UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

UMA METODOLOGIA PARA O DESENVOLVIMENTO DE APLICAÇÕES DISTRIBUÍDAS BASEADA NA TÉCNICA DE DESCRIÇÃO FORMAL ESTELLE

Dissertação submetida à Universidade Federal de Santa Catarina para a obtenção do grau de

MESTRE EM ENGENHARIA ELÉTRICA

ELIESER BOTELHO MANHAS JÚNIOR

FLORIANÓPOLIS, JULHO DE 1994

UMA METODOLOGIA PARA O DESENVOLVIMENTO DE APLICAÇÕES DISTRIBUÍDAS BASEADA NA TÉCNICA DE DESCRIÇÃO FORMAL ESTELLE

Elieser Botelho Manhas Júnior

Esta dissertação foi julgada adequada para a obtenção do título de
Mestre em Engenharia
especialidade **Engenharia Elétrica,**
área de concentração **Controle, Automação e Informática Industrial,**
e aprovada em sua forma final pelo Curso de Pós-Graduação

Prof. Vitório Bruno Mazzola, Dr.
Orientador

Prof. Roberto de Souza Salgado, Ph.D.
Coordenador do Curso de Pós-Graduação em Engenharia Elétrica

BANCA EXAMINADORA :

Prof. Vitório Bruno Mazzola, Dr. (Presidente)
Orientador

Prof. Bernardo Gonçalves Riso, Dr.

Prof. Jean-Marie Farines, Dr. Ing.

Prof. Jom da Silva Fraga, Dr.

À minha avó Ivone
À minha sobrinha Isabela

AGRADECIMENTOS

Ao Prof. Vitório Bruno Mazzola, pelo empenho na orientação deste trabalho e pela amizade demonstrada durante o nosso convívio.

Aos membros da banca examinadora, pela aceitação de participação e pelas críticas e sugestões que contribuíram para o enriquecimento do trabalho.

Aos professores e colegas do LCMI, em especial ao Roberto Ziller, Leonardo Kammer e Paulo Valim, pela paciência que tiveram comigo na solução de inúmeros problemas.

Aos companheiros de repúblicas: Alex; Aldebaro e Eduardo; Alexandre, Idmilson e Luís Otávio que conseguiram me agüentar durante todo este tempo.

Ao grande amigo Márcio Heidi Suguieda, pela demonstração de um companheirismo raro e invejável.

À Galera de Base e a todos aqueles com quem pude compartilhar muitos momentos de alegria e descontração.

À UFSC e à CAPES pelo suporte material e financeiro.

À Soraya, por todo o carinho demonstrado e que, mesmo nos momentos mais difíceis, sempre soube me incentivar e jamais me deixou desistir.

À minha família e aos meus amigos, pela confiança que sempre me foi dada, a qual foi imprescindível para que eu pudesse chegar até aqui.

Um agradecimento especial à minha mãe, pela dedicação e por tudo o que tem me ensinado, e a quem sou eternamente grato.

SUMÁRIO

SIGLAS

CCS	*Calculus of Communicating Systems*
CFM	Célula Flexível de Montagem
CSP	*Communicating Sequential Processes*
EC	*Estelle to C Compiler*
EDB	*Estelle simulator/DeBugger*
EDS	*Estelle Development System*
EDT	*Estelle Development Toolset*
ESTELLE	*Extended State Transition Language*
ESTIM	*Estelle SimulaTor based on an Interpretative Machine*
EWS	*Estelle WorkStation*
FI	Forma Intermediária
FIFO	*first-in-first-out*
ISO	*International Standardization Organization*
IUT	*Implementation Under Test*
LAAS-CNRS	*Laboratoire d'Automatique et d'Analyse des Systèmes du Centre National de la Recherche Scientifique* (França)
LCMI	Laboratório de Controle e MicroInformática
LOTOS	*Language Of Temporal Ordering Specification*
ML	*Meta-Language*
OSI	*Open System Interconnection*
PC	*Personal Computer*
PDU	*Protocol Data Unit*
PIPN	*Prolog Interpreted Petri Nets*
SDL	*Specification and Description Language*
SEDOS	*Software Environment for the Design of Open distributed Systems*
TCP	*Transmission Control Protocol*

TDF	Técnica de Descrição Formal
UDP	*User Datagram Protocol*
UFSC	Universidade Federal de Santa Catarina

RESUMO

Este trabalho apresenta uma metodologia de desenvolvimento de aplicações distribuídas. Tal metodologia é baseada na técnica de descrição formal Estelle, padronizada pela ISO (*International Standardization Organization*).

Primeiramente é introduzida e justificada a utilização de técnicas de descrição formal, após o que são apresentados os principais conceitos de Estelle e de uma versão, chamada Estelle*. Em seguida é discutida a importância da utilização de ferramentas automatizadas baseadas nas técnicas de descrição formal. Neste sentido, são apresentadas três ferramentas de suporte à Estelle, disponíveis no Laboratório de Controle e Microinformática (LCMI) da UFSC.

A metodologia, fundada sobre os conceitos de abstração e refinamentos sucessivos, é então introduzida, apresentando como as características de Estelle devem ser utilizadas nas diferentes etapas do desenvolvimento de uma aplicação distribuída.

Finalmente, a aplicação da metodologia é ilustrada utilizando um exemplo típico de sistema distribuído: o sistema de controle de uma célula flexível de montagem.

ABSTRACT

This work presents a methodology for developing distributed applications, which is based on the use of the Estelle Formal Description Technique.

Firstly, the use of Formal Description Techniques for developing distributed applications is discussed. Following, the main features of Estelle and Estelle (an extended version of Estelle) are introduced and a discussion about the use of software tools based on Estelle is carried out. In order to support this discussion, several Estelle-based software tools have been studied and introduced in the sequel.*

The methodology, which is based on abstraction and step-wise refinement concepts is then proposed, describing how the Estelle mechanisms can be applied for each step of application development.

Finally, the use of the proposed methodology is ilustrated by the development of a typical case of distributed application: the driving system of a flexible assembly cell.

CAPÍTULO 1

INTRODUÇÃO GERAL

Os sistemas distribuídos vêm assumindo, a cada dia, um papel de destaque nos desenvolvimentos em informática. Esta evolução se deve, de um lado, aos grandes avanços tecnológicos e, de outro lado, à crescente demanda por parte dos usuários destes sistemas. À medida que os sistemas evoluem, novas necessidades vão surgindo do ponto de vista das aplicações que podem ser desenvolvidas em torno destes sistemas.

Dentre os avanços tecnológicos ocorridos nos últimos anos, não se pode deixar de destacar os desenvolvimentos crescentes no domínio da microeletrônica, bem como aqueles da tecnologia de comunicação e interconexão, que permitiram o surgimento das redes de comunicação de dados interligando um grande número de equipamentos informatizados heterogêneos [Le Lann 81].

As motivações para a utilização de sistemas distribuídos são muitas, entre elas o seu maior desempenho, o aumento da confiabilidade e a melhora ao acesso aos dados em uma área geograficamente dispersa [Singhal 91]. Existem na literatura diversas definições de sistema distribuído, algumas mais amplas como a de [Liebowitz 78], outras mais restritas como a de [Enslow 78]. [Le Lann 81], por sua vez, estabelece algumas características lógicas e físicas presentes em um sistema distribuído:

- número arbitrário de processos usuários e de sistema;
- arquitetura modular consistindo de um número possivelmente variável de elementos de processamento;
- comunicação através de passagem de mensagens, sobre uma estrutura de comunicação compartilhada (excluindo a memória compartilhada);
- controle global do sistema, de modo a fornecer uma cooperação dinâmica entre os processos e o gerenciamento em tempo de execução;

- atrasos variáveis de transmissão de mensagens entre processos. Existe sempre um tempo não nulo entre a produção de um evento por um processo e a materialização dessa produção no processo destino (diferente da observação do evento pelo processo destino).

Devido a estas características, a concepção de aplicações distribuídas torna-se uma tarefa complexa já que diversos fatores, irrelevantes para as aplicações tradicionais, são de grande importância para as aplicações distribuídas. Não somente o comportamento individual de cada componente do sistema deve ser considerado, mas também o comportamento global do sistema e as suas interações. Como conseqüência, aspectos como o assincronismo, o paralelismo e o não-determinismo entre os componentes do sistema devem ser observados. Além dessas, outras considerações podem ser importantes dependendo da aplicação envolvida, como a questão da confiabilidade dos dados transmitidos, ou as restrições temporais em sistemas de tempo real.

1.1 - Objetivo do Trabalho

Os aspectos levantados anteriormente levam a crer que o desenvolvimento de uma aplicação distribuída deve ser fundado sobre um conjunto de regras ou uma metodologia que seja coerente e, quando possível, suportada por um conjunto de ferramentas de *software* para o apoio à concepção.

O objetivo do presente trabalho consiste, então, em apresentar uma metodologia de concepção de aplicações distribuídas, que permita uma exploração racional das vantagens oferecidas por tal sistema, através da consideração dos diversos aspectos característicos desta classe de aplicações. A metodologia a ser apresentada está fundada no trabalho desenvolvido em [Mazzola 91] e é baseada na utilização da técnica de descrição formal Estelle, técnica esta definida pela ISO (*International Standardization Organization*), para a concepção de protocolos de comunicação do modelo de referência OSI (*Open System Interconnection*) [Zimermmann 80].

Estelle foi escolhida pelo fato de oferecer um conjunto de facilidades de especificação orientadas aos protocolos de comunicação, mas que permitem levar em conta os aspectos e restrições típicos das aplicações distribuídas, de um ponto de vista mais global. Trabalhos anteriores puderam demonstrar a adequação de técnicas de descrição formal ao desenvolvimento de outras aplicações além dos protocolos de comunicação, citando-se por exemplo o trabalho realizado em [Mazzola 91] para sistemas de manufatura. Além disso, a disponibilidade de algumas ferramentas de apoio à Estelle contribuíram para esta escolha, uma vez que a utilização

de tais ferramentas é de fundamental importância no processo de concepção de uma aplicação distribuída.

1.2 - Organização do Trabalho

No capítulo 2 são introduzidos alguns aspectos referentes ao desenvolvimento de aplicações distribuídas, como o modelo do ciclo de vida e a questão da validação. Em seguida são introduzidas as técnicas de descrição formal e as justificativas para a utilização das mesmas. Estelle é então apresentada, destacando seus conceitos básicos, mecanismos de comunicação, a estrutura hierárquica e de ligações, o paralelismo e a noção de tempo. Finalmente, uma versão de Estelle chamada Estelle* é também apresentada.

O capítulo 3 é referente às ferramentas automatizadas de suporte à Estelle. Três ferramentas disponíveis no LCMI (Laboratório de Controle e Microinformática) da UFSC são detalhadas, procurando situá-las no processo de desenvolvimento de aplicações distribuídas e abordando também as facilidades que cada uma oferece ao usuário.

No capítulo 4 é apresentada a metodologia de desenvolvimento de aplicações distribuídas. Primeiramente, são introduzidos os conceitos de abstração e refinamentos sucessivos que formam a base da metodologia. A metodologia, que consiste em quatro níveis principais de especificação, é então introduzida, ressaltando-se principalmente os mecanismos de Estelle e os aspectos de validação relevantes para cada nível.

No capítulo 5, um exemplo de aplicação da metodologia proposta é apresentado. A aplicação consiste na especificação da tarefa de montagem de uma Célula Flexível de Montagem. Tal especificação é desenvolvida em níveis, de acordo com o que propõe a metodologia, sendo finalmente, implementada na rede de estações de trabalho do LCMI.

CAPÍTULO 2

A CONCEPÇÃO DE APLICAÇÕES DISTRIBUÍDAS

2.1 - Introdução

No desenvolvimento de aplicações distribuídas devem ser levadas em consideração algumas características que normalmente não são relevantes no desenvolvimento de uma aplicação tradicional, como questões de paralelismo, comunicação e outras. A dificuldade de expressar tais características de uma maneira informal, como a linguagem natural, motivou o desenvolvimento de Técnicas de Descrição Formal (TDF). Estas técnicas de descrição formal foram, então, desenvolvidas com o objetivo de permitir a produção de especificações completas, sem ambigüidades, claras e concisas [Vissers 88].

Neste capítulo serão vistas as principais etapas do desenvolvimento de aplicações distribuídas, apresentando um modelo do ciclo de vida do *software* e algumas técnicas de validação. A importância da utilização de técnicas de descrição formal será justificada, apresentando em seguida a técnica de descrição formal Estelle, mostrando seus aspectos sintáticos e semânticos; uma atenção também é dada a uma versão de Estelle , denominada Estelle*, que permite um maior grau de abstração em relação a algumas características de Estelle.

2.2 - As Etapas do Desenvolvimento de Aplicações Distribuídas

2.2.1 - O Modelo do Ciclo de Vida

O modelo do ciclo de vida do *software* representa as atividades que comportam o desenvolvimento e a evolução do *software*. Existem diversos modelos de ciclo de vida em uso, além de outros descritos na literatura. Cada modelo procura definir uma estrutura para a

descrição das etapas de desenvolvimento do sistema e dos produtos fornecidos por cada uma dessas etapas. As etapas e os nomes específicos variam de um modelo para outro mas, genericamente, as etapas a seguir são adequadas tanto para aplicações tradicionais como para aplicações distribuídas [Bochmann 90]:

- ***análise de requisitos***;
- ***projeto***;
- ***implementação***;
- ***manutenção.***

A figura 2.1 a seguir mostra as etapas concernentes ao desenvolvimento do *software*

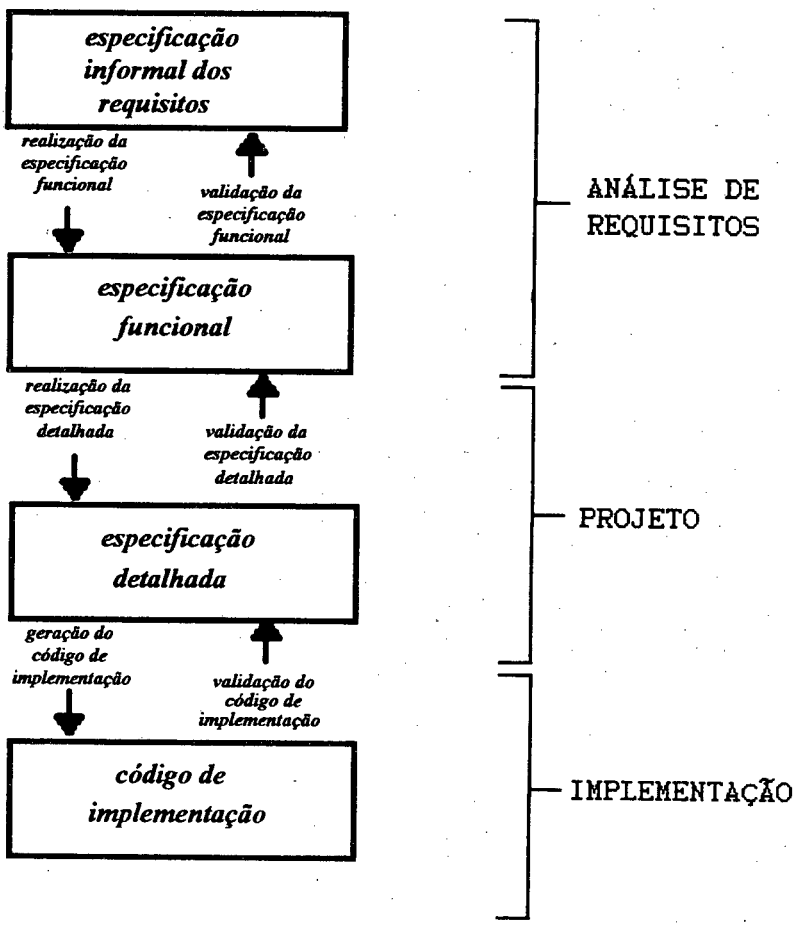


Figura 2.1 - As Etapas de Desenvolvimento do *Software*

Na etapa de **análise de requisitos** é realizada a definição do problema em função do pedido do usuário, tendo como resultado a *especificação informal dos requisitos* do sistema. A

especificação informal dos requisitos é apresentada sob a forma textual e normalmente apresenta diagramas para facilitar a compreensão do problema. Compreendidos os detalhes do problema, desenvolve-se a *especificação funcional* do sistema que deve apresentar um esboço da estrutura do sistema.

A etapa de **projeto** consiste em refinamentos sucessivos das especificações até a obtenção da implementação. Existe normalmente pelo menos uma especificação intermediária (mais detalhada) entre a especificação funcional e a implementação, chamada *especificação detalhada*. Cada uma dessas especificações mais detalhadas deve ser validada com relação à especificação mais abstrata, como será visto na seção seguinte.

A etapa de **implementação** consiste na geração do código executável, o qual deve refletir a estrutura do projeto e realizar as funções especificadas para o sistema. A validação do código de implementação é normalmente realizada através de testes.

2.2.2 - Aspectos da Validação de Aplicações Distribuídas

Neste trabalho o termo ***"validação"*** é utilizado com a mesma conotação de [Bochmann 90], ou seja, qualquer atividade realizada com o intuito de se obter uma especificação ou implementação que satisfaçam os seus requisitos mais abstratos e não contenham erros internos ou inconsistências.

Como foi citado na seção anterior, a validação não constitui uma etapa única e isolada, mas um passo realizado ao final de cada processo de refinamento, como pode ser visto na figura 2.1. Esses processos de validação são de vital importância visto que erros detectados nas primeiras etapas do desenvolvimento do *software* são bem menos custosos em termos de correção. Ainda, no caso da detecção de algum erro, algumas modificações podem ser realizadas na especificação mais abstrata, já que esta serve como uma referência para a validação da especificação mais detalhada.

As atividades de validação podem ser classificadas de acordo com o tipo de análise que é efetuada sobre uma dada especificação [Bochmann 90]:

- ***análise estática***: é a análise baseada no texto da especificação. Este tipo de análise permite a detecção de erros de sintaxe, regras de escopo, conformidade de tipos e outras condições semânticas. A análise estática é similar ao processo de compilação para as linguagens de programação.

- ***análise dinâmica***: neste tipo de análise é considerado o comportamento dinâmico do sistema especificado em um determinado ambiente de execução. Apesar da análise dinâmica ser normalmente mais complexa que a análise estática, ela é capaz de detectar certos erros que não podem ser detectados através da análise estática.

As técnicas de análise dinâmica apresentam características tipicamente complementares entre si. As principais técnicas utilizadas são a verificação, a simulação e o teste de implementação do protótipo. Recentemente foi proposta uma técnica intermediária entre a simulação e a implementação do protótipo, chamada experimentação [Jard 89, Jézéquel 91]. A seguir, serão analisadas as características de cada uma das técnicas acima.

2.2.2.1 - Verificação

O objetivo da verificação é provar formalmente que uma especificação ou implementação satisfaz certas propriedades desejadas, utilizando métodos rigorosos e matematicamente justificados. Dois diferentes tipos de verificação podem ser usados [Ernberg 91]:

- prova de propriedades individuais de uma especificação. Neste caso as propriedades são formalmente formuladas, utilizando por exemplo uma lógica modal, e depois verifica-se se o sistema satisfaz a essas propriedades. Exemplos de propriedades que podem ser verificadas desta forma são: ausência de bloqueio (*deadlock*), definição do comportamento da especificação em todas as situações e outras;

- comparação de duas especificações diferentes para provar que elas são "equivalentes" ou que uma é a implementação correta da outra.

2.2.2.2 - Simulação

A simulação é realizada em um ambiente simulado (na maioria dos casos centralizado) e consiste na observação de um modelo executável do sistema. Ao contrário da verificação, a simulação não cobre todas as possibilidades de execução do sistema. Todavia, ela pode ser aplicada a sistemas bastante complexos e, quando bem utilizada, permite a detecção de erros eficazmente. A sua principal dificuldade reside na necessidade de descrever formalmente e simular o ambiente de execução. Esse ambiente é geralmente muito simplificado pois não é

realista (nem de interesse) levar em consideração todos os parâmetros de um sistema real como, por exemplo, a influência exata do tamanho da mensagem nos atrasos de transmissão [Jard 89].

2.2.2.3 - Teste de Implementação do Protótipo

Esta técnica baseia-se na observação de um protótipo (ou do próprio sistema final) em um ambiente de execução real e a sua avaliação em relação à especificação. Um ponto importante nesta técnica é a seleção de casos de testes. Esses casos de testes devem ser apropriados de maneira a tentar cobrir os principais aspectos do comportamento do sistema, tendo como resultado um conjunto de testes, algumas vezes chamado de seqüência de testes. Para o caso em que a implementação é testada para verificar a sua conformidade com a especificação, esses testes são usualmente determinados com base na especificação. Esse tipo de teste é conhecido como teste de conformidade e é freqüentemente utilizado no caso de protocolos de comunicação.

Enquanto na área dos protocolos de comunicação OSI, seqüências de testes padrões são desenvolvidas para testar a conformidade das implementações com os padrões de protocolos, a seleção de casos de testes é ainda um item importante, visto que alguns requisitos adicionais como o desempenho e a robustez não são cobertos pelos testes de conformidade [Bochmann 90].

Como no caso da simulação, o teste de implementação do protótipo não abrange todos os comportamentos possíveis do sistema e, dessa forma, sua função principal é a detecção de erros eventuais. Uma outra dificuldade é a falta de ferramentas que permitem observar o comportamento de um sistema distribuído como um todo.

O estudo de técnicas de geração de seqüências de teste a partir de especificações formais é assunto de um trabalho realizado no LCMI para futura integração em um ambiente de auxílio ao teste [Silva 94].

2.2.2.4 - Experimentação

A experimentação é uma forma de simulação distribuída na qual o comportamento do modelo em questão é observado em um ambiente distribuído. Dessa maneira, o ambiente de execução não precisa ser modelado (como no caso anterior), sendo pois fornecido pela própria máquina. A experimentação em máquinas paralelas permite observar aspectos dependentes da

máquina alvo, além de fornecer indicações de desempenho em relação a critérios de eficácia ou de qualidade. Além disso, o poder de cálculo desse tipo de máquina torna possível a validação de algoritmos distribuídos constituídos de milhares de processos. A desvantagem principal da experimentação reside na dificuldade de efetuar medições e observações distribuídas, as quais necessitam de técnicas especiais [Jard 89, Jézéquel 91].

2.3 - As Técnicas de Descrição Formal

2.3.1 - O Porquê das Técnicas de Descrição Formal

As técnicas informais de descrição, como as descrições narrativas em linguagem natural, têm-se mostrado ineficientes como única forma de especificação de sistemas computacionais em geral. Quanto maior a complexidade do sistema, como no caso dos sistemas distribuídos, maior ainda é a ineficiência de tais técnicas. Essa ineficiência ocorre, principalmente, devido à existência de ambigüidades na especificação as quais podem dar origem a diferentes interpretações e, conseqüentemente, a erros de implementação [Sidhu 89].

As técnicas de descrição formal apresentam ainda outras características que enfatizam a importância da sua utilização na concepção de qualquer aplicação distribuída [Azema 87, Sidhu 89]:

- possibilitam a análise das especificações e, conseqüentemente, a detecção de erros desde a fase de definição das necessidades;
- servem como uma referência a todas as pessoas ou equipes envolvidas na concepção global da aplicação;
- servem de base comum para a utilização de ferramentas automatizadas de validação, geração de código e geração de seqüências de testes.

2.3.2 - Os Modelos de Base

As técnicas de descrição formal (TDF) desenvolvidas baseiam-se em classes diferentes de modelos. Em [Saqui 90a] é apresentada a seguinte classificação:

2.3.2.1 - Sistemas de Transições

Um sistema de transição é caracterizado por um conjunto (usualmente infinito) de possíveis estados Q, sendo que, em um dado instante qualquer, o sistema se encontra em um estado particular $q \in Q$. O sistema pode efetuar uma transição (considerada instantânea e atômica); para tal escreve-se $q \rightarrow q'$ para indicar que o sistema pode fazer a transição do estado q para o estado q' e dizer que q' é diretamente acessível a partir de q. Estando em um estado q, o sistema pode selecionar, após algum tempo finito, alguma transição que é possível de ser executada a partir de q. Essa liberdade introduz, na maioria dos casos, um não-determinismo no comportamento do sistema. Um sistema é dito, então, determinístico se, e somente se, para cada $q \in Q$ existe no máximo um q' tal que $q \rightarrow q'$; caso contrário, o sistema é não-determinístico [Bochmann 83]. Os modelos a seguir podem apresentar uma representação semântica de sistema de transições:

A) MÁQUINAS DE ESTADOS FINITAS

Uma máquina de estados finita [Danthine 80] é uma 5-tupla (X,I,O,N,M) onde X é um conjunto finito de estados; I é um conjunto finito de entradas; O é um conjunto finito de saídas; N é uma função de transição de estados e M é uma função de saída (e ação). N e M expressam o comportamento da máquina. Se, em qualquer estado, uma entrada é recebida, a função de saída irá indicar a saída a ser gerada e a função de transição de estados irá indicar o novo estado da máquina. Mensagens recebidas pertencem ao conjunto das entradas e mensagens enviadas ao conjunto das saídas, sendo também permitido introduzir, no conjunto das entradas, "eventos internos" ou "eventos nulos" que são necessários para modelar eventos que ocorrem fora da especificação mas diretamente relacionados com o seu comportamento;

B) REDES DE PETRI

A rede de Petri [Murata 89] é uma 5-tupla $PN=(P,T,F,W,M_0)$ onde P é um conjunto finito de lugares, T é um conjunto finito de transições, F é um conjunto de arcos, W é a função peso e M_0 é a marcação inicial. A rede de Petri possui dois tipos de nós, chamados *lugares* e *transições*, e conectados através e arcos que unem ou um lugar a uma transição ou uma transição a um lugar. Na representação gráfica da rede de Petri, os lugares são desenhados como círculos e as transições como barras ou caixas. Os arcos são rotulados com os seus pesos (inteiros positivos), onde um arco de peso k pode ser interpretado como um conjunto de k arcos paralelos. Uma *marcação* (estado) atribui a cada lugar um inteiro não negativo; se uma marcação atribui a lugar p um inteiro não negativo k, diz-se que o lugar p está *marcado com k fichas*. Na representação gráfica isto é feito desenhando-se k pontos pretos (fichas) no interior do lugar p.

Uma marcação é denotada por *M*, um vetor de tamanho *m* onde é o número total de lugares. O *p*-ésimo componente de *M*, denotado por *M(p)*, é o número de fichas no lugar *p*. Na modelagem, usando o conceito de *condições* e *eventos*, os lugares representam as condições e as transições representam os eventos. Uma transição (evento) possui um certo número de lugares de *entrada* e de *saída* representando as pré-condições e as pós-condições do evento, respectivamente. A representação do comportamento dinâmico de um sistema pode ser simulado por uma redé de Petri, através da mudança da marcação da rede, de acordo com as seguintes regras de disparo das transições:

- uma transição *t* é dita estar *habilitada* se cada lugar de entrada *p* de *t* está marcado com pelo menos *w(p,t)* fichas, onde *w(p,t)*é o peso do arco de *p* para *t*;
- uma transição habilitada pode ser ou não disparada (dependendo se o evento de fato acontece ou não);
- o disparo de uma transição retira *w(p,t)* fichas de cada lugar de entrada *p* de *t* e adiciona *w(t,p)* fichas em cada lugar de saída *p* de *t*, onde *w(t,p)* é o peso do arco de *t* para *p*.

Um outro tipo de sistema que pode ser representado como um sistema de transição são as *álgebras de processos*, nas quais um sistema distribuído é descrito por uma equação de comportamento citando-se, em particular, o CCS (*Calculus of Communicating Systems*) [Milner 80].

2.3.2.2 - Outros Modelos Formais

Outros modelos, que não utilizam explicitamente os sistemas de transições, foram desenvolvidos, tendo suas origens na teoria de linguagens da lógica; dentre estes modelos, destacam-se:

- as ***gramáticas formais*** [Harangozo 78] onde as frases geradas pela gramática correspondem às seqüências válidas de eventos que caracterizam o sistema;
- os ***tipos abstratos de dados algébricos*** nos quais, pelas regras de escrita é possível, de maneira semi-automática (definição de esquemas de provas), verificar as propriedades da especificação;

- a ***lógica temporal*** [Schwartz 82] que permite derivar seqüências válidas de eventos de um sistema a partir de asserções de lógica temporal caracterizando a estrutura dessa seqüências.

2.3.2.3 - Modelos Híbridos

Os modelos híbridos foram desenvolvidos para especificar as propriedades de um sistema através de duas técnicas complementares - por exemplo, redes de Petri e lógica temporal [Diaz 83] e, de uma maneira mais geral, para responder a uma necessidade de especificar conjuntamente o paralelismo e o tratamento dos dados como, por exemplo, um modelo misto rede de Petri/tipos abstratos algébricos [Martin 86].

Dentre estes modelos híbridos destacam-se as três TDFs (Estelle, LOTOS e SDL), adotadas como padrão internacional para especificação de protocolos de comunicação. A seguir, serão apresentadas as características principais da técnica de descrição formal Estelle.

2.4 - Estelle

Estelle (Extended State Transition Language) [Linn 85, Budkowski 87, Courtiat 87a, Estelle 88] é a segunda técnica de descrição formal desenvolvida e padronizada pela ISO baseada em uma máquina de estados finita estendida, que utiliza a linguagem de programação Pascal para a representação dos dados. A seguir serão apresentadas as características principais de Estelle.

2.4.1 - Conceitos Básicos

Uma especificação Estelle é constituída de uma coleção de componentes comunicantes denominados **módulos**. Cada módulo apresenta um número finito de pontos de interação (externos ou internos), através dos quais um conjunto de interações pode ser enviadas e recebidas.

O comportamento interno de um módulo é descrito por uma máquina de estados estendida na qual as ações são comandos Pascal. Um módulo pode ser ativo se possui pelo

menos uma transição. Caso contrário, o módulo é dito inativo (ou passivo). Finalmente, um módulo pode possuir um dos seguintes atributos: *systemprocess, process, systemactivity* ou *activity*, que serão descritos a seguir.

2.4.2 - A Estrutura Hierárquica e o Paralelismo

Os módulos que formam a arquitetura da especificação podem ser aninhados, ou seja, podem ser refinados em submódulos, constituindo uma estrutura hierarquizada e uma relação pai/filho entre os componentes da especificação. Essa hierarquia deve respeitar os seguintes princípios em relação aos atributos citados na seção anterior [Budkowski 87]:

- todo módulo ativo deve possuir um atributo;
- subsistemas (módulos *systemprocess* ou *systemactivity*) não podem ser aninhados no interior de um módulo com atributo;
- módulos *process* ou *activity* devem ser aninhados no interior de um subsistema;
- módulos *process* e *systemprocess* só podem ser refinados em módulos *process* ou *activity*;
- módulos *activity* e *systemactivity* só podem ser refinados em módulos *activity*.

Além disso, por coerência com os princípios acima, qualquer módulo que incorpore subsistemas deve ser inativo e sem atributo. Os atributos acima definem ainda a maneira pela qual as transições de uma especificação serão executadas, com relação aos aspectos de paralelismo e não-determinismo. Dessa forma são possíveis os seguintes tipos de paralelismo:

- ***paralelismo assíncrono*** entre subsistemas (*systemprocess* ou *systemactivity*). Cada um dos subsistemas seleciona um conjunto de transições prontas para disparar (no máximo uma por módulo) e as executa em paralelo, de forma independente. Por exemplo, entre os módulos A1 e A2 da figura 2.2;
- ***paralelismo síncrono*** entre transições de um mesmo subsistema. O subsistema seleciona as transições prontas para disparar (no máximo uma por módulo) e as executa simultaneamente. Somente após o término da execução de todas as transições uma próxima seleção de transições pode ocorrer. O paralelismo síncrono ocorre entre módulos *process* ou *activity* descendentes de um subsistema

systemprocess. Os módulos A11 e A12 da figura 2.2 apresentam este tipo de paralelismo;

- ***não-determinismo*** entre transições de um mesmo subsistema. O subsistema seleciona as transições prontas para disparar (no máximo uma por módulo) e executa apenas uma delas, sendo a escolha dessa transição não-determinista. O não-determinismo ocorre entre módulos *activity* descendentes de um subsistema *systemactivity*, como nos módulos A21 e A22 da figura 2.2.

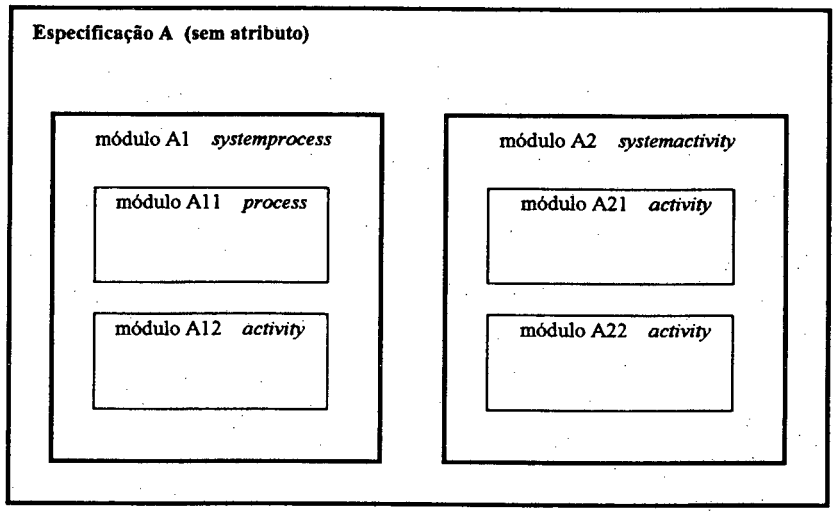


Figura 2.2 - Exemplo da Estrutura de uma Especificação Estelle

É importante notar que, independentemente do modo como são definidos os módulos, a escolha de qual transição - dentre várias selecionadas - será disparada no interior de um módulo é sempre não determinista.

2.4.3 - A Comunicação em Estelle

A comunicação entre módulos Estelle pode ser efetuada através de troca de mensagens ou através de um compartilhamento restrito de variáveis. Um módulo pode enviar uma mensagem (também chamada interação) a um outro módulo desde que exista uma ligação preestabelecida entre dois pontos de interação desses módulos. Uma interação recebida por um módulo no seu ponto de interação é anexada a uma fila FIFO ilimitada associada a este ponto de interação. Uma fila FIFO pode pertencer exclusivamente a um único ponto de interação e, nesse caso, é chamada de fila individual (*individual queue*). Pelo contrário, se a fila é compartilhada

entre todos os pontos de interação de um módulo, ela é chamada de fila comum (*common queue*). Uma interação só pode ser enviada por um ponto de interação que seja um ponto final de uma ligação de comunicação e, além disso, essa interação será recebida somente pelo outro ponto final dessa ligação.

Em relação ao compartilhamento de variáveis, este só é permitido entre um módulo filho e o seu módulo pai. Essas variáveis devem ser declaradas como *exported* pelo módulo filho, sendo que o acesso simultâneo a essas variáveis pelos módulos filho e pai é impedido devido ao princípio da prioridade pai/filho, na qual a execução das transições do módulo pai são prioritárias em relação às do módulo filho.

2.4.4 - A Estrutura de Ligações

A estrutura de comunicação define a ligação entre os pontos de interação dos módulos que podem ser conectados (*connect*) ou vinculados (*attach*). Dois pontos de interação são ditos conectados quando, ligam dois pontos de interação externos de dois módulos "irmãos" ou, ligam dois pontos de interação internos de um mesmo módulo ou, ainda, ligam um ponto de interação externo de um módulo e um ponto de interação interno do seu módulo pai. Dois pontos de interação são ditos vinculados quando ligam um ponto de interação externo de um módulo e um ponto de interação externo do seu módulo pai. A figura 2.3 a seguir mostra essas possíveis ligações.

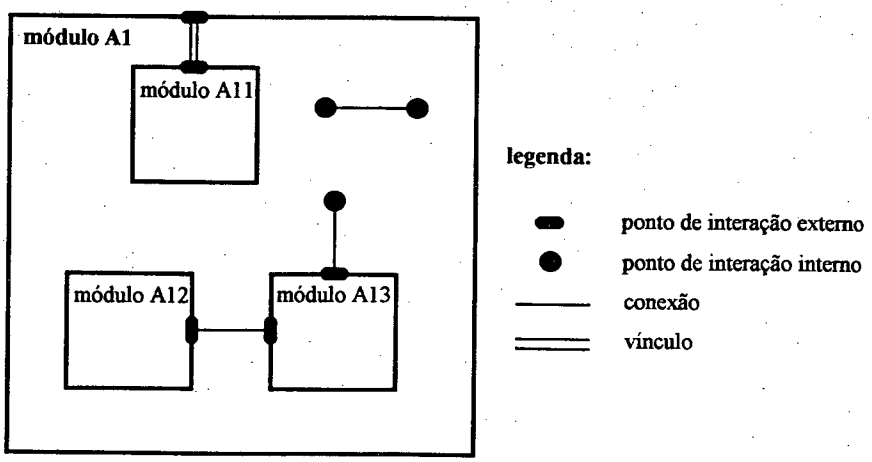


Figura 2.3 - A Estrutura de Ligações

Essas ligações entre pontos de interação devem, contudo, obedecer às seguintes restrições [Budkowski 87]:

- um ponto de interação pode ser conectado a no máximo um ponto de interação e não pode ser conectado a ele mesmo;
- um ponto de interação externo de um módulo pode ser vinculado a no máximo um ponto de interação externo do seu módulo pai e a no máximo um ponto de interação de seus módulos filhos;
- um ponto de interação externo de um módulo vinculado a um ponto de interação externo do seu módulo pai não pode estar simultaneamente conectado.

Como citado anteriormente, módulos que incorporam subsistemas são sempre inativos e, sendo assim, a estrutura dos subsistemas e suas ligações de comunicação, uma vez inicializadas, não podem ser alteradas, o que lhe atribui um comportamento **estático**. Por outro lado, a estrutura interna de cada subsistema pode variar, já que as ações de um módulo podem incluir comandos de criação e destruição de módulos filhos e de ligações de comunicação, caracterizando um comportamento **dinâmico**.

2.4.5 - A Noção de Tempo em Estelle

A noção de tempo é utilizada em Estelle para interpretar propriamente os *delays* ou atrasos. Tais atrasos são valores dinâmicos atribuídos a algumas transições que indicam o número de unidades de tempo que a execução dessas transições deve ser atrasada. Em Estelle é assumida a hipótese de que os tempos de execução das ações são desconhecidos, por serem considerados dependentes da implementação. As transições Estelle que devem ser atrasadas devem conter uma cláusula de atraso da forma "*delay*(E1, E2)". Essa cláusula indica que a execução de uma transição (se ela estiver habilitada) deve ser atrasada. Os tempos mínimo e máximo nos quais a transição pode ser atrasada são especificados pelas expressões inteiras E1 e E2, respectivamente. Nesse caso, uma implementação pode escolher um valor de atraso concreto no intervalo fechado determinado por essas expressões.

2.4.6 - Aspectos Sintáticos de Estelle

2.4.6.1 - Canais e Pontos de Interação

Os canais são construções que definem um conjunto específico de interações. Os pontos de interação fazem referência a esses canais e, dessa maneira, definem precisamente o conjunto de interações que podem ser enviadas e recebidas pelos mesmos, como pode ser observado na definição do canal a seguir:

```
channel IDENT_CANAL (PAPEL1, PAPEL2)
        by PAPEL1: IDENT_INTERAÇÃO (LISTA_PARÂMETROS);
                                .
                                .
                                .
                   IDENT_INTERAÇÃO (LISTA_PARÂMETROS);
        by PAPEL2: IDENT_INTERAÇÃO (LISTA_PARÂMETROS);
                                .
                                .
                                .
                   IDENT_INTERAÇÃO (LISTA_PARÂMETROS);
```

Pela definição acima, dois módulos **conectados** através de seus pontos de interação às extremidades desse canal desempenharão, respectivamente, as funções PAPEL1 e PAPEL2. O ponto de interação do módulo que desempenhar a função PAPEL1 poderá enviar as primitivas definidas em PAPEL1 e receber as primitivas definidas em PAPEL2. O contrário deve ocorrer com o ponto de interação do segundo módulo, o qual desempenhará a função PAPEL2. A definição dos pontos de interação para ambos os módulos seria, respectivamente:

```
p1: IDENT_CANAL (PAPEL1) e

p2: IDENT_CANAL (PAPEL2)
```

Nota-se, para o caso acima, que os pontos de interação p1 e p2 desempenham papéis opostos, pois como estão conectados, uma interação enviada por um dos pontos de interação deverá ser recebida pelo segundo e vice-versa. No caso em que dois pontos de interação estejam **vinculados** às extremidades desse canal, eles deverão desempenhar o mesmo papel, já que as interações recebidas por um deles deverão ser "repassadas" para o segundo. Assim, as definições de ambos os pontos de interação seriam:

```
p1: IDENT_CANAL (PAPEL1)
p2: IDENT_CANAL (PAPEL1)    ou,
```

p1: IDENT_CANAL (PAPEL1)
p2: IDENT_CANAL (PAPEL1)

2.4.6.2 - Módulos e Instâncias de Módulos

Uma instância de módulo é uma instância de um tipo de módulo genérico, podendo existir várias instâncias de um mesmo tipo de módulo simultaneamente durante a execução de uma especificação. Um módulo é especificado através da definição do cabeçalho do módulo e do corpo do módulo.

A definição do cabeçalho do módulo especifica o tipo do módulo cujos valores são instâncias com a mesma visibilidade externa, ou seja, com os mesmos pontos de interação, mesmas variáveis exportadas e mesmo atributo. Essa definição contém o nome do módulo e, opcionalmente, um atributo (*systemprocess, process, systemactivity* ou *activity*), uma lista de parâmetros formais e declarações de pontos de interação e variáveis exportadas.

Pelo menos uma definição de corpo de módulo é declarada para cada definição de cabeçalho do módulo. A definição do corpo do módulo contém o nome do corpo e uma referência ao nome do cabeçalho do módulo associado. O corpo do módulo é composto de três partes [Budkowski 87]:

- ***parte declaração***
- ***parte inicialização***
- ***parte transição***

A **parte declaração** contém declarações usuais do Pascal (constantes, variáveis, procedimentos e funções) e declarações de objetos específicos de Estelle como canais, módulos, variáveis módulo, estados (e conjuntos de estados) e pontos de interação internos.

A **parte inicialização** de um corpo de módulo especifica os valores de algumas variáveis do módulo com os quais cada nova instância de módulo inicia a sua execução. Em particular, podem ser atribuídos valores a variáveis locais e a variáveis de controle. Variáveis módulo podem ser inicializadas, ou seja, instâncias de módulos descendentes podem ser criadas.

A **parte transição** descreve o comportamento interno do módulo, através da declaração de um conjunto de transições. Cada transição é composta de um par **condição/ação**.

A condição da transição consiste em uma ou mais cláusulas do tipo: *from*, *when*, *provided*, *priority* e *delay*. A cláusula "*from* ESTADO" caracteriza o ESTADO de controle no qual deve-se encontrar a instância de módulo; a cláusula "*when* p.INT", permite verificar a presença da interação INT no topo da fila associada ao ponto de interação p; a cláusula "*provided* B", onde B é uma expressão booleana, habilita ou não a transição se a avaliação dessa expressão resulta, respectivamente, verdadeira ou falsa; a cláusula "*priority* N", onde N é uma constante não negativa, determina a prioridade dessa transição em relação às outras; a cláusula "*delay* (T1, T2)" especifica um tempo mínimo T1 e um tempo máximo T2 entre os quais a transição pode ser disparada após ter sido habilitada (contanto que ela permaneça habilitada por pelo menos T1 unidades de tempo). Algumas dessas cláusulas podem ser omitidas e no máximo uma cláusula de cada tipo pode aparecer na condição de uma transição. Além disso, as cláusulas *when* e *delay* são exclusivas entre si.

A ação de uma transição é composta de uma cláusula "*to* ESTADO" especificando o próximo ESTADO de controle da instância de módulo, após o disparo da transição (se omitida, o próximo estado é o mesmo estado corrente) e um bloco de instruções Pascal ("*begin...end*") que se executam no momento do disparo da transição.

As extensões Estelle em relação a Pascal consistem em comandos adicionais que permitem: criar instâncias de módulo ("*init*"), destruir intâncias de módulos ("*release*"), criar ligações entre pontos de interação ("*connect*", "*attach*"), destruir ligações entre pontos de interação ("*disconnect*", "*detach*") e enviar interações ("*output*"). Além desses, outros comandos são permitidos ("*all*", "*forone*", "*exist*").

As principais restrições quanto ao uso de Pascal adotadas em Estelle são as seguintes:

- todas as funções declaradas são puras, ou seja, sem efeitos colaterais;
- ponteiros só podem ser utilizados em construções puramente Pascal;
- uso restrito da função *goto*;
- as declarações *read* e *write* não podem ser utilizadas;
- o tipo *file* não é permitido;
- *conformance arrays* não podem ser utilizados.

2.4.7 - Estelle*

Estelle* [Courtiat 87b, Courtiat 88] é uma versão de Estelle desenvolvida com o objetivo de prover as facilidades necessárias para aumentar o nível de abstração de algumas características de implementação impostas por Estelle. Estas características de implementação, simplificadas em [Courtiat 87b], dizem respeito ao conceito de sistema, a prioridade entre transições, a semântica do tempo e o mecanismo de comunicação através de filas FIFO.

De modo a atingir esse objetivo, Estelle* apresenta as seguintes diferenças em relação a Estelle:

- os atributos *systemprocess* e *process* foram eliminados. Uma discussão a respeito da semântica do paralelismo encontra-se em [Courtiat 88];
- as prioridades entre transições (cláusula *priority* e prioridade pai/filho) foram removidas;
- a semântica do tempo foi derivada da semântica do tempo definida para as Redes de Petri Temporais [Berthomieu 83], na qual é suposto que o tempo de disparo de uma transição (o qual não pode ser expresso em Estelle) é muito pequeno em relação aos valores de tempo especificados na cláusula *delay* e pode ser considerado nulo;
- introdução de um mecanismo de comunicação síncrono (*rendez-vous*) [Courtiat 89]. O mecanismo de filas FIFO de Estelle pode não ser satisfatório quando aplicado à comunicação entre entidades de protocolos de camadas adjacentes. Primeiramente porque é imposto um esquema particular de implementação a um conceito abstrato (o conceito de pontos de acesso ao serviço). Além disso, o fato de representar primitivas de serviço entre entidades de protocolo através de interações armazenadas em filas introduz alguns problemas na interface entre camadas como colisões de primitivas de serviço, conflitos na alocação de pontos finais de conexão e impossibilidade de expressão do fluxo de controle *backpressure*, ou seja, considerando que um módulo receptor e um módulo fornecedor de dados são interconectados por uma fila infinita, o módulo receptor não consegue impedir o módulo fornecedor de colocar dados na fila.

O mecanismo de *rendez-vous* é um mecanismo explícito de sincronização no qual duas transições são disparadas simultaneamente, com uma possível passagem de valores. Com a adição desse mecanismo as cláusulas *when* e *output* permanecem dedicadas ao mecanismo de filas FIFO. Uma nova cláusula, o pedido de sincronização, é introduzida para expressar uma

sincronização explícita entre duas transições, conseqüentemente um pedido de sincronização pode aparecer somente a nível da guarda de uma transição. No máximo um pedido de sincronização pode existir na guarda de uma transição, o qual pode ser uma sincronização em **recepção** em algum ponto de interação P, ou uma sincronização em **emissão** em algum ponto de interação P. Em ambos os casos P deve ser um ponto de interação externo declarado como *no queue*. A notação para sincronização em recepção e sincronização em emissão (respectivamente **P?interação** e **P!interação**) é similar à notação utilizada em CSP (*Communicating Sequential Process*). A propósito, a notação usada para expressar as cláusulas de sincronização referem-se às usadas em CSP; de fato, o mecanismo de *rendez-vous* de Estelle* pode ser visto como uma generalização do *rendez-vous* de CSP para o caso de uma configuração dinâmica de instâncias de módulo [Courtiat 88].

2.5 - Conclusão

Neste capítulo foram apresentados alguns aspectos da concepção de aplicações distribuídas, enfatizando as etapas de concepção (análise de requisitos, projeto, implementação e manutenção) e dirigindo uma atenção especial às diversas técnicas de validação de aplicações distribuídas como a verificação, a simulação, a experimentação e o teste de implementação do protótipo.

Foi mostrado também que as técnicas de descrição formal tornaram-se indispensáveis para o desenvolvimento de aplicações distribuídas. Isto é função tanto do seu formalismo, que possibilita a obtenção de especificações claras e livres de ambigüidades, quanto da possibilidade de se utilizar ferramentas automatizadas de suporte às diversas etapas desse desenvolvimento. Deve-se salientar aqui que as técnicas de descrição formal vistas neste capítulo foram desenvolvidas visando principalmente a especificação de protocolos de comunicação, todavia a sua utilização pode ser estendida às demais áreas dos sistemas distribuídos. Dentre essas técnicas, foi dada uma ênfase maior a Estelle por ser ela a base do presente trabalho. Optou-se por utilizar Estelle, primeiramente por ser esta uma técnica com a qual inúmeros trabalhos têm sido realizados e, principalmente, pela disponibilidade de ferramentas de suporte, imprescindíveis para a utilização de qualquer técnica de descrição formal. Ainda neste capítulo, foi apresentada uma versão de Estelle, denominada Estelle*, caracterizada principalmente por permitir o desenvolvimento de especificações em um nível de abstração mais elevado.

CAPÍTULO 3

AS FERRAMENTAS AUTOMATIZADAS DE SUPORTE À ESTELLE

3.1 - Introdução

Foi visto no capítulo anterior que uma das principais vantagens da especificação de sistemas através de técnicas de descrição formal é a possibilidade de se utilizar ferramentas automatizadas de suporte a essas técnicas. Os benefícios do uso de tais ferramentas são indiscutíveis, principalmente no caso de aplicações distribuídas (inerentemente complexas) nas quais as atividades de validação, por exemplo, tornam-se inviáveis sem o emprego de ferramentas desse tipo. Como conseqüência, o desenvolvimento de ferramentas de suporte às técnicas de descrição formal tornou-se uma área de pesquisa de grande interesse.

Atualmente existem inúmeras ferramentas de suporte à Estelle, cada qual destinada a cobrir uma (ou mais) etapas do processo de desenvolvimento de uma aplicação distribuída. Em [Bochmann 87a] são listadas diversas dessas ferramentas. Dentre os mais variados tipos de ferramentas encontram-se editores orientados à sintaxe, simuladores, verificadores, geradores de código e geradores de seqüências de teste. Algumas ferramentas são desenvolvidas visando a sua integração em um ambiente como, por exemplo, o EWS (*Estelle Workstation*) [Ayache 89] que possui um editor orientado à sintaxe, um tradutor, um gerador de código C, um núcleo de implementação e um simulador orientado à depuração; e o EDS (*Estelle Development System*) [Chung 90] que possui um compilador Estelle, um analisador de máquinas de estado finitas, um gerador de referências cruzadas, um gerador de documentos e um módulo de gerenciamento de testes.

Neste capítulo serão apresentadas as características principais de três ferramentas automatizadas de suporte a Estelle disponíveis no LCMI. A primeira delas é o ambiente ESTIM, uma ferramenta de validação de especificações Estelle* que possui facilidades de simulação e verificação. A seguir será visto o EDT que consiste em uma ferramenta de simulação (EDB) e um compilador de Estelle para C (EC). A última ferramenta a ser apresentada será o ECHIDNA,

o qual permite uma simulação distribuída da especificação Estelle, através da geração de um código distribuído em linguagem C. No final do capítulo uma tabela com a sinopse das funcionalidades de cada ferramenta será mostrada.

3.2 - ESTIM

O ESTIM (Estelle SimulaTor based on an Interpretative Machine) [Saqui 89a, Saqui 90a, Saqui 90b, Courtiat 92] desenvolvido dentro do projeto SEDOS (*Software Environment for the Design of Open distributed Systems*) [Diaz 89a, Diaz 89b] é uma ferramenta de validação de especificações Estelle* que apresenta facilidades tanto de simulação quanto de verificação. O ESTIM foi escrito na linguagem de prototipagem ML (*Meta-Language*) [Wirsing 87] e executa sobre o sistema operacional UNIX.

O ESTIM suporta a versão Estelle* e, desta forma, é restrito aos mecanismos disponíveis nesta versão. Conseqüentemente, não é permitida a utilização dos atributos *systemprocess* e *process*. Com relação aos mecanismos de prioridade (cláusula *priority* e prioridade pai/filho) existe uma opção que permite habilitar tais mecanismos. Em contrapartida, o ESTIM suporta, além do mecanismo de comunicação através de filas FIFO, o mecanismo de comunicação síncrono (*rendez-vous*). Outro ponto é a implementação de uma semântica particular do tempo (cláusula delay) baseada nas Redes de Petri Temporais.

3.2.1 - A Simulação no ESTIM

O ESTIM oferece quatro tipos de funcionalidades com relação à simulação que são o acesso ao estado global da especificação, o controle da simulação, a estatística da simulação e os comentários de qualificação. A seguir são detalhadas cada uma destas funcionalidades.

3.2.1.1 - Acesso ao Estado Global da Especificação

Devido à atomicidade das transições Estelle, os objetos (Estelle ou Pascal) da especificação são basicamente acessíveis após cada disparo de transição. Os objetos que podem ser acessados são:

- a arquitetura da especificação, ou seja, as instâncias de módulos existentes;
- a árvore hierárquica caracterizando a relação entre as instâncias de módulos;
- a configuração dos pontos de interação ("*connect*" e "*attach*");
- o estado global das instâncias de módulos;
- as constantes, variáveis (locais e exportáveis) e as interações armazenadas nas filas com seus parâmetros;
- os intervalos de tempo de disparo dinâmicos.

Além disso, o usuário tem ainda a possibilidade de modificar alguns destes componentes, como o estado global de uma determinada instância de módulo, o valor de alguma variável pertencente a uma instância de módulo particular e o conteúdo das filas.

3.2.1.2 - Controle da Simulação

O controle da simulação no ESTIM pode ser realizado através de quatro modos de disparo das transições:

- ***modo passo a passo***: este modo tem como o objetivo a depuração da especificação. Ele possibilita ao usuário analisar o comportamento das instâncias de módulo, através da observação de como as variáveis e os estados globais são modificados após o disparo de cada transição e também das novas interações armazanadas nas filas. O usuário, neste modo, atua interativamente, selecionando a cada passo uma entre todas as transições disparáveis da especificação. Este modo permite ainda a investigação de algumas seqüências de disparos de transições predefinidas.
- ***modo randômico***: este é um modo próprio para analisar o comportamento global da especificação. É principalmente utilizado para detectar estados de bloqueio (*deadlock*). Aqui o usuário seleciona um número de transições a serem disparadas. O controle da simulação só retorna ao usuário após todas as transições terem sido disparadas ou se uma condição de bloqueio for atingida. O ESTIM oferece a possibilidade de guardar o traço da simulação em um arquivo, para uma posterior análise.

- ***modo automático controlado pelo usuário***: este modo permite ao usuário garantir uma maior "justiça" na seleção de instâncias de módulos que possuam transições disparáveis desabilitando o não-determinismo na seleção das transições. Em particular, as instâncias de módulo podem ser escolhidas segundo uma política *round-robbin*. Por outro lado, o usuário pode privilegiar o disparo de transições de entrada (transições cuja cláusula *when* não é vazia), de modo a observar as trocas de interações.

- ***modo execução***: este modo permite ao usuário observar uma seqüência de disparos de transições previamente armazenada em um arquivo.

3.2.1.3 - Estatística da Simulação

Independentemente do modo de disparo de transições escolhido, o ESTIM dispõe de algumas informações em relação à estatística da sessão de simulação. Em particular, são disponíveis as seguintes informações:

- a lista de transições que foram disparadas desde o início da sessão de simulação;
- a lista de transições não disparadas;
- a taxa de transições disparadas por instância de módulo;
- a lista de estados globais alcançados por instância de módulo.

3.2.1.4 - Comentários de Qualificação

Um comentário de qualificação ("*qualifying comment*") é um comentário especial que consiste em uma linha de texto escrita em ML e que permite inserir diretivas transparentes na especificação. Os delimitadores de início de comentário em Estelle são os caracteres (* ou { e os de final de comentário são *) ou }. Para que o ESTIM reconheça um comentário como sendo um comentário de qualificação os caracteres delimitadores de início devem ser seguidos de um caracter "dólar", sob a forma (*$ *comentário de qualificação* *). A utilização de comentários de qualificação permite realizar traços da sessão de simulação através da impressão de mensagens ou de valores de variáveis. Além disso, é permitido inserir também um "ponto de parada" dentro do bloco de uma transição. Essa extensão vai de encontro ao princípio da atomicidade das

transições de Estelle e, sendo assim, ao usuário só é permitido observar os objetos sem contudo poder efetuar quaisquer modificações no estado global da especificação.

3.2.2 - A Verificação no ESTIM

O método de análise utilizado pelo ESTIM é baseado na geração do **grafo de alcançabilidade**, ou seja, do grafo de estados acessíveis a partir do seu estado inicial. Como foi citado no capítulo 2, a análise desse grafo pode ser extremamente trabalhosa devido ao problema da explosão do espaço de estados. Um dos principais motivos dessa explosão do espaço de estados é a capacidade ilimitada das filas FIFO. Como tentativa de reduzir tal problema, o ESTIM oferece a opção de se limitar a capacidade das filas, através do comentário de qualificação (" *queue-length* <inteiro> ") diante de toda declaração de ponto de interação do tipo *individual-queue*. Mesmo assim, o grafo gerado pode atingir dimensões tais que o processo de análise seja ainda bastante difícil e técnicas de redução do grafo devem ser aplicadas para facilitar a sua análise.

O ESTIM utiliza uma técnica conhecida como **análise por abstração** (ou por projeção), a qual consiste em aplicar ao grafo de alcançabilidade técnicas de redução do grafo baseadas em relações de equivalência. Os tipos mais comuns de equivalência - bissimulação, observacional, teste e traço - serão vistos no capítulo seguinte.

3.2.3 - As Interfaces do ESTIM

A primeira interface do ESTIM é com a ferramenta GENESTIM. O GENESTIM primeiramente invoca um **tradutor**, que tem como entrada uma especificação Estelle* sobre a qual ele detecta erros de sintaxe e checa a semântica estática. O tradutor produz então uma **Forma Intermediária (FI)** que contém todas as informações semânticas relevantes da especificação original. Essa especificação FI é convertida pelo GENESTIM em uma árvore abstrata na linguagem ML, a qual servirá de entrada para o ESTIM (figura 3.1).

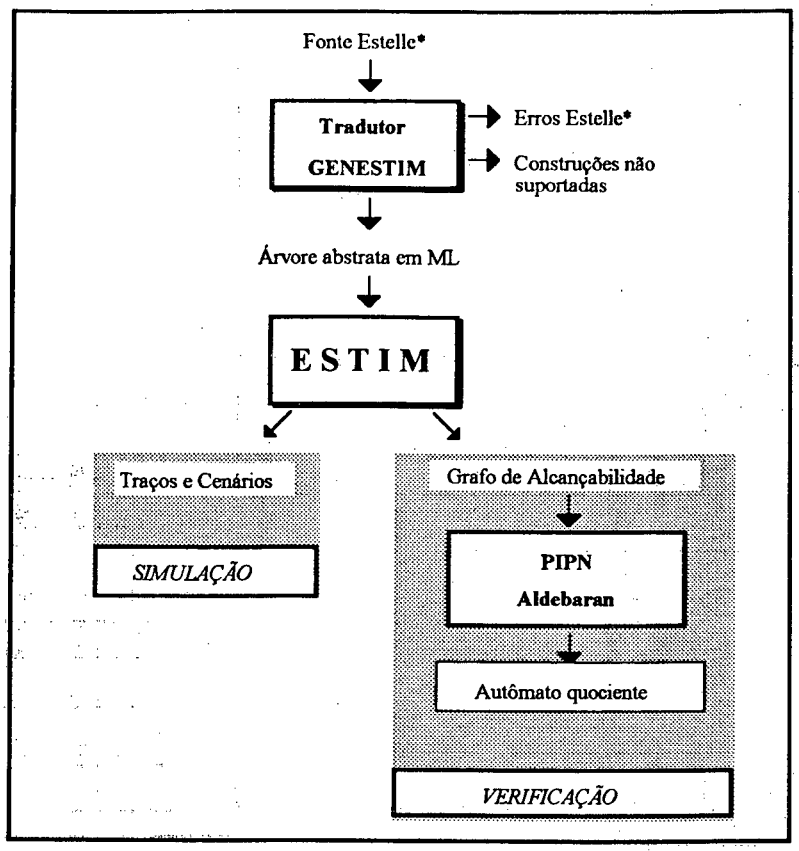


Figura 3.1 - As Interfaces do ESTIM

A segunda interface do ESTIM é com dois verificadores de sistemas de transição responsáveis pela redução do grafo de alcançabilidade. Esses verificadores são o PIPN (*Prolog Interpreted Petri Nets*) [Lloret 90], para projeções de traço (ou linguagem) e observacional e o Aldebaran [Fernandez 88] que realiza projeções de bissimulação, observacional e teste (figura 3.1).

3.3 - EDT

O EDT (Estelle Development Toolset) é um pacote composto de duas ferramentas: o **EDB**, destinado à simulação e o **EC**, que faz a compilação de uma especificação em Estelle para a linguagem "C". A seguir serão apresentadas as duas ferramentas.

3.3.1 - EDB (Estelle Simulator/Debugger)

O EDB [EDB 92] possui recursos de simulação e depuração semelhantes aos do ESTIM. Além destes recursos, o EDB permite que sejam inseridas nas especificações Estelle restrições temporais definidas pelo próprio usuário possibilitando, por exemplo, especificar explicitamente o tempo médio de execução de uma transição (que por *default* é considerado nulo). Assim, problemas dependentes da implementação e relacionados com o desempenho do sistema podem ser estudados num ambiente de simulação.

O EDB oferece ao usuário diversos comandos que permitem mostrar o valor dos objetos da especificação ou controlar o processo de simulação. Uma linguagem simples é definida permitindo ao usuário compor esses comandos para preparar o ambiente de simulação. Além disso, seqüências de comandos podem ser declaradas como **macrocomandos**, os quais podem ser utilizados como cenários de simulação.

As funcionalidades do EDB, bem como seus principais comandos, serão apresentados aqui procurando manter uma analogia com a apresentação das funcionalidades de simulação do ESTIM.

3.3.1.1 - Acesso aos Objetos da Especificação

Através do comando *display*, o usuário pode acessar tanto objetos da especificação Estelle quanto objetos internos do EDB. Exemplos de alguns objetos que podem ser acessados são:

- a árvore das instâncias de módulo;
- os pontos de interação de uma determinada instância de módulo;
- o estado global de uma instância de módulo;
- as variáveis do usuário;
- as interações armazenadas nas filas e seus parâmetros.

Os objetos acima, além de vários outros, podem ser mostrados na tela através do comando *display* ou ainda gravados em um arquivo utilizando o comando *file-display*.

O acesso aos objetos no EDB possui uma diferença do acesso aos objetos no ESTIM. Isto porque, no ESTIM, os comandos possuem um escopo global enquanto que no EDB os comandos são restritos à instância de módulo corrente. Por exemplo, o comando *dq* do ESTIM mostra o conteúdo de todas as filas da especificação. Já o comando equivalente *$dq* do EDB mostra apenas o conteúdo das filas associadas à instância de módulo corrente. Devido a tal característica o EDB possui **comandos de navegação** para que o usuário possa percorrer todas as instâncias de módulo durante a simulação.

Como no caso do ESTIM, o EDB também permite modificar alguns componentes da especificação, utilizando os **comandos de modificação**.

3.3.1.2 - Controle da Simulação

O EDB apresenta basicamente um modo passo a passo e um modo randômico para o controle do disparo das transições:

- ***modo passo a passo***: este modo de depuração é semelhante ao do ESTIM, porém oferece ao usuário uma liberdade maior quanto ao grau de precisão do disparo das transições, ou seja, permite ao usuário selecionar não apenas uma transição a ser disparada, como também uma instância de módulo na qual uma transição será disparada, ou ainda um subsistema no qual uma transição será disparada.

- ***modo randômico***: este modo permite que o usuário defina um número máximo de transições a serem disparadas ou um tempo máximo (em segundos) de simulação, ou ambos os casos. Como citado anteriormente, o usuário pode utilizar a linguagem definida pelo EDB para criar cenários de simulação (tais cenários podem ser declarados como macrocomandos). Além disso, é introduzida a noção de **observador**, que é um programa escrito na linguagem definida pelo EDB e executado na sessão de simulação que permite descrever uma situação particular que o usuário deseja observar ou detectar.

O observador pode mostrar na tela interativamente, ou gravar em um arquivo, algumas características que o usuário deseja observar, sendo ainda permitido ao usuário inserir mais de um observador simultaneamente. A seguir, tem-se um exemplo de utilização do observador, extraído e adaptado do manual de referência do EDB [EDB 92]. O observador do exemplo permite interromper a simulação quando todas as transições, em todos os módulos, houverem sido executadas pelo menos uma vez; neste caso a mensagem *"todas as transições foram*

executadas pelo menos uma vez" será impressa, bem como o número de vezes que cada transição foi disparada. Neste exemplo, assume-se ainda a existência de 5 instâncias de módulo, porém somente as intâncias 2, 3, 4 e 5 possuem transições; o símbolo # indica uma variável do usuário, o símbolo $ um comando do simulador EDB e as palavras em *itálico* são palavras-chave da linguagem definida pelo EDB.

```
set_observer{\
        #todas := true;\
        #num_intancia := 2;\
        do\
                if #num_intancia = 5\
                  then exit\
                fi;\
                if $nbuse(#num_intancia -> $lftr(#num_intancia)) = 0\
                        then #todas := false;\
                                exit
                fi;\
                #num_intancia := #num_intancia + 1\
        od;\
        if #todas\
                then print "todas as transições foram executadas pelo menos uma
                 vez";\
                    d$stat;\
                    break\
        fi\
}
```

3.3.1.3 - Estatística da Simulação

O EDB também fornece algumas informações com respeito a dados estatísticos da sessão de simulação. Através do comando *display*, o usuário tem acesso a algumas informações estatísticas, tais como:

- a transição mais vezes disparada de uma instância de módulo;
- a transição menos vezes disparada de uma instância de módulo;
- o número de vezes que uma certa transição foi disparada;
- a lista de transições de cada instância de módulo e o número de vezes que cada uma foi disparada.

Os comentários de qualificação serão apresentados na seção seguinte por se tratar de uma opção do compilador EC e não do simulador EDB.

3.3.2 - EC (Estelle-to-C Compiler)

O compilador Estelle [EC 92] é responsável pela geração de um código fonte em linguagem C a partir de uma especificação Estelle. O compilador é composto de duas ferramentas:

- um **tradutor**, que realiza uma análise completa - léxica, sintática e semântica - de uma especificação Estelle, obtendo como resultado uma **Forma Intermediária (FI)**;

- um **gerador de código C**, que a partir da Forma Intermediária de uma especificação Estelle retorna o código C desta especificação.

Cada uma dessas ferramentas pode ser usada separadamente (no caso do gerador de código é necessária a existência prévia de uma representação sob a Forma Intermediária), ou podem ser encadeadas de acordo com a opção escolhida pelo usuário.

Dentre as diversas opções oferecidas pelo compilador EC algumas delas merecem uma atenção especial. Uma dessas opções é a inserção de comentários de qualificação. Os comentários de qualificação do EC são semelhantes ao do ESTIM, com a diferença que no caso do EC os comentários são escritos na linguagem C, enquanto que no ESTIM são escritos na linguagem ML. A inserção de um caracter **$** após um delimitador de comentário indica que o mesmo é um comentário de qualificação. No EC não existe o chamado "ponto de parada" dentro de um comentário de qualificação, presente no ESTIM, já que para este fim já existem os observadores.

O compilador EC apresenta, na verdade, três tipos de comentários de qualificação. O primeiro deles é semelhante ao do ESTIM, ou seja, interpreta o comentário como sendo uma parte do código. Para tanto, o delimitador do comentário de qualificação deve ser seguido dos caracteres **C$**, como no seguinte exemplo:

(C printf("Mensagem"); *)*

Tal comentário será interpretado como uma parte de código C, da forma:

printf("Mensagem");

O segundo tipo (definido com **W$**) é interpretado como uma declaração em código C e deve ser utilizado para completar um tipo Estelle declarado como "..." e para completar uma constante Estelle declarada como "*any <TYPE>*", como nos exemplos a seguir:

type (W char * *) data_type = ... ;*

const (W 0 *) C = any INTEGER;*

Esse exemplos fornecem as seguintes declarações:

*typedef char * r<N>_data_type;*

#define r<N+1>_C 0

Pode-se notar no exemplo anterior que os identificadores originais *data_type* e *C* foram renomeados para *r<N>_data_type* e *r<N+1>_C*, respectivamente. Se o usuário desejar que os identificadores originais não sejam renomeados ele pode utilizar o terceiro tipo de comentário de qualificação, definido como **R$,** que evita a renomeação de qualquer identificador que vier após o comentário de qualificação (*R *). O exemplo a seguir mostra a sua utilização:

type (R *) (*W char * *) data_type = ... ;*

const (R *) (*W 0 *) C = any INTEGER;*

Com a inclusão do terceiro tipo de comentário de qualificação, os identificadores não são mais renomeados e ficam como no original:

*typedef char * data_type;*

#define C 0

Uma outra opção interessante é a possibilidade de traduzir um programa Pascal em um programa C correspondente. Esta opção possui algumas restrições com relação ao programa Pascal. Primeiramente ele deve ser "puro", isto é, deve estar de acordo com o subconjunto do Pascal ISO utilizado em Estelle e, além disso, algumas palavras-chave devem ser ajustadas ou acrescentadas.

Uma última opção importante possibilita a geração de um ***makefile***. Com esse *makefile* pode-se obter, de maneira automática, o módulo executável do código C gerado anteriormente. A partir daí, as especificações podem ser implementadas de maneira distribuída (como será visto no exemplo do capítulo 5).

3.4 - ECHIDNA

O projeto ECHIDNA [Jard 89, Jézéquel 91] tem como objetivo fornecer ferramentas para a obtenção de protótipos de algoritmos distribuídos em máquinas paralelas. O ambiente ECHIDNA possui um compilador Estelle para máquinas multi-processadores e um conjunto de ferramentas de *software* para observar os comportamentos do algoritmo distribuído sob experimentação. O conceito de experimentação foi visto na seção 2.2.2.4 e basicamente consiste na observação de um protótipo em um ambiente de simulação distribuído.

O modelo de máquina considerado no ECHIDNA compõe-se de sítios (processadores ou máquinas) que comunicam-se apenas por trocas de mensagens através de uma rede de comunicação ponto a ponto, sem perdas de mensagens. As mensagens são trocadas por meio de filas FIFO e possuem um tempo de transferência finito, porém imprevisível.

O ambiente ECHIDNA compreende as seguintes ferramentas:

- um compilador que traduz uma especificação formal Estelle em um conjunto de estruturas de dados (expressas em linguagem C), conjunto este que após compilado pelo compilador local é ligado a um núcleo de execução distribuído;
- núcleos de execução distribuídos para as máquinas Intel iPSC, FPS-T40, Telmat T-node, redes Sun (3 e 4), outras máquinas Unix e PC. A função do núcleo de execução distribuído é fornecer primitivas específicas de Estelle e Pascal. Ele realiza, dessa forma, a interface com o sistema distribuído subjacente controlando a alocação das tarefas, a comunicação distante e as entradas e saídas;
- um conjunto de ferramentas de software que permitem observar sem interferência o comportamento do algoritmo distribuído (construtor de tempo global, traços, registro de eventos e outros);
- um depurador simbólico (ao nível fonte Estelle) multi-janelas interativo (um processo por janela) disponível sob Suntools e X11/Motif.

3.4.1 - O Modelo Estelle do ECHIDNA

O modelo de Estelle adotado pelo ECHIDNA é um subconjunto (estático) da norma ISO. No ECHIDNA, uma especificação Estelle é a descrição do comportamento de um sistema fechado, ou seja, sem interação com o exterior. Um sistema consiste em um conjunto de

subsistemas executando-se em paralelo assincronamente, ou um conjunto de tarefas executando-se em paralelo (sendo que a este nível, o paralelismo pode ser síncrono ou assíncrono) [Jézéquel 91].

Neste subconjunto a parte dinâmica de Estelle não é considerada, ou seja, a arquitetura do sistema permanece inalterada após a sua inicialização. Duas classes de módulos são então consideradas: os módulos de estruturação (refinamentos), que não possuem a parte transição e desaparecem após a fase de configuração, e os processos reais (folhas) cujas transições não apresentam comandos dinâmicos como *init*, *connect*, *detach* e outros. Em conseqüência disso, alguns conceitos são simplificados: o compartilhamento de variáveis entre filhos de um módulo torna-se impossível e a distinção entre processos e atividades é irrelevante.

A última restrição semântica é quanto à cláusula *delay*, que não é implementada. Entretanto, existe um serviço de tempo global que pode ser utilizado para implementar a cláusula *delay*, se necessário. Existem ainda algumas restrições de ordem sintática: as instâncias de módulos devem ser inicializadas antes de serem conectadas, as filas comuns (*common queues*) não são permitidas, bem como o aninhamento de procedimentos e transições. Além destas, algumas particularidades são também rejeitadas como *any*, *all*, *exist*, *suchthat* e "...".

3.4.2 - A Simulação no ECHIDNA

A ferramenta de simulação do ECHIDNA pode ser utilizada ou no sistema de janelas Suntools, ou com o Motif que é uma interface padronizada para o sistema X Window (X11). Em ambos os casos, o ambiente de simulação é constituído de três tipos de janelas:

- a ***janela simulação***, que agrupa principalmente os parâmetros da simulação e os diferentes modos de simulação oferecidos ao usuário;
- as ***janelas de processos***, onde aparecem as informações dos processos (filas, estado corrente, variáveis) e as transições disparáveis;
- a ***janela histórico da simulação***, que registra de forma sucinta o histórico da simulação.

O ECHIDNA apresenta dois tipos de simulação: uma simulação passo a passo e uma simulação randômica.

- ***simulação passo a passo***: este tipo de simulação é semelhante ao das duas ferramentas vistas anteriormente, no qual o usuário indica a cada passo a transição a ser disparada;

- ***simulação randômica***: neste modo automático o usuário só retoma o controle da simulação ao final da mesma. O usuário indica o número de passos a serem executados e um número inteiro para o gerador de números (germe) que permite ao usuário reexecutar situações idênticas.

Dentro da simulação randômica o usuário possui ainda a opção de escolher entre três modos de simulação. Estes modos permitem representar os três tipos de paralelismo vistos na seção 2.4.2 que são:

- **modo síncrono**: neste modo um passo de simulação determina, para cada processo, uma transição escolhida aleatoriamente e dispara todas essas transições (uma por processo);

- **modo assíncrono**: neste modo um passo de simulação determina, para cada processo, uma transição escolhida aleatoriamente e dispara um certo número dessas transições (no máximo uma por processo);

- **modo seqüencial**: neste modo um passo de simulação determina uma transição escolhida aleatoriamente entre os processos e dispara essa transição.

3.5 - Análise Comparativa das Ferramentas

As ferramentas apresentam uma certa complementaridade entre si com relação a diversos aspectos. Primeiramente, o modelo Estelle suportado por cada ferramenta é diferente. O ESTIM suporta basicamente os recursos oferecidos pela versão Estelle* (permitindo a habilitação da cláusula *priority* e da prioridade pai/filho). O EDT suporta o padrão Estelle ISO, com mínimas restrições (as cláusulas *delay*, *any* e *priority* não são permitidas). O ECHIDNA suporta apenas um subconjunto estático de Estelle, ou seja, não permite a configuração dinâmica de instâncias de módulos através de transições que possuam comandos do tipo *init*, *connect*, *attach*, etc.

Um segundo aspecto diz respeito às etapas do processo de desenvolvimento cobertas por cada ferramenta. Quanto à *simulação*, as ferramentas ESTIM e EDT (neste caso o EDB) apresentam recursos semelhantes, como modos passo a passo e randômico, acesso ao estado

global da especificação e a dados estatísticos, além de executarem em um ambiente centralizado. O ECHIDNA, por sua vez, permite uma simulação distribuída da especificação em um ambiente gráfico multi-janelas; possui também os modos passo a passo e randômico, sendo que neste último pode-se selecionar o modo de simulação de acordo com o tipo de paralelismo desejado entre os processos: assíncrono, síncrono e seqüencial. Somente o ESTIM apresenta recursos de *verificação*, permitindo a geração do grafo de alcançabilidade e a posterior redução deste grafo baseada em relações de equivalência. Com relação à *implementação*, as ferramentas EDT (neste caso o EC) e ECHIDNA apresentam compiladores que geram o código em C de uma especificação Estelle; a diferença está na execução (distribuída) da especificação; no ECHIDNA, a distribuição dos módulos que executarão em máquinas diferentes é feita pela própria ferramenta; já no caso do EDT, o usuário deve separar estes módulos em especificações diferentes para então gerar os respectivos executáveis, os quais serão executados nas diferentes máquinas.

O último aspecto diz respeito às plataformas sobre as quais cada ferramenta pode ser utilizada. O ESTIM é utilizado sobre o sistema operacional UNIX em estações de trabalho SPARC@SUN. O EDT é utilizado sobre o UNIX System V (SPIX) em máquinas BULL-DPX 1000/2000, sobre o UNIX (HP-UX) em máquinas HP 9000/300 e sobre o UNIX BSD 4.2 em máquinas SUN. O ECHIDNA pode ser utilizado em máquinas Intel iPSC, FPS-T40, Telmat T-node, redes de estações SUN (3 e 4), outras máquinas UNIX e PC.

A tabela a seguir apresenta um resumo das principais características de cada ferramenta, situando-as também dentro do processo de desenvolvimento de aplicações distribuídas.

	VERSÃO DE ESTELLE SUPORTADA	VALIDAÇÃO		IMPLEMENTAÇÃO
		SIMULAÇÃO	VERIFICAÇÃO	
ESTIM	Estelle*	- Centralizada	- Análise por abstração: (PIPN / Aldebaran)	
EDT	Estelle ISO	- Centralizada (EDB)		Compilador Estelle para C (EC)
ECHIDNA	Subconjunto Estático de Estelle	- Distribuída - Ambiente multi-janelas		Compilador Estelle para C Núcleos de execução distribuídos: - Intel iPSC, FPS-T40, Telmat T-node, redes Sun (3 e 4) e outros

Tabela 3.1 - As Funcionalidades das Ferramentas Dentro do Processo de Desenvolvimento

3.5.1 - Seleção das Ferramentas Segundo as suas Funcionalidades

De acordo com a análise acima estabelecer-se-á um padrão para a utilização das ferramentas, levando-se em consideração as funcionalidades oferecidas por cada, no que diz respeito à utilização das mesmas para verificação, simulação e implementação:

- *Verificação*: como citado acima, a única das três ferramentas analisadas que oferece recursos para a verificação de especificações Estelle é a ferramenta ESTIM e, por esta razão, será utilizada para tal finalidade;

- *Simulação*: as três ferramentas vistas apresentam recursos de simulação. O ECHIDNA, apesar de poder realizar uma simulação distribuída da especificação, possui uma restrição considerável com relação ao modelo Estelle, que é a impossibilidade de se utilizar a configuração dinâmica de instâncias de módulos e, por este motivo, não será utilizado. As outras duas ferramentas, o ESTIM e o EDT (EDB) apresentam recursos semelhantes, estando a grande diferença no modelo Estelle suportado por cada uma. Sendo assim, a escolha de qual ferramenta utilizar para a simulação das especificações Estelle será feita adiante, quando da apresentação da metodologia (capítulo 4), na qual é colocada a questão dos diferentes modelos Estelle e definidas, então, as ferramentas adequadas para cada caso;

- *Implementação*: os recursos de implementação são oferecidos pelas ferramentas EDT (EC) e ECHIDNA. Devido à limitação do ECHIDNA mencionada acima, optou-se por utilizar a ferramenta EDT (EC) para a geração dos códigos de implementação das especificações Estelle.

INTERFACE GRÁFICA DE AUXÍLIO ÀS FERRAMENTAS AUTOMATIZADAS

Para facilitar a utilização das ferramentas vistas neste capítulo e visando uma futura integração das mesmas em um ambiente integrado de desenvolvimento de aplicações distribuídas, foi desenvolvida uma interface gráfica utilizando os recursos do *XView*. O *XView* é uma ferramenta para a construção de aplicações gráficas interativas que executam no sistema *X Window*. Esta interface é apresentada com detalhes no anexo I.

3.6 - Conclusão

Este capítulo ratificou a importância da utilização de ferramentas automatizadas de suporte às técnicas de descrição formal. Foram apresentadas as principais características de três ferramentas de suporte à técnica de descrição formal Estelle, disponíveis no LCMI: o ESTIM, o EDT e o ECHIDNA.

A apresentação de tais ferramentas foi realizada de modo a situar cada ferramenta dentro do processo de desenvolvimento de uma aplicação distribuída, ou seja, mostrando suas facilidades quanto à simulação, verificação e implementação. Neste sentido, pôde-se observar que as ferramentas possuem recursos complementares entre si, como visto na seção anterior.

A conseqüência direta disto é que nenhuma das ferramentas vistas neste capítulo é auto-suficiente para ser utilizada durante todo o desenvolvimento de alguma aplicação distribuída. Pelo contrário, elas devem ser utilizadas de maneira complementar, aproveitando ao máximo seus recursos, porém dentro das limitações impostas por cada uma. Dentro da metodologia que será apresentada no próximo capítulo este aspecto complementar das ferramentas voltará a ser abordado, na tentativa de utilizá-as nas diversas fases de desenvolvimento da melhor maneira possível.

CAPÍTULO 4

APRESENTAÇÃO DA METODOLOGIA

4.1 - Introdução

O objetivo final do processo de desenvolvimento de um sistema é obter uma "realização" do sistema, ou seja, uma instância concreta do sistema que preencha os requisitos do usuário [Pires 90]. No caso de uma aplicação distribuída, a obtenção de tal realização é uma tarefa árdua e a utilização de uma metodologia de desenvolvimento torna-se indispensável.

Este capítulo apresenta uma metodologia de utilização da técnica de descrição formal Estelle, procurando cobrir as diversas fases do desenvolvimento de um sistema enfocando em particular os aspectos de especificação formal, implementação e validação. Tal metodologia tem como base a técnica de refinamentos sucessivos e o conceito de abstração para definir os vários níveis de especificação do sistema.

São feitas primeiramente algumas considerações a respeito da técnica de refinamentos sucessivos. Em seguida, é discutida brevemente a questão da validação, quanto às técnicas de verificação e simulação. A metodologia será então apresentada, considerando para cada nível definido pela metodologia dois assuntos de interesse: primeiro, quais são os pontos de Estelle que devem ser considerados para o nível em questão e, segundo qual a metodologia de validação mais apropriada para tal nível.

4.2 - A Técnica de Refinamentos Sucessivos

Na técnica de *refinamentos sucessivos* (ainda chamada de *transformação de especificações*) o processo de desenvolvimento é conduzido em níveis. A cada nível uma especificação mais refinada é elaborada até se atingir o objetivo final que é a implementação do sistema. Esse processo baseia-se no conceito de abstração [Liskov 86], o qual consiste em

descrever, a cada nível de abstração, apenas os detalhes relevantes para o problema, abstraindo-se de detalhes considerados irrelevantes para tal nível. Dessa maneira, as dificuldades do problema não são encaradas todas de uma única vez e sim à medida que vão aparecendo em cada nível. Por exemplo, em um determinado nível de abstração do desenvolvimento do *software*, a especificação do seu comportamento pode ser relevante, enquanto que a maneira pela qual tal comportamento será implementado pode ser desconsiderada. Assim, possíveis problemas de implementação só serão tratados posteriormente quando muitos outros problemas já houverem sido solucionados.

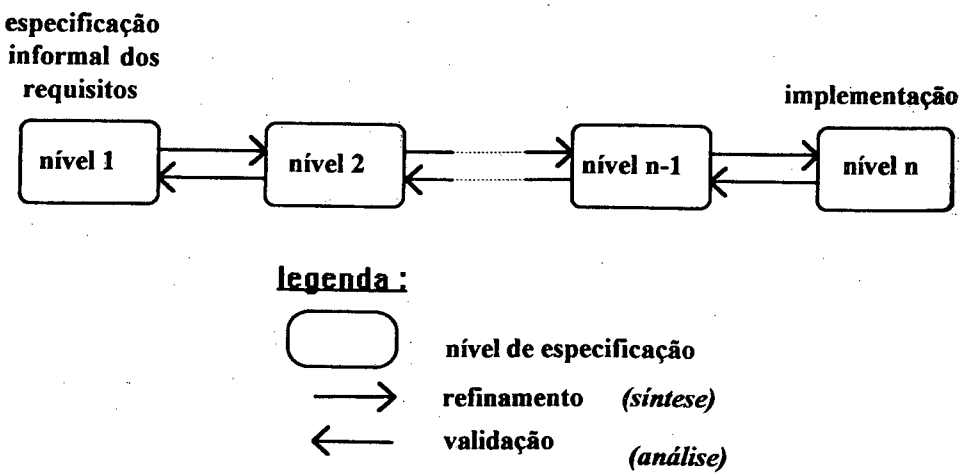


Figura 4.1 - O Processo de Refinamentos Sucessivos

Como conseqüência o processo de análise também é facilitado, pois após o término de cada nível a especificação pode (e deve) ser validada. Através desta validação os erros são detectados o quanto antes no processo de desenvolvimento. Como citado no capítulo 2, o custo de um erro é tanto menor quanto mais cedo ele for detectado, e pode-se imaginar quais as conseqüências de um erro ocorrido durante a análise dos requisitos e detectado somente após a implementação do sistema. A figura 4.1 dá uma visão do processo de desenvolvimento utilizando a técnica de refinamentos sucessivos onde cada nível representa uma especificação com um certo grau de abstração.

4.3 - Verificação X Simulação

Muitas vezes se questiona qual a técnica mais adequada para a validação de um aplicação distribuída, se a verificação ou se a simulação. Existem defensores das duas técnicas, porém uma justificativa convincente de que a verificação é muito superior à simulação ou vice-versa ainda não foi obtida. O fato é que nem a verificação nem a simulação podem ser

consideradas auto-suficientes para a tarefa de validação completa de uma aplicação distribuída pois cada uma possui algumas restrições relevantes. Aspectos sobre a verificação e a simulação foram abordados na seção 2.2.2.

Em [Groz 89] são colocadas algumas dificuldades e objeções quanto ao uso da simulação como forma de validação, sintetizadas a seguir:

- a simulação é fundamentada na aproximação dos resultados. Isto é incompatível com a verificação que fornece ou o resultado *correto* ou o resultado *incorreto*, ou seja, uma prova da correção;
- a simulação é limitada. Ao contrário da verificação, que pode fornecer os resultados *correto* ou *incorreto*, a simulação só pode permitir tirar conclusões a respeito de uma incorreção;
- a simulação é não-exaustiva. Enquanto a verificação é conclusiva sobre cada comportamento, a simulação não pode concluir sobre os comportamentos observados.

Diante disso, a verificação aparece a princípio como uma técnica incontestavelmente melhor que a simulação. Deve-se, entretanto, observar também as limitações da verificação. Os sistemas de prova são muitas vezes limitados no seu poder de descrição e de cálculo. Dessa forma, somente certos aspectos simplificados do sistema podem ser validados. A conseqüência dessa limitação é que a verificação acaba por não validar o sistema em si mas somente uma representação parcial [Jard 88].

Tem-se, então, por um lado a *verificação*, que permite tirar conclusões a respeito da correção de um sistema, desde que este seja representado em um nível de abstração bastante elevado; por outro lado a *simulação*, que não é conclusiva com relação à correção do sistema, mas que permite tratar uma descrição mais detalhada do sistema e, dessa forma, detectar erros inatingíveis pelas ferramentas de prova exaustiva.

As técnicas de validação devem ser vistas, portanto, sob uma óptica de *complementaridade* e não de concorrência. Essa complementaridade pode aparecer tanto em relação aos diferentes aspectos do mesmo sistema como em relação às várias fases de desenvolvimento do sistema. As diversas experiências realizadas [Phalippou 88] - inclusive sobre problemas reais - mostraram que diferentes técnicas não detectam os mesmos erros, evidenciando ainda mais a necessidade da utilização de maneira complementar das técnicas de validação.

4.4 - A Metodologia

A metodologia a ser apresentada é uma evolução da metodologia proposta inicialmente em [Mazzola 91]. A metodologia inicial consistia, basicamente, na definição de quatro níveis principais de especificação formal, tendo como base a estrutura mostrada na figura 4.1. O objetivo aqui é aprofundar a questão da validação de cada um desses níveis, considerando também a utilização das ferramentas vistas no capítulo 3.

Essa metodologia define, então, quatro níveis de especificação formal: uma *especificação funcional*, uma *especificação orientada a modelo*, uma *especificação detalhada* e uma *especificação orientada à implementação*. A definição destes quatro níveis justifica-se pelo fato de que, a cada passagem de um nível para o seguinte, um novo elemento de informação ao sistema é introduzido, ou seja, informações relativas à estrutura do sistema (especificação funcional); informações relativas ao comportamento dinâmico (especificação orientada a modelo); informações relativas aos dados e parâmetros (especificação detalhada); informações relativas ao ambiente de execução (especificação orientada à implementação). Entretanto, caso seja conveniente, cada um destes níveis pode ser expandido em subníveis, de modo que o elemento de informação do nível em questão também seja introduzido de maneira progressiva.

Cada um destes níveis de especificação será exposto enfocando-se dois aspectos principais. O primeiro deles trata da utilização dos mecanismos de Estelle, ou seja, quais os mecanismos considerados relevantes e quais os considerados irrelevantes em cada especificação. O segundo aspecto diz respeito ao processo de análise das especificações. Neste sentido, uma metodologia para a utilização das técnicas de validação e das ferramentas disponíveis também será apresentada.

Antes, porém, de apresentar os tipos de especificação citados acima, é conveniente falar da importância da etapa que precede o desenvolvimento de tais especificações, que é a **análise dos requisitos**. Grande parte dos erros encontrados nas etapas mais avançadas do desenvolvimento é resultado da negligência na realização desta etapa. O objetivo da análise dos requisitos é produzir, a partir da análise minuciosa das necessidades do usuário, uma primeira especificação do problema, dita **especificação informal dos requisitos**. Esta especificação será uma referência para todo o desenvolvimento do sistema e, portanto, deve merecer uma atenção especial por quem for executá-la. Uma dificuldade que normalmente ocorre no desenvolvimento de tal especificação é que muitas vezes o próprio usuário não sabe ao certo o que ele deseja que o sistema faça. Outra dificuldade surge quando, mesmo sendo o problema bem compreendido ele pode não ser descrito de forma suficientemente precisa para servir de base para as etapas posteriores [Liskov 86].

Assim, é importante que a especificação informal dos requisitos seja feita de maneira cautelosa e somente depois disso partir para a primeira especificação formal do problema, a **especificação funcional**. Tal especificação e as demais citadas anteriormente serão vistas nas seções seguintes.

4.4.1 - A Especificação Funcional

Na especificação funcional é definida a estrutura do sistema. Sua característica principal é que apenas o aspecto funcional deve ser especificado, evitando qualquer consideração a respeito da implementação dessas funções. Em termos da especificação Estelle, esta resume-se basicamente na definição dos seus módulos e submódulos, bem como na comunicação entre os mesmos.

4.4.1.1 - Mecanismos de Estelle(*) Utilizados

São utilizados a este nível basicamente os mecanismos de estruturação oferecidos pela linguagem. Dessa forma, devem ser definidos:

- a declaração dos canais, dos cabeçalhos de módulo e dos corpos de módulo associados. Tal declaração deve ser realizada também a nível dos submódulos. O conteúdo dos corpos de módulo e submódulo podem ser desconsiderados a este nível;

- as construções de instanciação estática, como a declaração de variáveis módulo (*modvar*), a inicialização de variáveis módulo (*init*) e o estabelecimento de ligações entre os módulos (*connect* e *attach*).

Um aspecto importante que deve ser considerado é quanto à definição dos atributos (*systemprocess/process* e *systemactivity/activity*) que devem ser associados à especificação e aos módulos. Ao se associar um atributo do tipo sistema (*systemprocess* ou *systemactivity*) a um módulo da especificação determina-se, em geral, uma separação física deste módulo em relação aos outros, o que a este nível é muitas vezes irrelevante. Segundo [Courtiat 87b], a utilização do atributo *systemactivity* associado ao nível da especificação e do atributo *activity* associado aos módulos permite representar de maneira abstrata a maioria das implementações que poderão ser derivadas a partir desta especificação. Diante disso, será adotada neste nível o modelo Estelle*.

4.4.1.2 - A Validação da Especificação Funcional

A validação a este nível consiste basicamente da análise estrutural estática da especificação no intuito de verificar a sua coerência. Esta análise pode ser realizada de duas formas dependendo da complexidade do sistema. Para exemplos mais simples a inspeção da especificação pode ser suficiente. Para o caso de sistemas maiores e mais complexos torna-se necessária a utilização de ferramentas automatizadas apropriadas. Tais ferramentas realizam normalmente uma análise completa do ponto de vista léxico, sintático e semântico (estático).

A análise estrutural da especificação permite detectar erros do tipo:

- não inicialização de uma variável módulo;
- ausência de conexão entre um módulo e outro;
- conexão incoerente entre módulos;

Poderiam ser utilizados aqui tanto o tradutor da ferramenta ESTIM quanto o tradutor da ferramenta EDT. Porém, como optou-se por utilizar o modelo Estelle* a este nível, torna-se conveniente a utilização do tradutor do ESTIM, que é próprio para este modelo.

4.4.2 - A Especificação Orientada a Modelo

Tendo-se validada a especificação funcional, o próximo objetivo é obter a especificação orientada a modelo. Esta especificação se aproxima de um modelo baseado em um sistema de transições e consiste no refinamento da especificação funcional, inserindo os mecanismos de Estelle que permitem definir o comportamento do sistema e distinguindo-se a parte de controle da parte de dados. Em uma especificação Estelle a parte de controle é modelada através das máquinas de estado finitas e, por conseguinte, o essencial a este nível é definir a parte transição dos corpos de módulo. A parte de dados da especificação é considerada irrelevante neste nível e deve ser introduzida posteriormente durante a especificação detalhada[1].

[1] Obviamente, a não inclusão da parte de dados não é uma imposição, mas deve ser evitada ao máximo e utilizada somente quando considerada imprescindível para a lógica da especificação.

4.4.2.1 - Mecanismos de Estelle(*) Utilizados

Com a introdução das máquinas de estado são introduzidas também as cláusulas Estelle vistas no capítulo 2. A este nível de abstração convém ainda utilizar a versão Estelle*. A conveniência do uso de Estelle* a este nível de especificação, onde almeja-se um bom poder de abstração, já teve sua justificativa abordada no capítulo 2 (cf. tópico 2.4.7). Ademais, a possibilidade de utilizar as facilidades de verificação da ferramenta ESTIM, restrita à versão Estelle*, parece também de grande interesse e não deve ser colocada de lado. Considerando então a utilização de Estelle*, tanto as cláusulas de comunicação do tipo *when* quanto as cláusulas de sincronização por *rendez-vous* do tipo *p!interação* e *p?interação* podem ser empregadas, permitindo explicitar os diferentes tipos de comunicação que podem ocorrer entre as instâncias de módulo.

As demais cláusulas também são empregadas a este nível. A cláusula *provided* pode ser utilizada para reduzir o não-determinismo existente em alguns módulos. As cláusulas *from* e *to* definem os estados de controle das máquinas de estado, descrevendo assim o comportamento de cada módulo. A cláusula *output* é empregada na parte **ação** das transições para explicitar uma troca de mensagens assíncrona entre as instâncias de módulo. Como a parte de dados é considerada irrelevante a este nível, as instruções Pascal, bem como as chamadas a funções e procedimentos devem ser evitadas e só utilizadas quando forem consideradas indispensáveis.

Com relação aos canais declarados na especificação funcional, devem ser indicados neste momento os tipos de interações que podem ser trocados através destes canais, porém se houverem parâmetros associados às interações eles podem ser desconsiderados por enquanto.

4.4.2.2 - A Validação da Especificação Orientada a Modelo

Na metodologia de validação da especificação orientada a modelo, além da análise estática deve-se realizar também uma análise dinâmica do sistema. Objetiva-se, com isso, validar também o comportamento dinâmico do sistema, que foi definido pela inclusão das máquinas de estados finitas.

O nível de abstração da especificação orientada a modelo permite, na maioria dos casos, a utilização da verificação como técnica de validação. O método clássico de verificação de sistemas de transições (como é o caso de Estelle e Estelle*) é a análise de alcançabilidade baseada na construção e análise do *grafo de alcançabilidade*, também conhecido como espaço

de estados global, onde é realizada uma exploração exaustiva de todas as possíveis interações entre um conjunto de processos comunicantes modelados como sistemas de transição.

Através desse método, diversas propriedades de *segurança* (*safety properties*)[2] podem ser checadas. Exemplos desta classe de propriedades são: ausência de *deadlock*, inexistência de eventos de comunicação não especificados, ausência de ciclos (*loops*) indesejáveis, isto é, ciclos infinitos nos quais nenhum progresso é feito. A grande limitação deste método é o problema da explosão combinatória do grafo. Diversas técnicas têm sido propostas para tentar minimizar tal problema, como seqüencias canônicas, projeção, regras de escolha de transições, técnicas estocásticas, técnicas probabilísticas e outras [Pehrson 90].

A TÉCNICA DE ANÁLISE POR ABSTRAÇÃO

Esta técnica de verificação consiste em aplicar sobre o grafo de alcançabilidade obtido[3] técnicas de redução baseadas em *relações de equivalência*, tendo como resultado da redução um *autômato quociente* que fornece uma visão abstrata do modelo. Para a geração do autômato deve ser definida sobre a arquitetura da especificação uma *partição*, tendo de um lado o *mundo observado* e do outro o *ambiente externo* (figura 4.2). Dessa forma, todos os eventos que ocorrem na fronteira da partição, ou seja, as interações entre o mundo observado e o ambiente externo são denominados **eventos observáveis**, sendo os demais considerados **eventos internos**. A distinção entre os dois tipos de eventos aparece a nível do autômato quociente, dependendo da relação de equivalência escolhida [Courtiat 91].

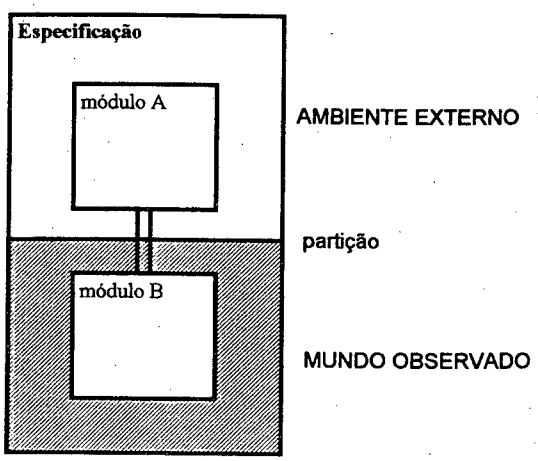


Figura 4.2 - Ilustração da Partição

[2] Propriedades de segurança são propriedades que asseguram que nenhum evento indesejável irá ocorrer.
[3] No ESTIM é gerado o grafo completo do espaço de estados e, dessa forma, o problema de explosão combinatória não é resolvido. Todavia, com a redução do grafo através da técnica de análise por abstração, o trabalho de análise é facilitado consideravelmente.

O objetivo do emprego de relações de equivalência é poder comparar duas especificações e provar que elas apresentam alguma relação entre si, ou seja, que uma apresenta as mesmas propriedades especificadas pela outra [Ernberg 91]. Estas relações são muito utilizadas para verificar a conformidade entre a "*especificação i*" e a "*especificação i-1*". Para que dois sistemas sejam equivalentes é necessário que exista uma *relação de bissimulação* entre os conjuntos de estados de ambos. A relação de bissimulação é definida da seguinte maneira:

Sejam G_1 e G_2 dois sistemas de transições de estados finitos com o conjunto de estados S_1 e S_2 respectivamente, tal que $\mathbf{S} = \mathbf{S}_1 \cup \mathbf{S}_2$. Uma relação de bissimulação entre G_1 e G_2 é uma relação $\mathbf{R} \subseteq \mathbf{S} \times \mathbf{S}$ tal que $< m, n > \in \mathbf{R}$ implica

1. se $m \xrightarrow{a} m'$ então $\exists n' : n \xrightarrow{a} n'$ e $< m', n' > \in \mathbf{R}$, e

2. se $n \xrightarrow{a} n'$ então $\exists m' : m \xrightarrow{a} m'$ e $< m', n' > \in \mathbf{R}$

Assim, dois sistemas de transições G_1 e G_2 são *fortemente equivalentes* se existe uma bissimulação relacionando os estados iniciais de ambos os sistemas. Isto significa que cada transição em um estado m de G_1 pode ser simulado por uma transição no estado n de G_2 e vice-versa.

A *equivalência forte* é a mais restritiva das equivalências pois considera igualmente importantes todas as transições, incluindo as transições internas. Tal equivalência é geralmente muito forte para verificação e dificilmente utilizada [Ernberg 91]. Outras relações de equivalência podem ser obtidas redefinindo (sutilmente) a noção de bissimulação e/ou por transformações nos grafos originais, como: a equivalência observacional, a equivalência de teste ou de aceitação e a equivalência de traço ou de linguagem.

A *equivalência observacional* é mais fraca que a anterior pois diferencia os eventos externos (observáveis) dos eventos internos. Para defini-la é necessário redefinir a noção de bissimulação apresentada acima.

Dados os estados m e m', escreve-se $m \overset{\alpha}{\Rightarrow} m'$ se m pode realizar o evento α de entrada ou de saída precedido ou sucedido por um número finito (possivelmente zero) de eventos internos τ e terminar em um estado m'.

Escreve-se $m \overset{\varepsilon}{\Rightarrow} m'$ se m pode realizar zero ou mais eventos internos τ e terminar em um estado m'. Redefinindo-se, então, a definição de bissimulação:

Sejam G_1 e G_2 dois sistemas de transições de estados finitos com o conjunto de estados S_1 e S_2 respectivamente, tal que $\mathbf{S} = \mathbf{S}_1 \cup \mathbf{S}_2$. Uma relação de bissimulação *fraca* entre G_1 e G_2 é uma relação $\mathbf{R} \subseteq \mathbf{S} \times \mathbf{S}$ tal que $< m, n > \in \mathbf{R}$ implica

1. se $m \xrightarrow{a} m'$ então $\exists n' : m \overset{\$}{\Rightarrow} m'$ e $< m', n' > \in \mathbf{R}$, e

2. se $n \xrightarrow{a} n'$ então $\exists m' : m \overset{\$}{\Rightarrow} m'$ e $< m', n' > \in \mathbf{R}$

A figura 4.3 a seguir ilustra um exemplo de dois autômatos com equivalência observacional.

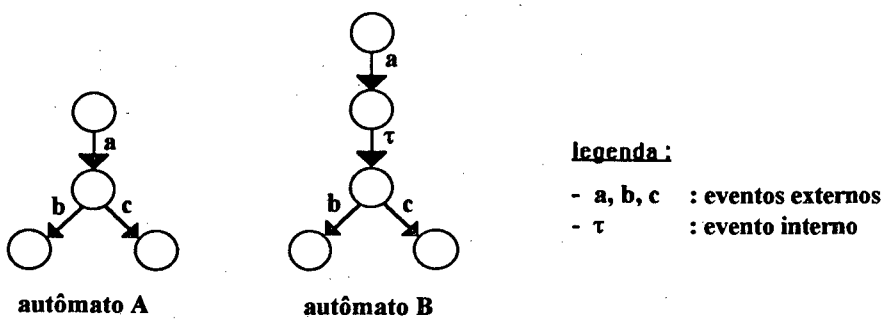


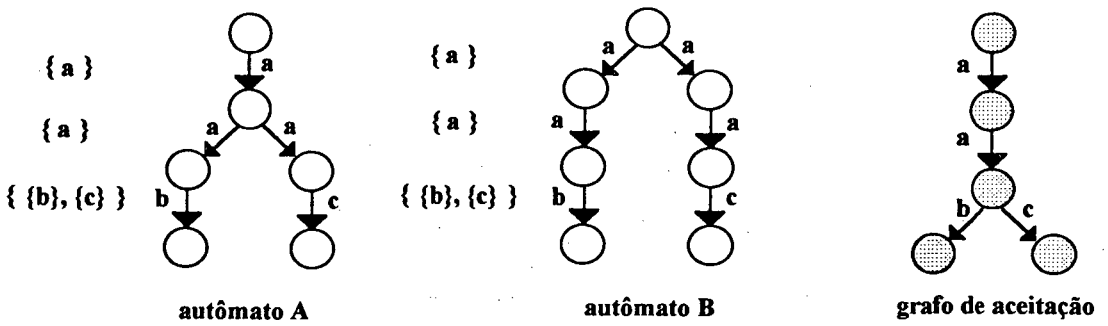
Figura 4.3 - Equivalência Observacional

A *equivalência de teste* ou *de aceitação* é mais fraca que as duas anteriores. As equivalências forte e observacional distinguem os sistemas de transição nos pontos exatos onde as escolhas não-deterministas são feitas, durante as suas execuções. A equivalência de teste, por sua vez, faz a distinção entre sistemas de transição que possuem um diferente grau de não-determinismo, porém é mais insensível com relação a onde tais escolhas não-deterministas são feitas [Ernberg 91]. É introduzida a noção de *grafo de aceitação*, que é um grafo determinista cujos estados contêm informações sobre *deadlocks* potenciais, "codificados" como *conjuntos de aceitação*. O conjunto de aceitação *n.acc* de um estado *n* representa um conjunto de conjuntos de transições: cada membro de *n.acc* é um conjunto de transições que representa a obrigação de se poder continuar do estado *n* com qualquer transição do conjunto.

Assim, para que dois sistemas de transição possuam uma equivalência de teste, deve existir uma bissimulação entre os seus grafos de aceitação. Todavia, requer-se ainda uma terceira condição para se definir a relação de bissimulação de teste:

3. Os conjuntos de aceitação *m.acc* e *n.acc* devem ser *compatíveis*, onde compatível significa que cada elemento de *m.acc* é um *superconjunto* de um elemento em *n.acc* e vice-versa. Um conjunto A é um superconjunto de um conjunto B se todos os membros de B são também membros de A.

A figura 4.4 mostra os autômatos A e B, com seus respectivos conjuntos de aceitação entre chaves e o grafo de aceitação referente a ambos.

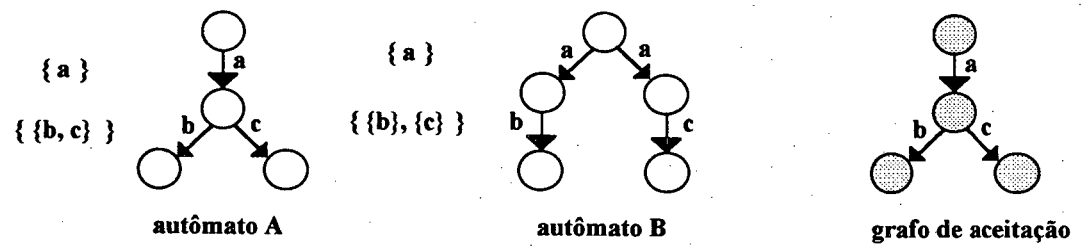


legenda :

- **a, b, c : eventos externos**
- **{ } : conjuntos de aceitação**

Figura 4.4 - Equivalência de Teste

A última e também a mais fraca das relações de equivalência é a *equivalência de traço* ou *de linguagem*. Tal equivalência não distingue entre diferentes graus de não-determinismo além de ser insensível a propriedades de *deadlock*. Para que dois sistemas de transição possuam uma equivalência de traço basta que exista uma bissimulação (fraca) entre os seus grafos de aceitação, desconsiderando os seus conjuntos de aceitação. A figura 4.5 ilustra um exemplo onde os dois autômatos apresentam equivalência de traço. Pode-se perceber que, apesar de possuírem o mesmo grafo de aceitação, seus conjuntos de aceitação são diferentes e, portanto, não possuem equivalência de teste.



legenda :

- **a, b, c : eventos externos**
- **{ } : conjuntos de aceitação**

Figura 4.5 - Equivalência de Traço

A metodologia de verificação utilizando as relações de equivalência e que normalmente é empregada baseia-se na análise dos autômatos quocientes gerados a partir de cada relação de equivalência, partindo-se da relação mais fraca para a mais forte. Sendo assim, inicia-se a verificação pela análise do autômato gerado pela equivalência de traço. Na análise deste autômato, normalmente simples, pode-se observar se o autômato realiza as funções esperadas. Como tal análise não permite obter informações sobre possíveis *deadlocks*, a verificação deve então prosseguir com a geração e análise dos demais autômatos, os quais são mais sensíveis ao não-determinismo e possibilitam a detecção de *deadlocks* [Mazzola 91].

A Simulação na Especificação Orientada a Modelo

A simulação deve ser utilizada como um complemento à verificação, no caso em que um erro é detectado. Dessa forma, uma simulação interativa (passo a passo) mostra-se bem conveniente para pesquisar a causa do erro, pois além de possibilitar o acesso aos objetos da especificação permite ainda explorar os possíveis caminhos que deram origem ao erro.

Com relação à utilização das ferramentas de suporte à Estelle, faz-se aqui uma referência à análise feita no capítulo 3 (seção 3.5.1), onde definiu-se um padrão para a utilização das mesmas. Com relação à verificação será utilizada a ferramenta ESTIM, já que é a única disponível para esta finalidade; com relação à simulação, não se havia tomado nenhuma posição, considerando-se que esta dependia da definição do modelo Estelle a ser utilizado. Na especificação orientada a modelo optou-se pela utilização do modelo Estelle* e, em vista disso, deve ser utilizada a ferramenta ESTIM também para a simulação da especificação.

4.4.3 - A Especificação Detalhada

A especificação detalhada é o último nível de refinamento antes de se partir para uma implementação. Seu objetivo, portanto, é explicitar todo o funcionamento do sistema, caracterizado principalmente pela inclusão de toda a parte de dados, desconsiderada até então.

Neste ponto, surge uma questão importante. Existe a necessidade de conversão da especificação, que até agora foi descrita em Estelle*, para uma especificação descrita sob o padrão Estelle ISO? A resposta vai depender do objetivo final da especificação. Se o objetivo é a obtenção de uma descrição formal e completa do sistema mas sem quaisquer considerações a respeito de uma implementação específica, tal conversão não é necessária já que a vantagem do uso de Estelle* está justamente na sua abstração quanto às características de implementação.

Entretanto, se o objetivo principal é a implementação do sistema, as características de implementação de Estelle podem agora ser benéficas, uma vez que o ambiente de implementação já deve ser conhecido. Particularmente, no caso onde se deseja obter um código de implementação a partir de uma ferramenta automatizada, deve ser levado em conta também a disponibilidade de uma ferramenta de geração de código que seja compatível com Estelle ou Estelle* ou ambas. Admitir-se-á doravante o uso do padrão Estelle ISO, considerando-se o interesse em relação à etapa de implementação do sistema e a disponibilidade de uma ferramenta de geração de código restrita a Estelle ISO.

4.4.3.1 - Implicações da Transformação do Modelo Estelle* em Estelle ISO

A conversão do modelo Estelle* para Estelle ISO implica em modificações nos mecanismos de comunicação. Deve-se transformar todas as comunicações através do mecanismo de *rendez-vous* em filas FIFO, definindo-se também a política de fila (*individual queue* ou *commom queue*) que se julgar adequada. Quando se faz esta conversão altera-se, de alguma forma, o comportamento da especificação, visto que existe uma diferença sensível entre os dois mecanismos de comunicação. Ao se transformar uma comunicação através de um mecanismo síncrono, como o *rendez-vous*, em uma comunicação através de um mecanismo assíncrono, como as filas FIFO (no caso particular de Estelle estas filas são ilimitadas), eleva-se consideravelmente o grau de não-determinismo da especificação. Em vista disso, essa transformação deve ser realizada cautelosamente, pois poderão aparecer novas situações (algumas indesejáveis), as quais não ocorriam antes devido à limitação imposta pela utilização do mecanismo de *rendez-vous*.

Algumas implicações da transformação do *rendez-vous* em filas FIFO podem ser vistas no exemplo que se segue. Tomam-se como exemplo três módulos, um módulo *A*, um módulo *B* e um módulo *meio de transmissão* (a partir de agora chamado de *meio*, por simplificação). Considera-se, então, que o módulo *A* pode enviar mensagens (uma a uma) do tipo *msgA* ao módulo *B* através do módulo *meio*; de maneira semelhante, o módulo *B* pode enviar mensagens (uma a uma) do tipo *msgB* ao módulo *A*, também através do módulo *meio* (figura 4.6).

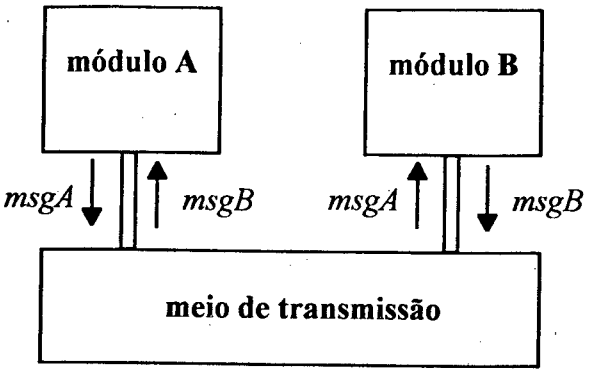


Figura 4.6 - Exemplo

Supondo, primeiramente, o exemplo acima sendo especificado em Estelle*. Os módulos *A* e *B* possuem duas transições cada um, sendo uma de envio e a outra de recepção de mensagens. As transições do módulo *A* são mostradas a seguir:

```
trans                (* transição de envio de mensagens do tipo msgA *)
    ipA!msgA
        begin
        end;

trans                (* transição de recepção de mensagens do tipo msgB *)
    ipA?msgB
        begin
        end;
```

As transições do módulo *B* são semelhantes. O módulo *meio* por sua vez possui quatro transições, as quais são responsáveis por receber uma mensagem de um módulo e enviá-la ao outro módulo. Estas transições são as seguintes:

```
trans
    from RECEBE_MSG to ENVIA_MSG_A
    ipA?msgA
        begin
        end;

trans
    from ENVIA_MSG_A to RECEBE_MSG
    ipB!msgA
        begin
        end;

trans
    from RECEBE_MENS to ENVIA_MSG_B
    ipB?msgB
        begin
        end;
```

```
trans
    from ENVIA_MSG_B to RECEBE_MSG
    ipA!msgB
        begin
        end;
```

A análise desta especificação, através da geração do seu grafo de alcançabilidade, permite obter algumas conclusões. Pode-se observar que o comportamento desta especificação é bastante simples (3 estados e 4 arcos), pois existem apenas dois caminhos possíveis: ou o módulo *A* envia uma mensagem ao módulo *B*, ou vice-versa. Este envio é feito de maneira "atômica", ou seja, quando um módulo envia uma mensagem ao *meio* esta mensagem será enviada em seguida ao módulo destino. Este fato impede, por exemplo, que um módulo envie uma próxima mensagem antes que o outro módulo tenha recebido a anterior. Este comportamento, apesar de não ficar explícito na especificação informal, acabou sendo obtido pela simplificação introduzida pelo uso do mecanismo de *rendez-vous*.

O que ocorre, então, quando é feita a transformação da especificação Estelle* em Estelle ISO (mais especificamente com relação à transformação *rendez-vous* → filas FIFO)? Como citado acima esta transformação deve ser cautelosa, já que comportamentos adversos poderão surgir. Tomando-se novamente o exemplo anterior, considera-se um primeiro caso no qual a comunicação por *rendez-vous* é simplesmente trocada pela comunicação através de filas FIFO. Neste caso, as transições de envio dos módulos ficariam da seguinte forma:

```
trans                (* transição de envio de mensagens do módulo A *)
  begin
    output ipA.msgA
  end;

trans                (* transição de envio de mensagens do módulo B *)
  begin
    output ipB.msgB
  end;
```

Como pode-se observar pelas transições acima, é possível o envio consecutivo de um número qualquer de mensagens, por qualquer um dos módulos, antes que o outro módulo receba alguma mensagem. Este caso seria o mais geral possível e levaria a uma explosão combinatória do grafo, devido à capacidade ilimitada das filas FIFO. Nota-se, com isso, que outras possibilidades que são "mascaradas" pela utilização do *rendez-vous*, aparecem com a utilização das filas FIFO.

Considera-se, como um segundo caso, que o comportamento obtido com a especificação Estelle*, ou seja, que um módulo só envie uma próxima mensagem após o outro

módulo receber a mensagem anterior, fosse realmente o comportamento desejado. Neste caso, a simples troca do *rendez-vous* pelas filas FIFO não é suficiente, pois como visto acima não existe a limitação quanto ao envio de mensagens pelos módulos. Uma maneira de se obter tal comportamento seria o envio de um reconhecimento (*ACK*) quando um módulo recebesse uma mensagem. Isto pode ser feito pela introdução de estados globais nos módulos *A* e *B*, de modo que, quando o módulo envia uma mensagem, ele passa a um estado de espera e só retorna ao estado de envio ao receber o reconhecimento de que a mensagem enviada foi recebida. Obviamente, quando se introduz estes estados globais, deve-se prever todos os possíveis eventos que podem ocorrer. Para este caso deve-se prever, por exemplo, que um módulo, mesmo estando em um estado de espera de reconhecimento, pode receber uma mensagem do outro módulo, antes de receber o *ACK*; neste caso, o módulo deve consumir a mensagem, enviar um reconhecimento ao outro módulo de que recebeu a mensagem, e continuar no estado de espera do *ACK*. A figura 4.7 mostra os estados globais do módulo *A*.

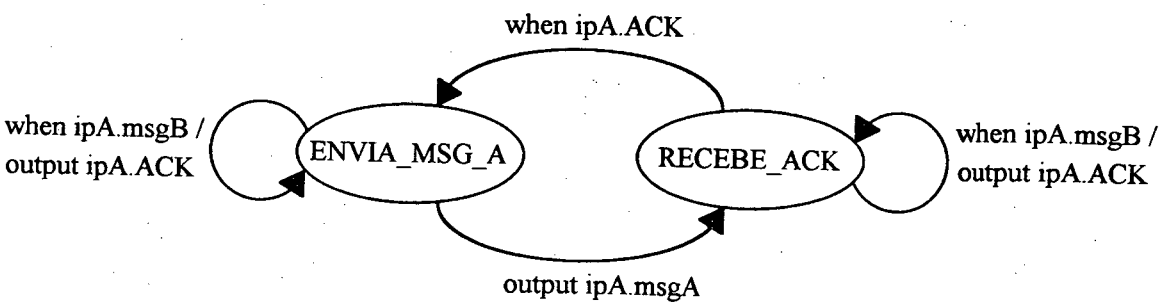


Figura 4.7 - Estados Globais do Módulo A do Exemplo

Com a inclusão destes estados obtém-se o comportamento desejado; mesmo assim o grafo de alcançabilidade cresce bastante (29 estados e 50 arcos) em relação ao grafo obtido para a especificação Estelle*. Isto ocorre porque, na comunicação através de filas FIFO, um módulo pode não estar habilitado para receber uma interação de outro módulo, porém ele não consegue impedir tal módulo de enviar uma interação, aumentado dessa forma o número de possibilidades da especificação.

Um outro aspecto que se coloca está relacionado com as cláusulas de envio e recepção; em uma especificação Estelle* não é permitido, em uma mesma transição, o uso simultâneo de uma cláusula de envio e de uma cláusula de recepção, o que é possível em Estelle ISO. Assim, a especificação Estelle* apresenta alguns estados globais que tornam-se desnecessários quando se transforma a especificação para Estelle ISO; estes estados devem, então, ser eliminados de modo a reduzir o grau de não-determinismo da especificação.

Este exemplo pôde mostrar que a transformação de *rendez-vous* para filas FIFO não é tão simples e que deve ser realizada de maneira cautelosa, caso contrário pode conduzir a situações indesejáveis. É importante, ao se realizar tal transformação, estar sempre ciente do comportamento que se deseja obter do sistema em questão.

4.4.3.2 - Mecanismos de Estelle Utilizados

A menos que se tenha um conhecimento da implementação final do sistema e se deseje alterar os atributos (utilizando também *systemprocess* e *process*), é preferível manter os atributos *systemactivity* e *activity* definidos anteriormente.

Em relação à parte de dados deve ser incluída a parte seqüencial da especificação, ou seja, as instruções, os procedimentos e as funções Pascal. Como conseqüência, os tipos Pascal devem ser definidos e todas as variáveis inicializadas. Por fim, os parâmetros associados às interações, que haviam sido desconsiderados até o momento, devem agora ser declarados.

4.4.3.3 - A Validação da Especificação Detalhada

A grande complexidade desta especificação devido ao elevado grau de não-determinismo inviabiliza, na grande maioria dos casos, a utilização da verificação como técnica de validação. Desse modo, simulação aparece como a solução mais adequada para este nível de detalhamento da especificação. Como visto acima, a transformação para filas FIFO pode alterar o comportamento da especificação e, por este motivo, a validação a este nível deve procurar verificar se o comportamento da especificação se mantém como o esperado.

A metodologia de simulação adotada consiste em duas etapas. A primeira é uma etapa de simulações rápidas destinada a encontrar erros mais triviais (como *deadlocks* que possam ter sido introduzidos com a transformação da especificação), através de simulações randômicas de curta duração; pode-se também realizar o traço de alguns eventos, como o traço de mensagens específicas, utilizando uma simulação interativa (passo a passo). As ferramentas de simulação normalmente oferecem ambos os modos (randômico e interativo).

A segunda etapa caracteriza-se por simulações mais longas que visam aumentar a confiabilidade do sistema. Evidentemente quanto mais intensa esta simulação maior será a sua

confiabilidade, já que uma simulação exaustiva (cobrindo todos os estados possíveis do sistema) não pode ser atingida.

Com relação às ferramentas de simulação a serem utilizadas, faz-se novamente uma referência à análise feita na seção 3.5.1 do capítulo 3. A escolha é feita novamente considerando o modelo Estelle utilizado neste nível de especificação. Considerando então o uso de Estelle ISO a ferramenta utilizada para a simulação neste nível é o simulador EDB do pacote EDT.

4.4.4 - A Especificação Orientada à Implementação

Neste último nível de especificação é importante um pleno conhecimento com relação à organização física do sistema. Deve-se definir então quais módulos executarão em máquinas diferentes, pois tal distinção levará a possíveis alterações dos atributos definidos anteriormente.

A especificação Estelle deverá sofrer então algumas modificações:

- o atributo de sistema associado ao nível mais alto da especificação (raiz) deve ser excluído. Um atributo de sistema deverá ser associado a todo módulo cujos submódulos (se houverem) forem comuns a uma mesma máquina;
- se alguns módulos devem ser alocados a uma mesma máquina mas não existe um módulo pai envolvendo-os, tal módulo deve ser criado com um atributo do tipo sistema.

A figura 4.8 a seguir mostra um exemplo destes procedimentos onde a especificação (a) é transformada na especificação (b). A primeira alteração é a eliminação do atributo sistema ao nível da raiz da especificação. Os módulos hachurados representam módulos de sistema e os módulos em branco representam atividades ou processos. Considerando, neste exemplo, que os submódulos A1 e A2 devem fazer parte de um mesmo sistema, o módulo A da especificação (b) deverá possuir um atributo de sistema; de modo semelhante, se os módulos B1 e B2 devem também fazer parte de um mesmo sistema e não existe nenhum módulo envolvendo-os, tal módulo deverá ser criado e atribuído como sistema.

O objetivo dessas primeiras modificações é definir todos os sistemas da especificação, já que a cada um deles irá corresponder uma unidade de implementação. Assim sendo, os códigos para a posterior implementação deverão ser gerados individualmente para cada sistema.

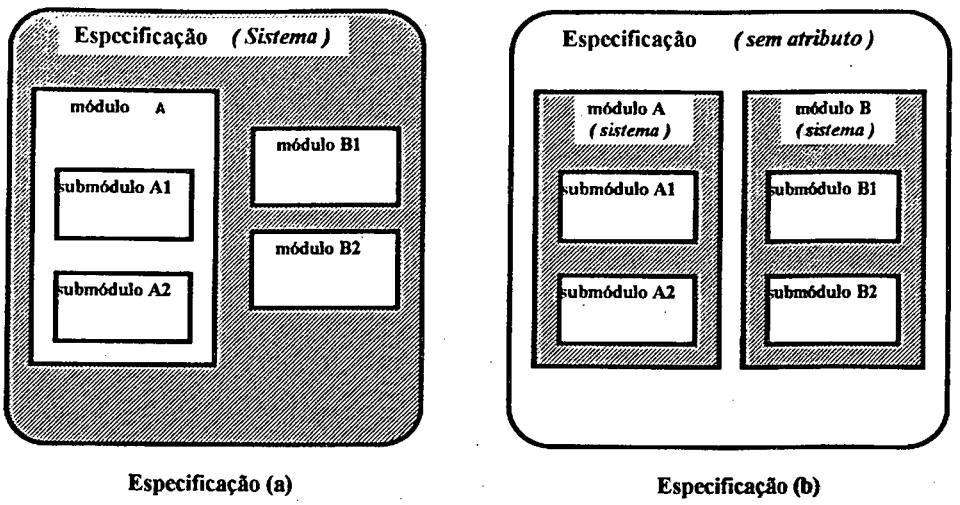


Figura 4.8 - Transformações na especificação Estelle

4.4.4.1 - Obtenção do Código de Implementação

A implementação é a última etapa do desenvolvimento da aplicação distribuída e concentra-se principalmente na obtenção do código de implementação. Com respeito à geração do código é novamente recomendada a utilização de uma ferramenta automatizada, pois com isso pode-se obter uma boa parte de código automaticamente. As vantagens da geração (semi-) automática de código serão vistas na seqüência.

GERAÇÃO SEMI-AUTOMÁTICA DO CÓDIGO DE IMPLEMENTAÇÃO

A especificação formal de um sistema envolve duas partes, uma parte **independente da máquina** e outra parte **dependente da máquina**. A parte independente da máquina inclui as transições de estado e as ações associadas das máquinas de estado finitas e pode ser completamente especificada. A parte dependente da máquina, por sua vez, inclui mecanismos de geração e detecção de eventos, meios de comunicação, gerenciamento de memória e outros itens que não podem ser completamente especificados por estarem relacionados com a arquitetura da máquina e com o sistema operacional.

Por diversas vezes no presente trabalho justificou-se a importância da utilização das técnicas de descrição formal. Dentre as suas vantagens destaca-se a possibilidade do emprego de compiladores para a produção de um código apropriado para a implementação [Alcântara 92, Ansart 86, Vuong 88]. Estes compiladores realizam uma checagem sintática e semântica da especificação traduzindo-a em um código fonte de alguma linguagem de alto nível como, C ou Pascal. A porção gerada automaticamente é a parte independente da máquina que para o caso

dos protocolos de comunicação, por exemplo, representa de 50 a 70% da implementação completa [Sidhu 90]. Já a parte dependente da máquina deve ser codificada a mão, porém muitas vezes o seu código ou parte dele pode ser reutilizado.

A utilização de uma ferramenta de geração automática do código, mesmo que gerando apenas uma parte dele, apresenta uma série de vantagens sobre as implementações totalmente codificadas à mão [Sidhu 90]:

- conformidade da porção gerada automaticamente com a especificação, já que a geração é realizada por um compilador;
- facilidade de manutenção, pois a cada alteração da especificação uma nova implementação pode ser gerada pelo compilador;
- aumento da confiabilidade da implementação, pois a parte gerada automaticamente não está sujeita a erros de codificação do programador;
- redução no custo de produção da implementação, visto que boa parte do código final é gerada rapidamente;
- o tamanho e o *throughput* das implementações geradas semi-automaticamente é, em alguns casos, comparável com as implementações da mesma especificação desenvolvidas manualmente.

4.4.4.2 - Modificações na Especificação Orientada à Implementação

A seguir são sugeridas algumas modificações a serem realizadas nesta especificação antes da geração do código de implementação. Tais modificações referem-se ao aspecto da comunicação entre as especificações, principalmente com relação à inclusão das chamadas de primitivas de comunicação. Tomando-se novamente o exemplo da figura 4.8 o primeiro passo é separar a especificação (b) em várias outras, de modo que cada uma represente um sistema a ser implementado separadamente (figura 4.9).

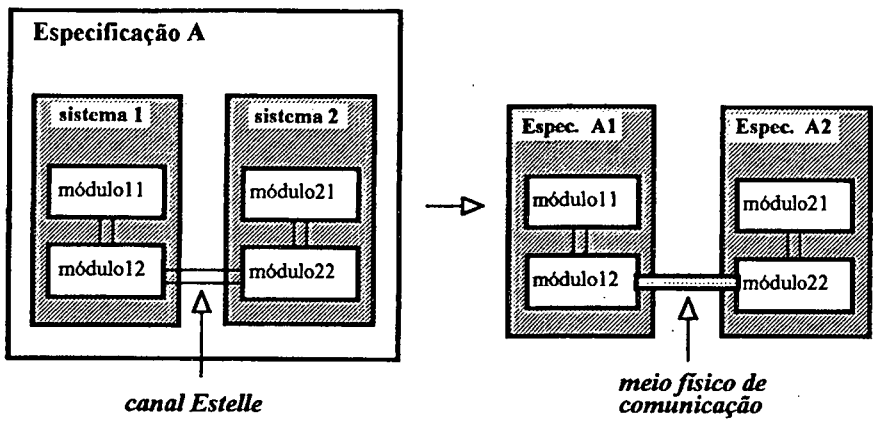


Figura 4.9 - A Explosão da Especificação Estelle em Várias Especificações

Antes de apresentar as modificações e para melhor compreendê-las, devem ser relembradas algumas regras de especificação em Estelle [Saqui 92]:

- a norma [Estelle 88] define uma especificação Estelle como um mundo fechado; em particular, a raiz da especificação não possui nenhum ponto de interação;
- a raiz da especificação tem normalmente a tarefa de criar a arquitetura estática de módulos; conseqüentemente, a raiz é considerada inativa, isto é, sua parte transição é vazia.

Segundo [Saqui 92], para se realizar a comunicação entre módulos localizados em diferentes sistemas não basta simplesmente trocar a comunicação virtual (canal Estelle e pontos de interação externos) por uma chamada às primitivas de comunicação. É preferível concentrar a comunicação com o exterior ao nível da raiz da especificação. Dessa forma, os módulos **módulo12** e **módulo22** do exemplo anterior (figura 4.9) são mantidos intactos; um ponto de interação interno é criado ao nível das raízes de cada especificação e conectados aos respectivos pontos de interação externos, como mostra a figura 4.10.

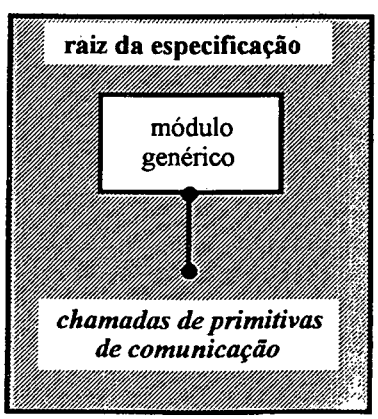


Figura 4.10 - A Arquitetura Geral de Implementação

Considerando-se então a comunicação como sendo realizada ao nível da raiz da especificação, devem ser acrescidas a ela:

- a declaração de pontos de interação internos de mesmo canal e papéis opostos àqueles dos pontos de interação externos dos módulos;
- na parte "*initialize*" da especificação, a conexão entre estes novos pontos de interação e aqueles dos módulos;
- as transições para a comunicação com o ambiente da especificação.

Tipicamente, estas transições tomam a forma ilustrada abaixo:

```
when ponto_de_interação_interno.ENVIO_DE_DADO
        begin
                "chamada da primitiva de envio de dados"
        end;

provided RECEPÇÃO_SOBRE_O_MECANISMO_DE_COMUNICAÇÃO_EXTERNO
        begin
                "chamada da primitiva de recepção de dados"
                output ponto_de_interação_interno.DADO_RECEBIDO(dado)
        end;
```

4.4.4.3 - O Teste da Implementação

Como a especificação orientada à implementação consiste basicamente no rearranjo dos módulos de modo a se obter os sistemas que serão posteriormente implementados, a validação neste nível se faz diretamente sobre a implementação em si. Esta última etapa de validação compõe-se então de testes de implementação cujo objetivo é verificar se as etapas anteriores foram realizadas de maneira eficaz e se o sistema obtido satisfaz às necessidades especificadas de início.

No caso dos protocolos de comunicação o principal tipo de teste realizado é conhecido como **teste de conformidade** [Rayner 87]. Este teste pode ser considerado como um caso de validação de consistência [Bochmann 87b], onde uma implementação de um protocolo, normalmente chamada de "implementação sob teste" (IUT, do inglês "*Implementation Under*

Test") é checada contra a especificação do protocolo, a qual atua como uma referência. A IUT é estimulada pela entrada de teste que é gerada por um ou vários módulos de teste. A saída gerada pela IUT em resposta à entrada de teste deve ser observada e comparada com a especificação do protocolo de modo a determinar se a saída observada está de acordo com a especificação. Aqui também é imprescindível a utilização de ferramentas automatizadas. As diversas ferramentas existentes oferecem facilidades para o registro de traços, teste interativo com o usuário, criação e observação de PDU (*Protocol Data Unit*), execução automática de casos de testes entre outras.

4.5 - Conclusão

Este capítulo apresentou uma metodologia de utilização da técnica de descrição formal Estelle (e da versão Estelle*) para o desenvolvimento de aplicações distribuídas. Esta metodologia foi baseada nos conceitos de abstração e refinamentos sucessivos, através dos quais um sistema é desenvolvido a partir de um nível de abstração elevado - considerando apenas os aspectos relevantes a este nível - e refinando-se este sistema até a obtenção do produto final. Este tipo de abordagem apresenta duas vantagens principais: a primeira decorre do fato que os problemas vão surgindo aos poucos, ou seja, de acordo com o nível de abstração em que se encontra o desenvolvimento do sistema. A segunda vantagem diz respeito à validação do sistema que, sendo este desenvolvido em níveis, pode também ser validado em níveis, isto é, após o término de um nível de refinamento o sistema pode (e deve) passar por uma etapa de validação, utilizando a técnica que se julgar mais adequada para tal nível.

Com relação a Estelle, foram apresentados os quatro níveis principais de especificação: a *especificação funcional*, a *especificação orientada a modelo*, a *especificação detalhada* e, por fim, a *especificação orientada à implementação*. Dentro de cada nível de especificação foram introduzidos os pontos de Estelle considerados relevantes, bem como a metodologia de validação mais indicada em tais níveis. É importante ressaltar que esta separação em quatro níveis não deve ser vista como uma estrutura rígida de especificação, mas sim como um arcabouço destinado a orientar o desenvolvimento das especificações. Neste sentido, nada impede que outros níveis intermediários possam vir a ser desenvolvidos, se isto for auxiliar ainda mais os processos de concepção e validação do sistema.

Outro aspecto com relação aos níveis de especificação se refere à transformação de um nível para outro. Neste sentido, o ponto mais crítico está na transformação da *especificação orientada a modelo* para a *especificação detalhada*, devido à mudança do modelo Estelle* para o Estelle ISO. É necessário que tal transformação seja feita cuidadosamente, de modo a se manter o comportamento desejado para o sistema.

CAPÍTULO 5

UM EXEMPLO DE APLICAÇÃO DA METODOLOGIA

5.1 - Introdução

Neste capítulo será apresentado um exemplo de aplicação da metodologia vista no capítulo anterior. O exemplo consiste na formalização e implementação, através de Estelle, da tarefa de montagem a ser realizada por uma Célula Flexível de Montagem (CFM), utilizando os conceitos de abstração e refinamentos definidos na metodologia. A escolha deste exemplo se justifica por se tratar de uma aplicação encontrada na indústria, que envolve problemas típicos de um sistema distribuído como paralelismo e sincronização. Além disso, a utilização de Estelle em uma aplicação deste tipo vem comprovar que o campo de aplicação desta técnica não se restringe apenas à area dos protocolos de comunicação, classe de aplicação para a qual Estelle foi concebida.

Este capítulo apresenta primeiramente a definição do exemplo de aplicação e em seguida os aspectos relativos à modelagem de uma CFM. A partir daí, a metodologia vista no capítulo 4 é utilizada sobre o exemplo proposto; são desenvolvidos os diversos níveis de especificações e as suas respectivas validações até o objetivo final que é a implementação da tarefa de montagem.

5.2 - Definição do Exemplo de Aplicação

O exemplo de aplicação escolhido para apresentar a metodologia proposta trata do sistema de controle de uma Célula Flexível de Montagem (CFM). Uma CFM consiste em um conjunto de componentes distribuídos que podem executar ações em paralelo visando a realização de uma determinada tarefa de montagem. Dessa forma, a modelagem da tarefa de montagem deverá considerar todos os possíveis problemas que poderão advir devido à característica distribuída dos componentes da célula. Tais problemas envolvem a sincronização

entre os componentes, bem como a alocação dos recursos a serem utilizados nas diversas operações.

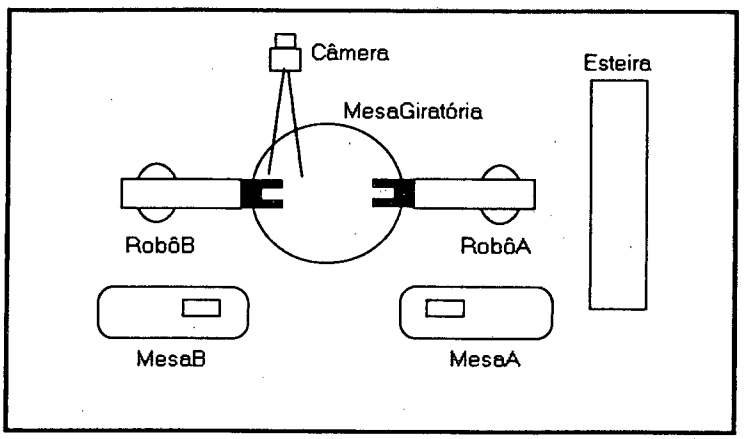


Figura 5.1 - A Célula Flexível de Montagem

A célula utilizada neste exemplo pode ser vista na figura 5.1 e é constituída dos seguintes componentes:

- a simulação é fundamentada na aproximação dos resultados. Isto é incompatível com a verificação que fornece ou o resultado correto ou o resultado incorreto, ou seja, uma prova da correção;
- dois robôs, **RobôA** e **RobôB**, responsáveis pelo deslocamento e montagem das peças;
- uma mesa, **MesaA**, para a alimentação de peças do tipo A;
- uma mesa, **MesaB**, para a alimentação de peças do tipo B e para o depósito de peças montadas incorretamente;
- uma **MesaGiratória**, que serve como um plano para o deslocamento das peças de um lado ao outro da célula e sobre o qual é feita a montagem das peças;
- uma **Câmera**, para a inspeção das peças montadas;
- uma **Esteira**, para o descarregamento das peças montadas corretamente.

5.3 - Modelagem de uma CFM

A técnica de modelagem da Célula Flexível de Montagem utilizada neste trabalho consiste no desenvolvimento de dois tipos de modelagem: uma **modelagem funcional**, para representar o conjunto de funções disponíveis na célula e uma **modelagem espacial**, para definir as diferentes restrições espaciais. Esta técnica de modelagem foi utilizada em [Mazzola 91] e é resultado de trabalhos efetuados pelo grupo de Robótica e Inteligência Artificial (RIA) do LAAS/CNRS (*Laboratoire d'Automatique et d'Analyse des Systèmes du Centre National de la Recherche Scientifique*) (França), sobre a modelagem e a programação de tarefas de montagem [Alami 89, Chochon 86].

5.3.1 - Modelagem Funcional da CFM

O objetivo da modelagem funcional é representar, de maneira abstrata, os diferentes componentes da CFM (robôs, câmeras, esteiras etc) os quais são capazes de provocar mudanças no estado de execução de uma tarefa de montagem. Devem ser definidos, então, os conceitos de **ator** e **agente** [Mazzola 91]:

- um **ator** é todo elemento capaz de manipular peças em uma CFM; os atores atuam no nível mais baixo da execução de uma tarefa de montagem e podem efetuar diversas operações sobre as peças como: deslocamentos, no caso de um braço de um robô; suportes, no caso de uma garra de um robô; inspeções e identificações, no caso de uma câmera; transformações tecnológicas, no caso de um torno; alimentações e descarregamentos de peças como no caso das esteiras;

- um **agente** representa um conjunto de atores indissociáveis e necessários à realização (completa ou parcial) de uma ação sobre uma ou mais peças, a qual provoca um avanço no processo de montagem. Um robô pode ser modelado por um agente composto por dois atores: um braço, para o deslocamento das peças e uma garra para o suporte das peças.

5.3.2 - Modelagem Espacial da CFM

O objetivo da modelagem espacial é representar o espaço disponível na célula caracterizando, desta forma, o estado de avanço da tarefa de montagem e os recursos espacias da

célula. Para a modelagem espacial são introduzidos os conceitos de **sítio, zona de trabalho** e **lugar** de uma CFM [Mazzola 91]:

- um **sítio** é todo o espaço de uma célula no qual pode-se encontrar uma peça, considerando que duas ou mais peças montadas constituem uma única peça. Como exemplo de sítios pode-se citar a garra de um robô e o espaço sob uma câmera de inspeção;

- uma **zona de trabalho** é toda a região do espaço onde dois ou mais sítios podem interagir. O espaço sobre o plano de montagem pode ser modelado como uma zona de trabalho, onde há a interação do sítio "plano de montagem" com o sítio "garra do robô".

- um **lugar** caracteriza uma classe de equivalência de sítios da CFM. Dois ou mais sítios são ditos equivalentes se eles realizam as mesmas funções (equivalência funcional) e têm acesso às mesmas zonas de trabalho (equivalência espacial). Considerando-se, por exemplo, um robô que apresente duas garras, cada garra caracteriza um sítio e as duas garras caracterizam um único lugar, por serem equivalentes tanto do ponto de vista funcional quanto espacial.

5.4 - Aplicação da Metodologia na Especificação do Exemplo

5.4.1 - Especificação Informal da Tarefa de Montagem

A tarefa de montagem apresenta as características essenciais e os principais problemas encontrados no controle de uma célula flexível de montagem. Ela consiste da montagem de duas peças *A* e *B*, produzindo uma peça *AB*.

A execução da tarefa pode ser descrita informalmente da seguinte maneira: as peças *A* e *B* são introduzidas manualmente pelo operador da célula sobre a **MesaA** e **MesaB**, respectivamente; o **RobôA** pega uma peça *A* da **MesaA** e coloca a mesma sobre o lado direito da **MesaGiratória**; a **MesaGiratória** realiza uma rotação de 180 para transferir a peça *A* para o lado esquerdo da célula onde será montada com uma peça *B*; a montagem é realizada pelo **RobôB**, após ter pego uma peça da **MesaB**; se, durante esta montagem, uma nova peça *A* é alimentada sobre a **MesaA**, o **RobôA** vai, em paralelo, realizar o seu posicionamento sobre o lado direito da **MesaGiratória**.

No final da montagem, a **Câmera** efetua a inspeção da peça montada *AB*; se o resultado da inspeção indicar uma peça montada corretamente (*ABok*) a **MesaGiratória** deve efetuar um novo giro de 180 para transferir a peça *ABok* ao lado direito da célula. A peça *ABok* será, então, transferida para a **Esteira** pelo **RobôA** para ser descarregada; no caso de uma peça incorreta (*ABer*), o **RobôB** transfere a peça para a **MesaB** para desmontagem e nova alimentação.

5.4.2 - A Especificação Funcional

Como visto no tópico 4.3.1 da metodologia a especificação funcional consiste basicamente na definição da estrutura do problema em termos dos módulos e sub-módulos. Neste primeiro nível de especificação formal a especificação apresenta um módulo supervisor da tarefa de montagem e os módulos representando os agentes da célula: *RobôA*, *MesaGiratória*, *RobôB*, *Câmera* e *Esteira*, como mostra a figura 5.2 a seguir.

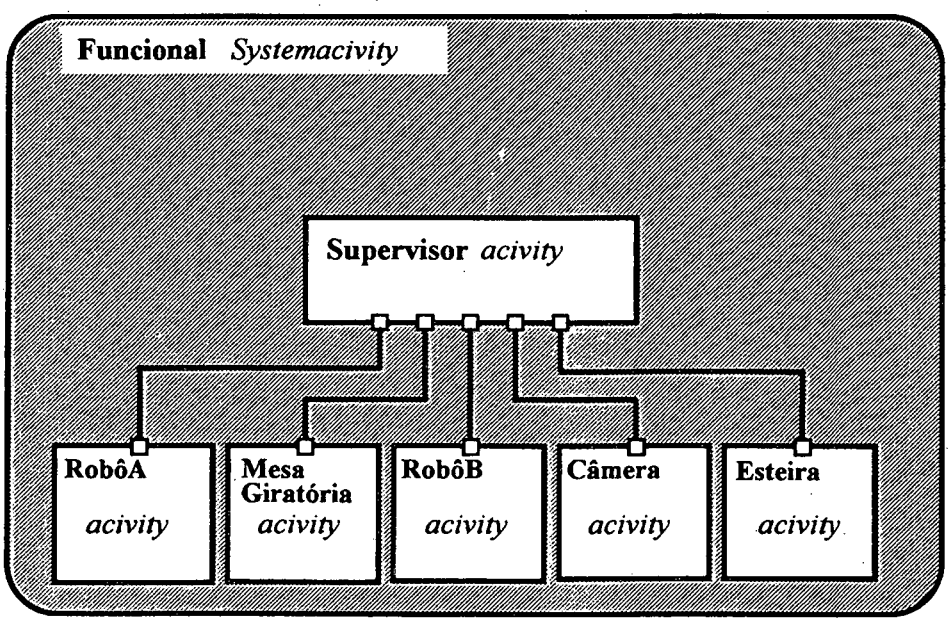


Figura 5.2 - A Especificação Funcional

A especificação funcional em Estelle da tarefa de montagem é vista a seguir:

```
SPECIFICATION EspecFuncionalTarefa SYSTEMACTIVITY;
default no queue;

CHANNEL SUPERVxROBOA (SUP,ROB);
  BY SUP: PlaceA; PlaceABOk;
  BY ROB: APlaced; ABOkPlaced;

CHANNEL SUPERVxMESAGIR (SUP,MES);
  BY SUP: Rotate;
  BY MES: Rotated;

CHANNEL SUPERVxROBOB (SUP,ROB);
  BY SUP: AssAB; UnloadABerr;
  BY ROB: ABAssembled; ABerrUnloaded;

CHANNEL SUPERVxCAMERA (SUP,CAM);
  BY SUP: InspectAB;
  BY CAM: ABInspectedOk; ABInspectedErr;

CHANNEL SUPERVxESTEIRA (SUP,EST);
  BY SUP: UnloadABOk;
  BY EST: ABOkUnloaded;

MODULE SupervisorType ACTIVITY;
  IP SupRoboA   : SUPERVxROBOA(SUP);
  IP SupMesaGir : SUPERVxMESAGIR(SUP);
  IP SupRoboB   : SUPERVxROBOB(SUP);
  IP SupCamera  : SUPERVxCAMERA(SUP);
  IP SupEsteira : SUPERVxESTEIRA(SUP);
END;

MODULE RoboAType ACTIVITY;
  IP RoboASup : SUPERVx ROBOA(ROB);
END;

MODULE MesaGirType ACTIVITY;
  IP MesaGirSup : SUPERVx MESAGIR(MES);
END;

MODULE RoboBType ACTIVITY;
  IP RoboBSup : SUPERVx ROBOB(ROB);
END;

MODULE CameraType ACTIVITY;
  IP CameraSup : SUPERVx CAMERA(CAM);
END;

MODULE EsteiraType ACTIVITY;
  IP EsteiraSup : SUPERVx ESTEIRA(EST);
END;

BODY SupervBody FOR SupervType;
END;

BODY RoboABody FOR RoboAType;
END;
```

```
BODY MesaGirBody FOR MesaGirType;
END;

BODY RoboBBody FOR RoboBType;
END;

BODY CameraBody FOR CameraType;
END;

BODY EsteiraBody FOR EsteiraType;
END;

MODVAR
  Superv    : SupervType;
  RoboA     : RoboAType;
  MesaGir   : MesaGirType;
  RoboB     : RoboBType;
  Camera    : CameraType;
  Esteira   : EsteiraType;

INITIALIZE
  BEGIN
    INIT Superv   WITH SupervBody;
    INIT RoboA    WITH RoboABody;
    INIT MesaGir  WITH MesaGirBody;
    INIT RoboB    WITH RoboABody;
    INIT Camera   WITH SupervBody;
    INIT Esteira  WITH RoboABody;
    CONNECT Superv.SupRoboA  TO RoboA.RoboASup;
    CONNECT Superv.SupMesaGir TO MesaGir.MesaGirSup;
    CONNECT Superv.SupRoboB  TO RoboB.RoboBSup;
    CONNECT Superv.SupCamera TO Camera.CameraSup;
    CONNECT Superv.SupEsteira  TO Esteira.EsteiraSup;
  END;

END. { EspecFuncionalTarefa }
```

Como foi explicitado no tópico 4.3.1 a validação da especificação funcional consiste na análise estrutural estática da especificação. Pela simplicidade do exemplo acima, uma simples inspeção do código bastaria para a sua validação. Entretanto, devido à facilidade da sua utilização, o GENESTIM (tradutor do ESTIM) serviu para comprovar a correção desta especificação.

5.4.3 - A Especificação Orientada a Modelo

Este segundo nível de especificação será caracterizado pela definição do comportamento do sistema o que traduz-se, principalmente, na inclusão dos corpos dos módulos e das interações trocadas pelos canais. Uma primeira etapa antes de partir para a especificação

Estelle consiste na identificação das ações elementares da tarefa de montagem e dos lugares necessários para a execução de cada uma destas ações.

As ações elementares são as seguintes:

- ***ColocaA***, que é realizada pelo RobôA e corresponde a pegar uma peça *A* da MesaA e colocá-la sobre o lado direito da MesaGiratória;

- ***Rotaciona***, que corresponde à uma rotação de 180° da MesaGiratória; esta ação pode ter dois efeitos simultâneos: o deslocamento de uma peça *A* do lado direito para o lado esquerdo da célula para a sua montagem; o deslocamento de uma peça corretamente montada (*ABok*) do lado esquerdo para o lado direito da célula, para o seu descarregamento posterior;

- ***MontaAB***, que é realizada pelo RobôB e corresponde a pegar uma peça *B* da MesaB e montá-la sobre uma peça *A* presente sobre o lado esquerdo da MesaGiratória;

- ***InspecionaAB***, que corresponde à uma inspeção, pela Câmera, de uma peça *AB* já montada sobre o lado esquerdo da MesaGiratória; esta ação corresponde a uma operação não previsível, onde o resultado, desconhecido até o final da sua execução, poderá ser ou uma peça montada corretamente (*ABok*) ou uma peça montada incorretamente (*ABer*);

- ***DescarregaABok***, que corresponde ao descarregamento, pelo RobôA, de uma peça montada corretamente; esta ação pressupõe a presença de uma peça *ABok* sobre o lado direito da MesaGiratória (impondo assim uma rotação da mesma após a ação de inspeção); ela representa, então, a tomada da peça *ABok*, sua deposição sobre a esteira e o seu descarregamento pela mesma;

- ***DescarregaABer***, que corresponde ao descarregamento de uma peça montada incorretamente; esta ação é efetuada pelo RobôB que pega a peça *ABer* sobre o lado esquerdo da MesaGiratória e a coloca sobre a MesaB, sabendo que esta peça será em seguida manipulada pelo operador (desmontagem e realimentação).

As ações de alimentação de peças *A* e *B* não são consideradas como ações elementares da tarefa de montagem especificada, visto que elas são realizadas por iniciativa do operador da célula, independentemente da lógica do *software*.

Com relação aos lugares da célula, deve-se lembrar que um *lugar* caracteriza uma classe de equivalência de sítios da célula, sendo esta equivalência considerada dos pontos de

vista funcional e espacial. A identificação dos lugares importantes da célula, para a representação do seu estado, depende do nível de abstração considerado para a definição das ações. Esta identificação foi feita em [Mazzola 91], relacionando as ações elementares aos sítios por elas utilizados. O resultado disto foi a definição de quatro lugares que permitem conhecer o estado da tarefa de montagem em qualquer intante dado: ***MD*** (mesa direita), ***LDMG*** (lado direito da mesa giratória), ***LEP*** (lado esquerdo da mesa giratória) e ***ME*** (mesa esquerda). A especificação orientada a modelo completa em Estelle está no anexo II. A estrutura de módulos, juntamente com a configuração dos canais e seus respectivos pontos de interação, da especificação orientada a modelo é identica à da especificação funcional mostrada na figura 5.2

5.4.3.1 - Validação da Especificação Orientada a Modelo

A técnica de validação utilizada a este nível foi a verificação formal da tarefa de montagem, com auxílio da ferramenta ESTIM. Definiu-se então uma partição (figura 5.3), tendo de um lado o módulo **Supervisor** (definido como o *mundo observado*) e, do outro, os módulos relativos aos agentes da célula (definidos como sendo o *ambiente externo*).

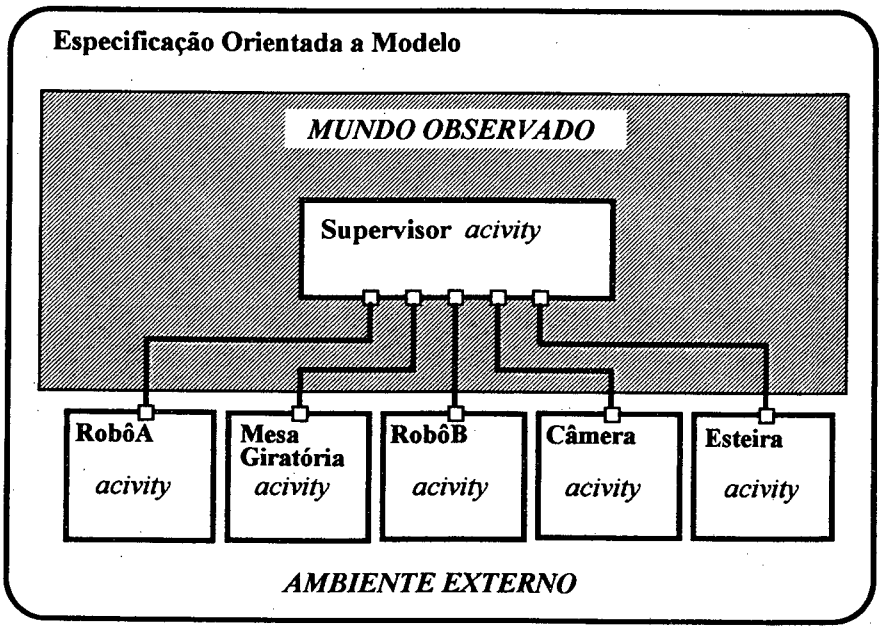


Figura 5.3 - Definição da Partição - Arquitetura de Verificação

Tendo-se definida a partição realizou-se uma análise do comportamento da célula. Esta análise foi feita da mesma forma que a realizada em [Mazzola 91] para um caso semelhante. A

única diferença é que aqui cada um dos agentes da célula é representado por um módulo distinto, enquanto naquela os agentes foram todos agrupados em um módulo genérico *célula*. Pôde-se constatar, então, a partir da geração do grafo e da sua projeção com relação à equivalência de traço algumas características do comportamento da célula, também obtidas em [Mazzola 91]:

- de qualquer estado do autômato é possível atingir o estado inicial, ou seja, a tarefa de montagem é reinicializável. Sendo o estado inicial caracterizado pela célula vazia, esta propriedade permite garantir que, em funcionamento normal, todas as peças que entram na célula podem ser montadas e descarregadas;
- partindo-se do estado inicial (célula vazia), pode-se ter no máximo 3 peças *A* no interior da célula (peças alimentadas não descarregadas).

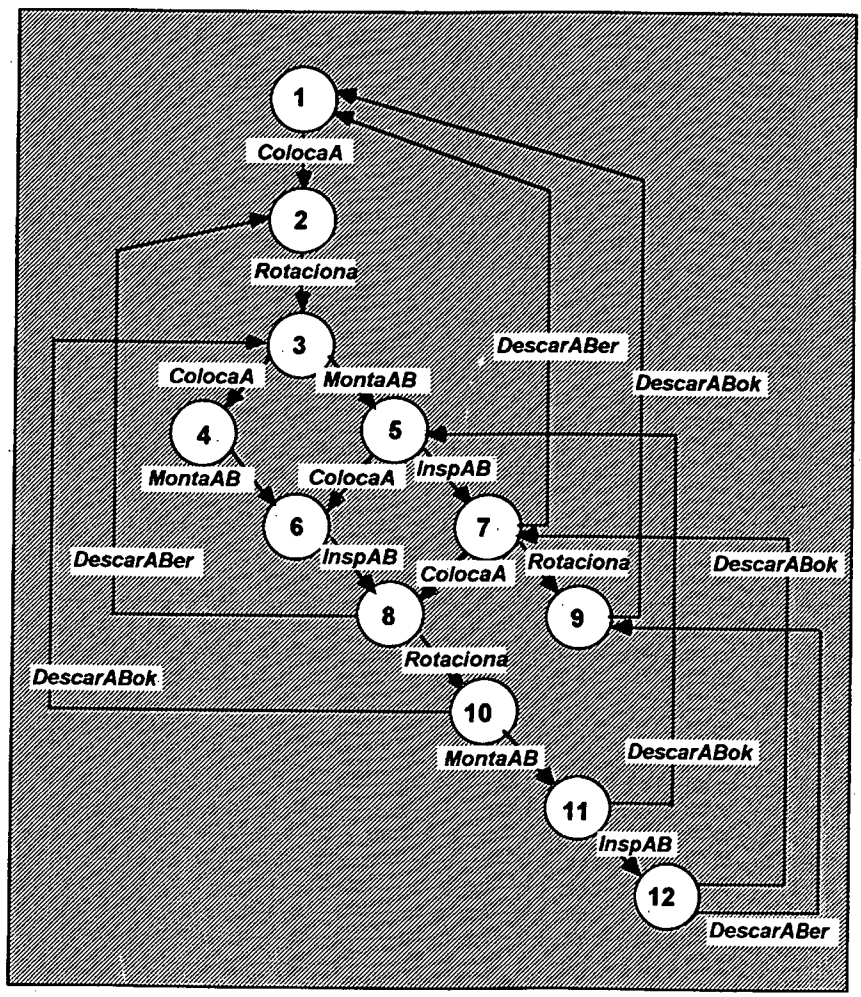


Figura 5.4 - Autômato Reduzido do Supervisor da Tarefa de Montagem - Equivalência de Traço

Esta análise permitiu verificar a coerência da especificação formal do supervisor da tarefa de montagem. Uma segunda análise pôde mostrar a presença de dois tipos de paralelismo:

- o primeiro tipo representa o paralelismo potencial durante a execução da tarefa de montagem, que é representado pelo entrelaçamento de transições representando as ações que não utilizam os mesmos recursos (funcionais e espaciais); um exemplo é a execução das ações *MontaAB* e *ColocaA*;

- o segundo tipo de não-determinismo representa o resultado de uma ação não previsível, como por exemplo a ação *InspecionaAB*;

Uma última análise foi efetuada sobre o autômato gerado com relação à equivalência observacional, a qual pôde finalmente mostrar a ausência de estados de *deadlock*. A figura 5.4 acima mostra o autômato reduzido do supervisor da tarefa de montagem utilizando a equivalência de traço.

5.4.4 - A Especificação Detalhada

O objetivo da especificação detalhada é explicitar todo o funcionamento do sistema. Na passagem da especificação orientada a modelo para a especificação detalhada foram realizadas duas transformações principais. A primeira diz respeito à comunicação entre os módulos; a comunicação síncrona através mecanismo de *rendez-vous* utilizada até então é substituída pela comunicação por filas FIFO; a segunda tranformação está relacionada com as transições correspondentes às ações da tarefa de montagem e será vista na seção seguinte. A arquitetura em módulos da especificação detalhada é mostrada na figura 5.5, a seguir.

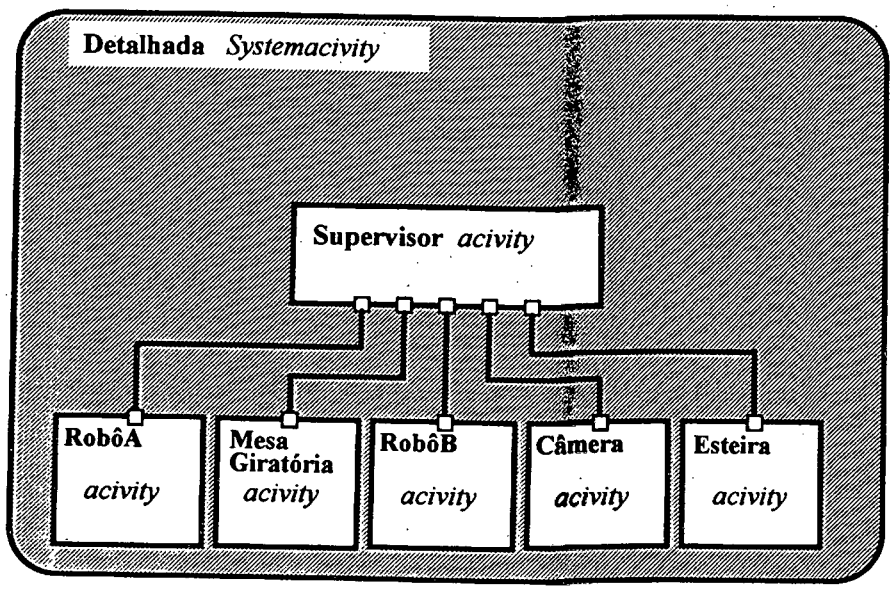


Figura 5.5 - A Especificação Detalhada

5.4.4.1 - Transformações nos Corpos dos Módulos

No módulo **Supervisor da Tarefa de Montagem** da especificação orientada a modelo cada ação da tarefa (como "*colocar peça A na mesa giratória*") correspondia a uma única transição Estelle, como será visto no exemplo abaixo. Isto porque, na verdade, todas as ações eram realizadas dentro do módulo **Supervisor**; a comunicação com os módulos agentes servia apenas para "informar" os módulos agentes que tais ações haviam sido realizadas. O objetivo disto era poder observar tais interações na verificação do supervisor da tarefa de montagem. Na especificação detalhada, as ações executadas por cada agente da célula passam a ser realizadas dentro dos respectivos módulos. Dessa forma o módulo **Supervisor** é responsável apenas pelo controle da execução das ações, isto é, quando a condição de uma transição é satisfeita, e tal transição é selecionada para ser disparada, uma ordem de execução de uma determinada ação é enviada ao respectivo módulo, o qual após a sua execução envia uma resposta dizendo que tal ação foi completada. Não é considerada aqui a hipótese de falha ou interrupção na execução de qualquer ação.

Como exemplo, são mostradas a seguir as transformações da ação "*coloca peça A na mesa giratória*". Na especificação orientada a modelo tal ação era representada em Estelle da seguinte forma:

```
TRANS
        PROVIDED (MesaA = A) AND (LadoDireitoMesaGir = EMPTY)
                TAREFAxCELULA!ColocaA
                BEGIN
                        MesaA                   = EMPTY;
                        LadoDireitoMesaGir      = A
                END;
```

Como pode-se observar, as variáveis que representam os lugares da célula já são atualizadas na própria transição; sendo assim, a ação "*coloca peça A na mesa giratória*" (e as demais ações da tarefa) são executadas com o disparo de apenas uma transição. Na especificação detalhada cada ação é composta de duas transições dentro do módulo **Supervisor da Tarefa de Montagem**; uma de envio da ordem de execução da ação e outra que recebe a confirmação da execução da ação.

Por outro lado, cada módulo representando um agente da célula contém uma transição para cada ação por ele executada; tal transição recebe a ordem de execução da ação através da cláusula Estelle *when* e envia a resposta de execução ao módulo **Supervisor da Tarefa de Montagem**. A mesma ação "*coloca peça A na mesa giratória*" na especificação detalhada fica da seguinte forma:

```
                    (*      No módulo Tarefa de Montagem      *)
TRANS
        PROVIDED (MesaA = A) AND (LadoDireitoMesaGir = EMPTY)
                AND NOT ControleRoboA
                BEGIN
                        ControleRoboA = TRUE;
                        OUTPUT Tarefa_RoboA.ColocaA
                END;


TRANS
        WHEN Tarefa_RoboA
                BEGIN
                        ControleRoboA          = FALSE;
                        MesaA                  = EMPTY;
                        LadoDireitoMesaGir     = A
                END;

                        (*      No módulo RobôA      *)
TRANS
        WHEN RoboATask.ColocaA
                BEGIN
                        OUTPUT RoboATask.AColocada
                END;
```

É importante notar dois aspectos desta nova especificação. O primeiro refere-se às variáveis que representam os lugares da célula que, obviamente, só poderão ser atualizadas após a célula haver recebido a resposta de que a ação foi completada. O segundo é conseqüência do primeiro e refere-se às variáveis de controle das transições. Uma vez disparada a transição que envia a ordem de execução da ação, as condições das transições são novamente avaliadas e, neste caso, como as variáveis da transição recém disparada ainda não foram atualizadas, esta transição estará erroneamente habilitada. A primeira solução imaginada foi criar estados globais para que uma transição, após ter sido disparada, só pudesse estar novamente habilitada quando o módulo **Supervisor da Tarefa de Montagem** recebesse a resposta do término dessa ação. Assim, haveria um estado global *Envia_Ordem* e os demais estados de espera de resposta como, por exemplo, *Aguarda_Peça_A_Colocada* ou *Aguarda_Peça_AB_Inspecionada*.

Após as primeiras análises, esta solução foi descartada pois restringia o paralelismo na execução das ações. O motivo é que as todas transições relativas às ordens de execução de ações

eram disparadas a partir do estado global *Envia_Ordem*. Após o disparo de qualquer uma dessas transições o módulo **Supervisor da Tarefa de Montagem** passava a um estado global de espera (como *Aguarda_Peça_A_Colocada*) e nenhuma outra ordem de execução de outra ação poderia ser enviada antes do final da execução da ação anterior, após o que o estado global do módulo retornaria ao estado *Envia_Ordem*.

A solução encontrada foi a eliminação dos estados globais e a criação de uma variável de controle para cada agente da célula, partindo-se do pressuposto que cada agente só pode executar uma ação por vez. Desta maneira, uma transição de ordem de execução de uma ação é desabilitada pela respectiva variável de controle após ter sido disparada. Além disso, enquanto tal ação é executada, outras ações podem ser executadas em paralelo, já que não há mais estados globais e as transições relativas a estas ações podem estar agora habilitadas. A especificação detalhada da tarefa de montagem está no anexo III.

5.4.4.2 - Validação da Especificação Detalhada

Duas técnicas foram utilizadas na tentativa de se validar a especificação detalhada. A primeira foi a execução de alguns cenários com o intuito de verificar a ocorrência de certos eventos desejáveis e a segunda foi a execução de uma simulação randômica. Tais simulações foram realizadas utilizando a ferramenta EDB.

EXECUÇÃO DE CENÁRIOS

Os cenários partiam do estado inicial da tarefa no qual a única transição habilitada era a transição de alimentação de uma peça *A*, visto que de acordo com a especificação uma peça *B* só pode ser alimentada quando houver uma peça *A* no lado esquerdo da mesa giratória.

O primeiro cenário testado consistia na alimentação de uma peça *A* e de uma peça *B*, tendo a peça *AB* montada um resultado positivo da inspeção. Este cenário obteve o resultado esperado, com a peça *AB* sendo descarregada pela esteira e a tarefa voltando ao seu estado inicial.

O segundo cenário testado era semelhante ao anterior tendo, porém, uma peça *AB* errada como resultado da inspeção; o resultado também foi o esperado, já que a peça foi corretamente descarregada pelo robô B para ser desmontada.

DISPARO RANDÔMICO DE TRANSIÇÕES

O disparo de 100 transições a partir do estado inicial da tarefa foi suficiente para observar algumas características do seu funcionamento, como a ausência de *deadlock* e o disparo de todas as transições da especificação. Constatou-se, também, pela análise das seqüências de disparo das transições, a ocorrência de ações executando em paralelo, como era previsto.

5.4.5 - A Especificação Orientada à Implementação

O desenvolvimento da especificação orientada à implementação constitui a última etapa antes da implementação do sistema. Seguindo a metodologia, a primeira modificação realizada foi a redefinição dos atributos de acordo com a organização física. Para o exemplo em questão não se dispunha de uma célula flexível de montagem para a implementação da tarefa de montagem. Dessa forma, uma implementação foi simulada utilizando-se a rede UNIX de estações de trabalho SUN, disponível no LCMI; a rede serviu como um ambiente distribuído no qual uma estação representava a tarefa de montagem e as demais representavam cada agente da célula. Já que cada módulo seria implementado em uma máquina diferente, cada módulo teve um atributo de sistema associado a ele, sendo retirado o atributo de sistema ao nível da raiz da especificação, como mostra a figura 5.6 a seguir.

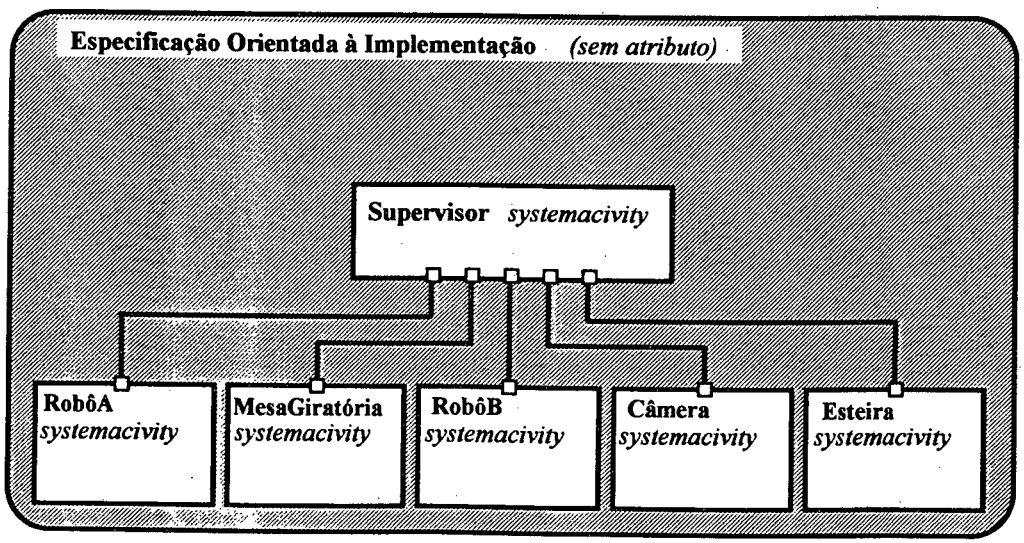


Figura 5.6 - A Especificação Orientada à Implementação

Isto feito, partiu-se para a implementação das especificações. A especificação da figura 5.6 foi expandida em seis especificações diferentes, uma para cada sistema. Antes, porém, da geração de seus códigos, foi necessário realizar algumas modificações, citadas no capítulo 4 (cf. tópico 4.4.4.2), de modo a adequar as especificações e permitir a comunicação entre as mesmas. Basicamente, as modificações feitas consistiram na adição, ao nível da raiz das especificações, de um ponto de interação interno e da interface com o meio de comunicação (neste caso, a rede UNIX do LCMI). Esta interface corresponde às transições que realizavam as chamadas de primitivas de comunicação. Na verdade, como será visto adiante, foi desenvolvida uma biblioteca de funções com o objetivo de auxiliar no processo de comunicação, deixando-a mais transparente do ponto de vista da especificação Estelle.

5.4.5.1 - A Utilização dos Soquetes UNIX

O soquete é um dos mecanismos de comunicação do tipo "passagem de mensagens" do UNIX, sendo composto de três elementos que definem a comunicação entre os processos: um *domínio de comunicação*, um *tipo* e um *protocolo* associado [Rochkind 85, SUN 90].

O *domínio de comunicação* determina se a comunicação será local (Domínio UNIX) ou remota (Domínio INTERNET) e está relacionado com a estrutura de endereçamento (família de endereços) e com o conjunto de protocolos que implementa os vários tipos de soquete dentro de um domínio (família de protocolos). No domínio UNIX o nome (endereço) do soquete representa o caminho ("*pathname*") de acesso ao soquete. No domínio INTERNET os endereços são formados pelo endereço da máquina na rede ("*hostname*") e por um número identificador chamado porta ("*port*").

O *tipo* do soquete determina a maneira como os dados são transmitidos. Os dois tipos usuais são o tipo STREAM e o tipo DATAGRAMA. A comunicação do tipo STREAM implica no estabelecimento de uma conexão entre os soquetes. Essa comunicação fornece um fluxo de dados bidirecional, confiável, seqüencial, não duplicado além de não manter os limites entre as mensagens. O próprio protocolo que implementa tal estilo se encarrega de retransmitir as mensagens recebidas com erros. A comunicação do tipo DATAGRAMA não utiliza conexões. É uma comunicação bidirecional que não garante a seqüencialidade, a confiabilidade e a não duplicação dos dados transmitidos. Os datagramas enviados são mantidos separados, ou seja, os limites entre as mensagens são preservados. Já que não existe o estabelecimento de uma conexão, cada mensagem a ser enviada deve ser endereçada individualmente, ou seja, o endereço do soquete destino deve ser fornecido a cada envio. Se o endereço estiver correto, ela geralmente será recebida, embora isso não seja garantido.

O *protocolo* é o conjunto das regras, formatos de dados e convenções que regulam a transferência de dados entre os participantes da comunicação. Geralmente existe, dentro de cada domínio, um protocolo para cada tipo de soquete. Por exemplo, no Domínio INTERNET o protocolo padrão ("*default*") para o soquete do tipo STREAM é o TCP (*Transmission Control Protocol*), enquanto que para o tipo DATAGRAMA o protocolo padrão é o UDP (*User Datagram Protocol*). Embora um protocolo diferente possa ser selecionado, o protocolo padrão satisfaz a quase todos os casos e é normalmente o protocolo utilizado.

Na comunicação através de soquetes cada processo participante da comunicação deve criar o seu próprio soquete (esses soquetes devem ser do mesmo tipo) estabelecendo, assim, um ponto final de comunicação. Além disso, um nome deve ser associado ao soquete para que os demais processos possam acessá-lo. A comunicação se dá através de primitivas de envio e recepção de acordo com o tipo dos soquetes. No tipo STREAM, uma conexão deve ser estabelecida antes da transferência dos dados e, dessa forma, as primitivas não precisam fornecer os endereços de destino e de origem. No tipo DATAGRAMA, porém, não existe o estabelecimento de uma conexão e, conseqüentemente, a primitiva de envio deve fornecer o endereço alvo, o mesmo acontecendo para a primitiva de recepção, se o processo receptor necessitar saber a origem do dado.

No exemplo da célula utilizou-se o domínio INTERNET pois os processos estavam distribuídos em diferentes estações da rede. Como não havia a necessidade de estabelecimento de conexões entre os soquetes, optou-se por uma comunicação do tipo DATAGRAMA, utilizando o seu protocolo padrão.

5.4.5.2 - A Utilização de uma Biblioteca de Funções para os Soquetes

Tendo-se definido então que os processos comunicantes executariam em máquinas diferentes (domínio INTERNET) e que tal comunicação seria do tipo DATAGRAMA, foi desenvolvida uma biblioteca de funções (***lib_inet.c***) para facilitar a criação dos soquetes e a comunicação entre os mesmos. Esta biblioteca é mostrada no anexo IV e possui as seguintes funções: *cria_soquete()*, *envia_mensagem()*, *recebe_mensagem()*, *soquete_nao_vazio()*. A função *soquete_nao_vazio()* testa a existência de alguma mensagem para leitura no soquete. Tal função é necessária visto que a primitiva de leitura *read()* do UNIX, utilizada pela função *recebe_mensagem()*, é bloqueante, ou seja, quando é chamada fica aguardando até que algum dado seja recebido. Como a cláusula *when* de Estelle requer uma leitura não-bloqueante, deve ser testada a existência de algum dado no soquete antes de se realizar a operação de leitura.

Assim sendo, a função *soquete_não_vazio()* faz uso da primitiva *select()*, que retorna o número de caracteres contidos no soquete.

5.4.5.3 - Preparando a Especificação Estelle do Supervisor da Tarefa de Montagem para a Implementação

Serão mostradas agora as principais modificações realizadas na especificação Estelle do Supervisor da Tarefa de Montagem[1]; a especificação completa está no anexo V. Como sugerido no tópico 4.4.4.2, os corpos dos módulos permaneceram inalterados, sendo as modificações feitas ao nível da raiz da especificação. Tais modificações foram:

- a inclusão de um ponto de interação interno para permitir a troca de mensagens entre o módulo Tarefa e a raiz da especificação;
- a declaração do tipo dos dados a serem utizados na comunicação entre as especificações. Neste caso optou-se pelo tipo enumerado;
- as declarações das variáveis relativas ao endereço do soquete (nome da estação remota e número da porta);

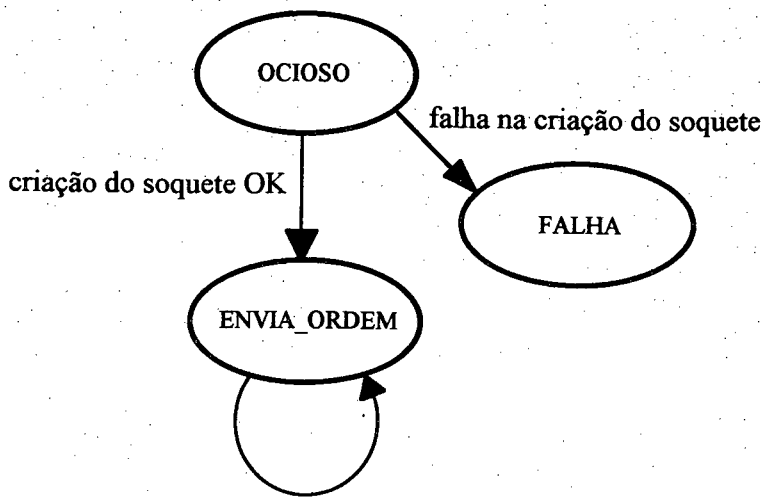


Figura 5.7 - A Máquina de Estados da Raiz da Especificação do Supervisor

Foram criados, então, três estados globais: OCIOSO, FALHA e ENVIA_ORDEM. A especificação é inicializada no estado OCIOSO a partir do qual ela chama a função para criar o

[1] As modificações realizadas nas demais especificações foram, de um modo geral, bastante semelhantes às do Supervisor e por esse motivo não serão apresentadas aqui.

soquete; se algum problema ocorre na criação do soquete, a especificação vai para o estado FALHA e aborta; caso contrário, ela passa para o estado ENVIA_ORDEM, no qual a variável módulo da tarefa é inicializada e o seu ponto de interação externo conectado com o ponto de interação interno da especificação. Neste estado, a especificação envia as ordens de execução das ações às respectivas estações e também recebe a confirmação do final da execução destas ações; ela permanece neste estado até que o usuário interrompa o processo. A figura 5.7 acima mostra a máquina de estados da raiz da especificação do Supervisor da Tarefa de Montagem.

Para o envio das mensagens foi criada uma transição para cada ordem de execução recebida do módulo SUPERVISOR. Em cada transição a variável *dado* é atualizada, de acordo com a ordem recebida; são atualizadas também variáveis relativas ao endereço do soquete e, por fim, é chamada a função de envio de mensagens. Como exemplo, a transição que envia a ordem de colocar a peça *A* na mesa giratória é mostrada a seguir:

```
TRANS
        WHEN Spec.PlaceA
        FROM ENVIA_ORDEM to same
                BEGIN
                        data := ColocaA;
                        num_port_destino := 10;
                        (*$C$ strcpy(_adctx->estacao_destino, "antares"); *)
                        (*$C$       envia_mensagem(_adctx->num_soquete,      _adctx-
>num_port_destino,
                                        _adctx->estacao_destino,                &_adctx->data,
sizeof(_adctx->data)); *)
                END;
```

Para a recepção da confirmação do final das ações, foi criada uma transição que testa a presença de algum dado no soquete e, em caso afirmativo, faz a leitura deste dado e o envia para o módulo SUPERVISOR. Esta transição é mostrada a seguir:

```
TRANS
        FROM ENVIA_ORDEM to same
                PROVIDED soquete_nao_vazio(num_soquete)
                BEGIN
                        (*$C$ recebe_dado(_adctx->num_soquete, &_adctx->data); *)
                        case data of
                                AColocada               : output Spec.APlaced;
                                ABokColocada            : output Spec.ABokPlaced;
                                MesaGirada              : output Spec.Rotated;
                                ABMontada               : output Spec.ABAssembled;
                                ABerrDescarregada       : output Spec.ABerrUnloaded;
                                ABResultOk              : output Spec.ABInspectedOk;
                                ABResultErr             : output Spec.ABInspectedErr;
                                ABOkDescarregada        : output Spec.ABOkUnloaded;
```

```
        end;
END;
```

Uma observação importante deve ser feita com relação às especificações relativas aos agentes da célula. Em cada especificação foi introduzido um atraso arbitrário nas transições que representavam as ações destes agentes, de modo a simular o tempo de execução destas ações. Com isto pôde-se verificar, quando da execução das especificações, a ocorrência de ações em paralelo, bem como situações de sincronização.

5.4.5.4 - A Geração Automática do Código

A geração automática do código das especificações expandidas da figura 5.6 foi realizada com auxílio do compilador Estelle-to-C (EC) do pacote EDT. A geração de cada código consistia em duas etapas: a geração do código fonte e a geração do código executável. Na geração do código fonte utilizou-se duas opções do compilador, a primeira para que o compilador pudesse interpretar os comentários de qualificação (opção "-q"); a segunda para que o compilador gerasse um *makefile* para o código (opção "-M"). Com essas opções, os seguinte tipos de arquivos são gerados:

- um arquivo do tipo *<nome_do_arquivo_fonte>.h*;
- um arquivo do tipo *<nome_do_arquivo_fonte>.c*;
- dois arquivos, um do tipo *<NOME_DA_ESPECIFICAÇÃO><número>.h* e um do tipo *<NOME_DA_ESPECIFICAÇÃO><número>.c*, para o corpo da especificação;
- dois arquivos, um do tipo *<NOME_DO_CORPO><número>.h* e um do tipo *<NOME_DO_CORPO><número>.c*, para cada corpo de módulo da especificação;
- um arquivo do tipo *<nome_do_arquivo_fonte>.mk*.

Além destes ainda são gerados os respectivos arquivos objetos (arquivos com extensão ".o") e arquivos com a lista de erros (arquivos com extensão ".li").

Para a geração do código executável utilizou-se o *makefile* gerado anteriormente no arquivo *<nome_do_arquivo_fonte>.mk*. A única alteração no *makefile* foi a inclusão do arquivo objeto da biblioteca de funções (*lib_inet.o*). A partir daí, gerou-se o arquivo executável de cada especificação, através do comando:

make -f <nome_do_arquivo_fonte>.mk

5.4.5.5 - A Execução Distribuída das Especificações

Tendo-se obtido os arquivos executáveis, a última etapa foi a execução destes nas respectivas estações de trabalho, como mostra a figura 5.8. Os arquivos foram executados em estações de trabalho previamente determinadas; isto porque, dentre os parâmetros da função de envio de mensagens, estão o número da porta e o nome da estação destino. Para a execução final utilizou-se o recurso multi-janelas do ambiente *OpenWindows*; assim, em uma única estação de trabalho, pôde-se abrir seis janelas de comando (*cmdtool*), conectar-se remotamente nas respectivas estações - através do comando *rlogin* - e, enfim, executar as especificações. Como os soquetes já devem estar criados para poderem receber dados, as especificações referentes aos agentes devem ser executadas antes da especificação do supervisor, pois é sempre o supervisor que envia os dados primeiro, e depois fica aguardando uma resposta.

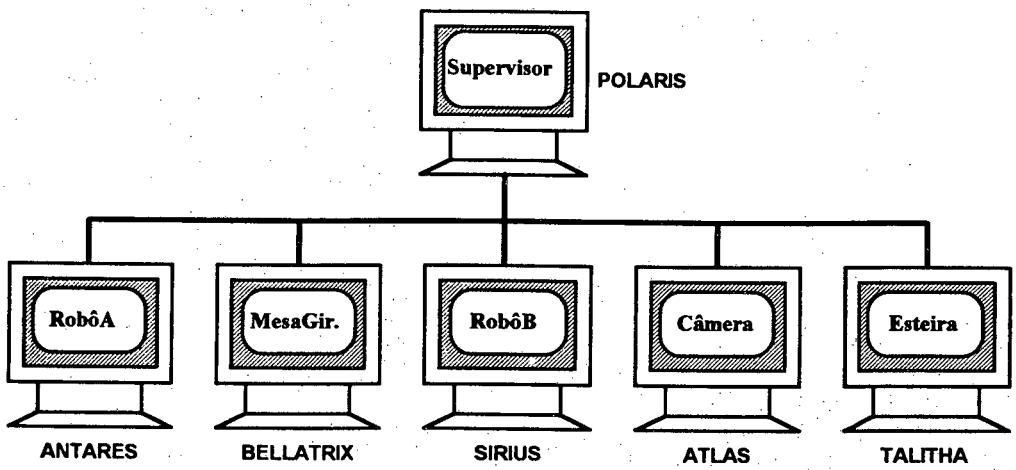


Figura 5.8 - Disposição das Especificações na Rede UNIX de Estações de Trabalho do LCMI

A figura 5.8 apresenta a configuração das especificações na rede de estações de trabalho do LCMI. A seguir serão apresentados dois trechos da execução do supervisor da tarefa de montagem:

```
ORDEM: Alimentar Mesa Direita ==> Peca A          (Entrada A1)
ORDEM: RoboA Colocar Peca A na Mesa Giratoria
ACK  : Peca A Colocada na Mesa Giratoria
ORDEM: Rotacionar a Mesa Giratoria
ACK  : Mesa Giratoria Rotacionada
ORDEM: Alimentar Mesa Esquerda ==> Peca B         (Entrada B1)
ORDEM: RoboB Montar Peca AB
ORDEM: Alimentar Mesa Direita ==> Peca A          (Entrada A2)
ORDEM: RoboA Colocar Peca A na Mesa Giratoria
ACK  : Peca AB Montada
ACK  : Peca A Colocada na Mesa Giratoria
ORDEM: Camera Inspecionar Peca AB
ACK  : Inspecao Peca AB Resultou OK
ORDEM: Rotacionar a Mesa Giratoria
```

```
ACK  : Mesa Giratoria Rotacionada
ORDEM: RoboA Colocar Peca ABOk na Esteira
ORDEM: Alimentar Mesa Esquerda ==> Peca B          (Entrada B2)
ACK  : Peca ABOk Colocada na Esteira
ORDEM: Alimentar Mesa Direita ==> Peca A           (Entrada A3)
ORDEM: Esteira Descarregar Peca ABOk
ORDEM: RoboB Montar Peca AB
ACK  : Peca ABOk na Esteira Descarregada           (Saída ABOk1)
   •
   •
   •
```

Este primeiro trecho é relativo ao início da execução e nele podem ser vistas a entrada da primeira peça *A* (Entrada A1) e a saída da primeira peça *AB* correta (Saída ABOk1). Neste intervalo observa-se a entrada da primeira peça *B* (Entrada B1), a entrada de uma segunda peça *A* (Entrada A2) e da sua respectiva peça *B* (Entrada B1) e, ainda, a entrada de uma terceira peça *A* (Entrada A3).

Pode-se notar também o paralelismo em várias ocasiões como, por exemplo, nas últimas três ordens de execução, onde o operador pode alimentar a MesaA com uma peça *A*, a Esteira pode descarregar uma peça *ABOk* e o RobôB pode montar uma peça *AB*. A seguir é apresentado um segundo trecho da execução:

```
   •
   •
   •
ORDEM: Alimentar Mesa Direita ==> Peca A          *
ACK  : Peca AB Montada
ORDEM: Esteira Descarregar Peca ABOk
ORDEM: RoboA Colocar Peca A na Mesa Giratoria     *
ACK  : Peca ABOk na Esteira Descarregada
ACK  : Peca A Colocada na Mesa Giratoria          *
ORDEM: Camera Inspecionar Peca AB
ACK  : Inspecao Peca AB Resultou OK
ORDEM: Rotacionar a Mesa Giratoria                *
ACK  : Mesa Giratoria Rotacionada                 *
ORDEM: Alimentar Mesa Esquerda ==> Peca B         *
ORDEM: RoboA Colocar Peca ABOk na Esteira
ORDEM: RoboB Montar Peca AB                       *
ACK  : Peca ABOk Colocada na Esteira
ORDEM: Esteira Descarregar Peca ABOk
ACK  : Peca AB Montada                                  *
ACK  : Peca ABOk na Esteira Descarregada
ORDEM: Camera Inspecionar Peca AB                 *
ORDEM: Alimentar Mesa Direita ==> Peca A
ACK  : Inspecao Peca AB Resultou Erro             *
ORDEM: RoboA Descarregar Peca ABErr               *
ORDEM: RoboA Colocar Peca A na Mesa Giratoria
ACK  : Peca ABErr Descarregada                    *
   •
   •
   •
```

Este segundo trecho mostra a entrada de uma peça *A*, a sua montagem com uma peça *B*, a inspeção desta peça resultando em uma peça incorreta e o seu descarregamento pelo RobôB sobre a MesaB (as ações diante das quais não aparece o símbolo * pertencem a outras peças também presentes na célula). Durante a execução pôde-se observar ainda outras situações em que aparecem ações executando em paralelo, além da sincronização na execução das ações.

5.5 - Conclusão

Este capítulo apresentou um exemplo de aplicação da metodologia proposta no capítulo 4. O exemplo escolhido foi a especificação da tarefa de montagem de uma célula flexível de montagem. Após a definição do exemplo, foram apresentados alguns conceitos relativos à modelagem de uma célula flexível de montagem, para então desenvolver as especificações sugeridas na metodologia.

Em cada especificação foram levantados os pontos principais relativos à utilização dos mecanismos estruturais de Estelle, sendo feitas também algumas análises com respeito à etapa de validação de cada especificação. Após o desenvolvimento e validação das especificações foi realizada uma experiência de implementação, tendo a rede de estações de trabalho do LCMI como ambiente distribuído. Após as alterações necessárias nas especificações, utilizou-se o compilador EC para a geração dos códigos de implementação os quais foram executados nas diferentes estações da rede.

Pelos resultados obtidos e mostrados na seção anterior pode-se dizer que a implementação teve um resultado positivo, mostrando dessa maneira que Estelle não se limita apenas ao campo dos protocolos de comunicação, sendo adequada também para outras áreas correlatas, como o caso da célula flexível de montagem, visto neste capítulo. Outro ponto a destacar diz respeito à metodologia utilizada, considerada satisfatória no desenvolvimento desta aplicação, pois permitiu o desenvolvimento desta, desde a fase inicial de especificação até a obtenção da sua implementação.

CAPÍTULO 6

CONCLUSÕES E PERSPECTIVAS

Este trabalho apresentou os resultados obtidos no estudo e na definição de uma metodologia coerente e suportada por um conjunto de ferramentas de software para a concepção de aplicações distribuídas. Esta metodologia é fundada sobre a técnica de descrição formal Estelle, definida na ISO, que apresenta um formalismo bastante orientado à resolução dos problemas típicos dos sistemas distribuídos, tais como a comunicação e a concorrência. A utilização de uma técnica de descrição formal, tal como Estelle, mostra-se imprescindível no desenvolvimento de qualquer aplicação distribuída, visto que este tipo de aplicação apresenta normalmente uma grande complexidade

Outro aspecto a ressaltar é a disponibilidade de ferramentas automatizadas que suportem tais técnicas. As ferramentas hoje existentes cobrem praticamente todo o espectro do processo de aplicações distribuídas; ferramentas orientadas à edição, à validação (análisadores sintáticos e semânticos, simuladores, verificadores formais) e à implementação são uma realidade, muitas delas integradas em ambientes de desenvolvimento. Esse ponto é de extrema importância pois, conforme visto durante o trabalho, o uso de qualquer técnica de descrição formal depende fortemente destas ferramentas. Neste sentido, foram apresentadas três ferramentas de suporte à Estelle disponíveis no LCMI - o ESTIM, o EDT e o Echidna - procurando-se mostrar as facilidades disponíveis em cada uma. Tais ferramentas apresentam características distintas com relação à etapa do desenvolvimento na qual elas se enquadram. O ESTIM apresenta facilidades de simulação e verificação; o EDT, facilidades de simulação e geração de código de implementação; o Echidna, por sua vez, permite a geração distribuída do código de implementação, bem como facilidades para uma simulação gráfica distribuída. Dessa forma, de posse destas três ferramentas, teve-se a possibilidade de utilizá-las em grande parte do processo de desenvolvimento, desde a validação (análise sintática e semântica, simulação, verificação) até a implementação do código gerado automaticamente.

A metodologia desenvolvida baseou-se nos conceitos de abstração e refinamentos sucessivos, sobre os quais foram definidos níveis de especificação, que partem da especificação funcional e vão se refinando até atingir a especificação orientada à implementação, último

passo antes da implementação final do sistema. Essa técnica mostrou-se bem adequada para tratar de sistemas de grande complexidade, como é o caso das aplicações distribuídas, pois ela permite que os obstáculos sejam defrontados passo a passo, em cada nível de especificação. Da mesma forma, o processo de análise também é efetuado em níveis, facilitando tal processo e evitando-se que um erro, ocorrido no início do desenvolvimento do sistema, seja detectado somente ao final deste, quando da sua implementação.

Para mostrar a eficiência desta metodologia, foi desenvolvido um exemplo de aplicação distribuída que consistiu na especificação formal da tarefa de montagem de uma Célula Flexível de Montagem. Neste exemplo foram desenvolvidas as especificações propostas na metodologia, dirigindo-se também uma atenção às etapas de validação. Por fim, o exemplo foi implementado utilizando-se as estações de trabalho do LCMI, que serviram com um ambiente distribuído no qual cada estação de trabalho representava um agente da célula flexível.

Através deste exemplo, pôde-se observar aspectos relevantes das aplicações ditribuídas, como a comunicação e o paralelismo. Isto serviu para mostrar que a técnica descrição formal Estelle pode ser bastante útil para o desenvolvimento de aplicações distribuídas que vão além do domínio dos protocolos de comunicação, para o qual tal técnica foi inicialmente concebida.

Embora os resultados obtidos neste trabalho tenham sido considerados satisfatórios, inclusive a partir da ilustração pela aplicação da metodologia sobre o exemplo acima citado, existem alguns aspectos relativos ao desenvolvimento de aplicações distribuídas que, apesar de terem feito parte das preocupações iniciais, não puderam ser esgotados no âmbito deste trabalho.

Comentando estes aspectos, espera-se criar motivações para novos estudos no sentido de aprimorar os resultados aqui obtidos, oferecendo assim, melhores condições para os programadores de aplicações distribuídas:

APLICAÇÕES COM RESTRIÇÕES TEMPORAIS

A escolha de Estelle como ferramenta de desenvolvimento foi plenamente justificada neste trabalho, principalmente pelos mecanismos apropriados à representação de problemas típicos das aplicações distribuídas, como o paralelismo e a comunicação. Aliam-se a isto, do ponto de vista metodológico, as facilidades de estruturação existentes em Estelle, a clara separação entre configuração e programação, a possibilidade de instanciação dinâmica de módulos e, é claro, a existência de um conjunto importante de ferramentas de suporte.

Por outro lado, a considerar-se a possibilidade do desenvolvimento de aplicações apresentando restrições de tempo, encontra-se em Estelle um ponto fraco. A semântica de tempo

em Estelle pode ser utilizada para a representação de mecanismos de *timeout* ou como uma forma de resolução de não-determinismo, mas a limitação existe quando o problema é o estabelecimento de limites temporais no contexto da aplicação.

A resolução deste problema estaria num estudo teórico de introdução de novos mecanismos temporais em Estelle. Este tipo de estudo é bastante complexo e vem sendo realizado no âmbito de outras técnicas de descrição formal, como no caso de LOTOS [Camargo 94].

DIVERSIDADE DE MODELOS EM ESTELLE

A escolha de diversas ferramentas para o suporte à metodologia de desenvolvimento aqui proposta justifica-se pelo fato de que, somente com a integração destas ferramentas, pode-se garantir a realização de um conjunto de tarefas determinantes ao desenvolvimento de uma aplicação distribuída.

Um inconveniente, porém, da adoção de diferentes ferramentas, é o fato destas considerarem diferentes modelos de comportamento para uma mesma técnica de descrição formal. Isto fica evidente quando compara-se, por exemplo, o modelo suportado pelo EDT (Estelle ISO) e aquele suportado pelo ambiente ESTIM (Estelle*). Como, em diferentes etapas, utilizam-se diferentes ferramentas, modelos de comportamento distintos são também considerados ao longo das etapas. Em boa parte dos casos, esta incompatibilidade em relação aos modelos pode gerar resultados não tão eficientes, mesmo com a utilização da metodologia proposta.

Um problema a ser tratado em um futuro trabalho será, sem dúvida, um estudo destes modelos para verificar as possibilidades de equacionamento das suas incompatibilidades no que diz respeito à representação de comportamento. Problemas importantes, como o da comunicação (filas FIFO e *rendez-vous*), foram apenas introduzidos e devem ser aprofundados no sentido de reduzir as incompatibilidades entre os diferentes níveis de especificação e validação propostos na metodologia.

INTERFACE AMIGÁVEL PARA AS FERRAMENTAS

As ferramentas utilizadas neste trabalho, apesar de apresentarem um grau bastante satisfatório no que diz respeito ao desempenho e correção de funcionamento, apresentam um sério inconveniente que são as interfaces de acesso. Isto é explicado pelo fato de que todas elas

são protótipos gerados em laboratórios de pesquisa, onde este aspecto não teve uma preocupação primordial.

Um trabalho interessante seria poder gerar interfaces de acesso a estas ferramentas que permitissem explorar os recursos gráficos disponíveis hoje nas estações de trabalho, tornando assim mais confortável e eficiente, o trabalho do programador de uma aplicação distribuída. É evidente que, para a realização de tal trabalho, seria necessário ter acesso ao código fonte das ferramentas consideradas, razão pela qual isto foi descartado como resultado a ser obtido no contexto deste trabalho.

REFERÊNCIAS BIBLIOGRÁFICAS

[Alami 89] R. Alami et al. "Conception en Estelle du Pilotage d'une Cellule Flexible d'Assemblage", Anais do Seminário Franco-Brasileiro em Sistemas Informáticos Distribuídos, Florianópolis, pp. 111-117, Setembro, 1989.

[Alcântara 92] S. C. Alcântara. A. C. P. Pedroza. *"Implementação Semi-Automática de Protocolos Descritos em Estelle"*, 9º Congresso Brasileiro de Automática, CBA, pp. 886-891, 1992.

[Ansart 86] J-P. Ansart *et al. "Software Tools for Estelle"*, Proceedings of the IFIP Conference on Protocol Specification, Testing and Verification VI, Montreal, pp. 55-61, 1986.

[Ayache 89] J-M. Ayache, J. Dufau, M. Huybrechts, E. Mattera. *"EWS: An Integrated Workstation for the Design and Automatic Generation of Distributed Software"*, Formal Description Techniques, North-Holland (Elsevier), pp. 85-89, 1989.

[Azéma 87] P. Azéma *et al. "Spécification Formelle et Conception des Systèmes Informatiques Distribués"*, Relatório LAAS nº 87306, LAAS-CNRS, 1987.

[Berthomieu 83] B. Berthomieu, M. Menasche. *"An Enumerative Approach for Analysing Time Petri Nets"*, Proceedings of IFIP, Paris, 1983.

[Bochmann 87a] G. v. Bochmann. *"Usage of Protocol Development Tools: The Results of a Survey"*, Proceedings of the IFIP Symposium on Protocol Specification, Testing and Verification, Zurich, pp.139-161, 1987.

[Bochmann 87b] G. v. Bochmann, G. W. Gerber, J. M. Serre. *"Semiautomatic Implementation of Communication Protocols"*, IEEE Transactions on Software Engineering, vol. 13, nº 13, pp. 989-1000, 1987.

[Bochmann 90] G. v. Bochmann. *"Protocol Specification for OSI"*, Computer Networks and ISDN System, vol. 18, pp. 167-184, 1990.

[Budkowski 87] S. Budkowski, P. Dembinski. *"An Introduction to ESTELLE: a Specification Language for Distributed Systems"*, Computer Networks and ISDN Systems, vol. 14, pp. 3-23, 1987.

[Camargo 94] M. S. Camargo, J-M. Farines. *"Tornando LOTOS Apta para Especificar Sistemas Tempo Real"*, Anais do 12º Simpósio Brasileiro de Redes de Computadores (SBRC), Curitiba, vol.1, pp. 245-264, Maio, 1984.

[Chochon 86] H. Chochon. *"Programmation et Conduite de Tâches d'Assemblage Robotisées: Modélisation et Processus Décisionnels"*, Tese de Doutorado, Toulouse, Novembro, 1986.

[Chung 90] A. Chung, D. Sidhu. *"Experience with an Estelle Development System"*, Proceedings of the ACM SigSoft International Workshop on Formal Methods in Software Development. Napa, California, pp. 8-17, Maio, 1990.

[Courtiat 87a] J-P. Courtiat, P. Dembinski, R. Groz, C. Jard. *"Estelle: An ISO Language for Distributed Algorithms and Protocols"*, TSI Technology and Science of Informatics, vol.6, nº 5, pp. 311-324, 1987.

[Courtiat 87b] J-P. Courtiat. *"How Could Estelle Become a Better FDT?"*, Proceedings of the IFIP Conference on Protocol Specification, Testing and Verification VII, Zurich, pp. 43-60, 1987.

[Courtiat 88] J-P. Courtiat. *"Estelle*: A Powerful Dialect of Estelle for OSI Protocol Description"*, Proceedings of the 8th IFIP Symposium on Protocol Specification, Testing and Verification, Atlantic City, 1988.

[Courtiat 89] J-P. Courtiat. *"Introducing a Rendez-Vous Mechanism in Estelle: Estelle*"*, The Formal Description Technique Estelle, North-Holland, pp. 175-203, 1989.

[Courtiat 91] J-P. Courtiat, M. Diaz, V. B. Mazzola, P. de Saqui-Sannes. *"Description Formelle de Protocoles et de Services OSI en ESTELLE et ESTELLE*: Expérience et Méthodologie"*, Colloque Francophone sur l'Ingénerie des Protocoles (CFIP'91), Pau, pp. 65-87, Setembro, 1991.

[Courtiat 92] J-P. Courtiat, P. de Saqui-Sannes. *"ESTIM: An Integrated Environment for the Simulation and Verification of OSI Protocols Specified in Estelle*"*, Computer Networks and ISDN Systems, vol. 25, pp. 83-98, 1992.

[Danthine 80] A. Danthine. *"Protocol Representation with Finite-State Models"*, IEEE Transactions on Communications, vol. COM-28, nº 4, pp. 632-643, Abril, 1980.

[Diaz 83] M. Diaz, G. Guidacci da Silveira. *"On the Specification and Validation of Protocols by Temporal Logics and Nets"*, Proceedings of the IFIP 83 Congress, Paris, Setembro, 1983.

[Diaz 89a] M. Diaz, C. Vissers, J-P Ansart. *"SEDOS Software Environment for the Design of Open Distributed Systems"*, in The Formal Description Technique Estelle, North-Holland, pp. 03-14, 1989.

[Diaz 89b] M. Diaz, C. Vissers. *"SEDOS. Designing Open Distributed Systems"*, IEEE Software, 1989, pp. 24-33.

[EC 92] *"EC - Estelle-to-C Compiler: User Reference Manual (Version 3.0)"*, INT Institut National des Télécomunications, 1992.

[EDB 92] *"EDB - Estelle Simulator/Debugger: User Reference Manual (Version 3.0)"*, INT Institut National des Télécomunications, 1992.

[Enslow 78] P. H. Enslow, Jr. *"What is a Distributed Data System?"*, Computer, vol. 11, nº 1, pp. 13-21, Janeiro, 1978.

[Ernberg 91] P. Ernberg *et al.* *"Guidelines for Specification and Verification of Communication Protocols"*, SICS Perspective, Relatório nº 1, Swedish Institute of Computer Science, 1991.

[Estelle 88] ISO IS 9074. *"ESTELLE - A Formal Description Technique Based on a Extended State Transition Model"*, International Standardization Organization, Novembro, 1988.

[Fernandez 88] J. C. Fernandez. *"Un Système de Vérification par Réduction de Process Communicants"*, Tese de Doutorado, Grenoble, 1988.

[Groz 89] R. Groz. *"Vérification de Propriétés Logiques des Protocoles et Systèmes Répartis par Observation de Simulations"*, Tese de Doutorado, Rennes, Janeiro, 1989.

[Harangozo 78] H. Harangozo. *"Protocol Definitions with Formal Grammars"*, Symposium on Computer Communication Protocols, Liège, Bélgica, Fevereiro, 1978.

[Hoare 78] C. A. R. Hoare. *"Communicating Sequential Process"*, Communications of the ACM, Agosto, 1978.

[Jard 88] C. Jard. *"Valider les Protocoles en Simulant"*, Conférence Invitée, Actes du Colloque Francophone sur l'Ingénerie des Protocoles, Eyrolles, 1988.

[Jard 89] C. Jard, J-M. Jézéquel. *"A Multi-Processor Estelle to C Compiler to Experiment Distributed Algorithms on Parallel Machines"*, 9th International Symposium on Protocol specification, Testing and Verification, Twente, Holanda, Junho, 1989.

[Jézéquel 91] J-M. Jézéquel, C. Jard. *"L'expérimentation d'Algorithmes Distribués sur Machines Parallèles avec ECHIDNA"*, Publicação Interna nº 603, IRISA, França, 1991.

[Le Lann 81] G. Le Lann. *"Motivations, Objectives and Caracterization of Distributed Systems"*, Lecture Notes in Computer Science, Distributed Systems, Architecure and Implementation, Springer-Verlag, pp. 1-8, 1981.

[Liebowitz 78] B. H. Liebowitz. *"Tutorial on Distributed Processing"*, IEEE, cap. 1, 1978.

[Linn 85] R. J. Linn. *"The Features and Facilities of Estelle"*, 5th Workshop on Protocol Specification, Testing and Verification, Moissac, Junho, 1985.

[Liskov 86] B. Liskov, J. Guttag. *"Abstraction and Specification in Program Development"*, MIT Press, 1986.

[Lloret 90] J. C. Lloret. *"Réseaux Prédicat/Transition Etiqueté pour la Modélisation et la Vérification de Systèmes Informatiques Répartis"*, Tese de Doutorado, Toulouse, Julho, 1990

[Martin 86] J-M. Martin. *"Réseaux de Donnés Abstraites et Application aux Protocoles de Signalisation"*, Tese do 3º Ciclo, Universidade Paul Sabatier, Toulouse, França, Julho, 1986.

[Mazzola 91] V. B. Mazzola. *"Contribution à la Conception de Systèmes Flexibles de Production: Application de la Technique de Description Formelle Estelle"*, Tese de Doutorado, Toulouse, Novembro, 1991.

[Milner 80] R. Milner. *"CCS, a Calculus for Communicating Systems"*, Lecture Notes in Computer Science, nº 92, Springer-Verlag, 1980.

[Murata 89] T. Murata. *"Petri Nets: Properties, Analysis and Applications"*, Proceedings of the IEEE, vol. 77, nº 4, Abril, 1989.

[Phalippou 88] M. Phalippou, R. Groz. *"Using Estelle for Verification. An Experience with the T.70 Teletex Transport Protocol"*, FORTE 88, Setembro, 1988.

[Pires 90] L. F. Pires, W. L. Souza. *"Step-wise Refinement Design Example Using LOTOS"*, Proceedings of the 3rd International Conference on Formal Description Techniques (FORTE 90), Madri, Espanha, pp. 289-306, 1990.

[Rayner 87] D. Rayner. *"OSI Conformance Testing"*, Computer Networks and ISDN Systems, 14, 1987, pp. 79-98.

[Rochkind 85] M. J. Rochkind. *"Unix: Programmation Avancée"*, Masson, 1985.

[Saqui 89a] P. de Saqui-Sannes, J-P Courtiat. *"ESTIM: The Estelle Simulator Prototype of the ESPRIT-SEDOS Project"*, Formal Description Techniques, North-Holland (Elsevier), 1989, pp. 15-29.

[Saqui 89b] P. de Saqui-Sannes, J-P. Courtiat. *"From the Simulation to the Verification of Estelle* Specifications"*, The 2nd International Conference on Formal Description Techniques (FORTE 89), Vancouver, pp. 1-12, 1989.

[Saqui 90a] P. de Saqui-Sannes. *"Prototypage d'un Environnement de Validation de Protocoles: Application à l'Approche Estelle"*, Tese de Doutorado, Toulouse, Abril, 1990.

[Saqui 90b] P. de Saqui-Sannes. *"The ESTIM User Manual"*, LAAS-CNRS, Novembro, 1990.

[Saqui 92] P. de Saqui-Sannes, C. Chassot. *"Implantation Automatique d'une Spécification Estelle en Environnement UNIX et TCP-IP"*, Publicação Interna, LAAS-CNRS, França, 1992.

[Schwartz 82] R. L. Schwartz, P. M. Melliar-Smith. *"From State Machine to Temporal Logic: Specification Methods for Protocol Standards"*, IEEE Transations on Communications, vol. COM-30, nº 12, pp. 2486-2496, Dezembro, 1982.

[Sidhu 89] D. P. Sidhu, A. Chung, T. Blumer. *"Experience with Formal Methods in Protocol Development"*, Computer Communication Review, pp. 81-101, 1989.

[Sidhu 90] D. P. Sidhu, T. P. Blumer. *"Semi-automatic Implementation of OSI Protocols"*, Computer Networks and ISDN Systems, 18, pp. 221-238, 1990.

[Silva 94] L. O. R. Silva. *"A Geração de Seqüências de Testes a Partir de Especificações Formais de Protocolos de Comunicação"*, Dissertação de Mestrado, CPGEEL-UFSC, Florianópolis-SC, em fase de conclusão, 1994.

[Singhal 91] M. Singhal, T. L. Casavant. *"Distributed Computer Systems"*, IEEE Computer, pp. 12-15, Agosto, 1991.

[SUN 90] SUN Microsystems. *"Network Programming Guide"*. cap. 10, 1990.

[Vissers 88] C. A. Vissers. *"FDTs for Open Systems - An ISO Perspective on FDTs"*, Invited Paper at the 1st International Conference on Formal Description Techniques, Stirling, Setembro, 1988.

[Vuong 88] S. T. Vuong, A. C. Lau, R. I. Chan. *"Semiautomatic Implementation of Protocols Using an Estelle-C Compiler"*, IEEE Transactions on Software Engineering, vol. 14, nº 3, pp. 384-393, Março, 1988.

[Wirsing 87] M. Wirsing, D. Sannella. *"Une Introduction à la Programmation Fonctionnelle: HOPE et ML"*, TSI Technology and Science of Informatics, vol. 6, nº 6, pp. 519-525, 1987.

[Zimmermann 80] H. Zimmermann. *"OSI Reference model: the ISO model of architecture for Open Systems Interconnection"*, IEEE Transactions on Communications, vol. COM-28, nº 4, pp. 425-432, Abril, 1980.

ANEXO I

INTERFACE GRÁFICA PARA AUXÍLIO NA UTILIZAÇÃO DAS FERRAMENTAS DE SUPORTE À ESTELLE

Visualmente, a interface compõe-se de duas telas, como pode ser observado na figura I.1. A tela superior é composta de um painel contendo um menu de escolha no qual o usuário possui a opção de selecionar a etapa de trabalho. De acordo com a opção selecionada, a tela inferior irá mostrar um dos três painéis de opções relativos às etapas citadas contando, cada qual, com os seus respectivos menus. O painel superior conta ainda com um botão (EXIT), no canto superior direito, através do qual o usuário pode abandonar a sessão. Nota-se, pela figura I.1, que a opção de edição é a opção *default* e já é mostrada na tela inicial. Na seqüência, serão apresentados os três painéis da tela inferior.

O PAINEL DE EDIÇÃO

No painel de edição o usuário tem a possibilidade de escolher entre três editores disponíveis: o *VI editor*, o *Emacs* e o editor de textos do *OpenWindows* (o *TextEditor*). Existe então um menu no qual o usuário seleciona o editor de preferência. Além deste menu, este painel contém também um botão (Especificação) onde o usuário pode editar uma nova especificação (opção NEW) ou carregar uma especificação já existente (opção OPEN), como mostra a figura I.1 a seguir.

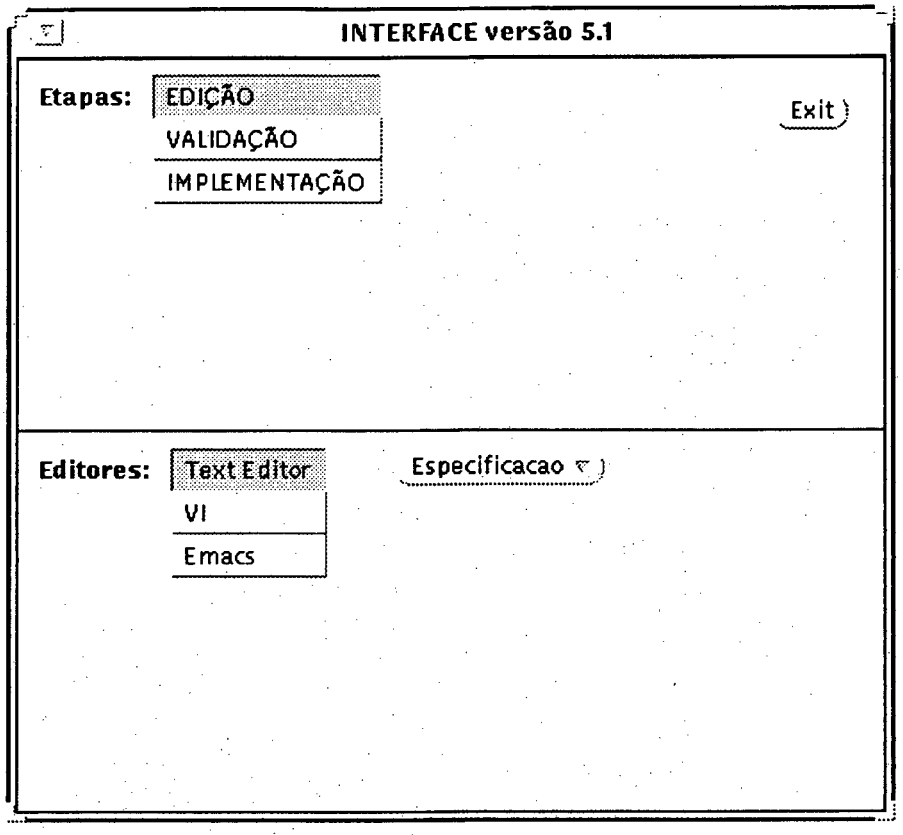


Figura I.1 - A Tela Inicial com o Painel (*default)* de Edição

O PAINEL DE VALIDAÇÃO

Este segundo painel é apresentado caso o usuário selecione a opção VALIDAÇÃO no painel superior. Neste caso será fornecido ao usuário um menu com três opções: a primeira opção é a de realizar a análise sintática de uma especificação descrita em Estelle*, utilizando a ferramenta GENESTIM; a segunda opção possibilita carregar uma especificação descrita em Estelle* através da ferramenta ESTIM, na qual o usuário tem as facilidades de simulação e verificação; por fim, é oferecida uma opção de simulação com a ferramenta EDB. Quando selecionada, esta opção fornece um *help* que auxilia na escolha das opções oferecidas pelo EDB.

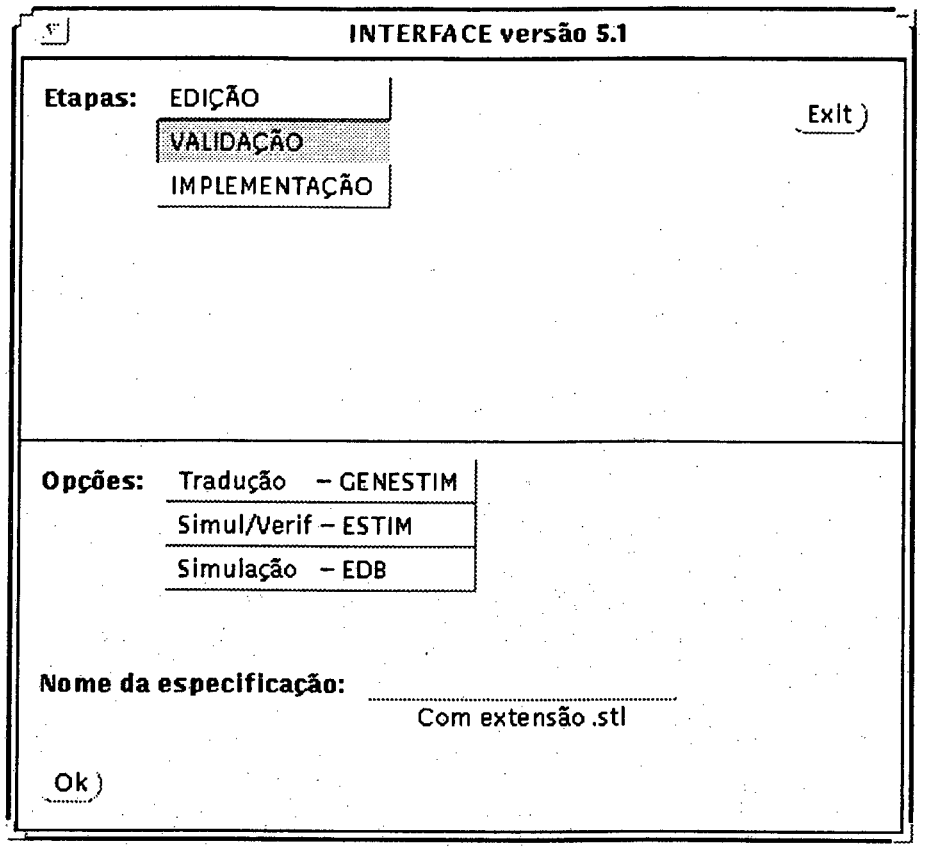


Figura I.2 - A Tela Principal e o Painel de Validação

Neste painel existe ainda um campo, no qual o usuário entra com o nome da especificação Estelle* desejada e um botão onde o usuário confirma a escolha selecionada. A figura I.2 mostra a tela principal com o painel VALIDAÇÃO.

O PAINEL DE IMPLEMENTAÇÃO

Este painel é apresentado na tela inferior caso o usuário selecione a opção IMPLEMENTAÇÃO da tela superior. Neste painel são oferecidas duas opções: a primeira é o acesso ao compilador Estelle-to-C (EC); ao ser selecionada esta opção é mostrado um *help* contendo as opções de translação e de geração de código que o usuário pode utilizar. A segunda opção é para a utilização da ferramenta Echidna para a geração de código; também neste caso é mostrado um *help* com as opções que tal ferramenta oferece ao usuário, como mostrado na figura I.3

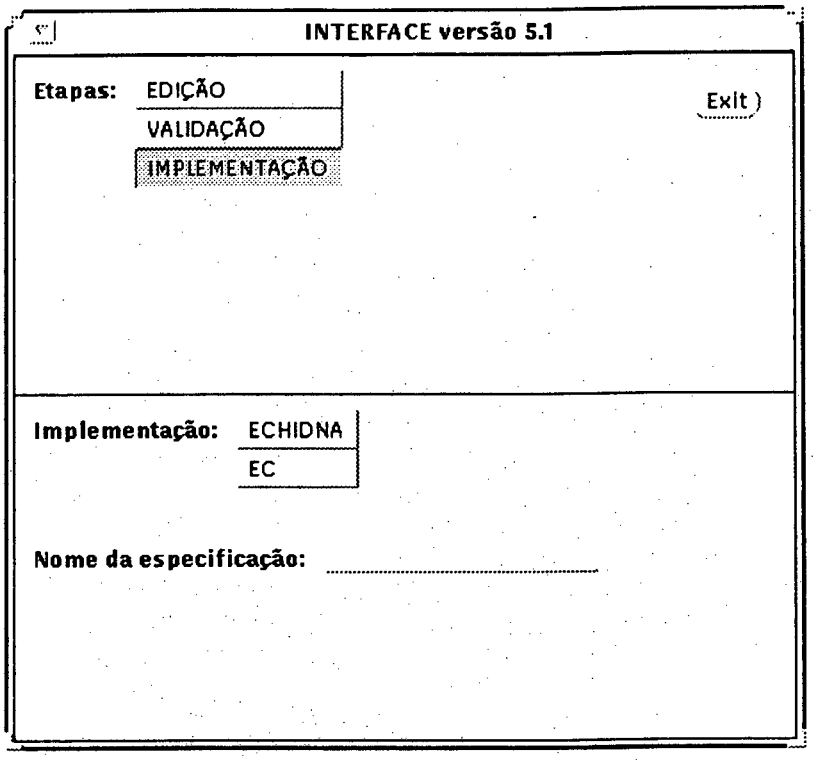


Figura I.3 - A Tela Principal e o Painel de Implementação

ANEXO II

A ESPECIFICAÇÃO ORIENTADA A MODELO DO SUPERVISOR DA TAREFA DE MONTAGEM

```
SPECIFICATION EspecOrientModelo SYSTEMACTIVITY;
default no queue;

TYPE
  Estadotarefa      = (A,B,AB,ABok,ABerr,EMPTY);


CHANNEL TAREFA_ROBOA (TRF,ROB);
  BY TRF: PlaceA; PlaceABok;

CHANNEL TAREFA_MESAGIR (TRF,MES);
  BY TRF: Rotate;

CHANNEL TAREFA_ROBOB (TRF,ROB);
  BY TRF: AssAB; UnloadABerr;

CHANNEL TAREFA_CAMERA (TRF,CMR);
  BY TRF: InspectAB;

CHANNEL TAREFA_ESTEIRA (TRF,EST);
  BY TRF: UnloadABok;


MODULE TaskType ACTIVITY;
  IP TroboA    : TAREFA_ROBOA(TRF);
     Tmesagir : TAREFA_MESAGIR(TRF);
     TroboB    : TAREFA_ROBOB(TRF);
     Tcamera   : TAREFA_CAMERA(TRF);
     Testeira  : TAREFA_ESTEIRA(TRF);
END;

MODULE RoboAType ACTIVITY;
  IP RoboAT: TAREFA_ROBOA(ROB);
END;

MODULE MesagirType ACTIVITY;
  IP MesagirT: TAREFA_MESAGIR(MES);
END;

MODULE RoboBType ACTIVITY;
  IP RoboBT: TAREFA_ROBOB(ROB);
END;

MODULE CameraType ACTIVITY;
  IP CameraT: TAREFA_CAMERA(CMR);
END;

MODULE EsteiraType ACTIVITY;
  IP EsteiraT: TAREFA_ESTEIRA(EST);
END;
```

```
BODY TaskBody FOR TaskType;

VAR
  MA,ME,CNV,LEMG,LDMG          : Estadotarefa;
  Inspecting                   : BOOLEAN;

INITIALIZE
  begin
    MA           := EMPTY;
    ME           := EMPTY;
    LEMG         := EMPTY;
    LDMG         := EMPTY;
    CNV          := EMPTY;
    Inspecting   := FALSE
  end;

TRANS

  PROVIDED (MA = EMPTY) AND (LDMG = EMPTY)
  name AlimentandoA:
    begin
          MA := A
    end;

  PROVIDED (ME = EMPTY) AND (LEMG = A)
  name AlimentandoB:
    begin
      ME := B
    end;

TRANS

{                          Placing A                                    }

  PROVIDED (MA = A) AND (LDMG = EMPTY)
  TroboA!PlaceA
          begin
            MA  := EMPTY;
            LDMG := A
          end;

{                              Rotating                                 }

  PROVIDED (LDMG IN [EMPTY,A]) AND (LEMG IN [EMPTY,ABok]) AND (LEMG <> LDMG) AND (MA = EMPTY)
  Tmesagir!Rotate
    VAR RPart : Estadotarefa;
          begin
            RPart := LDMG;
            LDMG  := LEMG;
            LEMG  := RPart
          end;

{                          Assembling AB                                }

  PROVIDED (ME = B) AND (LEMG = A)
  TroboB!AssAB
          begin
            ME  := EMPTY;
            LEMG := AB
          end;


{                          Inspecting AB                                }

  PROVIDED (LEMG = AB)  AND NOT Inspecting
  Tcamera!InspectAB
          begin
            Inspecting := TRUE
          end;
```

```
{                              Result of Inspecting                              }

  PROVIDED Inspecting  AND (LEMG = AB)
  name ResultingOK:
      begin
            LEMG                    := ABok;
            Inspecting        := FALSE
          end;

  PROVIDED Inspecting  AND (LEMG = AB)
  name ResultingER:
      begin
            LEMG              := ABer;
            Inspecting        := FALSE
          end;

{                              Unloading ABer                                    }

  PROVIDED (LEMG = ABerr)
  TroboB!UnloadABerr
          begin
           LEMG := EMPTY
          end;


{                              Unloading ABok                                    }

  PROVIDED (LDMG = ABok) AND (CNV = EMPTY)
  TroboA!PlaceABok
          begin
           LDMG := EMPTY;
           CNV := ABok
          end;

   PROVIDED (CNV = ABok)
   TEsteira!UnloadABok
          begin
           CNV := EMPTY
          end;

END;



{                              Corpos dos modulos do agente da celula            }

BODY RoboABody FOR RoboAType;
INITIALIZE
          begin end;
  TRANS
          RoboAT?PlaceA
                    begin
                    end;

          RoboAT?PlaceABok
                    begin
                    end;
END;


BODY MesagirBody FOR MesagirType;
INITIALIZE
          begin end;
  TRANS
           MesagirT?Rotate
                    begin
                    end;
END;
```

```
BODY RoboBBody FOR RoboBType;
INITIALIZE
        begin end;
 TRANS
         RoboBT?AssAB
                 begin
                 end;

          RoboBT?UnloadABerr
                 begin
                 end;
END;


BODY CameraBody FOR CameraType;
INITIALIZE
        begin end;
 TRANS
        CameraT?InspectAB
                 begin
                 end;
END;


BODY EsteiraBody FOR EsteiraType;
INITIALIZE
        begin end;
 TRANS
          EsteiraT?UnloadABok
                 begin
                 end;
END;


MODVAR
 Task    : TaskType;
 RoboA      : RoboAType;
 Mesagir          : MesagirType;
 RoboB : RoboBType;
 Camera           : CameraType;
 Esteira : EsteiraType;

INITIALIZE
 BEGIN
   INIT Task      WITH TaskBody;
   INIT RoboA    WITH RoboABody;
   INIT Mesagir  WITH MesagirBody;
   INIT RoboB    WITH RoboBBody;
   INIT Camera  WITH CameraBody;
   INIT Esteira   WITH EsteiraBody;
   CONNECT Task.TRoboA     TO RoboA.RoboaT;
   CONNECT Task.TMesagir   TO Mesagir.MesagirT;
   CONNECT Task.TRoboB     TO RoboB.RoboBT;
   CONNECT Task.TCamera   TO Camera.CameraT;
   CONNECT Task.TEsteira   TO Esteira.EsteiraT;
 END;

END. { EspecOrientModelo }
```

ANEXO III

A ESPECIFICAÇÃO DETALHADA DA TAREFA DE MONTAGEM

```
SPECIFICATION EspecDetalhada SYSTEMACTIVITY;
default individual queue;

TYPE
  Estadotarefa = (A, B, AB, ABok, ABerr, EMPTY);

CHANNEL TAREFA_ROBOA (TRF,ROB);
  BY TRF: PlaceA; PlaceABok;
  BY ROB: APlaced; ABokPlaced;

CHANNEL TAREFA_MESAGIR (TRF,MGR);
  BY TRF: Rotate;
  BY MGR: Rotated;

CHANNEL TAREFA_ROBOB (TRF,ROB);
  BY TRF: AssAB; UnloadABerr;
  BY ROB: ABAssembled; ABerrUnloaded;

CHANNEL TAREFA_CAMERA (TRF,CMR);
  BY TRF: InspectAB;
  BY CMR: ABInspectedOk; ABInspectedErr;

CHANNEL TAREFA_ESTEIRA (TRF,EST);
  BY TRF: UnloadABok;
  BY EST: ABokUnloaded;


MODULE TaskType ACTIVITY;
  IP TroboA:        TAREFA_ROBOA(TRF);
    TmesaGir:       TAREFA_MESAGIR(TRF);
    TroboB:         TAREFA_ROBOB(TRF);
    Tcamera:        TAREFA_CAMERA(TRF);
    Testeira:       TAREFA_ESTEIRA(TRF);
END;

MODULE RoboAType ACTIVITY;
  IP RoboAT: TAREFA_ROBOA(ROB);
END;

MODULE MesaGirType ACTIVITY;
  IP MesaGirT: TAREFA_MESAGIR(MGR);
END;

MODULE RoboBType ACTIVITY;
  IP RoboBT: TAREFA_ROBOB(ROB);
END;

MODULE CameraType ACTIVITY;
  IP CameraT: TAREFA_CAMERA(CMR);
END;

MODULE EsteiraType ACTIVITY;
  IP EsteiraT: TAREFA_ESTEIRA(EST);
END;
```

```
BODY TaskBody FOR TaskType;

VAR
        MA,MB,LDMG,LEMG,EST          : Estadotarefa;
            {MA => MesaA; MB => MesaB; LDMG => LadoDireitoMesaGiratoria; LEMG => LadoEsquerdoMesaGiratoria}
        Controle_robo_A,
        Controle_mesaGir,
        Controle_robo_B,
        Controle_camera,
        Controle_esteira                 :BOOLEAN;

INITIALIZE
  begin
        MA                               := EMPTY;
        MB                               := EMPTY;
        LEMG                             := EMPTY;
        LDMG                             := EMPTY;
        EST                              := EMPTY;
        Inspecting                       := FALSE;
        Controle_robo_A                  := FALSE;
        Controle_mesaGir                 := FALSE;
        Controle_robo_B                  := FALSE;
        Controle_camera                  := FALSE;
        Controle_esteira                 := FALSE
 end;

TRANS

  PROVIDED (MA = EMPTY) AND (LDMG = EMPTY)
        begin
          MA := A
        end;

  PROVIDED (MB = EMPTY) AND (LEMG = A)
   begin
     MB := B
   end;

TRANS

{               Coloca Peca A no Lado Direito da Mesa Giratoria                  }

  PROVIDED (MA = A) AND (LDMG = EMPTY) AND NOT Controle_robo_A
        begin
                Controle_robo_A := TRUE;
                output TroboA.PlaceA
        end;

  When TroboA.APlaced
        begin
                Controle_robo_A := FALSE;
                MA  := EMPTY;
                LDMG := A
        end;


{                       Rotaciona a Mesa Giratoria                              }

 PROVIDED (LDMG IN [EMPTY,A]) AND (LEMG IN [EMPTY,ABok]) AND (LEMG <> LDMG) AND (MA = EMPTY) AND NOT
Controle_mesaGir
        begin
                Controle_mesaGir := TRUE;
                output TMesaGir.Rotate
        end;

 When TMesaGir.Rotated
    VAR Aux_troca : Estadotarefa;
        begin
                Controle_mesaGir := FALSE;
                Aux_troca := LDMG;
                LDMG   := LEMG;
                LEMG   := Aux_troca
        end;
```

```
{                                   Montagem da Peca AB                                   }

  PROVIDED (MB = B) AND (LEMG = A) AND NOT Controle_robo_B
          begin
                  Controle_robo_B := TRUE;
                  output TRoboB.AssAB
          end;

  When TRoboB.ABAssembled
          begin
                  Controle_robo_B := FALSE;
                  MB  := EMPTY;
                  LEMG := AB
          end;

{                                   Inspecao Peca AB Montada                              }

  PROVIDED (LEMG = AB) AND NOT Controle_camera
          begin
                  Controle_camera := TRUE;
                  output TCamera.InspectAB;
          end;

  When TCamera.ABInspectedOk
          begin
                  Controle_camera := FALSE;
                  LEMG := ABok
          end;

  When TCamera.ABInspectedErr
          begin
                  Controle_camera := FALSE;
                  LEMG := ABerr
          end;

{                                   Descarregamento de Pecas ABer                         }

  PROVIDED (LEMG = ABerr) AND NOT Controle_robo_B
          begin
                  Controle_robo_B := TRUE;
                  output TRoboB.UnloadABerr
          end;



  When TRoboB.ABerrUnloaded
          begin
                  Controle_robo_B := FALSE;
                  LEMG := EMPTY
          end;

{                                   Descarregamento de Pecas ABok                         }

  PROVIDED (LDMG = ABok) AND (EST = EMPTY) AND NOT Controle_robo_A
          begin
                  Controle_robo_A := TRUE;
                  output TRoboA.PlaceABok
          end;

  When TRoboA.ABokPlaced
          begin
                  Controle_robo_A := FALSE;
                  LDMG := EMPTY;
                  EST := ABok
          end;

  PROVIDED (EST = ABok) AND NOT Controle_esteira
          begin
                  Controle_esteira := TRUE;
                  output TEsteira.UnloadABok
          end;
```

```
  When TEsteira.ABokUnloaded
            begin
                        Controle_esteira := FALSE;
                        EST := EMPTY
            end;

END;

BODY RoboABody FOR RoboAType;
INITIALIZE
            begin end;

TRANS

  When RoboAT.PlaceA
            begin
                        output RoboAT.APlaced
            end;

  When RoboAT.PlaceABok
            begin
                        output RoboAT.ABokPlaced
            end;
END;

BODY MesaGirBody FOR MesaGirType;
INITIALIZE
            begin end;

TRANS

  When MesaGirT.Rotate
            begin
                        output MesaGirT.Rotated
            end;
END;

BODY RoboBBody FOR RoboBType;
INITIALIZE
            begin end;

TRANS
  When RoboBT.AssAB
            begin
                        output RoboBT.ABAssembled
            end;
  When RoboBT.UnloadABerr
            begin
                        output RoboBT.ABerrUnloaded
            end;
END;

BODY CameraBody FOR CameraType;
INITIALIZE
            begin end;

TRANS
  When CameraT.InspectAB
            begin
                        output CameraT.ABInspectedOk
            end;
            begin
                        output CameraT.ABInspectedErr
            end;
END;
```

```
BODY EsteiraBody FOR EsteiraType;
INITIALIZE
        begin end;

TRANS
 When EsteiraT.UnloadABok
        begin
                output EsteiraT.ABokUnloaded
        end;
END;
MODVAR
  Task              : TaskType;
  RoboA    : RoboAType;
  MesaGir           : MesaGirType;
  RoboB             : RoboBType;
  Camera            : CameraType;
  Esteira           : EsteiraType;

INITIALIZE
  BEGIN
    INIT Tarefa     WITH TarefaBody;
    INIT RoboA      WITH RoboABody;
    INIT MesaGir    WITH MesaGirBody;
    INIT RoboB      WITH RoboBBody;
    INIT Camera     WITH CameraBody;
    INIT Esteira    WITH EsteiraBody;
    CONNECT Tarefa.TRoboA     TO RoboA.RoboAT;
    CONNECT Tarefa.TMesaGir   TO MesaGir.MesaGirT;
    CONNECT Tarefa.TRoboB     TO RoboB.RoboBT;
    CONNECT Tarefa.TCamera    TO Camera.CameraT;
    CONNECT Tarefa.TEsteira   TO Esteira.EsteiraT;
  END;

END. { EspecDetalhada }
```

ANEXO IV

A BIBLIOTECA DE FUNÇÕES PARA A UTILIZAÇÃO DOS SOQUETES UNIX

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/param.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/time.h>
#define MAXHOSTNAME 10

int
cria_soquete(numero)
unsigned short  numero;
{
        char myname[MAXHOSTNAME+1];
        struct sockaddr_in num_port_real ;
        struct hostent *hp ;
        int         s;
        if ((s = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
                perror ("erro na abertura do soquete");
                exit(1);
        };
        gethostname(myname,MAXHOSTNAME);
        hp = gethostbyname(myname);
        bzero((char *) &num_port_real, sizeof(num_port_real));
        bcopy(hp->h_addr,&(num_port_real.sin_addr.s_addr),hp->h_length) ;
        num_port_real.sin_port = 5000 + numero;
        num_port_real.sin_family=hp->h_addrtype;
        if (bind(s, &num_port_real, sizeof(num_port_real)) < 0) {
                perror("erro na associacao do nome ao soquete");
                exit(1);
        }
        return (s);
}


char*
recebe_mensagem(s, buffer)
int         s;
char        *buffer;
{
        if (read(s, (char *)buffer, 100) < 0)
                perror("erro na leitura do datagrama");
        return(buffer);
}
```

```
void
envia_mensagem(s, num_port_destino, estacao_destino, data, datalen)
int                     s;
unsigned short          num_port_destino;
char                    estacao_destino[MAXHOSTNAME+1];
char                    *data;
int                     datalen;
{
        struct sockaddr_in endereco_destino;
        char    hostname[MAXHOSTNAME + 1];
        struct hostent *hp, *gethostbyname();

        hp = gethostbyname(estacao_destino);
        bzero((char *) &endereco_destino, sizeof(endereco_destino));
        bcopy(hp->h_addr,&(endereco_destino.sin_addr.s_addr),hp->h_length);
        endereco_destino.sin_port = 5000 + num_port_destino;
        endereco_destino.sin_family = hp->h_addrtype;

        sleep(1);
        if (sendto(s, data, sizeof(data), 0, &endereco_destino,
                sizeof(endereco_destino)) < 0) {
                perror("erro no envio do datagrama");
                exit(1);
        }
}


int
soquete_nao_vazio(s)
        int     s;
{
        int se;
        fd_set mask;
        struct timeval delay;

        FD_ZERO(&mask);
        FD_SET(s,&mask);
        delay.tv_usec = 100000;
        if (se=select(s + 1,&mask,NULL,NULL,&delay)>0)
          {
                return(1);
          }
          else return(0);
}
```

ANEXO V

A ESPECIFICAÇÃO DO SUPERVISOR DA TAREFA DE MONTAGEM PREPARADA PARA A IMPLEMENTAÇÃO

```
SPECIFICATION Superv SYSTEMACTIVITY;
default individual queue;
timescale seconds;

FUNCTION soquete_nao_vazio(s: integer): boolean; primitive;

TYPE
        Estadotarefa = (A, B, AB, ABok, ABerr, EMPTY);
        socket_type = integer;
        data_type = (ColocaA, AColocada, ColocaABok, ABokColocada, RotacionaMesaGir, MesaGirRotacionada,
MontaAB, ABMontada, InspecionaAB, ABResultOk, ABResultErr, DescarregaABerr, ABerrDescarregada,
DescarregaABok, ABokDescarregada);

CHANNEL TAREFA_ESPECIFICACAO (TRF,ESP);
  BY TRF:  PlaceA; PlaceABok; Rotate; AssAB; UnloadABerr; InspectAB; UnloadABok;
  BY ESP:  APlaced; ABokPlaced; Rotated; ABAssembled; ABerrUnloaded; ABInspectedOk;
        ABInspectedErr; ABokUnloaded;

MODULE TarefaType ACTIVITY;
  IP Task:  TAREFA_ESPECIFICACAO(TRF);
END;

BODY TarefaBody FOR TarefaType;
VAR
        MA,MB,LDMG,LEMG,EST                     : Estadotarefa;
            {  MA => MesaA; MB => MesaB; LDMG => LugarDireitoMesaGir; LEMG => LugarEsquerdoMesaGir  }
        Controle_robo_A,
        Controle_mesaGir,
        Controle_robo_B,
        Controle_camera,
        Controle_esteira                :BOOLEAN;

INITIALIZE
  begin
        MA                              := EMPTY;
        MB                              := EMPTY;
        LEMG                            := EMPTY;
        LDMG                            := EMPTY;
        EST                             := EMPTY;
        Inspecting                      := FALSE;
        Controle_robo_A                 := FALSE;
        Controle_mesaGir                := FALSE;
        Controle_robo_B                 := FALSE;
        Controle_camera                 := FALSE;
        Controle_esteira                := FALSE
  end;

TRANS
  PROVIDED (MA = EMPTY) AND (LDMG = EMPTY)
        begin
                (*$C$ printf("\nORDEM: Alimentar Mesa A ==> Peca A\n"); *)
                (*$C$ sleep(3); *)
                MA := A
        end;
```

```
TRANS
 PROVIDED (MB = EMPTY) AND (LEMG = A)
          var cont2:integer;
          begin
                    (*$C$ printf("\nORDEM: Alimentar Mesa B ==> Peca B\n"); *)
                    (*$C$ sleep(3); *)
                    MB := B
          end;

TRANS

{                                   Coloca Peca A no Lado Direito da Mesa Giratoria                                    }

 PROVIDED (MA = A) AND (LDMG = EMPTY) AND NOT Controle_robo_a
          begin
                    (*$C$ printf("\nORDEM: RoboA Colocar Peca A na Mesa Giratoria\n"); *)
                    Controle_robo_a := TRUE;
                    output Task.PlaceA
          end;

 When Task.APlaced
          begin
                    (*$C$ printf("\nACK  : Peca A Colocada na Mesa Giratoria\n"); *)
                    Controle_robo_a := FALSE;
                    MA  := EMPTY;
                    LDMG := A
          end;

{                                   Rotaciona a Mesa Giratoria                                                        }

 PROVIDED (LDMG IN [EMPTY,A]) AND (LEMG IN [EMPTY,ABok]) AND (LEMG <> LDMG) AND (MA = EMPTY) AND NOT
Controle_mesaGir
          begin
                    (*$C$ printf("\nORDEM: Rotacionar a Mesa Giratoria\n"); *)
                    Controle_mesaGir := TRUE;
                    output Task.Rotate
          end;

 When Task.Rotated
   VAR Aux_troca : Estadotarefa;
          begin
                    (*$C$ printf("\nACK  : Mesa Giratoria Rotacionada\n"); *)
                    Controle_mesaGir := FALSE;
                    Aux_troca := LDMG;
                    LDMG  := LEMG;
                    LEMG  := Aux_troca
          end;

{                                   Montagem da Peca AB                                                               }

 PROVIDED (MB = B) AND (LEMG = A) AND NOT Controle_robo_b
          begin
                    (*$C$ printf("\nORDEM: RoboB Montar Peca AB\n"); *)
                    Controle_robo_b := TRUE;
                    output Task.AssAB
          end;
 When Task.ABAssembled
          begin
                    (*$C$ printf("\nACK  : Peca AB Montada\n"); *)
                    Controle_robo_b := FALSE;
                    MB  := EMPTY;
                    LEMG := AB
          end;

{                                   Inspecao Peca AB Montada                                                          }

 PROVIDED (LEMG = AB)  AND NOT Inspecting AND NOT Controle_camera
          begin
                    (*$C$ printf("\nORDEM: Camera Inspecionar Peca AB\n"); *)
                    Controle_camera := TRUE;
                    output Task.InspectAB;
          end;
```

```
  When Task.ABInspectedOk
        begin
              (*$C$ printf("\nACK  : Inspecao Peca AB Resultou OK\n"); *)
              Controle_camera := FALSE;
              LEMG := ABok
        end;

  When Task.ABInspectedErr
        begin
              (*$C$ printf("\nACK  : Inspecao Peca AB Resultou Erro\n"); *)
              Controle_camera := FALSE;
              LEMG := ABerr
        end;

{                          Descarregamento de Pecas ABer                          }

  PROVIDED (LEMG = ABerr) AND NOT Controle_robo_b
        begin
              (*$C$ printf("\nORDEM: RoboB Descarregar Peca ABErr\n"); *)
              Controle_robo_b := TRUE;
              output Task.UnloadABerr
        end;

  When Task.ABerrUnloaded
        begin
              (*$C$ printf("\nACK  : Peca ABErr Descarregada\n"); *)
              Controle_robo_b := FALSE;
              LEMG := EMPTY
        end;
{                          Descarregamento de Pecas ABok                          }

  PROVIDED (LDMG = ABok) AND (EST = EMPTY) AND NOT Controle_robo_a
        begin
              (*$C$ printf("\nORDEM: RoboA Colocar Peca ABOk na Esteira\n"); *)
              Controle_robo_a := TRUE;
              output Task.PlaceABok
        end;

  When Task.ABokPlaced
        begin
              (*$C$ printf("\nACK  : Peca ABOk Colocada na Esteira\n"); *)
              Controle_robo_a := FALSE;
              LDMG := EMPTY;
              EST := ABok
        end;

  PROVIDED (EST = ABok) AND NOT Controle_esteira
        begin
              (*$C$ printf("\nORDEM: Esteira Descarregar Peca ABOk\n"); *)
              Controle_esteira := TRUE;
              output Task.UnloadABok
        end;

  When Task.ABokUnloaded
        begin
              (*$C$ printf("\nACK  : Peca ABOk na Esteira Descarregada\n"); *)
              Controle_esteira := FALSE;
              EST := EMPTY
        end;

END;

MODVAR
  Tarefa    : TarefaType;
```

```
{                                   NIVEL  RAIZ DA ESPECIFICACAO                                    }

IP          Spec:  TAREFA_ESPECIFICACAO(ESP);

VAR         num_soquete             : integer;
            num_port_destino        : integer;
            estacao_destino         : array[1..11] of char;
            soq_leitura             : socket_type;
            soq_envio               : socket_type;
            data                    : data_type;
STATE
            OCIOSO, ENVIA_ORDEM, DEADLOCK;

INITIALIZE to OCIOSO
            begin
                        soq_leitura  := 10;
                        (*$C$ _adctx->num_soquete = cria_soquete(_adctx->soq_leitura); *)
            end;

trans

provided (num_soquete <> -1)
  from OCIOSO to ENVIA_ORDEM
            begin
              INIT   Tarefa      WITH TarefaBody;
              CONNECT Tarefa.Task  TO  Spec;
            end;

provided otherwise
  from OCIOSO to DEADLOCK
            begin
            (*$C$ printf("Falha na criacao dos soquetes\n"); *)
            (*$C$ exit(1); *)
            end;

trans
when Spec.PlaceA
  from ENVIA_ORDEM to same
            begin
                        data := ColocaA;
                        num_port_destino := 10;
                        (*$C$ strcpy(_adctx->estacao_destino, "antares"); *)
                        (*$C$    envia_dado(_adctx->num_soquete,   _adctx->num_port_destino,   _adctx->estacao_destino,
&_adctx->data, sizeof(_adctx->data)); *)
            end;

trans
when Spec.PlaceABok
  from ENVIA_ORDEM to same
            begin
                        data := ColocaABok;
                        num_port_destino := 10;
                        (*$C$ strcpy(_adctx->estacao_destino, "antares"); *)
                        (*$C$    envia_dado(_adctx->num_soquete,   _adctx->num_port_destino,   _adctx->estacao_destino,
&_adctx->data, sizeof(_adctx->data)); *)
            end;

trans
when Spec.Rotate
  from ENVIA_ORDEM to same
            begin
                        data := RotacionaMesaGir;
                        num_port_destino := 20;
                        (*$C$ strcpy(_adctx->estacao_destino, "bellatrix"); *)
                        (*$C$    envia_dado(_adctx->num_soquete,   _adctx->num_port_destino,   _adctx->estacao_destino,
&_adctx->data, sizeof(_adctx->data)); *)
            end;
```

```
trans
when Spec.AssAB
  from ENVIA_ORDEM to same
        begin
                data := MontaAB;
                num_port_destino := 30;
                (*$C$ strcpy(_adctx->estacao_destino, "sirius"); *)
                (*$C$   envia_dado(_adctx->num_soquete,  _adctx->num_port_destino,  _adctx->estacao_destino,
&_adctx->data, sizeof(_adctx->data)); *)
        end;


trans
when Spec.UnloadABerr
  from ENVIA_ORDEM to same
        begin
                data := DescarregaABerr;
                num_port_destino := 30;
                (*$C$ strcpy(_adctx->estacao_destino, "sirius"); *)
                (*$C$   envia_dado(_adctx->num_soquete,  _adctx->num_port_destino,  _adctx->estacao_destino,
&_adctx->data, sizeof(_adctx->data)); *)
        end;

trans
when Spec.InspectAB
  from ENVIA_ORDEM to same
        begin
                data := InspecionaAB;
                num_port_destino := 40;
                (*$C$ strcpy(_adctx->estacao_destino, "atlas"); *)
                (*$C$   envia_dado(_adctx->num_soquete,  _adctx->num_port_destino,  _adctx->estacao_destino,
&_adctx->data, sizeof(_adctx->data)); *)
        end;

trans
when Spec.UnloadABok
  from ENVIA_ORDEM to same
        begin
                data := DescarregaABok;
                num_port_destino := 50;
                (*$C$ strcpy(_adctx->estacao_destino, "talitha"); *)
                (*$C$   envia_dado(_adctx->num_soquete,  _adctx->num_port_destino,  _adctx->estacao_destino,
&_adctx->data, sizeof(_adctx->data)); *)
        end;

trans
  from ENVIA_ORDEM to same
        provided soquete_nao_vazio(num_soquete)
        begin
                (*$C$ recebe_dado(_adctx->num_soquete, &_adctx->data); *)
                case data of
                        AColocada               : output Spec.APlaced;
                        ABokColocada            : output Spec.ABokPlaced;
                        MesaGirRotacionada      : output Spec.Rotated;
                        ABMontada               : output Spec.ABAssembled;
                        ABerrDescarregada       : output Spec.ABerrUnloaded;
                        ABResultOk              : output Spec.ABInspectedOk;
                        ABResultErr             : output Spec.ABInspectedErr;
                        ABOkDescarregada        : output Spec.ABOkUnloaded;
                end;
        end;

END. { Superv }
```