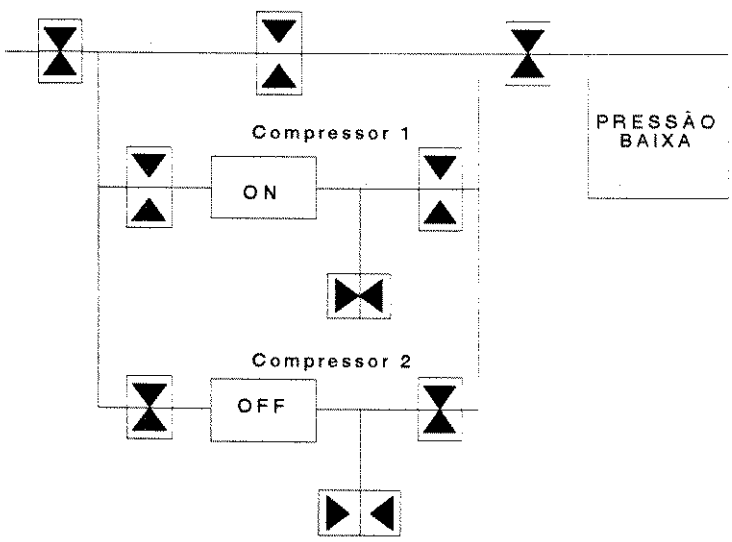


Figura 3.6. b) Estado "Ativado"

Figura 3.6. Protótipo de Animação

Considera-se que **Prototipação de Animação** é uma ferramenta prática e funcional para obtenção de retorno ao usuário, numa linguagem acessível e concreta.

Capítulo 4. METODOLOGIA "DESENVOLVIMENTO ESTRUTURADO PARA STR".

4.1. Introdução.

No decorrer da década de 80, a metodologia de desenvolvimento de sistemas criada por Yourdon-DeMarco [YOU89] recebeu extensões para manipular software de tempo real. As extensões desenvolvidas por Hatley [HAT91] e Ward-Mellor [WAR86] são atualmente as metodologias mais conhecidas para análise de sistemas de tempo real.

De acordo com Falk [FAL88], em sistemas críticos como os utilizados em aplicações aero-espaciais e de defesa, essas metodologias são utilizadas frequentemente; já em aplicações industriais como robótica e comunicação, sua utilização se encontra no início.

Mais recentemente, Paul Ward associado com outros pesquisadores, desenvolveu a metodologia ESML "Extended Software Modelling Language" [BRU87] que combina características de ambos os métodos : Ward-Mellor e Hatley. Segundo Paul Ward, "a evolução dos modelos citados está ainda no início e futuras mudanças serão inevitáveis".

Neste trabalho, a metodologia empregada é a desenvolvida por Ward-Mellor. Sua capacidade de modelar eventos assíncronos, como interrupções e falhas, é particularmente útil em aplicações industriais e sistemas controlados por microprocessadores. Sua disponibilidade em ferramentas CASE e para utilização pelo Instituto de Automação - CTI como metodologia de desenvolvimento, também foram fortes motivadores para a escolha dessa metodologia.

A Seção 4.2. discute as características dos STR, e a Seção 4.3. apresenta resumidamente as características da metodologia, realçando os pontos necessários para compreensão do prototipador implementado.

4.2. Sistemas de Tempo Real.

Sistemas de Tempo Real (STR) geralmente apresentam um conjunto de características que são distintas das de outros sistemas [DEU88]. Estas características impõem diversas necessidades que devem ser incluídas nas notações de modelagem para Sistemas de Tempo Real (Figura 4.1).

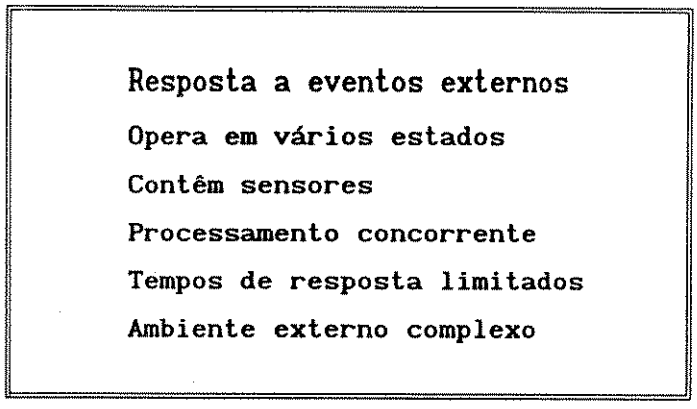


Figura 4.1. Características dos STR.

A característica considerada como a mais importante e complexa é a necessidade do sistema **receber** e **responder** a estímulos externos dentro de um intervalo de tempo bem determinado.

Devido ao acoplamento ao meio externo, sistemas de tempo real operam em múltiplos modos ou estados. A resposta para o mesmo dado de entrada pode ser diferente dependendo do estado do sistema.

Sistemas de Tempo Real são modelados para processamento concorrente de entradas múltiplas e simultâneas. Isto é principalmente uma questão de implementação, mas que impõe a necessidade de indicar paralelismo de fluxo de dados e de processamento no modelamento.

Finalmente, STRs são associados com ambientes externos complexos. Sensores, outros sistemas, operadores, usuários e canais de comunicação no ambiente do sistema, constituem um ponto crucial que deve ser muito bem interpretado pelos projetistas. Deve-se particionar o sistema em múltiplas visões, simples o suficiente para que os usuários possam compreender o modelamento proposto.

A metodologia de desenvolvimento de software de Ward & Mellor [FAL88], apresentada neste capítulo, procura representar os sistemas de tempo real realçando sua característica de permanecer em estados bem determinados, além de descrever as ações executadas como resultado de eventos ocorridos no ambiente.

4.3. Apresentação da Metodologia.

4.3.1. O Processo de Modelamento.

No processo de desenvolver sistemas, é comum que os analistas partam rapidamente para o desenvolvimento relegando a análise do sistema para "fases posteriores" do projeto. Esta atitude é onerosa para o desenvolvimento, gerando normalmente frequentes retornos a pontos que foram ignorados numa primeira aproximação.

Para contornar este problema, é frequentemente recomendado modelar o sistema inicialmente, revisar e modificar o modelo extensivamente e, então, usar o modelo como um plano para a implementação.

O processo de desenvolvimento de sistemas tem sido frequentemente dividido para refletir a distinção entre "o problema" e "a solução". De Marco [DEM78] e outros autores têm tentado enfatizar esta divisão, distinguindo modelos de sistema físicos e lógicos. Um refinamento desta idéia de separar "problema"/"solução" foi desenvolvido por McMenamin e Palmer [McM84]. A estrutura total da estratégia de modelagem usada por Ward & Mellor é baseada no trabalho de McMenamin e Palmer e envolve separação da **essência** (problema) de um sistema de sua **implementação**.

Considerando-se que um sistema deve atuar em um ambiente específico e dado que possui uma tarefa a cumprir, a metodologia "Desenvolvimento Estruturado para STR" apresenta o sistema através de dois modelos. O primeiro modelo, denominado **Modelo Essencial**, descreve "o que" o sistema deve fazer (as atividades essenciais) e "quais" são os dados que ele deve armazenar (a memória essencial). No segundo modelo, denominado **Modelo de Implementação**, o Modelo Essencial é adaptado para ser utilizado com a tecnologia de implementação disponível.

4.3.1.1. O Modelo Essencial.

A descrição do **modelo essencial** é composta de dois elementos:

O Modelo do Ambiente - é uma descrição do ambiente no qual o sistema opera, juntamente com os eventos externos aos quais o sistema deve responder.

O Modelo de Comportamento - é uma descrição do comportamento desejado do sistema. Este modelo é composto por dois elementos: o Esquema de Transformações e o Esquema de Dados.

O Esquema de Transformações é baseado na notação gráfica para Diagrama de Fluxo de Dados proposto por DeMarco [DEM78], com extensões para permitir modelagem de sistemas de tempo real representando aspectos de controle e de temporização [WAR86].

O Esquema de dados representa graficamente a organização das informações que devem ser mantidas pelo sistema.

4.3.1.2. O Modelo de Implementação.

O **Modelo de Implementação** enfatiza aspectos de "como" o sistema será efetivamente implementado. Para alcançar este objetivo, o modelo é apresentado em uma série de passos, distinguindo os diferentes aspectos em termos de características de concorrência.

Num primeiro nível, são utilizados **processadores** para realizar as atividades e armazenar os dados definidos pelo Modelo Essencial; os processadores são os únicos elementos que implementam uma concorrência verdadeira.

A seguir, é definida a tecnologia dentro do processador : **tarefas** ou **blocos de código** que podem ser manuseados como unidades. Dentro de um único processador, tarefas **simulam** concorrência, ou seja, cada tarefa se comporta como se possuisse todo o tempo de processamento disponível, entretanto, compartilha a unidade de processamento com todas as outras tarefas presentes no processador.

Finalmente, é introduzida a tecnologia de **módulos**, usualmente definida por uma linguagem de programação. **Módulos** são unidades que não exibem concorrência; uma vez ativados executam sua função e retornam ao chamador que está aguardando sua conclusão.

4.3.2. Modelando Transformações.

Este tópico apresenta a notação para modelar o **Esquema de Transformações**, que faz parte do Modelo Essencial da metodologia.

O Esquema de Transformações descreve os requisitos funcionais; segmenta estes requisitos em funções ou processos, e representa-os com uma rede interligada pelos fluxos de dados/controle. Seu propósito principal é mostrar como cada processo transforma seus fluxos de entrada em fluxos de saída e mostrar os relacionamentos entre estes dois fluxos.
Como já afirmado, o Esquema de Transformações é baseado na notação para diagrama de fluxo de dados proposto por DeMarco, com extensões para permitir a representação de sistemas de tempo real.

4.3.2.1. O Modelo de Fluxo de Dados (DFD).

O modelo proposto por DeMarco representa graficamente as transformações sofridas pelos dados até a obtenção dos resultados desejados. Para representar essas transformações são utilizados basicamente quatro elementos:

- **Fluxos de Dados** : representam o movimento dos dados dentro do sistema por meio de arcos direcionados e identificados por nomes.

- **Transformações** : são apresentados como círculos e representam os componentes funcionais básicos de um sistema [COO90]. Transformações de Dados recebem dados da entrada, transformam os dados de acordo com um algoritmo interno ou descrição do processo, e produzem saídas.

- **Armazenadores de Dados** : representados por um par de segmentos de reta paralelos. Um armazenador de dados atua como um depósito de dados cujos valores são modificados em pontos discretos no tempo. Um armazenador de dados é uma abstração de um arquivo.

- **Terminadores (Entidade Externa)** : representam a interface do sistema com o ambiente.

Na Figura 4.2 apresenta-se a notação para representar os elementos do DFD :

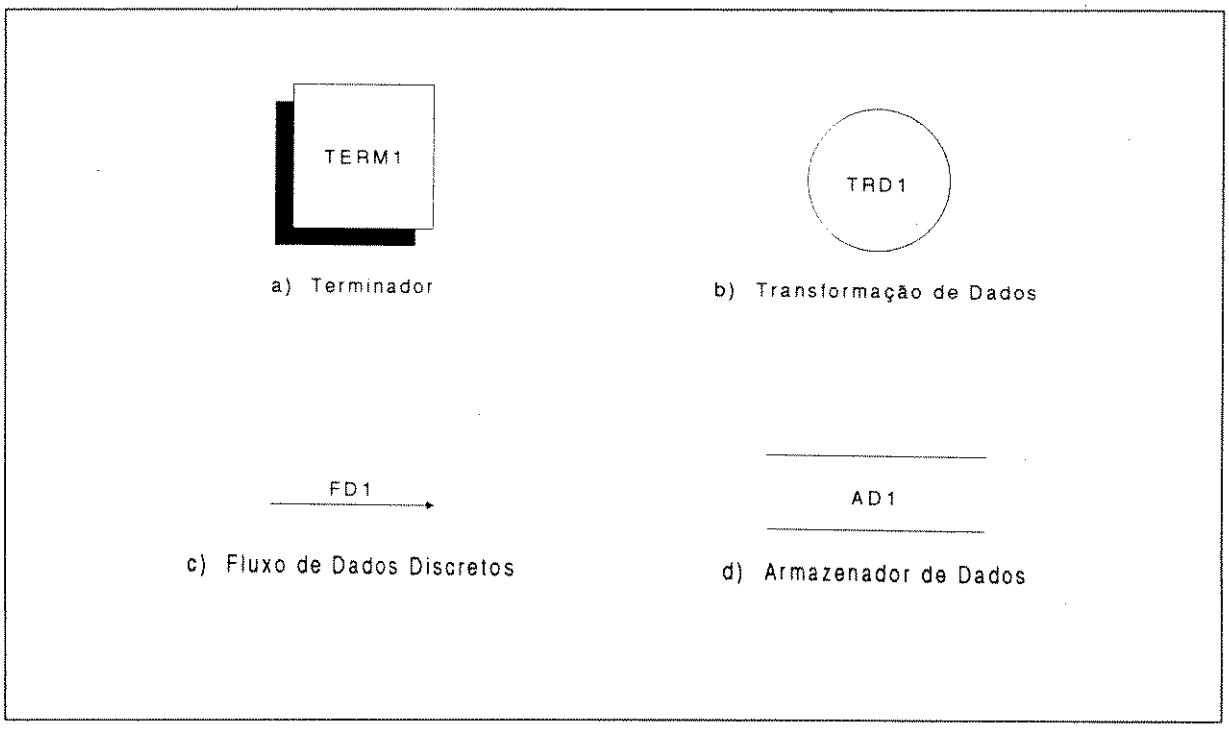


Figura 4.2. Elementos do DFD.

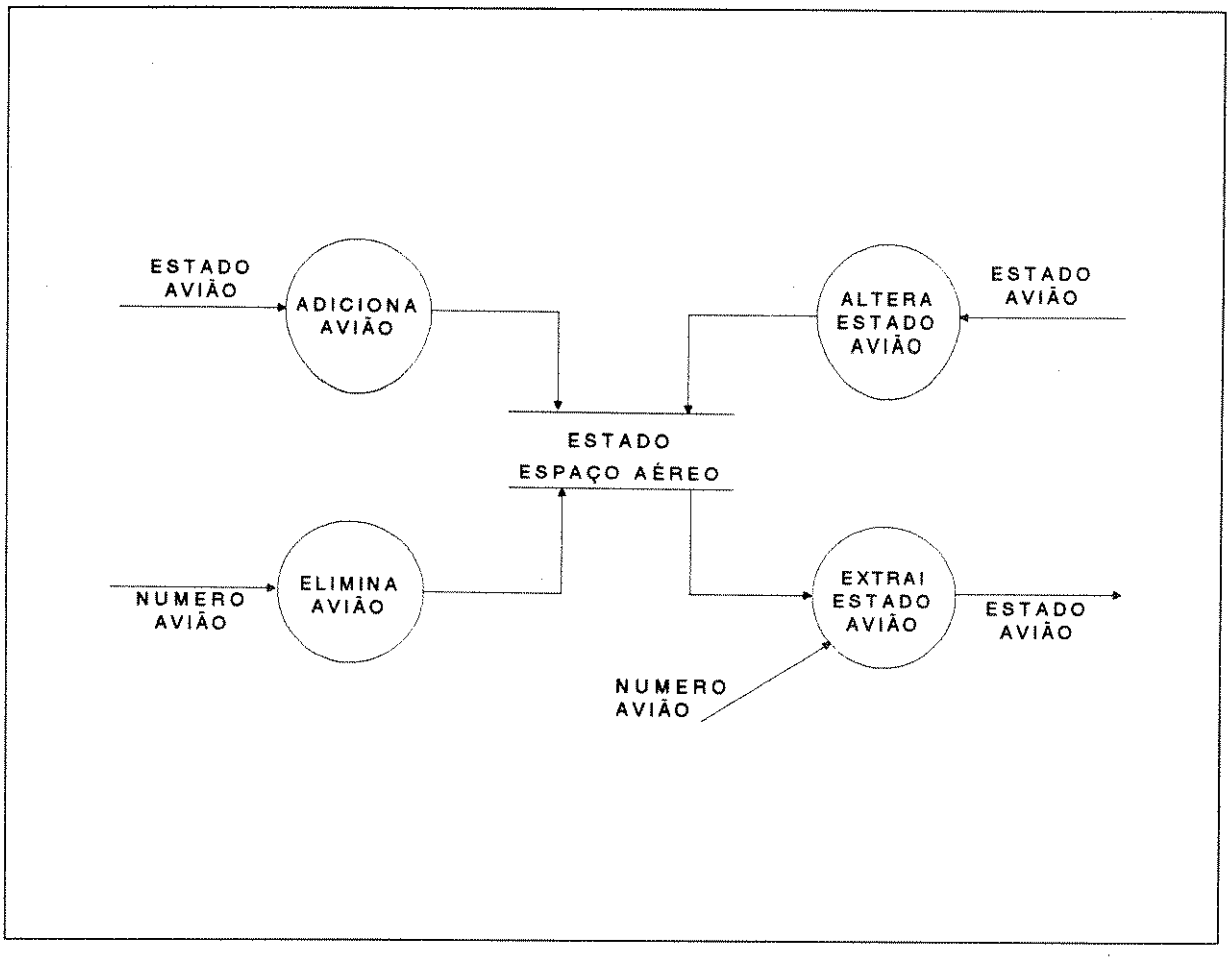


Figura 4.3. DFD de um sistema para controle de tráfego aéreo.

Na Figura 4.3 apresenta-se um sistema para controle de tráfego aéreo. O fluxo discreto "estado_avião" é composto pelos elementos <numero_avião, velocidade, altitude>. Quando um avião entra/sai do espaço aéreo controlado pelo sistema, tem suas características atualizadas no armazenador "estado_espaço_aéreo" que mantém o estado de todos os aviões presentes no sistema. O fluxo conectando o armazenador às transformações não possui nome, ele representa a disponibilidade do armazenador para a transformação. A transformação pode atualizar os dados presentes no armazenador ou simplesmente realizar uma leitura dos mesmos; o sentido dos dados no fluxo é determinado pela posição da seta no arco.

4.3.2.2. Extensões ao Modelo DFD.

A primeira extensão a ser realizada consiste em considerar o fator tempo com relação aos fluxos de dados do sistema. Dessa forma, **fluxos de dados contínuos** são associados a um valor ou a um conjunto de valores definido continuamente sobre um intervalo de tempo. Servem para representar características físicas do ambiente como temperatura, tensão elétrica ou pressão monitoradas pelo sistema. São representados por uma linha contínua com duas setas.

A presença de um fluxo contínuo numa transformação determina uma transformação contínua. Tal transformação encontra-se continuamente ativada e produzindo resultados. A Figura 4.4 representa um exemplo de uma transformação contínua. Neste caso, o fluxo contínuo "temperatura" está sempre presente e o objetivo da transformação é manter a temperatura dentro de limites aceitáveis. Para efetuar esta operação, a transformação continuamente envia sinais para o fluxo contínuo que controla o nível de um dispositivo.

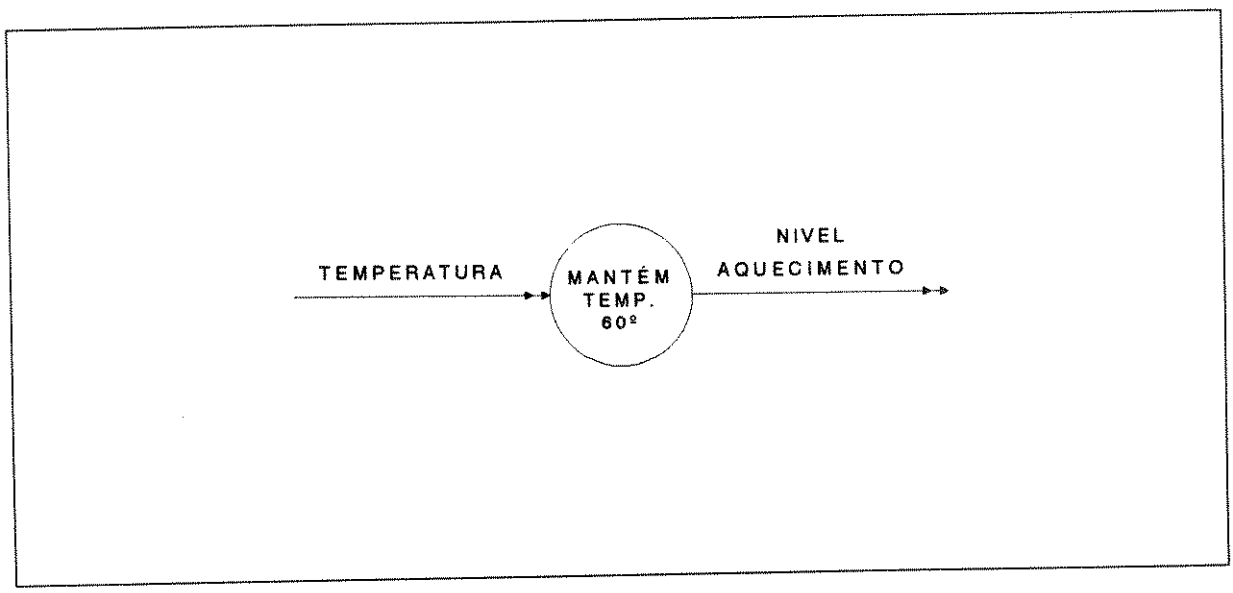


Figura 4.4. Fluxo Contínuo.

Fluxos de dados discretos, por outro lado, são associados a um valor variável ou a um conjunto de valores variáveis que são definidos em pontos discretos no tempo, sendo considerados nulos em outros pontos. Esses elementos são representados por uma linha contínua com uma única seta.

A existência de fluxos contínuos e discretos permite uma interpretação dinâmica da transformação , uma vez que a presença das informações no fluxo habilitam a transformação.

Dessa forma um fluxo discreto possui duas características: o conteúdo e a ocorrência do fluxo como entrada ou saída num instante específico do tempo.

Muitos sistemas de tempo real contêm fluxos que não possuem conteúdo, são simplesmente sinalizadores de que algo ocorreu, habilitando a transformação. Tais fluxos são denominados **fluxos de eventos** e são representados por uma linha tracejada orientada por uma única seta.

Uma transformação que aceita um fluxo de evento como entrada e produz fluxos de evento como saída é denominada transformação de controle. Ela é representada por um círculo tracejado.

É também possível definir **armazenadores de eventos**, que são representados por um par de segmentos de linha tracejada. Neste tipo de armazenador, os fluxos de eventos produzidos por uma ou mais transformações estão sujeitos a um atraso de armazenamento antes de serem utilizados.

A Figura 4.5 apresenta a notação gráfica destes elementos anteriormente descritos :

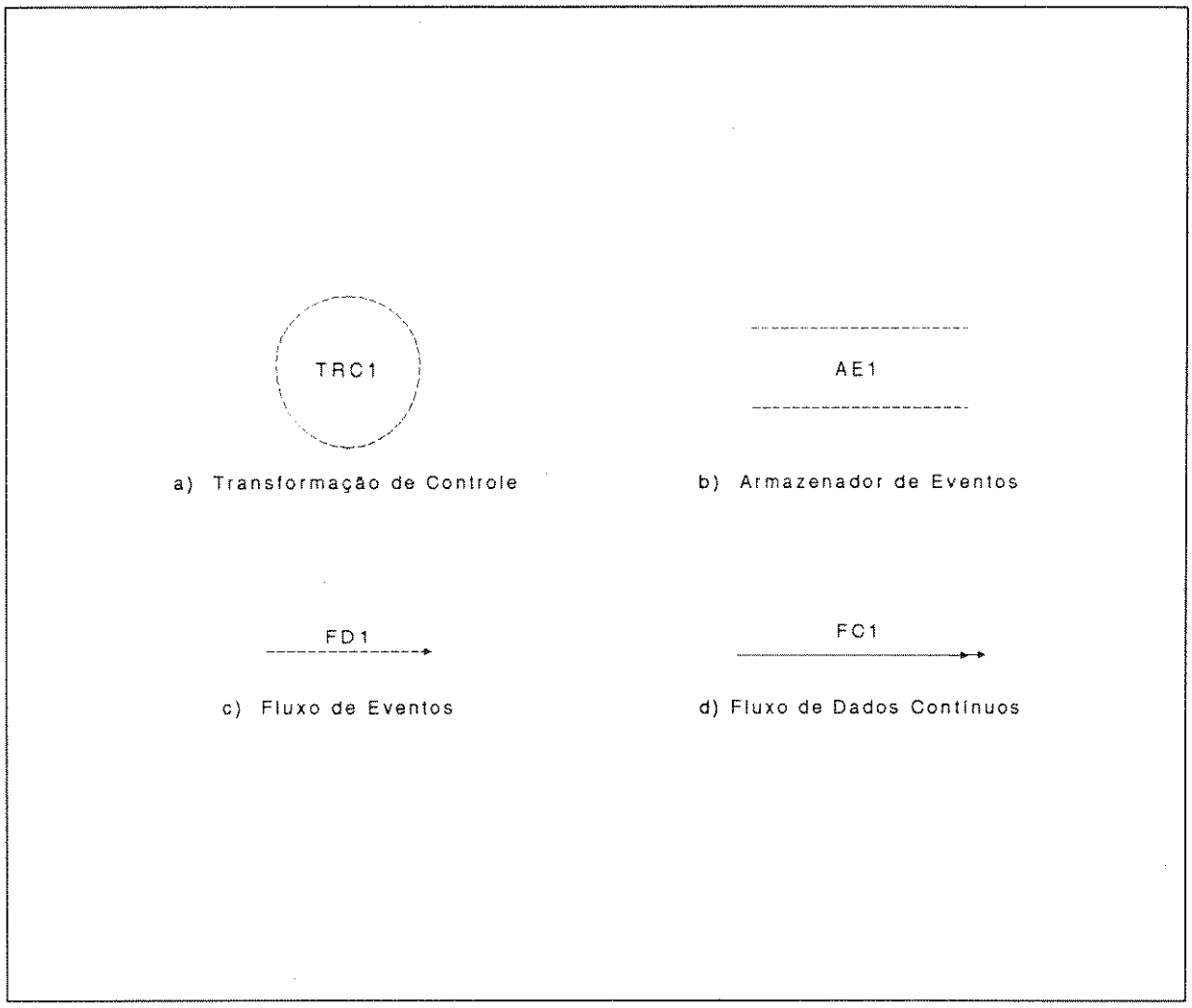


Figura 4.5. Elementos de Extensão ao DFD.

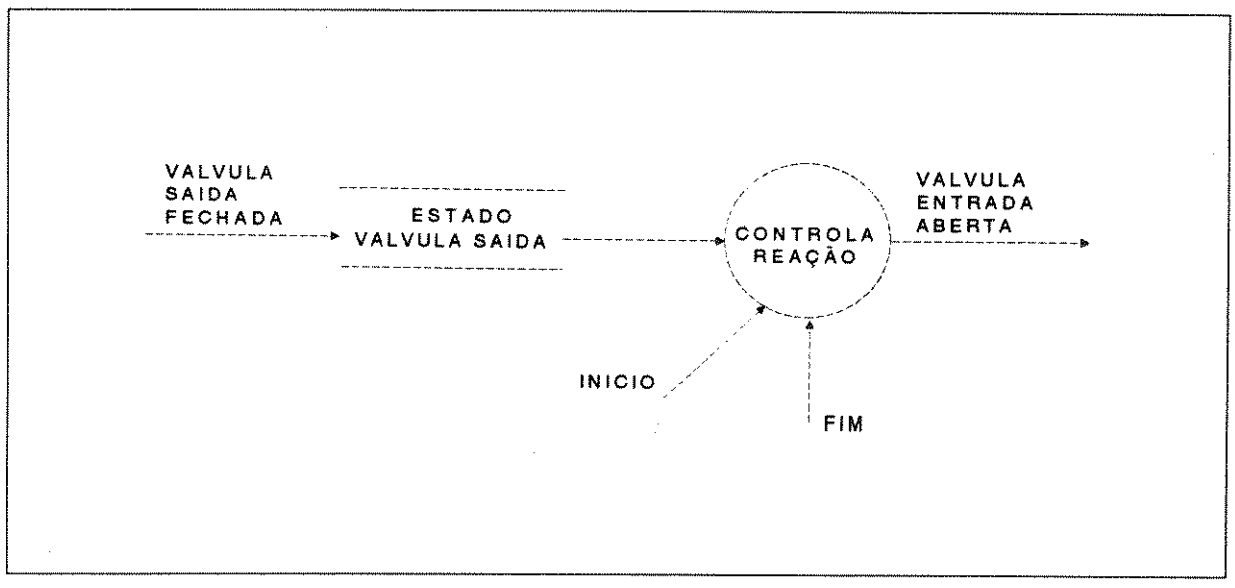


Figura 4.6. DFD com características de Controle.

Na Figura 4.6 temos um exemplo de utilização dos elementos descritos. A transformação de controle "controla_reação" recebe dois fluxos de eventos : "início", "fim" e um fluxo de eventos recebido do armazenador "estado_válvula_saída", e pode produzir um fluxo de evento denominado "válvula_entrada_aberta". Note que o fluxo "válvula_saída_fechada" é entrada para o armazenador de eventos, denominado "estado_válvula_saída" que mantém o evento até ele ser efetivamente consumido pela transformação de controle.

Finalmente vamos definir o conceito de **múltiplas instâncias** dentro de um sistema. É muito comum num sistema de tempo real, ter-se um elemento repetido diversas vezes dentro de uma mesma aplicação.

Para ressaltar esta característica na metodologia, representaremos múltiplas instâncias de transformações por um círculo duplo. Esta convenção permite declarar as transformações necessárias e enfatiza, ao mesmo tempo, a existência de múltiplas instâncias.

Este conceito somente se aplica a **sistemas equivalentes**, e não para sistemas similares. Esta convenção é somente útil quando alguma coordenação geral para as instâncias individuais é necessária. Na Figura 4.7, por exemplo, o conjunto de reações equivalentes pode ser desligado ao mesmo tempo utilizando o fluxo de evento "desativação_geral" que atinge a transformação de controle "controla_processo_individual" em todas as instâncias. Se os subsistemas são completamente independentes, é suficiente modelar um subsistema e construir múltiplas cópias.

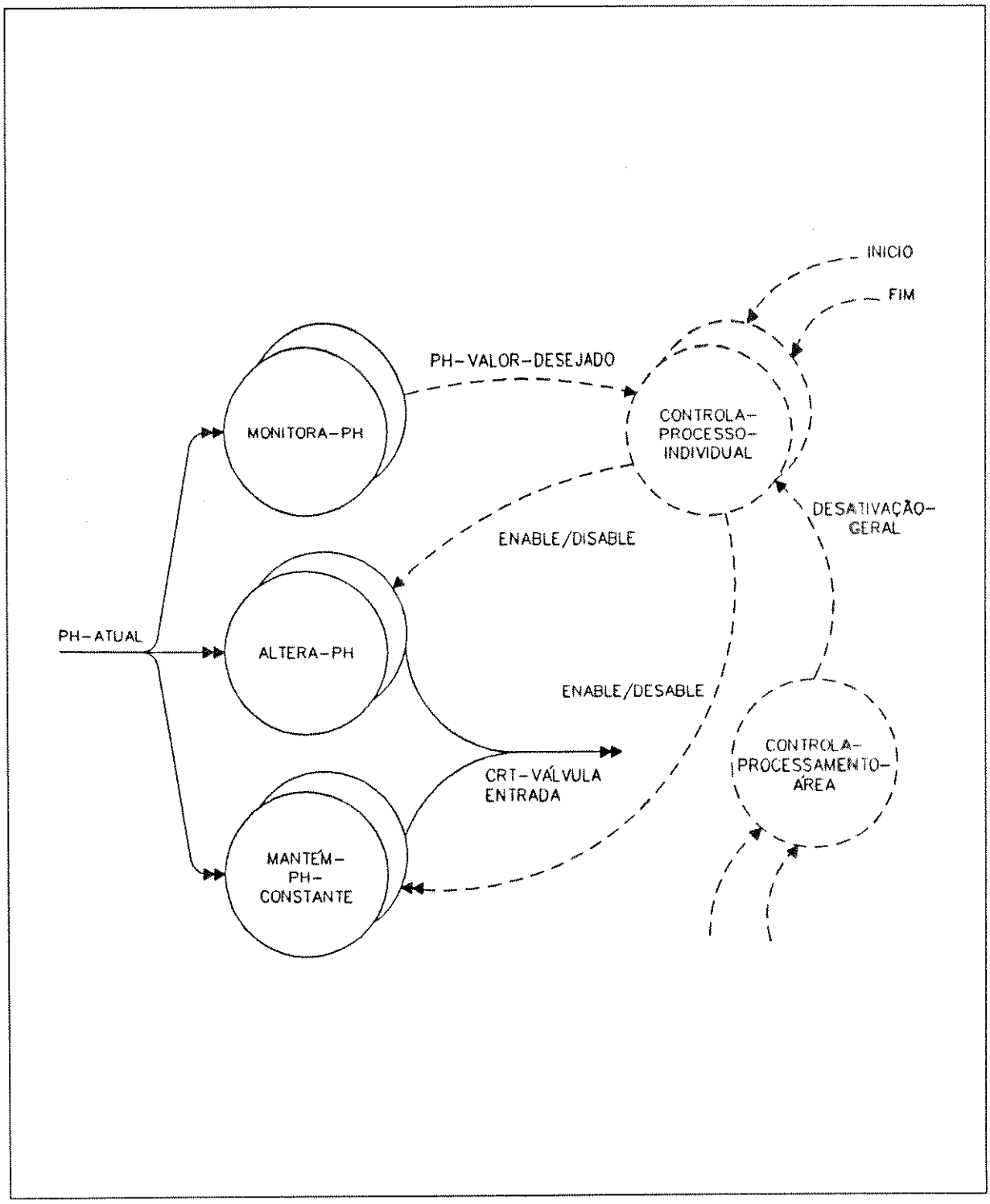


Figura 4.7. DFD com características de múltiplas instâncias.

4.3.3. A Composição do Esquema de Transformações.

Uma vez definidos os elementos básicos da metodologia, é necessário definir convenções para a sua perfeita utilização.

Antes deste passo, vamos ainda caracterizar uma classe particular de fluxos de eventos denominados "**prompts**". Estes fluxos não são "entradas verdadeiras" , simplesmente permitem a comutação entre os estados "Habilitado" e "Desabilitado" para uma transformação de dados. No estado "Desabilitado", a transformação ignora seus fluxos de entrada não gerando fluxos de saída.

Dessa forma, fluxos denominados "Enable", "Disable" e "Trigger" são caracterizados como "prompt". A Figura 4.8 apresenta algumas transformações ativadas por "prompts".

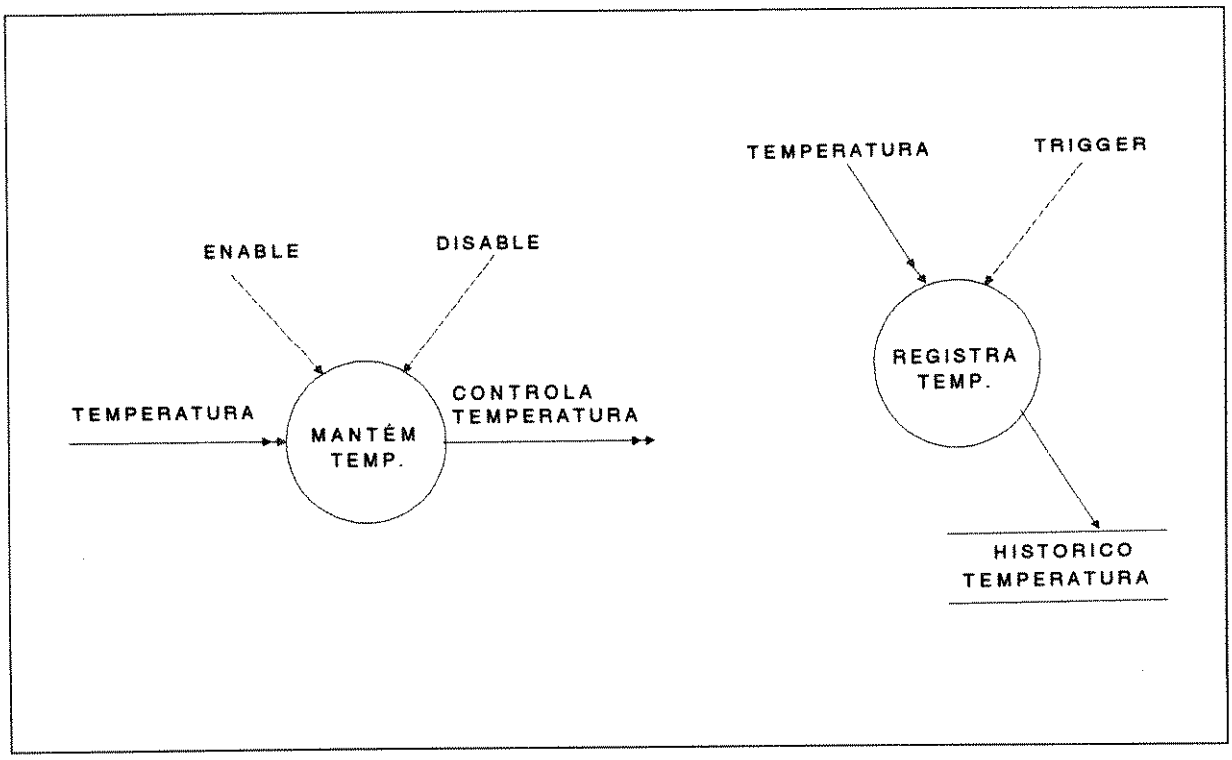


Figura 4.8. DFD com transformações ativadas por "Prompts".

Na Figura 4.8 a transformação "mantem_temp" muda para o estado "Habilitado" ao receber o "prompt enable". Quando a transformação está "Habilitada" ela processa o fluxo de entrada "temperatura" e continuamente gera o fluxo contínuo de saída "controla_temperatura". Por outro lado, ao receber o "prompt disable" a transformação passa a ignorar seu fluxo de entrada. O "prompt trigger" por outro lado, permite à transformação uma única execução, ou seja, uma única "coleta" do fluxo contínuo "temperatura" gerando uma atualização do armazenador "histórico_temperatura".

É necessário ressaltar que não existe exigência alguma de que uma transformação de dados possua "prompts". Uma transformação que não possua "prompts" está continuamente habilitada, sendo que, a única restrição para sua execução é a presença de todos os seus fluxos de entrada. Para separar aspectos contínuos e discretos de um sistema as seguintes convenções sobre fluxos devem ser observadas :

- Para uma transformação de controle, somente fluxos de eventos são permitidos como entradas e saídas.

- Para uma transformação de dados, somente fluxos de dados são permitidos como entrada, mas tanto fluxos de dados quanto fluxos de eventos podem ser produzidos como saída.

- Para transformações de dados e controle, fluxos de eventos rotulados "Enable", "Disable" ou "Trigger" são interpretados como "prompts", e não como entradas.

- Somente transformações de controle podem ser "prompt" de outras transformações.

Estas convenções, além de separar aspectos contínuos e discretos de um sistema, servem também para separar controle e dados.

4.3.4. Especificando Transformações de Controle.

Neste item serão descritas técnicas para modelar transformações de controle e descrever seu relacionamento com fluxos e armazenadores de eventos.

São apresentadas como círculos de linhas tracejadas e representam os componentes de um sistema que direcionam a operação de transformações de dados, ativando-as ou desativando-as através do envio de sinais.

Uma transformação de controle mapeia fluxos de eventos de entrada em fluxos de eventos de saída. Desde que fluxos de eventos são fundamentalmente livres de conteúdo, as transformações devem combinar o fluxo com uma memória interna para produzir o fluxo de saída apropriado.

A lógica da transformação de controle é descrita por um "**Diagrama de Transição de Estados**" [HOP79] que apresenta a sequência de entradas e saídas associadas a cada estado do sistema. As características de autômato serão apresentadas através de um exemplo. Suponha a transformação de controle da Figura 4.9 :

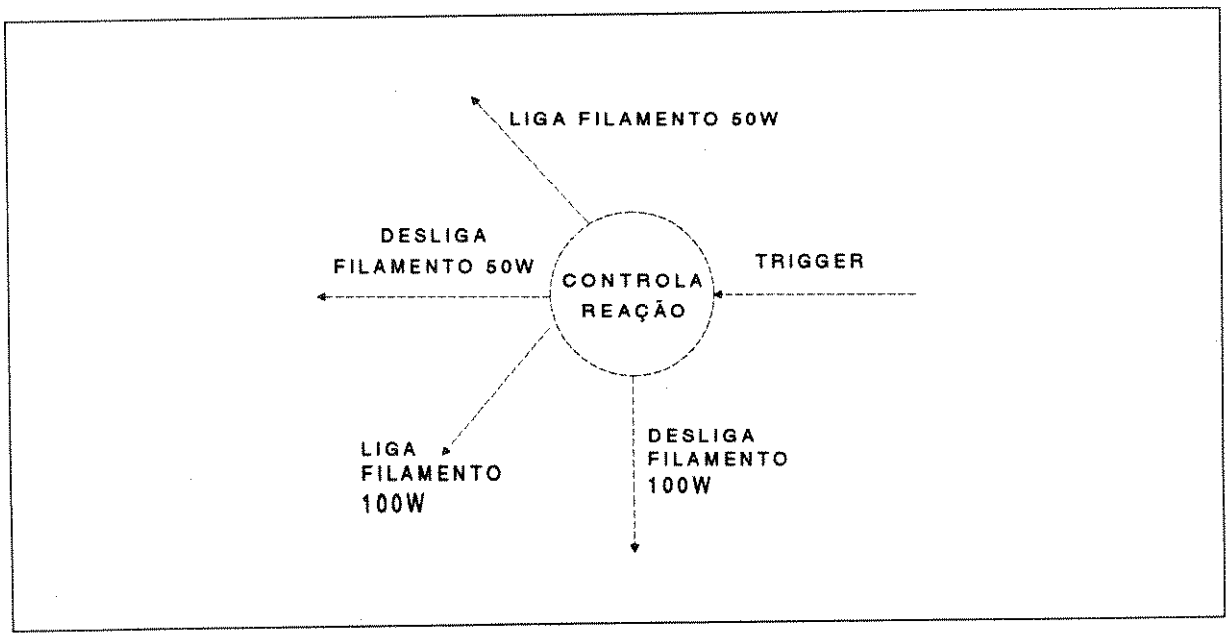


Figura 4.9. Transformação de Controle.

Esta transformação de controle representa uma lâmpada que possui dois filamentos : 50W e 100W; e um acionador que, quando ativado, comuta entre as seguintes saídas : acende 50W, apaga 50W e acende 100W, acende 100W e 50W, apaga a lâmpada. Com essas saídas é possível obter as seguintes luminosidades : apagado, 50W, 100W e 150W.

Um diagrama de estados que modela esta transformação é apresentado na Figura 4.10 :

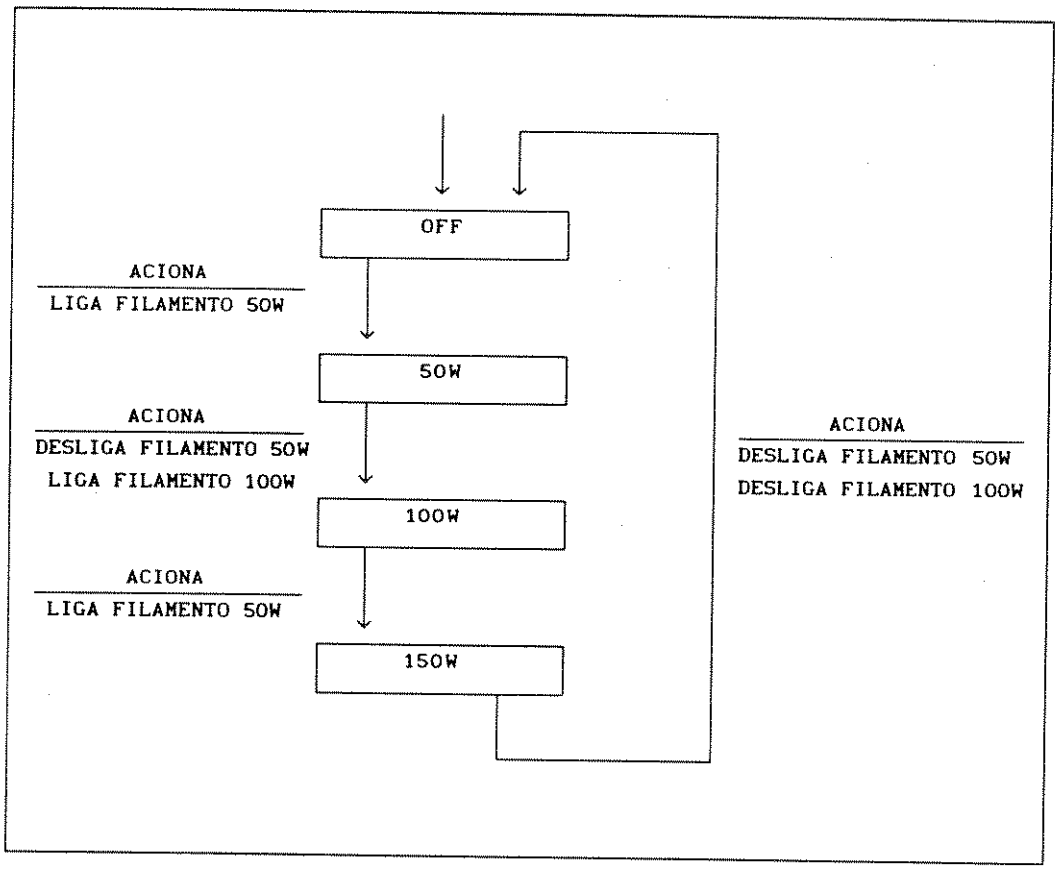


Figura 4.10. Diagrama de Estados.

Os componentes representados neste diagrama são **estados**, representados por retângulos; **transições** representadas por arcos orientados; a **condição de transição**, apresentada acima da linha próxima à transição; e a **ação da transição**, apresentada abaixo da linha próxima a transição.

- **Estado** : representação do sistema num dado instante. Permanece inalterado até ser ativado por um evento externo. Um dos estados da transformação é denominado **estado inicial** e representa o comportamento inicial do sistema antes de receber uma ativação. O estado inicial pode ser identificado por um estado recebendo um arco que não tem estado de origem.
  Um ou mais estados do diagrama são denominados **estados finais**. Os estados finais possuem transições de entrada mas não geram transições de saída.

- **Transição** : representação de mudança de um estado para outro. Associados a transições temos dois elementos : condições e ações.

- **Condições** : são eventos que permitem que o sistema realize uma transição. A condição é identificada com um fluxo de evento de entrada, sinalizando que a condição ocorreu.

- **Ações** : são realizadas quando as transições ocorrem. Uma ação é uma atividade indivisível que é identificada com um fluxo de evento de saída. Várias ações podem ser tomadas numa transição; elas são, por definição, tomadas instantâneamente na ocorrência de uma transição.

4.3.5. Especificando Transformações de Dados.

Assim como a transformação de controle foi descrita, a transformação de dados também deve ser detalhada para descrever quais operações são necessárias para gerar os fluxos de dados de saída a partir do fluxo de dados de entrada e dos armazenadores de dados.

Distintamente da transformação de controle, a transformação de dados possui inúmeras técnicas para a sua especificação.Estas técnicas podem ser agrupadas em duas classes:

a) **Procedimental** - descreve um conjunto de regras ordenadas através das quais é possível obter a saída dos fluxos de entrada. São exemplos desta classe:

- especificação através de uma linguagem de alto nível.
- PDL (Program Design Language).
- Pseudo-código.
- Português estruturado.

Destes elementos, tanto a especificação através de uma linguagem de alto nível, quanto o PDL, são passíveis de uma análise automatizada devido ao rigor sintático da sua descrição. Exatamente por esse motivo, corre-se o risco de direcionar a especificação para a implementação numa fase na qual isto não é desejável.

b) **Não Procedimental** - Este tipo de especificação procura obter critérios para determinar se a implementação é ou não uma solução para o problema. São exemplos dessa classe a especificação tabular e a especificação através de pré e pós-condições.

A especificação através de pré e pós-condições consiste em relacionar condições sobre valores de entrada com as condições correspondentes sobre os valores de saída. Como exemplo, no caso de diagramas de estado, a pré-condição especifica a condição e o estado inicial, e a pós-condição especifica a ação e o estado final.

4.3.6. Modelando Dados Armazenados.

Neste item será apresentada a notação responsável pelo modelamento do **Esquema de Dados**, associada com exemplos para sua melhor compreensão.

A notação utilizada é derivada do diagrama de entidades e relacionamentos proposto por Chen [CHE76], com algumas extensões e refinamentos propostos por Flavin [FLA81].

4.3.6.1. Notação Básica : Objetos e Relacionamentos.

A metodologia é composta por dois tipos básicos : objetos e relacionamentos.

Os objetos, apresentados como um retângulo com um nome em seu interior, representam um conjunto de entidades reais que compõe o sistema em análise.

Para conectar objetos são utilizados relacionamentos. Os relacionamentos, apresentados como losangos e identificados por um nome, representam uma ligação específica entre os objetos que eles conectam (Figura 4.11).

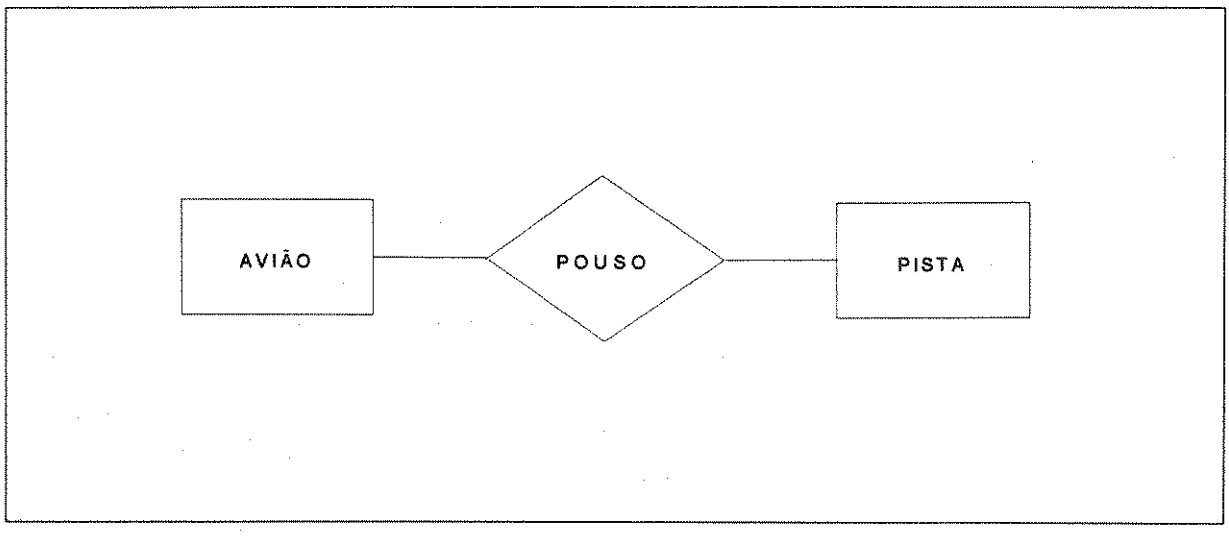


Figura 4.11. Esquema de Dados.

Na Figura 4.11 temos um **Esquema de Dados**, onde podemos identificar como **objetos**, avião e pista , e como **relacionamento**, pouso.
O relacionamento é multidirecional por definição; no exemplo, tanto **pistas sofrem pousos de aviões** como **aviões pousam em pistas** são afirmações legítimas.

No modelamento de sistemas de tempo real, utilizaremos os objetos para representar categorias de armazenadores; dessa forma, objetos representam uma abstração de vários mecanismos armazenadores como : arquivos, bancos de dados, buffers, pilhas, etc.

4.3.7. Hierarquia do Modelo.

O **Esquema de Transformações** é baseado no refinamento progressivo de diagramas de fluxo de dados. O primeiro passo é criar o **Esquema de Contexto** (Figura 4.12), cujo objetivo é mostrar o sistema como uma única transformação conectada aos fluxos de dados e fluxos de controle que caracterizam a interface entre o sistema e seu meio ambiente [DEU88].

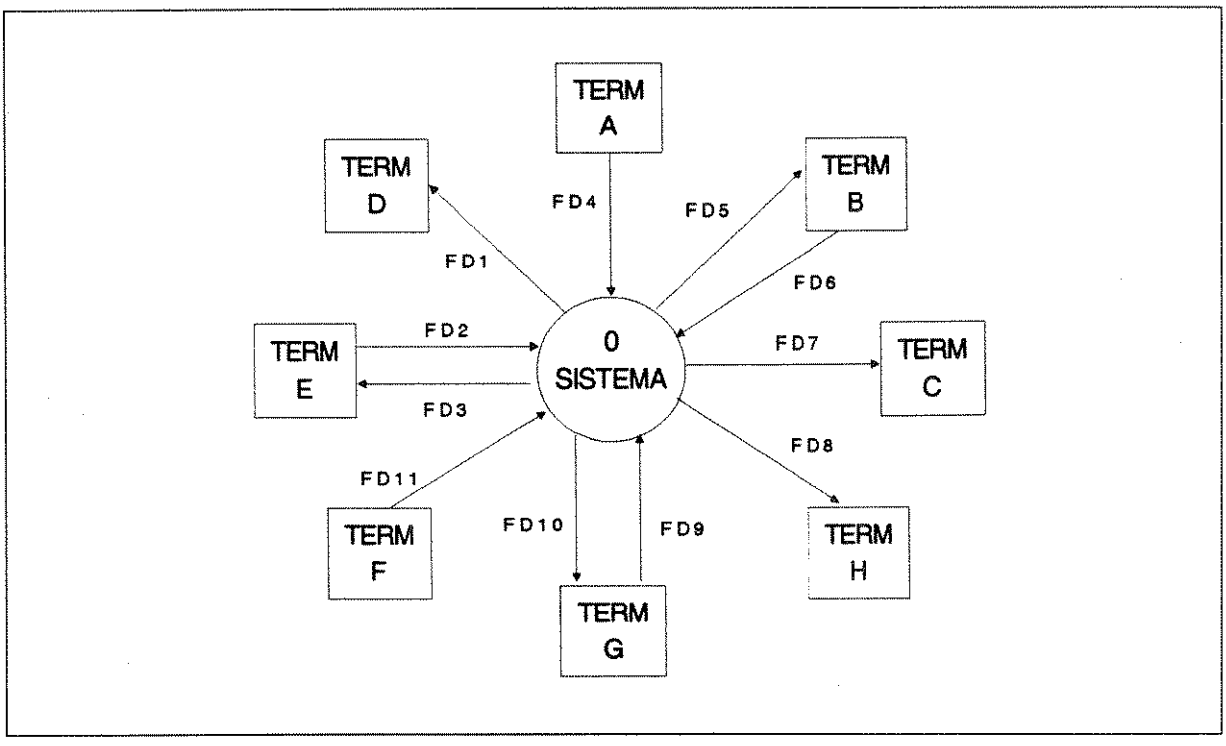


Figura 4.12. Diagrama de Contexto.

Os terminadores presentes no diagrama de contexto representam a interface com o meio; fisicamente podem assumir diversas formas como por exemplo : sensores, atuadores, operadores, linhas de comunicação, etc. Na verdade o diagrama de contexto representa a fronteira da implementação; a geração dos fluxos de entrada, associados a transformação do nível 0, na figura denominada "SISTEMA", não são objetos de interesse para a metodologia que se limita a definir claramente o conteúdo de tais fluxos.

A seguir, esta única transformação é decomposta para mostrar um **primeiro nível** de detalhes do sistema. O primeiro passo nesta decomposição consiste em criar uma Transformação de Controle que gerencia os eventos externos do sistema e o relacionamento dessa transformação com as transformações de dados que por ela são "habilitadas" ou "desabilitadas", de acordo com o estado do sistema.

A partir deste nível, transformações de dados e de controle podem ser decompostas em subníveis para facilitar a compreensão de seus relacionamentos.

Na Figura 4.13 temos uma representação da decomposição em sub-níveis. A transformação **G1** é decomposta nas transfomações **T1.1** e **T1.2** e a transformação **G2** foi decomposta na transformação **T2.1** e **T2.2**. O número de fluxos que são entrada/saída para a transformação de nível hierárquico imediatamente superior deve coincidir com o número de fluxos que são entrada/saída para as transformações geradas na decomposição, mantendo dessa forma a consistência do modelo. Segundo Ward & Mellor [WAR85], o limite inferior do processo de decomposição é alcançado quando a lógica de uma transformação pode ser rigorosamente descrita por uma especificação ocupando uma tela ou parte de uma tela; na Figura 4.13 temos as seguintes especificações : **S1.1**, **S1.2**, **S2.1**, **S2.2** e **S3**. Especificações são necessárias somente para o nível mais baixo das transformações, desde que os diagramas representam especificações para as transformações de alto nível que foram decompostas.

Pode-se concluir esta breve apresentação da hierarquia do modelo acrescentando que, a cada nível sucessivo de diagramas, expressa-se um refinamento dos diagramas de nível superior.

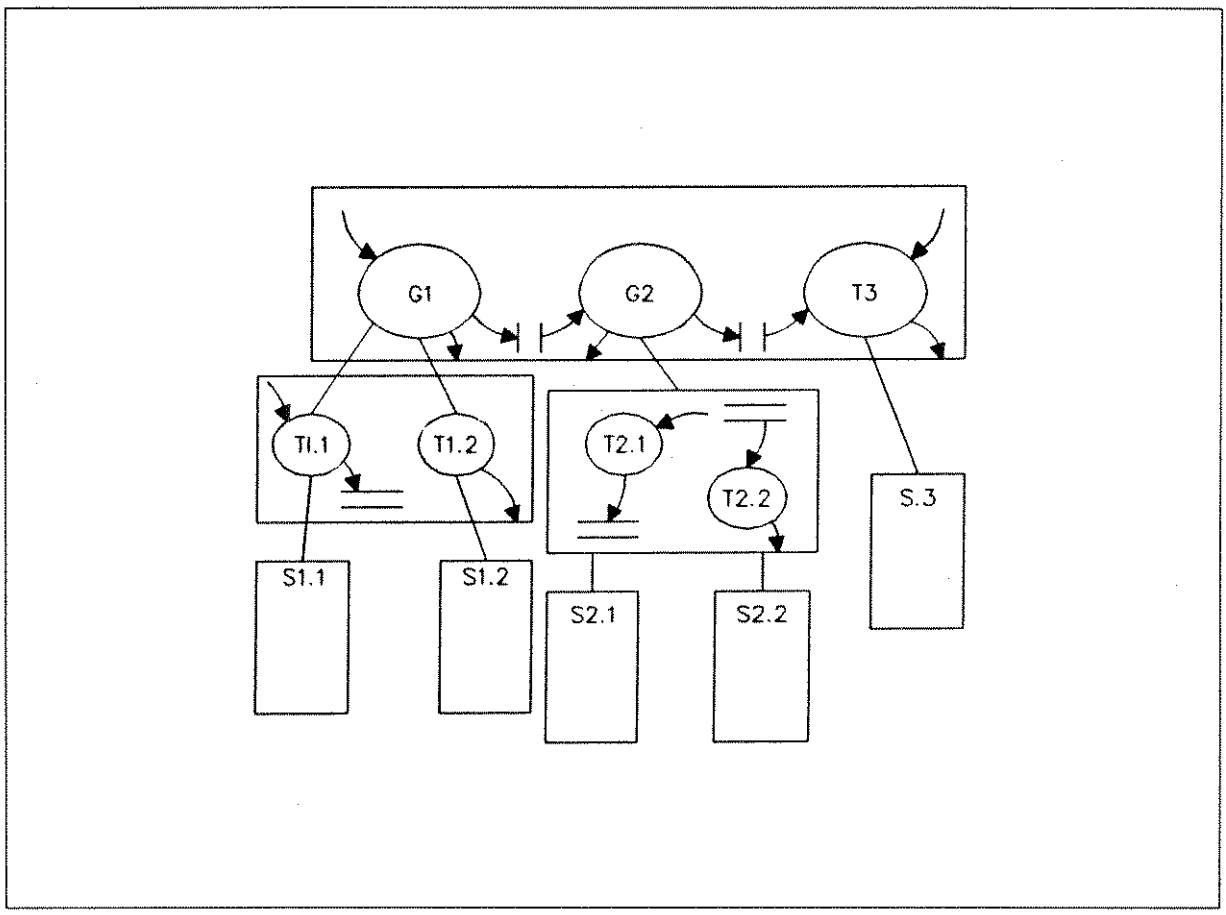


Figura 4.13. Hierarquia do Modelo.

Capítulo 5. UTILIZANDO A FERRAMENTA ProTR.

5.1. Introdução.

Na última década novas técnicas de engenharia de software foram desenvolvidas com a finalidade de apoiar o desenvolvimento de sistemas computacionais. Na área de especificação de sistemas de Tempo Real, uma das contribuições mais difundidas é a metodologia "Desenvolvimento Estruturado para Sistemas de Tempo Real", desenvolvida por Ward-Mellor.

Paralelamente a este processo, as técnicas de prototipação tem se firmado como um valioso auxílio ao desenvolvimento de sistemas computacionais [DEA83], minimizando o tempo de desenvolvimento e gerando produtos mais confiáveis.

O objetivo da ferramenta **ProTR** é gerar um protótipo do sistema alvo a partir da especificação realizada através da metodologia de desenvolvimento de software de Ward & Mellor, fornecendo ao projetista uma visão mais concreta do sistema proposto, antes da fase de implementação.

O ProTR pertence à classe **"Evolutiva"** de protótipos, apresentada no Capítulo 2, considerando-se que a especificação através da metodologia "Desenvolvimento Estruturado para STR" permite a identificação gradual das necessidades do sistema, propiciando a adaptação e correção do protótipo gerado, à medida que o usuário e o projetista conheçam melhor a solução do problema. Diversas versões do protótipo podem ser produzidas e avaliadas mediante sua execução, permitindo determinar a viabilidade do sistema.

Nas próximas Seções serão fornecidas as bases para geração do protótipo a partir da descrição do sistema através dos dois componentes fundamentais fornecidos pela metodologia de Ward & Mellor, apresentada no Capítulo 4.

O Esquema de Transformações apresenta as conexões entre as transformações do sistema através de fluxos. O Dicionário de Dados descreve os elementos que compõem o Esquema de Transformações, apresentando os tipos de dados de cada fluxo e dos armazenadores, além das operações realizadas pelas transformações.

A partir desses dados, o código fonte executável é criado, implementando as tarefas do sistema alvo. A Figura 5.1 apresenta as principais fases existentes na implementação deste processo.

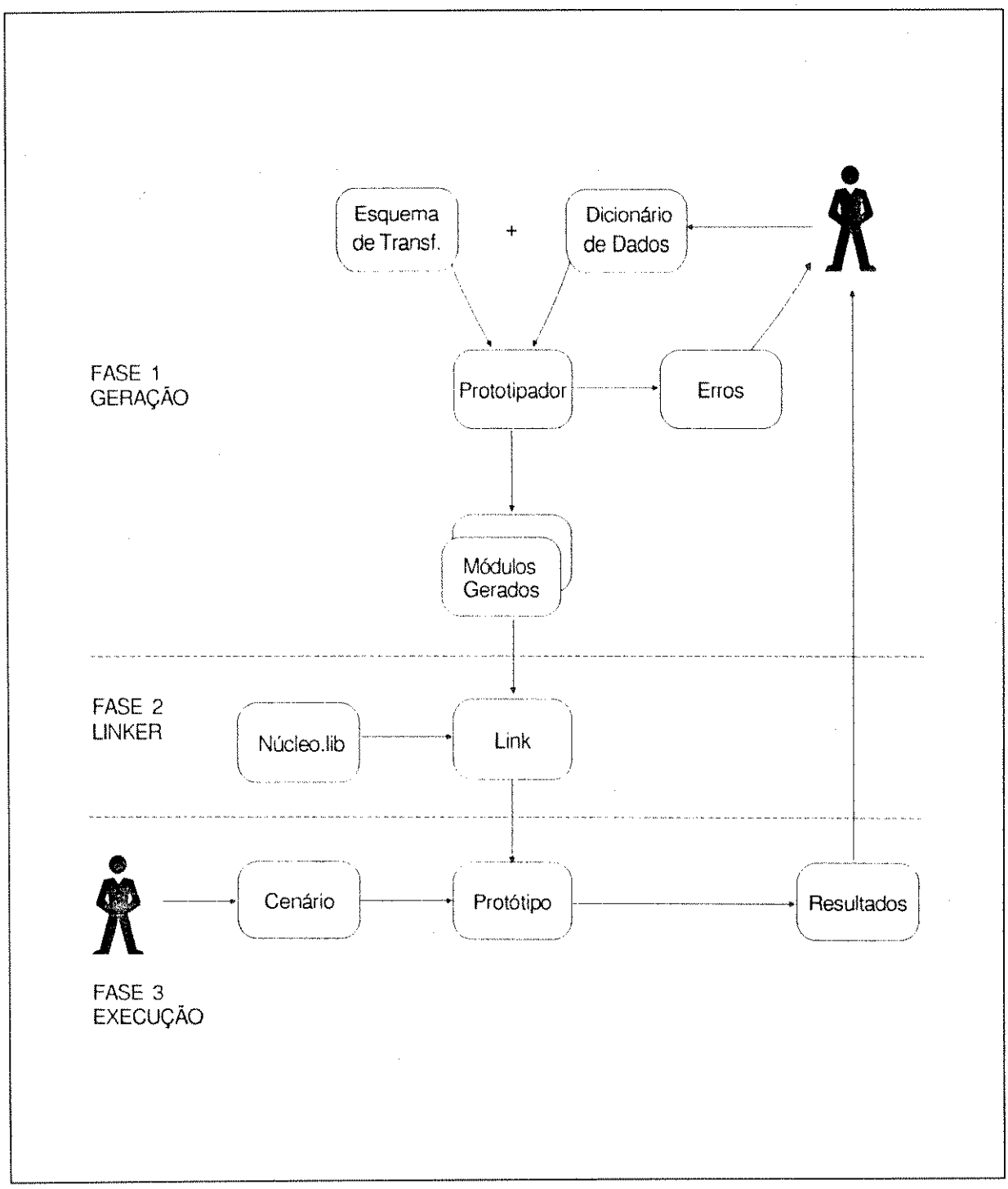


Figura 5.1. Utilização da Ferramenta.

Na **Fase 1**, a descrição do sistema é realizada com auxílio da metodologia "Desenvolvimento Estruturado para Sistemas de Tempo Real".

A descrição do Esquema de Transformações é realizada através de uma linguagem reconhecida por uma gramática ESLL(1) [SET83]. A especificação dessa linguagem pode ser obtida no Anexo C. Os componentes do diagrama são fornecidos ao prototipador que analisa sintaticamente a descrição, realiza testes de consistência semântica e valida o modelo do sistema. Nesta fase é gerada uma estrutura de dados com os componentes validados. O arquivo de descrição dos elementos do Esquema de Transformações é nomeado da seguinte forma : **<nome_sistema>.dfd**.

A descrição do Dicionário de Dados do sistema é realizada na própria linguagem de programação do programa a ser prototipado - linguagem C [KER78]. Este arquivo é denominado **<nome_sistema>.dic**.

O gerador do protótipo ProTR é ativado e, através da análise da estrutura de dados, gera um protótipo executável. Nesta fase, cada componente validado anteriormente tem suas características obtidas do Dicionário de Dados; essas características são analisadas e utilizadas na geração do protótipo. Durante esta fase, gera-se um arquivo de comandos **<nome_sistema>.bat**, para a compilação dos módulos gerados.

Na **Fase 2**, os módulos fontes protótipos gerados são ligados a um Núcleo de Tempo Real [AZE91a], especialmente desenvolvido para atender os requisitos da aplicação de prototipação. O arquivo executável, gerado nesta fase, é denominado **<nome_sistema>.exe**.

Na **Fase 3**, o usuário gera um Cenário que representa toda a sequência de eventos gerados para simular a interface com o ambiente. O Cenário é descrito através da Linguagem de Especificação de Cenários (LEC), que permite representar a cada instante o comportamento dos fluxos de entrada do sistema. O Cenário é definido em torno dos objetos do ambiente do sistema tais como, sensores, dispositivos e pessoas.

Finalmente, o protótipo gerado é executado e os resultados obtidos são armazenados num arquivo. O relatório com os "Resultados da Prototipação" consiste da apresentação do comportamento do sistema a partir da sequência das execuções das transformações a cada instante. Esta execução ocorre com a ativação do protótipo gerado através do seguinte comando : **<nome_sistema>.exe <nome_cenario>.cen.**

Alguns aspectos podem ser levantados quanto à sequência de fases imposta pela ferramenta. Esta divisão surge naturalmente se considerarmos uma das características mais importantes desta ferramenta : ser executável. Para alcançar este objetivo, limitado pelos recursos alocados ao projeto, optou-se pela utilização da linguagem **C** (soluções alternativas são consideradas na conclusão). As limitações impostas por esta escolha tornaram-se aparentes no decorrer do projeto. Uma das limitações mais sérias está associada à inexistência na linguagem de mecanismos que permitam o tratamento de processos concorrentes. Esta limitação impôs o desenvolvimento de um Núcleo de Tempo Real que fornecesse primitivas para garantir a sincronização/comunicação dos processos presentes no protótipo.

A utilização da linguagem **C**, associada à presença de um Núcleo de TR criou uma fase intermediária entre a descrição do sistema para o Prototipador (**Fase 1**) e a execução do mesmo pelo usuário (**Fase 3**). Nesta fase (**Fase 2**) os módulos em linguagem **C**, gerados automaticamente pelo prototipador, são compilados (compilador Microsoft C Versão 5.1) e ligados ao Núcleo para posterior execução.

Na Seção 5.2 é apresentada uma discussão a respeito da utilização do Modelo Essencial da metodologia "Desenvolvimento Estruturado para Sistemas de Tempo Real", na ferramenta ProTR.

A Seção 5.3 explora o sistema ProTR definindo sua interface com o usuário através dos seguintes tópicos : descrição dos elementos da metodologia, tratamento das prioridades das tarefas, descrição do Cenário e execução do protótipo.

5.2. Modelo Essencial X Modelo de Implementação.

A descrição da metodologia realizada por Ward & Mellor [WAR85] divide o processo de desenvolvimento do sistema em dois blocos denominados : Modelo Essencial e Modelo de Implementação.

Uma vez que o sistema deverá atuar num ambiente determinado, é possível determinar exatamente seu comportamento e quais dados serão manipulados através do Modelo Essencial.

A partir do Modelo Essencial é possível, através de transformações bem determinadas, adaptá-lo para a tecnologia particular utilizada na implementação; o resultado dessas transformações é denominado Modelo de Implementação (Figura 5.2).

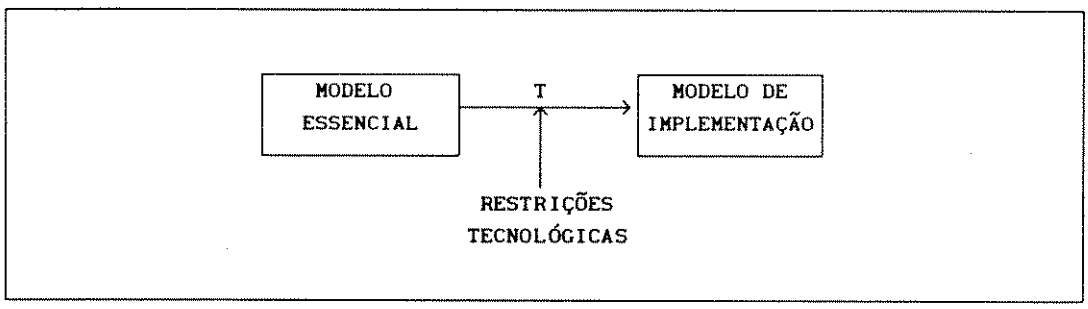


Figura 5.2. Modelo Essencial X Modelo de Implementação.

No Modelo de Implementação, a descrição do Modelo Essencial é filtrada de modo a obter os elementos básicos utilizados em computação : processador, tarefas e módulos.

Como o objetivo do prototipador é fornecer uma visão do comportamento do sistema para verificar sua funcionalidade, o modelo utilizado como referência no processo de prototipação é o Modelo Essencial, introduzindo o mínimo de distorções possíveis no protótipo obtido. Dessa forma, é necessário ressaltar que as técnicas de modelagem de processador, tarefas e módulos descritos na metodologia, para geração do Modelo de Implementação, não são consideradas na geração do protótipo.

5.3. Explorando o Sistema ProTR.

Nesta Seção são fornecidos elementos para permitir a execução do protótipo gerado e a observação dos resultados obtidos.

Basicamente a descrição do sistema para o prototipador é realizada por duas fontes distintas. A primeira descreve o Esquema de Transformações numa forma textual; neste caso, são apresentadas as conexões entre os diversos elementos que compõem a metodologia. Este tópico é coberto pela Seção 5.3.1.

A segunda descreve o Dicionário de Dados, onde é associada uma interpretação para cada elemento da metodologia. Esse tópico é coberto na Seção 5.3.2.

Uma vez que o sistema tenha sido descrito de forma apropriada para o prototipador, ele realiza verificações de consistência e gera o protótipo. De posse do protótipo, o usuário pode executá-lo, fornecendo cenários do sistema e analisando os resultados obtidos.

A Seção 5.3.3 apresenta o arquivo do Cenário que simula a interface do sistema com o meio ambiente, descrevendo a sequência de eventos que permitirá a execução do protótipo gerado a partir do "Esquema de Transformações" e do "Dicionário de Dados", descritos nas Sessões anteriores.

Na Seção 5.3.4 são apresentadas as prioridades atribuídas às transformações para execução do protótipo com auxílio de um Núcleo de Tempo Real.

Na Seção 5.3.5 é apresentado um roteiro para a execução do protótipo gerado a partir da descrição do sistema fornecida à ferramenta ProTR.

A seção 5.3.6 descreve o arquivo de resultados gerado durante a execução do protótipo.

Novamente, as descrições apresentadas são sucintas para não estender demasiadamente o trabalho. Os grafos sintáticos das linguagens utilizadas nas descrições são fornecidos nos anexos C, D e E.

5.3.1. Descrição do "Esquema de Transformações".

A descrição do diagrama para o prototipador é realizada através de uma linguagem reconhecida por uma gramática ESLL(1) [SET83], cujos grafos sintáticos são apresentados no Anexo C.

O arquivo de descrição dos elementos do "Esquema de Transformações" deve ser nomeado como segue : **<nome_sistema>.dfd**.

Nas próximas Seções serão apresentados informalmente os componentes do diagrama. Comentários podem ser inseridos em qualquer ponto da descrição, desde que sejam envolvidos por '/*' e '*/'.

5.3.1.1. Terminador.

Um Terminador é identificado por um nome composto por até 32 caracteres alfanuméricos. Na sua descrição são indicados os fluxos de entrada e saída. Não são permitidos :

- Nomes de Terminadores em branco;
- Nomes de Terminadores idênticos a de outros elementos do diagrama.

Sua descrição obedece a seguinte estrutura :

```
term <nome_terminador>;
 {
  entrada  {
    <classe_fluxo> <nome_fluxo>;
   }
  saida  {
    <classe_fluxo> <nome_fluxo>;
   }
 }
```

Basicamente são fornecidos o nome do elemento, seus fluxos de entrada e de saída, juntamente com o tipo. O símbolo **<classe_fluxo>** pode assumir os seguintes valores :

eve <nulo> - **fluxo de evento simples.**
eve trigger - **fluxo de evento com prompt trigger.**
eve enable - **fluxo de evento com prompt enable.**
eve disable - **fluxo de evento com prompt disable.**
con - **fluxo de dado contínuo.**
dis - **fluxo de dado discreto.**

Esta estratégia é adotada na descrição de todos os próximos elementos.

Exemplo :

```
term operador_linha;
 {
  entrada  {
    con estado_linha;
   }
  saida  {
    dis novo_tam_gar;
    eve liga;
    eve desliga;
   }
 }
```

5.3.1.2. Transformação.

Uma Transformação é identificada por uma referência numérica e por um nome composto por até 32 caracteres alfanuméricos. Na descrição da transformação são também indicados os fluxos de entrada e saída. Não são permitidos :

- Nomes de Transformações em branco;
- Nomes de Transformações idênticos a de outros elementos do diagrama, com exceção de "prompts" "**Enable**", "**Disable**" e "**Trigger**", associados à Transformação.

Sua descrição obedece a seguinte estrutura :

```
tran <nome_transformação> (<referência_transformação>);
 {
  entrada  {
    <classe_fluxo> <nome_fluxo>;
   }
  saida  {
    <classe_fluxo> <nome_fluxo>;
   }
 }
```

O campo "*referência_transformação*" indica o nível de hierarquia onde a transformação está posicionada na descrição do sistema.

Exemplo :

```
tran altera_pH (4);
 {
  entrada  {
    eve enable altera_pH;
    eve disable alteraqa_pH;
    dis novo_nivel_pH;
   }
  saida  {
    dis pH_referencia;
   }
 }
```

5.3.1.3. Armazenador.

Um Armazenador é identificado por um nome composto por até 32 caracteres alfanuméricos. Na descrição do armazenador são também indicados os fluxos de entrada e saída. Não são permitidos :

- Nomes de Armazenadores em branco;
- Nomes de Armazenadores idênticos a de outros elementos do diagrama.

Sua descrição obedece a seguinte estrutura :

```
arm <nome_armazenador>;
 {
  entrada {
    <classe_fluxo> <nome_fluxo>;
   }
  saida {
    <classe_fluxo> <nome_fluxo>;
   }
 }
```

Exemplo :

```
arm tam_atual_gar;
 {
  entrada  {
    dis tam_atual_gar;
   }
  saida  {
    dis tam_atual_gar;
   }
 }
```

5.3.1.4. Exemplo da Descrição do "Esquema de Transformações".

Para descrever o sistema completo para o ProTR, é necessário envolver os elementos que descrevem o diagrama com a seguinte estrutura :

```
sistema <nome_do_sistema>
     <elementos ...>
FIM
```

A seguir apresenta-se exemplo (Figura 5.3) de uma descrição, seguindo as estruturas apresentadas nas Seções anteriores.

O objetivo do sistema é calcular o valor atualizado de uma prestação com o auxílio de uma constante fixa de referência. Para descrever este sistema é necessário apresentar individualmente a transformação "correção" e o armazenador "referência" envolvidos pela estrutura que identifica o sistema "CALCULA_PRESTAÇÃO".

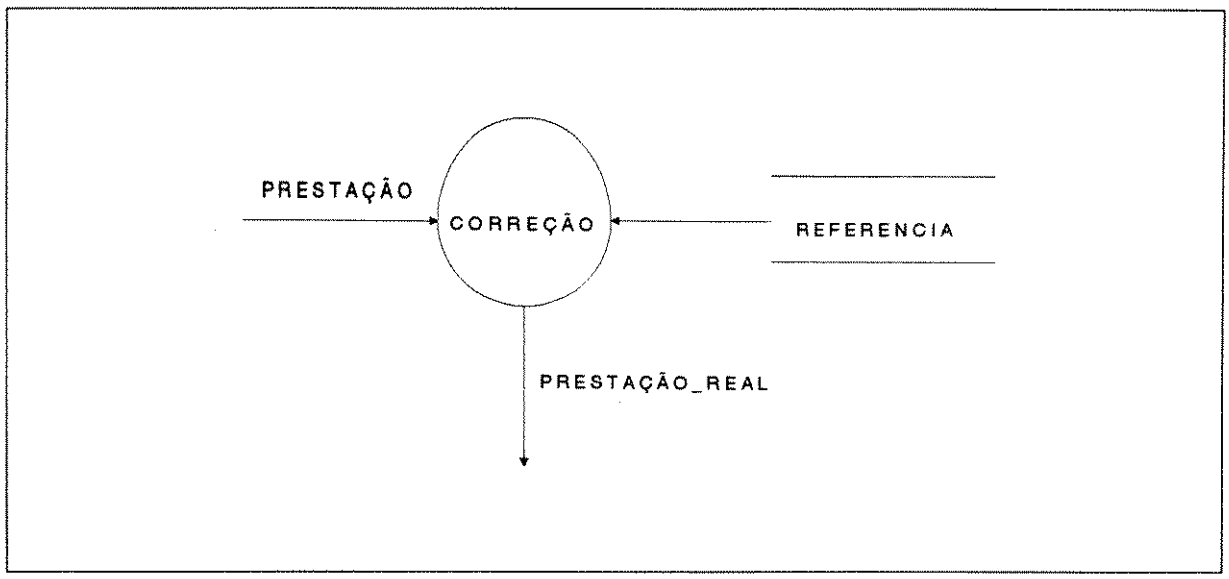


Figura 5.3. Diagrama do "Esquema de Transformações".

```
sistema CALCULA_PRESTAÇÃO

arm referência;   {
  saída   {
    dis referência;
   }
 }


tran correção (0);   {
  entrada   {
    dis prestação, referência;
   }
  saída   {
    dis prestação_real;
   }
 }

FIM
```

5.3.2. Descrição dos Elementos no Dicionário de Dados.

Para que o protótipo seja executável é necessário que as descrições dos componentes da metodologia obedeçam a uma linguagem de forma a garantir sua interpretação pelo prototipador.

O "Dicionário de Dados" descreve os elementos que compõem o Esquema de Transformações, apresentando os tipos de dados de cada fluxo e dos armazenadores, além das operações realizadas pelas transformações.

Com o objetivo de otimizar a geração do protótipo, a descrição formal dos elementos do Dicionário de Dados é realizada na própria linguagem do programa a ser prototipado - Linguagem "C" [KER78], com algumas extensões para representar a interface com os elementos da metodologia. Com isto, a linguagem de especificação é uma composição descrita através de grafos sintáticos, no Anexo D.

Os seguintes elementos da linguagem **C** não deverão ser utilizados na descrição do Dicionário de Dados pelos motivos apresentados abaixo :

- **Union** : não será permitido, pelo fato de introduzir ambiguidade na identificação do fluxo.

- **Pointers** : não serão usados pois permitem a "quebra" da estruturação da metodologia através do acesso direto a variáveis em outras transformações caminhando, dessa forma, em sentido oposto à base de metodologia, que introduz os fluxos de dados justamente para garantir um baixo acoplamento entre as transformações.

- **Bit Fields** : na declaração de um membro da estrutura, também não são permitidos pois não é fundamental economizar memória na geração do protótipo.

- **Modificadores** : (cdecl, far, fortran,...) não são permitidos pelo fato de serem específicas de uma implementação de compilador.

A Tabela 5.1 indica como é realizado esse modelamento.

TABELA 5.1. - DESCRIÇÃO DOS ELEMENTOS.

ELEMENTOS METODOLOGIA	DESCRIÇÃO
Fluxo de Dados	Através da definição do tipo em "C".
Fluxo de Evento	Não possui descrição (sem conteúdo).
Transformação de Dados	Função em "C".
Transformação de Controle	Tabela de Estados.
Armazenador de Dados	Através da definição do tipo em "C".
Armazenador de Evento	Não possui descrição (sem conteúdo).

Na descrição do corpo da transformação são utilizadas rotinas do Núcleo de Tempo Real. Estas rotinas, acessíveis ao usuário, são apresentadas no Anexo B.

A seguir apresenta-se a sintaxe utilizada para descrição de cada um dos elementos no "Dicionário de Dados".

```
fluxo <nome_fluxo>;
 {
  <tipo_fluxo>;
 }

arm <nome_armazenador>;
 {
  <tipo_armazenador>;
 }
```

O símbolo **<tipo_fluxo>** e **<tipo_armazenador>** descreve o tipo na linguagem "C", associado ao elemento **(int, float, etc)**.

```
tran discreto <nome_transformação> (<prioridade>,<refer>);
 {
  <corpo_função>
 }
```

```
tran contínua <nome_transformação> (<prioridade>,<período>,<refer>);
 {
  <corpo_função>
 }
```

O símbolo **<prioridade>** determina a prioridade na qual o processo associado à transformação deverá ser executado.

O símbolo **<refer>** determina o nível de hierarquia onde a transformação está situada na descrição do sistema.

O símbolo **<corpo_função>** representa o conjunto de comandos utilizados para implementar o algoritmo associado à transformação.

O símbolo **<período>** determina o período do processo associado às transformações contínuas.

```
tran controle <nome_transformação> (<prioridade>,<refer>);
 {
  est_inicial <nome_estado> : ação_inicial <classe_evento>
                              <nome_evento>;
  est_atual <nome_estado>;
    {
     (<nome_evento> <próximo_estado> : <classe_evento> <nome_evento>)
    }
    . . .
 }
```

O símbolo **<classe_evento>** determina o tipo do evento, e pode assumir os seguintes valores :

trigger - **fluxo de evento com prompt trigger.**
enable - **fluxo de evento com prompt enable.**
disable - **fluxo de evento com prompt disable.**
<nulo> - **fluxo de evento simples.**

A título de ilustração da sintaxe apresentada acima, seguem os seguintes exemplos, relacionados com o sistema apresentado no Anexo A :

```
fluxo novo_nivel_pH;
 {
  float novo_nivel_pH;
 }


arm pH_referencia;
 {
  float pH_referencia;
 }


tran discreto altera_pH (30,4);
 {
  pH-referencia = novo_nivel_pH;
  oswtds(pH_referencia);
 }


tran contínua reg_estado_reserva (12,15,2.2);
 {
  estado_reserva.pH_atual = pH_atual;
  estado_reserva.nivel_reserva = nivel_reserva;
  ossenfc(estado_reserva);          /* envia dado contínuo */
 {


tran controle controla_area (20,1);
 {
  est_inicial area_desabilitada : acao_inicial enable altera_pH;
  est_tual area_desabilitada;   {
    (habilita_area, area_habilitada : enable mantem_cond_reserva,
                                      disable altera_pH)
   }
  est_atual area_habilitada;   {
    (desabilita_area,area_habilitada : enable altera_pH,
                                       disable mantem_cond_reserva)
   }
 }
```

O arquivo de descrição do "Dicionário de Dados" deve ser nomeado como segue: **<nome_sistema>.dic**.

A seguir apresenta-se um exemplo da descrição dos elementos do Dicionário de Dados para o diagrama apresentado na Figura 5.3. Na descrição da transformação é utilizada uma função do núcleo de tempo real **"ossenfw"** que envia um fluxo discreto. A descrição desta função e de outras, acessíveis ao usuário, como já afirmado, é apresentada no Anexo B.

```
sistema CALCULA_PRESTAÇÃO


fluxo prestação;
 {
  long prestação;
 }


fluxo prestação_real;
 {
  long prestação_real;
 }


arm referencia;
 {
  struct
   {
    long desconto_basico;
    long limiar;
    long desconto_limiar;
   }
 }


trans discreto correcao(10,0);      /* prioridade 10, nivel 0 */
 {
  if(prestaçao > referencia_limiar)
    prestaçao_real = prestaçao * referencia.desconto_limiar;
  else
    prestaçao_real = prestacao * referencia.desconto_basico;
  ossenfw(prestaçao_real);
 }


FIM
```

5.3.3. Descrição do Cenário.

O Contexto do Sistema representa o nível mais alto na hierarquia do Esquema de Transformações. Basicamente, mantém todas as ligações do sistema com o ambiente; esta interface é representada através de um Cenário utilizado durante a execução.

O Cenário representa toda a sequência de eventos gerados para simular a interface com o ambiente. É descrito através da Linguagem de Especificação de Cenários (LEC) que permite representar a cada instante o comportamento dos fluxos de entrada do sistema. O Cenário é definido em torno dos objetos do ambiente do sistema tais como sensores, dispositivos e pessoas.

A descrição do Cenário obedece à seguinte sintaxe :

CENÁRIO ::= 'sistema' IDEN ':' ('evento' ¦ 'tempo') ';' DESC

DESC ::= ELEMENTO {ELEMENTO}* 'FIM'

ELEMENTO ::= 'NUMERO' ':' FLUXO

FLUXO ::= '{' CORPO {CORPO}* '}'

CORPO ::= (IDEN ':' ('(' VALOR ',' VALOR ')' ¦ VALOR) ¦ 'eve' IDEN)

VALOR ::= ('+' ¦ '-') NUMERO ['.' NUMERO]

IDEN :== CARACTER {CARACTER}*

Esta descrição obedece à sintaxe definida para expressões regulares estendidas **(ERE)** [SET83], cuja definição formal é :

a) Uma cadeia de símbolos **a** é uma ERE;

b) Se **a** e **b** são EREs, **ab** e **a¦b** são ERESs;

c) Se **a** é uma ERE, então **(a)**, **[a]**, **{a}*** são EREs;

d) Se **a** e **b** são EREs, **{a¦¦b}*** é uma ERE;

e) A expressão obtida pela aplicação de (b), (c) e (d) um número finito de vezes é uma ERE.

A interpretação dessas expressões é a seguinte :

(a) = a

[a] = (a ¦ vazio)

{a}* = vazio ¦ a ¦ aa ¦ ...

{a ¦¦ b}* = a{ba}*

É interessante observarmos que o símbolo **'(VALOR ,VALOR)'** permite descrever equações lineares do tipo **y = ax + b**, onde **a** e **b** são identificados com o elemento **VALOR** presente no lado direito da produção. O Grafo de Sintaxe da Gramática apresentada é descrito no Anexo E.

A execução do protótipo num ambiente controlado exige que determinados fluxos sejam ativados com as entradas presentes no sistema definitivo. Naturalmente, a determinação destes fluxos é direta, uma vez que os fluxos que interagem com o ambiente são aqueles conectados aos **Terminadores**.

Um outro ponto de interesse, é a determinação do **instante** no qual as entradas fornecidas pelo **Cenário** são consumidas. Numa primeira abordagem, um conjunto de fluxos é consumido e o próximo conjunto é consumido somente quando todas as transformações do sistema se encontram no estado desabilitado. Esta abordagem não alcançou os resultados esperados, pois dependendo do sistema, algumas transformações estão continuamente ativas. Na implementação atual, esta primeira abordagem foi substituída por duas estratégias distintas para definir o instante no qual as entradas do cenário são consumidas.

Na primeira estratégia, denominada "**Execução por Eventos**", cada conjunto de fluxo de entrada é consumido assim que o conjunto anterior tiver sido analisado.

Na segunda estratégia, denominada "**Execução por Tempo**", a cada conjunto de fluxo de entrada é associado um valor numérico que determina o instante em que o conjunto é consumido. Esse valor está associado ao relógio do Núcleo de Tempo Real; na implementação, com a utilização de sistemas IBM-PC, ocorrem 18,2 interrupções de relógio por segundo, ou seja, 1 "tick" a cada 54,94 ms.

Conforme descrito acima, conclui-se que o usuário deve ter a opção de escolher entre duas formas de comportamento durante a execução do protótipo: **tempo** e **evento**. Na execução por **tempo**, o usuário determina o momento exato no qual o conjunto de fluxos de entrada será consumido. Na execução por **evento**, cada conjunto de fluxos de entrada é consumido assim que o conjunto anterior for analisado.

Exemplo de descrição de Cenário (opção de execução - **Evento**), para o sistema descrito no Anexo A :

```
sistema Engarrafamento  :  evento;

0 : {
     novo_nivel_pH : 6.5;          /* inicializa nível do pH */
    }

1 : {
     eve habilita_area;            /* habilita area */
    }

2 : {
     pH_atual : 6.4;               /* valor corrente pH */
     nivel_reserva : 0.85;         /* nivel do reservatório */
    }
FIM
```

O arquivo de descrição do Cenário deve ser nomeado como segue : **<nome_cenário>.cen**.

Observe que o nome do arquivo do Cenário não está associado ao nome do sistema, permitindo, desta forma, que diversos Cenários sejam criados para execução do mesmo protótipo.

5.3.4. Alocando Prioridades.

Na geração do protótipo cada transformação é associada a uma tarefa cuja execução é coordenada pelo Núcleo de Tempo Real (NTR).

A maioria das implementações dos Núcleos de Tempo Real determinam a importância de cada tarefa de forma a poder selecionar, dentre as tarefas prontas para executar, a tarefa que ocupará o processador [MAG86]. Normalmente esta importância é determinada através de um número, inteiro e positivo, denominado prioridade. Na nossa implementação a prioridade de cada transformação é fornecida pelo usuário na descrição do sistema.

Como forma de orientação, vamos fornecer uma escala de prioridades, classificando as transformações em classes, conforme pode ser observado pela Tabela 5.2.

TABELA 5.2 - CLASSES DE TRANSFORMAÇÕES.

TRANSFORMAÇÃO	NÍVEL DE PRIORIDADE
Controle	Alta
Periféricas	Média Alta
Contínuas	Média
Com Prompt	Média Baixa
Simples	Baixa

As Transformações de Controle possuem o nível mais alto de prioridade. Isto se deve ao fato deste tipo de transformação ser responsável pela ativação/desativação de outras transformações, realizando, desta forma, uma gerência do sistema.

Transformações Periféricas são as que estão mais próximas do ambiente do sistema, ou seja, recebem e enviam fluxos para o ambiente. Desta forma, elas também são prioritárias para agilizar a troca de informações com o ambiente.

As Transformações contínuas constituem a próxima classe pelo fato de possuirem um tempo limite para o seu término, que é determinado pela próxima ativação periódica.

As Transformações com "Prompt" são mais prioritárias que as Transformações Simples por estarem mais próximas aos níveis de controle do sistema.

5.3.5. Sequência de Execução.

Uma vez que o protótipo tenha sido gerado, ele é passível de execução pelo usuário. Esta execução é coordenada por um Núcleo de Tempo Real denominado **NProTR**, em plataformas compatíveis com ambientes IBM/PC. O Núcleo **NProTR** foi especialmente desenvolvido para esta aplicação; sua base é originária de um conjunto de núcleos desenvolvidos para aplicações de tempo real [AZE91c].
A execução do código protótipo tem como objetivo proporcionar :

- teste e validação da estrutura de controle do sistema.

- análise de dimensionamento do sistema em termos de execução e análise de alocação de recursos disponíveis ao sistema.

- relatórios estatísticos para acompanhamento da dinâmica do sistema.

Uma vez iniciada a execução, esta segue normalmente até o fim da leitura dos dados do cenário, ou até uma interrupção do usuário (CTRL_X). Caso ocorra uma interrupção pelo usuário, estará disponível a tela de opções fornecida na Figura 5.4, permitindo uma consulta aos diversos relatórios de acompanhamento da execução naquele instante, ou que se dê continuidade à execução.

```
INTERFACE HOMEM x MÁQUINA  ProTR        Tempo = xxxxxx

1-  Estado das Transformações.
2-  Estado dos Fluxos Discretos.
3-  Estado dos Fluxos Contínuos.
4-  Estado dos Eventos.
5-  Estado dos Armazenadores de Dados.
6-  Estado dos Armazenadores de Eventos.
7-  Continuar Execução do Protótipo.
8-  Encerrar Execução do Protótipo.

ENTRE COM A OPÇÃO :
```

Figura 5.4. Menu de Opções na Execução.

A seguir apresentam-se exemplos destes relatórios :

1 - Estado das Transformações :

NOME TRANSF.	PID	ESTADO
controla_area	005	Trans aguardando evento
altera_pH	006	Trans aguardando fluxo discreto
reg_estado_reser	008	Trans dormindo
controla_entrada	009	Trans pronta para executar
estado_liga	014	Trans suspensa

2 - Estado dos Fluxos Discretos :

NOME FLUXO DISCRETO	PID SUSPENSO
novo_nivel_pH	006

3 - Estado dos Fluxos Contínuos :

NOME FLUXO CONTINUO	DADO PRESENTE	VALOR
pH_atual	SIM	6.4
nivel_reserva	SIM	0.85
estado_linha	NAO	

4 - Estado dos Eventos :

NOME EVENTO	EVENTO
habilita_area	AUSENTE
liga	AUSENTE
libera_garrafa	PRESENTE

5 - Estado dos Armazenadores de Dados :

NOME ARMAZENADOR	DADO PRESENTE	VALOR
pH_referencia	SIM	6.5
tam_atual_gar	NAO	
estado	NAO	

6 - Estado dos Armazenadores de Eventos :

NOME ARMAZENADOR	EVENTO PRESENTE
estado_válvula	PRESENTE

Como pode ser observado, a interpretação do **Menu de Opções** é direta. Merecem destaque somente os seguintes pontos.

- O campo **PID** ("Process identification") representa um identificador interno utilizado pelo Núcleo.

- O campo **valor** somente será preenchido quando o tipo do fluxo não for complexo (como "struct", por exemplo).

A sequência de execução do protótipo pode ser observada pelo algoritmo apresentado na Figura 5.5.

```
protótipo( )

  ciclo = 0;
  abrir "arquivo do cenário"
  abrir "arquivo de resultado"

  enquanto "arquivo de cenário não acabou"
    ler "cenário correspondente ao ciclo atual"
    atualizar fluxos de entrada
    executa até estabilizar
    ciclo = ciclo + 1

  fechar "arquivo de cenário"
  fechar "arquivo de resultado"
```

Figura 5.5. Algoritmo de Execução.

Como pode ser observado, o algoritmo de execução é extremamente simples; o único ponto que merece destaque é o comando "**executa até estabilizar**". Este comando está associado à estratégia utilizada para se determinar o instante no qual o próximo conjunto de entradas do arquivo de cenário será consumido. Conforme descrito anteriormente a execução do protótipo pode ser de dois tipos : **evento** e **tempo**.

Na execução por **evento**, os dados fornecidos pelo cenário são associados aos fluxos correspondentes gerando entradas para as transformações. A próxima leitura do cenário é ativada quando todos os fluxos fornecidos anteriormente são consumidos.

Na execução por **tempo**, o relógio de tempo real presente no núcleo é utilizado como referência para realizar a próxima leitura. Sempre que o valor do relógio se iguala com o fornecido no **cenário**, a leitura é realizada.

Um aspecto importante na execução do protótipo refere-se à **inicialização do sistema**, ou seja, o comportamento inicial do sistema difere do comportamento em regime. Os fluxos contínuos ainda não receberam dados, os armazenadores não contêm valores válidos e o estado das transformações é indefinido.

Nesta situação, o núcleo NProTR deve garantir que o protótipo permaneça num estado confiável até receber as entradas apropriadas. A relação abaixo representa o comportamento inicial dos elementos da metodologia :

- **Fluxo contínuo** - Não possui valor; as transformações contínuas associadas são suspensas até a presença de entradas.

- **Armazenador** - Não possui valor, caso alguma transformação execute uma leitura, é suspensa até a presença de dados.

- **Estado das Transformações** - Suspensas até que todas as condições para sua execução estejam satisfeitas.

5.3.6. Resultados da Prototipação.

Como afirmado na Seção 5.1, com a execução do protótipo é gerado um arquivo com os resultados da execução. O relatório presente neste arquivo consiste da apresentação do comportamento do sistema a partir da sequência das execuções das transformações a cada instante.

O arquivo **"Resultado da Execução"** do protótipo é gerado durante a execução e contém informações que permitem ao usuário verificar se as excitações provocadas pelo arquivo de cenário têm o tratamento apropriado pelo sistema. O arquivo de resultado é composto de três partes : **identificação**, **iniciação** e **corpo**.

Na **identificação**, apresenta-se o nome do arquivo de Cenário, que foi fornecido como entrada ao protótipo, juntamente com a data e a hora da execução. O formato do nome do arquivo de resultado obedece à seguinte sintaxe : **<nome_cenario>.sai**. A associação com o nome do arquivo do cenário permite rapidamente identificar o cenário associado à execução.

Na segunda parte, é realizada a **iniciação** de todas as transformações presentes no sistema, onde basicamente estas transformações atingem um estado estável e passam a aguardar fluxos/eventos que disparem sua execução.

A terceira parte está associada às entradas do cenário e corresponde ao **corpo** do arquivo de resultados. A cada entrada do cenário, é apresentada toda a atividade gerada a partir da mesma. Dois níveis de detalhes podem ser selecionados pelo usuário. No primeiro, somente os fluxos conectados com **terminadores** são apresentados; desta forma, o usuário pode observar o comportamento do protótipo através de suas interfaces com o ambiente. No segundo nível, todas as transações de fluxos/eventos e ativações de transformações são apresentadas. Neste nível, associado a cada fluxo, existe um identificador que acompanha o fluxo em todo o seu percurso dentro do sistema; todos os fluxos do primeiro nível também são apresentados, sendo identificados por dois caracteres "*".

Em cada atividade apresentada é também indicado o momento de sua ocorrência através do valor de um relógio que é incrementado a cada "tick" do sistema.

O formato do Relatório de Execução do diagrama da Figura 5.3. terá o aspecto apresentado a seguir.

RESULTADO DA EXECUCAO DO CENARIO Prestação

DATA : 29/01/93 HORA : 10:06:20

000000 Trans correcao aguardando habilitacao

Realizando leitura do cenário do tempo 10

000010 Trans oslecenario gerou fluxo discreto pretacao id=0
000010 Trans oslecenario gerou fluxo discreto prestacao valor=300
000010 Trans correcao recebeu fluxo discr prestacao id=0
000010 Trans correcao recebeu dado do arm referencia id=0
000011 Trans correcao gerou fluxo discr prestacao_real id=0 valor = 900

. . .

Na 1ª linha identifica-se o arquivo de cenário "Prestação", que foi fornecido como entrada ao prototipador, além da data e a hora da execução.

Na **iniciação** do arquivo é informado que a transformação "correção" está aguardando "habilitação" para executar. No caso desta transformação, a única condição de "habilitação" é a presença de dados no fluxo "prestação".

No **corpo** do arquivo de resultados ocorre a leitura de dados do arquivo de cenário juntamente com o consumo, pela transformação "correção", do fluxo "prestação" e do dado do armazenador "referência". Esse consumo ocorre no instante 10 de execução sendo o identificador do fluxo "prestação" igual a 0 e do armazenador "referência" igual a 0. Finalmente ocorre a ativação da transformação "correção" gerando o fluxo discreto "prestação_real" com valor igual a 900.

Na especificação do arquivo resultado da execução, um ponto que necessitou revisões foi a determinação dos fluxos que compõem o resultado da execução. Inicialmente, somente os fluxos de saída associados a Terminadores seriam mapeados. Esta aproximação é interessante para o usuário; entretanto, em caso de problemas, o analista teria dificuldade em localizar o erro somente com este elemento. Na implementação atual, todos os fluxos podem fazer parte do resultado da execução.

5.4. Características do Prototipador ProTR.

Na Seção 2.3 são levantadas as características desejáveis num ambiente de prototipação. Estas características serão revistas, levando em consideração os resultados alcançados com o ProTR.

- **Rapidez na Geração do Protótipo** - O ProTR permite a geração de código num tempo curto a partir da leitura dos requisitos do sistema fornecidos com a utilização da metodologia "Desenvolvimento Estruturado para STR".

- **Facilidade da Implementação de Alterações** - As alterações são rapidamente assimiladas pelo prototipador; o usuário necessita somente alterar a definição dos requisitos e executar novamente o prototipador.

- **Interface Amigável** - A interface com o usuário é um ponto crítico na ferramenta proposta. A entrada textual através de uma linguagem rígida exige treinamento do analista / usuário para execução do protótipo. Este ponto é reconhecidamente falho e passível de otimizações como descrito na conclusão.

- **Geração Automática de Código** - Partindo da descrição das transformações fornecida pelo usuário, o prototipador gera automaticamente o código fonte, liberando o usuário do ônus de se preocupar com fluxos de entrada e implementação de autômatos para representar aspectos de controle presentes na metodologia.
  Uma vez que analista e usuário estejam satisfeitos com o protótipo obtido, o código gerado pode ser utilizado como base na construção do sistema definitivo. Enquanto protótipo, toda a interface com o ambiente é realizada através da leitura do arquivo de cenário e pela geração do arquivo de resultados; estes arquivos simulam os terminadores no diagrama de contexto da metodologia. Naturalmente, o sistema definitivo deve interagir diretamente com os elementos do ambiente; dessa forma, o analista deve gerar tarefas que representem os terminadores gerando os fluxos originados nos mesmos, agora a partir de dados concretos obtidos no

ambiente. As novas tarefas criadas devem utilizar as primitivas do núcleo e ser ativadas periodicamente ou por interrupção. O núcleo NProTR pode ser utilizado como base para execução do sistema definitivo caso a máquina alvo seja similar ao sistema IBM-PC; nesse caso, o núcleo NProTR pode ser otimizado retirando-se as tarefas que lêem/geram dados nos arquivos de cenário/resultados. Um núcleo alternativo pode também ser utilizado desde que ofereça a mesma sintaxe e funcionalidade do núcleo NProTR.

- **Ambiente Integrado** - O sistema ProTR não representa um ambiente integrado desenvolvido (editores de texto, compiladores, etc). Apesar disto, os recursos necessários para sua utilização são limitados e restringem-se ao sistema operacional DOS, editor de texto e compilador para a linguagem **C**.

- **Protótipo Executável** - O protótipo fornecido pela ferramenta é executável produzindo ainda, a cada instante, um "mapa" de comportamento totalmente integrado aos termos utilizados no ambiente do usuário.

Capítulo 6. IMPLEMENTAÇÃO DO ProTR.

Neste capítulo são apresentados detalhes da implementação do prototipador ProTR. A descrição está dividida em duas Seções. A primeira apresenta a estratégia empregada para transformar os elementos abstratos da metodologia em elementos concretos, descritos através de uma linguagem de programação e passíveis de execução. Na segunda Seção, Estrutura do Software, é descrito "como" a transformação descrita acima é realizada, realçando as estruturas de dados utilizadas na implementação.

6.1. Prototipação dos Componentes.

Nesta Seção é descrita a estratégia utilizada na prototipação realizada de cada componente da metodologia "Desenvolvimento Estruturado para Sistemas de Tempo Real". A ênfase será dada aos aspectos de implementação de cada componente, visto que a descrição da semântica dos mesmos, já foi realizada no Capítulo 5. Na descrição da sequência de código gerada para cada componente, procurou-se apresentar de forma global a idéia da geração, sem entrar em detalhes específicos da linguagem C.

6.1.1. Transformação de Dados Discretos.

Na descrição dos elementos da metodologia, a presença de dados nos fluxos discretos de entrada de uma transformação implica sua ativação imediata. No mapeamento da metodologia, a ativação imediata é inviável pois exigiria a existência de um processador para cada transformação presente no sistema. Para modelar esta situação, os fluxos discretos foram implementados como uma fila FIFO; o número de elementos presentes na fila FIFO pode ser configurado pelo usuário na instalação do Núcleo de Tempo Real NProTR. No caso de estouro desta fila, o núcleo apresenta uma mensagem de erro.

A representação de transformação de dados discretos será descrita através das três classes de transformações definidas pela ferramenta.

6.1.1.1. Transformação Simples.

A Transformação Simples é aquela que recebe um conjunto de fluxos de entrada **(fi1,..., fin)**, executa uma sequência de operações e gera um conjunto de fluxos de saída **(fo1, ..., fom)** (Figura 6.1).

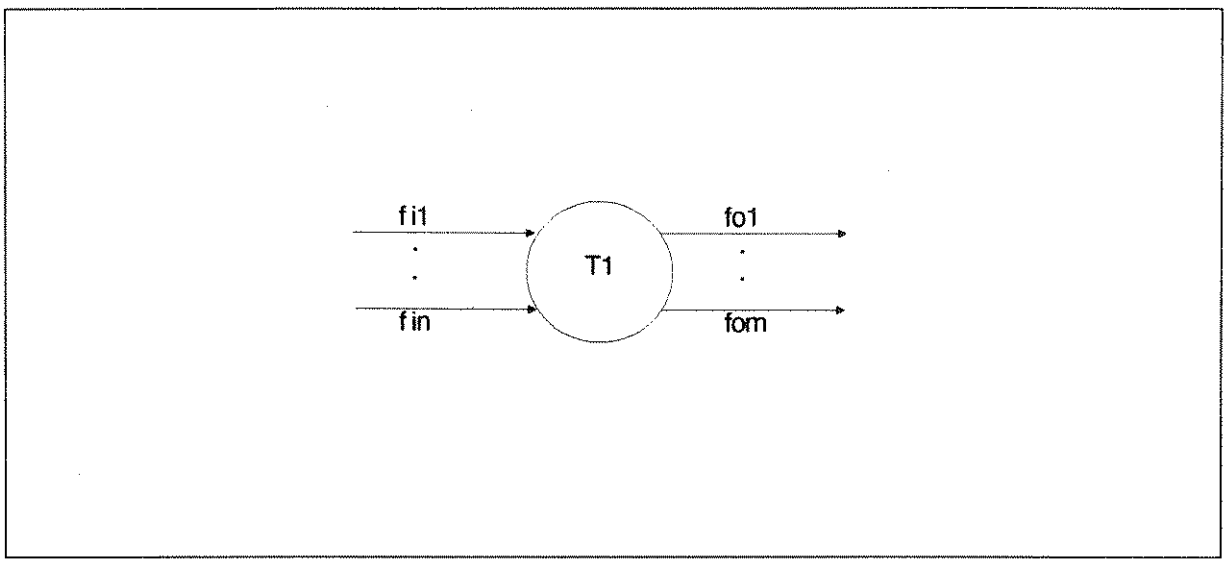


Figura 6.1. Transformação Simples.

A condição para execução dessa transformação é simplesmente a chegada de todas as instâncias dos fluxos de entrada.

A transformação a ser efetuada está descrita na Estrutura de Dados e, para efeito de representação, será denominada **TRANS_Ti**. Essa aproximação será seguida por todos os exemplos apresentados neste capítulo.

A Prototipação desse elemento é efetuada através da utilização de uma primitiva do Núcleo (**Receive_Flow**) que simplesmente recebe os fluxos de entrada e os armazena no caso da transformação não os ter ainda requisitado, implementando um mecanismo semelhante às estruturas "queues" descritas por Ward & Mellor.

Desta forma, o prototipador gera a seguinte sequência :

```
TASK_T1( )
 {
  tipo fi1;
  tipo fi2;
    ...
  tipo fin;
  for(;;)
   {
    Receive_flow (iden1,&fi1);
    Receive_flow (iden2,&fi2);
           .
           .
    Receive_flow (idenn,&fin);
    TRANS_T1;
   }
 }
```

Nesta sequência "**tipo**" representa o tipo da informação associada ao fluxo (int, float, etc); "**ideni**" representa um identificador interno ao núcleo de tempo real NProTR, este elemento é gerado automaticamente pelo prototipador. Os elementos símbolos "**tipo**" e "**ideni**" serão utilizados nas sequências apresentadas nos próximos ítens.

Note que os fluxos de saída **(fo1 ... fom)** não foram explicitamente representados na **TASK_T1()**; em princípio, sua geração está inserida dentro de **TRANS_T1**. Naturalmente, os fluxos de saída poderiam ser mantidos em estruturas internas e liberados ao final da tarefa; todavia, essa aproximação retarda a execução de outras transformações (possivelmente mais prioritárias) sem necessidade.

6.1.1.2. Transformação com "ENABLE" / "DISABLE".

Esta transformação apresenta fluxos de evento que comutam uma transformação entre os estados habilitado e desabilitado, denominados "Prompt" (Figura 6.2).

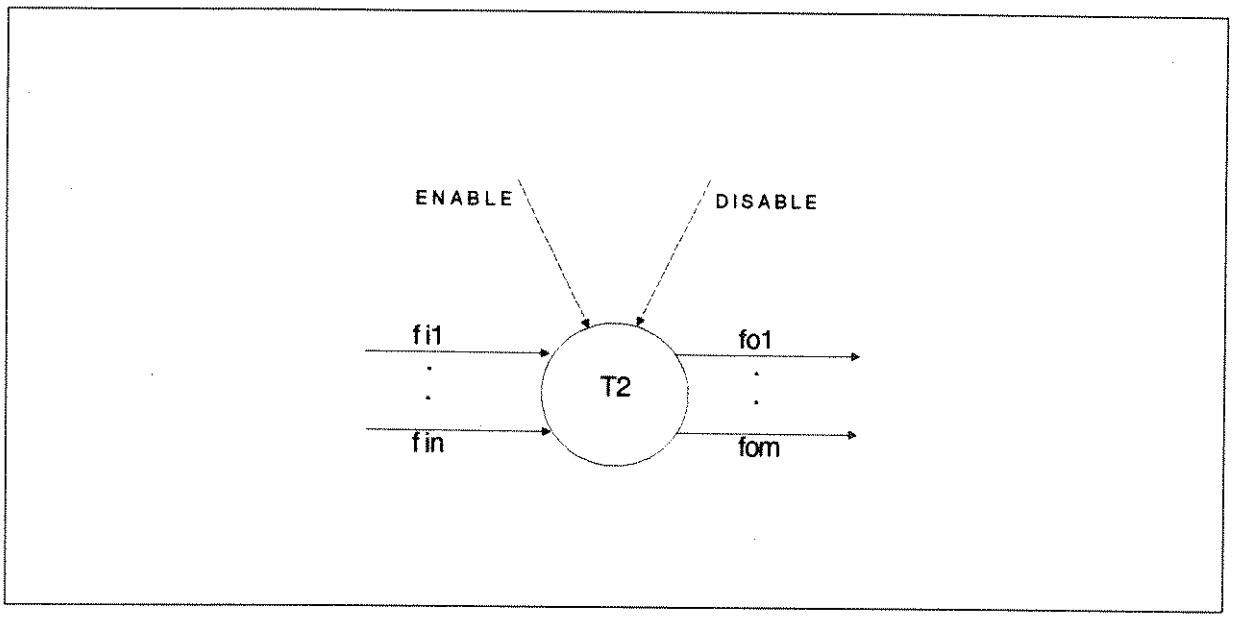


Figura 6.2. Transformação com "Enable" / "Disable".

A implementação desta transformação segue o procedimento descrito para a transformação simples sendo que, além da condição da presença de uma instância nos seus fluxos de entrada, é necessário que o "**Prompt**" "**Enable**" seja satisfeito, para tornar a transformação habilitada.

Se a transformação estiver no estado "**Desabilitada**", os dados disponíveis em seus fluxos de entrada serão descartados.

O Prototipador gera a seguinte sequência :

```
TASK_T2( )
 {
  tipo fi1;
  tipo fi2;
    ...
  tipo fin;
  for(;;)
   {
    Wait_enable( );
    Receive_flow(iden1,&fi1);
    Receive_flow(iden2,&fi2);
            .
            .
    Receive_flow(idenn,&fin);
    TRANS_T2;
   }
 }
```

6.1.1.3. Transformação com "Trigger".

A Transformação é habilitada pelo "**Prompt Trigger**" (Figura 6.3) permitindo uma execução, desde que exista dado disponível. Enquanto o "**Prompt Trigger**" não ocorrer, os dados presentes na entrada são descartados.

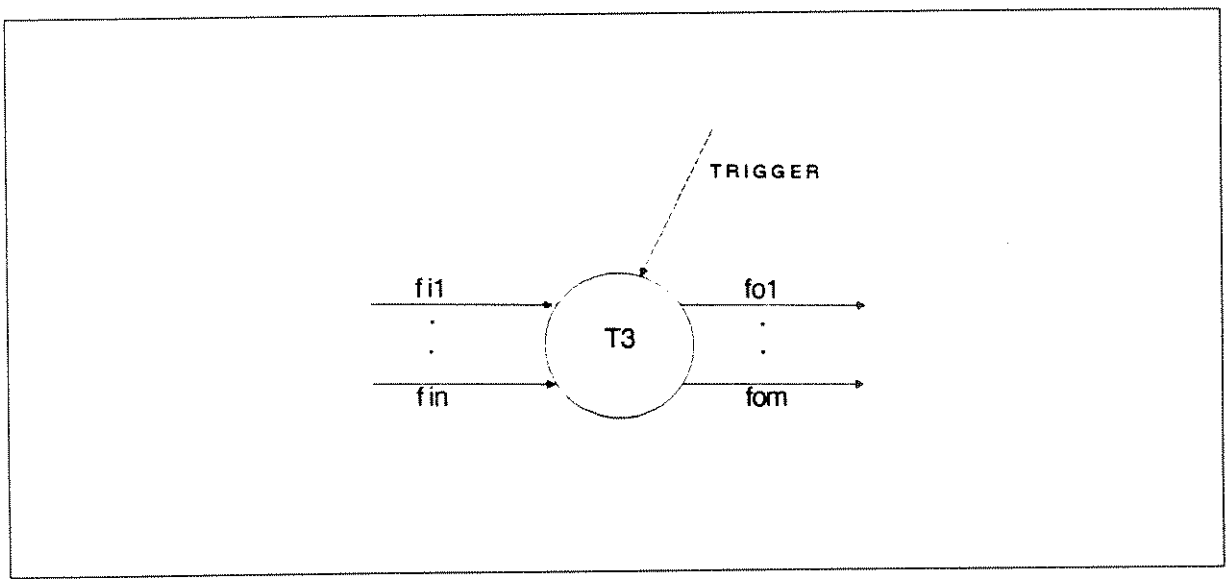


Figura 6.3. Transformação com "Trigger".

A execução desta transformação segue o procedimento descrito para transformação simples.

O Prototipador gera a seguinte sequência :

```
TASK_T3( )
 {
  tipo fi1;
  tipo fi2;
     ...
  tipo fin;
  for(;;)
   {
    Wait_trigger( );
    Receive_flow(iden1,&fi1);
    Receive_flow(iden2,&fi2);
            .
            .
    Receive_flow(idenn,&fin);
    TRANS_T3;
   }
 }
```

6.1.2. Transformação de Dados Contínuos.

6.1.2.1. Transformação Contínua Simples.

A transformação de dados contínuos (Figura 6.4) recebe um conjunto de fluxos de entrada contínuos **(fi1, ..., fin)**, executa uma sequência de operações e gera um conjunto de fluxos de saída que podem ser discretos ou contínuos **(fo1, ..., fom)**.

Os dados de entrada contínuos existem a todo instante dentro de um intervalo de tempo.

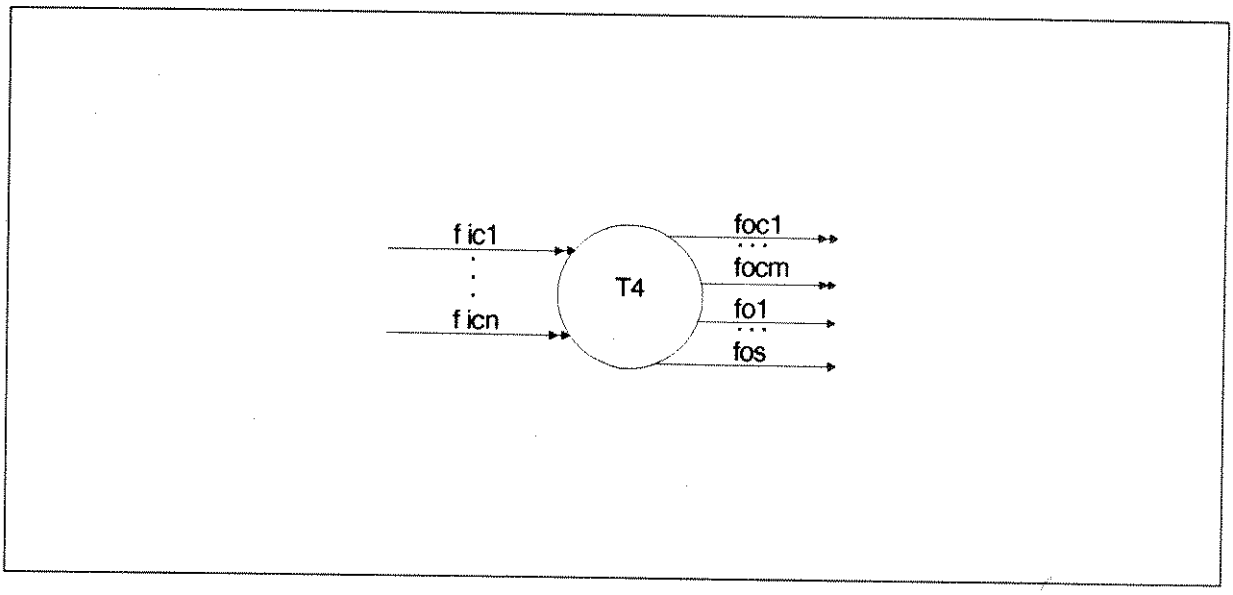


Figura 6.4. Transformação de Dados Contínuos.

A execução deste elemento ocorre a partir de uma primitiva do Núcleo de Tempo Real (**Receive_Flow_Continuous**) que recebe os fluxos de entrada contínuos, utilizando sempre o dado atual e descartando os anteriores.

A Prototipação gera a seguinte sequência :

```
TASK_T4( )
 {
  tipo fi1;
  tipo fi2;
    ...
  tipo fin;
  Receive_flow_continuous(iden1,&fi1);
  Receive_flow_continuous(iden2,&fi2);
                .
                .
  Receive_flow_continuous(idenn&fin);
  TRANS_T4;
 }
```

A transformação contínua será representada no protótipo através de uma tarefa periódica. O período de ativação será fornecido na descrição da tarefa pelo usuário.

6.1.2.2. Transformação Contínua com "Enable" / "Disable".

Nesta Seção é tratada a transformação contínua que apresenta "**Prompts**" "**Enable**" e "**Disable**", permitindo a comutação entre os estados "**Habilitado**" e "**Desabilitado**" (Figura 6.5).

Sua execução segue a descrita para a Transformação Contínua Simples sendo que, além da presença do dado periodicamente, é necessário que o "**Prompt Enable**" seja ativado, tornando a transformação habilitada.

Se a transformação estiver no estado "**Desabilitado**", os dados contínuos disponíveis serão descartados.

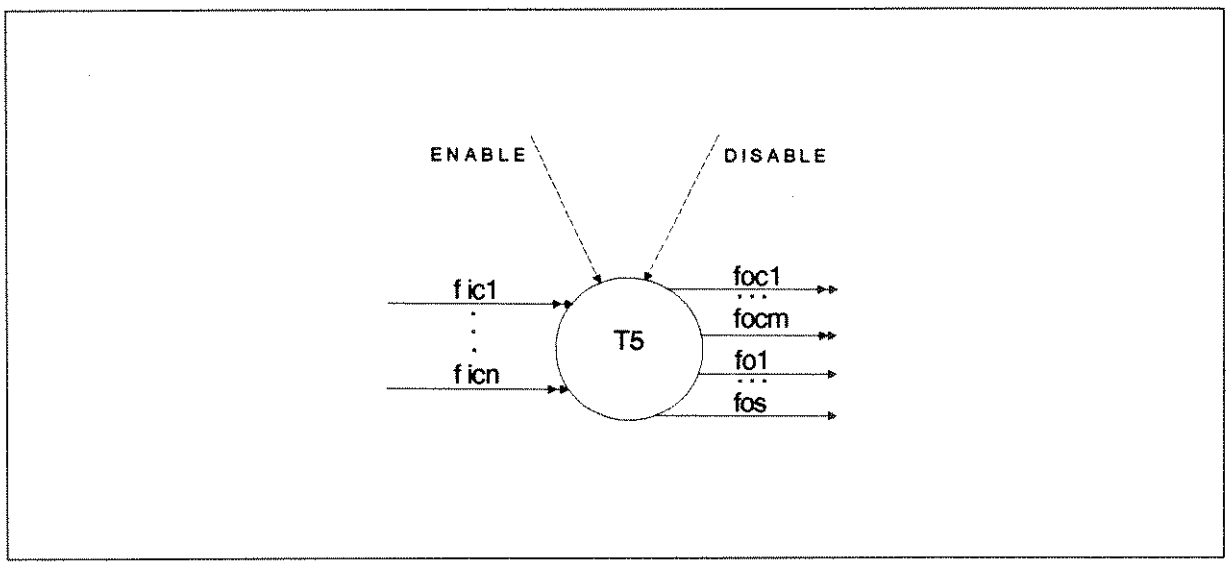


Figura 6.5. Transformação Contínua com "Enable" / "Disable".

A Prototipação gera a seguinte sequência :

```
TASK_T5( )
 {
  tipo fi1;
  tipo fi2;
    ...
  tipo fin;
  Wait_enable( );
  Receive_flow_continuous(iden1,&fi1);
  Receive_flow_continuous(iden2,&fi2);
                .
                .
  Receive_flow_continuous(idenn,&fin);
  TRANS_T5;
 }
```

Da mesma maneira como é feito com Transformações Contínuas Simples, a tarefa que representa a transformação será modelada através de tarefas periódicas, com o período fornecido pelo usuário.

6.1.2.3. Transformação Contínua com "Trigger".

A transformação contínua é habilitada pelo Prompt "**Trigger**" (Figura 6.6) permitindo uma execução com os dados disponíveis.

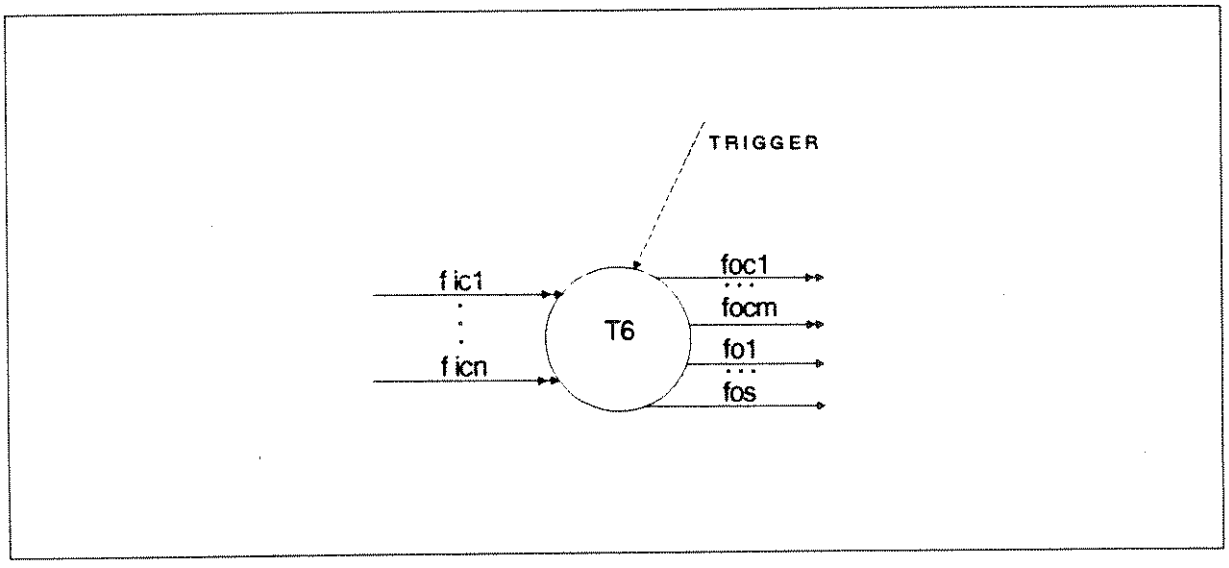


Figura 6.6. Transformação Contínua com "Trigger".

A Prototipação gera a seguinte sequência :

```
TASK_T6( )
 {
  tipo fi1;
  tipo fi2;
    ...
  tipo fin;
  for(;;)
   {
    Wait_Trigger( );
    Receive_flow_continuous(iden1,&fi1);
    Receive_flow_continuous(iden2,&fi2);
                  .
                  .
    Receive_flow_continuous(idenn,&fin);
    TRANS_T6;
   }
 }
```

Neste caso, a execução segue um esquema diverso das transformações contínuas anteriores. A tarefa que representa esta transformação é implementada como uma tarefa comum que aguarda a habilitação do "trigger" para executar.

6.1.3. Transformação de Controle.

6.1.3.1. Transformação de Controle Simples.

Na descrição dos elementos que compõem a metodologia, existem alguns fluxos que não possuem conteúdo, são simplesmente sinalizadores que indicam que algo ocorreu. Estes fluxos são denominados "Fluxos de Eventos".

A transformação que aceita fluxos de eventos como entrada e produz fluxos de eventos como saída, é denominada "Transformação de Controle" (Figura 6.7).

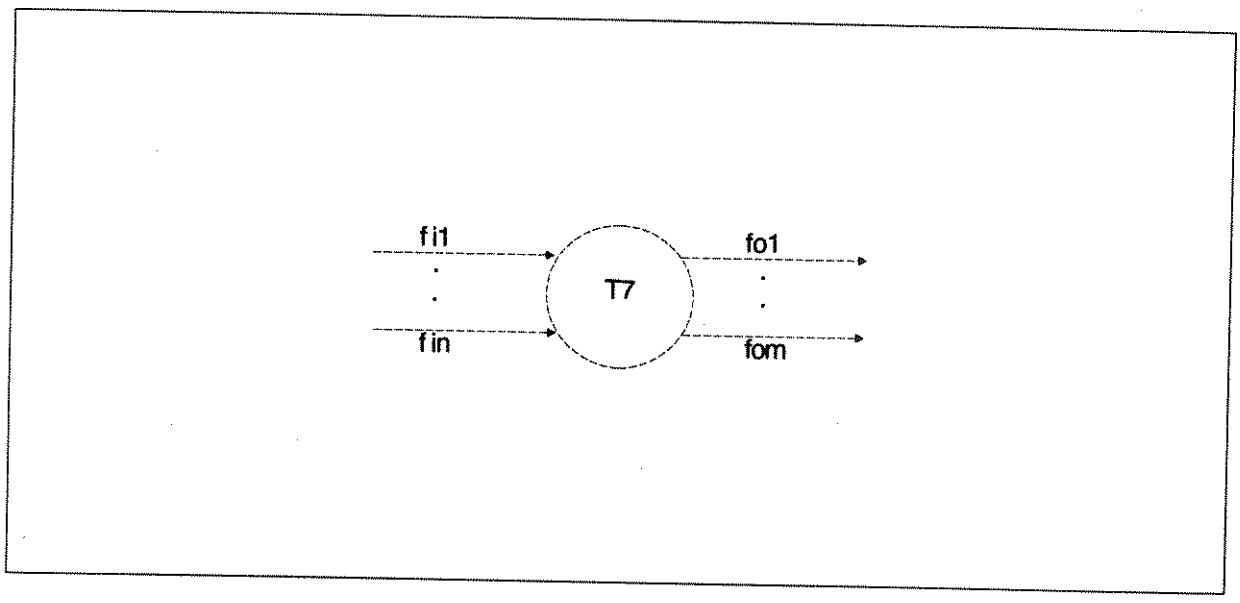


Figura 6.7. Transformação de controle.

Cada fluxo de evento de entrada é completamente processado antes que o próximo possa ser processado.

O modelo utilizado para descrever o comportamento da transformação de controle é o "Diagrama de Transição de Estados" [HOP79] que apresenta a sequência de entradas e saídas (Figura 6.8).

Os componentes representados neste diagrama são :

- **Estado** : representação do sistema num dado instante.
- **Transição** : representação de mudança de um estado para outro.

As Condições e Ações são associadas às transições.

- **Condições** : são eventos que permitem que o sistema realize uma transição. A condição é identificada com um fluxo de evento de entrada, sinalizando que a condição ocorreu.

- **Ações** : são realizadas quando as transições ocorrem.

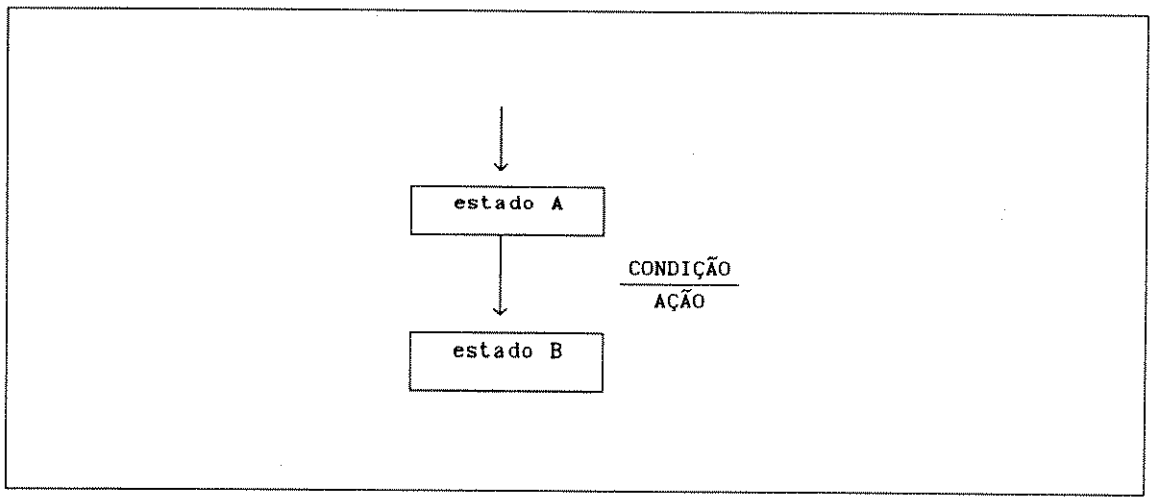


Figura 6.8. Diagrama de transição de estado.

A implementação do autômato se dará através de uma Tabela de Estados que será pesquisada a cada evento [SET83]. A tabela associada ao autômato da Figura 6.9 possui o aspecto descrito na Tabela 6.1.

TABELA 6.1. TABELA DE AUTÔMATO FINITO.

Estado Atual	Entrada	Próximo Estado	Ação
S0	A	S1	A0
	B	S2	A1
	C	S0	A2
S1	E	S0	A4
	F	S2	A5
S2	D	S1	A3

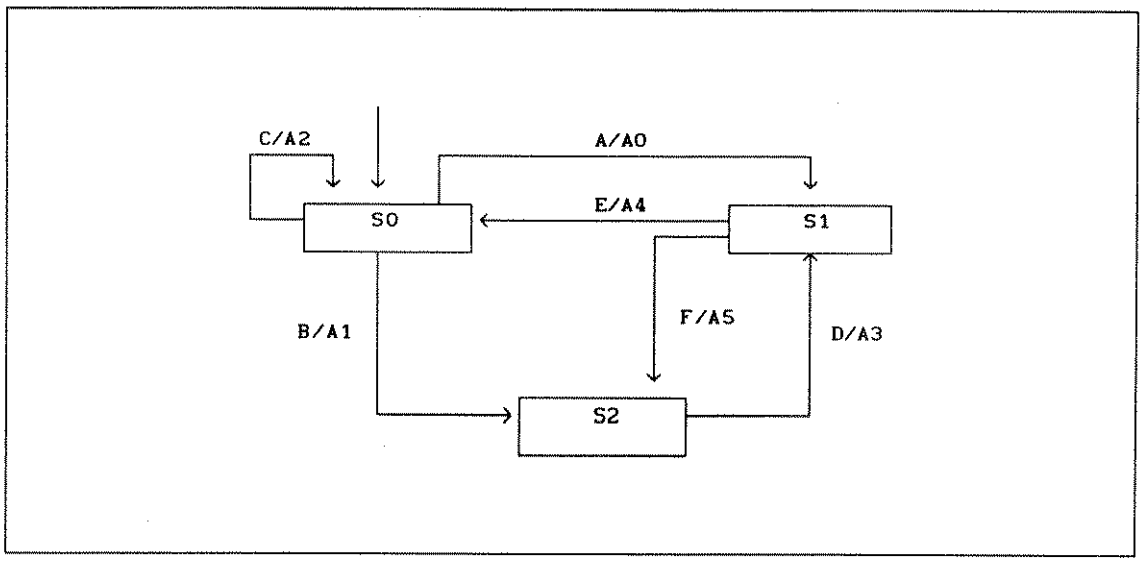


Figura 6.9. Autômato Finito.

Cada transformação de controle terá associada uma tabela representando o autômato correspondente. O tratamento de eventos será realizado através de três funções do Núcleo : **Note_event**, **Reset_event_store**, **Signal_event_store**.

A primeira função sinaliza a ocorrência de um evento, a segunda inicializa um armazenador de eventos e a terceira sinaliza o armazenador.

Desta forma, é possível definir a sequência gerada pelo prototipador para representar transformações de controle.

```
TASK_T7( )
 {
  estado_atual := estado_inicial;
  /* ações iniciais */
  for(;;)
   {
    osautomato(tabela_automato_T7,&estado_atual); /* executa automato */
   }
 }
```

A variável "**estado_atual**" mantém o estado atual do autômato sendo fornecido como parâmetro, juntamente com a tabela de estados, para a rotina do núcleo "**osautomato**" (executa_automato) que implementa efetivamente o autômato. O comentário **"ações iniciais"** representa ações que são executadas uma única vez na iniciação do sistema.

6.1.3.2. Transformação de Controle com "Enable" / "Disable".

Esta transformação apresenta os fluxos de eventos denominados "**Prompts**" - "**Enable**" e "**Disable**", que permitem a comutação entre os estados "**Habilitado**" e "**Desabilitado**" (Figura 6.10).

Sua implementação segue o procedimento descrito para a Transformação de Controle Simples sendo que, além da condição de um fluxo de evento de entrada, é necessário que o "**Prompt Enable**" seja satisfeito para que o evento seja processado. Se a transição estiver no estado "**Desabilitado**", o evento em questão será descartado.

O seu comportamento é descrito pelo "Diagrama de Transição de Estados", como já apresentado em 6.1.3.1.

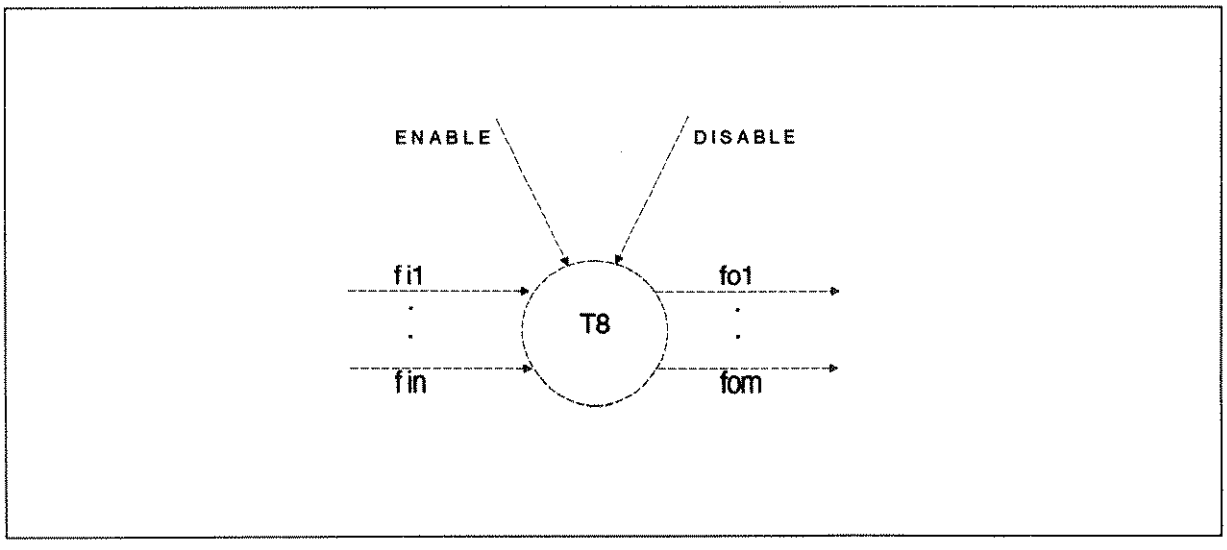


Figura 6.10. Transformação de Controle com "Enable" / "Disable".

A Prototipação gera a seguinte sequência :

```
TASK_T8( )
 {
  estado_atual := estado_inicial;
  /* ações iniciais */
  for(;;)
   {
   wait_enable( );
   osautomato(tabela_automato_T8,&estado_atual);  /* executa automato */
  }
 }
```

6.1.3.3. Transformação de Controle com "Trigger".

Esta transformação é habilitada pelo "**Prompt**" "**Trigger**" (Figura 6.11) permitindo uma execução com o dado disponível.

Sua implementação segue o procedimento descrito em 6.1.3.2 para a Transformação de Controle com "**Enable**"/"**Disable**" sendo, necessário neste caso, que o "**Prompt Trigger**" seja satisfeito para que o evento seja processado.

A prototipação gera a seguinte sequência :

```
TASK_T9()
 {
  estado_atual := estado_inicial;
  /* ações iniciais */
  for (;;)
   {
    wait_trigger( );
    osautomato(tabela_automato_T9,&estado_atual); /* executa automato */
   }
 }
```

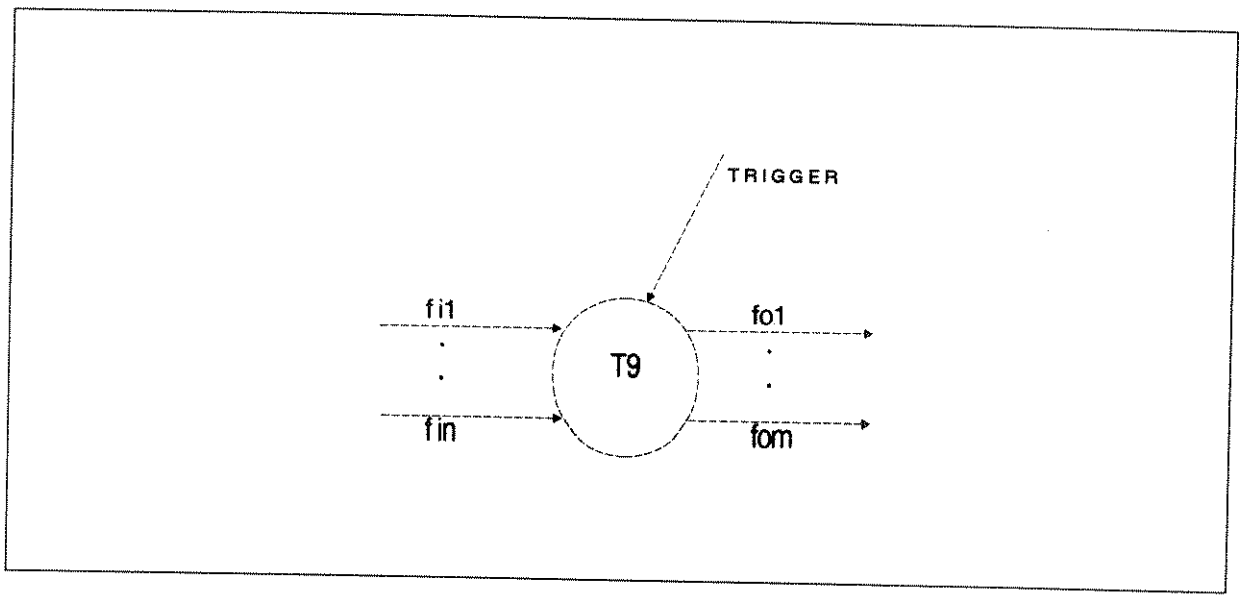


Figura 6.11. Transformação de Controle com "Trigger".

6.1.4. Armazenador.

Os armazenadores são utilizados para manter conjuntos de dados para posterior utilização.

Os dados no interior do armazenador podem ser atualizados e compartilhados por diversas transformações.

Os fluxos que conectam o armazenador à transformação não são rotulados, apenas apresentam a disponibilidade dos dados para a transformação.

6.1.4.1. Armazenador de Dados.

O acesso ao armazenador de dados (Figura 6.12) será realizado por 2 funções do Núcleo de Tempo Real apresentadas na Tabela 6.2. :

TABELA 6.2. FUNÇÕES ASSOCIADAS AO ARMAZENADOR DE DADOS.

FUNÇÃO	DESCRIÇÃO
Read_data_store	Lê armazenador de dados
Write_data_store	Escreve no armazenador de dados

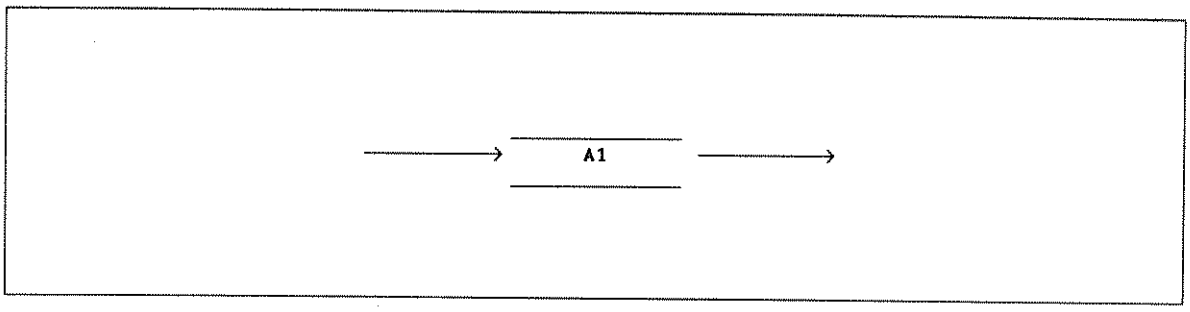


Figura 6.12. Armazenador de dados.

6.1.4.2. Armazenador de Eventos.

O tratamento do armazenador de eventos (Figura 6.13) é mais complexo do que o armazenador de dados exigindo funções distintas para sua implementação.

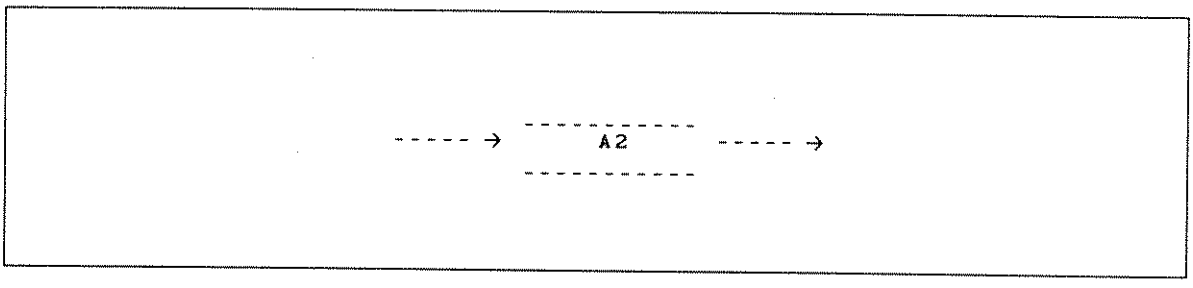


Figura 6.13. Armazenador de Eventos.

A Tabela 6.3 apresenta as funções utilizadas para tratar o armazenador de eventos.

TABELA 6.3. FUNÇÕES ASSOCIADAS AO ARMAZENADOR DE EVENTOS.

FUNÇÕES	DESCRIÇÃO
Reset_event_store	Inicializa armazenador de eventos
Signal_event_store	Sinaliza armazenador de eventos

6.1.5. Fluxo de Dados / Eventos / Controle.

Os fluxos de dados/eventos/controle são responsáveis pela movimentação da informação dentro do Modelo Essencial. Na prototipação realizada, esta movimentação será garantida através de funções internas ao Núcleo de Tempo Real. Esta estratégia garante a integridade da informação e libera o usuário de coordenar possíveis comutações de contexto ocorridas durante a movimentação.

A Tabela 6.4 apresenta resumidamente as funções implementadas no núcleo para tratamento de fluxos de informação. Uma descrição mais profunda pode ser obtida em [AZE91a].

TABELA 6.4. FUNÇÕES DE FLUXOS DE DADOS/EVENTOS/CONTROLE.

FUNÇÕES	DESCRIÇÃO
Send_flow	Envia fluxo discreto
Receive_flow	Recebe fluxo discreto
Send_flow_continuous	Envia fluxo contínuo
Receive_flow_continuous	Recebe fluxo contínuo
Enable_task	Envia prompt "Enable"
Disable_task	Envia prompt "Disable"
Wait_enable	Aguarda prompt "Enable"
Signal_trigger	Envia prompt "Trigger"
Wait_trigger	Aguarda prompt "Trigger"
Note_event	Sinaliza evento

6.1.6. Terminadores.

Os Terminadores são os elementos que atuam como fontes ou destinos das informações do sistema, ou seja, simulam a interface com o ambiente (Figura 6.14).

Caso os fluxos sejam de saída em relação ao terminador, seu conteúdo será representado através do arquivo de Cenário. Se forem de entrada em relação ao terminador, seu conteúdo será representado através do arquivo de resultados da simulação do sistema.

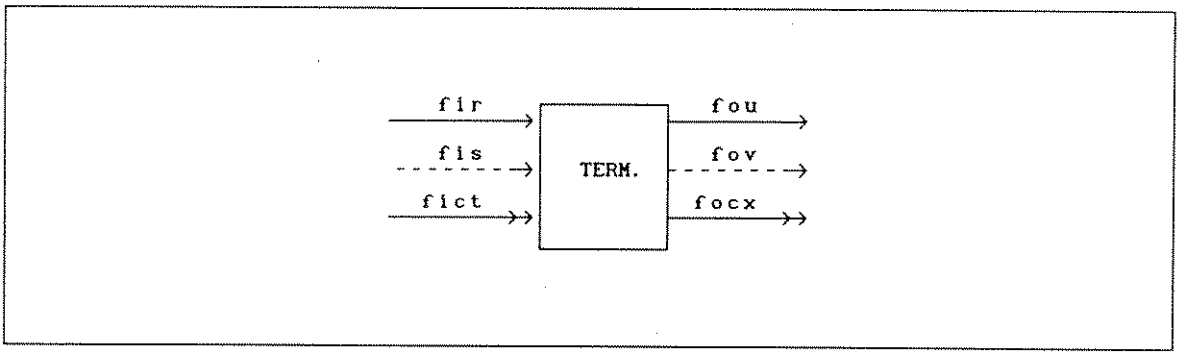


Figura 6.14. Terminador.

6.2. Estrutura do Software.

6.2.1. Descrição dos módulos.

O software do sistema ProTR está implementado da seguinte forma (Figura 6.15) :

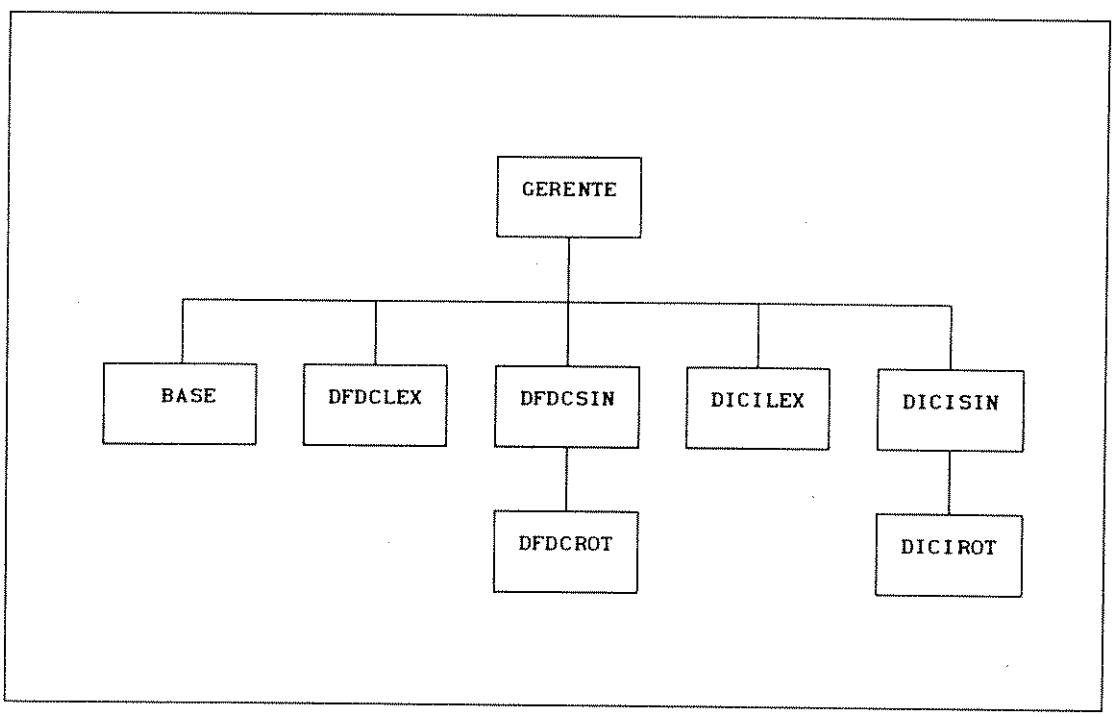


Figura 6.15. Estrutura do software.

- **Gerente** - Este módulo é responsável pela coordenação da execução do prototipador, ativando a leitura/análise da descrição do sistema alvo, fornecida pelo usuário.

- **Base** - Módulo responsável pela construção da Estrutura de Dados que representa o sistema, a partir da leitura dos dados de entrada do usuário. Apresenta as mensagens de erro de consistência detectados na descrição do "Esquema de Transformações" bem como os erros de ambiente do sistema ProTR.

- **Dfdclex** - Este módulo realiza a análise léxica da descrição do Esquema de Transformações.

- **Dfdcsin** - Este módulo realiza a análise sintática da descrição do Esquema de Transformações.

- **Dfdcrot** - Módulo responsável pela implementação das rotinas semânticas do Esquema de Transformações. Basicamente, preenche campos da Base de Dados e realiza análise de consistência dos elementos do diagrama.

- **Dicilex** - Este módulo realiza a análise léxica do Dicionário de Dados dos elementos da metodologia.

- **Dicisin** - Este módulo realiza a análise sintática do Dicionário de Dados.

- **Dicirot** - Módulo responsável pela implementação das rotinas semânticas do Dicionário de Dados. Basicamente, trata da geração dos módulos de código fonte do protótipo, na linguagem C.

Um dos pontos fundamentais na implementação do sistema ProTR está centrado na escolha das estruturas de dados utilizadas pelo sistema para organizar as informações da descrição do protótipo. As próximas Seções descrevem essas estruturas e a representação adotada na implementação.

6.2.2. Tabela de Identificadores (ti).

A análise do Esquema de Transformações e do Dicionário de Dados é realizada através de um compilador derivado do desenvolvido por Setzer [SET83]; ao mesmo tempo que efetuamos a análise sintática é também construída a tabela de identificadores do prototipador.

A tabela de identificadores mantém todos os nomes, juntamente com seus atributos, declarados pelo usuário na descrição do Esquema de Transformações. Esta tabela é posteriormente complementada durante a análise do Dicionário de Dados, sendo então utilizada para a geração do código.

A tabela adota como premissa a não existência da definição de dois elementos com o mesmo nome; entretanto, a metodologia de Ward & Mellor permite a existência de fluxos de eventos "**Trigger**", "**Enable**" e "**Disable**", como entrada de diversas transformações distintas.

Como solução para identificação destes fluxos de eventos, optou-se pela atribuição de nome idêntico ao da transformação de entrada à qual o fluxo está associado (Figura 6.16). No entanto, esta solução gerou um novo problema na inserção dos elementos na tabela dos identificadores (TI), pois era constatada a existência de outro elemento com o mesmo nome.

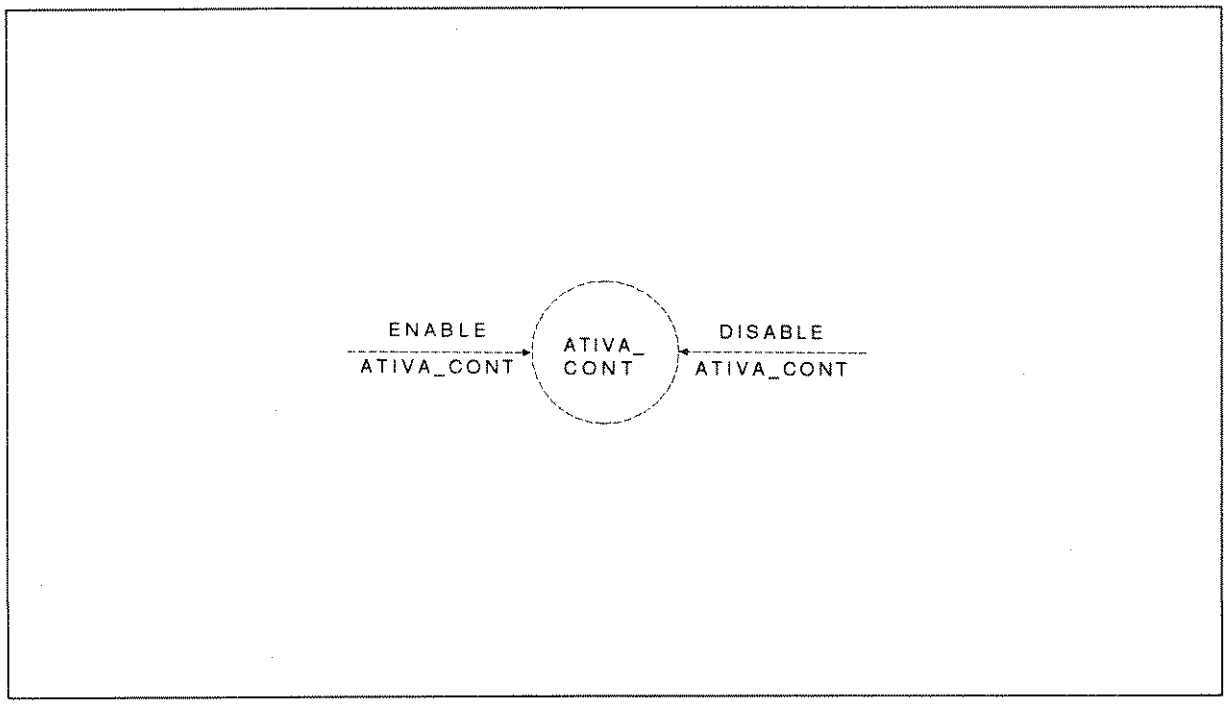


Figura 6.16. Transformação com Fluxos de Evento.

Para contornar este problema, os nomes dos "prompts" "**Trigger**", "**Enable**" e "**Disable**" foram acrescidos dos caracteres "não imprimíveis" \001, \002 e \003, de forma a garantir sua inserção na Tabela de Identificadores.

A mesma situação ocorreu em relação à identificação de fluxos associados a armazenadores de dados/eventos. Para permitir que os fluxos de dados / eventos possuam o mesmo nome do armazenador, os seus nomes são acrescidos dos caracteres "não imprimíveis" \004.

A inserção de informações na tabela é realizada através de rotinas semânticas associadas aos nós do grafo sintático que representa a linguagem reconhecida pelo analisador sintático.

A estrutura da tabela adota a técnica de "hash table" [KNU73] sendo a função "hash" utilizada baseada no algoritmo desenvolvido por P.J. Weinberger's [AHO86]. A Figura 6.17 apresenta o formato da estrutura "hash" adotada.

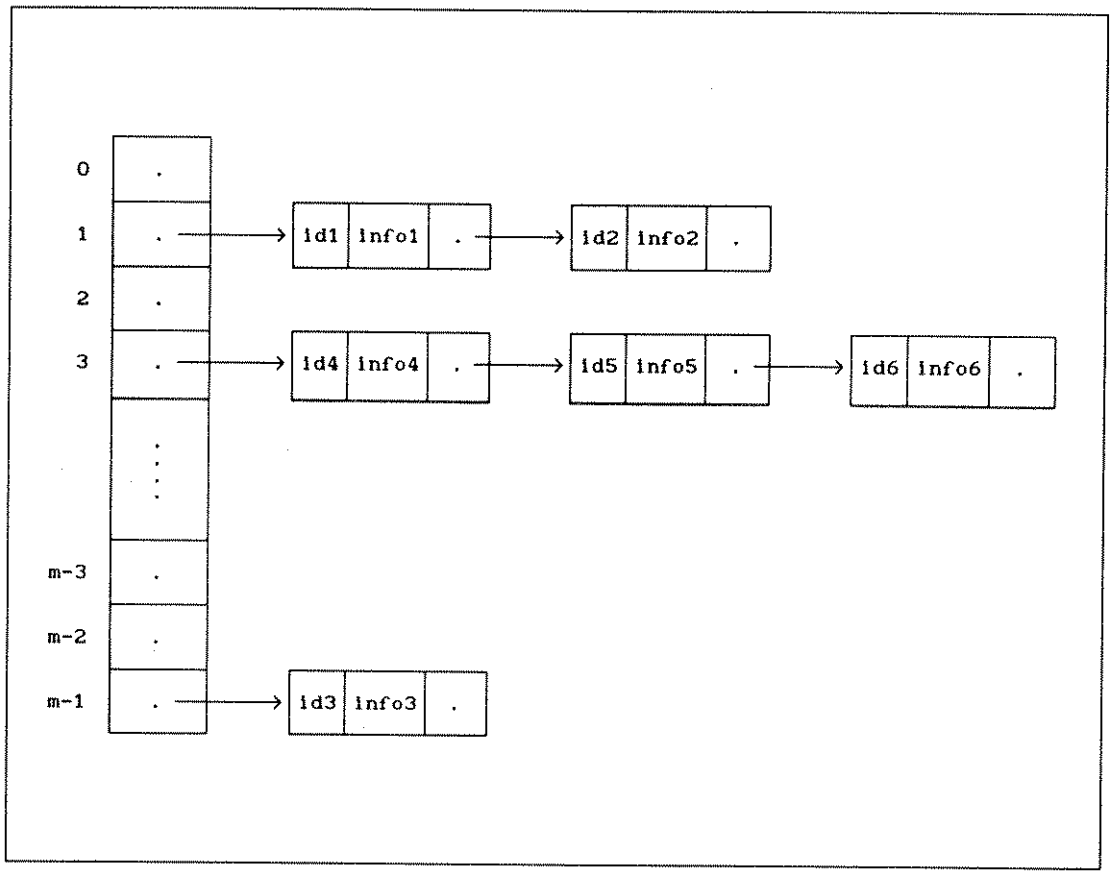


Figura 6.17. Organização da Tabela "hash".

6.2.3 O Grafo de Hierarquia.

Como já afirmado, a metodologia "Desenvolvimento Estruturado para Sistemas de Tempo Real" possui uma estrutura hierárquica; ou seja, a primeira representação cobre todo o sistema e suas interfaces com o ambiente. As representações seguintes são refinamentos das anteriores apresentando um nível maior de detalhes [WAR85].

A Figura 6.18 representa essa hierarquia através dos diversos níveis que compõem o sistema.

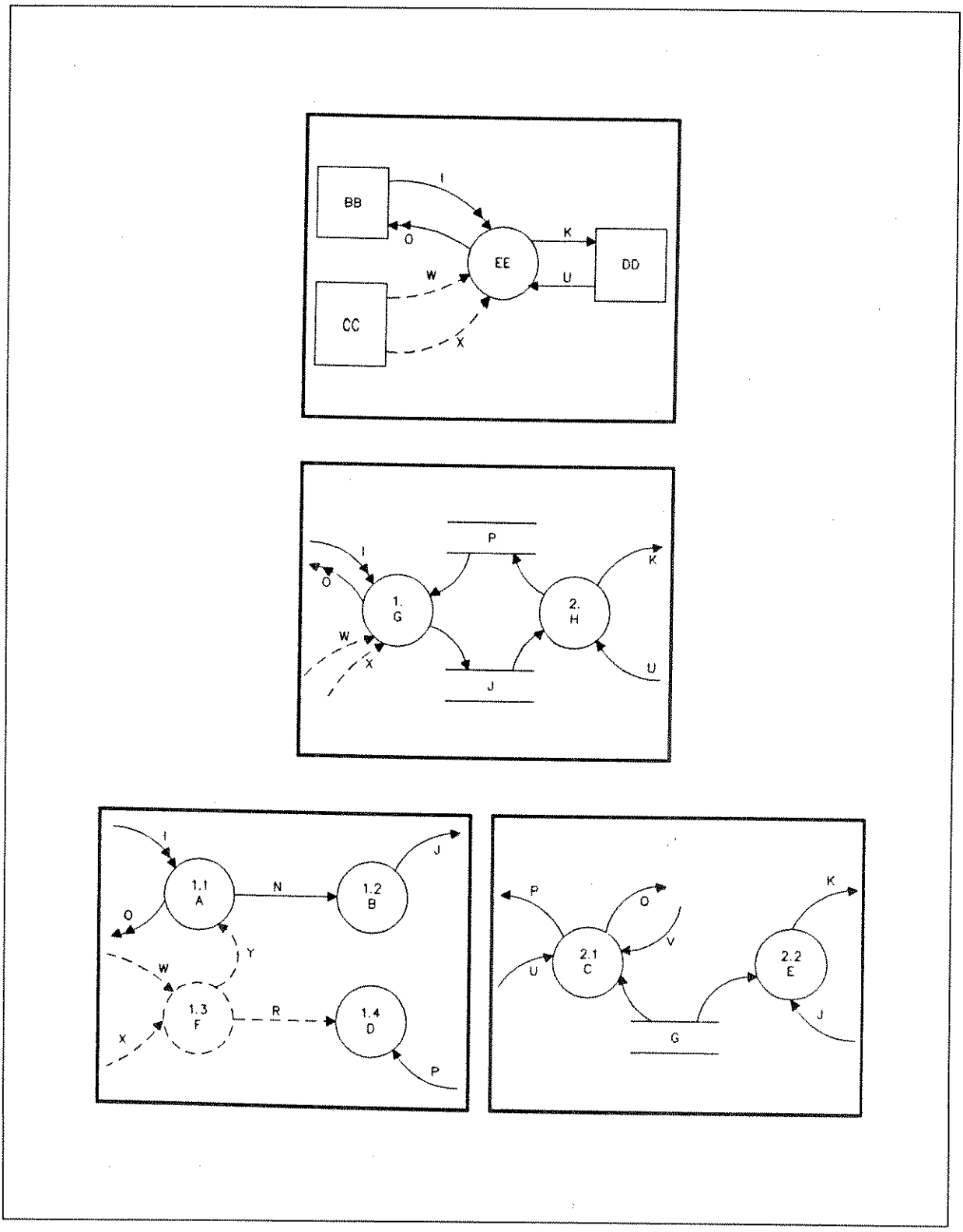


Figura 6.18. Hierarquia do "Esquema de Transformações".

Observe na Figura 6.18 onde no primeiro nível temos três Terminadores : **BB**, **CC** e **DD**, e Transformação **EE**. Na decomposição, todos os fluxos presentes no nível superior devem estar presentes no próximo nível, sendo proibida a "geração" / "desaparecimento" de fluxos. A decomposição prossegue até o segundo nível onde temos as transformações **A**, **B**, **F** e **D** originadas da transformação **G** e as transformações **C** e **E**, originadas da Transformação **H**.

A hierarquia do modelo será implementada através de "listas generalizadas" [HOR84]. Uma lista generalizada **A** é uma sequência finita de $n \geq 0$ elementos $a_1, a_2, \ldots, a_n$, onde a_i é uma partícula ou uma lista. Os elementos a_i que são listas, são denominados sublistas de **A**.

Um nó dessa lista possui o formato apresentado na Figura 6.19, onde **INFO** mantém a informação do nó, **LINK1** indica o início de outra lista e **LINK2** aponta para outro nó.

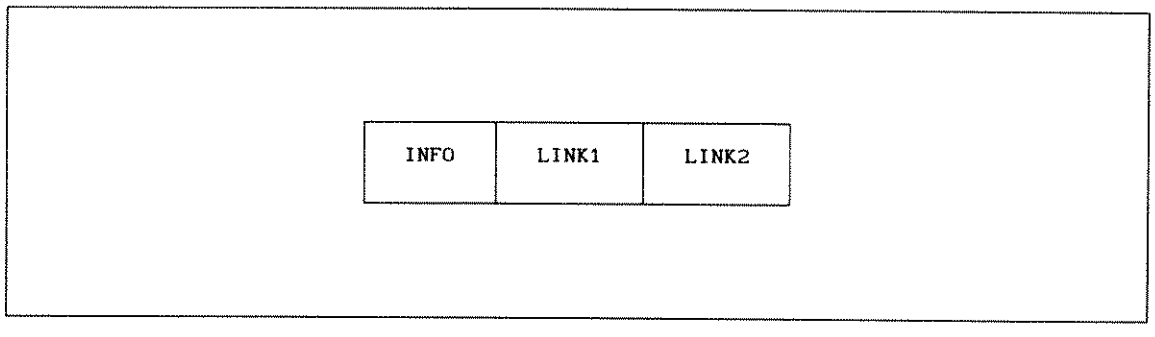


Figura 6.19. Formato de um nó da lista generalizada.

A raiz da lista generalizada que representa o sistema é identificada pela variável **trans_raiz**.

Com esses elementos, a representação hierárquica da Figura 6.18 pode ser observada na Figura 6.20.

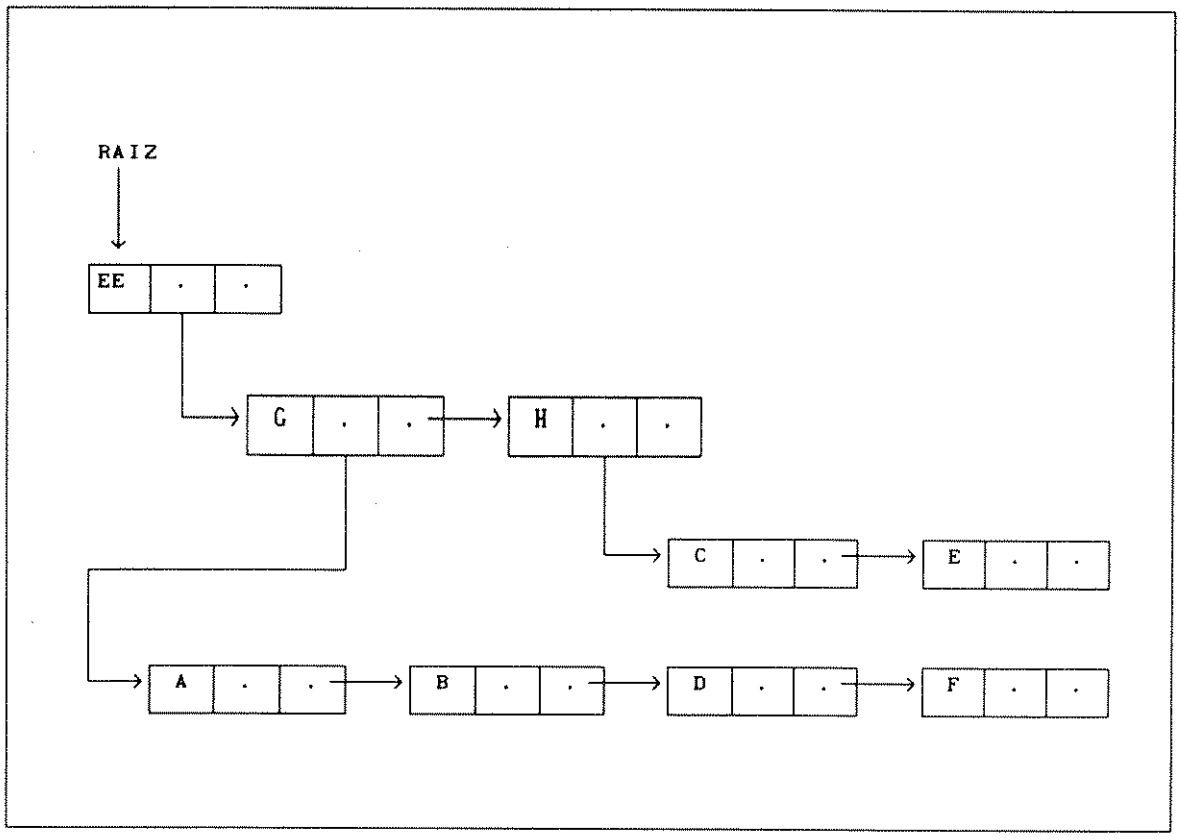


Figura 6.20. Representação hierárquica do "Esquema de Transformações"

6.2.4. Listas de Apoio.

Além das estruturas apresentadas nas Seções anteriores, foram implementadas listas lineares que permitem um acesso otimizado aos elementos do sistema. Essas listas são apresentadas abaixo :

- **fluxos** - contém os fluxos presentes no sistema.
- **arms** - contém os armazenadores presentes no sistema.
- **ters** - contém os terminadores presentes no sistema.
- **trans** - contém as transformações "folhas" presentes na hierarquia do sistema. As transformações "folhas" são transformações que não possuem mais expansões na hierarquia do sistema.

6.2.5. A Representação da Base.

Uma vez definido o modelo adotado para organizar os dados do prototipador, é necessário apresentar sua estrutura através da linguagem de programação C. Esta descrição é importante pelas implicações das escolhas realizadas sobre o desempenho do sistema. Inicialmente vamos apresentar a estrutura básica de uma entrada na tabela de identificadores (**ti**).

```
struct sti
 {
  char nome[TAMIDEN+2];              /* nome do identificador */
  char tipo;                         /* tipo do identificador */
  union  {
    struct stran tran;
    struct sfluxo fluxo;
    struct sarm arm;
    struct ster ter;
   } dado;
  struct sti *stiprox;          /* aponta para próximo elemento */
 }
```

Na estrutura *struct sti*, **tipo** indica o tipo do identificador podendo assumir os seguintes valores : **TRAN** (transformação), **FLUXO** (fluxo), **ARM** (armazenador), **TER** (terminador).

A cada um dos valores que **tipo** pode assumir, está associada uma estrutura para descrever em detalhes o tipo identificado.

6.2.5.1. A estrutura "stran".

A estrutura **"stran"** descreve as transformações presentes no sistema; seu formato é apresentado abaixo.

```
struct stran
 {
  char *tnome;                          /* nome da transformação */
  char refer[MAXNIVEL];                          /* referência */
  char ttipo;                           /* tipo da transformação */
  char testado;                /* estado atual da transformação */
  struct sfluxo *tifluxo[MAXFLUX];          /* fluxo de entrada */
  struct sfluxo*tofluxo[MAXFLUX];             /* fluxo de saída */
  struct stran *link1;                     /* início outra lista */
  struct stran *link2;                        /* início outro nó */
  struct stran *tranprx;         /* lista transf. geram tarefas */
  short período;                          /* período de ativação */
  short prioridade;             /* prioridade da transformação */
  int trsys;          /* identificador da transf. para o Núcleo */
 };
```

- **tnome** - indica o nome da transformação dentro do sistema.

- **refer** - apresenta a referência da transformação no modelo hierárquico do sistema.

- **ttipo** - indica o tipo da transformação e pode assumir os seguintes valores : **SIMPLES** (simples), **CONTI** (contínua), **CONTR** (controle).

- **testado** - indica o estado da análise da transformação, ou seja, se a análise foi ou não concluída.

- **tifluxo** e **tofluxo** - apontam para os fluxos de entrada e saída da transformação respectivamente.

- **link1** e **link2** - estão associados à estrutura hierárquica do sistema.

- **tranprx** - apontador utilizado pela lista das transformações "folhas" que geram tarefas.

- **período** - indica o período de ativação da transformação.

- **prioridade** - indica a prioridade da transformação.

- **trsys** - mantém o identificador da transformação para o Núcleo.

6.2.5.2. A estrutura "sfluxo".

Nesta estrutura são mantidas as características dos fluxos de informação do sistema. Seu formato é apresentado abaixo :

```
struct sfluxo
 {
  char *snome;                                  /* nome do fluxo */
  char stipo;                                   /* tipo do fluxo */
  char sestado;                            /* estado atual do fluxo */
  char sdecl[MAXDCL];                    /* declaração tipo do fluxo */
  struct sfluxo *flxprx;                          /* lista de fluxos */
  struct sarm *armptr;     /* ponteiro armazenador se fluxo ARM */
  int fsys;               /*identificador do fluxo para o Núcleo */
  char sarray;            /* indica se o tipo do fluxo é arranjo */
  char sligação;         /* indica se está ligado a TRAN/ARM/TER */
  struct stran *trans[NENTF];  /* entrada para transf. "folha" */
  char sdireção;            /* direção do fluxo - ENTRA/SAI/BIDI */
  char fdcl;             /* tipo do fluxo - char - int - long ... */
 };
```

- **snome** - indica o nome do fluxo dentro do sistema.

- **stipo** - indica o tipo do fluxo e pode assumir os seguintes valores : **EVE** (evento), **TRIGGER** (trigger), **ENABLE** (enable), **DISABLE** (disable), **CONT** (contínuo) e **DISC** (discreto).

- **sestado** - indica o estado da análise do fluxo; ou seja, se a análise foi ou não concluída.

- **sdecl** - mantém a declaração do tipo do fluxo (int, char, short, etc). Caso o tipo não seja estruturado, a definição do tipo é mantida em "*sdecl*"; se o tipo for estruturado, será mantido em "*sdecl*" um "*typedef*" para o tipo em questão.

- **flxprx** - ponteiro utilizado na construção da lista dos fluxos.

- **armptr** - mantém ponteiro para o armazenador para o qual o fluxo é entrada.

- **fsys** - mantém o identificador do fluxo para o Núcleo.

- **sarray** - indica se o fluxo é do tipo "array".

- **sligação** - indica se o fluxo está ligado a transformação ou armazenador ou terminador.

- **trans** - aponta para a transformação "folha" para a qual o fluxo corrente é entrada.

- **sdireção** - indica a direção do fluxo : **ENTRA** (entrada), **SAI** (saída) ou **BIDI** (bidirecional).

- **fdcl** - indica o tipo do dado presente no fluxo : char, int, long, etc.

6.2.5.3. A estrutura "sarm".

A estrutura "*sarm*" descreve as características dos armazenadores do sistema. Seu formato é apresentado abaixo :

```
struct sarm
 {
  char *anome;                          /* nome do armazenador */
  char atipo;                           /* tipo do armazenador */
  char aestado;                        /* estado do armazenador */
  struct sfluxo *armifluxo[MAXFLUX];          /* fluxo entrada */
  struct sfluxo *armofluxo[MAXFLUX];           /* fluxo saída */
  char adecl[MAXDCL];            /* declaração tipo armazenador */
  struct sarm *armprx;                /* lista de armazenadores */
  int asys;              /* identificador do arm. para o núcleo */
  char aarray;       /* indica se tipo do armazenador é arranjo */
  char atdcl;          /* tipo do fluxo - char - int - long ... */
 };
```

- **anome** - indica o nome do armazenador dentro do sistema.

- **atipo** - indica o tipo do armazenador, podendo assumir os seguintes valores : **EVENTO** (armazenador de eventos) e **DADOS** (armazenador de dados).

- **aestado** - indica o estado da análise do armazenador, ou seja, se a análise foi ou não concluída.

- **armifluxo** e **armofluxo** - apontam para os fluxos de entrada e saída do armazenador respectivamente.

- **adecl** - mantém a declaração do tipo do armazenador. Caso o tipo seja simples "*adecl*" contém o próprio tipo, senão, em caso de tipos compostos, será utilizado um "*typedef*" para definir o tipo.

- **armprx** - ponteiro utilizado na construção da lista de armazenadores.

- **asys** - mantém o identificador do armazenador para o Núcleo.

- **aarray** - indica se tipo do armazenador é arranjo */

- **atdcl** - indica o tipo do dado presente no armazenador : char, int, long, etc.

6.2.5.4. A estrutura "ster".

A estrutura "*ster*" mantém informações à respeito dos terminadores do sistema. Seu formato é apresentado abaixo :

```
struct ster
 {
  char *ternome;                          /* nome do terminador */
  char terestado;                        /* estado do terminador */
  struct sfluxo *terifluxo[MAXFlUX];            /* fluxo entrada */
  struct sfluxo *terofluxo[MAXFLUX];             /* fluxo saída */
  struct ster *terprx;                  /* lista de terminadores */
  int tsys;       /* identificador do terminador para o núcleo */
 };
```

- **ternome** - indica o nome do terminador dentro do sistema.

- **terestado** - indica o estado da análise do terminador, ou seja, se a análise foi ou não concluída.

- **terifluxo** e **terofluxo** - apontam para os fluxos de entrada e saída do terminador respectivamente.

- **terprx** - ponteiro utilizado na construção da lista de terminadores do sistema.

- **tsys** - mantém um identificador do terminador para o núcleo.

Capítulo 7. CONCLUSÃO.

O presente trabalho descreve a implementação de um prototipador para Sistemas de Tempo Real utilizando como técnica para descrição do sistema, o modelo desenvolvido por Ward & Mellor. Sua realização demandou esforços em duas linhas distintas de atuação.

A primeira linha está associada à estratégia adotada para a realização do projeto. Para permitir a geração de código a partir da especificação e sua posterior execução, soluções não contempladas na literatura pesquisada foram criadas, sendo que, duas merecem destaque :

. A estratégia de implementar primitivas no núcleo de tempo real que semânticamente possuem o mesmo comportamento dos elementos presentes na metodologia (movimentação de fluxo de dados/eventos, implementação de autômatos finitos e estruturas para representar armazenadores). Esta estratégia otimizou a geração de código, além de facilitar a evolução do protótipo para o sistema definitivo.

. Na execução do código prototipado, a escolha do instante de consumir um novo conjunto de entradas passou por várias opções antes de alcançar a solução proposta, onde o usuário pode escolher entre duas formas de atuação : consumo por tempo e consumo por evento.

A segunda linha está associada ao esforço necessário para efetivamente concretizar os objetivos propostos com os recursos computacionais disponíveis (1 IBM XT). Nesse esforço três pontos merecem destaque :

. A ausência de uma ferramenta gráfica para representar o sistema através da representação gráfica dos elementos da metodologia exigiu a definição de três linguagens (juntamente com os respectivos compiladores) para descrever os seguintes elementos : o Esquema de Transformações, o Esquema de Dados e o arquivo de Cenário utilizado durante a execução.

. A elaboração de um gerador de código para, a partir dos elementos da metodologia, criar o código fonte do sistema alvo.

. A implementação de um núcleo de tempo real totalmente direcionado para a metodologia, com cerca de 36 primitivas [AZE91a].

Os resultados obtidos correspondem às expectativas, permitindo apresentar ao usuário uma visualização da execução do sistema antes da implementação definitiva. Outro ponto que merece destaque é a geração automática de código. Esta geração permite que em ambientes monoprocessadores, sem sistemas de arquivos ("embedded systems"), o resultado da prototipação seja praticamente o sistema alvo.

Podemos observar na literatura trabalhos que apresentam propostas semelhantes ao ProTR. Nos próximos parágrafos apresentam-se as características de algumas destas ferramentas, previamente descritas no Capítulo 3.

Um trabalho interessante é apresentado por Blumofe e Hecht [BLU88]; a ferramenta apresentada, denominada "Teamworks", permite editar/simular sistemas descritos através da metodologia "Desenvolvimento Estruturado para Sistemas de Tempo Real". A execução realizada é simbólica, ou seja, somente o movimento dos fluxos é visualizado; as marcas não possuem valores e as transformações não possuem descrições funcionais; logo, não podem ser executadas. Este tipo de simulação é semelhante à execução de uma Rede de Petri [PET81]; ou seja, uma vez que marcas estejam presentes em todos os arcos de entrada, a transição é ativada e as marcas são inseridas nos arcos de saída. O usuário pode, no sistema "Teamworks", direcionar a inserção de marcas na saída em algumas situações especiais, estabelecendo o instante em que cada arco deve receber uma marcação.

Outro trabalho, desenvolvido por Luqui [LUQ91], na Naval Postgraduate School, California, propõe a criação de um prototipador para sistemas de tempo real, com as seguintes características.

- **Prototype System Description Language - PSDL** - Linguagem de alto nível para descrição das transformações do sistema.
- **Editor DFD modificado** - Utilização de um editor gráfico para representar, através de diagramas de fluxo de dados com restrições de temporização e controle, a movimentação de dados no sistema.
- **Sistema de Suporte à Execução** - Responsável pelo controle da execução do protótipo gerado.
- **Sistema de Gerência de Banco de Dados** - Responsável pelo armazenamento do protótipo e dos componentes reutilizáveis.

Uma característica importante na utilização de protótipos é a interface com o usuário; em sistemas que realçam elementos como : relatórios, telas e interface com banco de dados, a ênfase sobre as interfaces é natural e consequência direta da utilização final do produto. Em sistemas de tempo real, a interface com seres humanos tem importância secundária; nestes sistemas, a interface com o ambiente é mais relevante. Cooling e Hughes [COO89] focalizam este problema e propõem uma forma de comunicação do comportamento do sistema através de um ambiente gráfico interativo. Utilizando-se de esquemas animados, numa linguagem próxima à do usuário, a sequência de eventos gerada pela execução do protótipo é representada graficamente.

Dentre os trabalhos citados, aquele que encontra-se totalmente operacional é o desenvolvido pela Luqui; os demais encontram-se em desenvolvimento, com diversas propostas de otimização.

Numa comparação entre o **ProTR** e estes trabalhos, alguns pontos merecem destaque :

- **Tamanho** - O sistema ProTR é um software pequeno (cerca de 12808 linhas de código, com 117347 bytes de código executável); logo, comparações só fazem sentido se feitas com sistemas de tamanhos semelhantes.

- **Apelo Gráfico** - Um dos pontos mais críticos no ProTR está associado à utilização de recursos gráficos para a entrada de dados e apresentação dos resultados; este é um ponto a ser destacado nas próximas versões.

- **Execução Funcional** - O sistema ProTR permite execução das transformações com os fluxos carregando valores concretos. Como afirmado por Blumofe e Hecht [BLU88], esta característica é extremamente desejável pois permite uma visão mais realista do sistema sendo prototipado.

- **Análise de Propriedades** - O sistema ProTR permite que erros semânticos cometidos pelo usuário na descrição do sistema sejam detectados e corrigidos. Os erros mais comuns são duplicidade de nomes e inconsistências entre os diversos níveis que compõem a descrição do sistema. A ferramenta "Teamwork", por outro lado, fornece testes de "deadlock" e de alcance (a partir de um estado, quais transformações podem ser ativadas).

- **Interface com o Usuário** - Com exceção do trabalho de Cooling & Hughes, a apresentação dos resultados da prototipação ao cliente não tem recebido a atenção devida na área de Sistemas de Tempo Real. A ferramenta ProTR oferece ao usuário a possibilidade de testar o protótipo através de cenários que simulam a interface com o ambiente. O arquivo com os resultados da execução do protótipo fornecem ao usuário, a cada instante, uma posição a respeito das conexões externas do sistema. Naturalmente, telas gráficas representando o sistema possuem um efeito muito mais agradável para o usuário; entretanto, a construção destas telas pode onerar o processo de prototipação, uma vez que cada sistema prototipado deve possuir um conjunto de telas apropriado para representar a interface com o ambiente.

Como já afirmado, apesar dos resultados promissores, a ferramenta ProTR possui limitações, pontos passíveis de melhoria, que deverão ser tratadas em versões futuras. Como exemplo podemos citar :

- **Ausência de recursos gráficos** : O objetivo do desenvolvimento da ferramenta ProTR foi a aquisição de conhecimentos na área de prototipação associada a sistemas de tempo real; dessa forma, a construção de um ambiente gráfico para utilização da ferramenta foi adiada para uma fase futura. A idéia básica é associar o ProTR com uma ferramenta CASE que possua o tratamento da técnica de Ward-Mellor evitando, desse modo, a utilização de recursos na construção de um ambiente facilmente encontrado no mercado.

- **Extensão à Linguagem de Descrição do Cenário** : A geração dos elementos pertencentes ao ambiente poderia conter equações mais complexas do que constantes e equações lineares; todavia, seu tratamento por parte do protótipo seria complexo; além disso, essas equações podem ser aproximadas pelos elementos fornecidos.

- **Dicionário de Dados mais amigável** : A descrição das transformações de dados é realizada através da Linguagem **C**. A necessidade de descrever os elementos da metodologia através de uma linguagem de programação limita a atuação do usuário e dificulta o tratamento de erros gerados na descrição. Uma linguagem mais próxima ao usuário, desvinculada da linguagem de programação **C** e utilizando o vocabulário da aplicação, seria altamente desejável como otimização ao sistema.

- **Execução através de telas pictóricas** : Um dos objetivos das técnicas de prototipação é garantir a participação do usuário durante as diversas fases de desenvolvimento do sistema. Entretanto, os resultados da execução do Cenário podem se apresentar áridos para um usuário leigo, apesar de serem plenamente satisfatórios para o analista. Para minimizar este problema, uma extensão desejável à ferramenta ProTR seria a possibilidade da criação de telas pictóricas, onde o sistema é

representado numa linguagem próxima ao usuário como : válvulas, medidores, níveis de recipientes, esteiras, atuação de robos, etc.

- **Resultado da Execução do Protótipo** : Um outro ponto passível de otimização está relacionado com a determinação dos fluxos que farão parte do resultado da execução do protótipo. Na versão atual, o usuário pode selecionar entre receber informações de todos os fluxos, ou somente os fluxos conectados aos Terminadores do sistema. Para aumentar o controle do usuário sobre os resultados da execução, é desejável que ele possa determinar exatamente quais fluxos devem compor o resultado, ou determinar um nível de hierarquia a ser monitorado.

8. BIBLIOGRAFIA.

[AHO86] Aho A.V., Sethi R., Ullman J.D., Compilers : Principles, Techniques and Tools, Addison-Wesley Publishing Co., Massachusetts CA, 1986, pg. 433-438.

[ALA84] Alavi M., "An Assessment of the Prototyping Approach to Information Systems Development", Communications of the ACM, vol. 27, no. 6, pg. 556-563, June 1984.

[ALA86] Alavi M., "Application Prototyping", Systems Development Management, Auerbach Publishers Inc., 1986.

[AZE91a] Azevedo H., Azevedo G.D.F., Especificação de Software do Núcleo de Tempo Real NProTR, DTIA 002/91, Instituto de Automação, Centro Tecnológico para Informática, Outubro 1991.

[AZE91b] Azevedo G.D.F., Azevedo H., Jino M., "ProTR - Um Prototipador para Sistemas de Tempo Real", II Semana da Computação, Escola de Engenharia da Fundação Municipal de Ensino de Piracicaba, 04 a 08 de Novembro de 91.

[AZE91c] Azevedo H., "Técnicas de Escalonamento para Sistemas Hard Real Time - Análise e Implementação", Dissertação de Mestrado em Engenharia Elétrica, UNICAMP, Março 1991.

[AZE91d] Azevedo G.D.F., Azevedo H., Jino M., "Uma Ferramenta para Prototipar Sistemas de Tempo Real", 1ª Jornada USP - SUCESU - SP de Informática e Telecomunicações, 05 a 07 de Julho de 1993.

[BAL90] Balda D.M., Gustafson D.A., "Cost Estimation Models for the Reuse and Prototype Software Development life-cycles", ACM Sigsoft Software Engineering Notes, vol.15, no.3, pg. 1-18, July 1990.

[BEN89] Benimoff N.I., Whitten W.B., "Human Factors Approaches to Prototyping and Evaluating User Interfaces", AT & Technical Journal, vol.68, no.5, pg. 44-55, September/October 1989.

[BER90] Berzins V., Luqi, "An Introduction to the Specification Language Spec", IEEE Software, pg. 74-84, March 1990.

[BLU88] Blumofe R., Hecht A., "Executing Real-Time Structured Analysis Specifications", ACM Sigsoft Software Engineering Notes, vol.13, no. 3, pg. 1-18, July 1988.

[BOA83] Boar B.H., Application Prototyping - A Requirements Definition Strategy for the 80s, Wiley - Interscience, 1984.

[BOE84] Boehm B.W., Gray T.E., Seewaldt T., "Prototyping Versus Specifying: A Multiproject Experiment", IEEE Transactions on Software Engineering, vol. SE-10, no. 3, pg. 290-303, May 1984.

[BRU85] Bruno G., Marchetto G., "Rapid Prototyping of Control Systems using High Level Petri Nets", IEEE, pg. 230-235, 1985.

[BRU86] Bruno G., Marchetto G., "Process-Translatable Petri Nets for the Rapid Prototyping of Process Control Systems", IEEE Transactions on Software Engineering, vol.SE-12, no.2, February 1986.

[BRU87] Bruyn W, Randall J., Keskar D., Ward P., "ESML : An Extended Systems Modeling Language Based on the Data Flow Diagram", ACM Sigsoft Software Engineering Notes, vol.13, no.1, pg. 58-67, January 1988.

[BUR86] Burns A., Kirkham J.A., "The Construction of Information Management System Prototypes in ADA", Software - Practice and Experience, vol. 16(4), pg. 341-350, April 86.

[CER87] Cerveny R.P., Garrity E.J., Hunt R.G., Kirs P.J., Sanders G.L., Sipior J.C., "Why Software Prototyping Works", Datamation, pg. 97-103, 1987.

[CHE76] Chen P.P., "The Entity-Relationship Model : Toward a Unified View of Data", ACM Transactions on Database Systems, vol. 1, no. 1, pg. 9-36, Março 1976.

[COO89] Cooling J.E., Hughes T.S., "The Emergence of Rapid Prototyping as a Real-time Software Development Tool", II International Conference on Software Engineering for Real-time Systems, Cirencester, UK, pg. 18-20, September 89.

[COO90] Coomber C.J., Childs R.E., "A Graphical Tool for the Prototyping of Real-Time Systems", ACM Sigsoft Software Engineering Notes, vol.15, no. 2, pg. 70-82, April 1990.

[CVR84] Ramamoorthy C.V., Prakash A., Tsai W, Usuda Y., "Software Engineering : Problems and Perspectives", Computer, pg. 191-209, October 1984.

[DAV82] Davis A.M., "Rapid Prototyping using Executable Requirements Specifications", Special Issue on Rapid Prototyping, ACM Sigsoft Software Engineering Notes, vol.7, no.5, pg.39-44, December 1982.

[DEA83] Dearnley P.A., Mayhew P.J., "In Favour of Systems Prototypes and their Integration into the System Development Cycle", The Computer Journal, vol. 26, no. 1, pg. 36-42, 1983.

[DEM78] DeMarco T., Structured Analysis and System Specification, Prentice-Hall, 1978.

[DEU88] Deutsch M.S., "Focusing Real-time Systems Analysis on User Operations", IEEE Software, pg. 39-50, September 1988.

[FAL88] F. Howard, "CASE Tools emerge to handle real-time systems", Computer Design, pg. 53-74, January 1988.

[FLA81] Flavin M., Fundamental Concepts of Information Modeling, New York, Yourdon Press, 1981.

[GEH82] Gehani N.H., "A Study in Prototyping", Special Issue on Rapid Prototyping, ACM Sigsoft Software Engineering Notes, vol. 7, no. 5, pg. 71-74, December 82.

[GIL85] Gilhooley I.A., "A Productive Approach to Systems Development", Systems Development Management, Auerbach Publishers Inc., 1985.

[GIL86] Gilhooley I.A., "The Impact of Prototyping on Systems Development", Systems Development Management, Auerbach Publishers Inc., 1986.

[GIL87] Gilhooley I.A., "Productive Systems Development with Prototyping", Journal of Information Systems Management, pg. 15-22, 1987.

[HAT91] Hatley D.J., Pirbhai I.A., Estratégias para Especificação de Sistema de Tempo Real, Editora McGraw-Hill Ltda e Makron Books do Brasil Editora Ltda, 1991.

[HER89] Herrero A.T., Rocha M.A., Borges M.R.S., "Experiências na Utilização do Enfoque da Prototipagem Rápida no Desenvolvimento de Sistemas de Informação", XXII Congresso Nacional de Informática - SUCESU - SP, pg. 239-244, 1989.

[HOP79] Hopcroft J.E., Ullman J.D., Introduction to Automata Theory, Languages and Computation, Addison-Wesley, 1979.

[HOR84] Horowitz E., Sahni S., Fundamentos de Estruturas de Dados, Editora Campos, Rio de Janeiro, pg. 145-157, 1984.

[JON86] Jones S.T., "Application Prototyping with a Microcomputer", Systems Development Management, Auerbach Publishers Inc., 1986.

[KER78] Kernighan B.W., Ritchie D.M., The C Programming Language, New-Jersey, Prentice-Hall Inc., 1978.

[KEU82] Keus H.E., "Prototyping : a more reasonable approach to system development", Issue on Rapid Prototyping, ACM Sigsoft Software Engineering Notes, vol.7, no. 5, pg. 94-95, December 82.

[KLI86] Klingler D.E., "Rapid Prototyping Revisited", Datamation, pg. 131-132, October 1986.

[KNU73] Knuth D.E., The Art of Computer Programming, vol.1, Addison-Wesley, Massachusetts, 1973.

[LEE85] Lee S., "On Executable Models for Rule-based Prototyping", IEEE, pg. 210-215, 1985.

[LIP86] Lipp, Mark E., "Types of Prototyping", in Prototyping : State of the Art Report, Pergamon Infotech Limited, England, 1986.

[LUQ88] Luqi, Ketabchi M., "A Computer-Aided Prototyping System", IEEE Software, pg. 66-72, 1988.

[LUQ88a] Luqi, Berzins V., "Rapidly Prototyping Real-Time Systems", IEEE Software, pg. 25-36, September 1988.

[LUQ88b] Luqi, Berzins V., Yeh R.T., "A Prototyping Language for Real-Time Software", IEEE Transactions on Software Engineering, vol. 14, no.10, pg. 1409-1423, October 1988.

[LUQ88c] Luqi, Berzins V., "Execution of a High Level Real-Time Language", Computer Society Press Reprint, Reprinted from Proceedings of the Real-time Systems, Symposium, Huntsville, AL., pg. 69-76, December 6-8, 1988.

[LUQ89] Luqi, "Software Evolution Through Rapid Prototyping", Computer, pg. 13-25, May 1989.

[LUQ90] Luqi, Barnes P.D., Zyda M., "Graphical tool for computer-aided prototyping", Information and Software Technology, vol.32, no.3, pg. 199-206, April 1990.

[LUQ91] Luqi, "The Role of Prototyping Languages in Case", International Journal of Software Engineering and Knowledge Engineering, vol.1, no.2, pg. 131-149, 1991.

[LUQ93] Luqi, "Real-time Constraints in a Rapid Prototyping Language", Computer Language, vol.18, no.2, pg. 77-103, 1993.

[MAG86] Magalhães M.F., "Software para Tempo Real", Editora da Universidade Estadual de Campinas, 1986.

[MAS82] Mason R.E.A., Carey T.T., Benjamin A., "ACT/1 : A Tool for Information Systems Prototyping", Special Issue on Rapid Prototyping ACM Sigsoft Software Engineering Notes, vol.7, no.5, pg. 120-126, December 1982.

[MAY87] Mayhew P.J., Dearnley P.A., "An Alternative Classification", The Computer Journal, vol. 30, no. 6, pg. 481-484, 1987.

[MAY90] Mayhew P.J., Dearnley P.A., "Organization and Management of Systems Prototyping", Information and Software Technology, pg. 245-252, 1990.

[McM84] McMenamin S., Palmer J., Essential System Analysis, New York, Yourdon, 1984.

[MEL90] Melendez Filho R., Prototipação de Sistemas de Informações : Fundamentos, Técnicas e Metodologia, Rio de Janeiro : LTC - Livros Técnicos e Científicos, 1990.

[PCC90] PC-Case (PC-TR) - Manual do Usuário, Base Tecnologia S.A., 1990.

[PET81] Peterson J.L., Petri Net Theory and The Modeling of Systems, Prentice-Hall, 1981.

[POM91] Pomberger G., Bischofberger W., Kolb D., Pree W., S. Holber, "Prototyping-oriented software development - concepts and tools", Structured Programming, pg. 43-60, 1991.

[PRE88] Pressman R.S., Software Engineering - A Practitioner's Approach, 2ª edição, 1987, Mcgraw-Hill.

[RYA85] Ryan H.W., "Designer's Workbench", Systems Development Management, Auerbach Publishers Inc. 1985.

[SET83] Setzer V.W., Melo I.S.H., A Construção de um Compilador, Rio de Janeiro, Editora Campos, 1983.

[STA82] Stavely A.M., "Models as Executable Designs", Special Issue on Rapid Prototyping ACM Sigsoft Engineering Notes, vol.7, no.5, pg.167-168, December 1982.

[STA88] Stankovic J.A., "Misconceptions about Real-Time Computing A Serious Problem for Next Generation Systems", IEEE Computer, October 1988.

[STE90] Stephens M.A., Bates P.E., "Requirements engineering by prototyping experiences in development of estimating system", Information and Software Technology, vol.32, no.4, pg. 253-257, May 1990.

[STU87] Stus J.M., "Considerações sobre o Desenvolvimento por Protótipos", XX Congresso Nacional de Informática - SUCESU - SP, pg. 1044-1047, 1987.

[VAC86] Vacca J.R., "Evolutionary Systems Development : An Overview", Systems Development Management, Auerbach Publishers Inc., 1986.

[YOU89] Yourdon E., Modern Structured Analysis, Yourdon Press Computing Series, 1989.

[WAR85] Ward P.T., Mellor S., Structured Development for Real-Time Systems, vols. I, II, III, New-Jersey, Prentice-Hall, 1985.

[WAR86] Ward P.T., "The Transformation Schema : An Extension of the Data Flow Diagram to Represent Control and Timing", IEEE Transactions on Software Engineering, vol. SE 12, no.2, pg. 198-210, February 1986.

- ANEXO A -

EXEMPLO DO PROTR

A. EXEMPLO DO ProTR.

Neste anexo, para visualizar as fases de execução do sistema ProTR, é apresentado como exemplo o sistema "Engarrafamento" [WAR85].

A.1. Descrição do Sistema "Engarrafamento".

Este sistema consiste de uma Linha de Engarrafamento alimentada por um reservatório contendo o líquido a ser engarrafado. A figura A.1 mostra alguns detalhes deste sistema [WAR85].

As principais tarefas de controle do sistema consistem em gerenciar o nível do líquido no reservatório e seu pH, gerenciando o movimento e preenchimento dos vasilhames e trocando informações com o operador da linha e com o supervisor do sistema.

O nível do líquido no reservatório é controlado pelo monitoramento do sensor de nível e abertura/fechamento da válvula de entrada do líquido. O pH do líquido é monitorado por um sensor de pH; sempre que está fora dos limites, uma válvula de controle do pH libera pequenas quantidades de um líquido neutralizante para corrigir a discrepância entre o o nível necessário e o atual.

As garrafas a serem preenchidas seguem o seguinte fluxo :

- Uma garrafa é liberada para a plataforma de escala, sendo ativado um sensor.

- A válvula do reservatório é aberta e uma certa quantidade de líquido é depositada na garrafa. A plataforma de escala mede o peso da garrafa com seu conteúdo, ou seja, determinam quando está completa, o que provoca o fechamento da válvula.

- A garrafa é rotulada especificando o pH; o operador da linha tampa, remove a garrafa e sinaliza o sistema após esta última operação.

O operador da linha pode, manualmente, iniciar ou parar a linha, e o supervisor pode sinalizar o sistema habilitando ou desabilitando a operação.

Para o início de operação da linha, é necessário que a área esteja habilitada, o início de linha seja sinalizado, o contacto da garrafa esteja desligado e a escala da plataforma esteja registrando valor menor que 0.1 grama.

O supervisor da área é mantido informado a respeito do pH corrente e nível do reservatório, estando apto a alterar o pH, se necessário.

Se durante a operação do sistema, o pH está fora dos limites, todas ações de controle são suspensas.

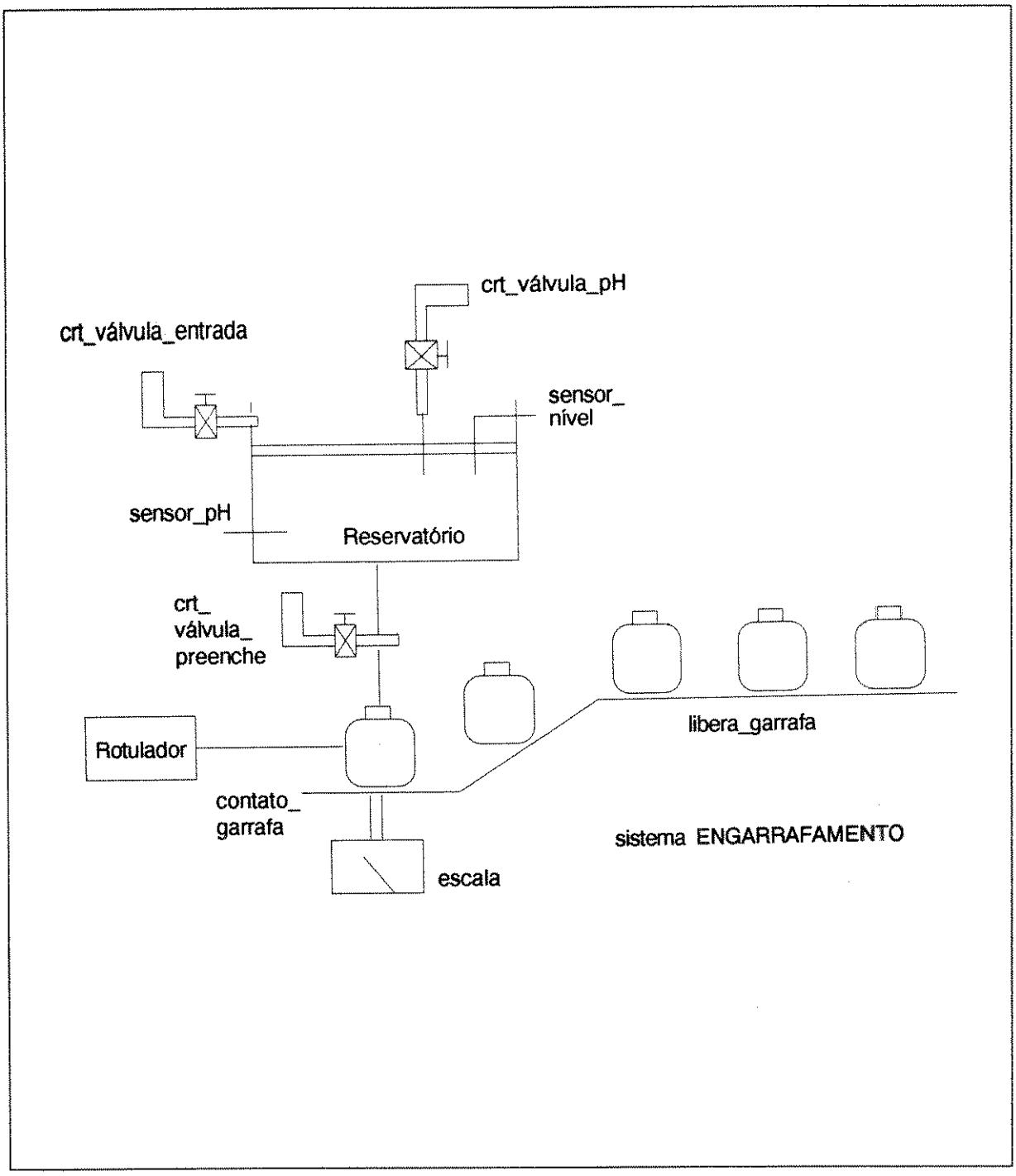


Figura A.1. Configuração do Equipamento.

O primeiro passo na modelagem é criar o Esquema Contexto, que resume os dados e eventos de controle definidos no modelo.

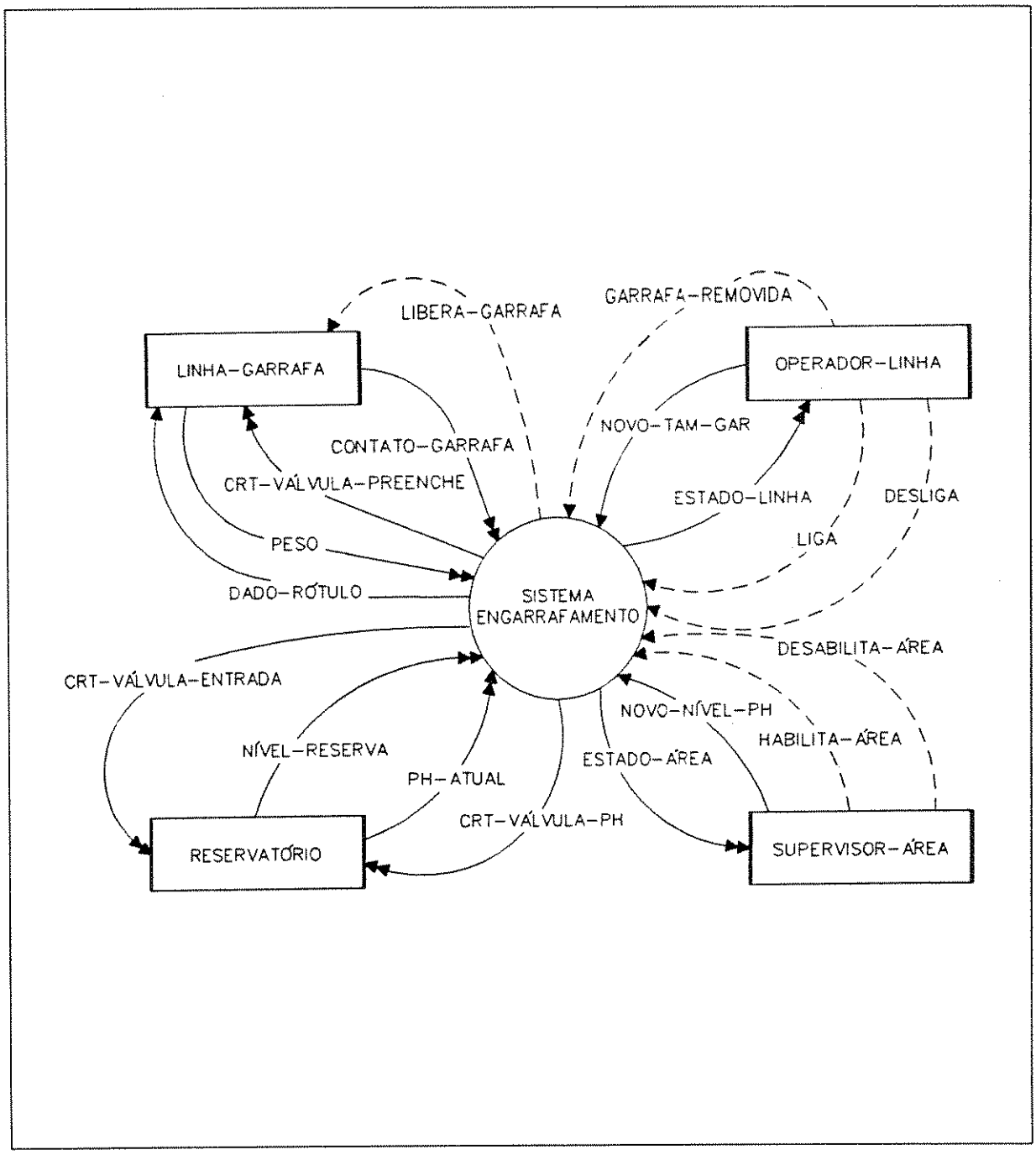


Figura A.2. Esquema Contexto.

A Figura A.3 apresenta a decomposição da única transformação do Esquema de Contexto para mostrar o 1º nível de detalhes do sistema.

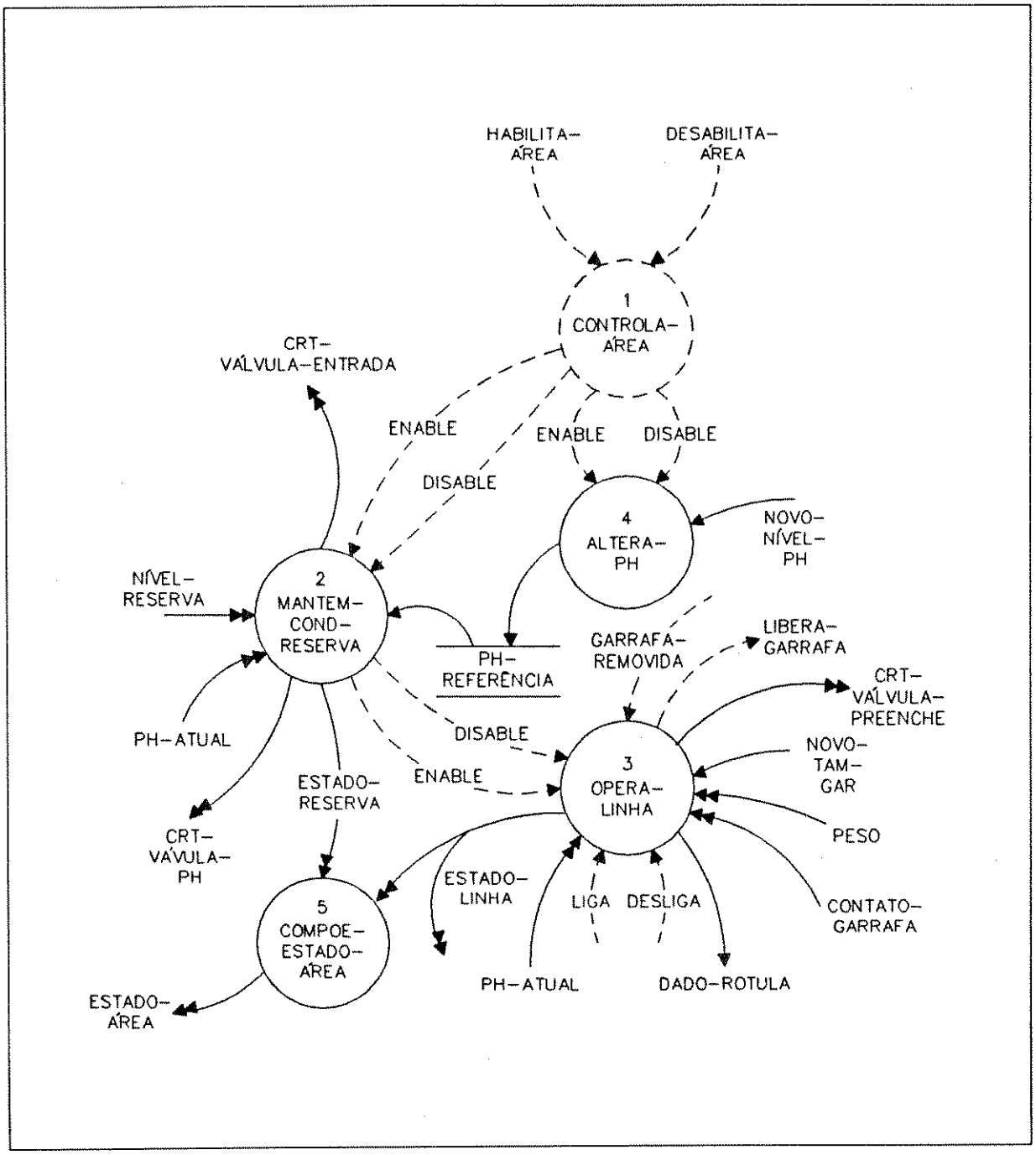


Figura A.3. Sistema Engarrafamento.

O primeiro passo na decomposição é criar uma transformação de controle que gerencia a área completa a nível de sistema [DEU88]. A figura A.4 mostra esta transformação de controle como um diagrama de transição de estados.

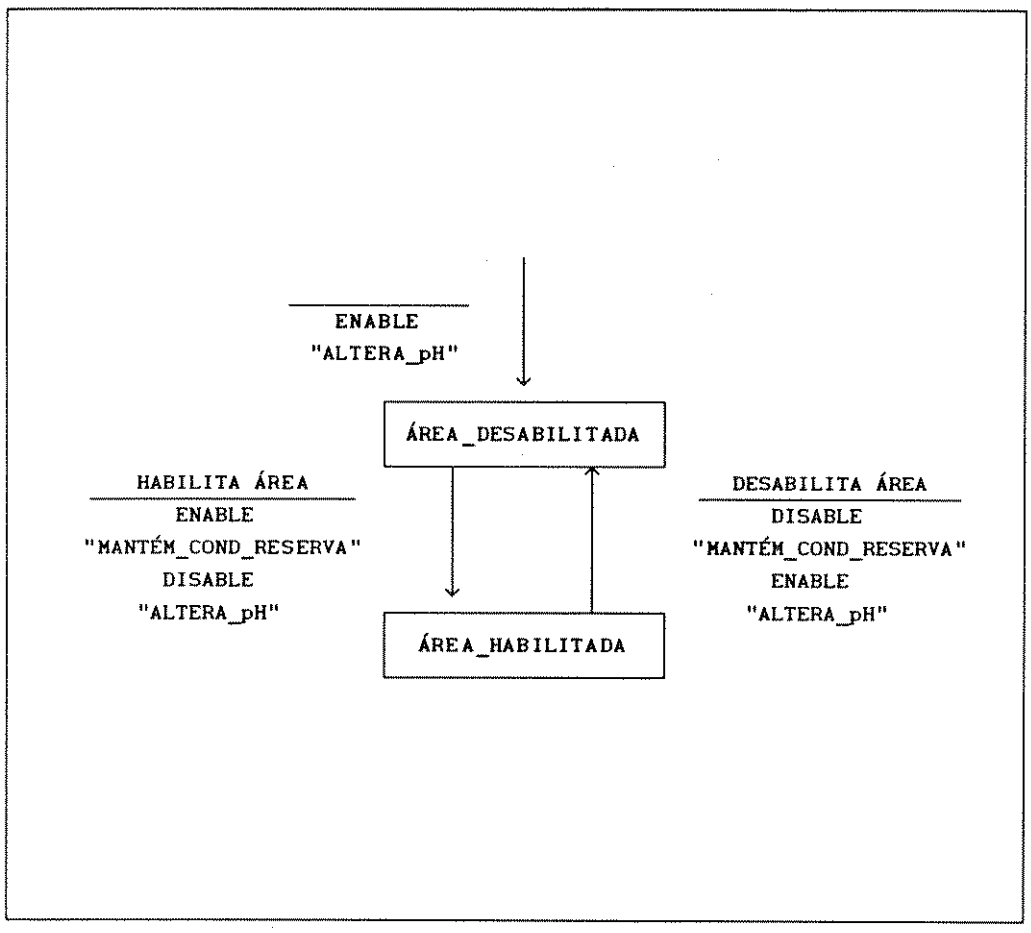


Figura A.4. Controla_area.

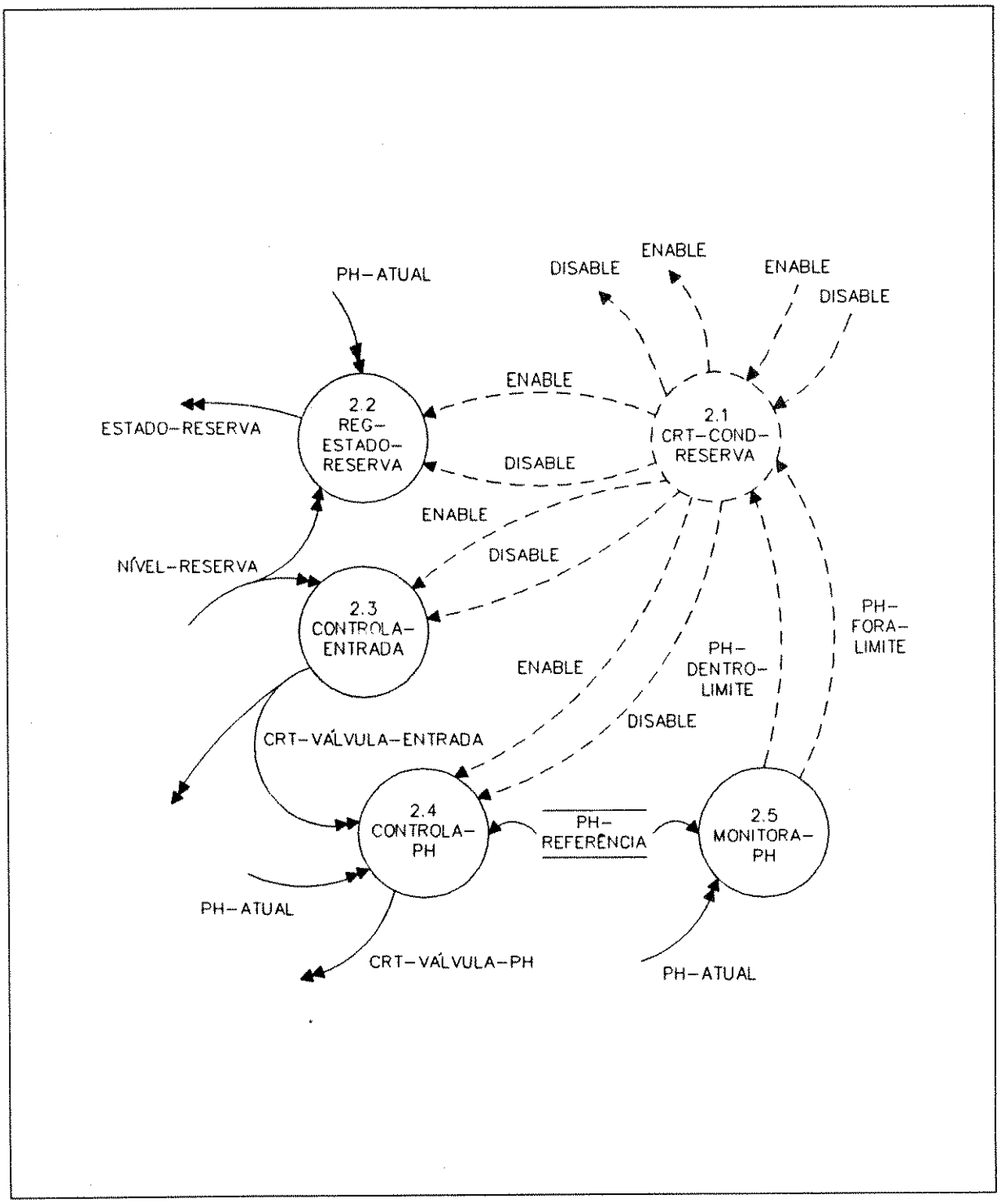


Figura A.5. Mantem_cond_reserva.

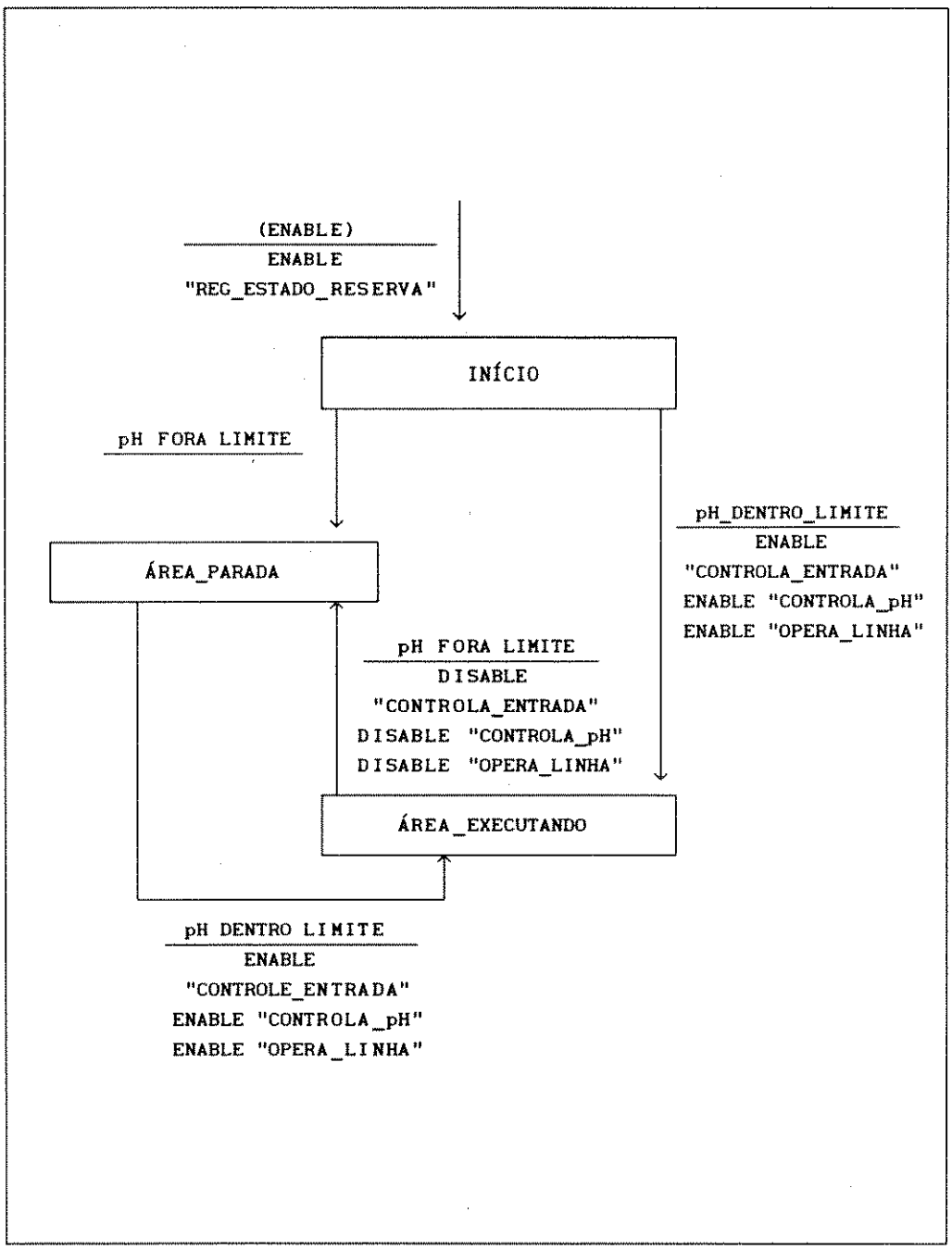


Figura A.6. Crt_cond_reserva.

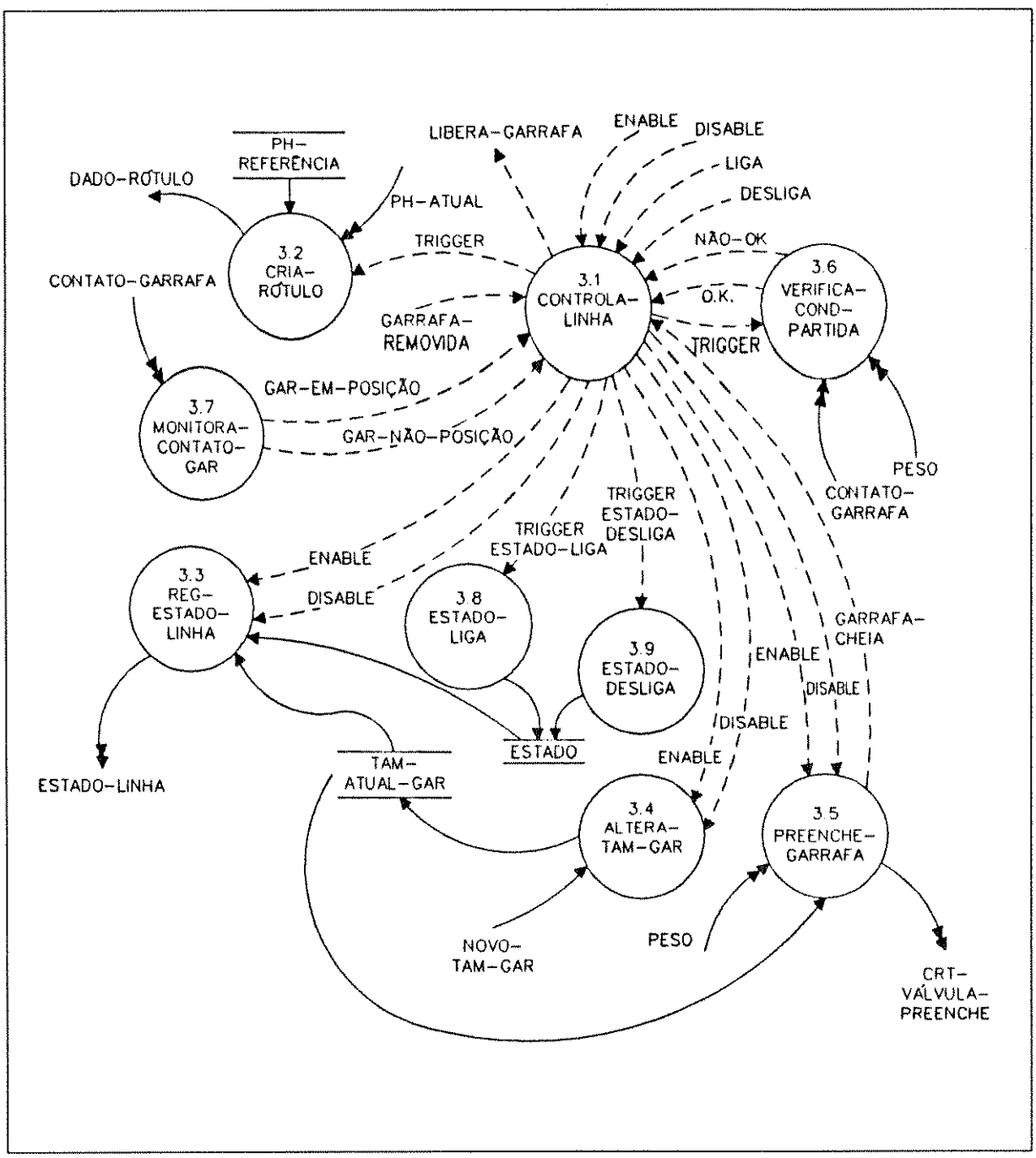


Figura A.7. Opera_linha.

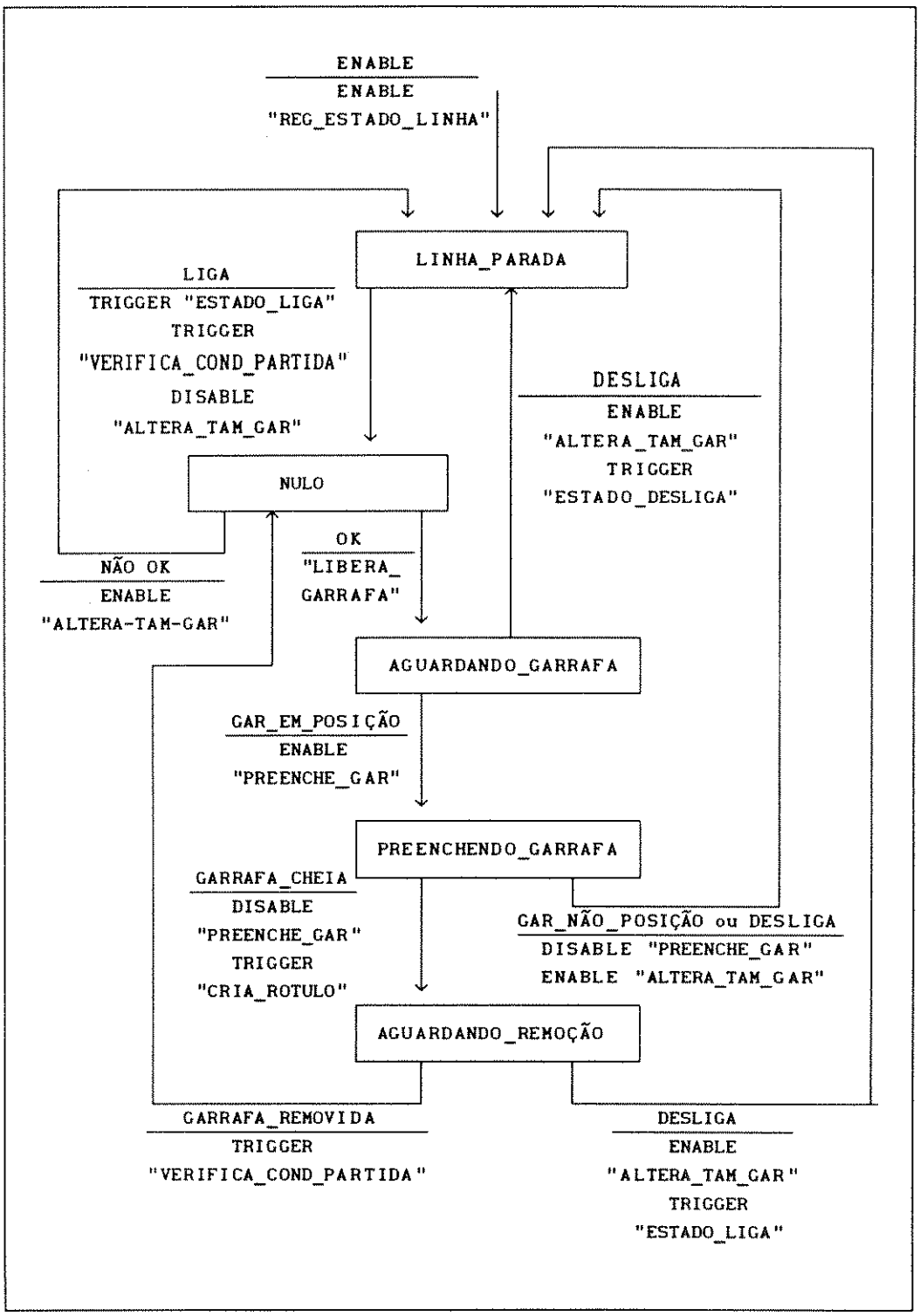


Figura A.8. Controla_linha.

A.2. Descrição do Esquema de Transformações.

Neste item apresenta-se o arquivo de descrição formal das transformações.

```
sistema ENGARRAFAMENTO;


/* Terminador linha_garrafa  -  Figura A.2 */
term linha_garrafa;
 {
  entrada
   {
    dis dado_rotulo;
    con crt_valvula_preenche;
    eve libera_garrafa;
   }
  saida
   {
    con peso;
    con contato_garrafa;
   }
 }



/* Terminador operador_linha  -  Figura A.2 */
term operador_linha;
 {
  entrada
   {
    con estado_linha;
   }
  saida
   {
    dis novo_tam_gar;
    eve garrafa_removida;
    eve liga;
    eve desliga;
   }
 }
```

```
/* Terminador reservatorio  -  Figura A.2 */
term reservatorio;
 {
  entrada
   {
    con crt_valvula_entrada;
    con crt_valvula_pH;
   }
  saida
   {
    con nivel_reserva;
    con pH_atual;
   }
 }


/* Terminador supervisor_area  -  Figura A.2 */
term supervisor_area;
 {
  entrada
   {
    con estado_area;
   }
  saida
   {
    dis novo_nivel_pH;
    eve habilita_area;
    eve desabilita_area;
   }
 }


/* Armazenador pH_referencia  -  Figura A.3 */
arm pH_referencia;
 {
  entrada
   {
    dis pH_referencia;
   }
  saida
   {
    dis pH_referencia;
   }
 }
```

```
/* Armazenador tam_atual_gar  -  Figura A.7 */
arm tam_atual_gar;
 {
  entrada
   {
    dis tam_atual_gar;
   }
  saida
   {
    dis tam_atual_gar;
   }
 }
```

```
/* Armazenador estado  -  Figura A.7 */
arm estado;
 {
  entrada
   {
    dis estado;
   }
  saida
   {
    dis estado;
   }
 }
```

```
/* Transformação sistema_engarrafamento  -  Figura A.2 */
tran sistema_engarrafamento (0);
 {
  entrada
   {
    con contato_garrafa;
    con peso;
    dis novo_tam_gar;
    eve garrafa_removida;
    eve liga;
    eve desliga;
    eve habilita_area;
    eve desabilita_area;
    dis novo_nivel_pH;
    con nivel_reserva;
    con pH_atual;
   }
```

```
  saida
   {
    dis dado_rotulo;
    con crt_valvula_preenche;
    eve libera_garrafa;
    con estado_linha;
    con crt_valvula_entrada;
    con crt_valvula_pH;
    con estado_area;
   }
 }


/* Transformação controla_area  -  Figura A.3 */
tran controla_area (1);
 {
  entrada
   {
    eve habilita_area;
    eve desabilita_area;
   }
  saida   {
    eve enable mantem_cond_reserva;
    eve disable mantem_cond_reserva;
    eve enable altera_pH;
    eve disable altera_pH;
   }
 }


/* Transformação mantem_cond_reserva  -  Figura A.3 */
tran mantem_cond_reserva (2);
 {
  entrada
   {
    eve enable mantem_cond_reserva;
    eve disable mantem_cond_reserva;
    dis pH_referencia;
    con pH_atual;
    con nivel_reserva;
   }
  saida
   {
    con crt_valvula_entrada;
    con crt_valvula_pH;
    con estado_reserva;
    eve enable opera_linha;
    eve disable opera_linha;
   }
 }
```

```
/* Transformação opera_linha  -  Figura A.3 */
tran opera_linha (3);
 {
  entrada   {
    eve garrafa_removida;
    dis novo_tam_gar;
    con peso;
    con contato_garrafa;
    eve desliga;
    eve liga;
    con pH_atual;
    eve enable opera_linha;
    eve disable opera_linha;
   }
  saida   {
    eve libera_garrafa;
    con crt_valvula_preenche;
    dis dado_rotulo;
    con estado_linha;
   }
 }


/* Transformação altera_pH  -  Figura A.3 */
tran altera_pH (4);
 {
  entrada   {
    eve enable altera_pH;
    eve disable altera_pH;
    dis novo_nivel_pH;
   }
  saida   {
    dis pH_referencia;
   }
 }


/* Transformação compoe_estado_area  -  Figura A.3 */
tran compoe_estado_area (5);
 {
  entrada
   {
    con estado_reserva;
    con estado_linha;
   }
  saida
   {
    con estado_area;
   }
 }
```

```
/* Transformação crt_cond_reserva  -  Figura A.5 */
tran crt_cond_reserva (2.1);
 {
  entrada
   {
    eve enable crt_cond_reserva;
    eve disable crt_cond_reserva;
    eve pH_dentro_limite;
    eve pH_fora_limite;
   }
  saida
   {
    eve enable opera_linha;
    eve disable opera_linha;
    eve enable reg_estado_reserva;
    eve disable reg_estado_reserva;
    eve enable controla_entrada;
    eve disable controla_entrada;
    eve enable controla_pH;
    eve disable controla_pH;
   }
 }


/* Transformação reg_estado_reserva  -  Figura A.5 */
tran reg_estado_reserva (2.2);
 {
  entrada   {
    eve enable reg_estado_reserva;
    eve disable reg_estado_reserva;
    con pH_atual;
    con nivel_reserva;
   }
  saida
   {
    con estado_reserva;
   }
 }


/* Transformação controla_entrada  -  Figura A.5 */
tran controla_entrada (2.3);
 {
  entrada
   {
    con nivel_reserva;
    eve enable controla_entrada;
    eve disable controla_entrada;
   }
```

```
  saida
   {
    con crt_valvula_entrada;
   }
 }


/* Transformação controla_pH  -  Figura A.5 */
tran controla_pH (2.4);
 {
  entrada
   {
    eve enable controla_pH;
    eve disable controla_pH;
    con crt_valvula_entrada;
    con pH_atual;
    dis pH_referencia;
   }
  saida
   {
    con crt_valvula_pH;
   }
 }


/* Transformação monitora_pH  -  Figura A.5 */
tran monitora_pH (2.5);
 {
  entrada
   {
    con pH_atual;
    dis pH_referencia;
   }
  saida
   {
    eve pH_dentro_limite;
    eve pH_fora_limite;
   }
 }
```

```
/* Transformação controla_linha  -  Figura A.7 */
tran controla_linha (3.1);
 {
  entrada
   {
    eve enable controla_linha;
    eve disable controla_linha;
    eve liga;
    eve desliga;
    eve nao_ok;
    eve ok;
    eve garrafa_removida;
    eve gar_em_posicao;
    eve gar_nao_posicao;
    eve garrafa_cheia;
   }
  saida
   {
    eve libera_garrafa;
    eve trigger verifica_cond_partida;
    eve trigger cria_rotulo;
    eve enable reg_estado_linha;
    eve disable reg_estado_linha;
    eve enable altera_tam_gar;
    eve disable altera_tam_gar;
    eve enable preenche_garrafa;
    eve disable preenche_garrafa;
    eve trigger estado_liga;
    eve trigger estado_desliga;
   }
 }


/* Transformação cria_rotulo  -  Figura A.7 */
tran cria_rotulo (3.2);
 {
  entrada
   {
    con pH_atual;
    eve trigger cria_rotulo;
    dis pH_referencia;
   }
  saida
   {
    dis dado_rotulo;
   }
 }
```

```
/* Transformação reg_estado_linha  -  Figura A.7 */
tran reg_estado_linha (3.3);
 {
  entrada
   {
    eve enable reg_estado_linha;
    eve disable reg_estado_linha;
    dis tam_atual_gar;
    dis estado;
   }
  saida
   {
    con estado_linha;
   }
 }


/* Transformação altera_tam_gar  -  Figura A.7 */
tran altera_tam_gar (3.4);
 {
  entrada
   {
    dis novo_tam_gar;
    eve enable altera_tam_gar;
    eve disable altera_tam_gar;
   }
  saida
   {
    dis tam_atual_gar;
   }
 }


/* Transformação preenche_garrafa  -  Figura A.7 */
tran preenche_garrafa (3.5);
 {
  entrada
   {
    eve enable preenche_garrafa;
    eve disable preenche_garrafa;
    dis tam_atual_gar;
    con peso;
   }
  saida
   {
    eve garrafa_cheia;
    con crt_valvula_preenche;
   }
 }
```

```
/* Transformação verifica_cond_partida  -  Figura A.7 */
tran verifica_cond_partida (3.6);
 {
  entrada
   {
    eve trigger verifica_cond_partida;
    con peso;
    con contato_garrafa;
   }
  saida
   {
    eve ok;
    eve nao_ok;
   }
 }


/* Transformação  monitora_contato_gar  -  Figura A.7 */
tran monitora_contato_gar (3.7);
 {
  entrada
   {
    con contato_garrafa;
   }
  saida
   {
    eve gar_em_posicao;
    eve gar_nao_posicao;
   }
 }


/* Transformação estado_liga  -  Figura A.7 */
tran estado_liga(3.8);
 {
  entrada
   {
    eve trigger estado_liga;
   }
  saida
   {
    dis estado;
   }
 }
```

```
/* Transformação estado_desliga  -  Figura A.7 */
tran estado_desliga(3.9);
 {
  entrada
   {
    eve trigger estado_desliga;
   }
  saida
   {
    dis estado;
   }
 }


FIM
```

A.3. Descrição do Dicionário de Dados.

O Dicionário de Dados permite completar as informações dos componentes da metodologia anteriormente definidos no arquivo de descrição formal dos elementos. As informações apresentadas referem-se ao tipo de cada fluxo, armazenador, e ao corpo das funções das transformações de dados e de controle. Com referência à descrição das transformações, nesta fase, são apenas apresentadas as de último nível (que não são mais explodidas).

```
sistema ENGARRAFAMENTO;

fluxo estado_reserva;
 {
  struct
   {
    float pH_atual;
    float nivel_reserva;
   }estado_reserva;
 }

fluxo estado_linha;
 {
  struct
   {
    int tam_atual_gar;
    unsigned char estado;
   }estado_linha;
 }

fluxo estado_area;
 {
  struct
   {
    float pH_atual;
    float nivel_reserva;
    int tam_atual_gar;
    unsigned char estado;
    }estado_area;
 }
```

```
fluxo crt_valvula_entrada;
 {
  float crt_valvula_entrada;        /* controle para valvula que libera o */
                                    /* liquido no reservatorio */
 }


fluxo nivel_reserva;
 {
  float nivel_reserva;
 }


fluxo pH_atual;
 {
  float pH_atual;                        /* pH do liquido no reservatorio */
 }


fluxo crt_valvula_pH;
 {
  float crt_valvula_pH;       /* controle permite alteracao quimica do pH */
 }


fluxo novo_nivel_pH;
 {
  float novo_nivel_pH;
 }


fluxo crt_valvula_preenche;
 {
  float crt_valvula_preenche;
 }


fluxo novo_tam_gar;
 {
  int novo_tam_gar;                                              /* litros */
 }


fluxo peso;
 {
  float peso;          /* peso registrado na escala preenchimento garrafa */
 }
```

```
fluxo contato_garrafa;
 {
  unsigned int contato_garrafa;                /* TRUE = 1    /   FALSE = 0 */
 }


fluxo dado_rotulo;
 {
  struct
   {
    float pH_atual;
    float pH_referencia;
   }dado_rotulo;
 }


arm pH_referencia;
 {
  float pH_referencia;
 }


arm estado;
 {
  unsigned char estado;
 }


arm tam_atual_gar;
 {
  int tam_atual_gar;
 }


tran controle controla_area (20,1);
 {
  est_inicial area_desabilitada : acao_inicial enable altera_pH;
  est_atual area_desabilitada;
   {
    (habilita_area,area_habilitada  :  enable  mantem_cond_reserva,  disable
    altera_pH)
   }
  est_atual area_habilitada;
   {
    (desabilita_area,  area_desabilitada  :  disable  mantem_cond_reserva,
    enable altera_pH)
   }
 }
```

```
tran discreto altera_pH (30,4);
 {
  pH_referencia = novo_nivel_pH;
  oswtds(pH_referencia);                          /* armazena novo valor do ph */
 }


tran controle crt_cond_reserva (20,2.1);
 {
  est_inicial inicio : acao_inicial enable reg_estado_reserva;
  est_atual inicio;
   {
    (pH_fora_limite, area_parada)
    (pH_dentro_limite, area_executando : enable controla_entrada, enable
    controla_pH,enable opera_linha)
   }
  est_atual area_parada;
   {
    (pH_dentro_limite, area_executando : enable controla_entrada, enable
    controla_pH,enable opera_linha)
   }
  est_atual area_executando;
   {
    (pH_fora_limite, area_parada : disable controla_entrada, disable
    controla_pH,disable opera_linha)
   }
 }


tran continua reg_estado_reserva (12,15,2.2);
 {
  estado_reserva.pH_atual = pH_atual;
  estado_reserva.nivel_reserva = nivel_reserva;
  ossenfc(estado_reserva);
 }


tran continua controla_entrada (22,15,2.3);
 {
  if(nivel_reserva < 0.83)
    crt_valvula_entrada = 1;                       /* abre valvula */
  else
    crt_valvula_entrada = 0;                       /* fecha valvula */
  ossenfc(crt_valvula_entrada);
 }
```

```
tran continua controla_pH (22,10,2.4);
 {
  crt_valvula_pH = 0;                                    /* mantem valvula */
  if(fabs(pH_atual - pH_referencia) > 0.3)              /* fabs-valor absoluto */
    crt_valvula_pH = 0;                                 /* fecha valvula do ph */
  else
    crt_valvula_pH = 1;                                    /* abre valvula */
  ossenfc(crt_valvula_pH);
 }


tran continua monitora_pH (22,10,2.5);
 {
  if(fabs(pH_atual - pH_referencia) > 0.3)            /* fabs vr. absoluto */
    osnote(pH_fora_limite);
  else
    osnote(pH_dentro_limite);
 }


tran controle controla_linha (20,3.1);
 {
  est_inicial linha_parada : acao_inicial enable reg_estado_linha;
  est_atual linha_parada;
   {
    (liga, nulo : trigger estado_liga, trigger verifica_cond_partida,
    disable altera_tam_gar)
   }
  est_atual nulo;
   {
    (nao_ok, linha_parada : enable altera_tam_gar)
    (ok, aguardando_garrafa : libera_garrafa)
   }
  est_atual aguardando_garrafa;
   {
    (gar_em_posicao, preenchendo_gar : enable preenche_garrafa)
    (desliga, linha_parada : trigger estado_desliga, enable altera_tam_gar)
   }
  est_atual preenchendo_gar;
   {
    (garrafa_cheia, aguardando_remocao : disable preenche_garrafa,trigger
    cria_rotulo)
    (gar_nao_posicao, linha_parada : disable preenche_garrafa, enable
    altera_tam_gar)
    (desliga, linha_parada : disable preenche_garrafa, enable
    altera_tam_gar)
   }
```

```
  est_atual aguardando_remocao;
   {
    (desliga, linha_parada : trigger estado_desliga, enable altera_tam_gar)
    (garrafa_removida, nulo : trigger verifica_cond_partida)
   }
 }


tran continua cria_rotulo (22,0,3.2);
 {
  dado_rotulo.pH_atual = pH_atual;
  dado_rotulo.pH_referencia = pH_referencia;
  ossenfw(dado_rotulo);
 }


tran continua reg_estado_linha (22,15,3.3);
 {
  estado_linha.tam_atual_gar = tam_atual_gar;
  if(estado == 0)
    estado_linha.estado = FALSE;                          /* linha desligada */
  else
    estado_linha.estado = TRUE;                             /* linha ligada */
  ossenfc(estado_linha);
 }


tran discreto altera_tam_gar(30,3.4);
 {
  tam_atual_gar = novo_tam_gar;
  oswtds(tam_atual_gar);
 }


tran continua preenche_garrafa (12,10,3.5);
 {
  float peso_total;
  peso_total = tam_atual_gar;

  if(peso < (peso_total * 0.98) )/* 99.9% preenchido */
    {
     crt_valvula_preenche = 1;                  /* abre valvula de controle */
     ossenfc(crt_valvula_preenche);
    }
  else
```

```
    {
     crt_valvula_preenche = 0;                    /* fecha valvula de controle */
     ossenfc(crt_valvula_preenche);
     osnote(garrafa_cheia);
    }
 }


tran continua verifica_cond_partida (12,0,3.6);
 {
  if(peso <= 0.19 && contato_garrafa == 0)
    osnote(ok);
  if(peso > 0.19 || contato_garrafa == 1)
    osnote(nao_ok);
 }


tran continua monitora_contato_gar (12,10,3.7);
 {
  if(contato_garrafa == 1)
    osnote(gar_em_posicao);
  else
    osnote(gar_nao_posicao);
 }


tran discreto estado_liga (30,3.8);
 {
  estado = 1;                                          /* linha ON */
  oswtds(estado);
 }


tran discreto estado_desliga (30,3.9);
 {
  estado = 0;                                        /* linha = OFF */
  oswtds(estado);
 }


tran continua compoe_estado_area(22,15,5);
 {
  estado_area.pH_atual = estado_reserva.pH_atual;
  estado_area.nivel_reserva = estado_reserva.nivel_reserva;
  estado_area.tam_atual_gar = estado_linha.tam_atual_gar;
  estado_area.estado = estado_linha.estado;
  ossenfc(estado_area);
 }

FIM
```

A.4. Código Fonte Protótipo.

Após a execução do ProTR, os módulos de código fonte gerados são :

```
/*********************************************************************/
/*                                                                   */
/* ProTR - Prototipador para Sistemas de Tempo Real                  */
/* Nome do Sistema: ENGARRAFAMENTO                                   */
/* Nome do Modulo: ENGARR.H                                          */
/*                                                                   */
/* Centro Tecnologico para Informatica - CTI                         */
/* Instituto de Automacao - IA                                       */
/*                                                                   */
/* Analista Responsavel: Glaucia Dantas Franco Azevedo               */
/*                                                                   */
/* Data: 09/06/91                                                    */
/* 1991 - Todos os direitos reservados.                              */
/*********************************************************************/


typedef struct {
float pH_atual ;
float nivel_reserva ;
} tyestado_reserva;


typedef struct {
int tam_atual_gar ;
unsigned char estado ;
} tyestado_linha;


typedef struct {
float pH_atual ;
float nivel_reserva ;
int tam_atual_gar ;
unsigned char estado ;
} tyestado_area;


typedef struct {
float pH_atual ;
float pH_referencia ;
} tydado_rotulo;
```

```
/* transformacoes do sistema */

extern void compoe_estado_area(void);
extern void altera_pH(void);

extern void estado_desliga(void);
extern void estado_liga(void);
extern void monitora_contato_gar(void);
extern void verifica_cond_partida(void);
extern void preenche_garrafa(void);
extern void altera_tam_gar(void);
extern void reg_estado_linha(void);
extern void cria_rotulo(void);
extern void controla_linha(void);
extern void monitora_pH(void);
extern void controla_pH(void);
extern void controla_entrada(void);
extern void reg_estado_reserva(void);
extern void crt_cond_reserva(void);
extern void controla_area(void);
```

```
/*********************************************************************/
/*                                                                   */
/* ProTR - Prototipador para Sistemas de Tempo Real                  */
/* Nome do Sistema: ENGARRAFAMENTO                                   */
/* Nome do Modulo: ENGARR01.C                                        */
/*                                                                   */
/* Centro Tecnologico para Informatica - CTI                         */
/* Instituto de Automacao - IA                                       */
/*                                                                   */
/* Analista Responsavel: Glaucia Dantas Franco Azevedo               */
/*                                                                   */
/* Data: 09/06/91                                                    */
/* 1991 - Todos os direitos reservados.                              */
/*********************************************************************/


#include "NUCLEO.H"
#include "BOTTLE.H"
#include <math.h>
#include <string.h>
#include <time.h>

#include <stdio.h>

extern FILE *osresult;/* ponteiro p/arquivo de resultados */
extern FILE *oscena;/* ponteiro p/arquivo de cenario */

struct tyauto _controla_area[ ] = {      /* Inicializa tabela do automato */
        { 0, 0, 1,
                {2,3,EMPTY},
                {7,6,EMPTY} },
        { 1, 1, 0,
                {3,2,EMPTY},
                {7,6,EMPTY} },
};


/*
 *  NOME DA TRANSFORMACAO DE CONTROLE: controla_area
 */

void controla_area(void)
{

short estatual = 0;

/* ACOES INICIAIS */

osenable(6);
```

```
for(;;) {  /* Comando 'for' p/automato */
osautomato(_controla_area,&estatual);
}
}


void main(int argc, char *argv[])
{

systag_t pid;
char data[9];
char hora[9];
systag_t trans[NENTF];
systag_t fluxos[MAXFLUX];
char cena_file[32]; /* arquivo descricao cenario */
char res_file[32]; /* arquivo res. prototipacao */

if(argc < 2)  {
  fprintf(stderr,"ERROAM1  -  Faltaram  parametros  !!!  -  formato  =  sistema
          <arq>\n");
  exit (1);
  }


/* abrir arquivo de cenario */

strcpy(cena_file,argv[1]);
strcat(cena_file,".cen");
if((oscena = fopen((char*)cena_file,"r")) == NULL)
   {
    fprintf(stderr,"ERRO/AM2 - Nao abriu arquivo cenario %s\n",cena_file);
    return;
    }


/* abre arquivo de saida */

strcpy(res_file,argv[1]);
strcat(res_file,".sai");
if((osresult = fopen((char *)res_file,"w")) == NULL)
   {
    fprintf(stderr,"ERRO/AM3 - Nao abriu arquivo de saida %s\n",res_file);
    return;
    }


_strdate(data);
_strtime(hora);
fprintf(osresult,"RESULTADO DA EXECUCAO DO CENARIO %s\n\n\n",argv[1]);
fprintf(osresult,"DATA:%s  HORA:%s\n\n",data,hora);
```

```
/* inicializa nucleo */

pid = osintpg("main",1,1);

/* inicia tarefas do nucleo */

fluxos[0] = EMPTY;
oscrepr("ostimp",2,1,ostimp,2,(long)0,(long)800,fluxos);
oscrepr("osinterhm",3,1,osinterhm,3,(long)0,(long)800,fluxos);
oscrepr("oslecenario",4,1,osanasin,4,(long)0,(long)800,fluxos);

/* aloca semaforo que marcara entradas do cenario */

osalcena();


osnomesis = "ENGARRAFAMENTO";


/* Transformacoes do prototipo */

fluxos[0] = EMPTY;
oscrepr("compoe_estado_area",21,2,compoe_estado_area,22,(long)15,
(long)800,fluxos);
fluxos[0] = 0; /* fluxo = novo_nivel_pH */
fluxos[1] = EMPTY;
oscrepr("altera_pH",6,4,altera_pH,30,(long)0,(long)800,fluxos);
fluxos[0] = EMPTY;
oscrepr("estado_desliga",18,5,estado_desliga,30,(long)0,(long)800, fluxos);
fluxos[0] = EMPTY;
oscrepr("estado_liga",14,5,estado_liga,30,(long)0,(long)800,fluxos);
fluxos[0] = EMPTY;
oscrepr("monitora_contato_gar",20,2,monitora_contato_gar,12,(long)10,
(long)800,fluxos);
fluxos[0] = EMPTY;
oscrepr("verifica_cond_partida",15,7,verifica_cond_partida,12,(long)0,
(long)800,fluxos);
fluxos[0] = EMPTY;
oscrepr("preenche_garrafa",17,6,preenche_garrafa,12,(long)10,
(long)800,fluxos);
fluxos[0] = 2; /* fluxo = novo_tam_gar */
fluxos[1] = EMPTY;
oscrepr("altera_tam_gar",16,4,altera_tam_gar,30,(long)0, (long)800,fluxos);
fluxos[0] = EMPTY;
oscrepr("reg_estado_linha",13,6,reg_estado_linha,22,(long)15,
(long)800,fluxos);
fluxos[0] = EMPTY;
oscrepr("cria_rotulo",19,7,cria_rotulo,22,(long)0,(long)800,fluxos);
fluxos[0] = 4; /* fluxo evento = liga */
fluxos[1] = 9; /* fluxo evento = desliga */
```

```
fluxos[2] = 5; /* fluxo evento = nao_ok */
fluxos[3] = 6; /* fluxo evento = ok */
fluxos[4] = 12; /* fluxo evento = garrafa_removida */
fluxos[5] = 8; /* fluxo evento = gar_em_posicao */
fluxos[6] = 11; /* fluxo evento = gar_nao_posicao */
fluxos[7] = 10; /* fluxo evento = garrafa_cheia */
fluxos[8] = EMPTY;
oscrepr("controla_linha",11,8,controla_linha,20,(long)0,(long)800, fluxos);
fluxos[0] = EMPTY;
oscrepr("monitora_pH",12,2,monitora_pH,22,(long)10,(long)800,fluxos);
fluxos[0] = EMPTY;
oscrepr("controla_pH",10,6,controla_pH,22,(long)10,(long)800,fluxos);
fluxos[0] = EMPTY;
oscrepr("controla_entrada",9,6,controla_entrada,22,(long)15,
(long)800,fluxos);
fluxos[0] = EMPTY;
oscrepr("reg_estado_reserva",8,6,reg_estado_reserva,12,(long)15,
(long)800,fluxos);
fluxos[0] = 3; /* fluxo evento = pH_dentro_limite */
fluxos[1] = 2; /* fluxo evento = pH_fora_limite */
fluxos[2] = EMPTY;
oscrepr("crt_cond_reserva",7,8,crt_cond_reserva,20,(long)0,(long)800,
fluxos);
fluxos[0] = 0; /* fluxo evento = habilita_area */
fluxos[1] = 1; /* fluxo evento = desabilita_area */
fluxos[2] = EMPTY;
oscrepr("controla_area",5,3,controla_area,20,(long)0,(long)800,fluxos) ;


/* inicia fluxos discretos */

trans[0] = 6; /* Transformacao = altera_pH */
trans[1] = EMPTY;
osalflxd("novo_nivel_pH",0,sizeof(float  ),2,TRUE,trans);

trans[0] = 16; /* Transformacao = altera_tam_gar */
trans[1] = EMPTY;
osalflxd("novo_tam_gar",2,sizeof(int  ),3,TRUE,trans);

trans[0] = EMPTY;
osalflxd("dado_rotulo",1,sizeof(tydado_rotulo  ),9,TRUE,trans);



/* inicia fluxos continuos */

osalflxc("estado_reserva",2,sizeof(tyestado_reserva  ),9,FALSE);

osalflxc("estado_area",9,sizeof(tyestado_area  ),9,TRUE);
```

```
osalflxc("pH_atual",0,sizeof(float  ),2,TRUE);

osalflxc("nivel_reserva",1,sizeof(float  ),2,TRUE);

osalflxc("crt_valvula_pH",4,sizeof(float  ),2,TRUE);

osalflxc("crt_valvula_entrada",3,sizeof(float  ),2,TRUE);

osalflxc("estado_linha",5,sizeof(tyline_status  ),9,TRUE);

osalflxc("contato_garrafa",8,sizeof(unsigned int  ),6,TRUE);

osalflxc("peso",6,sizeof(float  ),2,TRUE);

osalflxc("crt_valvula_preenche",7,sizeof(float  ),2,TRUE);


 /* inicia fluxos de eventos */

trans[0] = 11; /* Transformacao = controla_linha */
trans[1] = EMPTY;
osalev("garrafa_cheia",10,FALSE,trans[0]);

trans[0] = 11; /* Transformacao = controla_linha */
trans[1] = EMPTY;
osalev("gar_nao_posicao",11,FALSE,trans[0]);

trans[0] = 11; /* Transformacao = controla_linha */
trans[1] = EMPTY;
osalev("gar_em_posicao",8,FALSE,trans[0]);

trans[0] = 11; /* Transformacao = controla_linha */
trans[1] = EMPTY;
osalev("ok",6,FALSE,trans[0]);

trans[0] = 11; /* Transformacao = controla_linha */
trans[1] = EMPTY;
osalev("nao_ok",5,FALSE,trans[0]);

trans[0] = 7; /* Transformacao = crt_cond_reserva */
trans[1] = EMPTY;
osalev("pH_fora_limite",2,FALSE,trans[0]);

trans[0] = 7; /* Transformacao = crt_cond_reserva */
trans[1] = EMPTY;
osalev("pH_dentro_limite",3,FALSE,trans[0]);

trans[0] = 5; /* Transformacao = controla_area */
trans[1] = EMPTY;
osalev("desabilita_area",1,TRUE,trans[0]);
```

```
trans[0] = 5; /* Transformacao = controla_area */
trans[1] = EMPTY;
osalev("habilita_area",0,TRUE,trans[0]);

trans[0] = 11; /* Transformacao = controla_linha */
trans[1] = EMPTY;
osalev("desliga",9,TRUE,trans[0]);

trans[0] = 11; /* Transformacao = controla_linha */
trans[1] = EMPTY;
osalev("liga",4,TRUE,trans[0]);

trans[0] = 11; /* Transformacao = controla_linha */
trans[1] = EMPTY;
osalev("garrafa_removida",12,TRUE,trans[0]);

trans[0] = EMPTY;
osalev("libera_garrafa",7,TRUE,trans[0]);


/* inicia armazenadores de dados */

osalarm("estado",2,sizeof(unsigned char  ),0);

osalarm("tam_atual_gar",1,sizeof(int  ),3);

osalarm("pH_referencia",0,sizeof(float  ),2);


/* inicia armazenadores de eventos */


/*suspende processo principal 'main' */

ossuspr(pid);
}
```

```
/*********************************************************************/
/*                                                                   */
/* ProTR - Prototipador para Sistemas de Tempo Real                  */
/* Nome do Sistema: ENGARRAFAMENTO                                   */
/* Nome do Modulo: ENGARR02.C                                        */
/*                                                                   */
/* Centro Tecnologico para Informatica - CTI                         */
/* Instituto de Automacao - IA                                       */
/*                                                                   */
/* Analista Responsavel: Glaucia Dantas Franco Azevedo               */
/*                                                                   */
/* Data: 09/06/91                                                    */
/* 1991 - Todos os direitos reservados.                              */
/*********************************************************************/


#include "NUCLEO.H"
#include "BOTTLE.H"
#include <math.h>
#include <string.h>
#include <time.h>

struct tyauto _crt_cond_reserva[ ] = {  /* Inicializa tabela do automato */
        { 0, 2, 2,
                {EMPTY},
                {EMPTY} },
        { 0, 3, 3,
                {2,2,2,EMPTY},
                {9,10,11,EMPTY} },
        { 2, 3, 3,
                {2,2,2,EMPTY},
                {9,10,11,EMPTY} },
        { 3, 2, 2,
                {3,3,3,EMPTY},
                {9,10,11,EMPTY} },
};


/*
 *  NOME DA TRANSFORMACAO DE CONTROLE: crt_cond_reserva
 */

 void crt_cond_reserva(void)
 {

 short estatual = 0;
```

```
/* ACOES INICIAIS E PRIMEIRA EXECUCAO */

oswaien();
osenable(8);
osautomato(_crt_cond_reserva,&estatual);/* Primeira Execucao */

for(;;) {  /* Comando 'for' p/automato */
oswaien();
osautomato(_crt_cond_reserva,&estatual);
}
}


/*
 *  NOME DA TRANSFORMACAO DE DADOS: reg_estado_reserva
 */

void reg_estado_reserva(void)
{
float pH_atual ;
float nivel_reserva ;
tyestado_reserva estado_reserva ;

oswaien();  /* WAIT ENABLE */
osrcfc(0,&pH_atual);  /* RECEIVE FLOW CONTINUOUS */
osrcfc(1,&nivel_reserva);  /* RECEIVE FLOW CONTINUOUS */

estado_reserva .pH_atual =pH_atual ;
estado_reserva.nivel_reserva =nivel_reserva ;
ossenfc (2, &estado_reserva);
}


/*
 *  NOME DA TRANSFORMACAO DE DADOS: controla_entrada
 */

void controla_entrada(void)
{
float nivel_reserva ;
float crt_valvula_entrada ;

oswaien();  /* WAIT ENABLE */
osrcfc(1,&nivel_reserva);  /* RECEIVE FLOW CONTINUOUS */
if (nivel_reserva <0.83)
crt_valvula_entrada =1;
else
crt_valvula_entrada =0;
ossenfc (3, &crt_valvula_entrada);
}
```

```
/*
 *  NOME DA TRANSFORMACAO DE DADOS: controla_pH
 */

void controla_pH(void)
{
float crt_valvula_entrada ;
float pH_atual ;
float pH_referencia ;
float crt_valvula_pH ;
oswaien();  /* WAIT ENABLE */
osrcfc(3,&crt_valvula_entrada);  /* RECEIVE FLOW CONTINUOUS */
osrcfc(0,&pH_atual);  /* RECEIVE FLOW CONTINUOUS */
osrdds(0,&pH_referencia);  /* READ DATA STORE */

ph_valve_ctl =0;
if (fabs (pH_atual -pH_referencia )>0.3)
crt_valvula_pH =0;
else
crt_valvula_pH =1;
ossenfc (4, &crt_valvula_pH);
}


/*
 *  NOME DA TRANSFORMACAO DE DADOS: monitora_pH
 */

void monitora_pH(void)
{
float pH_atual ;
float pH_referencia ;
osrcfc(0,&pH_atual);  /* RECEIVE FLOW CONTINUOUS */
osrdds(0,&pH_referencia);  /* READ DATA STORE */

if (fabs (pH_atual -pH_referencia )>0.3)
osnote (2);
else
osnote (3);
}
```

```
/*********************************************************************/
/*                                                                   */
/* ProTR - Prototipador para Sistemas de Tempo Real                  */
/* Nome do Sistema: ENGARRAFAMENTO                                   */
/* Nome do Modulo: ENGARR03.C                                        */
/*                                                                   */
/* Centro Tecnologico para Informatica - CTI                         */
/* Instituto de Automacao - IA                                       */
/*                                                                   */
/* Analista Responsavel: Glaucia Dantas Franco Azevedo               */
/*                                                                   */
/* Data: 09/06/91                                                    */
/* 1991 - Todos os direitos reservados.                              */
/*********************************************************************/

#include "NUCLEO.H"
#include "BOTTLE.H"
#include <math.h>
#include <string.h>
#include <time.h>

struct tyauto _controla_linha[ ] = {/* Inicializa tabela do automato*/
        { 0, 4, 1,
                {1,1,3,EMPTY},
                {14,15,16,EMPTY} },
        { 1, 5, 0,
                {2,EMPTY},
                {16,EMPTY} },
        { 1, 6, 3,
                {4,EMPTY},
                {7,EMPTY} },
        { 3, 8, 5,
                {2,EMPTY},
                {17,EMPTY} },
        { 3, 9, 0,
                {1,2,EMPTY},
                {18,16,EMPTY} },
        { 5, 10, 8,
                {3,1,EMPTY},
                {17,19,EMPTY} },
        { 5, 11, 0,
                {3,2,EMPTY},
                {17,16,EMPTY} },
        { 5, 9, 0,
                {3,2,EMPTY},
                {17,16,EMPTY} },
        { 8, 9, 0,
                {1,2,EMPTY},
                {18,16,EMPTY} },
```

```
        { 8, 12, 1,
                {1,EMPTY},
                {15,EMPTY} },
};


/*
 *  NOME DA TRANSFORMACAO DE CONTROLE: controla_linha
 */

 void controla_linha(void)
 {

 short estatual = 0;

/* ACOES INICIAIS E PRIMEIRA EXECUCAO */

oswaien();
osenable(13);
osautomato(_controla_linha,&estatual);/* Primeira Execucao */


for(;;) {  /* Comando 'for' p/automato */
oswaien();
osautomato(_controla_linha,&estatual);
}
}


/*
 *  NOME DA TRANSFORMACAO DE DADOS: cria_rotulo
 */

void cria_rotulo(void)
{
float pH_atual ;
float pH_referencia ;
tydado_rotulo dado_rotulo ;

for(;;) {  /* Inicio comando 'for' do prototipador */

oswaitg();  /* WAIT TRIGGER */
osrcfc(0,&pH_atual);  /* RECEIVE FLOW CONTINUOUS */
osrdds(0,&pH_referencia);  /* READ DATA STORE */

dado_rotulo .pH_atual =pH_atual ;
dado_rotulo.pH_referencia =pH_referencia ;
ossenfw (1, &dado_rotulo);
}
}
```

```
/*
 *  NOME DA TRANSFORMACAO DE DADOS: reg_estado_linha
 */

void reg_estado_linha(void)
{
int tam_atual_gar ;
unsigned char estado ;
tyestado_linha estado_linha ;

oswaien();  /* WAIT ENABLE */
osrdds(1,&tam_atual_gar);  /* READ DATA STORE */
osrdds(2,&estado);  /* READ DATA STORE */

estado_linha.tam_atual_gar = tam_atual_gar ;
if (estado ==0)
estado_linha.estado =FALSE ;
else
estado_linha.estado =TRUE ;
ossenfc (5, &estado_linha);
}



/*
 *  NOME DA TRANSFORMACAO DE DADOS: altera_tam_gar
 */

void altera_tam_gar(void)
{
int novo_tam_gar ;
int tam_atual_gar ;

for(;;) {   /* Inicio comando 'for' do prototipador */
oswaien();  /* WAIT ENABLE */
osrcfw(2,&novo_tam_gar);  /* RECEIVE FLOW */

tam_atual_gar = novo_tam_gar ;
oswtds (1, &tam_atual_gar);
}
}
```

```
/*
 *  NOME DA TRANSFORMACAO DE DADOS: preenche_garrada
 */

void preenche_garrafa(void)
{
int tam_atual_gar ;
float peso ;
float crt_valvula_preenche ;
float peso_total ;

oswaien();  /* WAIT ENABLE */
osrdds(1,&tam_atual_gar);  /* READ DATA STORE */
osrcfc(6,&peso);  /* RECEIVE FLOW CONTINUOUS */
peso_total = tam_atual_gar ;
if (peso <(peso_total *0.98))
{
crt_valvula_preenche =1;
ossenfc (7, &crt_valvula_preenche);
}
else
{
crt_valvula_preenche =0;

ossenfc (7, &crt_valvula_preenche);
osnote (10);
}
}


/*
 *  NOME DA TRANSFORMACAO DE DADOS: verifica_cond_partida
 */

void verifica_cond_partida(void)
{
float peso ;
unsigned int contato_garrafa ;

for(;;) {  /* Inicio comando 'for' do prototipador */

oswaitg();  /* WAIT TRIGGER */
osrcfc(6,&peso);  /* RECEIVE FLOW CONTINUOUS */
osrcfc(8,&contato_garrafa);  /* RECEIVE FLOW CONTINUOUS */
if (peso <=0.19&&contato_garrafa ==0)
osnote (6);
if (peso >0.19||contato_garrafa ==1)
osnote (5);
}
}
```

```
/*
 *  NOME DA TRANSFORMACAO DE DADOS: monitora_contato_gar
 */

void monitora_contato_gar(void)
{
unsigned int contato_garrafa ;
osrcfc(8,&contato_garrafa);  /* RECEIVE FLOW CONTINUOUS */

if (contato_garrafa ==1)
osnote (8);
else
osnote (11);
}


/*
 *  NOME DA TRANSFORMACAO DE DADOS: estado_liga
 */

void estado_liga(void)
{
unsigned char estado ;

for(;;) {  /* Inicio comando 'for' do prototipador */

oswaitg();  /* WAIT TRIGGER */

estado =1;
oswtds (2, &estado);
}
}


/*
 *  NOME DA TRANSFORMACAO DE DADOS: estado_desliga
 */

void estado_desliga(void)
{
unsigned char estado ;

for(;;) {  /* Inicio comando 'for' do prototipador */
oswaitg();  /* WAIT TRIGGER */


estado =0;
oswtds (2, &estado);
}
}
```

```
/*********************************************************************/
/*                                                                   */
/* ProTR - Prototipador para Sistemas de Tempo Real                  */
/* Nome do Sistema: ENGARRAFAMENTO                                   */
/* Nome do Modulo: ENGARR04.C                                        */
/*                                                                   */
/* Centro Tecnologico para Informatica - CTI                         */
/* Instituto de Automacao - IA                                       */
/*                                                                   */
/* Analista Responsavel: Glaucia Dantas Franco Azevedo               */
/*                                                                   */
/* Data: 09/06/91                                                    */
/* 1991 - Todos os direitos reservados.                              */
/*********************************************************************/


#include "NUCLEO.H"
#include "BOTTLE.H"
#include <math.h>
#include <string.h>
#include <time.h>


/*
 *  NOME DA TRANSFORMACAO DE DADOS: altera_pH
 */

void altera_pH(void)
{
float novo_nivel_pH ;
float pH_referencia ;

for(;;) {   /* Inicio comando 'for' do prototipador */
oswaien();  /* WAIT ENABLE */
osrcfw(0,&novo_nivel_pH);  /* RECEIVE FLOW */

pH_referencia =novo_nivel_pH ;
oswtds (0, &pH_referencia);
}
}
```

```
/*********************************************************************/
/*                                                                   */
/* ProTR - Prototipador para Sistemas de Tempo Real                  */
/* Nome do Sistema: ENGARRAFAMENTO                                   */
/* Nome do Modulo: ENGARR05.C                                        */
/*                                                                   */
/* Centro Tecnologico para Informatica - CTI                         */
/* Instituto de Automacao - IA                                       */
/*                                                                   */
/* Analista Responsavel: Glaucia Dantas Franco Azevedo               */
/*                                                                   */
/* Data: 09/06/91                                                    */
/* 1991 - Todos os direitos reservados.                              */
/*********************************************************************/


#include "NUCLEO.H"
#include "BOTTLE.H"
#include <math.h>
#include <string.h>
#include <time.h>


/*
 *  NOME DA TRANSFORMACAO DE DADOS: compoe_estado_area
 */

void compoe_estado_area(void)
{
tyestado_reserva estado_reserva ;
tyestado_linha estado_linha ;
tyestado_area estado_area ;
osrcfc(2,&estado_reserva);  /* RECEIVE FLOW CONTINUOUS */
osrcfc(5,&estado_linha);  /* RECEIVE FLOW CONTINUOUS */

estado_area .pH_atual =estado_reserva .pH_atual ;
estado_area .nivel_reserva =estado_reserva .nivel_reserva ;
estado_area .tam_atual_gar =estado_linha .tam_atual_gar ;
estado_area .estado =estado_linha .estado ;
ossenfc (9, &estado_area);
}
```

A.5. Execução do Protótipo.

A execução do protótipo é coordenada pelo Núcleo de Tempo Real a partir da leitura dos eventos externos expressos no arquivo do Cenário. A seguir apresenta-se a Descrição de Eventos no Cenário.

```
sistema ENGARRAFAMENTO : evento;

0 : {
     novo_nivel_pH : 6.5;                  /* inicializa nivel do pH */
    }

1 : {
     eve habilita_area;                          /* habilita area */
    }

2 : {
     pH_atual : 6.4;                        /* valor corrente pH */
     nivel_reserva : 0.85;                /* nivel do reservatorio */
    }

3 : {
     eve liga;                             /* operador liga linha */
     contato_garrafa : 0;                   /* gera evento NAO_OK */
     peso : 0.1;                               /* peso da garrafa */
    }

4 : {
     eve desliga;                        /* operador desliga linha */
                              /* p/ poder inserir tamanho garrafa */
    }

5 : {
     novo_tam_gar : 2;                /* operador insere novo tamanho */
    }

6 : {
     eve liga;                             /* operador religa linha */
    }

7 : {
     contato_garrafa : 1;                        /* libera garrafa */
    }

8 : {
     peso : 2.0;                                  /* garrafa cheia */
    }
```

```
9 : {
     eve garrafa_removida;
     contato_garrafa : 0;
     peso : 0.1;
    }

FIM
```

A.6. Resultado da Execução.

Os resultados da execução são armazenados em um arquivo consistindo da apresentação do comportamento do sistema.

```
RESULTADO DA EXECUCAO DO CENARIO bottle


DATA: 09/11/91   HORA: 09:58:28

000000    Trans verifica_cond_partida aguardando Trigger
000000    Trans preenche_garrafa aguardando habilitacao
000000    Trans reg_estado_reserva aguardando habilitacao
000000    Trans controla_linha aguardando habilitacao
000000    Trans crt_cond_reserva aguardando habilitacao
000000    Trans controla_area habilita Trans altera_pH
000000    Trans reg_estado_linha aguardando habilitacao
000000    Trans cria_rotulo aguardando Trigger
000000    Trans controla_pH aguardando habilitacao
000000    Trans controla_entrada aguardando habilitacao
000000    Trans estado_desliga aguardando Trigger
000000    Trans estado_liga aguardando Trigger
000000    Trans altera_tam_gar aguardando habilitacao

Realizando leitura do conjunto de eventos numero 0

000000    Trans oslecenario gerou fluxo discr novo_nivel_pH id=0
000000 **Trans oslecenario gerou fluxo discr novo_nivel_pH id=0 valor = 6.5
000000    Trans altera_pH recebeu fluxo discr novo_nivel_pH id=0
000000    Trans altera_pH gerou dado para arm pH_referencia id = 1

Realizando leitura do conjunto de eventos numero 1

000002    Trans oslecenario sinalizou evento habilita_area
000002 **Trans oslecenario sinalizou evento habilita_area
000002    Trans controla_area recebeu evento habilita_area
000002    Trans controla_area habilita Trans crt_cond_reserva
000002    Trans controla_area requisita que Trans altera_pH seja desabilitada
000002    Trans crt_cond_reserva habilita Trans reg_estado_reserva

Realizando leitura do conjunto de eventos numero 2

000004    Trans oslecenario gerou fluxo continuo pH_atual id = 1
000004 **Trans oslecenario gerou fluxo continuo pH_atual valor =6.4
000004    Trans monitora_pH recebeu fluxo continuo pH_atual id = 1
000004    Trans reg_estado_reserva recebeu fluxo continuo pH_atual id = 1
000004    Trans oslecenario gerou fluxo continuo nivel_reserva id = 1
```

```
000004 **Trans oslecenario gerou fluxo continuo nivel_reserva valor =0.85
000004   Trans reg_estado_reserva recebeu fluxo continuo nivel_reserva id =1
000004   Trans reg_estado_reserva gerou fluxo continuo estado_reserva id = 1
000004   Trans compoe_estado_area recebeu fluxo continuo estado_reserva id=1
000004   Trans monitora_pH recebeu dado do arm pH_referencia id = 1
000004   Trans monitora_pH sinalizou evento pH_dentro_limite
000004   Trans crt_cond_reserva recebeu evento pH_dentro_limite
000004   Trans crt_cond_reserva habilita Trans controla_entrada
000004   Trans crt_cond_reserva habilita Trans controla_pH
000004   Trans crt_cond_reserva habilita Trans controla_linha
000004   Trans controla_linha habilita Trans reg_estado_linha
000004   Trans controla_entrada recebeu fluxo continuo nivel_reserva id = 1
000004   Trans controla_entrada gerou fluxo continuo crt_valvula_entrada id
         = 1
000004 **Trans controla_entrada gerou fluxo continuo crt_valvula_entrada
         valor =0
000004   Trans controla_pH recebeu fluxo continuo crt_valvula_entrada id = 1
000004   Trans controla_pH recebeu fluxo continuo pH_atual id = 1
000004   Trans controla_pH recebeu dado do arm pH_referencia id = 1
000004   Trans controla_pH gerou fluxo continuo crt_valvula_pH id = 1
000004 **Trans controla_pH gerou fluxo continuo crt_valvula_pH valor =1

Realizando leitura do conjunto de eventos numero 3

000006   Trans oslecenario sinalizou evento liga
000006 **Trans oslecenario sinalizou evento liga
000006   Trans oslecenario gerou fluxo continuo contato_garrafa id = 1
000006 **Trans oslecenario gerou fluxo continuo contato_garrafa valor =0
000006   Trans monitora_contato_gar recebeu fluxo continuo  contato_garrafa
         id = 1
000006   Trans oslecenario gerou fluxo continuo peso id = 1
000006 **Trans oslecenario gerou fluxo continuo peso valor =0.1
000006   Trans monitora_contato_gar sinalizou evento gar_nao_posicao
000006   Trans controla_linha recebeu evento liga
000006   Trans controla_linha enviou Trigger p/ trans estado_liga
000006   Trans controla_linha enviou Trigger p/ trans verifica_cond_partida
000006   Trans  controla_linha  requisita  que  Trans  altera_tam_gar  seja
         desabilitada
000006   Trans verifica_cond_partida recebeu fluxo continuo peso id = 1
000006   Trans verifica_cond_partida recebeu fluxo continuo  contato_garrafa
         id = 1
000006   Trans verifica_cond_partida sinalizou evento ok
000006   Trans verifica_cond_partida aguardando Trigger
000006   Trans controla_linha recebeu evento ok
000006   Trans controla_linha sinalizou evento libera_garrafa
000006 **Trans controla_linha sinalizou evento libera_garrafa
000006   Trans controla_linha recebeu evento gar_nao_posicao
000006   Trans estado_liga gerou dado para arm estado id = 1
000006   Trans estado_liga aguardando Trigger
```

```
Realizando leitura do conjunto de eventos numero 4

000008   Trans oslecenario sinalizou evento desliga
000008 **Trans oslecenario sinalizou evento desliga
000008   Trans controla_linha recebeu evento desliga
000008   Trans controla_linha enviou Trigger p/ trans estado_desliga
000008   Trans controla_linha habilita Trans altera_tam_gar

000008   Trans estado_desliga gerou dado para arm estado id = 2
000008   Trans estado_desliga aguardando Trigger

Realizando leitura do conjunto de eventos numero 5

000010   Trans oslecenario gerou fluxo discr novo_tam_gar id=0
000010 **Trans oslecenario gerou fluxo discr novo_tam_gar id=0 valor=2
000010   Trans altera_tam_gar recebeu fluxo discr novo_tam_gar id=0
000010    Trans monitora_contato_gar recebeu fluxo continuo contato_garrafa
         id = 1
000010   Trans monitora_contato_gar sinalizou evento gar_nao_posicao
000010   Trans controla_linha recebeu evento gar_nao_posicao
000010   Trans controla_pH recebeu fluxo continuo crt_valvula_entrada id = 1
000010   Trans controla_pH recebeu fluxo continuo pH_atual id = 1
000010   Trans controla_pH recebeu dado do arm pH_referencia id = 1
000010   Trans controla_pH gerou fluxo continuo crt_valvula_pH id = 2
000010 **Trans controla_pH gerou fluxo continuo crt_valvula_pH valor =1
000010   Trans monitora_pH recebeu fluxo continuo pH_atual id = 1
000010   Trans monitora_pH recebeu dado do arm pH_referencia id = 1
000010   Trans monitora_pH sinalizou evento pH_dentro_limite
000010   Trans crt_cond_reserva recebeu evento pH_dentro_limite
000010   Trans altera_tam_gar gerou dado para arm tam_atual_gar id = 1
000010   Trans reg_estado_linha recebeu dado do arm tam_atual_gar id = 1
000010   Trans reg_estado_linha recebeu dado do arm estado id = 2
000011   Trans reg_estado_linha gerou fluxo continuo estado_linha id = 1
000011 **Trans reg_estado_linha gerou fluxo continuo estado_linha valor
         =ESTRUTURA
000011   Trans compoe_estado_area recebeu fluxo continuo estado_linha id = 1
000011   Trans compoe_estado_area gerou fluxo continuo estado_area id = 1
000011 **Trans compoe_estado_area gerou fluxo continuo estado_area valor
         =ESTRUTURA

Realizando leitura do conjunto de eventos numero 6

000012   Trans oslecenario sinalizou evento liga
000012 **Trans oslecenario sinalizou evento liga
000012   Trans controla_linha recebeu evento liga
000012   Trans controla_linha enviou Trigger p/ trans estado_liga
000012   Trans controla_linha enviou Trigger p/ trans verifica_cond_partida
000012   Trans controla_linha requisita que Trans altera_tam_gar seja
         desabilitada
```

```
000012   Trans verifica_cond_partida recebeu fluxo continuo peso id = 1
000012   Trans verifica_cond_partida recebeu fluxo continuo  contato_garrafa
         id = 1
000012   Trans verifica_cond_partida sinalizou evento ok
000012   Trans verifica_cond_partida aguardando Trigger
000012   Trans controla_linha recebeu evento ok
000012   Trans controla_linha sinalizou evento libera_garrafa

000012 **Trans controla_linha sinalizou evento libera_garrafa
000012   Trans estado_liga gerou dado para arm estado id = 3
000012   Trans estado_liga aguardando Trigger

Realizando leitura do conjunto de eventos numero 7

000014   Trans oslecenario gerou fluxo continuo contato_garrafa id = 2
000014 **Trans oslecenario gerou fluxo continuo contato_garrafa valor =1
000015   Trans reg_estado_reserva recebeu fluxo continuo pH_atual id = 1
000015   Trans reg_estado_reserva recebeu fluxo continuo nivel_reserva id =1
000015   Trans reg_estado_reserva gerou fluxo continuo estado_reserva id = 2
000015   Trans compoe_estado_area recebeu fluxo continuo estado_reserva id=1
000015   Trans compoe_estado_area recebeu fluxo continuo estado_linha id = 1
000015   Trans compoe_estado_area gerou fluxo continuo estado_area id = 2
000015 **Trans compoe_estado_area gerou  fluxo  continuo  estado_area  valor
         =ESTRUTURA
000015   Trans reg_estado_linha recebeu dado do arm tam_atual_gar id = 1
000015   Trans controla_entrada recebeu fluxo continuo nivel_reserva id = 1
000015   Trans controla_entrada gerou fluxo continuo crt_valvula_entrada  id
         = 2
000015 **Trans controla_entrada  gerou  fluxo  continuo  crt_valvula_entrada
         valor =0
000015   Trans reg_estado_linha recebeu dado do arm estado id = 3
000015   Trans reg_estado_linha gerou fluxo continuo estado_linha id = 2
000015 **Trans reg_estado_linha  gerou  fluxo  continuo  estado_linha  valor
         =ESTRUTURA
000020   Trans monitora_contato_gar recebeu fluxo  continuo  contato_garrafa
         id = 2
000020   Trans monitora_contato_gar sinalizou evento gar_em_posicao
000020   Trans controla_linha recebeu evento gar_em_posicao
000020   Trans controla_linha habilita Trans preenche_garrafa
000020   Trans preenche_garrafa recebeu dado do arm tam_atual_gar id = 1
000020   Trans preenche_garrafa recebeu fluxo continuo peso id = 1
000020   Trans preenche_garrafa gerou fluxo continuo crt_valvula_preenche id
         = 1
000020 **Trans preenche_garrafa gerou  fluxo  continuo  crt_valvula_preenche
         valor =1
000020   AVISO: trans preenche_garrafa estourou seu periodo
000020   Trans preenche_garrafa recebeu dado do arm tam_atual_gar id = 1
000020   Trans preenche_garrafa recebeu fluxo continuo peso id = 1
000020   Trans preenche_garrafa gerou fluxo continuo crt_valvula_preenche id
         = 2
```

```
000020 **Trans preenche_garrafa gerou  fluxo  continuo  crt_valvula_preenche
        valor =1
000020   Trans monitora_pH recebeu fluxo continuo pH_atual id = 1
000020   Trans monitora_pH recebeu dado do arm pH_referencia id = 1
000020   Trans monitora_pH sinalizou evento pH_dentro_limite
000020   Trans crt_cond_reserva recebeu evento pH_dentro_limite
000020   Trans controla_pH recebeu fluxo continuo crt_valvula_entrada id = 2
000020   Trans controla_pH recebeu fluxo continuo pH_atual id = 1
000020   Trans controla_pH recebeu dado do arm pH_referencia id = 1
000020   Trans controla_pH gerou fluxo continuo crt_valvula_pH id = 3
000020 **Trans controla_pH gerou fluxo continuo crt_valvula_pH valor =1

Realizando leitura do conjunto de eventos numero 8

000022   Trans oslecenario gerou fluxo continuo peso id = 2
000022 **Trans oslecenario gerou fluxo continuo peso valor =2
000030   Trans preenche_garrafa recebeu dado do arm tam_atual_gar id = 1
000030   Trans preenche_garrafa recebeu fluxo continuo peso id = 2
000030   Trans monitora_contato_gar recebeu fluxo  continuo  contato_garrafa
        id = 2
000030   Trans monitora_contato_gar sinalizou evento gar_em_posicao
000030   Trans reg_estado_reserva recebeu fluxo continuo pH_atual id = 1
000030   Trans reg_estado_reserva recebeu fluxo continuo nivel_reserva id =1
000030   Trans reg_estado_reserva gerou fluxo continuo estado_reserva id = 3
000030   Trans preenche_garrafa gerou fluxo continuo crt_valvula_preenche id
        = 3
000030 **Trans preenche_garrafa gerou  fluxo  continuo  crt_valvula_preenche
        valor =0
000030   Trans preenche_garrafa sinalizou evento garrafa_cheia
000030   Trans controla_linha recebeu evento gar_em_posicao
000030   Trans controla_linha recebeu evento garrafa_cheia
000030   Trans controla_linha  requisita  que  Trans  preenche_garrafa  seja
        desabilitada
000030   Trans controla_linha enviou Trigger p/ trans cria_rotulo
000030   Trans monitora_pH recebeu fluxo continuo pH_atual id = 1
000030   Trans monitora_pH recebeu dado do arm pH_referencia id = 1
000030   Trans monitora_pH sinalizou evento pH_dentro_limite
000030   Trans crt_cond_reserva recebeu evento pH_dentro_limite
000030   Trans controla_pH recebeu fluxo continuo crt_valvula_entrada id = 2
000030   Trans controla_pH recebeu fluxo continuo pH_atual id = 1
000030   Trans controla_pH recebeu dado do arm pH_referencia id = 1
000030   Trans controla_pH gerou fluxo continuo crt_valvula_pH id = 4
000030 **Trans controla_pH gerou fluxo continuo crt_valvula_pH valor =1
000030   Trans reg_estado_linha recebeu dado do arm tam_atual_gar id = 1
000030   Trans reg_estado_linha recebeu dado do arm estado id = 3
000030   Trans reg_estado_linha gerou fluxo continuo estado_linha id = 3
000030 **Trans reg_estado_linha  gerou  fluxo  continuo  estado_linha  valor
        =ESTRUTURA
000030   Trans controla_entrada recebeu fluxo continuo nivel_reserva id = 1
```

```
000030   Trans controla_entrada gerou fluxo continuo crt_valvula_entrada id
         = 3
000030 **Trans controla_entrada  gerou  fluxo  continuo  crt_valvula_entrada
         valor =0
000030   Trans compoe_estado_area recebeu fluxo continuo estado_reserva id=3
000030   Trans compoe_estado_area recebeu fluxo continuo estado_linha id = 3
000030   Trans compoe_estado_area gerou fluxo continuo estado_area id = 3
000030 **Trans compoe_estado_area gerou  fluxo  continuo  estado_area  valor
         =ESTRUTURA
000031   Trans cria_rotulo recebeu fluxo continuo pH_atual id = 1
000031   Trans cria_rotulo recebeu dado do arm pH_referencia id = 1
000031   Trans cria_rotulo gerou fluxo discr dado_rotulo id=0
000031    **Trans cria_rotulo gerou fluxo discr dado_rotulo id=0 valor =
         ESTRUTURA
000031   Trans cria_rotulo aguardando Trigger

Realizando leitura do conjunto de eventos numero 9

000032   Trans oslecenario sinalizou evento garrafa_removida
000032 **Trans oslecenario sinalizou evento garrafa_removida
000032   Trans oslecenario gerou fluxo continuo contato_garrafa id = 3
000032 **Trans oslecenario gerou fluxo continuo contato_garrafa valor =0
000032   Trans oslecenario gerou fluxo continuo peso id = 3
000032 **Trans oslecenario gerou fluxo continuo peso valor =0.1
000032   Trans controla_linha recebeu evento garrafa_removida
000032   Trans controla_linha enviou Trigger p/ trans verifica_cond_partida
000032   Trans verifica_cond_partida recebeu fluxo continuo peso id = 3
000032   Trans verifica_cond_partida recebeu fluxo continuo  contato_garrafa
         id = 3
000032   Trans verifica_cond_partida sinalizou evento ok
000032   Trans verifica_cond_partida aguardando Trigger
000032   Trans controla_linha recebeu evento ok
000032   Trans controla_linha sinalizou evento libera_garrafa
000032 **Trans controla_linha sinalizou evento libera_garrafa
000040   Trans preenche_garrafa aguardando habilitacao
000040   Trans monitora_contato_gar recebeu fluxo  continuo  contato_garrafa
         id = 3
000040   Trans monitora_contato_gar sinalizou evento gar_nao_posicao
000040   Trans controla_linha recebeu evento gar_nao_posicao
000040   Trans monitora_pH recebeu fluxo continuo pH_atual id = 1
000040   Trans monitora_pH recebeu dado do arm pH_referencia id = 1
000040   Trans monitora_pH sinalizou evento pH_dentro_limite
000040   Trans controla_pH recebeu fluxo continuo crt_valvula_entrada id = 3
000040   Trans controla_pH recebeu fluxo continuo pH_atual id = 1
000040   Trans controla_pH recebeu dado do arm pH_referencia id = 1
000040   Trans controla_pH gerou fluxo continuo crt_valvula_pH id = 5
000040 **Trans controla_pH gerou fluxo continuo crt_valvula_pH valor =1
000040   Trans crt_cond_reserva recebeu evento pH_dentro_limite
000045   Trans reg_estado_reserva recebeu fluxo continuo pH_atual id = 1
000045   Trans reg_estado_reserva recebeu fluxo continuo nivel_reserva id= 1
000045   Trans reg_estado_reserva gerou fluxo continuo estado_reserva id = 4
```

```
000045   Trans controla_entrada recebeu fluxo continuo nivel_reserva id = 1
000045   Trans controla_entrada gerou fluxo continuo crt_valvula_entrada  id
         = 4
000045 **Trans controla_entrada  gerou  fluxo  continuo  crt_valvula_entrada
         valor =0
000045   Trans compoe_estado_area recebeu fluxo continuo estado_reserva id=4
000045   Trans compoe_estado_area recebeu fluxo continuo estado_linha id = 3
000045   Trans compoe_estado_area gerou fluxo continuo estado_area id = 4
000045 **Trans compoe_estado_area gerou  fluxo  continuo  estado_area  valor
         =ESTRUTURA
000045   Trans reg_estado_linha recebeu dado do arm tam_atual_gar id = 1
000045   Trans reg_estado_linha recebeu dado do arm estado id = 3
000045   Trans reg_estado_linha gerou fluxo continuo estado_linha id = 4
000045 **Trans reg_estado_linha  gerou  fluxo  continuo  estado_linha  valor
         =ESTRUTURA
000050   Trans monitora_contato_gar recebeu fluxo  continuo  contato_garrafa
         id = 3
000050   Trans monitora_contato_gar sinalizou evento gar_nao_posicao
000050   Trans controla_linha recebeu evento gar_nao_posicao
000050   Trans monitora_pH recebeu fluxo continuo pH_atual id = 1
000050   Trans monitora_pH recebeu dado do arm pH_referencia id = 1
000050   Trans monitora_pH sinalizou evento pH_dentro_limite
000050   Trans crt_cond_reserva recebeu evento pH_dentro_limite
000050   Trans controla_pH recebeu fluxo continuo crt_valvula_entrada id = 4
000050   Trans controla_pH recebeu fluxo continuo pH_atual id = 1
000050   Trans controla_pH recebeu dado do arm pH_referencia id = 1
000050   Trans controla_pH gerou fluxo continuo crt_valvula_pH id = 6
000050 **Trans controla_pH gerou fluxo continuo crt_valvula_pH valor =1


FINAL EXECUCAO PROTOTIPO
```

- ANEXO B -

ROTINAS DO NÚCLEO ACESSÍVEIS AO USUÁRIO

NOTE_EVENT.

- **Sinopse da Rotina.**

  osnote (nome_evento).

- **Parâmetros de Entrada.**

  nome_evento - nome do evento.

- **Descrição da Rotina.**

  Sinaliza o evento identificado por "nome_evento".
  Caso a tarefa associada ao evento esteja suspensa aguardando-o, ela será ativada e o evento consumido, senão o evento será armazenado.
  Caso a transformação de controle associada ao evento esteja desabilitada, através de "Disable", o evento será ignorado.
  Múltiplas sinalizações de eventos são consideradas, ou seja, eventos são implementados como contadores.

RESET_EVENT_STORE.

- **Sinopse da Rotina.**

  osrevs (nome_arm_evento).

- **Parâmetros de Entrada.**

  nome_arm_evento - nome do armazenador de eventos.

- **Descrição da Rotina.**

  Retorna o armazenador de eventos identificado por "nome_arm_evento" para o seu valor inicial (sem evento sinalizado).

SEND_FLOW.

- **Sinopse da Rotina.**

  ossenfw (nome_fluxo_discreto).

- **Parâmetros de Entrada.**

  nome_fluxo_discreto - nome do fluxo discreto.

- **Descrição da Rotina.**

  Envia dado para um fluxo discreto identificado por "nome_fluxo_discreto". Caso o processo receptor da mensagem esteja desabilitado, através de "Disable"/"Trigger", o dado será simplesmente ignorado.
  Se o processo receptor estiver ativo mas não puder receber no instante do envio, o dado será armazenado até que o processo o requisite.
  Caso o processo seja desabilitado nesse interim, os dados armazenados serão ignorados.
  O tamanho da fila de dados é limitado; caso ocorra "overflow", um erro será gerado e o processamento interrompido.

SEND_FLOW_CONTINUOUS.

- **Sinopse da Rotina.**

  ossenfc (nome_fluxo_contínuo).

- **Parâmetros de Entrada.**

  nome_fluxo_contínuo - nome do fluxo contínuo.

- **Descrição da Rotina.**

  Envia dado para o fluxo contínuo identificado por "nome_fluxo_contínuo". Como se trata de um fluxo contínuo, o dado será simplesmente copiado sobre o dado anterior. Se existirem transformações aguardando o dado, elas serão ativadas.

SIGNAL_EVENT_STORE.

- **Sinopse da Rotina.**

  ossigevs (nome_evento).

- **Parâmetros de Entrada.**

  nome_evento - nome do evento.

- **Descrição da Rotina.**

  Sinaliza o evento identificado por "nome_evento".
  Caso exista alguma tarefa aguardando por esse evento, ele será consumido, senão ele será armazenado.
  Múltiplas sinalizações são consideradas, ou seja, armazenadores de eventos são implementados como contadores.

WRITE_DATA_STORE.

- **Sinopse da Rotina.**

  oswtds (nome_armazenador).

- **Parâmetros de Entrada.**

  nome_armazenador - nome do armazenador de dados.

- **Descrição da Rotina.**

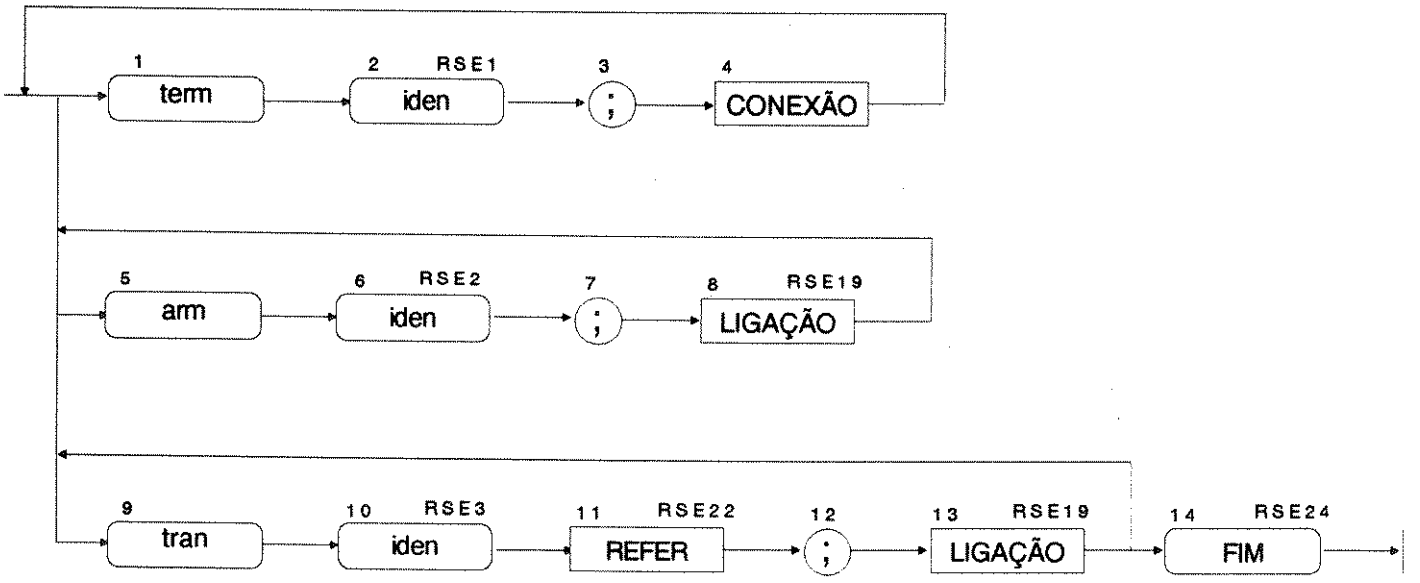
  Atualiza dados presentes no armazenador de dados identificado por "nome_armazenador". Se essa for a primeira escrita e existirem transformações aguardando dados, elas serão ativadas.

- ANEXO C -

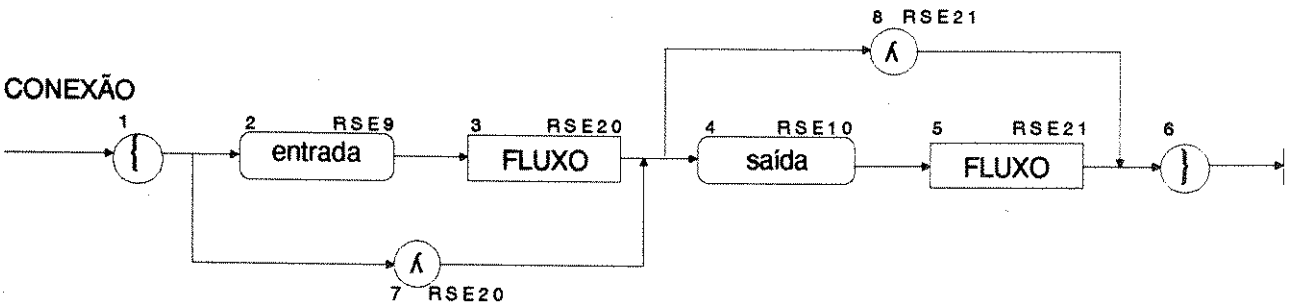
GRAFO SINTÁTICO DO "ESQUEMA DE TRANSFORMAÇÕES"

SISTEMA

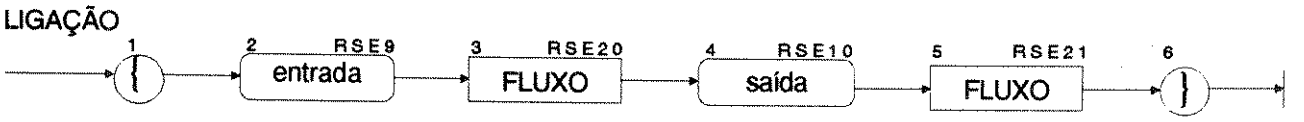
BLOCO



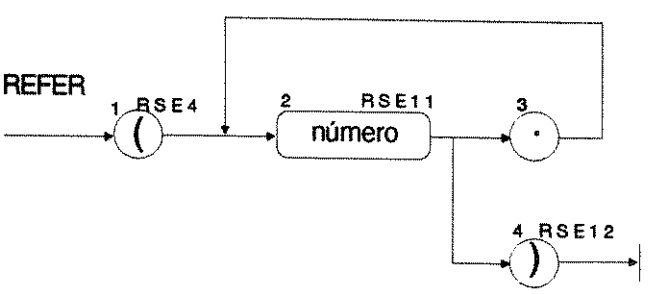
CONEXÃO

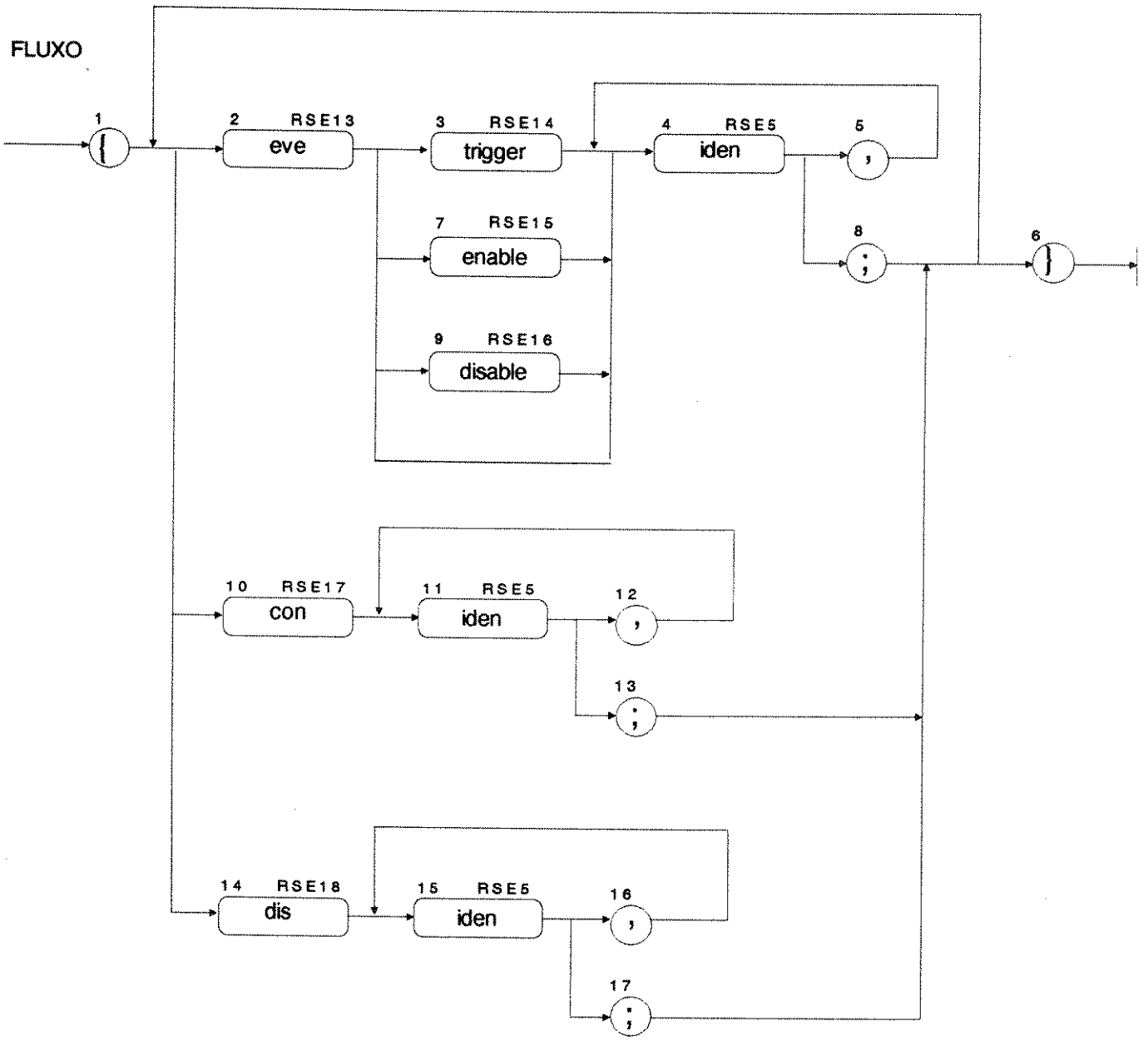


LIGAÇÃO



REFER




FLUXO
1
{
2
RSE13
eve
3
RSE14
trigger
4
RSE5
iden
5
,
7
RSE15
enable
8
;
6
}
9
RSE16
disable
10
RSE17
con
11
RSE5
iden
12
,
13
;
14
RSE18
dis
15
RSE5
iden
16
,
17
;

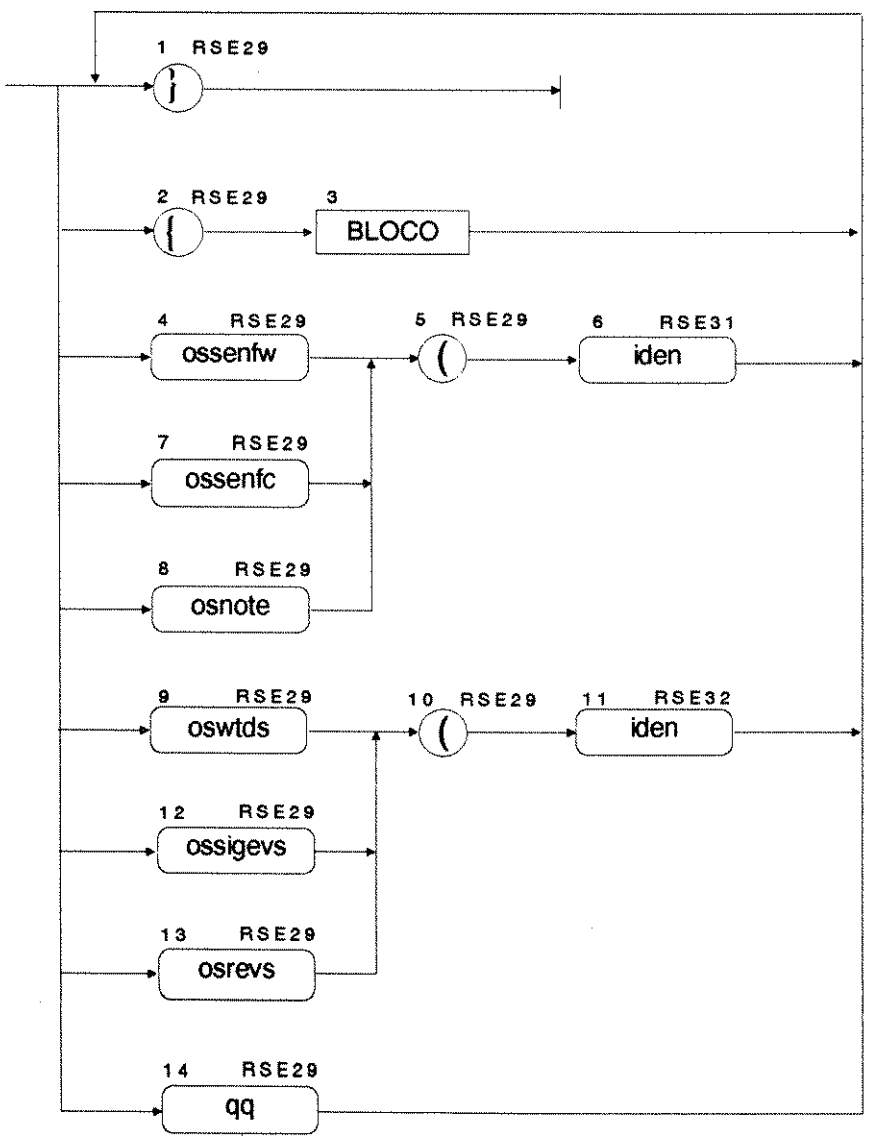
- ANEXO D -

GRAFO SINTÁTICO DO "DICIONÁRIO DE DADOS"

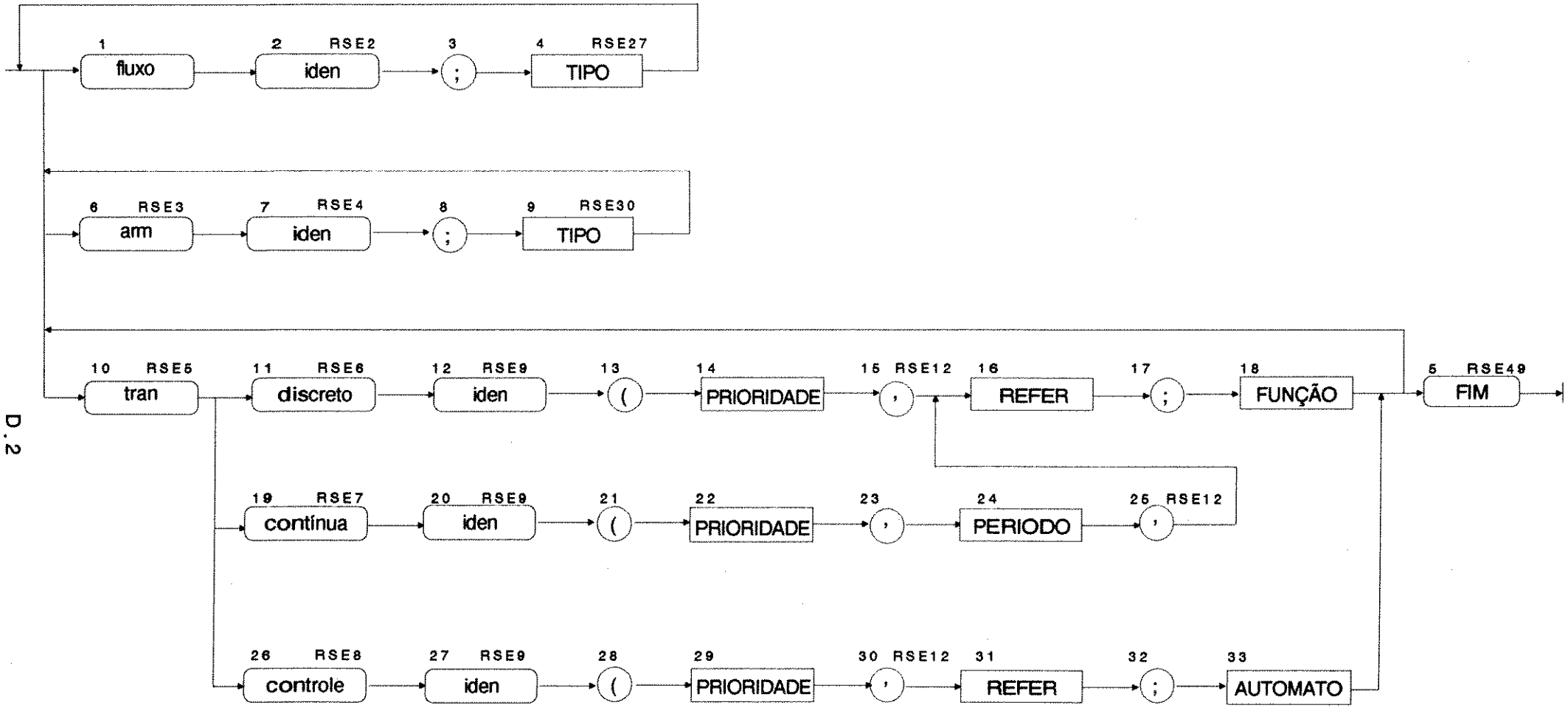
DICIONARIO

FUNÇÃO

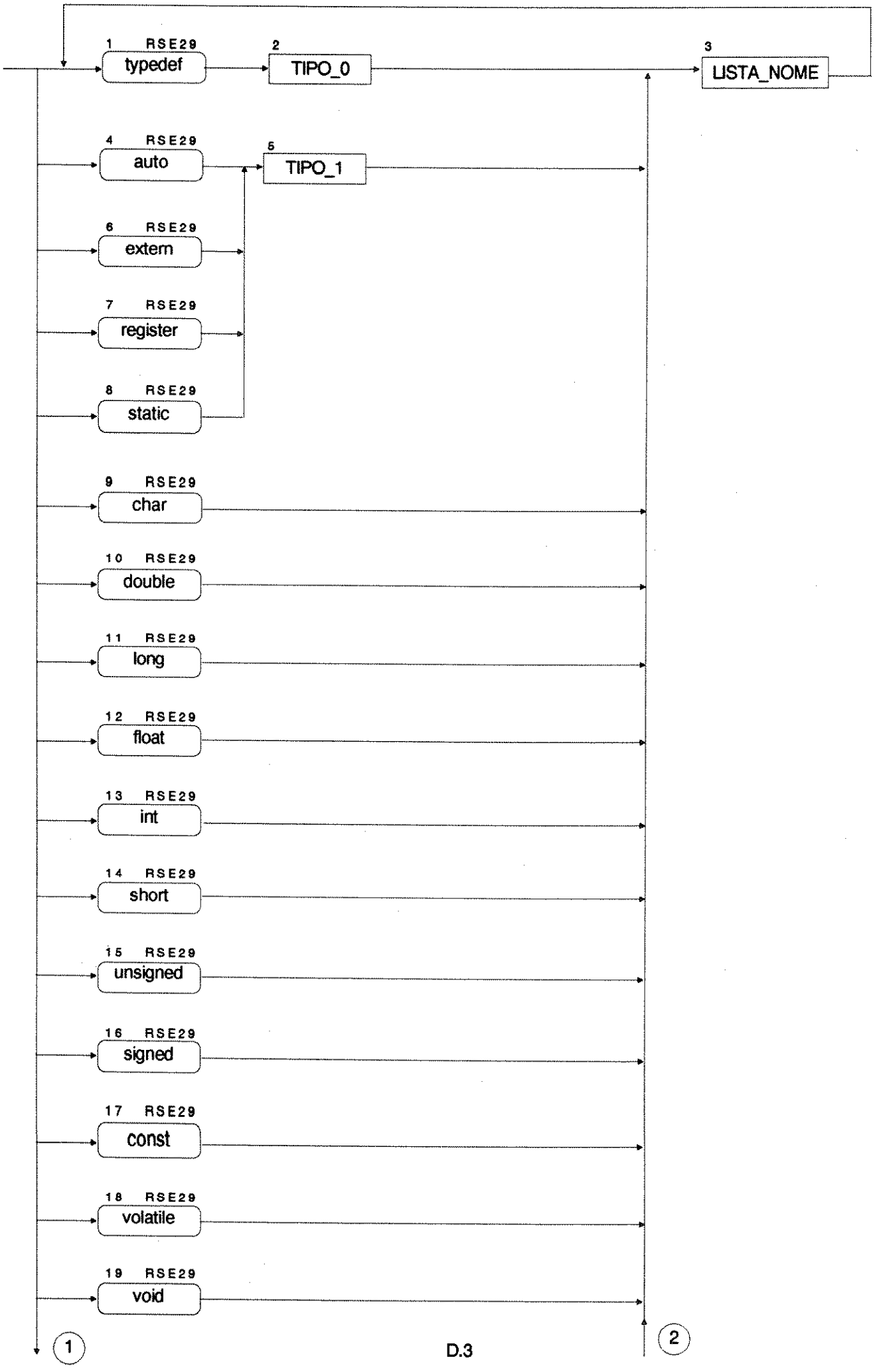
BLOCO

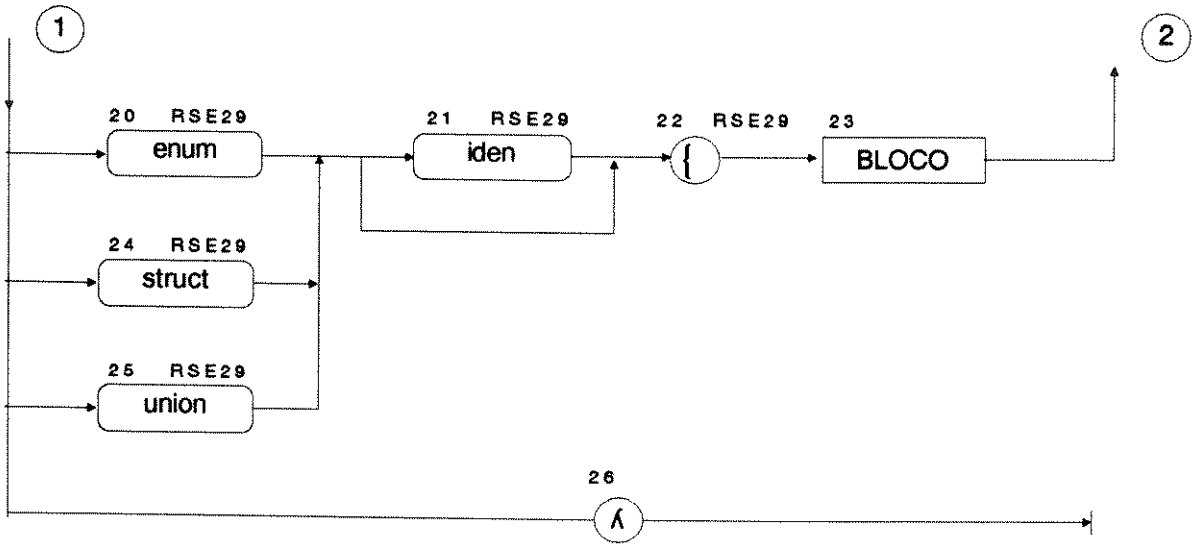


DESC


1
fluxo
2
RSE2
iden
3
;
4
RSE27
TIPO
6
RSE3
arm
7
RSE4
iden
8
;
9
RSE30
TIPO
10
RSE5
tran
11
RSE6
discreto
12
RSE9
iden
13
(
14
PRIORIDADE
15
RSE12
,
16
REFER
17
;
18
FUNÇÃO
5
RSE49
FIM
19
RSE7
contínua
20
RSE9
iden
21
(
22
PRIORIDADE
23
,
24
PERIODO
25
RSE12
,
26
RSE8
controle
27
RSE9
iden
28
(
29
PRIORIDADE
30
RSE12
,
31
REFER
32
;
33
AUTOMATO

DECLARAÇÃO


1 RSE29
typedef
2
TIPO_0
3
LISTA_NOME
4 RSE29
auto
5
TIPO_1
6 RSE29
extern
7 RSE29
register
8 RSE29
static
9 RSE29
char
10 RSE29
double
11 RSE29
long
12 RSE29
float
13 RSE29
int
14 RSE29
short
15 RSE29
unsigned
16 RSE29
signed
17 RSE29
const
18 RSE29
volatile
19 RSE29
void
1
2

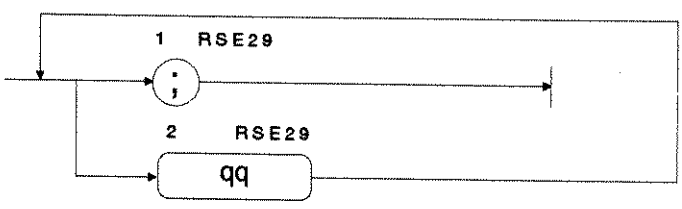


NOTA - O grafo de DECLARAÇÃO não trata declarações complexas (que possuem parenteses "(" de tipos definidos através de typedef).
Ex.: no (*x);
A implementação atual do analisador da base de dados não pode distinguir entre :

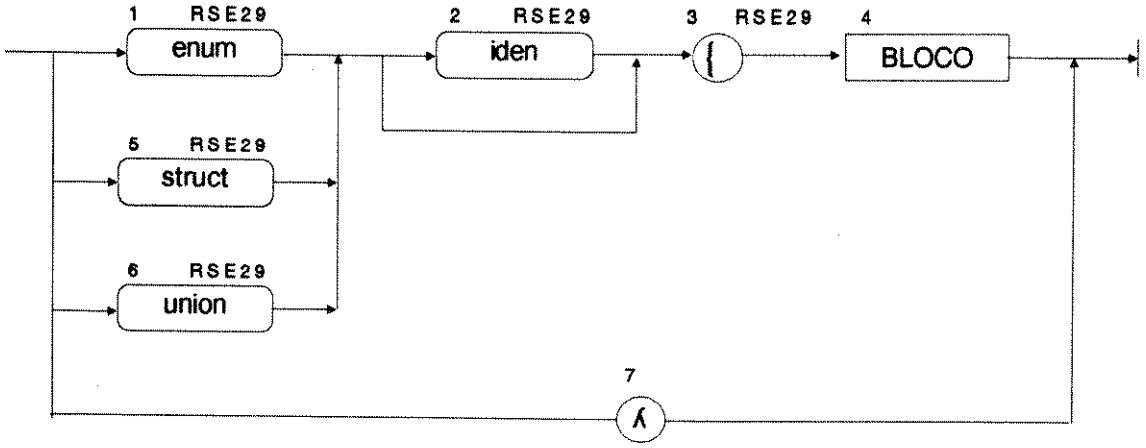
- 1. x ponteiro para typedef nó.
- 2. função "nó" recebendo *x como parâmetro.

Somente uma nova implementação do analisador que englobe toda a linguagem C pode resolver o problema. Nesta versão, as construções utilizando "typedef" devem ser evitadas.

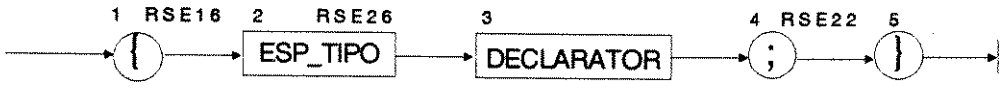
LISTA_NOME



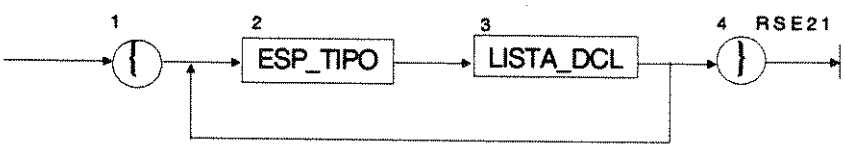
TIPO_1



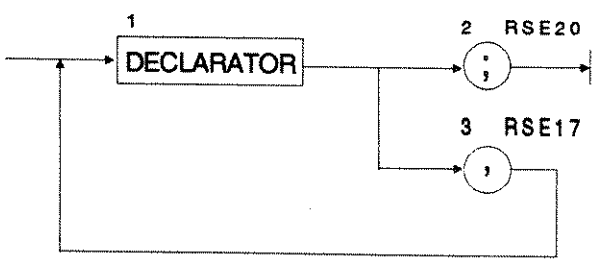
TIPO



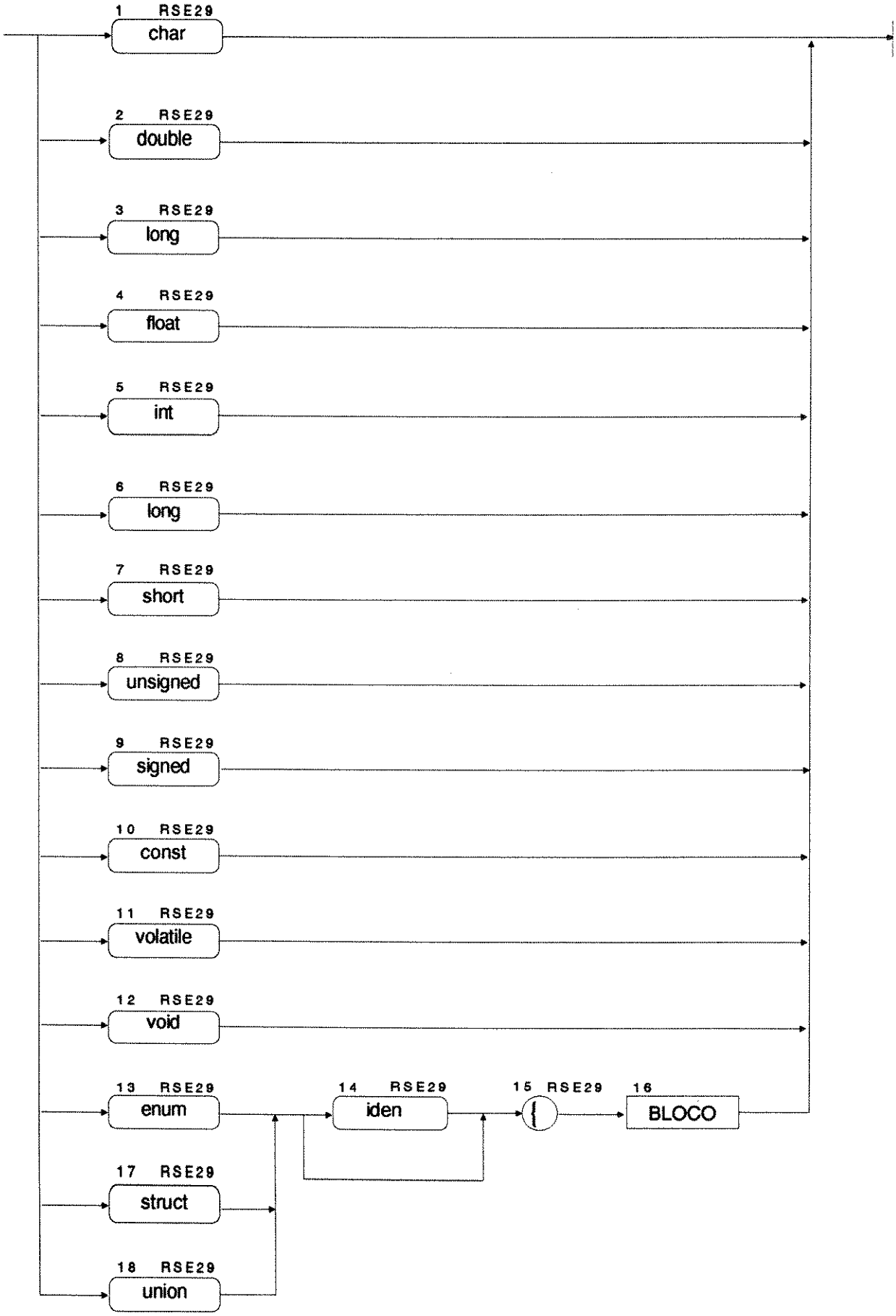
LISTA_MEMBROS



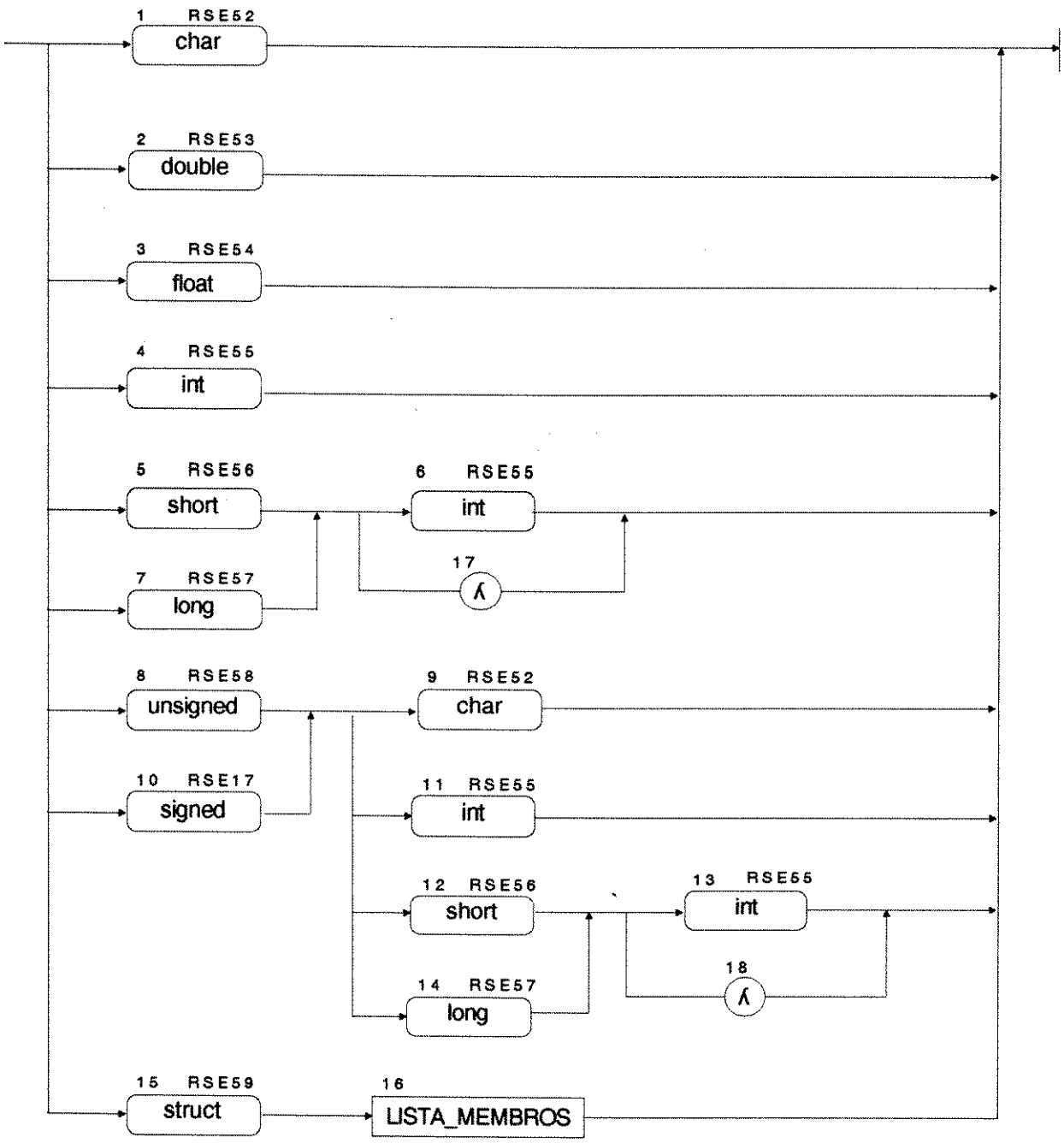
LISTA_DCL



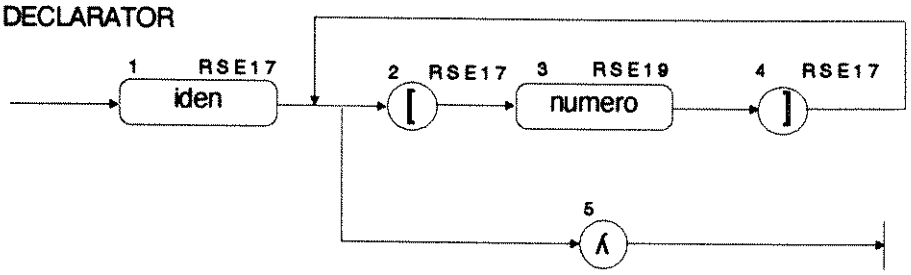
TIPO_0


1 RSE29
char
2 RSE29
double
3 RSE29
long
4 RSE29
float
5 RSE29
int
6 RSE29
long
7 RSE29
short
8 RSE29
unsigned
9 RSE29
signed
10 RSE29
const
11 RSE29
volatile
12 RSE29
void
13 RSE29
enum
14 RSE29
iden
15 RSE29
{
16
BLOCO
17 RSE29
struct
18 RSE29
union

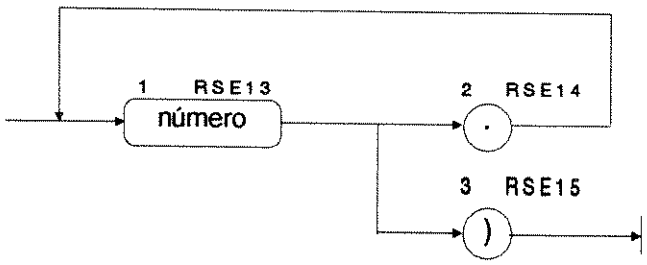
ESP_TIPO


1 RSE52
char
2 RSE53
double
3 RSE54
float
4 RSE55
int
5 RSE56
short
6 RSE55
int
17
λ
7 RSE57
long
8 RSE58
unsigned
9 RSE52
char
10 RSE17
signed
11 RSE55
int
12 RSE56
short
13 RSE55
int
18
λ
14 RSE57
long
15 RSE59
struct
16
LISTA_MEMBROS

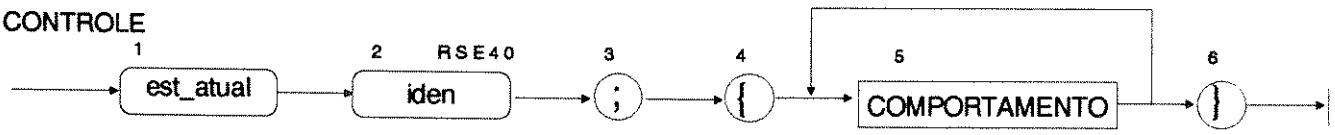
DECLARATOR



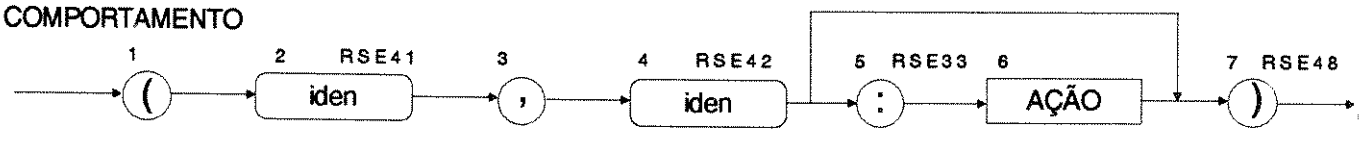
REFER



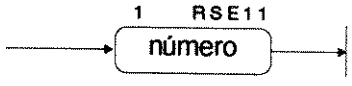
CONTROLE



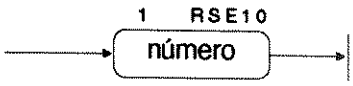
COMPORTAMENTO

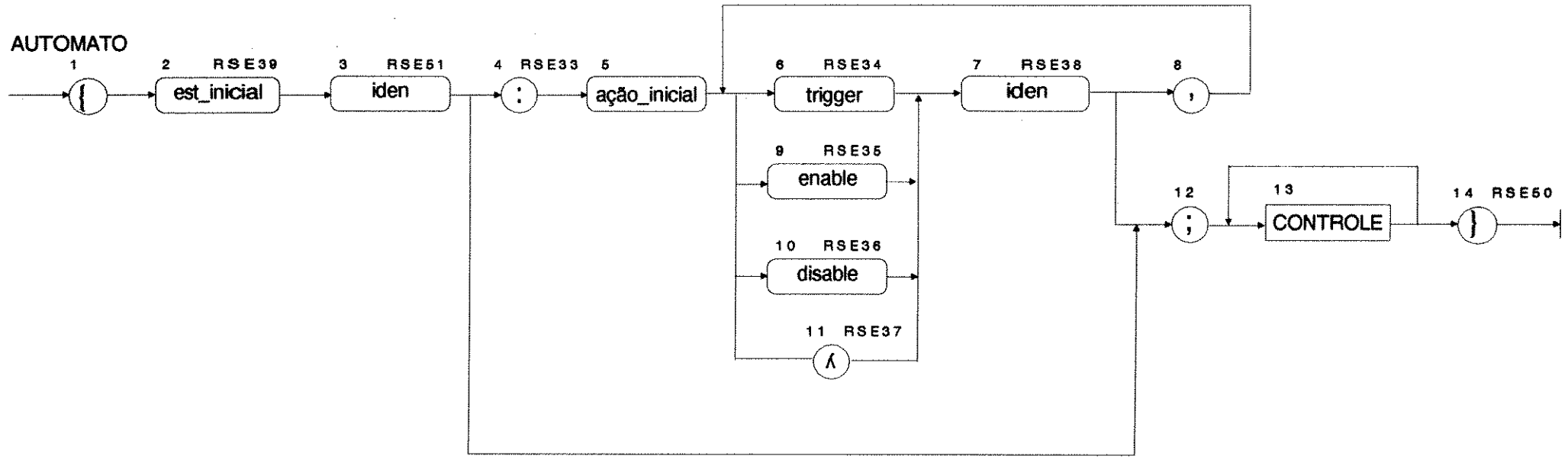


PERIODO

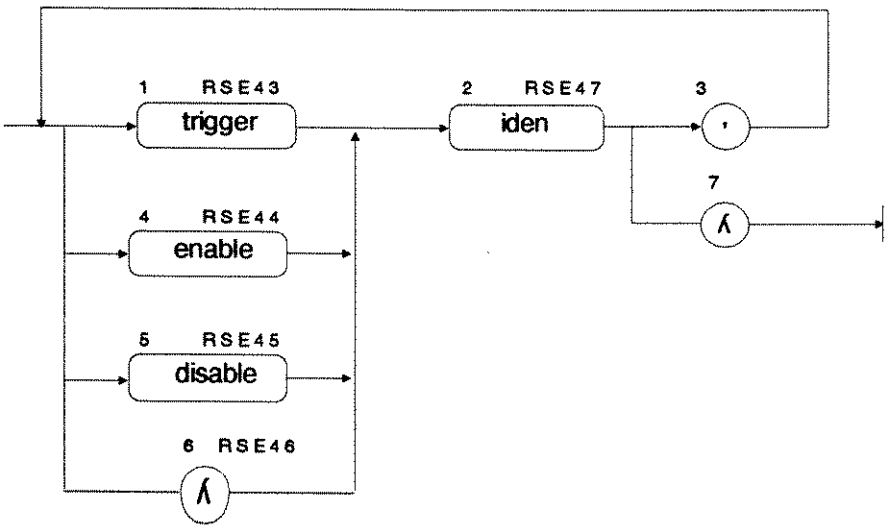


PRIORIDADE




AUTOMATO
1
{
2 RSE39
est_inicial
3 RSE51
iden
4 RSE33
:
5
ação_inicial
6 RSE34
trigger
7 RSE38
iden
8
,
9 RSE35
enable
10 RSE36
disable
11 RSE37
λ
12
;
13
CONTROLE
14 RSE50
}

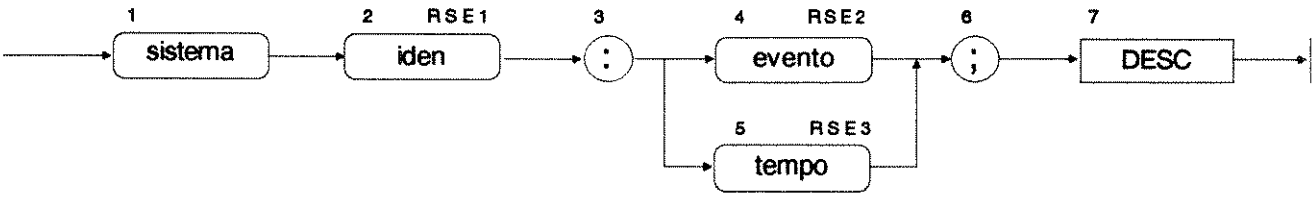
AÇÃO


1 RSE43
trigger
2 RSE47
iden
3
,
7
λ
4 RSE44
enable
5 RSE45
disable
6 RSE46
λ

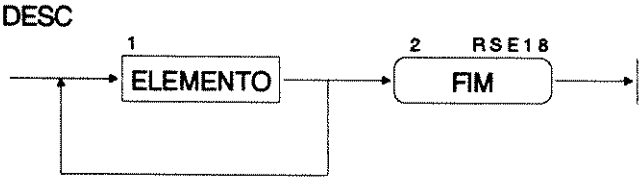
- ANEXO E -

GRAFO SINTÁTICO DO "CENÁRIO"

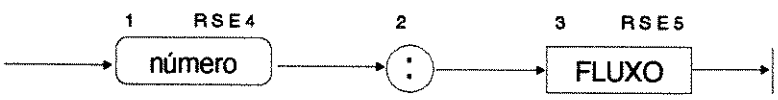
CENARIO



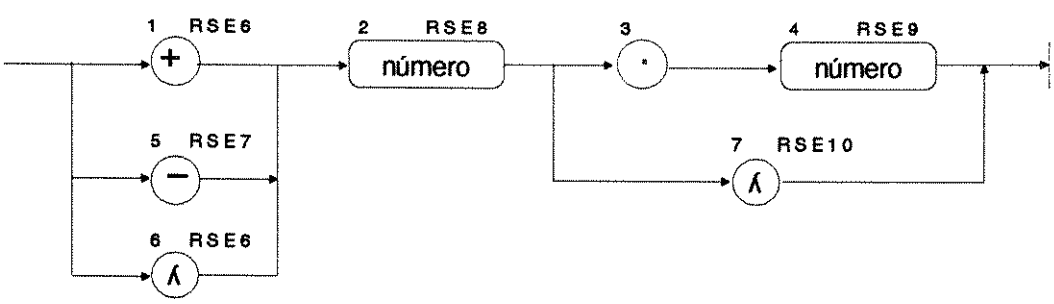
DESC

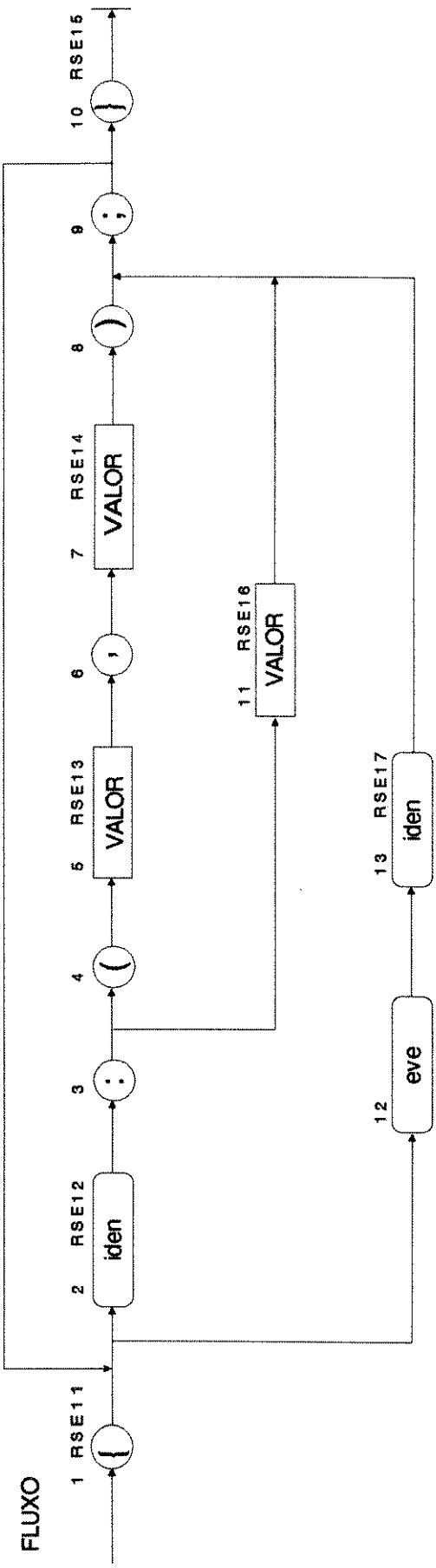


ELEMENTO



VALOR




FLUXO
1 RSE11
{
2 RSE12
iden
3
;
4
(
5 RSE13
VALOR
6
,
7 RSE14
VALOR
8
)
9
;
10 RSE15
}
11 RSE16
VALOR
12
eve
13 RSE17
iden

E . 2