59197-0

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

SEMÂNTICA E UMA FERRAMENTA PARA O MÉTODO SADT

por

Adagenor Lobato Ribeiro

Dissertação submetida como requisito parcial
para a obtenção do grau de Mestre em
Ciência da Computação

Prof. Dr. Daltro José Nunes
Orientador

Porto Alegre, outubro de 1991

UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Ribeiro, Adagenor Lobato

Semântica e uma ferramenta para o método SADT/Adagenor Lobato Ribeiro – Porto Alegre: CPGCC da UFRGS, 1991.
190 p. : il.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Sul, Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, 1991. Orientador: Nunes, Daltro José.

Dissertação: Método, Engenharia de Software, SADT, Definição de requisitos, Especificação de software, Case, Simulação, VDM, Ambientes de desenvolvimento de software, Semântica.

" Bem-aventurados os pobres em espírito
porque deles é o Reino dos Céus.

Bem aventurados os *humildes*
porque *herdarão* a terra.

Bem aventurados os *aflitos*
porque serão consolados.

Bem aventurados os que tem fome e sede de justiça,
porque serão saciados.

Bem aventurados os misericordiosos,
porque alcançarão misericórdia.

Bem aventurados os *puros de coração*,
porque verão a Deus.

Bem aventurado os que promovem a paz
porque serão chamados filhos de Deus.

Bem aventurados os perseguidos por causa da justiça,
porque deles é o Reino dos Céus.

Bem aventurados sois, quando vos injuriarem e vos perseguirem e, mentindo, disserem todo o mal contra vós por causa de mim. Alegrai-vos e regozijai-vos, porque será grande a vossa recompensa nos Céus, pois foi assim que perseguiram os profetas, que vieram antes de vós. "

Mt 5,3-12

À minha família, aos diletos amigos e a Lena

AGRADECIMENTOS

A Universidade Federal do Pará, através de seu Departamento de Informática, pela liberação e apoio financeiro.

A Universidade Federal do Rio Grande do Sul, pela aceitação de meu nome para integrar a turma do mestrado de 1988, bem como pelo ensino ministrado e a oportunidade de realização de pesquisa na área de informática aplicada.

Ao MEC-SESu que através da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES nos proporcionou auxílio financeiro, recebido em forma de bolsa, durante a realização do curso.

Em especial ao meu orientador Prof. Dr. Daltro José Nunes, pelo apoio, amizade, empenho, dedicação, compreensão e pelas sugestões valiosas que muito me auxiliaram na elaboração desta dissertação.

Aos professores e funcionários do Curso de Pós-Graduação em Ciência da Computação da UFRGS.

Aos colegas do Projeto PROSOFT Paulo do Carmo, Fabrício, Cláudia, Leila Ribeiro, Francisco, Ausberto, Luís, André Caldas, e Daniel Wobeto.

Ao Altino Pavan, um grande auxiliar de pesquisa e excelente pessoa humana, que trabalhou comigo no âmbito do Projeto PROSOFT, não medindo dificuldades, atuando com dedicação e incansávelmente em todos os momentos.

A todos os colegas do PGCC da UFRGS, destacando: Marcelo Pimenta, Ana Tereza Martins, Eloi Favero, Ana Bazan, Paulo Barrela, Roberta Gomes, Denise, Janete, Manoel Menezes, Liliana, Cláudia, Augustini,Rosana, Edson, Maria, Xavier, Laerte, Chico, Aliomar, Javan, Gladimir, Flavinho, Márcio,Rui, Ana Lizete, João Paulo Kitajima, Benhur, Eduardo Peres, Heriberto, Katia, Perin, Marco, Nina Edelweiss, Rafael, Renata, Dotti, Campani, Yara e Ewerton.

Ao Professor Benedito Melo Acioly da Universidade Federal de Pernambuco, a Professora Graçaliz Pereira Dimuro da Universidade Católica de Pelotas e ao Professor Luiz Antonio de Moro Palazzo da Universidade Federal de Pelotas, pelas sugestões, incentivo, companheirismo, amizade e saudável convívio.

A Professora Conceição Rangel Fiúza de Melo, da Universidade Federal do Pará pelo incentivo, apoio e pelo seu papel de motivadora, para que seus alunos busquem realizar cursos avançados, e acima disso agradeço seu empenho na defesa de meus interesses junto ao Departamento de Informática da UFPA.

Ao Júlio Correa dos Santos e família, pela sua amizade, incentivo e convívio. Este agradecimento é uma forma de guardar um grande amigo que fica na fria e as vêzes muito quente cidade de Porto Alegre.

A Suzi Reis Westphal do Citibank, pelo tratamento sempre cordial, pela amizade, força e acima de tudo pelo ser humano que é.

A minha família em especial a meus pais, Celina Lobato Ribeiro e Osvaldino do Nascimento Ribeiro, pessoas de quem pelas contingências da vida, tenho subtraído tempo de convívio deste o curso primário. Agradeço por tudo, principalmente pelo amor que representa a unidade maior entre nós.

As bibliotecárias Tânia Fraga e Margarida Schmit, que durante todo o curso, sempre nos atenderam com prestesa e dedicação.

Ao meu amigo Prof. Dr. Luís Carlos Lobato Botelho, que com seu exemplo de vida acadêmica, também me levou para esse universo que é o conhecimento.

Ao meu amigo Sérgio Façanha da Silva, pelo apoio, incentivo e principalmente pela sólida amizade ao longo dos últimos doze anos.

Ao Prof. Dr. José Seixas Lourenço, ex-reitor da Universidade Federal do Pará, que mesmo desejando meu nome para chefia do Departamento de Informática da UFPA, permitiu minha liberação para a realização do curso de mestrado.

Aos amigos Rubens Donati Jorge e Stela Faciola, pessoas com as quais tive oportunidade de trabalhar em prol da evolução da informática na educação do Estado do Pará.

Ao Engenheiro Sergio Mendes, do Centro de Estudos Superiores do Estado do Pará - CESUPA, pela sua amizade, seu idealismo, vontade de construir e por sua inabalável crença no crescimento deste país apartir das bases educacionais.

Aos membros do comitê de leitura desta dissertação: Profa. Dra. Ana Maria de Alencar Price - UFRGS, Prof. Dr. José Palazzo Moreira de Oliveira - UFRGS e Prof. Dr. Roberto da Silva Bigonha - UFMG.

A Eliene Sulzbacher pela amizade e apoio logístico durante a realização desta dissertação.

Ao Prof. Celso Maciel da Costa pela confiança, idealismo e amizade.

A Lena pela sua compreensão, pelo seu carinho, pelo seu apoio espiritual, pelo seu amor.

A DEUS, Alfa e Ômega, *O todo poderoso*, que possibilitou a minha existência, a minha inteligência e os meios para prosseguir nos caminhos da minha vida. Agradeço, e que eu possa usar os dons que me foram dados, para multiplicar a paz, o amor e o conhecimento na terra entre os seres humanos.

SUMÁRIO

1 INTRODUÇÃO 19

1.1 A definição e especificação de requisitos de software 19

1.2 Métodos & semânticas 21

1.3 O método SADT como ferramenta de apoio a definição de requisitos . 22

1.4 Propósitos de uma ferramenta para engenharia de software 22

1.5 Pontos e contextos na ferramenta SADT 23

1.6 A dissertação no âmbito do PROSOFT 23

1.7 A parte teórica na dissertação 24

1.8 Trabalhos relacionados 24

1.9 Organização da dissertação 29

2 O MÉTODO SADT 31

2.1 Origens do método 31

2.2 Modelo do sistema, propósito e ponto de vista 33

2.3 Validação do modelo do sistema 34

2.4 A linguagem gráfica do método: (SA) 39

2.4.1 A primitiva básica da linguagem gráfica 42

2.4.2 A conectividade no diagrama 44

2.4.3 Dualidade de dados e atividades 48

2.4.4 Dominância entre caixas 49

2.4.5 Fronteiras em diagramas SADT 50

2.4.6 A interconexão de uma coleção de diagramas 51

2.4.7 O conceito de mecanismos 52

2.5 A Técnica de projeto: (DT) 54

2.6 Análise, projeto e implementação usando SADT 55

2.7 SADT x SOFTech 57

2.8 Regra de ativação 58

2.9 SADT interfaceado com outros métodos 65

2.10 Considerações finais 65

3 A INTER-RELAÇÃO DO SADT COM SISTEMAS DE FLUXO DE DADOS 69

3.1 A importância da semântica para o método 69

3.2 Extensões necessárias para "executar" o SADT 71

3.2.1 Tipo de dado como um "token" 74

3.2.2 O modelo "dataflow machines" 74

3.2.3 SADT e "dataflow machines" 76

3.2.4 Regras de ativação para uma atividade 77

3.3 A simulação discreta no modelo SADT 78

4 ABORDAGEM OPERACIONAL PARA CONECTIVOS E ATIVIDADES **81**

4.1 Construtos do método & semântica 81

4.2 A linguagem PSDL na abordagem operacional dos construtos 83

4.3 A definição da semântica dos conectivos 84

4.3.1 Definição semântica do conectivo BRANCH 85

4.3.2 Descrição do significado de BRANCH 85

4.3.3 Definição semântica do conectivo JOIN 86

4.3.4 Descrição do significado de JOIN 87

4.3.5 Definição da semântica do conectivo SPREAD 87

4.3.6 Descrição do significado de SPREAD 88

4.3.7 Definição semântica do conectivo BUNDLE 89

4.3.8 Descrição do significado de BUNDLE 90

4.3.9 Definição semântica conectivo ORBRANCH 90

4.3.10 Descrição do significado de ORBRANCH 91

4.3.11 Definição semântica do conectivo ORJOIN 92

4.3.12 Descrição do significado de ORJOIN 94

4.4 A definição semântica de atividade 94

4.4.1 Estados de uma atividade 96

4.4.2 Atividade produzindo saída 96

4.4.3 Atividade em processamento 97

4.4.4 Atividade pronta para disparar 98

4.5 Diagrama utilizando construtos do método SADT 101

4.6 Comentários e observações 102

5 A SIMULAÇÃO DA EXECUÇÃO DE ESPECIFICAÇÕES SADT 105

5.1 Sistemas & Modelos 105

5.2 A construção de modelos para simulação 109

5.3 A construção de modelos em SADT 110

5.4 A introdução do conceito de tempo no SADT 112

5.5 Simulação do SADT orientada à eventos 113

6 A CLASSE SADT PARA A CONSTRUÇÃO DA FERRAMENTA GRÁFICA 117

6.1 O Paradigma do Ambiente PROSOFT 117

6.2 Abordagem denotacional para desenvolvimento de software 118

6.3 A classe SADT 119

6.3.1 SADT 120

6.3.2 Rota 122

6.3.3 Diagrama 123

6.3.4 Atividade 124

6.3.5 Conectivo 126

6.4 O Ambiente de Tratamento de Objetos SADT 127

6.5 Operações da ferramenta SADT 128

6.5.1 Operações sobre atividades 128

6.5.2 Operações sobre conectivos 128

6.5.3 Operações sobre fluxo 130

6.5.4 Operações sobre interface 131

6.5.5 Operações sobre diagramas 131

6.5.6 Operações de análise 132

6.5.7 Operações complementares 133

7 ESPECIFICAÇÃO FORMAL DA SEMÂNTICA E DA EXECUÇÃO POR SIMULAÇÃO DE SISTEMAS DESCRITOS EM SADT 135

7.1 O sistema para simulação do SADT 136

7.2 Organização da especificação 136

7.3 Definição do domínio semântico 137

7.3.1 Anotações 138

7.3.2 A instanciação de um objeto 140

7.4 Restrições sobre os domínios semânticos 142

7.4.1 Função de boa formação dos objetos 142

7.5 Restrição aos objetos da classe 142

7.5.1 Observações 144

7.6 Definição da sintaxe das operações 145

7.6.1 O domínio sintático 146

7.7 Definição das assinaturas das operações 147

7.7.1 Assinaturas 148

7.8 Definição das funções semânticas 149

7.8.1 Inicialização do sistema 149

7.8.2 Função execute 150

7.8.3 Funções auxiliares da função execute 158

7.8.4 Função semântica simular 163

7.8.5 Funções auxiliares da função simular 163

7.9 O refinamento da especificação em direção à implementação 167

7.10 Considerações sobre a especificação formal apresentada 168

8 CONCLUSÕES E EXTENSÕES **169**

BIBLIOGRAFIA **175**

LISTA DE FIGURAS

2.1 Múltiplos pontos de vista 34

2.2 Hierarquia de diagramas SADT [ROS 83] 35

2.3 O diagrama SADT [ROS 83a] 35

2.4 O ciclo autor/leitor no SADT 37

2.5 Refinamento semântico no SADT 41

2.6 Interconecção de modelos em pontos de vista x e y 42

2.7 A caixa SADT (SADT box) 43

2.8 Caixa SADT representando a expr. algébrica AX + B = Y 44

2.9 Estrutura de construção de setas 45

2.10 O conectivo JOIN 45

2.11 O conectivo BUNDLE 46

2.12 O conectivo OR JOIN 46

2.13 O conectivo BRANCH 47

2.14 O conectivo SPREAD 47

2.15 O conectivo OR BRANCH 47

2.16 Caixas representado atividades ou dados 49

2.17 Dominância entre atividades 49

2.18 Fronteiras no diagrama SADT 50

2.19 Fronteiras e interfaces 51

2.20 Diagrama detalhando caixa pai [ROS 85] 53

2.21 A notação de chamada SADT [DIC 78] 54

2.22 Comutação e compartilhamento de modelos via chamada 55

2.23 Notação para regra de ativação 59

2.24 Atividade dispara pela regra padrão 60

2.25 Atividade não dispara pela regra padrão 61

2.26 Exemplo de regra de ativação 61

2.27 Detalhamento da atividade A3 62

2.28 Diagrama com seqüencialização [DIC 81] 64

3.1 Diagrama de atividades 72

3.2 Grafo Correspondente ao Diagrama 73

3.3 Um exemplo de "dataflow machine" 75

3.4 Filas de "tokens" como tipo de dado 76

4.1 Simulação de atividades SADT 100

4.2 Exemplo de utilização dos construtos do SADT 103

5.1 Sistema em simulação 107

5.2 Relacionamento atividade x evento x processo[PRI 86] 114

5.3 Níveis de simulação no SADT 116

6.1 Descrição de uma atividade 129

6.2 Refinamento de atividade 129

6.3 Nomeação de um fluxo em um diagrama 130

6.4 Inserção de uma interface em um diagrama 131

6.5 Análise de um diagrama 133

6.6 Início de sessão no Ambiente PROSOFT 134

7.1 Sistema de reserva e atendimento de passageiros 141

7.2 Diagrama de assinaturas 148

Resumo

A definição de requisitos tem sido reconhecida como uma das mais críticas e difíceis tarefas em engenharia de software. A necessidade de ferramentas de suporte é essencial.

Nos dias de hoje, entre os vários métodos existentes para apoiar a fase de requisitos, destaca-se o SADT (Structured Analysis and Design Techniques) devido a sua capacidade de representar modelos.

Este trabalho estabelece semântica para o método SADT, baseando-se na inter-relação do método aos sistemas de fluxo de dados(redes, grafos e máquinas de fluxo). Faz-se, inicialmente, uma abordagem operacional para a semântica de seus construtos básicos e, posteriormente discute-se a possibilidade de executar especificações através de simulação.

Uma ferramenta para suportar o método SADT foi projetada e construída e é apresentada. Ela foi definida a partir de um modelo, denotado por uma classe, através de uma sintaxe abstrata. Essa ferramenta foi implementada no ambiente PROSOFT, fornecendo para o usuário mais de quarenta operações de apoio à construção/manipulação de diagramas.

O trabalho também apresenta a especificação formal em VDM - Vienna Development Method, da semântica dos principais construtos do método SADT, bem como uma proposição de execução de especificações através de simulação. São ainda indicadas direções nas quais o trabalho pode ser estendido.

PALAVRAS CHAVE

Método, Engenharia de Software, SADT, Definição de Requisitos, Especificação Formal, Engenharia de Software Assistida por Computador (CASE), Simulação, Método do Desenvolvimento de Viena - VDM, Especificação Funcional, Semânticas.

Abstract

The definition of systems requirements has been known as one of the most critical and dificult tasks as far as the software engineering is concerned. The need support is essential.

Nowadays, among the various methods devised to support the phase of requirements, a special emphasis is given to the SADT method (Structured Analysis and Design Techniques), due to its capability of representing models.

This work set semantic for the SADT method, based primarily upon the interrelation of the method to the systems of dataflow (nets, graphs and dataflow machines). It deals with an approach of operational semantics to its basic constructs, and it will, afterwards, discuss the possibility of carry out specifications by simulation.

A tool was built to support the SADT method, and it was defined by a model denoted by a class, through an abstract syntax. This tool was implemented in the PROSOFT environment, providing for the user, more than forty support operations for the construction /manipulation of diagrams.

This work also presents the formal specification of the semantics of the main constructs of the SADT method in VDM - Vienna Development Method; as well as an execution proposal of specifications through simulation.

Directions have been indicated concerning the extension of the research.

KEY WORDS

Method, Software Engineering, SADT, Requirements Definition, Formal Specification, Computer Assisted Software Engineering (CASE), Simulation, System Dataflow, VDM, Funcional Specification, Semantics.

1 INTRODUÇÃO

1.1 A definição e especificação de requisitos de software

Em qualquer área do conhecimento, para se resolver determinado problema, é preciso em primeiro lugar conhecer e entender o problema. Como as demais áreas da engenharia, a construção de software defronta-se com a mesma necessidade.

A produção de um bem em um típico ambiente das engenharias convencionais obedece a um ciclo de vida ou linha fabril, o que possibilita que se façam acompanhamentos, testes, revisões, validações etc. Isso é possível porque nesses ambientes de produção, a construção de qualquer produto é sistemática. Esse modelo muito vigente no mundo moderno, com ligeiras modificações, é amplamente utilizado na manufatura de bens de natureza física.

A engenharia de software manufatura produtos de natureza lógica, e, em consequência, é difícil estabelecer um mapeamento do modelo convencional de manufatura para ela. Em [YET 80] afirma-se que o problema central em grandes e complexos sistemas de software é *continuidade* e *mudança* e não desenvolvimento. Um produto de natureza lógica, para ser bem construído, necessita de maior empenho de seus projetistas em sua concepção, para que seja *precisamente definido*, atendendo às necessidades atuais de seus usuários, e permitindo a evolução futura em seu ambiente de vida.

A definição de requisitos deve considerar primordialmente que todos os sistemas são sistemas sociais [ROS 83]. Um sistema não existe isoladamente,

pertence a um todo maior, e nesse contexto, pessoas, máquinas e outros sistemas interagem entre si e com o sistema em questão, participando de um ambiente mais complexo.

A necessidade de se definir requisitos de software de uma maneira *clara* e *precisa* motivou o surgimento de muitos métodos para tratar o problema. [RIB 90] apresenta um amplo estudo desse universo. Segundo [GEH 86] os requisitos de software são frequentemente negligenciados, pela falta de consciência de sua importância, o que de certa forma está desaparecendo. A não utilização de um *sólido* método para definir e especificar requisitos de software traz como consequências o retardamento do projeto e a elevação de seus custos. Definir requisitos de software é fazer, com o apoio ou não de ferramentas automáticas, a *busca*, *coleta* e *entendimento* de características ou propriedades do sistema em estudo [FRE 83]. Sendo a definição dos requisitos a primeira fase a ser realizada no processo de construção de um software, essa definição deve expressar integral e corretamente todas as necesidades do usuário do produto (sistema), não sendo permitido duplicidade de interpretação do significado (semântica) do sistema.

Segundo Lehman, citado por [MEN 88], o processo de desenvolvimento de software inicia-se com uma exposição informal dos requisitos do sistema a ser desenvolvido. A partir desse enunciado em linguagem natural, deriva-se por transformações duas decomposições. A primeira é a especificação formal do sistema, e a segunda o programa que a implementa. Para essa proposição, a especificação formal é vista como uma *teoria* da qual o enunciado do problema em linguagem natural e sua implementação são *modelos*. Nessa abordagem visualizam-se dois processos: o primeiro é o de *abstração* que por transformações sucessivas chega a especificação formal do problema. O segundo processo é o de *concretização* que leva ao programa que implementa o sistema. O entendimento e o registro dos requisitos de software constituem-se em um grande problema na engenharia de software, onde não se projeta com apoio de fórmulas ou tabelas, o que Lehman defende, são na verdade processos de transformações linguísticas, sendo que a concretização apresenta a propriedade de que, duas transformações quaisquer "tn" e "tn+1" devem ser logicamente equivalentes. O conjunto de transformações, deve preservar o *comportamento* externo do sistema, mas poderão ser alterados os mecanismos que *simulam* o comportamento produzido.

A definição e a especificação de requisitos constituem o documento que *descreve* os requisitos funcionais de um sistema. Esse documento deve explorar de forma exaustiva o propósito de "O QUE" o sistema faz, não o "COMO" será feito. Dessa forma, esse documento funciona como uma espécie de *contrato* que agrega os requerimentos do usuário e também um documento *prescritivo* que fornece a base para o projetista preparar sua solução.

Para [YEH 84], uma das grandes dificuldades para a realização satisfatória da fase de definição de requisitos refere-se ao aspecto comunicação entre usuários e analistas. O usuário geralmente é leigo na área de conhecimento do analista e vice-versa, sendo ambos especialistas em suas próprias áreas. Assim uma descrição *exata* do comportamento de um sistema é uma tarefa difícil.

1.2 Métodos & semânticas

Os métodos de engenharia de software surgidos nos últimos anos, de uma forma ou de outra, são proposições para resolução da vasta gama de problemas na área. O estudo anteriormente citado de [RIB 90] faz uma avaliação desses métodos, segundo um conjunto de predicados estabelecidos. Um predicado muito importante estabelecido nesse estudo diz respeito à semântica de um método, consubstanciada em *informal*, *semi-formal* e *formal*. Os métodos de semântica informal [BLA 83] e [TEI 83] por exemplo possuem uso muito restrito. Métodos semi-formais, com intenso uso de notações diagramáticas [CAM 86],[DeM 78],[GAN 79],[HAM 74],[LUD 86],[MUL 79], [NET 88] e [ROS 83a] por exemplo, encontram grande aceitação por parte da indústria e comércio [MEN 88],[NUN 87]. E finalmente os métodos de semântica formal [BJO 78],[BJO 81],[BJO 82],[BJO 88], [DEG 90],[MAR 88],[MEY 85],[GUT 78],[JON 80] e [TER 86] ainda apresentam pouca utilização na área comercial ou industrial, sendo uma excessão em fase inicial o uso do VDM por alguns centros de processamento de dados europeus. Mas em termos acadêmicos, os métodos de semântica formal apresentam interesse crescente.

1.3 O método SADT como ferramenta de apoio a definição de requisitos

Esta dissertação apresenta um estudo do método SADT [ROS 83a], proposição de estensões, ferramenta gráfica de apoio à construção de uma árvore/rede de diagramas e a especificação formal da execução de especificações através de simulação. A escolha desse método deve-se em primeiro lugar a sua natureza extremamente cartesiana, e a subjacente formalidade nele presente.

Em um levantamento de métodos de apoio à definição e especificação de requisitos de software feito por [LEI 87], o SADT aparece como o método mais citado. Um outro ponto a destacar nessa escolha é a grande diversidade de sistemas onde ele já foi aplicado com sucesso. A literatura mundial na área o cita constantemente como um bom método para definição de requisitos.

1.4 Propósitos de uma ferramenta para engenharia de software

Uma ferramenta de engenharia de software deve permitir ao projetista trabalhar dispensando o uso de lápis, papel e borracha, além de oferecer todo um suporte computadorizado para o gerenciamento de suas atividades. A ferramenta deve oferecer ao seu usuário um conjunto de facilidades. É oportuno lembrar que ARCHIMEDES, o grande matemático grego, não inventou a alavanca por inventar, mas porque ela representa um substancial avanço para o tratamento da cinemática dos corpos rígidos. Uma ferramenta para engenharia de software possui o mesmo sentido filosófico de uma alavanca, e deverá possibilitar a aceleração da produtividade, com eficiência, potência e não simplesmente a automação de um procedimento manual. Alia-se a isso a necessidade hoje de que esse ferramental à disposição de um engenheiro de software deve possuir alto grau de robustez, segurança e ergonomia.

1.5 Pontos e contextos na ferramenta SADT

A ferramenta que apresentada nesta dissertação é um subconjunto da linguagem SA do SADT. O estilo SADT trabalha com refinamento semântico. A definição de uma função do sistema é estruturada na forma de uma árvore com vários níveis, onde o nodo raiz é uma declaração vaga e abrangente da função. Os níveis de refinamento da solução se sucedem até que seja alcançado um ponto em que o significado fique claro. Essa forma de desenvolvimento da solução de um problema por *Divisão e Conquista* é a mais natural para o ser humano, e uma boa ferramenta não pode ignorar esse fato.

O método SADT trabalha com racionalidade limitada, baseando-se nas limitações da mente humana para o processamento de informação/conhecimento. Através dessa idéia são impostas limitações relacionadas à quantidade de informação que uma pessoa pode receber, processar e lembrar-se num determinado intervalo de tempo. A ferramenta proposta implementa essa característica.

O estilo SADT, em cada nível hierárquico de decomposição funcional, com a racionalidade da informação, apresenta uma estrutura em rede, onde conectivos gerenciam todo o interfaceamento de suas atividades. Esse estilo apresenta-se satisfatório para o desenvolvimento de software, porque focaliza pontos essenciais para a construção de uma especificação mais precisa.

1.6 A dissertação no âmbito do PROSOFT

Esta dissertação é desenvolvida dentro do Projeto PROSOFT. O PROSOFT [NUN 90] é um ambiente de desenvolvimento de software orientado a modelos. O objetivo desse projeto é a construção de um ambiente de desenvolvimento e aplicação de ferramentas de software. Nesse projeto, tanto o ambiente quanto as ferramentas, chamadas no PROSOFT de ATOs - Ambientes de Tratamento de Objetos, são orientados a modelos. Cada modelo no PROSOFT é representado por uma *classe*, exibida gráficamente e definida com o apoio de uma sintaxe abstrata. Uma classe no ambiente pode ser composta por outras classes hierarquicamente estruturadas.

As classes que compõem o modelo são chamadas os domínios semânticos do modelo.

O ambiente é projetado para permitir o máximo de flexibilidade e expansão, isso significa que novas ferramentas poderão ser integradas ao ambiente, sem maiores problemas. Cada ambiente faz a administração de um conjunto de operações que criam objetos de acordo com as especificações dos usuários.

A ferramenta implementada no PROSOFT como parte desta dissertação é o Ambiente de Tratamento de Objetos SADT. Esse ato implementa um subconjunto da linguagem diagramática criada por Douglas T. Ross [ROS 83a], oferecendo ao usuário um abrangente conjunto de operações. Para tal foi definida uma classe, definida como uma sintaxe abstrata, usando tipos do VDM [BJO 78], a qual foi completamente implementada no ambiente, provando-se assim o paradigma do ambiente, e obtendo-se a integração funcional, sintática e semântica de ferramentas no PROSOFT.

1.7 A parte teórica na dissertação

Na parte teórica deste trabalho apresenta-se um estudo abrangente do método SADT, faz-se o relacionamento deste aos sistemas de fluxo de dados, redes, grafos, e "dataflow machines". Com base nisso foi definida uma semântica operacional para os principais construtos do método, o que possibilitou descrever uma forma de simulação discreta do modelo. Esse corpo de conhecimentos forneceu subsídios para a especificação formal em VDM da semântica e da simulação do SADT.

1.8 Trabalhos relacionados

O SADT é um método que foi publicado pela primeira vez, [ROS 83],[ROS 83a] pelo seu inventor Douglas T. Ross, professor do Massachusetts Institute of Technology - M.I.T., em janeiro de 1977. Desde então, a sua aceitabilidade na comunidade mundial de computação tem se verificado, tanto em meios acadêmicos como comerciais e industriais. Nesta dissertação fez-se um levantamento de trabalhos relacionados

a temática aqui exposta e, foram encontrados os seguintes trabalhos:

Em [TOL 89] é proposto o método ANA-RE como um método para análise e especificação de requisitos. O trabalho apresenta o protótipo de um ambiente para suportar a especificação automatizada de sistemas de software. ANA-RE foi desenvolvido a partir do SADT de forma a estendê-lo, tornando-o mais simples, adequando o mesmo a um contexto automatizado. O protótipo do ambiente foi implementado em SMALLTALK e suporta a definição e manipulação de diagramas "SADT-like" sob a forma textual. O ANA-RE difere em vários aspectos do projeto apresentado nesta dissertação. Toda a sua interface é textual, o que pode dificultar a sua utilização junto ao usuário, que como qualquer ser humano, processa muito mais rapidamente uma informação de natureza gráfica do que textual. O SADT possui uma linguagem essencialmente gráfica. Os aspectos sintáticos apresentados em [TOL 88] são os já conhecidos com pequenas modificações. A discussão da semântica do método nesse trabalho não é bem clara e é apresentada informalmente. A implementação do SADT em ANA-RE não permite a exibição da árvore de desenvolvimento, sabidamente isso dificulta o entendimento e visualização da solução como um todo. A ferramenta implementada como parte prática desta dissertação possibilita nem só essa forma de exibição, como outras facilidades de uma moderna interface homem-máquina como a oferecida pelo PROSOFT.

[TRA 80] descreve EDDA, uma linguagem de alto nível de especificação e programação no estilo do SADT. EDDA é na verdade uma linguagem "data flow", combinando a estrutura de entendimento e hierarquia do SADT com a precisão e fundamentação teórica das Redes de Petri. O trabalho de Trattnig e Kerner com a linguagem EDDA apresenta-se como adequado para análise de requisitos e especificação de sistemas, para problemas como processamento distribuído, arquitetura de computadores, fluxo de dados e multiprocessamento. As definições semânticas de EDDA são fornecidas através de Redes de Petri estendidas. A linguagem oferece a possibilidade de geração de código diretamente da especificação. Propriedades dinâmicas tais como tratamento de "deadlock", "overflow" de filas e estimativas de tempo de respostas podem ser derivadas diretamente do código [TRA 80]. A linguagem é apresentada pelos seus autores como adequada para especificação e projeto em modo interativo.

A linguagem EDDA tem uma parte gráfica denominada G-EDDA,

cujos construtos sintáticos são um mapeamento direto do SADT como apresentado em [ROS 83a]. Há também uma forma simbólica da linguagem S-EDDA que possui as mesmas regras semânticas válidas para G-EDDA. Como já citado, a definição semântica de EDDA foi estabelecida com Redes de Petri, mas dois importantes conectivos do SADT não foram definidos, SPREAD e BUNDLE; e as razões da não definição não são explicadas. A tradução da forma gráfica G-EDDA para simbólica S-EDDA é feita de forma manual. O Ato SADT apresentado como parte prática nesta dissertação, apresenta uma interface para construção de diagramas completamente gráfica, com uso de "mouse" por exemplo, não sendo gerado código, mas esse não é o objetivo deste trabalho. Na definição semântica que apresenta-se neste trabalho para o SADT, *todos* os conectivos e a atividade são definidos.

Um outro importante trabalho relacionado a esta dissertação é um CASE de origem francesa denominado SPECIF-X [LIS 87], desenvolvido pela IGL, empresa sediada em Paris.

As principais funções executadas pelo SPECIF-X são de três tipos: A primeira de *Produção* que permite ao usuário a criação e atualização de diagramas na formas textual e gráfica, permitindo adicionalmente a edição de glossários e textos. O segundo tipo de função oferecido pelo SPECIF-X é o de *Organização* que permite ao usuário estruturar diagramas dentro de modelos. Finalmente o terceiro tipo de função oferecido são as de *Verificação* de sintaxe ou consistência do modelo, e a geração da documentação do modelo criado. Essa ferramenta faz controle de versões de diagramas, permitindo renomear, duplicar ou eliminar diagramas. Boa parte dessas funções, o protótipo experimental implementado nesta dissertação realiza com bastante eficiência. O SPECIF-X traz a inovação de operar em ambiente multiusuário, permitindo a troca de diagramas entre autores, produção compartilhada e naturalmente tratando os problemas de concorrência e integridade de informações.

A organização das funções do SPECIF-X são baseadas sobre o esquema do SADT. A ferramenta pode ser usada para vários projetos. Cada projeto pode comandar a criação de vários modelos, os quais podem diferir por seus pontos de vista e propósito.

A verificação da sintaxe do modelo pode ser feita tanto em um dado diagrama, ou em um modelo ou sub-modelo. Dentro desse contexto o SPECIF-X

permite verificar os relacionamentos entre atividades e dados descritos em um modelo. Por exemplo, SPECIF-X pode listar todos os itens do modelo que podem ser afetados por uma atualização, eliminação ou inserção de um elemento (atividade, conectivo, fluxo, interface) em um modelo existente. O suporte de documentação automática também oferece os documentos clássicos que podem constituir parte dos documentos de requisitos do SADT tal como: Lista de índices ou o Glossário geral que combina todas as definições de itens de dados criados, para os correspondentes níveis de diagramas, na realidade, uma espécie de dicionário de dados. A interface para o usuário SPECIF-X é dividida em três níveis, que o sistema reconhece. No primeiro nível está o administrador do sistema, no segundo o líder de projeto e no último os especificadores SADT.

SPECIF-X é um sistema de produção de especificações em ambiente automatizado, mas a verificação da correção do modelo, segue exatamente o ciclo autor/leitor proposto em [ROS 83],[ROS 83a],[DIC 81]. Esse sistema, dentre suas proposições de extensões futuras, pretende estabelecer condições para execução do modelo via simulação. O artigo de [LIS 87] não diz como obter essa forma de simulação.

A proposição do protótipo experimental de uma ferramenta baseada no SADT aqui apresentada, não é tão ampla assim, mas a abrangência das operações fornecidas é em nível industrial. O protótipo implementado como parte prática desta dissertação opera em modo monousuário. Por outro lado, a definição da semântica do método aqui apresentada atinge um dos objetivos sendo buscado pelo SPECIF-X, que é dar meios para a execução de uma especificação via simulação. Assim possibilita-se a construção de cenários de teste e geração de protótipos. Isso possibilita que se observe o comportamento do sistema do lado de fora, sem maiores detalhamentos ou conhecimento de suas variáveis internas.

Em [LEA 88] encontra-se o relato da experiência inicial de desenvolvimento de um ambiente de engenharia de software na UFRGS, através do Projeto PROSOFT [NUN 87]. Esse trabalho visa a automação parcial do método SADT, como ferramenta para a fase de análise de requisitos. O trabalho apresenta uma estrutura de dados definida com apoio da notação do Método de Jackson [BLA 83] e a sua conversão para tipos da linguagem PASCAL. Adicionalmente esse trabalho apresenta um conjunto de operações disponíveis para o usuário. A interface

do sistema é gráfica sendo possível exibir a árvore de desenvolvimento do sistema. O aspecto mais interessante desse trabalho é a implementação de uma função de *Roteamento* automático, onde com a indicação do ponto de partida de um fluxo e seu ponto de chegada, o sistema estabelece uma rota, evitando cruzar atividades ou conectivos no diagrama. As operações para o trato de interfaces externas parecem pouco definidas. Outro aspecto a ser observado é que operações de remoção de um fluxo são realizadas somente localmente a um diagrama. Nesse trabalho a operação de remover um fluxo em um diagrama, não atinge os seus refinamentos. Somente a operação de análise do diagrama é que poderá verificar tal fato. Essa solução não foi adotada aqui. A operação de retirada de um fluxo de um diagrama qualquer por exemplo, implica na retirada em cascata dos fluxos derivados nos demais diagramas refinados, isso em uma visão dirigida pela sintaxe do método.

A proposição de ferramenta aqui apresentada, não implementa uma função para roteamento automático, mas, em contra partida, produz a consistência real para todas as operações envolvendo fluxo. Um outro aspecto a considerar é que a classe de dados que instancia os objetos no ambiente é distinta, oferecendo atributos adicionais que consideram o tempo em atividades, além de possuir uma natureza recursiva. O conjunto de operações oferecidas pelo Ato-SADT para o usuário é muito mais poderoso e diversificado.

O sistema SAFP2 [NEU 82], desenvolvido pela ITT Corp. para o projeto de sistemas de tempo real, é um outro trabalho relacionado, que é uma extensão do SADT, que, diferentemente do trabalho de [TRA 80] em EDDA, não é baseado sobre um formalismo como Redes de Petri. O sistema SAFP2 acrescenta ao SADT aspectos de fluxo de controle. [NEU 82] informa que esse sistema apresenta muitas restrições no que se refere a implementação.

Por fim entre os trabalhos relacionados, encontra-se em [PRI 84] uma descrição de SAMM - Systematic Activity Modelling Method, que é um sistema desenvolvido pela Boeing Company, o qual foi influenciado pelas primitivas experiências da companhia com SADT e o desenvolvimento de DECA, uma ferramenta de projeto e validação de software. A motivação para o desenvolvimento de SAMM foi a necessidade de realizar análise de sistema para o processo de fabricação de aeronaves. O objetivo era prover um esquema de representação gráfica baseado nos conceitos de Hori de células de atividades, os quais devem ser simples bastante

para uso de um não especialista e suportado por uma ferramenta automática. Dessa maneira o projeto de SAMM foi dirigido pelos conceitos de refinamento semântico, pelo qual uma função é exposta através de uma estrutura de árvore. Foram utilizados também, os conceitos de racionalidade limitada, que já comentamos o que significa nesta dissertação e, finalmente teoria dos grafos, sintetizando que se um sistema pode ser modelado via um grafo dirigido, isso possibilita se verificar a consistência do modelo. SAMM provavelmente é uma das primeiras ferramentas a implementar um subconjunto do SADT.

1.9 Organização da dissertação

No capítulo 2 apresenta-se o método SADT, baseado principalmente nos trabalhos de [ROS 83],[ROS 83a],[LEI 83],[DIC 81],[THO 78],[LEA 88] e [TOL 89]. Nesse capítulo faz-se uma caracterização bastante extensa do método, explorando principalmente as construções da linguagem gráfica do SADT. Analisa-se alguns dos aspectos gerenciais de utilização do método e, finalmente discute-se acerca do interfaceamento do SADT com outros métodos, bem como comenta-se acerca de extensões que se fazem necessárias ao método, possibilitando maior formalidade do mesmo, objetivando a incorporação futura de experimentação.

No capítulo 3 descreve-se a interrelação do SADT aos sistemas de fluxo de dados. É caracterizada a natureza do modelo de desenvolvimento do método como formando uma rede, sendo que esta na realidade constitui um grafo, o que é uma classificação já definida para algumas metodologias em [FAV 89]. Aliado a esses conceitos, incorpora-se os princípios de "dataflow machines" como "tokens", regras de disparo, habilitação de um nodo e outros pontos importantes. Todo esse contexto permite que se visualise o SADT em um domínio mais formal e portanto, possível de execução.

No capítulo 4 descreve-se uma abordagem operacional para os conectivos e atividades do SADT. Apresenta-se um estudo detalhado da natureza desses construtos. Essa apresentação dentro de uma ótica de "dataflow machines" permite caracterizar o comportamento (semântica) operacional de cada construto. Adicionalmente para cada construto básico mostra-se em uma linguagem "PSDL-like" [LUQ

88], um esquema básico onde descreve-se operacionalmente o funcionamento do construto.

No capítulo 5 descreve-se a simulação da execução de especificações SADT. Apresenta-se em linhas gerais a adequação de se aplicar princípios de simulação discreta ao modelo. São apresentados alguns conceitos importantes do mundo da simulação e um pequeno exemplo de aplicação a um hipotético sistema.

No capítulo 6 descreve-se a classe para a construção da ferramenta gráfica, baseada em um subconjunto da linguagem do SADT. Essa classe é definida como um domínio semântico de VDM. A importância desse capítulo é central para o desenvolvimento da ferramenta. A classe representa completamente o domínio de interesse do problema em estudo, permite a geração dos objetos e possibilita a realização do conjunto de operações propostas. Apresenta-se uma rápida sessão de utilização da ferramenta construída.

No capítulo 7 descreve-se a especificação formal em VDM [BJO 78] da semântica e da simulação do SADT. Define-se um domínio semântico objetivando a execução de especificações de sistemas descritos em SADT, baseado nos conceitos de componentes (atividades e conectivos) produtores e consumidores de informação. São definidas as condições para boa formação de objetos, domínios sintáticos e funções semânticas. A especificação é toda comentada e apresentada em níveis hierárquicos diferentes.

No capítulo 8 apresentam-se as conclusões acerca da dissertação desenvolvida, principais aspectos a serem considerados, objetivos buscados e alcançados, importância do trabalho desenvolvido como um todo, a principal contribuição ao mundo acadêmico, literatura que trata as fronteiras do problema e sugestões para estensões e futuros trabalhos.

2 O MÉTODO SADT

2.1 Origens do método

O método SADT - Structured Analysis Design Technique foi criado por Douglas T. Ross e publicado pela primeira vez em 1977 [ROS 83], [ROS 83a]. As origens do SADT se encontram nos trabalhos de Ross e seus colegas da SOFTech[1]sobre Teoria dos Plexos [STR 78], e no trabalho de Hori [ROS 83] intitulado "Human-directed activity cell model" publicado como relatório técnico em 1972, do qual foram filtradas idéias acerca dos aspectos organizacionais de modelagem estruturada [STR 78]. Outras importantes áreas do conhecimento humano que contribuíram para a criação SADT foram a Teoria Geral dos Sistemas, bem como o próprio corpo de conhecimentos estabelecido da engenharia de software. O SADT adicionalmente foi refinado e desenvolvido pela SOFTech como resultado de seu extensivo uso desde 1973.

O domínio de aplicabilidade do método é bastante amplo, já tendo sido usado em aplicações comerciais e administrativas, sistemas de tempo real, sistemas de inteligência artificial [GRE 84], sistemas de telecomunicações [NEU 82], sistemas de manufatura, sistemas de aquisição e processamento de imagem [BIR 85] e sistemas de defesa. O SADT constitui-se em uma técnica de modelagem de propósitos gerais, sendo aplicável a uma vasta gama de problemas. O método é muito difundido na Europa e nos Estados Unidos. No Brasil seu uso ainda se restringe aos meios acadêmicos [LEA 88] e [TOL 89]; são as experiências brasileiras envolvendo o SADT. A primeira realizada na UFRGS orientada pelo Prof. Daltro Nunes e a segunda na

[1]Empresa norte-americana de engenharia de software que possui o registro do método SADT, e comercializa muitos produtos dele originados.

UNICAMP orientada pela Profa. Cláudia Medeiros.

No artigo pioneiro apresentando o método [ROS 83a], Ross afirma que:

> *O começo real do movimento científico e tecnológico deu-se no período renascentista, com a idéia de que as leis comportamentais da natureza podem ser entendidas, analisadas e manipuladas para realizar propósitos usuais.*

Observando, entretanto, que tal idéia sozinha não foi suficiente até a criação e evolução do conceito de linguagem "Blueprint" [ROS 83], através da qual se tornou possível expressar de maneira exata como as partes de um sistema podem ser combinadas formando um todo, e, como cada tarefa específica é descrita e/ou manipulada. A linguagem de "Blueprint" permite a descrição de um esquema, plano, mapa, projeto, "frame" ou uma formulação. Essa conceituação de linguagem é uma prática mundialmente adotada na descrição (especificação) de projetos em todos os ramos da engenharia. Ross imbuído da necessidade de tratar a construção de software como um projeto de engenharia "adotou" as idéias básicas contidas na linguagem "Blueprint", o que está bem caracterizado na notação apresentada em [ROS 83a]. O "Blueprint" na engenharia constitui-se em uma completa e rica linguagem, apresentando formulações científicas, matemáticas e geométricas, notações, medidas e nomes. Não é uma mera descrição de um objeto ou processo que se encontra em uma planta de um projeto de engenharia, mas também um modelo. As várias engenharias possuem suas linguagens de "Blueprint" particulares, devido a diversidade de áreas de conhecimento, papéis e necessidades das pessoas que as usam, mas em todas está rigorosamente presente e bem estruturada a informação, de forma entendível.

A linguagem do SADT para a descrição de sistemas foi fortemente baseada no conceito "Blueprint", objetivando apresentar especificações de forma clara e precisa, padronizada, e lógica, orientada a equipes e possibilitando a validação do modelo definido.

O SADT trabalha com o princípio de divisão e conquista[2], e tem como

[2]Esse princípio foi estabelecido por DESCARTES em seu "Discours de la Méthode" encontra-se

norma o fato de que um problema antes de ser resolvido, ele deve ser propriamente definido.

Em [ROS 85] encontra-se que SADT é um método adequado para tratar a complexidade através de uma disciplina organizada de pensamento, orientada a equipes, que é acompanhada por uma documentação gráfica e textual concisa, completa e legível. Como um método para Engenharia de Software o SADT tem sido usado com bastante sucesso para a fases de definição de requisitos.

2.2 Modelo do sistema, propósito e ponto de vista

O método SADT trabalha com modelos e seus princípios para tal estão baseados na idéia de que sistemas podem ser definidos pelo relacionamento de seus objetos (dados) e os acontecimentos que os envolve (atividades) e que isto é similar, por exemplo, à linguagem natural, que relaciona nomes e verbos para formar expressões usuais [ROS 83a]. Cada modelo em SADT consiste em uma hierarquia de diagramas, descrevendo o sistema de um particular ponto de vista. Assim um mesmo sistema, pode ser representado por um número de modelos, correspondentes a diferentes visões.

O modelo de um sistema descrito em SADT constitui uma árvore de diagramas, e essa hierarquia descreve o sistema de um ponto de vista identificado para um particular propósito. Um modelo SADT ainda pode fornecer outros subsídios relevantes para o seu entendimento como texto explanatório, índices de nodos, notas de rodapé e etc... Os diferentes pontos de vista de um sistema, presente nos múltiplos modelos, são normalmente requeridos para um adequado entendimento da solução modelada. A figura 2.1 [DIC 81] ilustra o conceito de pontos de vista usados no método.

A estruturação do modelo de sistema como uma hierarquia de diagramas caracteriza perfeitamente o princípio de divisão e conquista na resolução de problemas, ou um refinamento semântico. A figura 2.2 [STR 78] mostra uma

"...de dividir cada uma das dificuldades que eu examinasse em tantas parcelas quanto fosse possível, e quanto fosse requerido para melhor resolvê-las"

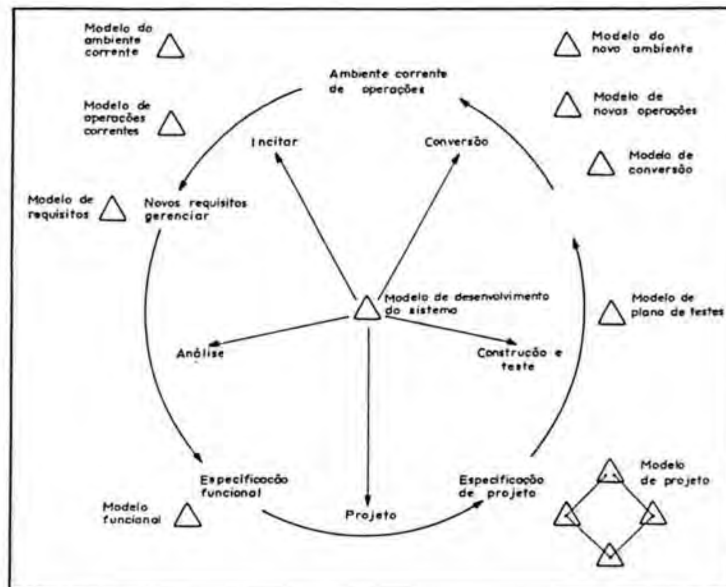


Fig. 2.1 Múltiplos pontos de vista

hierarquia de diagramas.

No SADT, diagramas são utilizados para a representação de uma especificação de requisitos de um sistema, sendo uma simplificação do método [ROS 85]. Cada diagrama é composto por um conjunto de atividades, conectivos e fluxos, cada um dos quais identificados por uma descrição apropriada. As atividades são representadas graficamente por retângulos (caixas), os fluxos são representados por arcos conectando as caixas. Os arcos podem apresentar pontos de convergência ou divergência de fluxos, o que caracteriza de forma global o conceito de conectivos no SADT. A figura 2.3, adaptada de [ROS 83a] ilustra isso.

2.3 Validação do modelo do sistema

Um fato indiscutível em termos de serviços de processamento da informação é que os usuários procuram, sempre, a solucão mais correta possível de seus problemas. Precisão, completeza e confiabilidade, são requisitos não-funcionais, indispensáveis para o sucesso de um software. [ROM 85], em um interessante artigo, discute que mesmo considerando a importância dessa classe de requisitos ditos não-funcionais, até hoje ainda não foi possível formalizá-los, de maneira completa e satisfatória, o

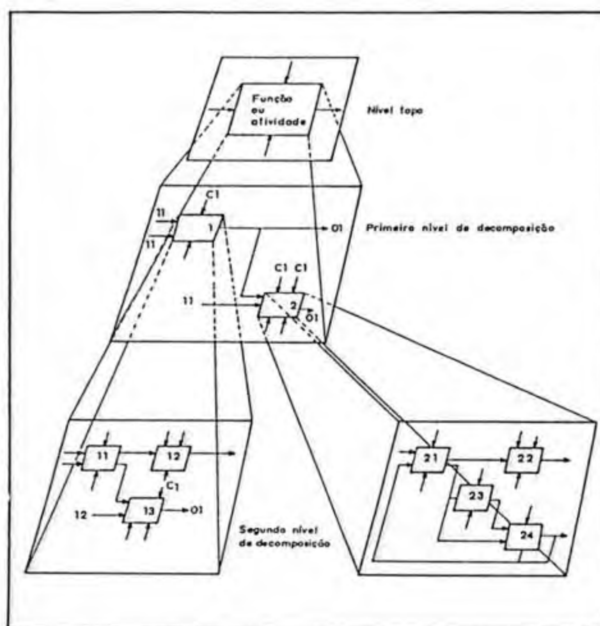


Fig. 2.2 Hierarquia de diagramas SADT [ROS 83]

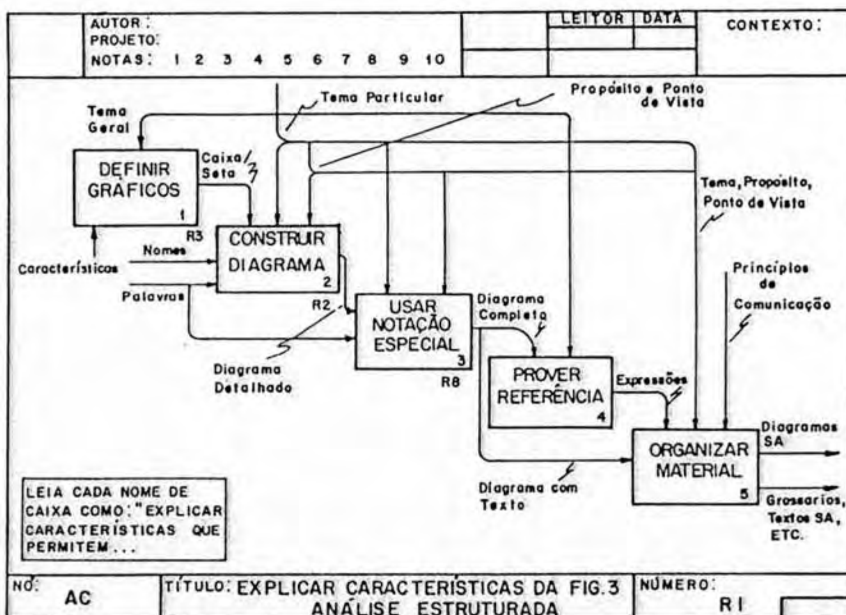


Fig. 2.3 O diagrama SADT [ROS 83a]

que sem dúvida implica na qualidade. Considerando esse fato, SADT é um método que enfatiza em seu contexto os aspectos da organização, o ambiente onde "viverá" o sistema sendo desenvolvido, procurando assim maximizar a "captura" de requisitos importantes para o desenvolvimento de um sistema com certo grau de completeza.

A definição de requisitos de análise [LEI 87] é um processo no qual "o que é para ser feito" é elicitado e modelado. O SADT é considerado um método bom para essa fase no ciclo de vida de desenvolvimento de software. Definir requisitos é uma tarefa complexa e, segundo [BLA 83], fundamentalmente deve considerar os seguintes aspectos.

- ANÁLISE DE CONTEXTO - O que faz o ambiente do problema, observando-se como e quais são as relações entre o problema e o ambiente.
- ESPECIFICAÇÃO FUNCIONAL - Que funções devem ser implementadas, e quais as restrições para levar em consideração na fase de projeto.
- RESTRIÇÕES DE PROJETO - Que restrições devem ser consideradas, sobre como o sistema requerido é para ser construído e implementado.

A definição de requisitos de um sistema é o ponto de partida para a sua construção, sendo uma atividade, de natureza extremamente complexa e de características próprias. Sistemas podem ser de uma mesma categoria, porém jamais serão iguais, cada sistema é um sistema em particular. Na Engenharia de Software o principal problema que se enfrenta é a complexidade. Um sistema não representa a solução de um único problema isolado, mas vários problemas relacionados, atendendo a múltiplos usuários com requisitos de especificação em constante mudança. Além disso, o desenvolvimento de software necessita de um efetivo gerenciamento, e nesse contexto, a definição de requisitos precisa da participação integrada de uma equipe de desenvolvimento. Isso demanda uma definição clara da interação que deverá existir entre essas pessoas. O método SADT antecipa esta necessidade, estabelecendo títulos e funções apropriadas ao time de desenvolvimento, o que [ROS 83] denomina de ciclo autor/leitor. O objetivo desse ciclo é promover a revisão do modelo, o que é feito de maneira informal, validando assim a especificação contida no diagrama. É importante que se diga que esse método de revisão tem se mostrado eficiente. O SADT busca descrever a solução de um problema com uma linguagem

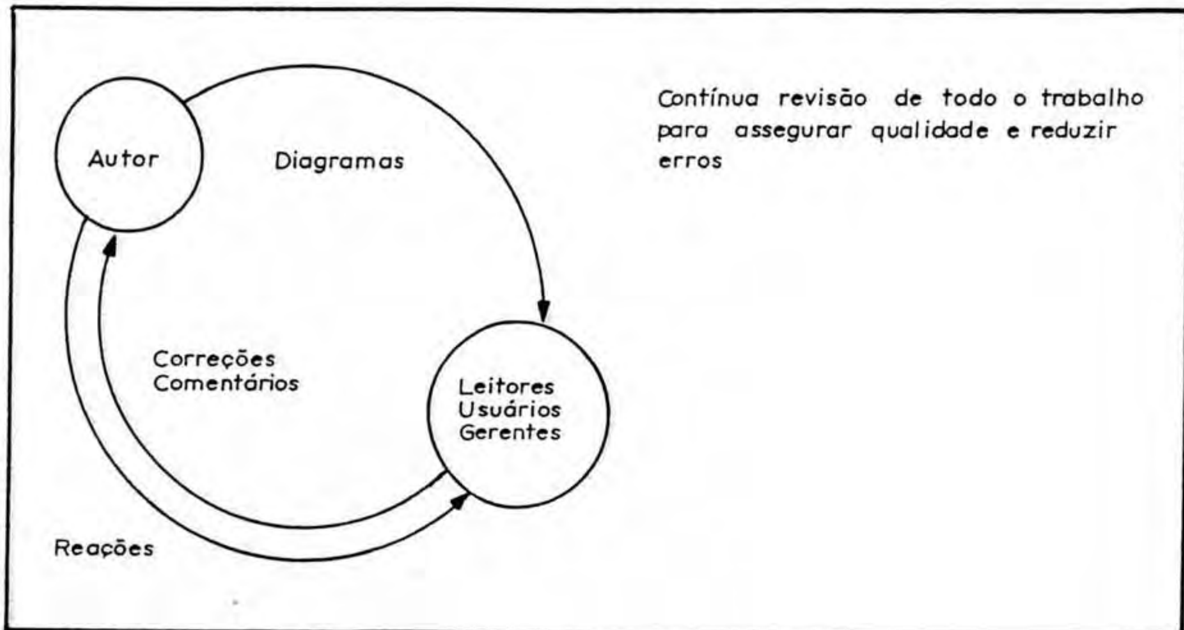


Fig. 2.4 O ciclo autor/leitor no SADT

orientada a aplicação. Sendo assim, os revisores do problema são usuários e projetista, e tal revisão consegue atingir também os requisitos não-funcionais que são muito importantes [ROM 85].

A figura 2.4 [ROS 85] ilustra o ciclo autor/leitor para a validação da especificação.

O conjunto completo de títulos e funções para o ciclo autor/leitor do SADT, como apresentado em [ROS 83] para uma correta aplicação do método, é o seguinte:

TÍTULOS E FUNÇÕES

Autor Pessoa que estuda requisitos e restrições,analisa funções do sistema e as representa por modelos baseados em diagramas SADT.

Comentarista Usualmente atores que devem rever e comentar de forma escrita, os modelos criados por outros atores.

Leitor Pessoa que lê diagramas SADT para dar apoio técnico, mas não precisa fazer comentários escritos.

Especialista Pessoas de quem os autores obtém informações especializadas acerca de requisitos e restrições, através de entrevistas.

Comitê Técnico Um grupo de pessoas com mais experiência, para revisar a análise, a cada nível da decomposição. Qualquer um deles dá a solução e distribuição técnica ou recomenda uma decisão para o gerente do projeto.

Bibliotecário Pessoa com a responsabilidade de manter um arquivo centralizado de toda a documentação do projeto, fazer cópias,distribuição de material para leitores, manter registros, etc.

Gerente do Projeto Um membro do projeto que tem a responsabilidade técnica final para cumprir a análise e projeto do sistema.

Monitor Pessoa conhecedora do SADT, que ajuda e aconselha o pessoal do projeto, no uso e aplicação do método.

Instrutor Pessoa conhecedora do SADT, que instrui autores e comentaristas, que usam o SADT pela primeira vez.

No esforço para a definição correta de requisitos do sistema, o papel do autor é normalmente desempenhado por analistas treinados, e com bastante experiência no uso do SADT. O ciclo autor/leitor facilita a revisão do modelo construído. A aplicação desse ciclo permite que as várias partes envolvidas no desenvolvimento de um sistema, como analistas, projetistas e usuários, possam interagir de forma integrada, por uma contínua e efetiva comunicação, permitindo uma correta definição de requisitos, por uma compreensível documentação. Os diagramas são parte importante dessa documentação, e neles é feita a crítica e revisão da solução proposta ao problema em estudo. Esse processo tira partido da estrutura de um modelo SADT, desde assim que decisões podem ser vistas no contexto do modelo e, contestadas enquanto alternativas forem ainda viáveis.

Em toda parte envolvida no projeto, esboços de versões de diagramas envolvendo modelos são distribuídas para membros do projeto para revisão.

Comentaristas fazem suas sugestões e escrevem diretamente sobre cópias dos diagramas. As mudanças e correções são feitas e todas as versões entram numa lista, num esquema de arquivos. O bibliotecário providencia cópias de diagramas, faz a distribuição mantendo um suporte de registro, fazendo com isso, indiretamente um controle de versões diagramáticas e textuais do sistema em desenvolvimento. Esse processo documenta todas as decisões e razões porque essas decisões foram tomadas. Essa feição extremamente organizacional é que tem produzido o sucesso da aplicação do SADT a grandes e complexos projetos na engenharia de software[3].

2.4 A linguagem gráfica do método: (SA)

A linguagem gráfica do SADT fornece elementos para que um analista possa de maneira disciplinada expressar os requisitos do problema em estudo, usando uma linguagem natural evolucionária, apropriada ao domínio da aplicação. A função básica da linguagem SA é relacionar estruturas e comunicar unidades de pensamento [ROS 83a].

O método SADT é composto de duas grandes partes [ROS 85]: Uma linguagem de diagramação denominada "Structured Analysis" (SA) e uma técnica de projeto "Design Technique" (DT). Nessa seção apresenta-se um estudo básico somente da linguagem gráfica de diagramação (SA) posteriormente será exposta, especificamente a técnica de projeto (DT).

A análise estruturada no SADT é feita com base na seguinte trilogia de princípios [ROS 85]:

I DECOMPOSIÇÃO "TOP DOWN" por refinamentos sucessivos, resultando em uma análise estruturada e hierárquica em vários níveis de detalhamento;

II LIMITAÇÃO DA INFORMAÇÃO, o que significa estabelecer um limite aceitável de decomposição funcional de qualquer assunto em um máximo de seis componentes e um mínimo de três. O limite de seis (6) assegura que, no contexto

[3] A literatura corrente cita [CRI 78] que o SADT foi utilizado para a especificação de um sistema de defesa, que era descrito em linguagem natural, em um volume de 2000 páginas, o qual ficou reduzido a 40 diagramas.

geral, o significado de cada componente não seja muito difícil de entender. Em [STR 78] é citado que experimentos psicológicos mostram que é difícil ao ser humano incorporar mais do que 5 a 7 conceitos distintos de uma só vez, e que o limite inferior de 3 é utilizado para assegurar a existência de detalhes suficientes, de maneira a fazer que a decomposição seja interessante;

III MODELAGEM segundo uma orientação, que significa que toda solução de um problema é um modelo baseado em um contexto, um propósito e um ponto de vista. O contexto estabelece o assunto do modelo como parte de um todo maior, cria um limite com o ambiente através da descrição de interfaces externas. O propósito estabelece o objetivo do modelo. O ponto de vista determina o que pode ser visto no contexto, e qual a perspectiva para isso.

A linguagem diagramática do SADT [ROS 83a] fornece um conjunto finito de construções, com as quais o engenheiro de software pode compor estruturas ordenadas de qualquer tamanho, que ele necessitar. Essa notação é completa e exata, sendo composta de caixas e arcos. As caixas [4] representam partes do todo, em uma forma precisa, e os arcos representam as interfaces entre as partes. Os diagramas SADT compostos de caixas, arcos e anotações em linguagem natural são aplicáveis tanto para a modelagem de atividades quanto para dados.

Um modelo SADT [ROS 83a] é uma seqüência organizada de diagramas, cada um com um texto associado. O diagrama de mais alto nível representa o todo. Cada diagrama de nível mais baixo mostra uma quantidade limitada de detalhes sobre um tópico. Além disso, cada diagrama de nível mais baixo relaciona-se exatamente com a parte de nível mais alto do modelo, isso preserva o relacionamento lógico de cada componente. Veja a figura 2.5 à seguir.

Um modelo SADT é uma representação de uma estrutura hierárquica de um sistema, decomposto com o firme propósito de entendimento. Um modelo é estruturado e gradualmente exibe mais e mais detalhes. Mas essa profundidade é limitada pelas restrições e a satisfação limitada pelo ponto de vista. As prioridades descritas por sua intenção determinam os níveis da decomposição "top down". Múltiplos modelos coinciliam vários pontos de vista e vários estágios da realização do sistema. A figura 2.6 a seguir [DIC 78] mostra dois modelos de descrição de um

[4] "boxes" na notação original.

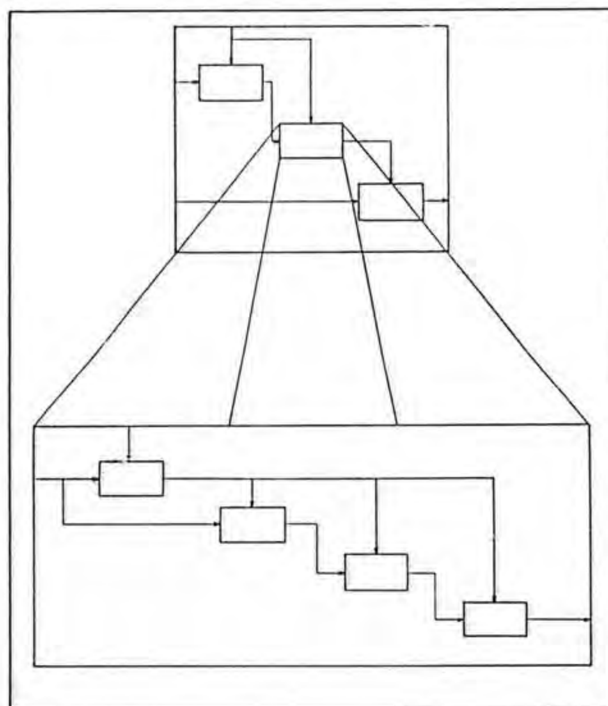


Fig. 2.5 Refinamento semântico no SADT

mesmo sistema (i. e., a descrição do sistema com diferentes pontos de vista, no caso visão x e visão y) compartilhando os detalhes comuns. Consistência pode ser examinada"amarrando" (*tying*) os diferentes modelos para um sistema único, tornando explícito seus interrelacionamentos. Vale a pena lembrar neste ponto, que o conjunto de diagramas contidos em um modelo SADT, "sempre" descreve o sistema de um identificado ponto de vista para um particular propósito.

A linguagem SA é a grande ferramenta do método para a análise de problemas via decomposição funcional, habilitando assim o processo inverso, a síntese estruturada para alcançar um dado fim. Para o contexto do SADT, os elementos reais da construção de um bloco de análise ou síntese podem ser de qualquer classe. Pode-se incorporar quadros, palavras e expressões de qualquer categoria no modelo estrutural de decomposição oferecido pelo método, desde que, evidentemente dentro do contexto.

A universalidade e precisão da linguagem SA torna-a particularmente efetiva para a definição de requisitos, podendo ser usada para qualquer problema,

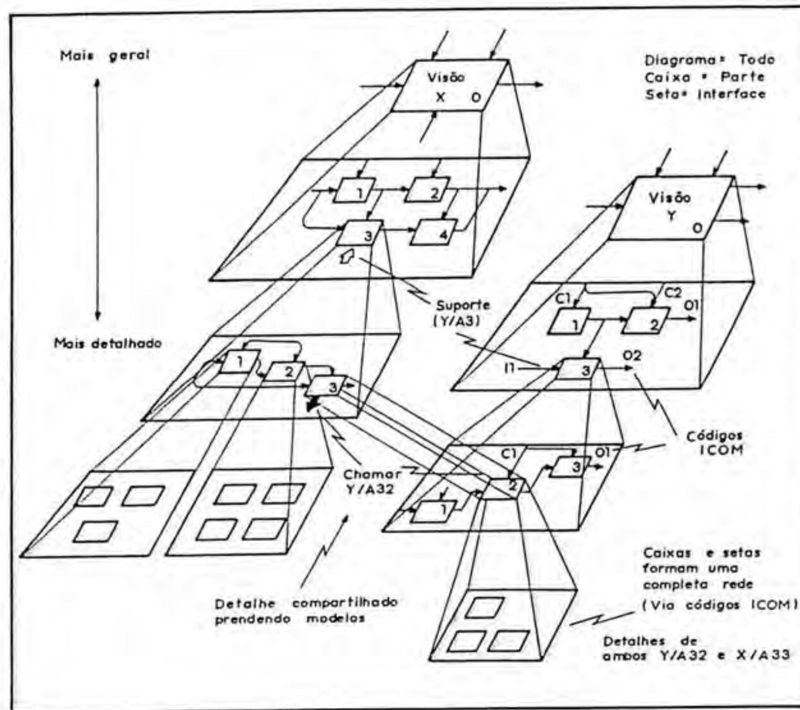


Fig. 2.6 Interconecção de modelos em pontos de vista x e y

de qualquer natureza, onde seja possível estabelecer um enfoque sistêmico.

2.4.1 A primitiva básica da linguagem gráfica

A primitiva básica do SADT é a caixa (SADT- box). A caixa é a construção da linguagem que serve para a representação de uma atividade ou de um dado. A literatura corrente também fornece a denominação de "actgrama" e "datagrama" respectivamente para conjuntos de atividades e dados.

Existem quatro tipos de setas que podem ser ligadas a uma caixa, as quais segundo a localização de conexão representam informações distintas. As setas entrando no lado esquerdo da caixa representam Entradas, setas saindo do lado direito representam Saídas. As setas entrando por cima da caixa são Controles e as setas entrando na base da caixa são Mecanismos.

As setas de entrada em uma caixa representam para o caso de uma

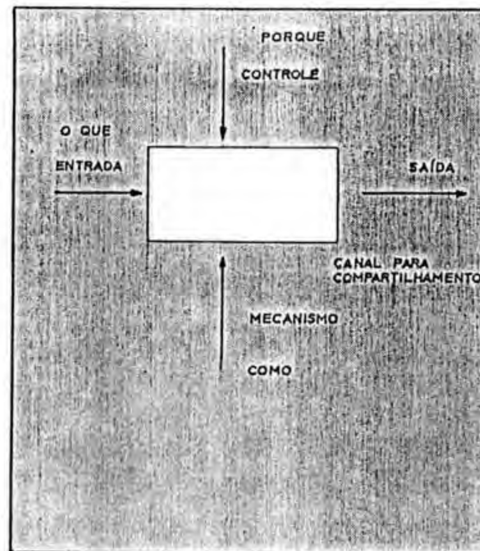


Fig. 2.7 A caixa SADT (SADT box)

atividade o(s) dado(s) disponível(veis) para leitura ou consumo [THO 78], enquanto que a seta(s) de saída representa o dado que é produzido pela atividade. A seta de Controle "governa" a maneira pela qual a transformação ocorre. Em geral a atividade tem por objetivo a transformação de um dado de entrada em um dado de saída. Setas de entrada e de controle são referenciadas coletivamente como entradas, mesmo porque uma atividade pode não ter a seta específica de entrada, mas obrigatóriamente precisa ter a seta de controle.

A seta de macanismo na base da caixa é muito poderosa, sendo uma importante parte da sintax do SADT. Mas ela pode ser omitida em uma diagramação sem prejuízos maiores. Mesmo considerando que seja essencial que um diagrama deve ser sintática e semânticamente correto pode-se fazer tal omissão até que as outras três setas estejam totalmente entendidas. O mecanismo representa somente o "agente" pessoa, organização, procedimento, programa, hardware, software, ou qualquer outro "agente" capaz de levar a cabo a transformação representada pela caixa.

O exemplo a seguir tirado de [ROS 85], ilustra bem os pontos discutidos sobre a primitiva básica do SADT. Considerando que se precisa representar diagramáticamente, a expressão algébrica AX + B = Y através de uma atividade.

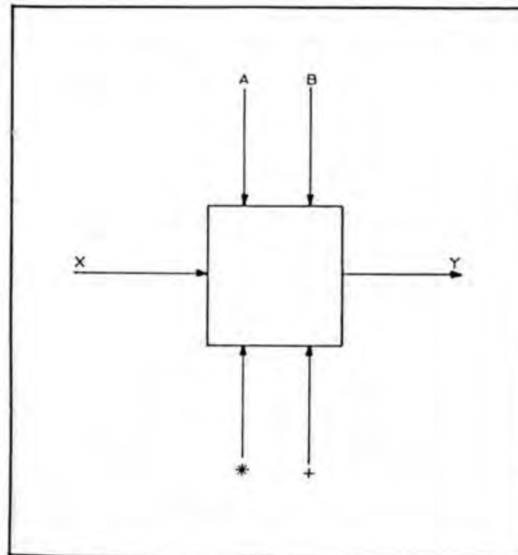


Fig. 2.8 Caixa SADT representando a expr. algébrica AX + B = Y

O argumento X é a reta de entrada na caixa, A e B são as setas de controle que governam a transformação, Y como resultado da transformação é a seta de saída. As operações aritméticas ("*" e "+" sendo específicos operadores), representam os mecanismo que levam a cabo a transformação. A fórmula como um todo é a própria caixa. Observa-se que somente as setas de entrada, controle e saída é que participam da transformação no senso do que é consumido e do que é produzido. As saídas de uma caixa no diagrama SADT proporcionam entradas e ou controles para outras caixas. Saídas de alguma caixa são saídas do sistema para o ambiente externo ao diagrama e, este ambiente pode ser tanto um diagrama pai como o próprio ambiente externo ao sistema propriamente dito. Similarmente entradas e controles são gerados de saídas de outras caixas ou do ambiente externo.

2.4.2 A conectividade no diagrama

[ROS 83a] lista algumas facilidades para compor e decompor arcos; conseguindo a replicação, decomposião e composição de dados. A figura 2.9 ilustra isso.

As construções gráficas de setas apresentadas na figura 2.9, mostram uma das mais importantes contribuições do método para notações diagramáticas

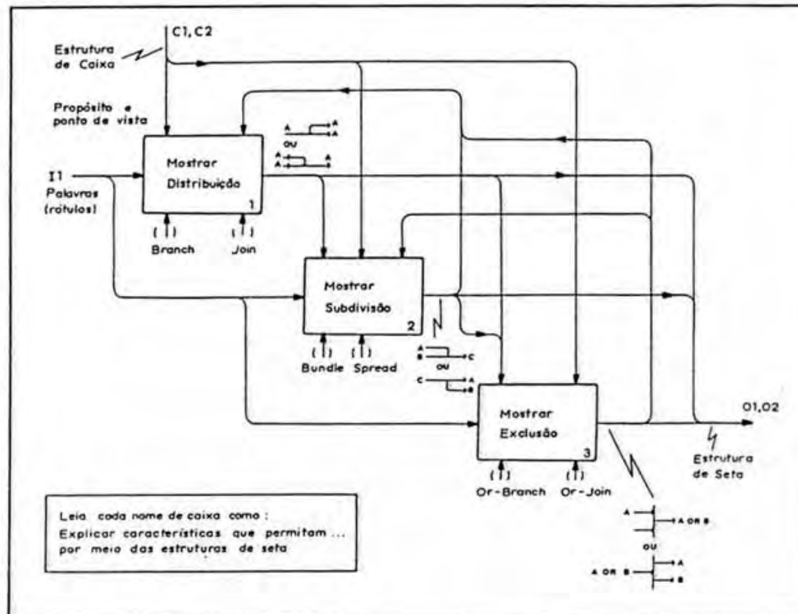


Fig. 2.9 Estrutura de construção de setas

na definição de requisitos. Essas construções são os conectivos, em número de seis, sendo três para composição de dados ("JOIN", "BUNDLE" e "OR JOIN") e três para decomposição de dados ("BRANCH", "SPREAD" e "OR BRANCH"). As figuras 2.10 a 2.15 à seguir, revelam os papéis definidos para os conectivos do SADT.

Observe que na figura 2.9 exemplifica-se o uso de uma linguagem composta dos "esquemas gráficos" dos conectivos produzidos por cada atividade para rotular as setas no diagrama. Esses pequenos símbolos gráficos manifestam os significados de "BRANCH" e "JOIN" para a atividade de Mostrar Distribuição, "BUN-

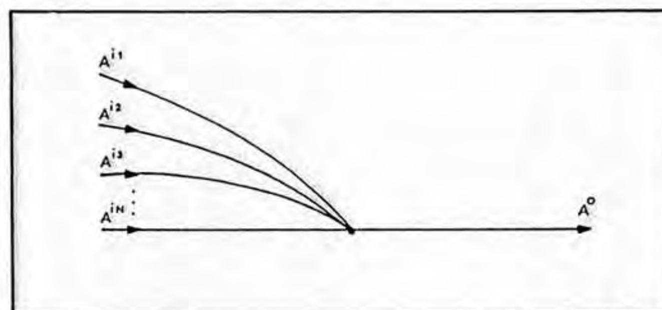


Fig. 2.10 O conectivo JOIN

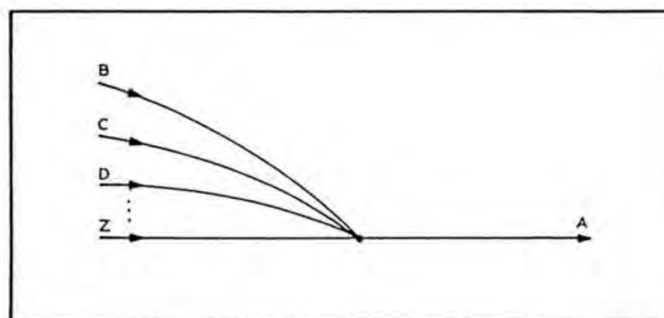


Fig. 2.11 O conectivo BUNDLE

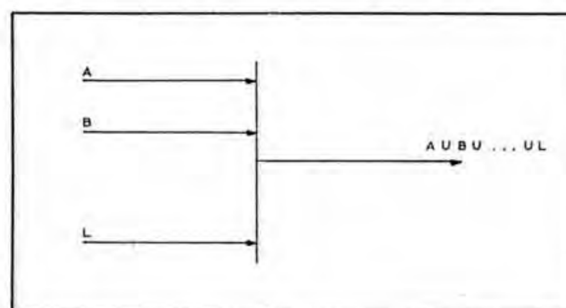


Fig. 2.12 O conectivo OR JOIN

DLE" e "SPREAD" para a atividade de Mostrar Sub-divisão, bem como as duas formas de exclusão lógica para OR em "OR-BRANCH" e "OR- JOIN" produzidos pela atividade Mostrar Exclusão. O uso de conectivos já foi feito em diagrama nesta dissertação.

Os pequenos quadros como rótulos na figura 2.9 mostram como a seta exprime o apropriado significado dos conectivos do método. Os conectivos de exclusão lógica "OR-BRANCH" e "OR-JOIN" raramente são utilizados, a não ser quando contribuírem para um melhor entendimento do problema modelado [ROS 83a]. Em muitas circunstâncias, as setas representam restrições ou dominância, uma ou outra para suprir a necessidade de entendimento dos requisitos em forma clara, meramente por conexão topológica. Segundo o próprio Ross [ROS 83a], essa é também a razão porque o método não fornece notação gráfica para outras funções lógicas como AND por exemplo, pois essas funções estão fora do escopo para o nível de comunicação da linguagem básica SA. Os conectivos existentes num diagrama permitem que o contexto total de interpretação de um modelo SA seja esboçado, muito precisamente sobre algum domínio matemático ou lógico; junto a tal domínio,

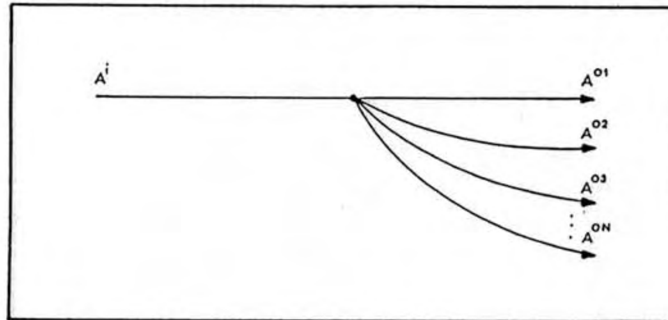


Fig. 2.13 O conectivo BRANCH

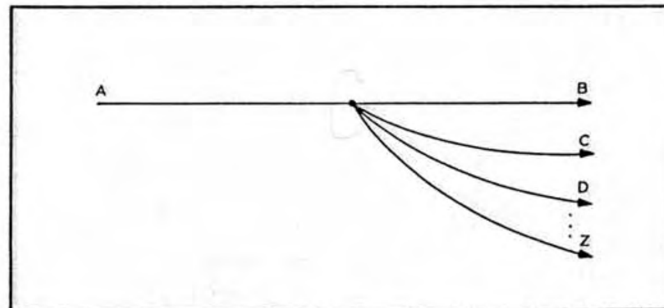


Fig. 2.14 O conectivo SPREAD

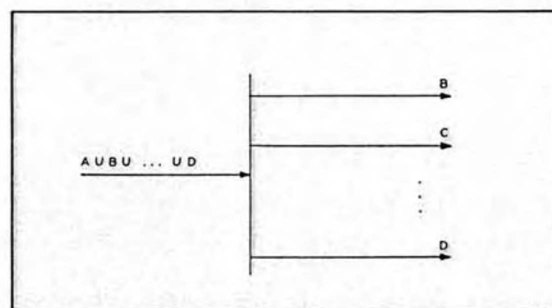


Fig. 2.15 O conectivo OR BRANCH

uma linguagem mais apropriada para expressão de requisitos deve ser escolhida. Em termos lógicos, uma expressão nominal e verbal nos rótulos ("labels") pode exprimir as condições. Isso é preferível a distorcer a linguagem SA dentro de um detalhado papel de comunicação para o qual ela não foi projetada ou intencionada preencher.

2.4.3 Dualidade de dados e atividades

Um diagrama SADT, como já foi visto, é composto somente de caixas, conectivos, arcos e uma anotação em linguagem natural.

Os diagramas para representar atividades são denominados "actigramas" e para representar objetos de "datagramas". Nos "actigramas" as caixas representam atividades e os arcos, dados; já em "datagramas", caixas representam dados e arcos as atividades. Isso permite que se modele um sistema a partir das atividades que caracterizam o sistema ou a partir dos dados que o constituem. Os modelos de dados e atividades são duais.

Um modelo SADT completo é composto de "actgramas" e "datagramas" de um mesmo assunto modelado, sendo portanto verificada a total dualidade entre modelos. Em [ROS 85] ele diz que "M é um modelo de A se M pode ser usado para responder questões sobre A".

Em geral, modelos têm nomes arbitrários, e dentro de um modelo cada combinação caixa/diagrama tem um número, de nodo que é "A" ou "D" (dependendo se o tipo de modelo é de atividade ou de dados) seguido pelo número da caixa de seu ancestral. A caixa pai de todas as demais caixa é numerada por A-0 ou D-0. O primeiro refinamento é denominado como A0 ou D0, o segundo refinamento A-1 ou D-1 de acordo com o caso e assim por diante. A caixa A23 por exemplo, é a terceira caixa do detalhamento da segunda caixa do diagrama A0.

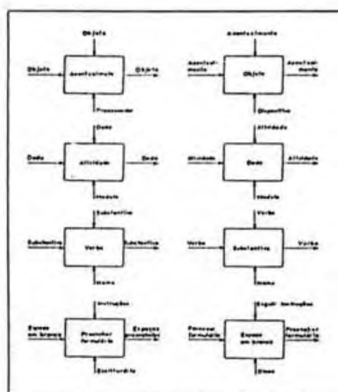


Fig. 2.16 Caixas representado atividades ou dados

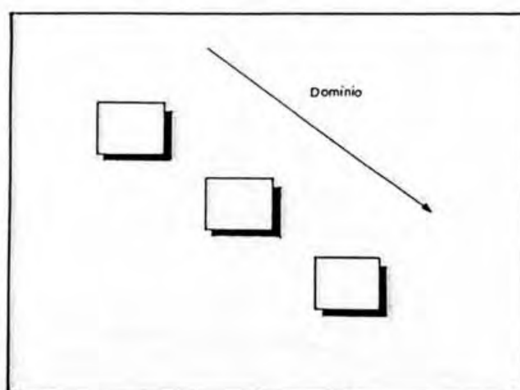


Fig. 2.17 Dominância entre atividades

2.4.4 Dominância entre caixas

Em um diagrama SADT, a posição de uma caixa indica o domínio desta em relação a outra imediatamente abaixo. O propósito, dessa construção sintática, na linguagem, é auxiliar a correta interpretação de atividades ou dados no diagrama. Isso permite que o projetista possa anotar dependência funcional ou paralelismo entre atividades ou entre dados. O mecanismo utilizado no SADT para mostrar a dominância é o "layout" de escada. A figura 2.17 [ROS 83a] ilustra dominância.

UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA

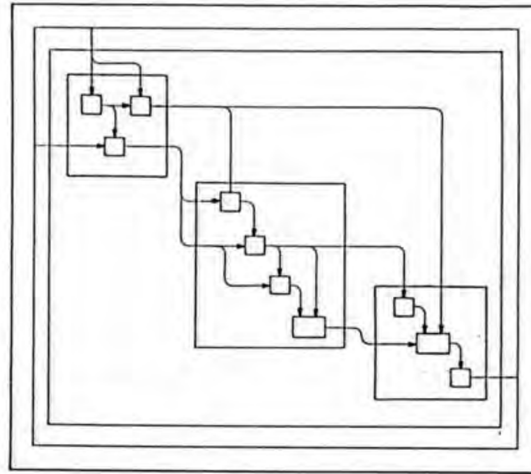


Fig. 2.18 Fronteiras no diagrama SADT

2.4.5 Fronteiras em diagramas SADT

No diagrama SADT toda seta que relaciona ou conecta contextos de fronteiras entre diagramas aninhados deve participar em ambos: de uma interface e para uma interface. Toda seta externa a um diagrama vai aparecer ausente de sua origem ou de sua destinação, devido ao fato de as caixas relevantes não aparecerem sobre seu diagrama. A figura 2.18 ajuda a explicar melhor isso. Há um diagrama que apresenta uma visão parcial de três níveis de aninhamento de caixas SA, uma dentro da outra em algum modelo, algo como trazer todos os diagramas de um modelo para um mesmo nível. Observa-se na figura que excetuando três arcos, toda seta desenhada é uma coneção completa de/para. A caixa do meio no segundo nível, tem quatro níveis finos de caixa dentro dela,e por seu turno está contida dentro da caixa maior desenhada na figura 2.18. Considerando as setas do meio no segundo nível de caixa, nota-se que somente duas delas são setas internas, todas as demais são externas. Mas observa-se também que todas as setas externas àquele nível são na realidade internas com relação ao modelo como um todo. Cada uma dessas setas vai de uma caixa para outra caixa em um nível mais descendente de caixa, em cada caso. Em conexão completa, as setas penetram as fronteiras das caixas do nível intermediário ainda que não existam todas aquelas fronteiras.

As duas únicas fronteiras que existem na realidade na figura 2.18 são aquelas características de toda decomposição SA. A primeira é a fronteira exterior

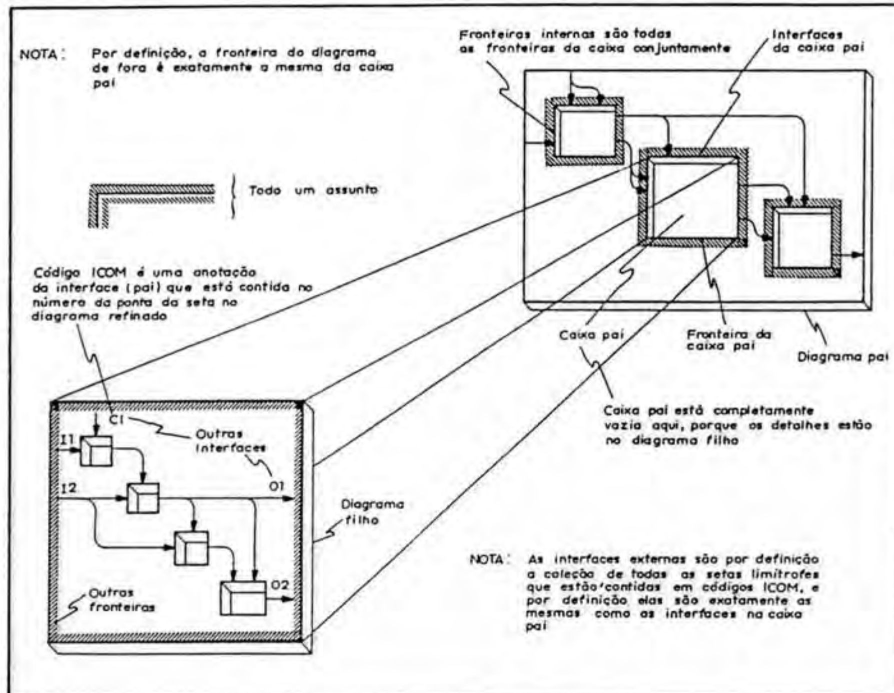


Fig. 2.19 Fronteiras e interfaces

a qual está localizada no ponto mais afastado da borda na fig. 2.18. A segunda é a fronteira interna, a qual é representada pelo conjunto completo de bordas de todas as caixas de nível mais descendente, que é considerado uma fronteira simples. Como foi estabelecido nos parágrafos precedentes, a máxima SA requer que as fronteiras interna e externa devem ser entendidas como exatamente a mesma, de tal modo que o assunto é meramente decomposto, e, não alterado em todo o caso. O entendimento de como expressar esses aninhamentos de diagramas em um modelo SADT é objeto de estudo da próxima sub-secção.

2.4.6 A interconexão de uma coleção de diagramas

O entendimento de como a estruturação da fig. 2.18 é expressado na linguagem diagramática SA, sendo feito através de um relacionamento entre fronteiras e interfaces, caixas e diagramas, e diagrama pai com respectivos refinamentos semânticos. A figura 2.19 ilustra como se expressa em SA esses relacionamentos.

A figura 2.19 mostra que no conjunto de caixas, o diagrama intermediário é desenhado como se ele fosse perfurado na massa da caixa de onde ele é retirado como um pedaço à parte. Embora as dimensões do pedaço retirado estejam distorcidas, e os pontos de interfaces não apresentem proporcionalidade desejável em relação à parte de onde extraiu-se essa parte, por definição o diagrama fronteira externo é realmente o mesmo, igualmente a caixa pai fronteira (isto é, o diagrama filho corrente é uma parte removida da camada que contém o diagrama pai). Essa figura também mostra que na decomposição hierárquica como um todo, a fronteira de um diagrama pai é a coleção de todas as suas caixas filhos, considerando fronteiras como uma simples entidade. Pela fig. 2.19 observa-se que as setas externas do diagrama filho penetram através das fronteiras externas deste e são, na realidade, as mesmas setas das interfaces da caixa pai. Portanto, as conexões que têm de ser feitas são claras. Mas para flexibilidade da representação gráfica, encontra-se em [ROS 83a] que as setas externas do corrente diagrama não necessitam ter o mesmo "layout" geométrico, relacionamento, ou rotulação como as correspondentes pontas sobre o diagrama pai, as quais são desenhadas sobre um diagrama completamente diferente. A fim de permitir essa flexibilidade, é proposto no método, a construção de um esquema especial de codificação denominado de código ICOM. O código ICOM começa com uma das letras I-C-O-M (estabelecidas para INPUT, CONTROL, OUTPUT e MECHANISM) concatenado com um número inteiro seqüencial, indicando a ordem de especificação das setas na caixa pai, de cima para baixo e da esquerda para a direita. O código ICOM na caixa pai localizada no topo da árvore de diagramas deve ser escrito sobre o fim não conectado das setas externas do diagrama, assim que o casamento do diagrama fronteira para a fronteira da caixa pai é bem definido. A figura 2.20 [ROS 85] mostra uma caixa e seu diagrama detalhado, incluindo metanotas (indicada por números circulados) explicando o diagrama mas não incluído o seu significado.

2.4.7 O conceito de mecanismos

Um importante aspecto do vocabulário de projeto é o conceito de mecanismo[5] no SADT. Um mecanismo é um modelo distinto SADT, que foi isolado por razões de

[5]Quando o tipo de mecanismo explicado aqui for utilizado, ele deverá aparecer obrigatoriamente na diagramação.

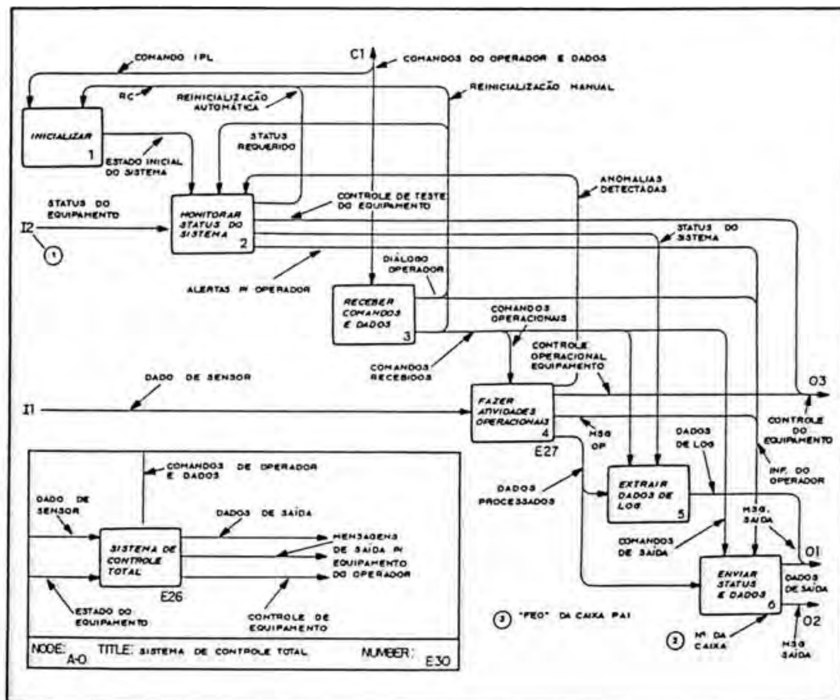


Fig. 2.20 Diagrama detalhando caixa pai [ROS 85]

compartilhamento e mudança. Esse mecanismo é naturalmente adequado para abordagem de ocultamento da informação, tal como tipos abstratos de dados, monitores e níveis de abstração. Grandes decisões de projeto podem ser encapsuladas dentro de um mecanismo.

O mecanismo no SADT pode ser chamado de uma maneira similar ao conceito de chamada de uma subrotina em linguagem de programação. A notação mostrada na figura 2.21 sobre um diagrama no modelo significa que a decomposição do ponto de vista do modelo X é continuada no modelo Y do ponto de vista do modelo Y para a caixa identificada por C. O modelo Y é dito "suportar" o modelo X e isto pode ser mostrado graficamente como na figura citada.

A chamada de referência de expressão pode, na realidade, ser condicional, dando várias diferentes chamadas de caixa, cada uma com uma específica condição sinalizada da particular caixa para ser chamada, visando suprir o detalhamento. Portanto, dependendo de condições parametrizadas, uma mesma caixa é intercambiável. Similarmente, um mecanismo pode suportar diferentes modelos, e

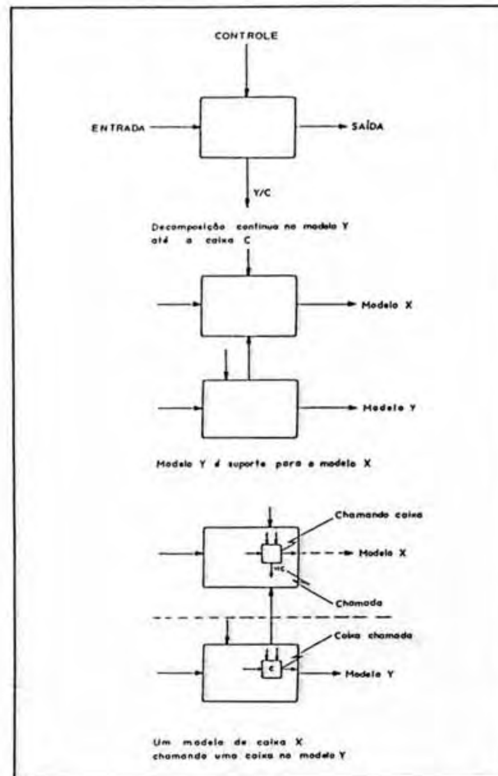


Fig. 2.21 A notação de chamada SADT [DIC 78]

de qualquer um desses modelos uma chamada ordinal ou condicional pode ser feita.

Um nodo (ou diagrama) pode ser compartilhado entre muitos modelos. A figura 2.22 representa esses compartilhamentos. Observa-se que essa provisão permite precisa representação dentro de uma global alternativa de projeto. Adicionalmente, facilidades para qualquer classe de problema pode ser modelada uma vez e aplicada em muitos lugares.

2.5 A Técnica de projeto: (DT)

"Design Techniques (DT)" É uma técnica de projeto que orienta e controla o processo de desenvolvimento fazendo uso da linguagem diagramática de SA, para a geração de documentos e entendimento do assunto analisado.

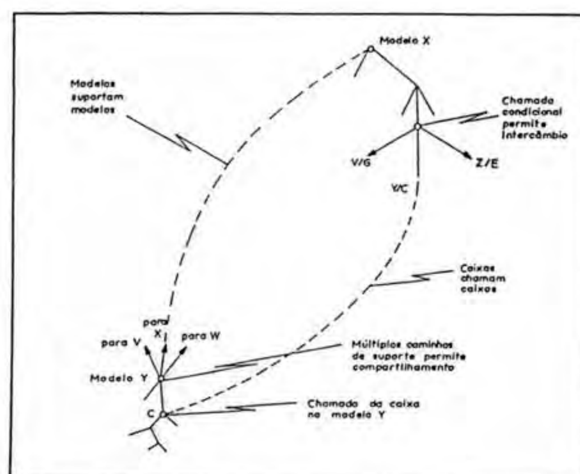


Fig. 2.22 Comutação e compartilhamento de modelos via chamada

O método SADT, tanto através da SA quanto da técnica de projeto DT [ROS 85] não pretende resolver problemas, sendo somente um mecanismo de expressar, elicitar, entender, manipular e verificar opções de solução de problemas.

DT - A Técnica de Projeto do SADT oferece uma base técnica para o desenvolvimento e gerenciamento de um modelo de sistema, através do ciclo autor/leitor para a revisão dos diagramas. Os papéis para esse ciclo definido anteriormente na secção 2.3 é suportado por revisões feitas por um comitê técnico, que também tem a tarefa de resolver os aspectos técnicos onde forem necessários. Embora se identifique um grande número de papéis como citado na secção 2.3; não é necessário um grande número de pessoas para executá-lo [BIR 85]: duas pessoas podem usar o método SADT com sucesso.

2.6 Análise, projeto e implementação usando SADT

O procedimento completo de análise, projeto e implementação, usando o SADT, pode ser sumarizado informalmente [BIR 85] como descrito a seguir:

1. O modelo de ciclo autor/leitor é iniciado pelo modelo de atividade.

2. Uma vez fixado o modelo de atividade (porém antes da decisão do projeto começar a ser feita), um ciclo de autor/leitor é ajustado indo para o modelo de dados.

3. Quando os ciclos para os dois modelos são estabelecidos, eles podem ser conectados, devendo descrever exatamente o mesmo sistema, e logicamente serem duais.

4. Para o nível do modelo de atividade corrente, mecanismos são identificados, que podem necessitar existir em um sistema de computação, se o nível for para ser implementado diretamente dos diagramas.

5. Estes mecanismos são inseridos nos diagramas.

6. O ciclo autor/leitor será estabelecido novamente em cada novo nível de decomposição.

7. Quando todas as caixas são representadas pelos mecanismos, o sistema pode ser codificado dos diagramas junto com as especificações de dados, resultantes do modelo de dados.

8. Uma vez que estejam realizados os passos 1 - 7,o sistema pode ser inspecionado para verificar ineficiências que admitam alguma forma de otimização externa.

A técnica de projeto do SADT claramente tem sua base estabelecida no ciclo autor/leitor, utilizado durante o desenvolvimento e manutenção do sistema. Diagramas ou partes de modelos são criados por "autores" e distribuídos para revisão. Um autor produz "kits" que podem ser compostos por diagramas, textos, glossário e qualquer outra informação considerada relevante. A revisão é feita por "comentaristas" e "leitores". Existe uma diferença entre um comentarista e um leitor: espera-se comentários do primeiro, não do segundo. A revisão é feita por escrito, em uma cópia do "kit" sendo formada por comentários que são enviados ao autor. O autor responde a cada um dos revisores, utilizando a mesma cópia do "kit". Caso o comentarista concorde com a resposta do autor, arquiva a cópia do "kit", caso contrário, é marcada uma reunião para solucionar as diferenças. Se a reunião não alcançar resultados satisfatórios, o problema é enviado para a decisão das chefias que podem optar pela re-análise do problema por parte do autor.

O criador do SADT aconselha que uma análise seja feita por um autor através de dois modelos complementares: um modelo funcional e outro sobre a informação. Em outras palavras, a orientação do primeiro modelo é a de dar relevância às atividades ou funções do sistema, enquanto que o segundo modelo dá relevância às informações ou estruturas de dados. Um modelo SADT completo, segundo [ROS 83a], deve conter:

- ACTIGRAMAS (Diagramas do modelo Funcional);
- DATAGRAMAS (Diagramas do modelo de Informação);
- FEOs ("For Exposition Only" - Diagramas utilizados para Comentários);
- TEXTO (Acompanhando cada diagrama com explicações em linguagem natural);
- ÍNDICE DE NODOS (cada diagrama é um nodo da árvore que representa o desenvolvimento da solução);

2.7 SADT x SOFTech

O método SADT é de propriedade da SOFTech [LEA 88], sendo oferecido para a comunidade usuária na forma de: análise, projeto e desenvolvimento e/ou como transferência de tecnologia.

Para a forma de análise, projeto e desenvolvimento, profissionais da SOFTech executam um contrato básico de ação, como autores, enquanto o cliente participa como usuário/leitor, a fim de solucionar dificuldades e problemas de análise e projeto do sistema.

A forma denominada de transferência de tecnologia é feita através do arrendamento e licença a SOFTech, a qual provê um pacote de serviços, incluindo um curso de três semanas de treinamento, doze semanas de suporte da aplicação e monitoramento periódico. O SADT conta com grande aceitação por parte de grandes usuários, que obtiveram os serviços de SOFTech no processo de aquisição

da tecnologia. Dentre os muitos usuários do método [LEA 88] pode-se citar: Center for Naval Analysis,Federal Reserve Bank of Boston, ITT, Mitre Corporation,U.S. Air Force, U.S. Army, U.S. Navy,Boeing Comercial Airplane Company, Bell Telephone Manufacturing Company, General Motors Corporation, Kongsberg Vaapenfabrik, Standard Electric e a Xerox.

2.8 Regra de ativação

A Regra de Ativação do SADT é considerada como parte do núcleo gráfico da linguagem [DIC 81], e tem como propósito apresentar a informação sobre as quais combinações de entradas e controles ativam a função representada pela caixa, e, por conseguinte, produzem qual combinação de saída. Evidente que em conjunto representam a potencialidade de paralelismo ou de seqüenciamento entre as atividades de um modelo dispostas em um diagrama.

Algumas observações importantes devem ser feitas antes de se falar propriamente de regra de ativação no SADT.

Obs. 1 Em um modelo SA, as setas representam Relacionamentos obrigatórios e não simplesmente fluxo de dados.

Obs. 2 A origem de uma seta que entra em uma caixa precede a esta em algum sentido. A topologia da rede implica em relações de precedência de cada seta, o que pode indicar seqüencialidade ou paralelismo de atividades. Em algum momento por alguma razão, pode ser necessário impor um "seqüenciamento" de atividades em um diagrama.

A regra de ativação é normalmente aplicada a diagramas de atividades, quando o método SADT é utilizado como método de especificação na fase de projeto de software. A notação para regra de ativação proposta em [DIC 81] fornece uma maneira de comunicar ambas informações (seqüenciamento e paralelismo), que devem ser mostradas em mais detalhes em um modelo.

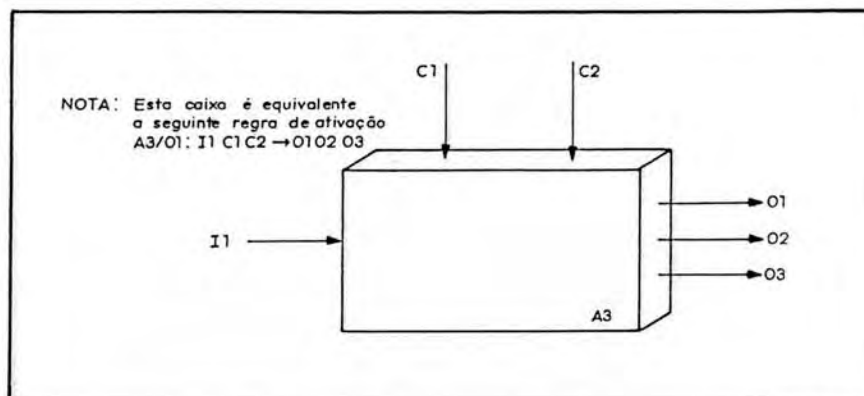


Fig. 2.23 Notação para regra de ativação

As regras de ativação apresentadas em [DIC 81] utilizam lógica de primeira ordem e notação de máquinas de estado finito, sendo agregadas no diagrama, próximo as caixas a que correspondem, permitindo ao projetista, uma forma agradável de leitura.

A forma geral de indicar uma regra de ativação é [DIC 81]:

<Código-de-Caixa> / <Número-da-Regra> :
<Pré-Condicao> ---> <Pós-Condicao>

onde:

<Código-de-Caixa> = Nome da caixa para a qual a regra e aplicada.

<Número-da-Regra> = É uma numeração sequencial de regras.

<Pré-Condicao> = É um conjunto de combinações formada pelos rótulos das setas de entrada e de controle, neces sários para a ativação da caixa.

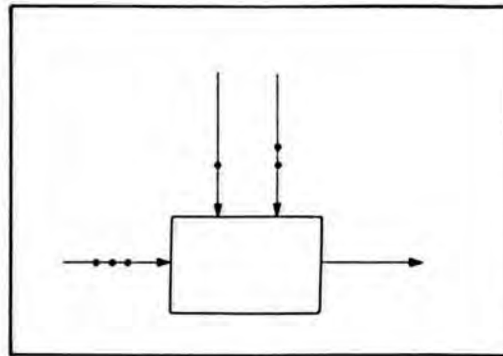


Fig. 2.24 Atividade dispara pela regra padrão

```
<Pós-Condicao>        = É o conjunto formado pela com
                        binação dos rótulos das setas de
                        saída, produzido pela ativação
                        da caixa.
```

As pré-condições de uma regra de ativação podem especificar a ausência de entradas ou controles através do uso de uma barra horizontal, colocada sobre os rótulos apropriados. Se as entradas ou controles forem especificadas com uma barra horizontal colocada sobre cada um dos respectivos rótulos nas pós-condições, isso significa dizer que aquelas entradas ou controles são "consumidos" pela ativação da caixa, necessitando serem gerados para a próxima ativação da atividade. A regra de ativação pode ser usada para fornecer informações ao leitor que desejar mais detalhes acerca do disparo de uma atividade. Caso ela não seja explicitamente determinada, com o uso da notação aqui apresentada. Será assumida a Regra Padrão de ativação de uma caixa, que determina que o disparo de uma atividade só poderá ocorrer, quando todas as suas entradas e controles estiverem presentes. Isso estabelece a condição necessária para que de forma padrão qualquer atividade produza todas as suas saídas. A figura 2.24 ilustra isso.

A importância de ser possível estabelecer uma regra de ativação não padrão para uma atividade, levanta-se quando, algum sub-conjunto de entradas e controles for suficiente para produzir um subconjunto de saída de interesse do projetista, na solução, ainda que parcial, de seu problema. A figura 2.25 ilustra um exemplo de uso de regra de ativação, em uma diagramação, que descreve o problema

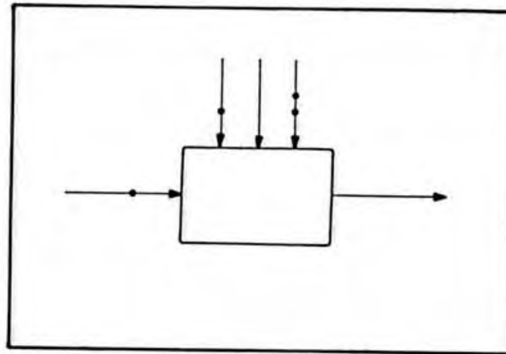


Fig. 2.25 Atividade não dispara pela regra padrão

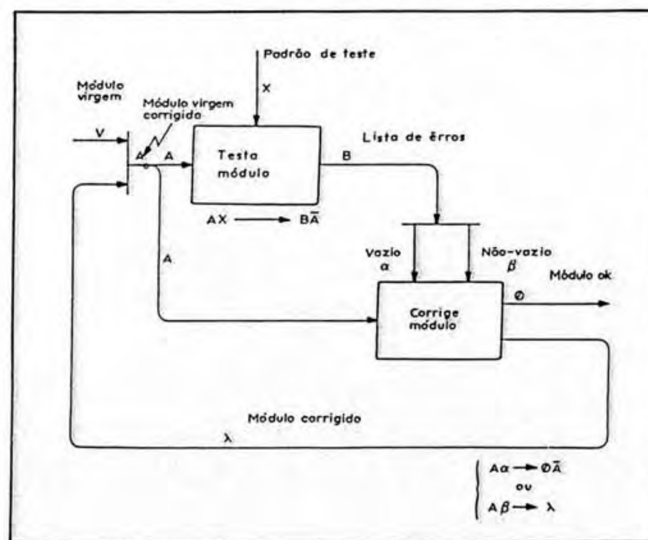


Fig. 2.26 Exemplo de regra de ativação

de testar módulos de software.

O que figura 2.27 apresenta é uma regra de ativação geral, a notação de todas as entradas e controles domínios da função de ativação, que leva a todas as saídas, contradomínios da função. Considerando que o projetista estabelece duas regras válidas para A3:

Regra-1 : A presença da entrada $I1$ e do Controle $C1$ é suficiente para produzir a saída $O2$.

Regra-2 : A presença da entrada $I1$ e do Controle $C2$ é suficiente para produzir as saídas $O1$ e $O3$.

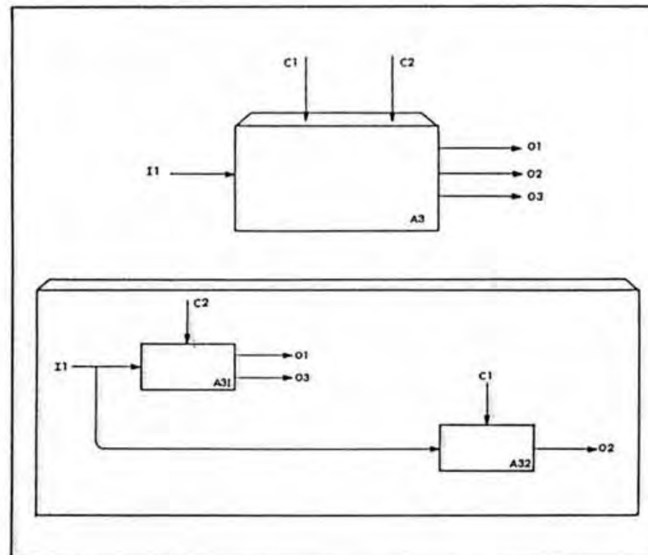


Fig. 2.27 Detalhamento da atividade A3

Para que essa afirmação seja válida, o lado interno da caixa A3 ao ser detalhado deverá justificar esse conjunto de pré e pós condições. A figura 2.27 mostra o detalhamento da A3.

Na ausência de detalhamento, as equações lógicas da regra de ativação, podem ser usadas, de uma maneira mais confortável pelo projetista, relacionando entradas, controles e saídas. As regras a seguir, especificam precisamente o detalhamento mostrado na figura 2.27.

$A3/01\!: I1C1\text{—}O2$

$A3/02\!: I1C2\text{—}O1O3$

Nesta secção já discutimos que uma pré-condição pode ser especificada com a ausência de entradas ou controles, colocando-se uma barra vertical sobre o nome do "label". Essa mesma notação pode ser utilizada em uma pós-condição, o que significa o cancelamento de uma pré-condição a qual é parte de uma pós-condição. O termo apropriado aparece na pós-condição com a barra de negação. Ainda no contexto da caixa A3 poder-se-ia escrever:

$A3/01\!: I1\overline{C1}\,\overline{C2}\text{—}O2$

$A3/02: I1\overline{C1}C2—O1O3$

$A3/03: I1\overline{C1}C2—O1O3\overline{I1}$

Mostra-se na figura 2.28 um diagrama SADT [DIC 81] de projeto detalhado, apresentando paralelismo, e verifica-se como a aplicação de uma regra adequada de ativação para algumas atividades, permite fazer o seqüenciamento destas. Nesse diagrama as atividades "Criar Entrada" e "Determinar Prioridade" são executadas em paralelo. A esse paralelismo explícito, o projetista pode impor um seqüenciamento de atividades. Para indicar que a atividade "Determinar Prioridade" deve ser executada antes da atividade "Criar Entrada", a seguinte regra de ativação impõe a sequência desejada.

$A4/01: DK—\overline{S}M$

$A2/01: SFG—\overline{S}H$

A saída S da atividade "Determinar Prioridade" é artificial, indicando que a função foi executada, e S é então consumida pela atividade "Criar Entrada". A saída artificial S foi criada somente como um artifício para impor seqüencialidade, como mostra a linha pontilhada no diagrama.

As regras de ativação devem ser consistentes em um modelo. Segundo [DIC 81] é possível verificar se as regras de ativação em um diagrama correspondem as regras de ativação na caixa pai, mas o artigo não informa como fazer tal verificação.

Na literatura consultada [DIC 81],[BIR 85] e [TOL 89] acerca de pré e pós condição no SADT, para a realização desta dissertação, não existem considerações com relação a interpretação que deve ser dada a :

1. Inexistência de uma entrada ou controle nas pré-condições de uma regra de ativação.
2. Estado das interfaces não especificadas como consumidas nas pós-condições de uma regra de ativação.

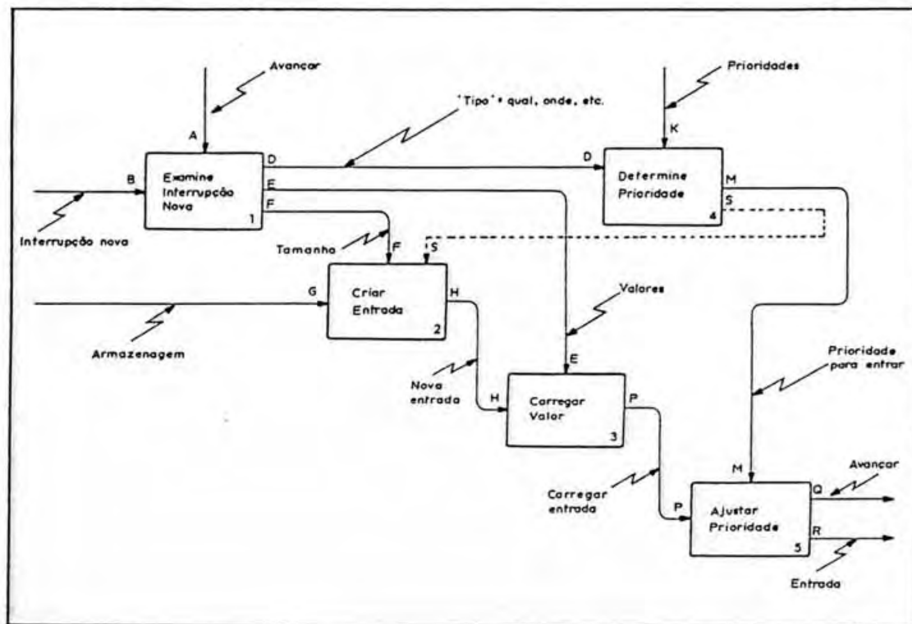


Fig. 2.28 Diagrama com seqüencialização [DIC 81]

3. Possibilidade de validação de duas pré-condições de regras de ativação diferentes de uma mesma caixa. Para esse caso são válidas duas interpretações:

   (a) A especificação das interfaces deve ser complementar de uma regra em relação às demais, não permitindo a validação de regras de ativação diferentes num mesmo instante.

   (b) A ordem de especificação das regras deve ser utilizada para a escolha de uma única regra.

Diagramas de projeto detalhado, anotado com regras de ativação, e, informação quantitativa sobre as características de atividades e dados, fornecem as bases para se produzir especificações de módulos, a partir dos quais o código vai ser gerado.

2.9 SADT interfaceado com outros métodos

O método SADT é frequentemente associado a outros métodos e ferramentas, como parte em um processo de desenvolvimento de software. Como a maioria dos usuários do SADT o utiliza na forma de diagramas de atividades, existem hoje algumas experiências de sucesso, de seu interfaceamento com PSL/PSA, Projeto Estruturado de Yourdon, Método de Jackson, Warnier-Orr, HIPO, Nassi-Schneiderman, Redes de Petri, Linguagens PDL-like e últimamente, há uma proposição em [BIR 85] de fazer seu interfaceamento com o método VDM. Para as aplicações de modelagem de dados via diagramas de dados, o SADT já foi interfaceado para a área de banco de dados [ROS 85] com métodos como ER - entidades e relacionamentos, Bachman-Style, Codasyl, IDEF-1 e projeto de banco de dados relacional. A SOFTech e a Mitre Corporation [ROS 85] tem realizado esforços gráficos, buscando uma maneira de ligar os diagramas SADT com os pacotes da linguagem ADA, em ambientes de desenvolvimento de software.

[DIC 81] descreve alguns dos benefícios de incorporar o SADT a algumas técnicas bem conhecidas da engenharia de software. As técnicas são Projeto Estruturado de Yourdon, Método de Jackson e a parte de projeto modular de Parnas.

2.10 Considerações finais

O SADT é um método bastante geral, sendo aplicável a uma ampla gama de problemas. Em termos práticos é conhecido como aplicável a definição de requisitos, no ciclo de vida convencional de desenvolvimento de software. A forma final da definição de requisitos em SADT pode ser considerada como um contrato entre usuário e a equipe de desenvolvimento [BLA 83], o que atinge um dos objetivos de métodos de especificação de software em geral. Para que o SADT seja introduzido em uma organização, não existe a necessidade de criar uma padronização para acomodá-lo.

A abrangência do método SADT é um fato a ser destacado, o seu ferramental permite que se modele assuntos de natureza filosófica, política, social, legal, científica ou artística, e esses modelos podem efetivamente comunicar as idéias

a outras pessoas [ROS 85]. Em [ROS 83] e [ROS 83a] afirma-se que o método pode ser aplicado durante todo o ciclo de vida de um projeto de desenvolvimento de software, modelando o ambiente e operações correntes, os requisitos, as especificações de projeto, o plano de testes, o plano de conversão, as operações novas o ambiente novo, além do desenvolvimento propriamente dito.

A especificação funcional de um sistema através do SADT [BIR 85] vai consistir em uma coleção completa e consistentes de modelos de pontos de vista, cada um sendo uma hierarquia de diagramas SADT e texto explanatório.

O conjunto de construções do núcleo gráfico do método é grande, mas na prática utilizam-se poucos. Eles foram em sua grande maioria, "pensados" para uma época em que se projetava sistemas, unicamente com o uso de papel, lápis e borracha. A documentação gerada pelo método é difícil de ser mantida manualmente; para grandes projetos é impraticável sem o apoio de ferramentas automáticas.

Em um levantamento de métodos correntes para definição de requisitos [LEI 87], SADT é o método mais citado. A força do SADT está em fornecer uma rica e precisa linguagem gráfica SA para a modelagem de sistemas, e assim, é mais aplicável para a definição e a elicitação de requisitos de análise.

Manuais da SOFTech [LEI 87] apresentam formas de como coletar a informação para aplicação do método, com base na idéia de fazer listas de atividades e de objetos que compõem o sistema. Em termos de literatura, há uma certa carência de material sobre o SADT em geral, sendo a fonte mais citada o clássico artigo de Ross [ROS 83a], e essa dificuldade é maior em relação aos modelos de dados do SADT. Em [LEI 83] encontra-se algumas heurísticas sobre como usá-los. Uma crítica muito séria em relação ao método é que ele não reconhece a importância do princípio de generalização, o que não é incomum em datagramas, para ter um dos objetos decomposto em sub-classes.

O SADT não força o projetista a aceitar um novo, ou até mesmo um particular método de projeto. Particularmente ele disciplina e direciona o pensamento do projetista, enquanto possibilita uma norma para desenhar, ver, comparar e aplicar alternativas em uma forma racional, que ajuste-se para a específica condição

do projeto.

Em [SIE 85] encontra-se a crítica de que ao método faltam mecanismos de controle e também o problema de tracejamento (navegação) em especificações. Uma alternativa para tratar esse problema proposta no método CORE [MUL 79] é fornecendo diagramas hierárquicos adicionais.

A validação de uma especificação SADT é feita informalmente através do ciclo autor/leitor. Esse mecanismo adquire mais consistência quando dispoe-se de um modelo de atividades e de seu dual modelo de dados, já que essas diferentes visões reforçam o entendimento do problema, reconhecendo-se mais facilmente as inconsistências do modelo desenvolvido.

As setas podem ser vistas como canais de comunicação e servem para diferentes propósitos. A grande inovação do SADT é a diferenciação entre entrada, controle (restrição) e o uso de mecanismo. Nem o método de Gane para análise estruturada - SSA [GAN 79] ou o de Tom De Marco [DeM 78] apresentam essa característica. O fato de atividades em um modelo apresentarem-se sempre dispostas na diagonal principal em número mínimo de três e em máximo de seis dá ao SADT uma elegância e disciplina também não oferecida por [GAN 79] e [DeM 78]. Não se trata só de uma questão de beleza, mas legibilidade de uma especificação. Os conectivos do SADT é outro conjunto de construtos, que acrescenta exatidão a uma especificação, além de enriquecer a sua semântica, mesmo considerando que a linguagem do SADT é composta de elementos gráficos.

O método SADT trouxe novos caminhos para a área de análise de problemas, definição de requisitos e especificação funcional. Isso foi possível porque o método permite rigorosa expressão de um alto nível de idéias que anteriormente, eram vistas como "nebulosas" para se tratar tecnicamente. Em pouco tempo, o SADT fez escola, a partir do surgimento de alguns métodos, ferramentas e linguagens como CORE [MUL 79], EDDA [TRA 80], o método TAGS [SIE 85] e a linguagem RML [GRE 83]. Todos esses métodos têm sido aplicados aos mais variados problemas, nas mais diversas áreas do conhecimento humano.

O poder de anotação e expressão para requisitos do SADT foi adotado como base para um sistema de aquisição do conhecimento acerca de problemas, na

tese de doutorado de Grenspan [GRE 83], onde o SADT é combinado com a linguagem RML de modelagem de requisitos. Essa é a experiência pioneira de utilização do método de Ross em Inteligência Artificial. [GRE 83] reconhece que a formalização do SADT é muito difícil de ser estabelecida.

O SADT não suporta uma forma de experimentação (simulação) ou execução da especificação do sistema nele descrito via um protótipo. Essa impossibilidade está presente até hoje, porque ainda não foi publicada nenhuma representação formal de seus construtos gráficos fundamentais. Para que uma especificação SADT possa ser executada por simulação, é necessário que sejam definidas semânticas para as principais construções do método, aquelas obrigatoriamente presentes em qualquer diagramação SADT. Esta dissertação define um domínio semântico em VDM para a construção de uma ferramenta gráfica baseada num subconjunto da linguagem do método. Define e especifica formalmente a semântica e a simulação de especificações diagramáticas SADT.

3 A INTER-RELAÇÃO DO SADT COM SISTEMAS DE FLUXO DE DADOS

3.1 A importância da semântica para o método

O SADT é um método "data flow", similar a Redes de Petri[REI 82], SSA[GAN 79] e De Marco[DeM 78]. A descrição de um sistema em SADT é feita em modo "top down", começando com a descrição geral dos efeitos e propriedades do sistema como um todo e, sucessivamente adicionando detalhes pela divisão do sistema em subsistemas. Esse processo é encerrado, ao atingir-se um grau aceitável de detalhamento da solução . Estrutura-se assim, uma hierarquia de diagramas, que constitui um modelo do sistema com um propósito e um ponto de vista. Em cada diagrama, o conjunto de atividades e setas de interfaces representam as funções do sistema e os dados sobre os quais elas operam e/ou produzem. Essa combinação de construtos da linguagem expressa o entendimento do problema pelo analista de forma clara e precisa, permitindo a comunicação da solução para o usuário. Embora alguns autores considerem a linguagem do SADT complexa para o leigo [LUD 86], essa afirmativa deve ser repensada diante do amplo número de aplicações e usuários hoje existentes.

Até hoje o SADT é classificado [BIR 85],[LEI 87], [LUD 86] e [RIB 90] como método semi-formal. A apresentação do método em [ROS 83a] estabelece uma sintaxe para a linguagem gráfica, tanto em nível de diagrama, estruturado em caixas de atividades com setas e rótulos, como em nível de modelo, com a aplicação

de regras sintáticas hierárquicas (código ICOM, mecanismos de chamada de outros diagramas ou modelos inteiros) bem definidas, que permite por exemplo que de uma caixa pai, possa-se ir para a caixa filho e vice-e-versa. Mesmo reconhecendo que com todos esses recursos, não precisamente estabelecidos para métodos similares, e que o uso do SADT determina uma disciplina organizada de ação e pensamento, a semântica do método não foi até o presente momento publicada [ROS 83], [ROS 83a], [DIC 81], [THO 78],[ROS 85], [STR 78], [LEA 88],[TOL 89], [ROU 82],[LEI 83], [ANA 79], [YEH 84], [FRE 81] e [LIS 87]. A semi-formalidade do SADT, impõe por exemplo, o ciclo autor/leitor como único expediente para a validação da especificação . A linguagem gráfica do método anotada nos diagramas deve prover essa possibilidade, daí ser dita uma linguagem de comunicação de idéias [ROS 83a].

A inexistência de uma definição semântica para o SADT dificulta o entendimento pleno das potencialidades oferecidas pelos principais construtos do método. E em consequência, toda a riqueza de uma linguagem para especificação de requisitos pode ser dessa forma desconsiderada. Existem várias publicações de especificação de sistemas em SADT,entre as quais podemos destacar [LUD 86] que apresentam erros, tanto sintáticos como semânticos.

Os métodos mais recentes para especificação de requisitos de software, como VDM [BJO 82] por exemplo, trazem como uma de suas principais características, a forma clara e precisa de descrever as várias entidades manipuladas no processo de criação de software, além de fazer uma exposição de forma correta e objetiva de todos os interrelacionamentos das diversas entidades [COH 86], [GEH 86]. Métodos formais oferecem uma base apropriada para tratar o processo de desenvolvimento em alto nível de abstração, em modo disciplinado confiável e exato.

O SADT tem algumas deficiências. O modelo de sistema descrito em SADT não apresenta um suporte para experimentação (elaboração de protótipos) que permita a validação da especificação, antes de se avançar para as outras etapas no ciclo de vida de desenvolvimento. Devido a esse fato, permanece o ciclo autor/leitor como único referencial de validação da especificação do modelo de sistema descrito em SADT. A experimentação ou prototipação de um sistema descrito em SADT só pode ser feita, a partir do momento em que se definir precisamente a semântica dos construtos do método, e de posse disso, construir um simulador da especificação SADT, para possibilitar a execução simbólica do modelo. Nas secções seguintes,

examina-se as extensões necessárias para "executar" o SADT, comenta-se sobre tipo de dado como um "token", aborda-se o conceito de "dataflow machines" como um modelo teórico para algumas partes deste trabalho e, definem-se regras para ativar uma atividade complementando com uma rápida discussão de simulação.

3.2 Extensões necessárias para "executar" o SADT

A possibilidade de introduzir a experimentação no modelo de sistema descrito em SADT é obtida desde que sejam adotadas às seguintes extensões para o método.

O SADT é uma rede, e como tal pode também ser visto como um grafo. Em [NEU 82],[JON 87] encontram-se interessantes trabalhos onde são discutidos conceitos teóricos de grafos e de fluxos de dados. Nesse contexto o SADT se enquadra perfeitamente, podendo ser visto em sua essência como um grafo dirigido, que pode conter ciclos em alguns de seus nodos. Isso significa dizer que um modelo, quando tomado no todo considerando todos os seus diagramas, pode ser visto como uma grande estrutura de redes. E essa estrutura é essencialmente um grande grafo, onde nodos podem conter outros grafos, e o conjunto todo constitui um hipergrafo. Nessa visão, cada atividade e cada conectivo terão representação através de um nodo do grafo. As setas ligando as diversas atividades e conectivos constituem os arcos, indicando os diversos caminhos de dados na rede. A figura 3.1 a seguir ilustra um diagrama de um sistema fictício, onde cada atividade é denominada simbolicamente com uma letra maiúscula do alfabeto inglês, cada conectivo pela letra "c" seguida de um número sequencial, e cada fluxo por uma letra minúscula. Observe que a atividade central possui um ciclo e os conectivos presentes são pertencentes às categorias de composição e decomposição de dados.

O SADT, visto pela ótica de redes/grafos e "data flow machines" permite que de maneira clara e precisa, possa-se especificar o comportamento dinâmico de muitos tipos de sistemas. Em princípio, permite a representação de sistemas não-determinísticos e paralelos, através da especificação dos estados do sistema e as mudanças entre estados.

É de reconhecimento geral que trabalhar em nível de redes para es-

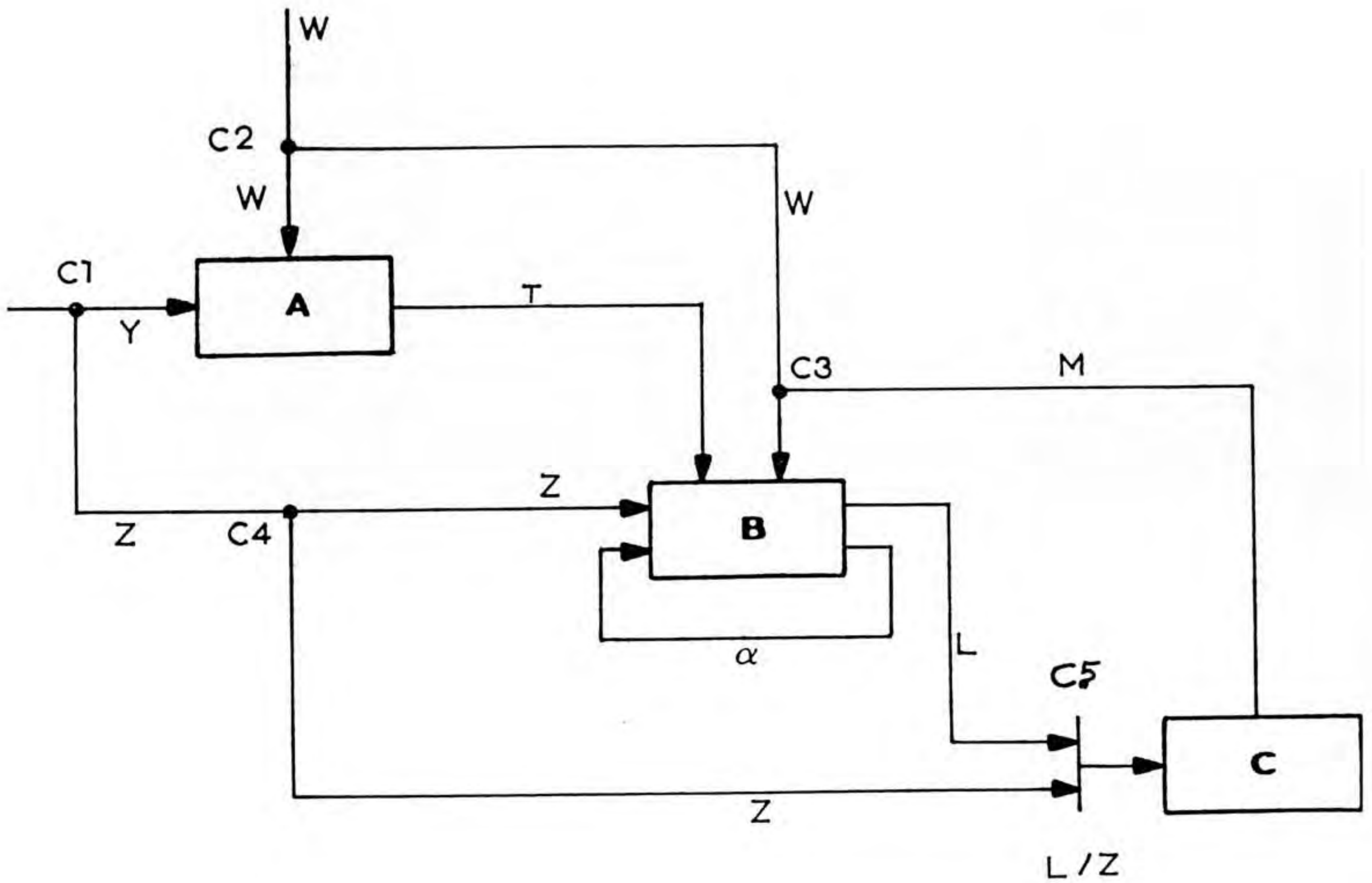


Fig. 3.1 Diagrama de atividades

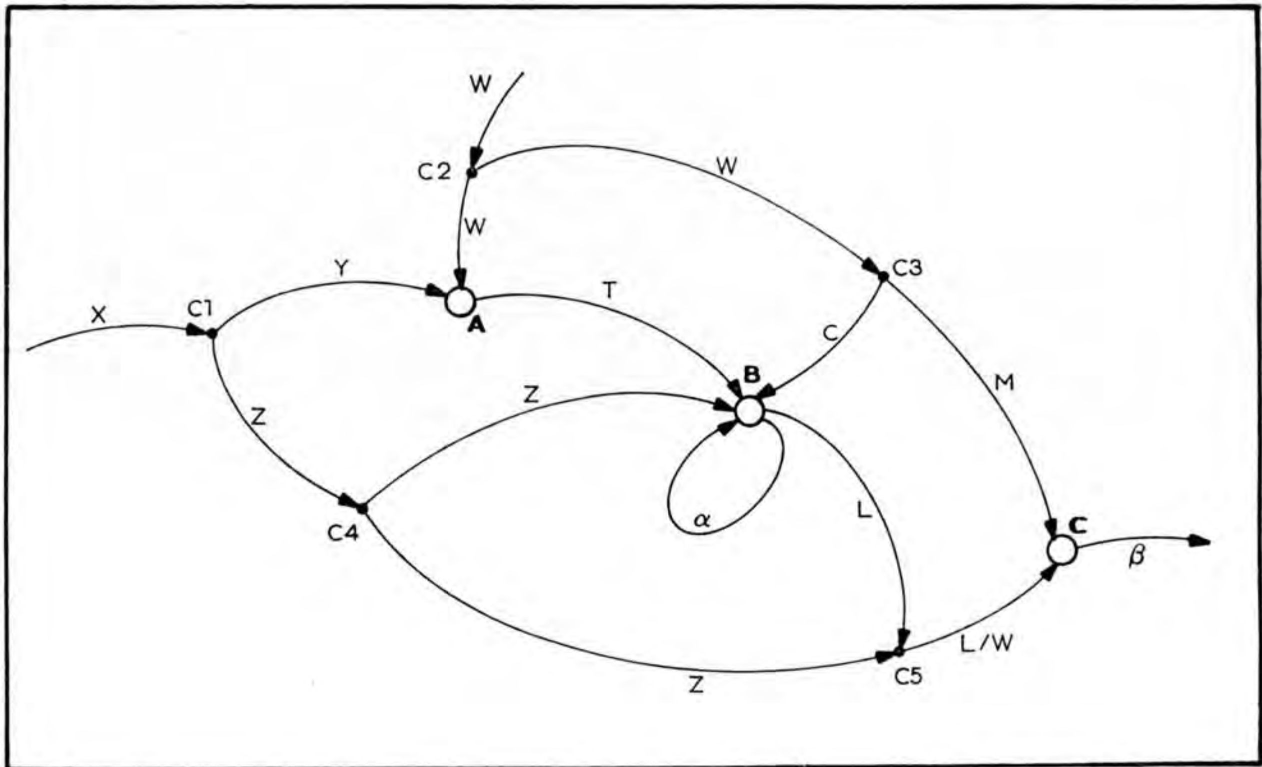


Fig. 3.2 Grafo Correspondente ao Diagrama

pecificar um sistema não é tarefa fácil. As especificações tornam-se demasiadamente grandes e o entendimento do problema muito difícil. O SADT oferece a possibilidade de se trabalhar em um nível mais alto, disciplinado, com limitação da informação em cada nível (diagrama), anotando o modelo com palavras do domínio da aplicação e utilizando conectivos para as operações de interfaceamento das atividades modeladas.

3.2.1 Tipo de dado como um "token"

Em nível da construção de um modelo SADT, a preocupação do projetista é descrever os requisitos do problema em resolução . O modelo completo composto do diagrama pai e os vários refinamentos ligados via códigos ICOM constitui a especificação do sistema, que em um nível mais baixo é a rede, onde são válidas conceituações da teoria dos grafos. O grafo correspondente ao modelo pode ser visto como uma "dataflow machine". Cada identificação de fluxo, que é uma estrutura de dados no SADT, ou mais precisamente um tipo de dado que se movimenta, entrando numa caixa de atividade ou num ponto de conectivo, ou saindo, sendo produzido pelo disparo de uma atividade ou de um conectivo. Essa informação será representada para as necessidades de simulação de uma especificação, como um "token". Uma categoria de objeto para o qual não se procura representação.

3.2.2 O modelo "dataflow machines"

Nesse modelo de "dataflow machine" uma ação só poderá ocorrer se todos os seus "tokens"de entrada estiverem presentes. No grafo da figura 3.3 os nodos A e D encontram-se em condições de disparar. Isso significa que os "tokens" de entrada serão consumidos e serão produzidos "tokens" de saída. [JON 87] diz que formalmente grafos de fluxos de dados são considerados descrições bidimensionais de uma ordenação parcial sobre eventos computacionais. Os grafos de fluxos de dados são estruturados por dependência de dados. Cada nodo nesse grafo representa ações atômicas. Os arcos conectam nodos dependentes unidirecionalmente, mostrando dessa forma a direção da dependência. Esses arcos são conectados para pontos de entrada e saída dos nodos. Os "tokens" são, então, os tipos (estruturas) de dados

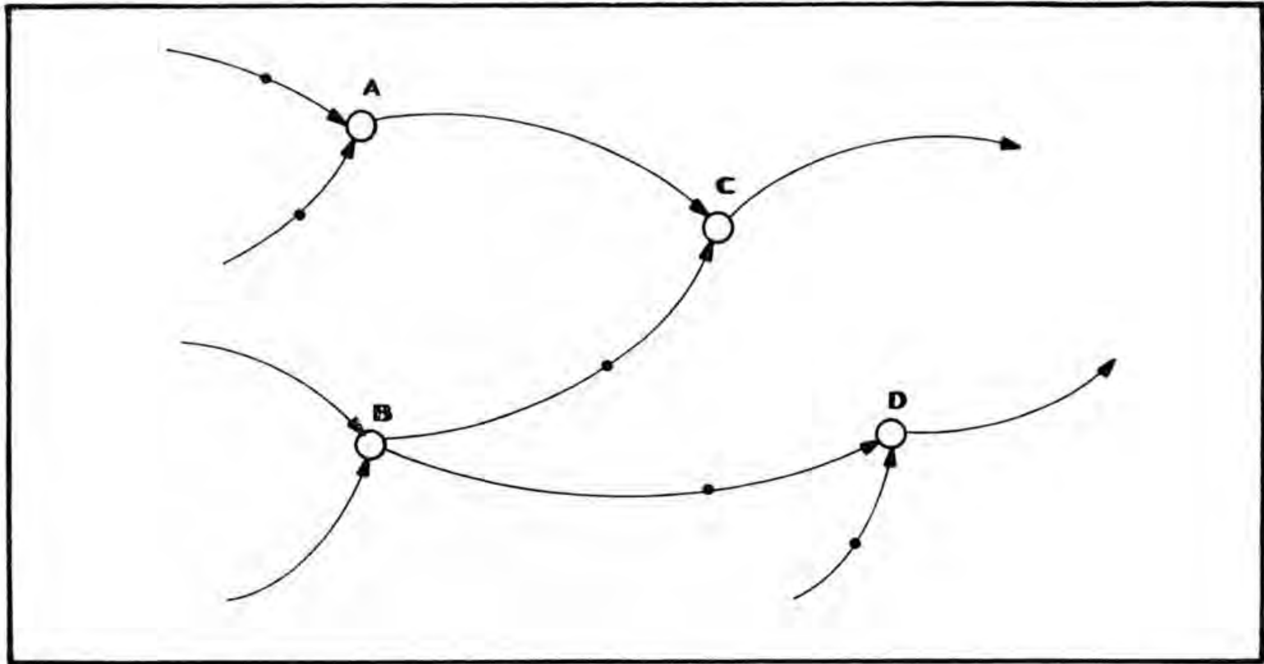


Fig. 3.3 Um exemplo de "dataflow machine"

que navegam ao longo dos arcos. A regra de disparo é geral e válida para qualquer nodo no grafo. Se um só fluxo de dados de entrada não apresentar "token" o nodo não vai disparar. O disparo determina o consumo de entrada(s) e a produção de saída(s) mais tarde. O "token" resultante será produzido sobre um arco(s) de saída(s). Essa é a única restrição sobre execução de tarefas em "dataflow machines".

O conceito de máquina "dataflow" surge da consideração direta de execução desses grafos. Os dados sempre são representados efetivamente movendo-se ao longo dos arcos. Observa-se que o grafo mostrando a dependência de dado não impede uma ordenação linear. E essa ordenação parcial dirige naturalmente ao paralelismo. Em qualquer instante do tempo, mais de um nodo pode encontrar-se em condições de disparar. Qualquer ou todos os nodos em condições de disparar podem ser ativados em paralelo, desde que eles sejam independentes. É importante voltar a atenção para esses pontos, porque a teoria de redes é o referencial básico para a grande maioria dos métodos de desenvolvimento orientados a fluxo de dados.

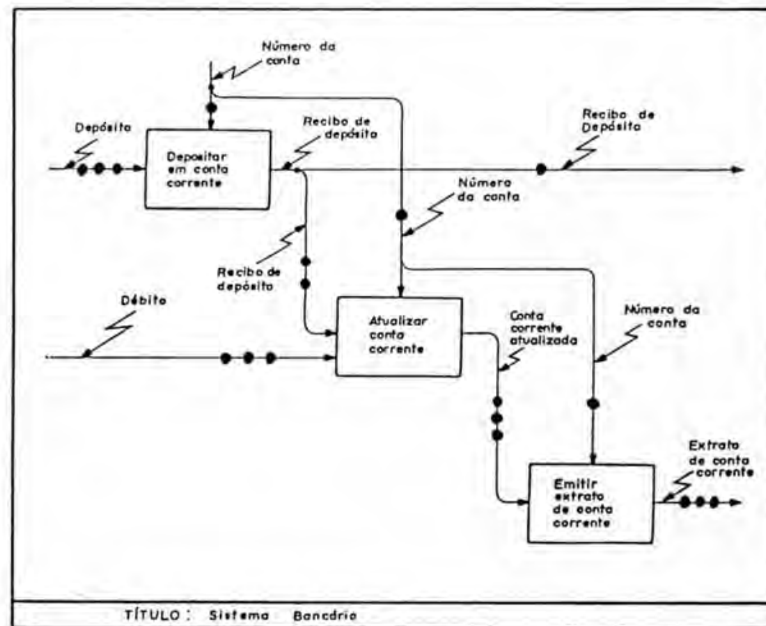


Fig. 3.4 Filas de "tokens" como tipo de dado

3.2.3 SADT e "dataflow machines"

A fundamentação teórica de "dataflow machines" é perfeitamente aderente ao SADT, no sentido de simular o comportamento dinâmico do sistema nele modelado. Nesse sentido, as extensões necessárias ao SADT objetivando a simulação são bem poucas. Primeiro a notação gráfica do método permanece exatamente a mesma. Os arcos continuarão como setas, observando-se que para os propósitos de simulação, nestas setas serão dispostas filas de "tokens" que representam um tipo de dado qualquer em conveniência com sua denominação .

As filas de "tokens" de comprimento vazio (considerado um "default") ou de um elemento de dado ou vários elementos serão dispostas sobre os arcos. Teoricamente cada arco pode comportar um número infinito de "tokens".

3.2.4 Regras de ativação para uma atividade

Para o propósito de introduzir, conceitualmente, extensões que permitam simular a execução de uma especificação SADT é extremamente importante que seja estabelecida uma regra de ativação válida para quaisquer atividades no modelo. Essa regra de ativação torna-se então a lei básica para a "realização" de uma atividade, em particular, de um subsistema e do sistema como um todo.

Diante do exposto, fica estabelecido como regra de ativação o seguinte:

R1 : No conjunto formado pelos fluxos de entrada e controle de uma atividade deverá existir pelo menos um "token" presente em cada fluxo.

R2 : Toda atividade ao ser disparada "consome" um "token" de cada fluxo de entrada e de controle e, após, decorrido um intervalo de tempo sorteado para sua execução, produzirá um "token" em sua saída(s).

R3 : Uma atividade primitiva é atômica, no sentido de que ela dispara como descrito em R1 e R2 e, caso contrário, não haverá "consumo" ou "produção " de "token" pela atividade.

Qualquer atividade SADT satisfazendo a regra R1 ao ser disparada "consome" um "token" de cada entrada e de cada controle, e após seu tempo de processamento produzirá um "token" que será enfileirado no(s) fluxo(s) de saída(s).

No modelo proposto para simulação do SADT, não se determina quanto tempo passará para que uma atividade fique pronta para disparar; pois esse tempo é indeterminado. Um exemplo da vida prática é a compra de passagens em uma companhia aérea. Para que seja realizada a venda ao cliente pela companhia, necessita-se da solicitação deste, o que é uma entrada para o sistema. O que não se pode "a priori" determinar quando acontecerá. Os controles de tal atividade seriam a data de viagem, disponibilidade de poltronas, fumante ou não, etc. Mas essa atividade ao ser disparada, tem sua faixa de tempo mínimo e máximo admissível tanto pela companhia como pelo cliente para ser realizada. O modelo de simulação proposto nesta dissertação prevê que para cada atividade primitiva do SADT, ao ser

disparada, é feito um sorteio de um tempo (em uma distribuição de probabilidades que pode ser Normal ou Poisson) ao fim do qual se produzirá os "tokens" de saída da atividade, simbolizando o tipo de dado gerado.

3.3 A simulação discreta no modelo SADT

A simulação discreta do modelo SADT com o disparo da atividade inicial ocorre dentro de um tempo específico de simulação do modelo como um todo. O consumo e a produção de "tokens" pelas diversas atividades na rede nos diferentes níveis hierárquicos do modelo dar-se-á em instantes discretos do tempo de simulação, o que caracteriza os vários estados no sistema, sub-sistemas e atividades modeladas. Os conectivos, mesmo sendo micro-atividades de composição e decomposição de dados, tem ocorrências instantâneas. É importante observar que os elementos que constituem a rede SADT, representado pelos diagramas, atividades e conectivos, sempre obtêm a rede em um estado, e a levam a outro estado, nos diversos instantes do tempo de simulação do modelo.

A possibilidade de simular uma especificação descrita em SADT, permite que se façam verificações, como por exemplo se o modelo representado pela rede continua ao longo do tempo realizando (ou processando) suas atividades e estabelecendo suas conexões ou, se tal execução para parcialmente ou completamente em determinado instante. É possível verificar se, algum subsistema ou se alguma atividade nunca acontece, se existe "deadlock", ou se o sistema como um todo realiza sua função completa ou parcialmente em um ciclo completo de simulação .

O acréscimo ao SADT de uma visão da base teórica de redes, grafos e "dataflow machines" faz o método ficar muito mais rico. Perde a imprecisão da semi-formalidade, e ganha a exatidão e concisão de um método formal. Supre-se assim uma das deficiências básicas do SADT apontada em [LEI 87], que refere-se a possibilidade de experimentação do modelo construído.

A execução de um modelo SADT por simulação, não significa que se tenha que desenhar um novo diagrama específico de rede. Usa-se a mesma notação como definida em [ROS 83a] e, confortavelmente, especifica-se o problema em uma

ferramenta de apoio à construção de diagramas. Após, é feita a inicialização da rede, informando-se os tipos de dados ("tokens") necessários ao início da simulação, temporização para atividade primitiva (nesse momento pode-se escolher distribuição Normal ou Poisson) e o tempo para simulação do sistema. Esta dissertação pretende fornecer uma base teórica que possibilite a execução da especificação SADT via simulação. Para tal é proposta a definição semântica dos construtos do método, úteis a esse propósito.

A simulação via computador não foi implementada, embora, a classe especificada para a ferramenta forneça subsídios para tal. O que se oferece adicionalmente aqui é a especificação formal da semântica do método, o que possibilita a construção do simulador.

O projetista ao utilizar a notação do SADT para especificar um sistema visando a execução da especificação via simulação, como forma de validar sua especificação, deve ter em mente o seguinte:

1. Os nomes de fluxos de dados no diagrama continuam como definidos em [ROS 83a], mas deve ser observado que essa nomenclatura deve refletir uma estrutura de dados em movimento, a qual pode ser abstraída através de um número natural. Os "tokens" serão representados por números naturais para os propósitos de simulação, sendo dispostos em filas nos diversos arcos do modelo. Isso significa a aplicação da disciplina FIFO no modelo.

2. Atividades serão anotadas também como em [ROS 83a], possuindo nome (um verbo no imperativo), um comentário representado por um texto associado, uma complexidade, ou seja se a atividade é primitiva ou refinada e, se for decomposta, as suas partes não-decomponíveis tem um tempo mínimo e máximo associado a sua execução.

3. Os arcos de chegada tanto para atividades como para conectivos, identificam com um número natural a quantidade de "tokens" enfileirados para serem consumidos pela atividade ou pelo conectivo.

4. Arcos saindo de uma atividade ou de um conectivo também identificam, com um número natural, a fila de "tokens" neles justaposta.

Os diversos fluxos de um diagrama podem ter origem ou destino externo ao diagrama. Para uma atividade refinada, os arcos de origem ou destino externo, na verdade, são internos ao diagrama pai. O único diagrama cujos fluxos de entrada e controle que possuem origem externa e a saída destino externo é o diagrama topo, considerando que ele não represente nenhum subsistema no modelo. No diagrama topo, quando a atividade principal (raiz) dispara, ela não trabalha como uma unidade atômica, e sim ativa seus diversos subsistemas, os quais a definem como um todo. Pode-se dizer que uma atividade topo com diversas entradas, diversos controles e diversas saídas, poderá ser utilizada em diferentes níveis de subsistemas, os quais serão ativados e produzirão suas saídas espalhadas pelo tempo de simulação.

Para o modelo de simulação proposto, uma atividade é um transformador de dados; interessa o fato de que a transformação ocorrerá ou não; que os "tokens" de entrada e controle serão consumidos e os "tokens" de saída produzidos. Como a transformação ocorre, se por meios eletrônicos ou humanos, não vem ao caso no modelo.

O modelo de sistema descrito em SADT e executado via simulação, permitirá a "visualização" do comportamento do sistema, representado pelos diversos estados, nos diversos instantes do tempo de simulação.

Uma atividade não primitiva pode ser disparada, mesmo que nem todos seus "tokens" de entradas e controles estejam presentes. Isso acontecerá, desde que alguma(s) de suas atividades refinadas, satisfaça a regra padrão de disparo, ou seja, a presença de pelo menos um "token" em cada entrada e em cada controle. O que se pode afirmar nesse contexto, é que sempre que uma atividade não primitiva disparar, nem sempre produzirá "tokens" em todos os seus arcos de saída. Tudo depende da configuração existente na conectividade das atividades no diagrama refinado. Mas em síntese, o que ocorre aqui, é o respeito a regra R1 de que uma atividade primitiva vai ser executada somente quando todos os tokens de entrada e controle estiverem disponíveis. Esses "tokens" serão consumidos pela atividade a qual produzirá outros "tokens" no(s) fluxo(s) de saída(s).

4 ABORDAGEM OPERACIONAL PARA CONECTIVOS E ATIVIDADES

4.1 Construtos do método & semântica

A definição semântica precisa dos construtos básicos do SADT, que não foi até hoje apresentada, constitui-se em uma base formal, que permite a execução do modelo via simulação e acrescenta formalidade ao método, que assim passa a ser mais exato e rigoroso no trato dos modelos nele especificados. Essa é uma contribuição muito importante deste trabalho. A utilização do método com o conhecimento da semântica associada a cada construto básico, que produzirá descrições do comportamento de sistemas com muito mais precisão e, ainda, uma base formal para a experimentação do modelo. Lembrar que a impossibilidade de experimentação do modelo de sistemas descritos em SADT é considerada uma das maiores desvantagens de uso do método. A validação de uma especificação é feita até hoje através do ciclo autor/leitor [LEI 87].

A linguagem gráfica do método SA apresentada em [ROS 83a] fornece 40 construções gráficas para utilização na especificação de sistemas. Ross além de apresentar um método excepcional para a definição de requisitos de análise, que naquele momento já contava vários anos de uso no ambiente industrial da Softech, incorporou a este, construções gráficas de apoio a própria atividade de análise. Assim, por exemplo, encontra-se dentre essas 40 construções, notação para evitar o cruzamento de linhas, para setas limítrofes que objetivam representar no diagrama

filho as interfaces associadas ao diagrama pai. Nesse contexto também se enquadra notação específica para indicar que determinada caixa posssui um refinamento, seta bidirecional, seta mista, "tunneling", nota de rodapé, glossário, índice de nodos, formulário, documento e etc.

Hoje, diante dos modernos recursos computacionais, sistemas avançados de comunicação homem-máquina, formulários eletrônicos e biblioteca de objetos por exemplo, boa parte das construções propostas em [ROS 83a] não necessitam mais ser utilizadas. Elas foram pensadas para uma época em que não havia a possibilidade de realizar a função de análise de sistemas com apoio computarizado.

O SADT sobrevive ao tempo, tanto em meios acadêmicos como industriais, porque possui um conjunto mínimo de construções no núcleo gráfico de sua linguagem muito rico em informação e, portanto, poderoso para definição e especificação funcional de requisitos. É desse pequeno conjunto de construções gráficas que este trabalho se ocupa, definindo sua semântica para a consecução dos objetivos estabelecidos.

O SADT é muito rico e bastante complexo. Segundo [CER 86] a experiência tem mostrado que um diagrama pode ser interpretado erradamente por alguém que conheça o método, mas que não seja seu autor. A clássica utilização do ciclo autor/leitor para a validação de uma especificação SADT acaba sendo uma estratégia no método de deixar o projetista expressar tudo o que desejar e, posteriormente, remover as inconsistências na revisão.

A especificação de um sistema em SADT é feita construindo-se um modelo explícito, que representa a realidade. A intuição do projetista é o único recurso que ele pode utilizar para mostrar que o modelo construído é equivalente a realidade. Para essa equivalência, não existe prova formal. A intuição cognitiva do projetista é a única coisa na qual ele confia de que sua especificação modela adequadamente a realidade. Como o SADT até hoje apresenta-se como um método semi-formal, isso dificulta ainda mais validar uma especificação. Como dito em [DIC 81] um diagrama é denso em informação que pode ser interpretado erradamente.

A definição da semântica para o conjunto básico de construtos do SADT torna-o mais formal, permitindo assim uma maior aproximação entre um

modelo e a realidade modelada. Obtém-se o significado real e preciso de cada construto, bem como da aplicação do método à especificação de requisitos de um sistema. Isso significa dizer que cada construto possui uma única definição semântica, a qual caracteriza completamente o seu comportamento, seja em plano individual ou em composição com outros construtos. Noutras palavras, o significado de um objeto composto é criado a partir do significado de seus objetos componentes.

O método SADT sobreviveu ao tempo, ao surgimento de outras propostas, pela riqueza informacional que ele apresenta [PRI 84]. Mas a realidade dos fatos, é que muitas pessoas da área desconhecem o potencial do SADT.

[ROS 83a] apresenta, entre as 40 construções gráficas do SADT, um conjunto de seis conectivos e a atividade. A função estabelecida para os conectivos, é interfacear o conjunto de atividades pertencentes a um diagrama. O conectivo é um construto rico em informação, tem um significado e sua adequada utilização simplifica, torna mais funcional, e adiciona legibilidade a uma especificação . Cada conectivo do SADT realiza uma função dentro do diagrama, seja ligando atividades ou conectivos entre si. A cada atividade em um diagrama, o analista atribui funções específicas, as quais interessa às entradas (entrada + controle) e às saídas produzidas. Os conectivos ligam as diversas atividades do diagrama, realizando micro-atividades de composição ou decomposição de dados, nos tipos de dados presente no(s) seu(s) fluxo(s) de entrada(s). Além disso, as combinações entre conectivos e atividades em um diagrama, determina completamente, a topologia da rede, podendo assim ficar completamente determindo o sequenciamento ou paralelismo de atividades no diagrama SADT.

4.2 A linguagem PSDL na abordagem operacional dos construtos

Para expressar a semântica operacional dos conectivos e posteriormente da atividade, uma linguagem "PSDL-like" [LUQ 88] foi escolhida. As razões da escolha dessa linguagem vinculam-se ao fato de que ela possibilita tratar a abstração de uma forma elegante e simples. Além disso, a linguagem PSDL suporta decomposição de funções, dados e abstração de controle. Sendo PSDL uma linguagem para sistemas

baseados em fluxo de dados e permitindo a especificação "caixa preta" para cada componente. Isso a torna adequada para o tratamento operacional dado para os costrutos do SADT apresentados neste capítulo.

A linguagem PSDL [LUQ 88] fornece a abstração de "operadores". O operador em PSDL é uma função ou uma máquina de estado. Quando um operador dispara, ele lê um objeto de dado de cada entrada(s) e escreve mais um objeto de dado sobre cada saída(s). O objeto de saída produzido quando uma função dispara depende somente do corrente estado do(s) valor(e)s de entrada(s).

4.3 A definição da semântica dos conectivos

O SADT possui seis conectivos, sendo três de composição de dados e três para decomposição de dados. Em [ROS 83a], o conjunto de conectivos são apresentados com os seguintes propósitos:

- serem explícitos sem introduzir desordem, utilizando para tal o conceito de pipelines;
- serem concisos e claros utilizando o conceito de cabos e ligações múltiplas e
- mostrarem exclusão lógica, utilizando o conceito de alternativas explícitas.

No primeiro propósito enquadram-se os conectivos denominados de BRANCH e JOIN, no segundo os conectivos BUNDLE e SPREAD e finalmente no terceiro propósito, os conectivos ORJOIN e ORBRANCH.

Nesta secção apresenta-se estudo detalhado de cada conectivo, definindo a semântica em modo operacional através de uma linguagem "PSDL-like" visando à simulação. Será apresentado para cada construto, a sua especificação na linguagem e a descrição textual de seu significado.

4.3.1 Definição semântica do conectivo BRANCH

O conectivo BRANCH de decomposição de dados ou distribuição [ROS 83a] e tem como propósito ser explícito sem provocar desordem no diagrama. Esse construto se estabelece através da representação do conceito de pipeline.

```
operador conectivo BRANCH
especificacao
     entrada  A^i                     : fila de tokens
     saida    A^o1, A^o2, A^o3,...,A^on: fila de tokens
     descricao
          se  A^i > 0
             entao
               A^o1 <-- * + 1
               A^o2 <-- * + 1
               A^o3 <-- * + 1
                      :
                      :
               A^on-1 <-- * + 1
               A^on  <-- * + 1
               A^i   <-- * - 1
             senao null
fim operador conectivo BRANCH
```

4.3.2 Descrição do significado de BRANCH

A elaboração da semântica do conectivo BRANCH é produzida como segue: Em cada fluxo de dado de entrada ou de saídas, existe uma fila (vazia ou não) de "tokens". O nome de cada fluxo, simboliza o tipo de dado presente. Para o conectivo BRANCH, o tipo de dado presente na entrada será replicado nos vários fluxos de sua saída. A semântica associada a essa construção é a seguinte: primeiro é verificada a existência da fila de "tokens" no fluxo de entrada do conectivo. Caso essa fila seja não nula, então retira-se um "token" da fila disposta no fluxo de entrada e, a cada fila disposta nos diversos fluxos de saída, as quais podem conter ou não "tokens", é adicionado

um novo "token" que tem o mesmo nome dos da fila de entrada. O símbolo " * " representa a quantidade atual de "tokens" presente em um fluxo.

4.3.3 Definição semântica do conectivo JOIN

O conectivo JOIN é de composição de dados, ou como apresentado em [ROS 83a] para mostrar distribuição. Na verdade JOIN realiza a operação inversa de BRANCH. O propósito do conectivo é ser explícito e evitar confusão em um diagrama. O construto é estabelecido pelo conceito de pipelines (dutos), condutores e fios.

```
operador conectivo JOIN
especificacao
     entrada    A^i1, A^i2, A^i3, ..,
                A^in-1, A^in         : fila de tokens
     saida      A^o                  : fila de tokens
     descricao
          se A^i1 > 0  e  A^i2 > 0  e
             A^i3 > 0  e  ...       e
             A^in-1 > 0  e  A^in > 0
             entao
               A^o <-- * + 1
               A^i1 <-- * - 1
               A^i2 <-- * - 1
                   :
                   :
               A^in-1 <-- * - 1
               A^in <-- * - 1
             senao null
fim operador conectivo JOIN
```

4.3.4 Descrição do significado de JOIN

A elaboração da semântico do conectivo JOIN é produzida como segue: Nos fluxos de entradas e saída existem filas de "tokens" representando um tipo particular de dado, que é associado ao nome do fluxo. No caso do conectivo JOIN, os nomes dos fluxos de entrada e do fluxo de saída são exatamente os mesmos. A semântica do construto é definida da seguinte maneira: primeiro é verificada a existência da fila de "tokens" nos diversos fluxos de entrada; se em cada fluxo a fila for não vazia, então a fila de saída do conectivo terá o seu tamanho atual "*", acrescida de um elemento (nada impede que a fila de saída encontre-se em algum momento vazia). Nas diversas filas de "tokens" das entradas tomar-se-á o comprimento atual "*" reduzido de um elemento. Neste conectivo, a simples existência de um fluxo de entrada com fila vazia é suficiente para que ele não realize sua função de composição de dados.

4.3.5 Definição da semântica do conectivo SPREAD

O conectivo SPREAD é de decomposição de dados, grupo ao qual pertence também o BRANCH. O propósito para SPREAD [ROS 83a] é introduzir concisão e clareza nos diagramas. O conceito em cima do qual ele é estabelecido é o de representação de cabos e fios múltiplos a partir de setas. Esse conectivo atua sobre um tipo de dado composto na entrada, apresentando na saída suas partes componentes.

```
operador conectivo SPREAD
especificacao
     entrada   A          : fila de tokens
     saida     B, C, D,
               E, F       : fila de sub-tokens
     descricao
          se A > 0
             entao
               B <-- * + 1
               C <-- * + 1
               D <-- * + 1
               E <-- * + 1
               F <-- * + 1
               A <-- * - 1
             senao null
fim operador conectivo SPREAD
```

4.3.6 Descrição do significado de SPREAD

A elaboração da semântica do conectivo SPREAD é feita, levando em consideração a natureza do conectivo que é de decomposição de dados. A representação gráfica de SPREAD é a mesma de BRANCH, mas o seu significado é outro. O nome do fluxo de dados de entrada, sempre se refere a um tipo de dado composto por outros tipos (sub-tipos). Os nomes presentes nos fluxos de saída referem-se aos tipos de dados que compõem o tipo de entrada. O SPREAD realiza uma separação da informação de entrada. Nesta proposição de semântica para o conectivo, visando a simulação de seu comportamento, o tipo de dado de entrada é simbolizado por uma fila de "tokens" e em cada braço de saída uma fila de "sub-tokens" simbolizando os diversos sub-tipos de dados. A elaboração da semântica do conectivo é interpretada da seguinte maneira. Primeiro é verificada a existência de "tokens" na fila de entrada do conectivo. Se o tamanho dessa fila for superior a zero (fila não vazia), então as filas dos diversos ramos de saída, cada uma composta de "sub-tokens" simbolizando os sub-tipos de dados é acrescida de um elemento ao seu comprimento atual ("*"). A fila de entrada terá o seu tamanho atual "*" reduzido de uma unidade (-1). Para que

esse conectivo realize sua função, a fila de entrada não deve ser vazia, mas quaisquer de seus ramos de saída, a qualquer, momento pode apresentar uma fila vazia ou de qualquer tamanho.

4.3.7 Definição semântica do conectivo BUNDLE

BUNDLE é um conectivo de composição de dados. É uma construção estabelecida através da representação de cabos e fios múltiplos a partir de setas. A representação gráfica de BUNDLE é a mesma de JOIN, mas a rotulação do conjunto de setas é diferente. A função realizada por esse conectivo em um diagrama é atuar estruturalmente sobre os diversos tipos de dados presentes em seus múltiplos fluxos de entrada, e, a partir daí, formar na saída, uma super-estrutura que é composta dos diversos tipos de dados das entradas. O que o conectivo faz é uma composição de informação uma espécie de "empacotamento".

```
operador conectivo BUNDLE
especificacao
     entrada  A, B, C, D :  fila de sub-tokens
     saida    E          :  fila de tokens
     descricao
          se A > 0  e  B > 0  e
             C > 0  e  D > 0
             entao
               E <-- * + 1
               A <-- * - 1
               B <-- * - 1
               C <-- * - 1
               D <-- * - 1
             senao null
fim operador conectivo BUNDLE
```

4.3.8 Descrição do significado de BUNDLE

O conectivo BUNDLE tem sua função de elaboração semântica como segue: em cada fluxo de entrada com seu distinto nome, indicando um particular subtipo de dado, dispõe-se de uma fila de "sub-tokens" objetos aos quais não pretendemos dar representação nesse contexto, importando-nos somente a denominação do tipo ou subtipo de dado. A denominação de "sub-tokens" é dada devido à natureza composicional do conectivo, às entradas que comporão a saída. A representação gráfica de BUNDLE é semelhante a JOIN, mas o significado é outro. A interpretação desse significado é a seguinte: Primeiro é verificado se em todos os fluxos de dados de entrada existem filas não vazias de "sub-tokens". Caso isso seja verdade, a fila de saída terá seu valor atual "*" acrescido de um "token", e os diversos fluxos de entrada terão suas filas de "sub-tokens" reduzidas de um elemento. Se pelo menos uma fila de entrada for vazia, a operação não será realizada.

4.3.9 Definição semântica conectivo ORBRANCH

O conectivo ORBRANCH é de decomposição de dados. O seu propósito é mostrar exclusão através do conceito de alternativas explícitas.

Em [ROS 83a] comenta-se que na maioria das boas diagramações esse conectivo é moderadamente usado. É válido afirmar isso porque, em muitas circunstâncias, o fato das setas representarem restrições em vez de dominância, pode-se necessitar entender um diagrama em forma clara através das conexões topológicas. Para [ROS 83a] essa é a razão pela qual não existe construto para AND e outras funções lógicas, porque essas outras funções não se enquadram para o nível de comunicação da linguagem básica do SADT.

```
operador conectivo ORBRANCH
especificacao
     entrada   A              : fila de tokens
     saida  escolha de
               B              : fila de tokens
               C              : fila de tokens
               D              : fila de tokens
                       :
                       :
               Y              : fila de tokens
               Z              : fila de tokens
            fim escolha
     descricao
          se A > 0
            entao
              escolha-rand(saida)
              escolhido = alfa
              atualiza-saida(escolhido)
              A <-- * - 1
            senao null
fim operador conectivo ORBRANCH
```

4.3.10 Descrição do significado de ORBRANCH

O conectivo ORBRANCH deve ter o número de fluxos de saída igual ou superior a dois (2). Isso é explicável porque a semântica do conectivo parte do princípio que ao verificar que a fila de "tokens" na entrada é não vazia, proceder-se-á uma escolha aleatória entre os diversos fluxos de saída, independentemente do existência ou não de filas de tokens nessas saídas. No esquema apresentado para OR- BRANCH, representa-se essa operação por *escolha-rand(saída)* que é uma função que recebe como parâmetros o conjunto de saída, faz a escolha randômica de um fluxo, retornando o nome do fluxo escolhido na da variável ALFA. A variável ESCOLHIDO é utilizada para servir de parâmetro para a função que atualiza a fila no fluxo de saída escolhido, tomando o tamanho atual "*" e adicionando um novo elemento. Essa operação foi descrita assim por motivos de clareza, mas ela poderia ser escrita

de forma mais compacta como:

atualiza-saida(escolha-rand(saida))

Realizado esse passo, a fila de "tokens" na entrada do conectivo terá seu valor atual representado por "*" reduzido de um elemento. Deve-se observar que a definição dessa classe de conectivo mostra realmente uma forma de exclusão lógica, usando para tal escolha randômica dentre os fluxos de saída.

4.3.11 Definição semântica do conectivo ORJOIN

O conectivo ORJOIN é da categoria de composição de dados, mas como veremos é uma forma de composição que obedece a uma semântica bem particular. O propósito desse conectivo [ROS 83a] é mostrar exclusão através do conceito de alternativas explícitas. A operação é feita sobre o conjunto de informações que aparecem nos diversos fluxos de entrada, resultando em um único fluxo de saída.

A definição semântica para o conectivo ORJOIN é a mais complexa. Relembrando [ROS 83a] recomenda que esse construto, só deve ser utilizado quendo trouxer clareza e concisão na diagramação de um determinado problema.

O estudo sucinto da semântica definida mostra que esse é o construto de mais complexa definição. Mas esse fato, pelo menos abaliza as recomendações de Ross, como também demonstra a riqueza de um método como SADT.

```
operador conectivo ORJOIN
especificacao
     entrada escolha de
              B             : fila de tokens
              C             : fila de tokens
              D             : fila de tokens
                     :
                     :
              Y             : fila de tokens
              Z             : fila de tokens
         fim escolha
     saida
              P             : fila de tokens
     descricao
     se todos-fluxos(entrada) = vazio
       entao null
       senao
          se todos-fluxos(entrada) <> vazio
            entao
              retira-fila(esc-rand1(entrada))
              poe-na-fila(saida)
            senao se alguns-fluxos(entrada) <> vazio
                    entao
                     retira-fila(esc-rand2(entrada))
                     poe-na-fila(saida)
                    senao se um-fluxo(entrada) <> vazio
                            entao
                             retira-fila-unica(entrada)
                             poe-na-fila(saida)
                            senao null
                          fim se{ um-fluxo <> vazio }
                  fim se{ alguns-fluxos <> vazio }
          fim se{ todos-fluxos <> vazio }
     fim se{ todos-fluxos = vazio }
fim operador conectivo ORJOIN
```

4.3.12 Descrição do significado de ORJOIN

A elaboração da semântica do conectivo OR JOIN foi a mais complexa entre todos os construtos até aqui vistos. A informação de entrada, no esquema é feita por uma escolha, o que significa dizer que no conjunto de entradas pode haver fila de "tokens" em todos os fluxos, em alguns fluxos, em somente um fluxo ou em nenhum fluxo. A função semântica então é a seguinte: primeiro é verificado se todos os fluxos de entrada tem fila vazia, neste caso não será realizada nenhuma ação, ou seja não haverá disparo do conectivo. O próximo passo é verificar se em todos os fluxos de entrada existem filas de "tokens" não vazias, se for verdade, a função RAND1 aplicada ao conjunto de entradas, faz a escolha aleatória de uma delas, retira um "token" e a fila de saída terá seu valor atual "*" acrescido de um elemento. Caso não seja verdade a presença de "tokens" em todas as entradas, parte-se para verificar se existem "tokens" em um número de entradas superior a duas, se for verdade, então utiliza-se a função RAND2, que aplicada sobre o conjunto de entradas faz esse tipo de verificação, realizando um sorteio randômico entre os fluxos com filas diferentes de vazio, aplicando então sobre o fluxo sorteado a função RETIRA-FILA(entrada) que faz a atualização devida na entrada, após o que a função POE-NA-FILA(saída) atualiza a fila de "tokens" de saída. Caso só exista um único fluxo de entrada que apresente fila de "tokens" diferente de vazio, usa-se a função RETIRA-FILA-ÚNICA(entrada), sendo o resto do procedimento semelhante ao que já foi explicado anteriormente. Deve-se observar que a fila de "tokens" de saída nesse conectivo é de comprimento qualquer, no decorrer do tempo.

4.4 A definição semântica de atividade

A linguagem gráfica do SADT parte do princípio de que sistemas podem ser compreendidos e descritos relacionando dados e atividades. Para [ROS 83a], essa é a forma como entende-se situações reais e problemas, sendo a maneira pela qual as linguagens naturais podem representar objetos utilizando-se substantivos e verbos para representar atividades.

Uma atividade é uma unidade de tarefa em um processo de software [KAT 89]. Ela é completamente caracterizada por suas entradas, controles e saídas,

e essas interfaces,denotam por exemplo, especificação de requisitos, documentos de projeto, relatórios de análise, código de programa ou qualquer outro produto de software relacionado à atividade e armazenado em uma base de objetos.

Em um sistema de processamento da informação, uma atividade representa uma função que transforma as entradas (no contexto de uma atividade SADT entradas são os fluxos de entrada propriamente ditos mais os fluxos de controle) em saídas. A seta de mecanismo na atividade, como visto no capítulo 2, representa efetivamente o agente que leva a cabo a função. Neste trabalho, como na maioria das publicações sobre SADT, não se utiliza a seta de mecanismo, esta pode de certa forma ser considerado mais como um detalhe notacional orientado para implementação.

A representação de uma atividade SADT por uma caixa tem o propósito de limitar um contexto onde se representa uma função . Esse limite estabelece o que faz parte do contexto e o que não faz [ROS 83a]. Uma atividade que não possui refinamento, é denominada primitiva, o caso contrário, denomina-se de complexa, é a atividade que possui sub-atividades em outro diagrama.

Pode-se sintetizar e dizer que uma atividade é uma transformação de um assunto (objeto ou dado) de um estado antes para um estado depois. Nesse sentido um diagrama de atividades SADT é um modelo de computação que transforma entradas em saídas. Se for visualizado um diagrama pai, dentro desse contexto transformacional, sem a preocupação com a caracterização dos vários estados internamente produzidos, para se chegar a um estado final no diagrama pai, isso tem um caráter de semântica denotacional, onde o princípio básico é construir especificações como modelos explícitos de um sistema. No SADT isso é expresso por uma atividade, mapeando diretamente o seu significado (semântica), a partir de suas entradas controles para as respectivas saídas, além evidentemente, das funções que manipulam os dados. Pode-se dizer isso, porque uma definição denotacional não especifica passos de uma computação, como ocorre em semântica operacional.

A linguagem gráfica do SADT traz em sua essência, uma forte dose de formalidade, sendo no entanto de uso simples e fácil ao entendimento de um leigo. Como visto na secção anterior, cada conectivo tem uma semântica própria, o que se descreveu foi como operacionalmente eles atuam. Deve-se lembrar neste ponto,

que as interfaces em um diagrama, representada pelo conjunto de conectivos, é um recurso que nem um outro método orientado a fluxo de dados [GAN 79], [DeM 78] e [REI 82] apresenta.

A atividade é a construção básica oferecida pelo SADT, que juntamento com os conectivos, forma o conjunto básico de elementos que dispõe-se para descrever um sistema, e tal especificação, deve modelar precisamente a realidade, mostrando as mudanças que podem ocorrer sobre o modelo, ou seja, as diversas transições entre os estados do sistema.

4.4.1 Estados de uma atividade

O conjunto de atividades em um diagrama SADT expressa as funções que um sistema realiza. Para definir a semântica do construto atividade, considerando o objetivo de execução desta por simulação, parte-se do princípio que: uma atividade qualquer em determinado instante do tempo poderá estar num dos *estados* seguintes:

1) Produzindo saída

2) Pronta para disparar

3) Em processamento

4.4.2 Atividade produzindo saída

Na proposição de simulação para uma atividade primitiva, introduziu-se o conceito de tempo. Isso significa que para a realização de cada atividade, será "sorteado" um tempo (em uma distribuição de probabilidades que pode ser Normal ou Poisson), dentro de um intervalo mínimo e máximo pré-estabelecido pelo especificador do sistema.

Uma atividade produzindo saída é aquela que foi disparada, e seu tempo de processamento está concluído. Então pode-se descrever operacionalmente isso como:

```
{ Atividade produzindo saida }
 se  tempo = 0
     entao se Entrada > 0 e Controle > 0
               entao Saida <-- * + 1
                     Inativo  <-- VERDADE
               senao null
     senao null
```

4.4.3 Atividade em processamento

Existe um tempo de simulação do modelo como um todo e um tempo sorteado para a realização de uma atividade primitiva. Para uma atividade produzir uma saída, ela deve primeiro estar em processamento ou seja, "consumindo" seu tempo sorteado para execução. Por exemplo, para uma atividade A foi sorteado 10 unidades de tempo para sua execução, isso significa que do momento do disparo de A até que essa atividade produza saída, à variável tempo será asinalado os seguintes valores:

tempo = 10 { atividade "A" disparou }

tempo = 9 { atividade "A" em processamento }

tempo = 8 { atividade "A" em processamento }

tempo = 7 { atividade "A" em processamento }

: : :

: : :

tempo = 0 { atividade "A" produz saída }

Uma atividade em processamento tem então o seguinte esquema:

```
{ Atividade em Processamento }
 se tempo > 0
     entao tempo <-- * - 1
     senao null
```

Para o controle da simulação significa que ao passar por uma atividade onde a variável Inativo = FALSO é verificado o seu tempo consumido. Caso este tempo não tenha sido esgotado, o contador de tempo da atividade terá seu valor atual "*", decrementado de uma unidade. Quando o tempo dessa atividade chegar a zero ela produzirá sua saída.

4.4.4 Atividade pronta para disparar

Uma atividade está pronta para disparar, quando as filas de "tokens" nos seus fluxos de entrada e controle apresentam tamanho diferente de vazio e está inativa. O esquema para disparo é o seguinte:

```
{ Atividade pronta para disparar }
 se Controle > 0 e
    Entrada  > 0 e
    Inativo  = VERDADE
      entao { Sorteia tempo para atividade }
         tempo-atv <-- randomico(minimo,maximo)
         { Consumir ''tokens'' de entrada e controle }
         Controle <-- * - 1
         Entrada  <-- * - 1
      senao null
```

Pelo exposto, uma atividade em simulação no modelo de sistema especificado em SADT, em determinado instante do tempo de simulação, poderá encontrar-se produzindo saída.

Considerando que na fila de saída, neste exato instante do tempo de simulação ainda exista vaga para comportar mais um "token", que será produzido pela atividade A, e esta preenche todas as condições para tal, ou seja existem "tokens" em todas as entradas e controles, e a sua variável de estado Inativo, tem como valor booleano FALSO.

Atividade em processamento é aquela que apresenta a variável Inativo com valor FALSO e ainda não consumiu todo o seu tempo sorteado para processamento. A figura 4.1 ilustra um conjunto de exemplos interessantes para atividades no contexto de uma simulação simbólica:

Quaisquer das atividades da figura 4.1 desde que o tempo sorteado para processamento, em determinado instante da simulação apresente valor superior a zero, elas se encontram em pleno processamento. A atividade 'A" encontra-se em processamento, mas quando seu tempo terminar, ela não produzirá saída; a condição para tal é que as filas de "tokens" na(s) entradas(s) e controle(s) sejam de tamanho superiores a zero. Essa situação pode ser diferente, se em vez de isolada, essa atividade for parte de um diagrama, e sua(s) entrada(s) e controle(s) são saídas de uma outra atividade, a qual produziu "tokens" para essas interfaces, enquanto "A" encontrava-se em processamento. Para a atividade "B" vale o mesmo que se disse para "A" somente deve ser observado que para "B" produzir saída, só depende que seu fluxo de controle receba um "token" da saída de alguma atividade, ou que tal "token" seja fornecido por uma operação de inicialização se for o caso. Para as atividades "C" e "D" a situação é semelhante ao caso da atividade "B". Finalmente para a atividade "E", em um determinado momento do tempo de simulação, se o seu tempo de processamento (que lhe resta) for maior do que zero, como as demais ela se encontra em processamento, quando seu tempo for totalmente consumido, ela produzirá um "token" de saída e o valor da variável Inativo torna-se VERDADEIRO, mas como ela tem condições de disparar novamente, isso acontecerá, sendo sorteado um novo tempo para processamento, dentro de seu intervalo mínimo e máximo, serão consumidos "tokens" de entrada(s) e controle(s) passando a variável Inativo para o valor FALSO.

UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA

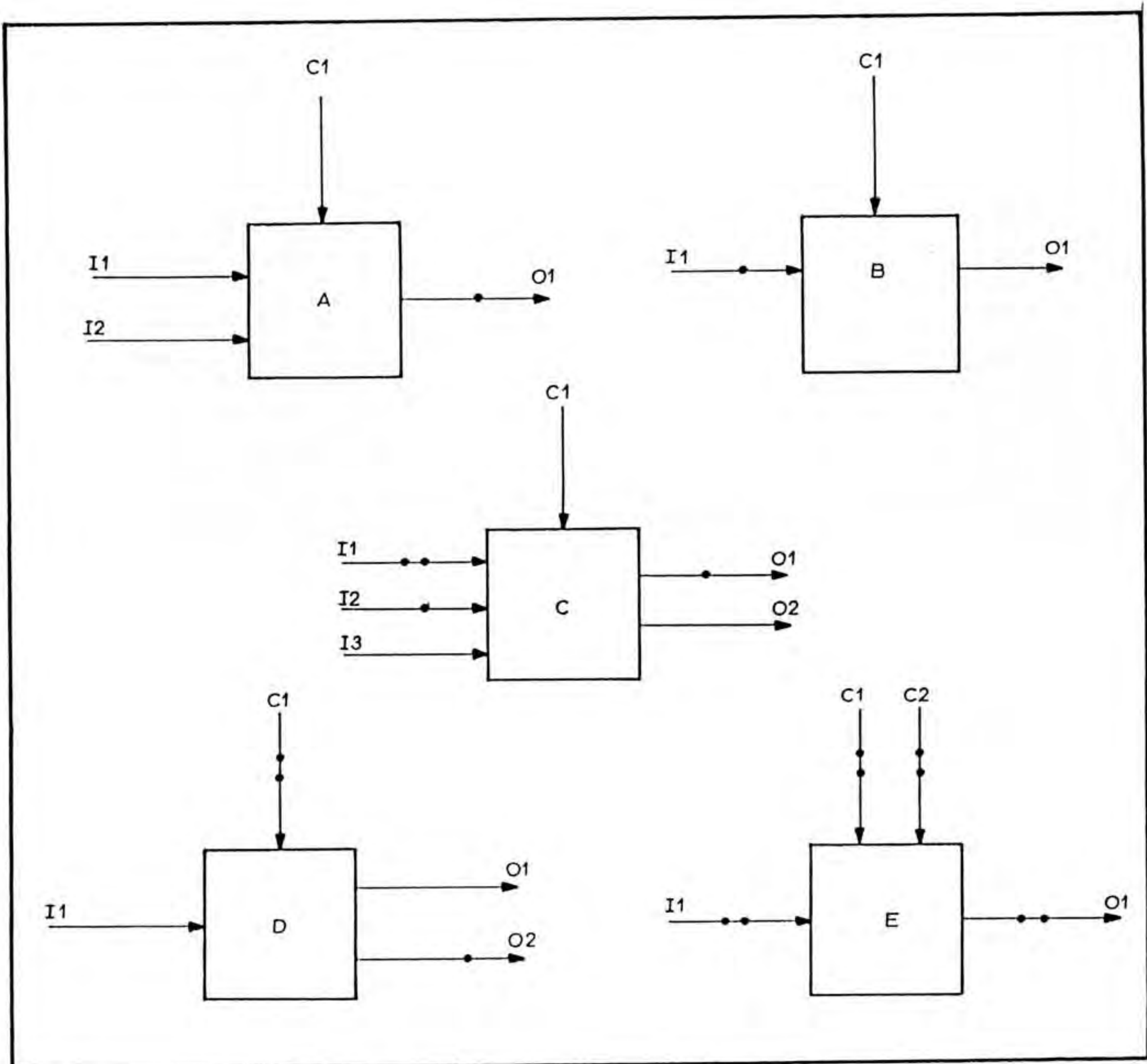


Fig. 4.1 Simulação de atividades SADT

4.5 Diagrama utilizando construtos do método SADT

O exemplo modelado inicia mostrando a entrada de um *módulo virgem* de software V, que aparece como uma das entradas do conectivo C1 (ORJOIN). A semântica definida para esse construto, determina que se pelo menos um fluxo de entrada conter "token", ele será consumido e gerado um "token" no fluxo de saída. Dessa maneira a entrada do conectivo C2 (BRANCH) recebe um "token" que representa o *módulo virgem* de software V. O módulo V, de acordo com a semântica definida para o conectivo BRANCH, será replicado com o nome *módulo A* como entrada para às atividades TESTAR SINTAXE e CORRIGIR ERROS DE SINTAXE.

A atividade TESTAR SINTAXE é disparada, considerando a existência de "tokens" X de controle que representam no modelo as regras de sintaxe estabelecidas para a verificação. Essa atividade produzirá uma *lista de erros.*

A atividade CORRIGIR ERROS DE SINTAXE ficou aguardando o término da atividade TESTAR SINTAXE para ser disparada, embora durante todo o tempo com uma cópia do *módulo virgem* inicial à sua entrada. O disparo dessa atividade e consequente conclusão, produz *módulo sem erros de sintaxe.* Esse módulo é o "token" de entrada para o conectivo C3 (ORBRANCH). Esse conectivo opera de tal forma, que para a existência de "tokens" em sua entrada, será escolhida aleatoriamente quaisquer de suas saídas onde aparecerá o "token" produzido. No caso aqui apresentado as saídas de C3 conectam entradas para os conectivos C5 (BRANCH) e C4 (ORJOIN). Se a escolha for a entrada para o conectivo C4, neste momento *módulo corrigido* B ainda não existe porque a atividade CORRIGIR ADVERTÊNCIAS não aconteceu. Então a semântica de C4 determina que para uma única entrada com "token", este será consumido, sendo produzido o "token" de saída, que neste caso chama-se *módulo corrigido.* Este volta para C1 e repetir-se-á o ciclo de execução anteriormente descrito para as atividades TESTAR SINTAXE e CORRIGIR ERROS DE SINTAXE.

No caso de ter sido selecionada a saída do conectivo C3 que serve de entrada para o conectivo C5, este replicará a informação de entrada *módulo sem erros de sintaxe* no fluxo de entrada das atividades COMPILAR e CORRIGIR

ADVERTÊNCIAS.

A execução da atividade COMPILAR é governada por diretivas de compilação. Ao ser executada, produz *advertências* que controlam a atividade CORRIGIR ADVERTÊNCAS.

A atividade CORRIGIR ADVERTÊNCIAS, só pode acontecer após a compilação do módulo e, portanto, ficou parada enquanto COMPILAR era executada. A saída produzida por CORRIGIR ADVERTÊNCIAS é um módulo com as *advertências* corrigidas que serve de entrada para o conectivo C6 (ORBRANCH). Este ao ser executado em um fluxo coloca *código* e em outro modulo corrigido B. Este é entrada para C4 e, o processo como um todo se repete.

Neste estudo de caso, objetivando simplicidade de apresentação, não tecemos comentários acerca do tempo de processamento de cada atividade no diagrama. Essa apreciação de tempo para processamento de uma atividade é apresentada em detalhes no capítulo seguinte desta dissertação.

4.6 Comentários e observações

O que se apresentou nessa parte da dissertação foi uma definição semântica para simulação de conectivos e atividade SADT. É importante que se diga, que a proposição para simulação do método é absolutamente inédita, porque até hoje o SADT não recebeu nem uma proposição de extensão nesse sentido, e sempre foi tratado como método semi-formal. Esse tratamento, a literatura especializada também dispensa ao método de Análise Estruturada, como apresentado em [GAN 79] e [DeM 78].

O estabelecimento de semântica para conectivos e atividade do SADT, visualizando a natureza de rede do método, ajustada ao conceito de "dataflow machine" e naturalmente teoria dos grafos, constitui-se em uma grande contribuição ao método originalmente criado por Ross e seus colegas da Softech [ROS 83]. Essa definição semântica para as principais construções gráficas do método, tornam-o um método formal, com sintaxe(já definida em [ROS 83a]) e semântica precisamente definida. Isso permitirá uma aplicação mais rigorosa do método, na definição e

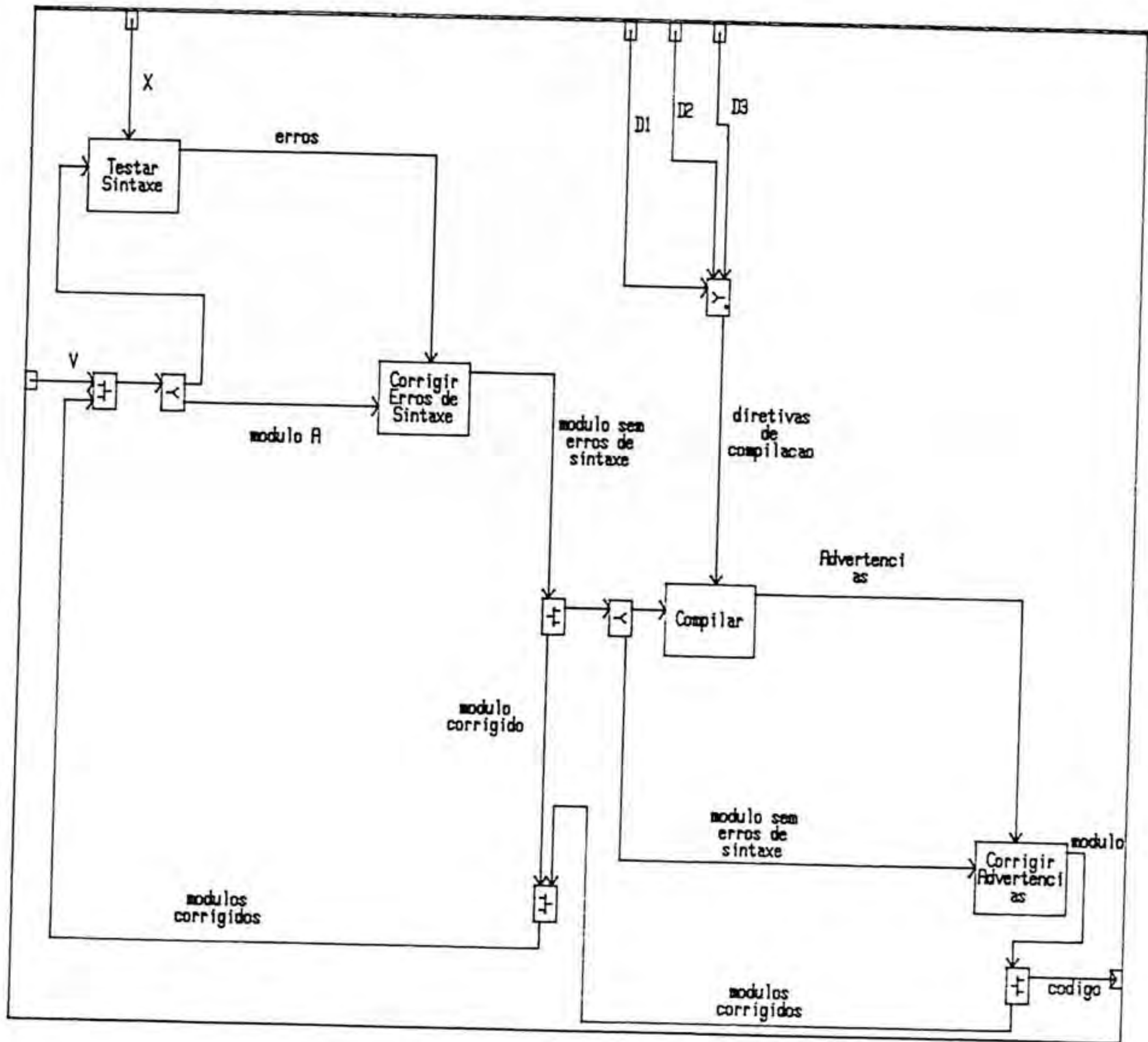


Fig. 4.2 Exemplo de utilização dos construtos do SADT

especificação de requisitos de sistemas em geral.

O conceito de tempo para a execução de uma atividade é também algo absolutamente novo no método. Isso pode permitir que se tire determinadas métricas do desempenho do sistema modelado, mas esse não é o objetivo aqui, embora possa ser estendido em futuros trabalhos.

A topologia de uma rede no contexto do SADT pode ser completamente determinada pelos conectivos, através dos quais se estabelece seqüencialidade ou paralelismo no modelo, e com a simulação de cada componente verificar concorrência, atividades que mais demoraram para produzir suas saídas, atividades que passaram mais tempo paradas em todo um ciclo de simulação, a parte do sistema que está produzindo "engarrafamento", a dependência funcional entre as atividades e outras características que possam contribuir para um maior entendimento do sistema, tanto por parte do projetista como por parte do usuário.

5 A SIMULAÇÃO DA EXECUÇÃO DE ESPECIFICAÇÕES SADT

5.1 Sistemas & Modelos

Um sistema na acepção mais geral do termo é uma agregação de objetos reunidos segundo uma interdependência ou interação regular [GOR 69]. Objetos distintos ou entidades possuem propriedades diferentes. Para [GOR 69] qualquer processo que causa alterações no sistema é uma atividade. Toda atividade é disparada pelo acontecimento de eventos ou fatos. O estado do sistema é a descrição de todos os seus objetos e atividades realizadas sobre estes num determinado instante do tempo. Os eventos representam alterações no estado do sistema. O tracejamento dos sucessivos estados do sistema permite verificar o comportamento do sistema.

O exemplo a seguir é um fictício sistema descrito num diagrama SADT, que contém três atividades "A","B" e "C". Observe que no instante t = 0; o sistema apresenta-se como inicializado, tendo sido disposto no fluxo de entrada I1 uma fila de 3 "tokens", no controle C1 uma fila de 2 "tokens" no controle C2 foi colocado somente 1 "token". Todas as atividades desse sistema são primitivas, tendo portanto cada uma, um intervalo de tempo de execução associado. Os conectivos não possuem tempo associado de execução, são de execução instantânea. Na realidade qualquer atividade leva tempo para ser executada, e o conectivo é uma microatividade, mas o seu tempo de execução, comparado ao de uma atividade pode ser considerado desprezível. No diagrama tem-se um conectivo "'SPREAD"

simbolizado pela letra grega α e um "BRANCH" simbolizado pela letra β. A tabela a seguir apresenta as informações de tempo para cada atividade do diagrama.

Atividade	Tempo-Mínimo	Tempo-Máximo	Tempo-Sorteado
A	2	4	3,2
B	1	3	1
C	2	6	2

Veja a seguinte sequência do mesmo diagrama, representando os vários estados do sistema.

A execução dessa especificação através de simulação se dá da seguinte forma:

No instante t = 0 o sistema foi inicializado com a colocação de "tokens" nos fluxos I1, C1 e C2. Todas as atividades estão paradas e nenhum conectivo foi acionado.

No instante t = 1 a atividade "A" dispara e consome um "token" do fluxo de entrada I1 e outro do fluxo de controle C1. O conectivo α é acionado e produz um "token" para o fluxo de controle da atividade "B" e outro "token" para o fluxo de controle da atividade "C". Exceto a atividade "A", as demais neste instante permanecem paradas. No instante t = 3 a atividade "A" continua seu processamento, e "B" e "C" aguardando sua conclusão, e enquanto isso não acontecer, o conectivo β não será acionado, permanecendo portanto, neste instante os "tokens" de controle produzidos pelo conectivo α, aguardando para serem consumidos.

No instante t = 4 a atividade "A" completa seu processamento e produz um "token" de saída. Ainda nesse instante as atividade "B" e "C" estão paradas, ou seja a variável INATIVO de ambas apresenta valor verdadeiro. Ao ser produzido o "token" de saída da atividade "A", a sua variável Inativo também se torna verdadeira.

No instante t = 5, será ativado o conectivo β (que é um BRANCH) que produzirá um "token" para a entrada da atividade "B" e outro para a atividade "C".

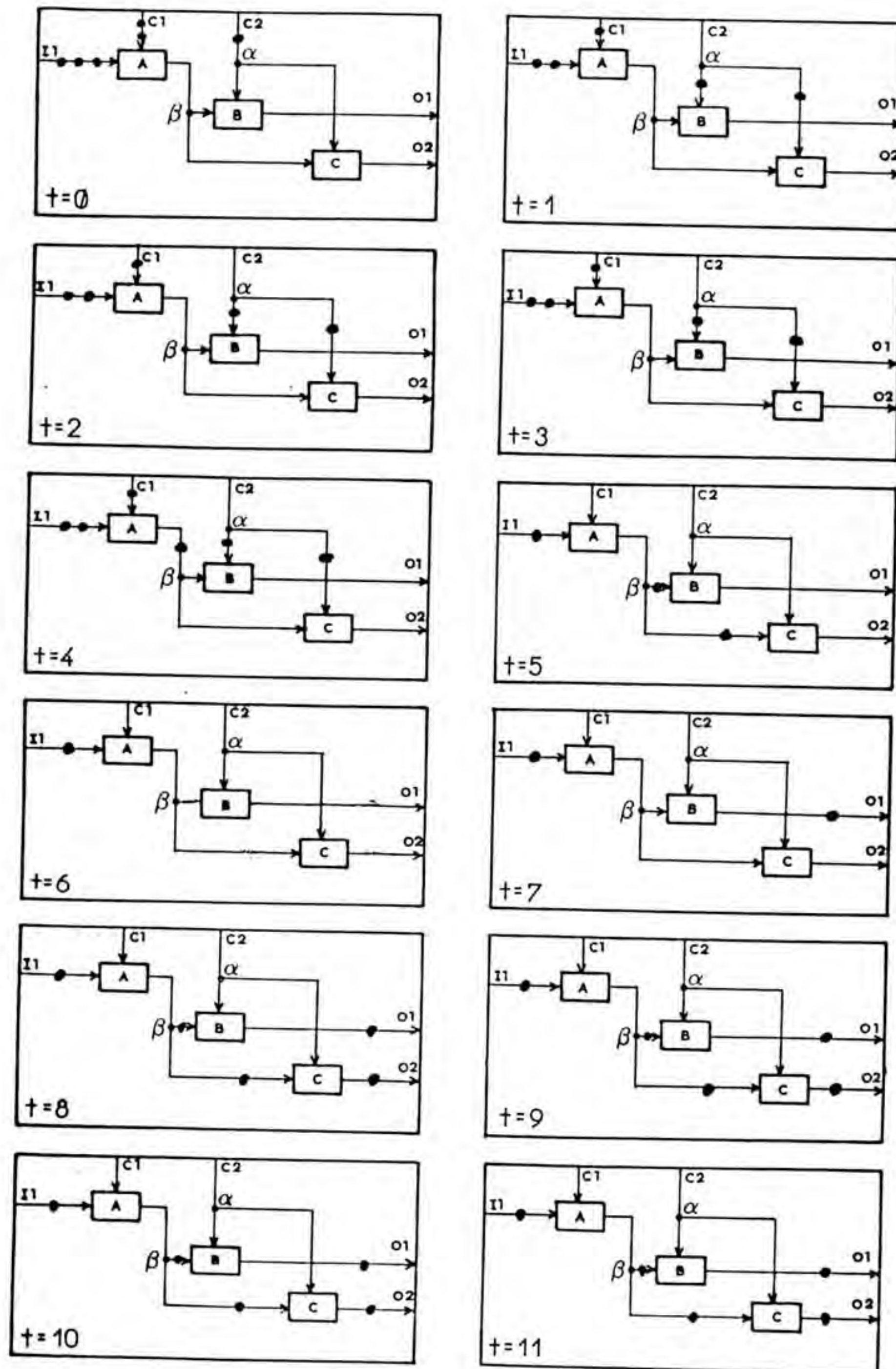


Fig. 5.1 Sistema em simulação

Nesse mesmo instante, verifica-se que a atividade "A" produziu sua saída, e voltou a disparar desta vez o seu tempo sorteado foi 2. As atividades "B" e "C" também foram disparadas neste instante com os tempos sorteados de 1 e 2 respectivamente.

No instante t = 6 completa-se o tempo de processamento da atividade "B", e é produzido um "token" na sua saída, e seu estado passa para inativa, e, na ausência de "tokens" nesse instante nos seus fluxos de entrada e controle ela não volta a disparar. As atividades "A" e "C" continuam o processamento de suas funções.

No instante t = 7, as atividades "A" e "C" que estavam realizando suas funções em paralelo, produzem seus "tokens" de saída, passando o seu estado para inativas. A atividade "B" continua parada. O conectivo β é acionado, colocando "tokens" nas entradas das atividades "B" e "C".

No instante t = 8 todas as atividades do diagrama apresentam estado inativo, e nenhuma encontra-se pronta para disparo. A primeira atividade ("A") tem um "token" restante em seu fluxo de entrada, seu fluxo de controle encontra-se vazio, ela foi executada duas vezes, no decorrer desta simulação. As interfaces de entrada e controle das atividades "B" e "C" apresentam uma situação semelhante a da atividade "A". E foi produzido um "token" em cada saída do sistema. Nesse estado o sistema permanecerá até que seja atingido o tempo final estipulado para a simulação, no caso exemplo t = 10.

O conectivo "SPREAD" foi acionado uma só vez, o "BRANCH" cuja entrada é a saída da atividade "A", duas vezes.

A simulação realizada nesse conjunto de diagramas, como dito em [GOR 69] foi feita tracejando os sucessivos estados do sistema, e assim pode-se verificar o progresso e o comportamento deste.

O estudo de um sistema real de uma forma que permita a sua compreensão em detalhes, e a possibilidade de realizar previsões acerca de seu comportamento em múltiplas situações, é feito a partir do estabelecimento de uma modelo. O modelo deve reunir de forma legível, com amigabilidade e com bastante lógica, todas as informações relevantes ao sistema objeto de estudo.

O objetivo de se construir um modelo do sistema em estudo é possibilitar que a simulação discreta deste consiga reproduzir o comportamento (a semântica) do sistema. Isso é conseguido quando define-se os estados do sistema e se constroi atividades que o movem de um estado para outro.

5.2 A construção de modelos para simulação

A produção do modelo de um sistema [GOR 69] para o estudo do seu comportamento ao longo do tempo compreende duas grandes tarefas. A primeira é estabelecer a estrutura do modelo. Nessa tarefa são determinados os limites do sistema e identificadas as atividades e os diversos objetos (dados) envolvidos. A segunda grande tarefa da construção de um modelo é o fornecimento dos objetos (dados) a serem manipulados pelas diversas atividades que constituem o sistema como um todo.

Afirma-se em [GOR 69] que não existem regras para a construção de um modelo, mas, princípios gerais que descrevem diferentes pontos de vista, para o julgamento da informação que deve nele ser incluída. Neste ponto, verifica-se que a idéia de ponto de vista do SADT [ROS 83a] também segue esse mesmo conceito. Ainda em [GOR 69], ele diz que a construção de um modelo deve ser modular, o sistema deve ser graficamente representado como um bloco-diagrama, e devem aparecer somente informações relevantes para o sistema, as quais devem ser apresentadas de uma maneira exata. Todas as observações de [GOR 69] enquadram-se no espírito de construção de um modelo de sistema em SADT, na forma que aqui está se tratando. Isso é possível porque na proposição de semântica apresentada para o método, cada construção gráfica tem significado preciso.

A base de tempo de simulação, no modelo que proposto, avança aos saltos, sendo incrementado periodicamente de um intervalo de tempo, e assim o modelo é dito ser de tempo discreto, essa é a definição adotada em [ZEI 76].

A literatura de simulação é muito vasta; são feitas muitas menções dos propósitos da modelagem e usos de modelos. Um dos propósitos mais relevantes é, sem dúvida, o uso do modelo para estudar o comportamento do sistema ao longo do tempo. [KIV 67] considera a simulação como o uso de uma representação numérica

para estudar o comportamento de um sistema. Para [SHA 75] a simulação é um conceito onde está incluído o processo de modelagem do sistema, em suas palavras ele diz o seguinte: "Simulação é o processo de desenvolver o modelo de um sistema real e conduzir experimentos com este modelo, tanto com o propósito de entender o comportamento do sistema, como o de avaliar várias estratégias para a operação do sistema". Nesse pensamento, qualquer modelo ou representação de um sistema é uma simulação, e esse pensamento reflete o amplo uso que é feito do termo.

5.3 A construção de modelos em SADT

A utilização do método SADT para a construção de um modelo do sistema é fundamental para a obtenção de resultados confiáveis ainda na fase de definição de requisitos. A resolução de problemas requer que primeiro estes sejam bem definidos. O método SADT de acordo com um levantamento de [LEI 87], que abrange mais de 120 referências na área de requisitos, aparece como o método mais adequado para definir requisitos de sistemas.

A construção de um modelo para simulação requer um preciso entendimento do sistema, o que é possível agora descrever com o SADT, porque suas construções gráficas, possuem semânticas precisamente definidas. A possibilidade de usar o conceito de propósito e ponto de vista, permite que sejam construídos, uma variedade de modelos do sistema.

A construção de modelos é uma atividade essencialmente complexa, e em alguns campos até considerada como uma arte. O SADT simplifica essa atividade, se for considerado que ele impõe uma disciplina de pensamento ao projetista [ROS 85], com base em seus conceitos de limitação da informação e refinamento semântico. Muitos outros métodos contemporâneos ao SADT não "forçam" essa disciplina.

O modelo de sistema em SADT possui o nível de abstração que o projetista desejar. Nesse modelo são expressos os elementos relevantes do sistema em estudo, com um propósito e um ponto de vista.

A definição de um sistema deve englobar somente os objetos e acontecimentos circunspectos a uma dada realidade em estudo, e essa é uma possibilidade presente para o usuário do método SADT.

O método SADT, enriquecido com a semântica aqui proposta, pode ter um sistema nele modelado, factível de execução via simulação, isso porque em sua essência, ele é uma estrutura matemática e lógica, e nessa ótica, é possível experimentar uma forma de "imitar" o comportamento do sistema. A simulação do sistema descrito usando a linguagem gráfica do SADT vai permitir que várias observações sejam realizadas para dar subsídios às várias conclusões sobre o sistema.

A especificação de um sistema no método SADT é a sua descrição via um modelo, que poderá ser simulado, e essa experimentação permite inferências sobre o sistema como [SOA 90]:

- Não necessitar implementar o sistema, para conhecer o seu comportamento. A simulação do modelo oferece essa informação.
- A possibilidade de fazer experimentos em sistemas de alto custo operacional, ou que oferecam riscos em sua operação real.

E o modelo construído para simulação poderá ser empregado para os objetivos de:

1. Como uma ferramenta explanatória na definição de um sistema.
2. Como uma ferramenta de análise para a detecção e eliminação de pontos críticos no sistema.
3. Como uma ferramenta para síntese e avaliação de soluções propostas.
4. Como uma ferramenta de planejamento para desenvolvimentos futuros.

Um dos propósitos desta dissertação é possibilitar que o sistema descrito em SADT forneça um modelo para simulação, ou seja descrever algo de forma

compreensível para processamento em computador. Para tal o projeto de uma classe de dados, que descreve-se no próximo capítulo, que especifica todas as informações necessárias para a construção de diagramas SADT, fornece elementos para representar dados para simulação.

O conceito principal da modelagem para simulação é a descrição de estado do sistema. O sistema modelado em SADT é representado por um conjunto de tipos abstratos de dados, presentes nos vários fluxos, na forma de "tokens" e um conjunto de atividades interfaceadas pelos diversos fluxos ligados ou não a conectivos. Como já foi dito, os tipos abstratos de dados presentes nos vários fluxos são filas de "tokens". Uma combinação de "tokens" com fluxos e atividades em determinado instante do tempo determina um estado do sistema. O movimento de "tokens" na rede simula a mudança do sistema de um estado para outro.

5.4 A introdução do conceito de tempo no SADT

A introdução do conceito de tempo nos diagramas SADT, nas atividades primitivas, como uma variável aleatória, torna-o uma rede de atividades estocásticas. Essa rede, que é o SADT, constitui assim um modelo formal de computação. A interpretação do modelo SADT nesse contexto refere-se ao comportamento fornecido pelo conjunto de atividades, as quais podem acontecer sequencial ou paralelamente. E essas ocorrências estão sujeitas a regras de precedência e/ou paralelismo, o que pode ser determinado também pelo conjunto de interfaces através dos conectivos.

A introdução do conceito de tempo no método SADT possibilita a derivação de métricas de desempenho por exemplo, a partir da especificação funcional de um determinado comportamento de sistema. Em [RAM 80] encontra-se um trabalho interessante sobre temporização em estruturas de redes. [MER 87] também faz um tratamento do problema em uma aplicação da área de redes de comunicação.

A atribuição do conceito de tempo ao SADT é feita considerando o tempo somente para atividades primitivas, uma atividade não primitiva, possue tempo em função de seus refinamentos e, a execução neste modelo é "bottom up"ou seja das folhas do diagrama para a raiz. Uma atividade no modelo entra em execução,

quando existirem "tokens" em todos os seus fluxos de entradas e de controles. Sendo satisfeita essa condição, a atividade dispara imediatamente, e seu processamento consumirá em um tempo sorteado em seu intervalo máximo e mínimo, findo o que ela produzirá suas saídas. No modelo de simulação, adota-se uma estrutura similar a proposta em [MER 87], e nesse modelo existe um retardo entre a habilitação de uma atividade e o seu disparo, sendo que este é instantâneo. O que se propõe em termos de temporização para o SADT fundamenta-se no estudo de [MER 87], sendo que os intervalos de tempo entre a habilitação de uma atividade e o seu disparo seguem uma determinada distribuição probabilística.

5.5 Simulação do SADT orientada à eventos

A simulação discreta do modelo de um sistema especificado em SADT é um recurso através do qual pode-se verificar o comportamento do sistema. Para esse tipo de simulação, o estado do sistema só pode mudar nos tempos de eventos. Uma vez que o estado do sistema permanece constante entre tempos de eventos, uma descrição completa do estado do sistema pode ser obtida avançando o tempo simulado de um evento para outro. Esse é o mecanismo usado em muitas linguagens de simulação discreta.

A formulação de um modelo SADT para a simulação discreta começa com a construção do modelo do sistema, a definição da mudança nos estados, que podem ocorrer em cada tempo de evento, a descrição das atividades nas quais os objetos do sistemas (dados) se envolvem e a descrição do processo através do qual as entidades do processo fluem.

No contexto de simulação discreta [PRI 86] apresenta uma interessante figura relacionando os conceitos de atividade, evento e processo.

Assim pode-se observar que eventos acontecem em pontos isolados do tempo, no qual decisões devem ser tomadas de forma a iniciar ou terminar uma atividade.

Na simulação orientada a evento, o sistema é modelado pela definição

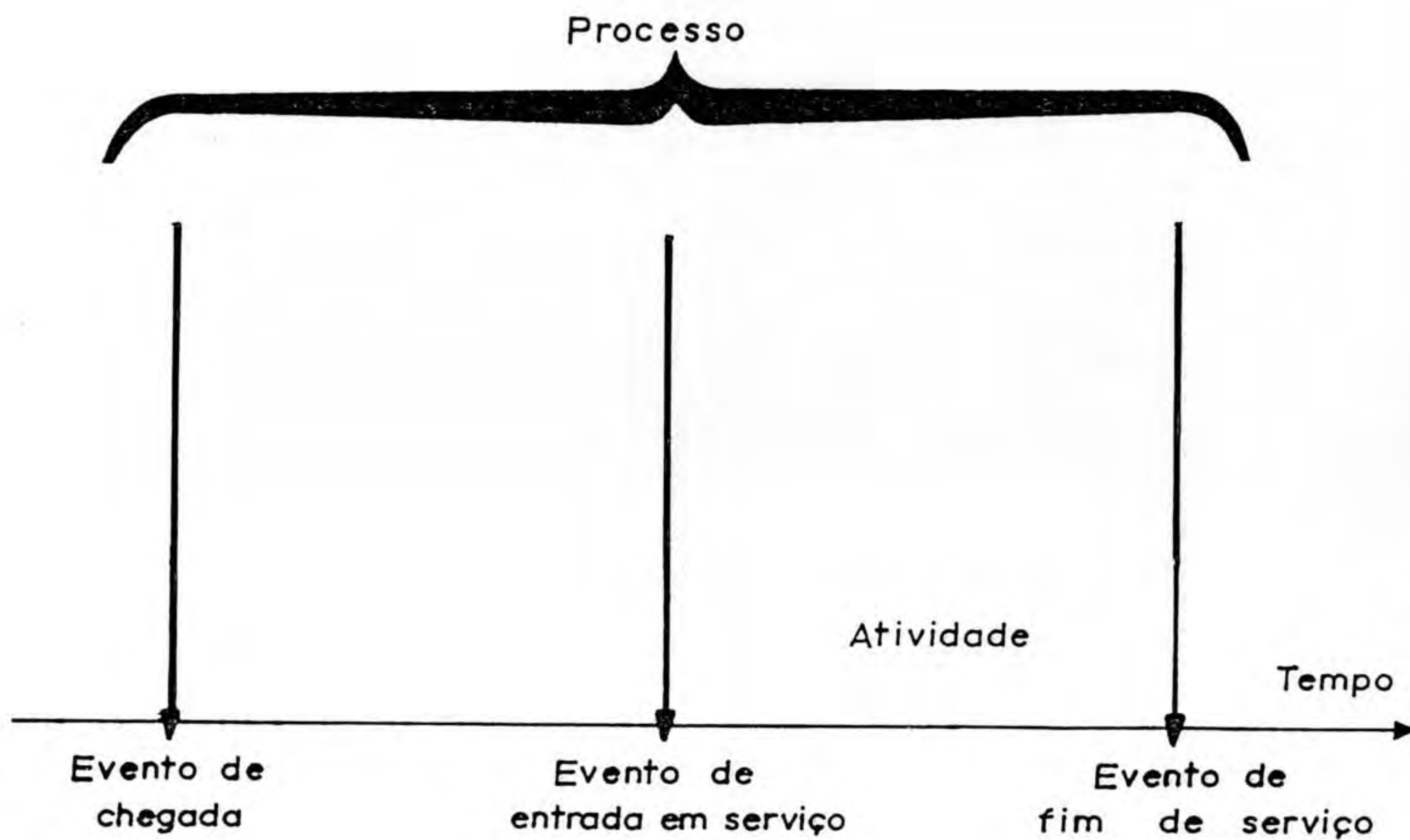


Fig. 5.2 Relacionamento atividade x evento x processo[PRI 86]

das mudancas que ocorrem no tempo de evento. O modelador determina os eventos que podem causar a mudanca no estado do sistema e então desenvolve a lógica associada com cada tipo de evento. A simulação do sistema é produzida pela execução da lógica associada a cada evento, em uma sequência ordenada do tempo. Em [KRA 88] apresenta-se um estudo para a animação de uma especificação de requisitos descrita em CORE [MUL 79], exatamente dessa maneira.

No modelo SADT ao ser simulado, deve-se considerar a simulação a nível de sistema, representado pelo diagrama principal, nível de subsistema representado pelos diagramas secundários e a nível de atividades, fluxo e conectivos. Para uma atividade primitiva sorteia-se um tempo dentro de seu intervalo mínimo e máximo, caso a atividade seja refinada volta-se ao nível de um novo sub-sistema. O diagrama à seguir em notação Warnier [WAR 73] ilustra isso de forma bem clara.

Dessa forma, a possibilidade de tornar uma especificação SADT executável através de simulação é perfeitamente possível. A classe para a construção da ferramenta gráfica, apresentada no próximo capítulo, já prevê informação para essa finalidade.

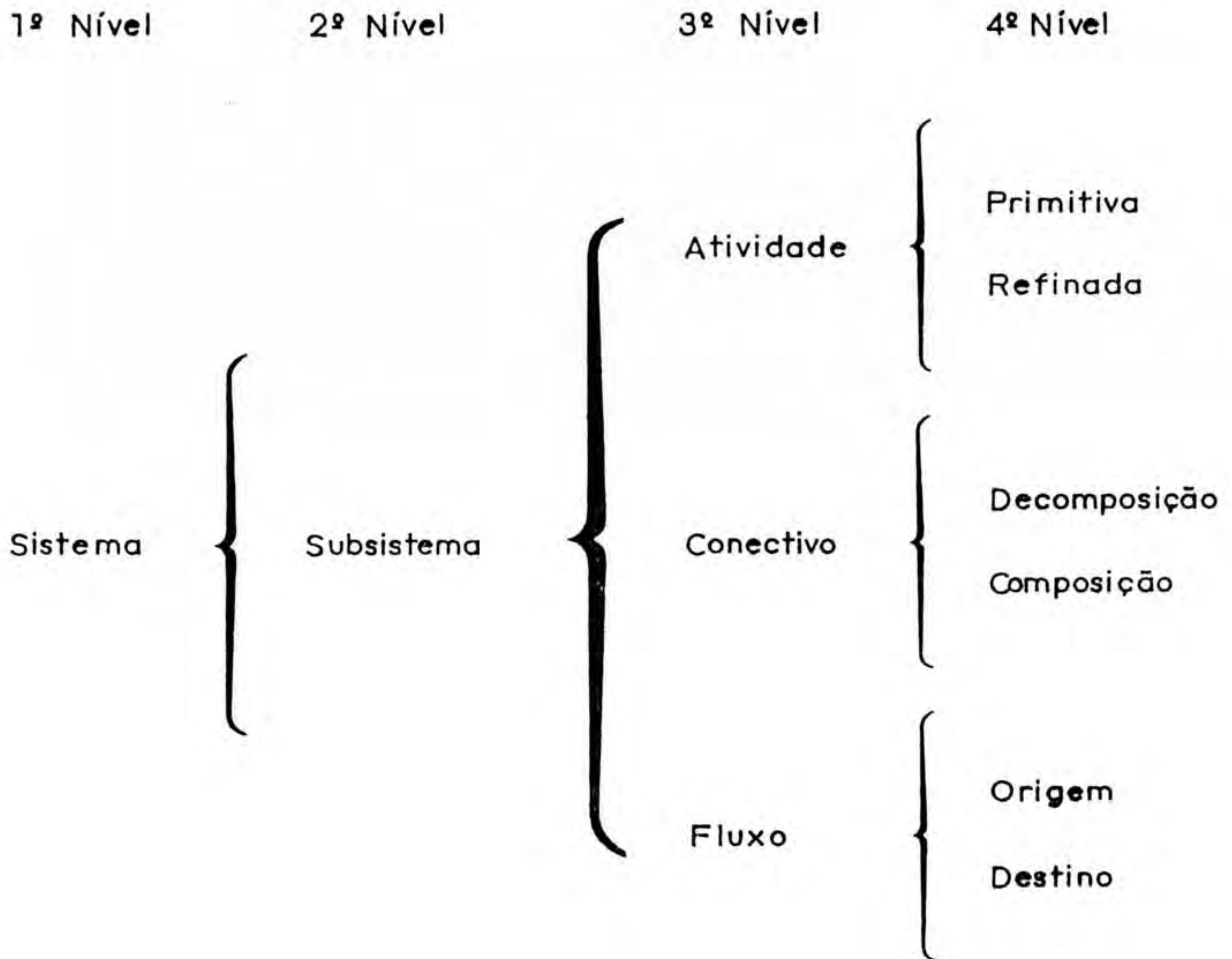


Fig. 5.3 Níveis de simulação no SADT

6 A CLASSE SADT PARA A CONSTRUÇÃO DA FERRAMENTA GRÁFICA

Este capítulo apresenta a classe de dados, projetada para a construção de uma ferramenta gráfica que implementa um sub-conjunto da linguagem SA do método SADT. É importante que se diga que, no ambiente PROSOFT versão atual, a definição de uma classe que descreve a linguagem de um método de engenharia de software a ser implementado é o passo inicial e de importância fundamental. A classe deve prever todos os objetos possíveis de serem instanciados no ambiente, além de que, se tal classe não for suficientemente completa, não se chegará a nenhuma ferramenta. A descrição formal da classe definida para o SADT é apresentada em VDM. As operações sobre o objeto instanciado nessa classe, são apresentadas neste capítulo, em modo informal. Uma sessão de uso da ferramenta, é mostrada através de algumas telas.

Apresenta-se também um panorama geral do Ambiente PROSOFT e alguns aspectos relativos ao método denotacional para desenvolvimento de software.

6.1 O Paradigma do Ambiente PROSOFT

Antes de apresentar a classe que descreve a linguagem gráfica do SADT, brevemente apresenta-se o PROSOFT, que é o ambiente de desenvolvimento de software utilizado para a implementação da ferramenta. Isso se faz necessário porque o

paradigma é o mesmo, tanto do ambiente, quanto das ferramentas nele construídas ou a ele integradas.

O Ambiente PROSOFT encontra-se presentemente em desenvolvimento no Instituto de Informática da UFRGS, tendo como pesquisador líder o Prof. Daltro José Nunes. Esse projeto tem por objetivo um ambiente de desenvolvimento que auxilie o engenheiro de software na construção de ferramentas a partir de ferramentas já existentes no PROSOFT, bem como oferecer aos usuários as ferramentas construídas para a solução de seus problemas [NUN 89].

O PROSOFT é um ambiente para construção de ferramentas, estendível e baseado na abordagem orientada a modelos. Uma ferramenta de engenharia de software a ser construída no PROSOFT é denotada por uma classe, especificada através de uma sintaxe abstrata e representada graficamente. Para cada classe no PROSOFT é definido um conjunto de operações primitivas, com as quais se pode gerar/alterar/consultar instâncias da classe. O PROSOFT também permite que uma classe faça referencias a outras classes pertencentes ao ambiente, possibilitando assim, a construção de uma ferramenta em função de outras ferramentas, obtendo-se dessa forma a reusabilidade de software. O PROSOFT é um ambiente especificado com rigorosismo matemático, e dentro dessa ótica cada uma de suas ferramentas pode ser definida como uma álgebra, representada pelo conjunto de objetos gerados pela classe e pelo conjunto de operações válidas para a classe.

Pelo fato de o ambiente de desenvolvimento utilizado nesta dissertação ser orientado a modelos, isso significa dizer que para se construir uma ferramenta utilizando o PROSOFT, o primeiro passo a ser dado é construir um modelo que represente o sistema. Esse modelo é denotado pela classe que descreve o comportamento (semântica) do sistema.

6.2 Abordagem denotacional para desenvolvimento de software

A abordagem denotacional para o desenvolvimento de software [BJO 78] define semânticas pelo mapeamento dos objetos a serem definidos, para alguns objetos

matemáticos conhecidos, como conjuntos, tuplas,produtos cartesiano, união de classes, mapemanetos e tipos primitivos. Em [BJO 78],[BJO 81],[BJO 82] e [BJO 88] encontra-se um estudo bastante amplo desses objetos.

No Método do Desenvolvimento de Viena - (VDM), os objetos alvo dos mapeamentos são referenciados como "denotações", sendo assumido que seus significados já sejam conhecidos. O VDM é um método para especificação formal de software orientado a modelos. O modelo do sistema em VDM é representado por uma classe, que constitui o domínio semântico do problema. Assim sendo, a primeira tarefa do especificador, usuário do VDM, é estabelecer a classe dos objetos, para os quais serão definidas semânticas.

A ferramenta implementada como parte desta dissertação é um subconjunto da linguagem SA do SADT. O núcleo gráfico dessa linguagem apresenta 40 construções [ROS 83a]. Vamos utilizar o VDM para especificar a classe do SADT, porque as abstrações dos objetos matemáticos utilizadas em VDM são as mesmas implementadas no ambiente PROSOFT [NUN 90].

A classe para o SADT será especificada através de uma sintaxe abstrata. Essa sintaxe permitirá oferecer uma classe de objetos, mais claramente estruturados, e as operações da ferramenta serão construídas sobre a estrutura sintática, o que é uma regra do método denotacional, onde a semântica de um construto composto é derivada da semântica de seus componentes. Pode-se então dizer que a semântica das sintaxes abstratas do VDM constitui uma classe de dados, ou domínio semântico, o que representa o modelo do sistema que se pretende especificar.

6.3 A classe SADT

A classe especificada aqui constitui o modelo do sistema que pretende-se implementar. Esta classe tem natureza informacional bastante gráfica, porque a ferramenta o exige. Isto é, foi implementado um subconjunto do núcleo gráfico do SADT [ROS 83a]. O método formal VDM mostrou-se bastante útil na tarefa de especificar a classe do SADT, devido aos poderosos tipos abstratos de dados oferecidos pelo método. A abstração aqui apresentada traz algumas ligeiras modificações, quando

se trata de tipos primitivos. O VDM não possui um tipo como coordenada ou texto, mas esses tipos foram incluídos na especificação, porque são úteis para descrever a real natureza do SADT, tanto no aspecto gráfico da ferramenta quanto para a necessidade documentacional de alguns construtos do método.

6.3.1 SADT

$$
\begin{array}{lrcl}
(1) & SADT & :: & TITULO \times INTERFACE \times \\
 & & & DESCRICAO \times DIAGRAMA \\
(2) & TITULO & = & String \\
(3) & INTERFACE & :: & TIPO* \\
(4) & DESCRICAO & = & Texto \\
(5) & DIAGRAMA & = & COMBINACAO* \\
(6) & TIPO & = & ENTRADAS \bigcup CONTROLES \bigcup SAIDAS \\
(7) & ENTRADAS & = & ENTRADA* \\
(8) & CONTROLES & = & CONTROLE* \\
(9) & SAIDAS & = & SAIDA* \\
(10) & ENTRADA & = & ROTA \\
(11) & CONTROLE & = & ROTA \\
(12) & SAIDA & = & ROTA
\end{array}
$$

Tipos abstratos de dados do método VDM [MAR 88] são apresentados em notação BNF, juntamente com as equações do domínio semântico.

A definição de um tipo abstrato de dado do VDM pode ser descrita por uma das seguintes equações:

$$\text{s-type} ::= \text{type-nome} = \text{type}$$

s-type ::= type-nome :: type

Dessa maneira a definição de tipos no VDM possui duas formas de equações ou regras, representadas pelos símbolos "=" e "::". Portanto, as equações do domínio semântico tem a seguinte sintaxe abstrata:

NOME = type

NOME :: type

O significado do símbolo "=" é o da igualdade da matemática, que na definição de um tipo do VDM diz respeito a substituição no identificador a esquerda, da ocorrência do lado direito.

O símbolo "::" é utilizado quando deseja-se ter objetos nomeáveis e decomponíveis em suas partes. Isto é, objetos compostos descritos na forma de produto cartesiano.

Anotações

A definição de qualquer problema usando a notação diagramática do SADT começa com o diagrama A-0 que contém a caixa pai de todos os demais diagramas. Esse primeiro diagrama possui características próprias: é composto por uma única caixa, não possui conectivos, e todas suas interfaces originam-se ou dirigem-se para o ambiente externo ao sistema sendo definido. É uma espécie de diagrama de contexto. Devido a essas peculiaridades, esse diagrama é descrito separadamente.

A descrição desse primeiro nível no método VDM é feita através de um objeto composto ou produto cartesiano.

Nessa descrição tudo o que se encontra do lado direito do sinal de igual, escrito em letras minúsculas em negrito, são tipos considerados primitivos. *TIPO* e *COMBINAÇÃO* são tuplas a serem especificadas em passos subsequentes, bem como *ROTA*.

Um elemento da classe SADT é uma árvore que tem como folhas:

- Um conjunto unitário composto pelo *TITULO* do diagrama.
- Uma lista de *INTERFACE*, representada pela tupla *TIPO*.
- Uma *DESCRICAO* formada por um texto que descreve (documenta) o diagrama.
- Lista de *DIAGRAMA* representada pela tupla *COMBINACAO*.
- O *TIPO* de interfaces são *ENTRADAS* ou *CONTROLES* ou *SAIDAS*, que são tuplas de *ENTRADA*, *CONTROLE* ou *SAIDA*.
- a *ENTRADA*, *CONTROLE* ou *SAIDA* são do tipo *ROTA* no diagrama.

6.3.2 Rota

$$
\begin{array}{lrcl}
(13) & ROTA & :: & TRACADO \times SEMANTICA \times NOME \\
(14) & TRACADO & = & SEGMENTO* \\
(15) & SEGMENTO & = & Coord \\
(16) & SEMANTICA & = & Nat0 \\
(17) & NOME & = & String
\end{array}
$$

Anotações

ROTA é uma classe auxiliar, onde estão especificadas as informações necessárias para se definir um fluxo.

Na especificação do diagrama SADT, cada fluxo de entrada, controle ou saída em uma atividade, bem como qualquer fluxo simples no diagrama, é definido como uma *ROTA*.

- *ROTA* é um produto cartesiano formado por *TRACADO*, *SEMANTICA* e *NOME*.
- *TRACADO* é definido como uma tupla de segmento.
- *SEMANTICA* é definida por um número natural (que pode ser zero) e, representa a fila de "tokens" para a simulação do tipo de dado presente no fluxo.
- *NOME* é definido como um "string".

6.3.3 Diagrama

(18)	*DIAGRAMA*	=	*COMBINACAO*∗
(19)	*COMBINACAO*	=	*COMPONENTE* × *LIGACAO*
(20)	*COMPONENTE*	=	*ATIVIDADE* ∪ *CONECTIVO*

Anotações

- *DIAGRAMA* é formado por uma tupla *COMBINACAO*.

- *COMBINACAO* é o produto cartesinao de *COMPONENTE* e *LIGACAO*.
- *COMPONENTE* é uma *ATIVIDADE* ou um *CONECTIVO*

6.3.4 Atividade

A atividade no SADT é um conceito equivalente ao de processo no método de Gane [GAN 79] ou, a bolha dos diagramas do método de Tom De Marco [DeM 78]. A diferença é que o SADT usa além das setas de entrada e saída, seta(s) para controle. A especificação de atividade aqui apresentada, considera informações de denominação de atividade, e comentários acerca da função desempenhada. Além disso informações de natureza gráfica, complexidade e tempo para execução. Essa última informação foi incluída visando a simulação.

(21)	*ATIVIDADE*	::	*NOME* × *POSICAO*× *COMPLEXIDADE* × *COMENTARIO*× *TEMPO*
(22)	*NOME*	=	*String*
(23)	*POSICAO*	=	*PONTO*∗
(24)	*PONTO*	=	*Coord*
(25)	*COMPLEXIDADE*	=	*PRIMITIVA* ∪ *REFINADA*
(26)	*PRIMITIVA*	=	*Bool*
(27)	*REFINADA*	=	*INTERNA* ∪ *EXTERNA*
(28)	*INTERNA*	=	*SADT*
(29)	*EXTERNA*	=	*String*
(30)	*COMENTARIO*	=	*Texto*
(31)	*TEMPO*	::	*FAIXA* × *TEMPO_SIMUL*
(32)	*FAIXA*	::	*MINIMA* × *MAXIMA*
(33)	*MINIMA*	=	*Nat0*
(34)	*MAXIMA*	=	*Nat0*
(35)	*TEMPO_SIMUL*	=	*Nat0*
(36)	*COMENTARIO*	=	*Texto*

Anotações

- *ATIVIDADE* é abstraída nesta classe como uma árvore que é composta de *NOME*, *POSICAO*, *COMPLEXIDADE*, *COMENTARIO* e *TEMPO*.

- *NOME* utilizado para identificar uma atividade é definido por um "string".

- *POSICAO* é definida por uma lista de *PONTO* onde cada ponto é definido por uma coordenada.

- *COMPLEXIDADE* é o que determina se uma atividade é *REFINADA* ou *PRIMITIVA*. Uma Atividade *PRIMITIVA* não é desdobrada em outro(s) diagrama(s) sendo esse fato assinalado por um valor booleano. Atividade *REFINADA* é àquela que possui filho(s) representado(s) por outro(s) diagrama(s). Esse refinamento pode ser interno ou externo. Todo refinamento interno é um novo diagrama *SADT* ou seja, a chamada recursiva da classe. Refinamento externo refere-se a possibilidade de uso de um diagrama SADT já definido e presente em uma biblioteca do sistema, daí ser definido através de um "string".

- *COMENTARIO* é um texto livre sobre a atividade.

- *TEMPO* é o produto cartesiano entre *FAIXA*, *TEMPO_SIMUL*.

- *FAIXA* refere-se ao intervalo de tempo para a execução de uma atividade, sendo determinado em termos de *MINIMA* e *MAXIMA*.

- *MINIMA* e *MAXIMA* são definidas por um número natural, podendo ser zero.

- *TEMPO_SIMUL* é o tempo de simulação da execução de uma atividade. Esse tempo é dado por um número natural.

6.3.5 Conectivo

$$
\begin{array}{lrcl}
(37) & CONECTIVO & :: & POSICAO \times CATEGORIA \\
(38) & POSICAO & = & Coord \\
(39) & CATEGORIA & = & BRANCH \bigcup SPREAD \\
 & & & ORBRANCH \bigcup JOIN \\
 & & & BUNDLE \bigcup ORJOIN \\
(40) & BRANCH & = & Bool \\
(41) & SPREAD & = & Bool \\
(42) & ORBRANCH & = & Bool \\
(43) & JOIN & = & Bool \\
(44) & BUNDLE & = & Bool \\
(45) & ORJOIN & = & Bool \\
(46) & LIGACAO & = & RELACAO* \\
(47) & RELACAO & = & FLUXO* \\
(48) & FLUXO & = & ROTA
\end{array}
$$

Anotações

- *CONECTIVO* é abstraído pelo produto cartesinao *POSICAO* e *CATEGORIA*.
- *POSICAO* de um conectivo é definida por uma coordenada.
- *CATEGORIA* é abstraído pela união de classes *BRANCH*, *SPREAD*, *ORBRANCH*, *JOIN*, *BUNDLE* e *ORJOIN*. Todos esses conectivos na classe são definidos como booleanos.
- *LIGACAO* é definida como uma tupla de *RELACAO*.
- *RELACAO* é definida por uma tupla de *FLUXO*
- *FLUXO* é definido por uma *ROTA*

6.4 O Ambiente de Tratamento de Objetos SADT

O Ambiente de Tratamento de Objetos - ATO SADT é a ferramenta implementada no PROSOFT a partir da definição da classe que descreve a linguagem gráfica do método. Ao definir-se o SADT como uma classe, o projeto da ferramenta segue fortemente o paradigma de objetos [NUN 87],[NUN 89] e [NUN 90] isso significa que no PROSOFT cria-se o ambiente construindo um objeto pertencente a classe definida.

O SADT é uma método adequado para a fase de definição de requisitos, e, como uma ferramenta para o engenheiro de software, deve prover recursos modernos de comunicação homem-máquina, no sentido de tornar a atividade de um projetista mais segura e ergonômica. Diante disso, implementou-se um subconjunto do núcleo gráfico do SADT, visando facilitar a tarefa de especificar sistemas em notação diagramática do método, bem como gerenciar o processo de criação da solução como um todo. Assim é oferecida uma interface gráfica de alto nível, sendo que todas as operações do ambiente possuem verificação automática no momento em que são realizadas.

A ferramenta implementa um conjunto de operações básicas e de relevância para o contexto de um CASE. Vários esforços de automação do SADT segundo [ROS 85] não foram bem sucedidos, o método possui natureza complexa, o que fica bem evidenciado na definição abstrata da classe que foi apresentada descrevendo a ferramenta.

A ferramenta SADT proposta nesta dissertação, utiliza conceitos e técnicas provenientes de sistemas CAD e dos atuais ambientes de desenvolvimento. A utilização desses conceitos se fez necessária, porque o SADT usa vários simbolos gráficos para a representação de seus objetos de trabalho. A ferramenta foi projetada, com a idéia básica de seu usuário poder armazenar e visualizar graficamente a solução de um problema em análise.

A capacidade gráfica apresentada pela ferramenta é uma de suas características primordiais. A importância dos recursos gráficos em uma ferramenta foi muito delineada no trabalho de [RAM 87] onde afirma-se que uma imagem gráfica,

oferece um complemento natural a outras formas de comunicação, fato já registrado em [ROS 83,ROS 83a]. O tempo necessário para que o ser humano compreenda o significado de uma imagem gráfica é muito pequeno. Uma imagem possui uma quantidade muito grande de informação, e possui universalidade.

6.5 Operações da ferramenta SADT

Estabelecida a classe e incorporada ao Ambiente PROSOFT, o passo seguinte é criar um objeto e exibí-lo. Assim, construtivamente inicia-se a fase de definição do conjunto de operações oferecido pela ferramenta.

As operações definidas, objetivam oferecer ao usuário um conjunto abrangente de opções visando atendê-lo de forma confortável e segura.

Os objetos gráficos do SADT são diagramas, que são constituídos de atividades, conectivos e fluxos. As operações definidas para a ferramenta permitem inserções, remoções, modificações, exibições, movimentações etc. desses objetos no ambiente.

6.5.1 Operações sobre atividades

As operações definidas para atividade no Ato SADT foram as de inserção, remoção, nomeação, descrição, exibição de descrição, alteração de descrição, refinamento e remoção de refinamento. Esta última uma operação que deve ser usada com muito cuidado. Todas as sub-árvores refinadas de uma dada atividade desparecerão.

6.5.2 Operações sobre conectivos

Os conectivos do SADT são um recurso poderoso para diagramações. As operações definidas para esses construtos foram de: Inserção e remoção. Quaisquer dessas operações é feita bastando para tal, que o usuário selecione no menu essa opção.

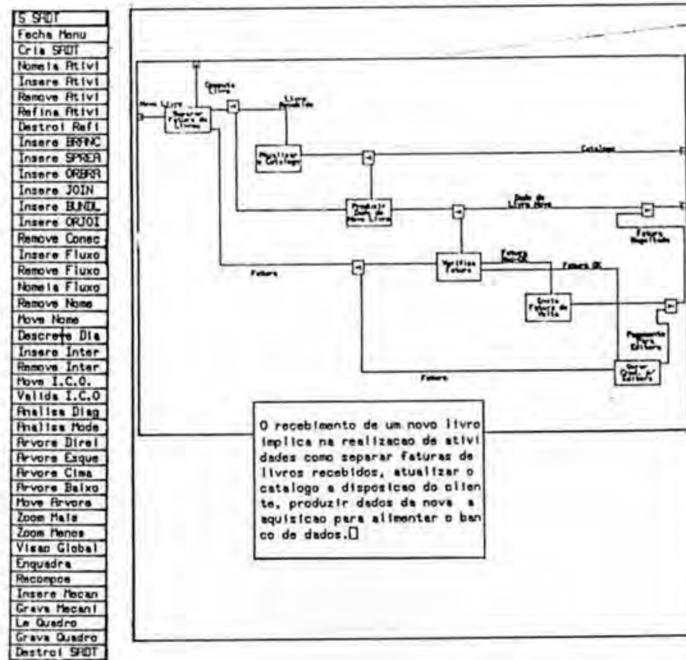


Fig. 6.1 Descrição de uma atividade

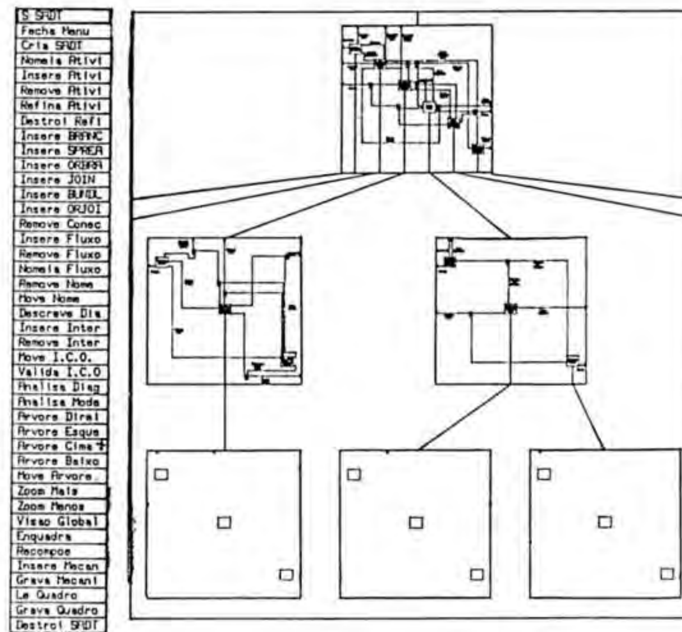


Fig. 6.2 Refinamento de atividade

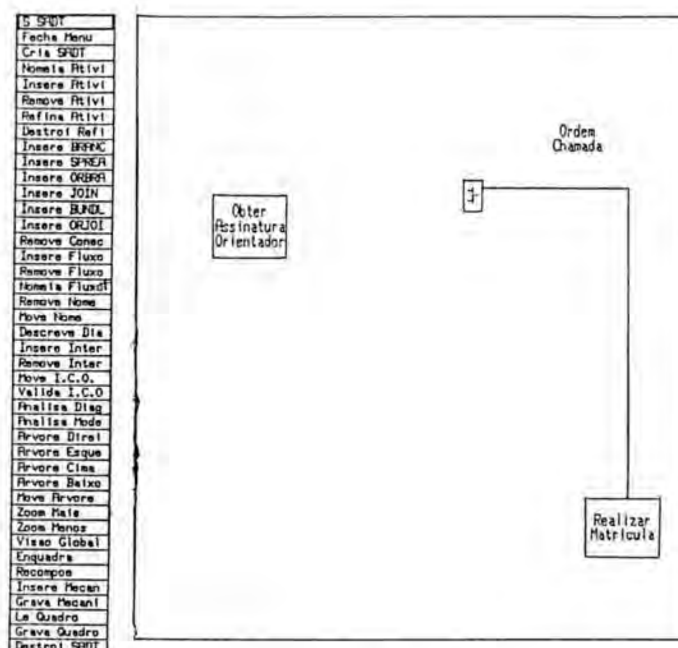


Fig. 6.3 Nomeação de um fluxo em um diagrama

Na operação de remoção de um conectivo, o sistema verifica se o conectivo possui fluxos na entrada e na saída, solicita uma confirmação e exclui o conectivo e todos os fluxos origem ou destino a ele. A execução dessa operação implica em percorrer a árvore de diagramas removendo as interfaces e fluxos dependentes do conectivo removido.

6.5.3 Operações sobre fluxo

O SADT é um método orientado a fluxo de dados. As operações para tratar esse construto no método são as de inserção, nomeação, remover nome do fluxo, alterar posição do nome e remoção.

A operação de remoção é realizada com o usuário apontando o fluxo a ser removido, em algum ponto bem próximo de algum segmento pertencente ao fluxo. Caso o fluxo esteja associado a uma atividade refinada será excluída a interface no diagrama de refinamento, bem como os fluxos no refinamento a ele ligados, como uma cascata, assim a consistência do modelo é perfeitamente garantida.

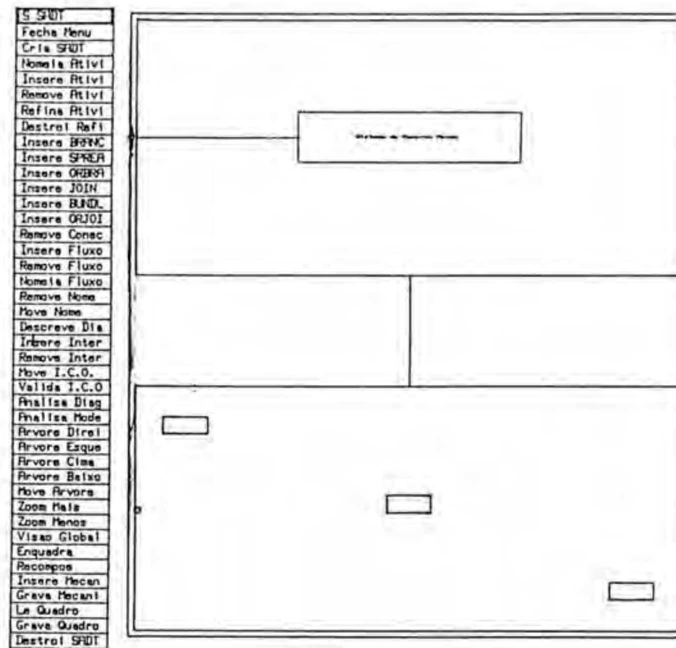


Fig. 6.4 Inserção de uma interface em um diagrama

6.5.4 Operações sobre interface

As operações definidas para interface foram as de inserção, remoção e movimentação de ICO (Input,Control e Output). Esta última operação é fornecida ao usuário, objetivando oferecer a possibilidade de se movimentar os pontos de interfaces gerados em um diagrama refinamento, sem mudar a ordem dessa geração. A operação possibilita apresentar os diagramas em uma forma mais legível. O movimento de uma interface em um diagrama refinado, não implica em movimentação desses pontos na atividade que lhe deu origem.

6.5.5 Operações sobre diagramas

O modelo de sistema definido com o SADT constitui em uma árvore de diagramas, desde a raiz até as folhas que compoem os últimos refinamentos. As operações definidas para uma árvore de diagramas foram as seguintes: Mover árvore à direita, à esquerda, para cima e para baixo. Adicionalmente a essas operações dispõe-se de Zoom+ e Zoom- na árvore ou em um de seus diagramas, recomposição de diagramas,

visão global e enquadramento.

6.5.6 Operações de análise

Essas operações permitem ao usuário fazer a verificação de sua diagramação acerca de um determinado problema, quanto a correção sintática. Essa correção pode ser verificada tanto no modelo como um todo, como num diagrama em particular.

Analisa diagrama

Esta operação oferece a possibilidade do usuário verificar se um determinado diagrama na árvore de desenvolvimento, está de acordo com as regras sintáticas definidas para o método SADT. Essas regras são as seguintes:

- Toda atividade obrigatóriamente deve possuir pelo menos um fluxo de controle e um fluxo de saída, sendo portando o fluxo de entrada não obrigatório.
- Todo conectivo de decomposição de dados (Branch, Spread e Or-Branch deve possuir um único fluxo de entrada e um mínimo de duas saídas [ROS 83a] e [ROS 85].
- Todo conectivo de composição de dados (Join, Bundle e Or-Join) deve possuir pelo menos dois fluxos de entrada e, somente um fluxo de saída.
- Todas as interfaces externas ao diagrama devem ser utilizadas.
- O fluxo de saída de uma atividade só pode ser entrada ou controle para uma atividade, entrada para um conectivo ou ligado a uma interface externa de saída, ou entrada para a mesma atividade que o gerou.

Essa operação só é realizada para diagramas refinados no modelo.

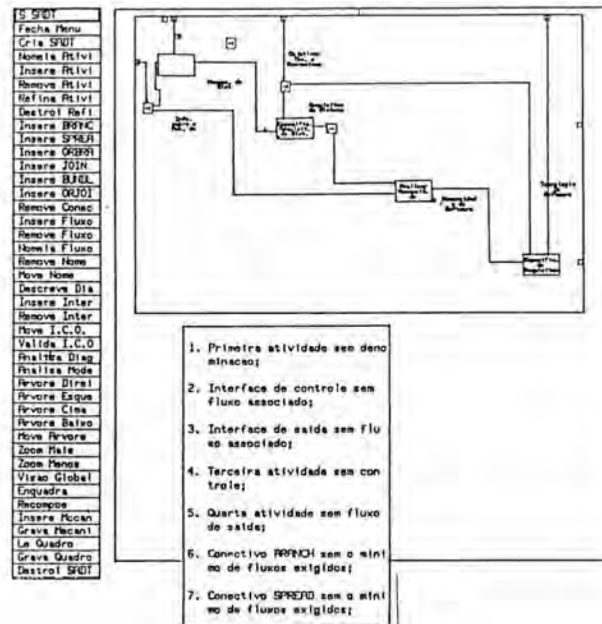


Fig. 6.5 Análise de um diagrama

Analisa Modelo

Esta operação permite ao usuário realizar uma análise do modelo, sendo verificado se cada refinamento possui o mesmo número e tipo de interfaces da atividade que o gerou. Essa operação é executada quando o usuário solicita uma análise no diagrama principal. Para realiza-la o usuário solicita a análise no menu e aponta na raiz.

6.5.7 Operações complementares

As operações complementares dizem respeito mais ao contexto da ferramenta junto ao ambiente de implementação. Dessa forma, nesse conjunto encontram-se as operações de iniciar sessão, criar SADT, ler e gravar quadro, gravar e inserir mecanismo, destruir SADT e finalmente, encerrar uma sessão PROSOFT.

MENU ATOS
atoset
atomap
sadt visual
cadastro
traitref
sortedop
conseq props
cons props
assert props
trait
texto
dicionario
espaco
quad
sadt
nsd
outros
quadro
diretorio
menu
classe
cenario
fim

S SADT
Fecha Menu
Cria SADT
Nomeia Ativi
Insere Ativi
Remove Ativi
Refina Ativi
Destroi Refi
Insere BRANC
Insere SPREA
Insere ORBRA
Insere JOIN
Insere BUNDL
Insere ORJOI
Remove Conec
Insere Fluxo
Remove Fluxo
Nomeia Fluxo
Remove Nome
Move Nome
Descreve Dia
Remove Descr
Insere Inter
Remove Inter
Move I.C.O.
Valida I.C.O
Analisa Diag
Analisa Mode
Insere Mecan
Grava Mecani
Le Quadro
Grava Quadro
Extingue Jan

SADT VISUAL
Fecha menu
Objeto Direi
Objeto Esque
Sobe Objeto
Desce Objeto
Move Objeto
Zoom Menos
Zoom Mais
Visao Global
Enquadra
Recompoe

Fig. 6.6 Início de sessão no Ambiente PROSOFT

7 ESPECIFICAÇÃO FORMAL DA SEMÂNTICA E DA EXECUÇÃO POR SIMULAÇÃO DE SISTEMAS DESCRITOS EM SADT

Neste capítulo é mostrada uma especificação formal do sistema para execução do SADT, via simulação. A especificação foi escrita em VDM [BJO 78],[JON 80] e a notação utilizada é a encontrada em [RIB 88].

A especificação formal aqui apresentada é feita com base puramente no problema de *executar* sistemas descritos em SADT. Para que isso seja possível, será determinada uma classe apropriada ao problema alvo, definida a semântica formalmente, para cada construto de interesse do SADT para a simulação e finalmente apresentado o simulador e o executor dessas especificações.

A proposição, apresentada nesta dissertação de execução de especificações SADT através de simulação, se constitui em uma importante aplicação de um método formal na área de engenharia de software. O VDM já foi utilizado para especificar sistemas de arquivos [BJO 82], bancos de dados[BJO 82], partes de um sistema operacional[BJO 78], compiladores para ALGOL e PASCAL[BJO 82], semântica de máquinas de fluxo de dados [JON 87] e outros sistemas. A formalização do SADT é uma proposição absolutamente nova, principalmente no sentido de fornecer uma base para a execução de uma especificação via simulação. As

dissertações apresentadas no Brasil [LEA 88] e [TOL 89] não deram tratamento formal ao método, nem definiram semânticas para seus construtos. O projeto francês SPECIF-X [LIS 87] levanta a possibilidade de que a simulação possa vir a ser conseguida no futuro, mas não apresenta proposições concretas nesse sentido. Esta dissertação apresenta como parte prática a implementação de uma ferramenta para apoio ao engenheiro de software no uso de diagramações SADT no ambiente PROSOFT [NUN 90] e, como parte teórica a especificação formal da semântica e da simulacão do método SADT.

7.1 O sistema para simulação do SADT

O sistema para simulação da "execução" de especificação de sistemas descritos em SADT é apresentado, considerando a aderência do método aos sistemas de fluxos de dados, como apresentado no capítulo 3. Atividades consomem e produzem "tokens" tanto quanto os conectivos interfaceando-as. Para atingir o objetivo de executar uma especificação SADT via simulação, deve-se estabelecer precisamente o comportamento (semântica) de seus construtos, e de posse disso, fazer a simulação em modo discreto.

7.2 Organização da especificação

A seção 7.3 mostra o modelo denotacional do sistema de simulação para o SADT. Na seção 7.4 comenta-se acerca de restrições ao domínio semântico definido anteriormente. A seção 7.5 apresenta as equações que estabelecem as restrições aos objetos do domínio semântico. A seção 7.6 apresenta a definição da sintaxe das operações necessárias, para a execução do modelo por simulação. A seção 7.7 define as "signatures" das operações principais e auxiliares para a execução das especificações SADT, por simulação. Na seção 7.8 são definidas as funções semânticas do modelo. Na seção 7.9 comenta-se acerca dos refinamentos que podem ser incorporados à especificação, visando direcioná-la à implementação. Finalmente na seção 7.10 são tecidas considerações, sobre a especificação apresentada.

7.3 Definição do domínio semântico

Domínio semântico é um reticulado contínuo, completo e de base contável [MAR 88]. A especificação do domínio semântico é a tarefa mais importante que se faz ao se especificar um problema em VDM.

Os domínios semânticos são definidos a partir de objetos matemáticos bem definidos [MAR 88],[RIB 88] e [JON 80]. Os objetos pertencentes a uma classe assim definida, "herdam" as operações matemática de seus tipos primitivos.

Esse passo tão fundamental é uma tarefa difícil de ser realizada. Um domínio semântico foi definido no capítulo anterior, quando tratou-se da ferramenta para definição de requisito, que foi implementada no PROSOFT, como parte dessa dissertação. A classe utilizada para a implementação da ferramenta prevê simulação, mas é de natureza essencialmente gráfica, já que a ferramenta construída possui essa ênfase.

O domínio semântico que deve ser definido para a execução de especificações SADT através de simulação é um domínio diferente. Não interessa neste momento, preocupações de natureza gráfica, o que importa é reunir em uma classe, as informações de interesse para o problema da simulação. Desta forma, foi estabelecido o domínio que se segue, visualizando uma especificação de um sistema em SADT, em termos de produtores e consumidores de informação. Todo produtor é um componente (um construto) válido no método. Além disso, considera-se que outro dado muito importante para efeitos de simulação é o estado, tanto do sistema como um todo, como de seus componentes individuais. Outra informação importante é a concepção de fluxos de dados como filas, tendo estas informações como origem, destino, comprimento inicial e comprimento final. Cada a atividade é um processo a ser simulado, e a este, agrega-se outro conjunto de informações essenciais para a simulação. Com base nessas premissas foi definido o domínio que se segue.

$$
\begin{array}{rrcl}
(49) & \mathit{SISTEMA} & = & (\mathit{PRODUTOR} \times \mathit{CONSUMIDOR})\text{-set} \\
(50) & \mathit{CONSUMIDOR} & = & \mathit{COMPONENTE} \\
(51) & \mathit{COMPONENTE} & :: & \mathit{NOMEC} \times \mathit{TIPO} \times \\
 & & & \mathit{INTERFACES} \times \mathit{ORDEM} \\
(52) & \mathit{NOMEC} & = & \mathit{String} \\
(53) & \mathit{TIPO} & = & \{\mathit{ATIVIDADE}, \\
 & & & \mathit{BRANCH}, \mathit{SPREAD}, \\
 & & & \mathit{ORBRANCH}, \mathit{JOIN}, \\
 & & & \mathit{BUNDLE}, \mathit{ORJOIN}, \\
 & & & \mathit{EXTERNO}\} \\
(54) & \mathit{INTERFACES} & = & \{E, K, S\} \\
(55) & \mathit{ORDEM} & = & \mathit{Nat}1 \\
(56) & \mathit{PRODUTOR} & = & \mathit{COMPONENTE} \\
(57) & \mathit{ESTADO} & :: & \mathit{FILA}\text{-set} \times \mathit{PROCESSO}\text{-set} \\
(58) & \mathit{FILA} & :: & \mathit{ORIGEM} \times \mathit{DESTINO} \times \\
 & & & \mathit{COMPINIC} \times \mathit{COMPFIN} \\
(59) & \mathit{ORIGEM} & = & \mathit{COMPONENTE} \\
(60) & \mathit{DESTINO} & = & \mathit{COMPONENTE} \\
(61) & \mathit{COMPINIC} & = & \mathit{Nat} \\
(62) & \mathit{COMPFIN} & = & \mathit{Nat} \\
(63) & \mathit{PROCESSO} & :: & \mathit{NOMEP} \times \mathit{TMIN} \times \\
 & & & \mathit{TMAX} \times \mathit{TEMPOSIMUL} \times \\
 & & & \mathit{SORTEIO} \\
(64) & \mathit{NOMEP} & = & \mathit{String} \\
(65) & \mathit{TMAX} & = & \mathit{Nat} \\
(66) & \mathit{TMIN} & = & \mathit{Nat} \\
(67) & \mathit{TEMPOSIMUL} & = & \mathit{Nat} \\
(68) & \mathit{SORTEIO} & = & \{\mathit{Poisson}, \mathit{Normal}\}
\end{array}
$$

7.3.1 Anotações

49 O sistema de simulação é aqui seletivamente abstraído pelo objeto matemático conjunto, sendo cada elemento deste, o par *PRODUTOR* e *CONSUMIDOR*.

50 *CONSUMIDOR* é um *COMPONENTE* do SADT.

51 *COMPONENTE* é abstraído seletivamente como o produto cartesiano de *NOME*, *TIPO*, *INTERFACE* e uma *ORDEM*.

52 Nome do componente *NOME*é representado por um "string".

53 O *TIPO* de um componente, é um dos elementos constantes do conjunto de construtos básicos em um diagrama SADT. Esses construtos são: a *ATIVIDADE*, *BRANCH*,*SPREAD*,*ORBRANCH*,*JOIN*,*BUNDLE*,*ORJOIN*e um elemento *EXTERNO* a um diagrama.

54 As *INTERFACES* de um componente são os elementos do conjunto $\{E,K,S\}$, indicando entrada, controle e saída.

55 A *ORDEM* é representada por um natural, indicando seqüencia de uma interface, relacionada a um componente.

56 Tal como o *CONSUMIDOR*, um *PRODUTOR* de informações também é um *COMPONENTE*.

57 O *ESTADO* de um sistema é abstraído seletivamente pelo produto cartesiano, representado pelos conjunto de *FILAS* e de *PROCESSOS*, presentes no sistema em um determinado instante.

58 Uma *FILA* no sistema de simulação é abstraída em uma árvore que possui: *ORIGEM*, *DESTINO*, comprimento inicial *COMPINIC* e comprimento final *COMPFIN*.

59 Qualquer *FILA* em um diagrama SADT tem *ORIGEM* em um *COMPONENTE*. Não esquecendo que entre os componentes válidos figura o meio *EXTERNO* ou interfaces.

60 O *DESTINO* de qualquer *FILA* é um *COMPONENTE*, isso se faz necessário, para que os diversos fluxos fiquem completamente determinados. O componente destino, similarmente a origem também pode ser o meio *EXTERNO*.

61 O comprimento inicial de uma fila *COMPINIC* é representado por um natural e determina o seu número de "tokens" antes de um ciclo de simulação.

62 O comprimento final de uma fila, *COMPFIN* é representado por um natural e determina o número de "tokens" presentes na fila após um ciclo de simulação.

63 Um *PROCESSO* presente em um diagrama, é representado por uma atividade SADT, consistindo em *NOMEP*, tempo mínimo *TMIN* e máximo *TMAX* de execução, tempo de simulação *TEMPOSIMUL* e um sorteio.

64 O nome de um processo *NOMEP* é um "string".

65 O menor tempo que um processo pode ser executado *TMIN* é abstraído por um natural excluindo o zero.

66 O maior tempo que um processo leva para ser executado *TMAX* também é abstraído por um natural excluindo o zero.

67 O tempo de simulação de um processo *TEMPOSIMUL* é abstraído por um natural, incluindo o zero, valor que quando for atingido em tempo de simulação, é indicativo de que o processo deve produzir suas saídas.

68 O *SORTEIO* para um processo representa a escolha de uma distribuição de probabilidades, para a determinação de seu intervalo de execução. Essa distribuição pode ser a de Poisson ou Normal.

A descrição da classe do sistema de simulação vista acima pode ser lida como uma fórmula, visto que ela descreve completamente o ambiente para simular a execução de um sistema especificado em SADT. A fórmula define o domínio semântico, isto é, a classe dos objetos com os quais se trabalha no sistema de simulação.

7.3.2 A instanciação de um objeto

Qualquer objeto SADT é pertencente à classe sistema definida, e pode ser um mapeamento da seguinte forma. Esse mapeamento permite que se opere com o objeto, aplicando o conjunto de funções semânticas definidas neste capítulo, para a obtenção da execução da especificação, através de simulação. Deve ser observado que no domínio semântico apresentado inexiste qualquer informação de natureza gráfica.

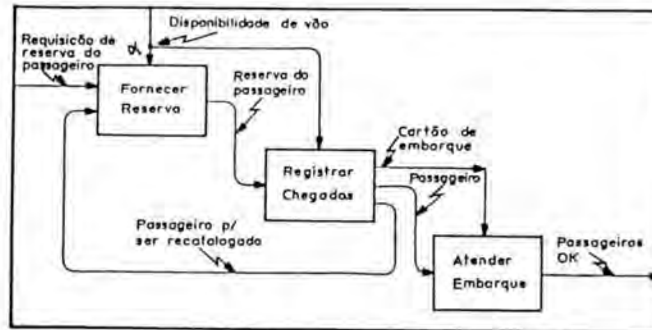


Fig. 7.1 Sistema de reserva e atendimento de passageiros

Assim, obtem-se uma classe, cuja preocupação principal é reunir informações estritamente essenciais para a simulação de sistemas descritos em SADT.

A instanciação apresentada a seguir, refere-se a um fictício sistema para reserva e atendimento de passageiros em um aeroporto. Apresenta-se somente um diagrama com três atividades básicas: fornecer reserva, registrar chegadas e atendimento de embarque.

```
Sistema =
  {((Fornecer Reserva, A,s,1) , (Registrar Chegada,A,e,1)),
   ((Registrar Chegada, A,s,1), (Atender Embarque,A,k,1)),
   ((Registrar Chegada,A,s,2) , (Atender Embarque,A,e,1)),
   ((Registrar Chegada,A,s,3) , (Fornecer Reserva,A,e,s)),
   ((Atender Embarque,A,s,1)  , (_,Ext,s,1)),
   ((_,Ext,e,1)               , (Fornecer Reserva,A,e,1)),
   ((_,Ext,e,1)               , (Alfa, Branch,e,1)),
   ((Alfa,Branch,s,1)         , (Fornecer Reserva,A,k,1)),
   ((Alfa,Branch,s,2)         , (Registrar Chegadas,A,k,1))}
```

7.4 Restrições sobre os domínios semânticos

Na criação do modelo que permita a execução do SADT por simulação interessa determinar como ocorre esse processo. O sistema na sua generalidade é visto no contexto do objeto matemático conjunto, onde cada elemento é composto por uma árvore *PRODUTOR* x *CONSUMIDOR* de informação. Esses componentes básicos apresentam um estado a cada instante, caracterizado por seus conjuntos de *FILA* e *PROCESSO*.

7.4.1 Função de boa formação dos objetos

A restrição aos objetos de uma classe feita com o uso da função "is-wf" ("is-well-formedness constraints") objetiva indicar como construir um objeto pertencente a classe definida, e que seja de interesse. No VDM isso é feito através de um predicado, o que permite caracterizar melhor os objetos com os quais se trabalha. A idéia central e fazer uma limitação à escolha dos objetos que estão na classe definida.

No sistema de simulação é de interesse determinar que os objetos do sistema, consumidores e produtores de informação, sejam bem formados. A descrição do domínio semântico para o sistema de simulação capturou a essência, de como abstratamente visualiza-se o sistema, mas os domínios definidos contém objetos como atividades, conectivos de composição e de decomposição de dados, além de elementos externos(interfaces), os quais obrigatoriamente deverão satisfazer as restrições a seguir especificadas. É necessário que essas observações sejam feitas, porque se assim não se proceder, torna-se irreal e impraticável a execução de especificações descritas em SADT através de simulação.

7.5 Restrição aos objetos da classe

A restrição aos objetos pertencentes a uma classe em uma especificação VDM, diz respeito a seletividade que se faz aos objetos da classe. Isso significa que através de um predicado descreve-se as restrições que satisfaçam ao problema em análise.

É na realidade separar o subconjunto dos objetos válidos, para a especificação do problema. Esse subconjunto deve garantir que um componente passa de um estado para outro estado e continua sendo um componente válido. Desta forma, o domínio de partida é o componente e o contradomínio ou conjunto de chegada também é um componente.

(69) $COMPONENTE : COMPONENTE$

```
69.0  is_wf_COMPNTE(mk-COMPNTE(n, t, inter, elem))=componente
 .1   Cases componente
 .2       (Atividade →
 .3               if inter = e
 .4                   then elem ≤ 1
 .5                   else elem ≥ 1),
 .6       (Branch,
 .7       Spread,
 .8       OrBranch →
 .9               if inter = e
 .10                  then elem = 1
 .11                  else elem ≥ 2),
 .12      (Join,
 .13      Bundle,
 .14      OrJoin →
 .15              if inter = e
 .16                  then elem ≥ 2
 .17                  else elem = 1),
 .18      (Externo →
 .19              if inter = e ∨ c ∨ s
 .20                  then elem = 1
 .21                  else elem False)
```

7.5.1 Observações

Para um componente do sistema, que contém nome, tipo, interfaces e a ordem destas, ser bem formado, as seguintes restrições devem ser satisfeitas.

Os componentes de um diagrama SADT para os propósitos de simulação são os seguintes: *ATIVIDADE*, *BRANCH*, *SPREAD*, *ORBRANCH*, *JOIN*, *BUNDLE*, *ORJOIN* ou *EXTERNO* (interface).

Caso o componente seja uma *ATIVIDADE*, a sua interface de entrada

poderá ou não existir, e suas interfaces de controle e saída deverão obrigatoriamente existir em número de pelo menos uma para cada desses tipos.

Caso o componente seja um conectivo de decomposição de dados (*BRANCH*, *SPREAD*, *ORBRANCH*), ele deverá apresentar somente uma interface de entrada e duas ou mais de saída.

Caso o componente seja um conectivo de composição de dados (*JOIN*, *BUNDLE*, *ORJOIN*), ele deverá ter um mínimo de duas interfaces de entrada e somente uma interface de saída.

Caso o componente seja *EXTERNO*, sua origem deverá ser como um fluxo de entrada ou de controle e seu destino como um fluxo de saída. E esse componente, em cada caso, deverá ser constituído de um único elemento.

A função "is-wf" foi definida relativa ao domínio de componentes, ou seja a classe de todos os componentes do diagrama SADT. Assim, entende-se por componentes, a classe daqueles objetos, que satisfazem o predicado (69). Isso significa dizer que qualquer manipulação, ou melhor, qualquer transformação de um processo sobre um componente do diagrama, leva-o a outro estado, satisfazendo as restrições de boa formação. Dessa maneira, o predicado (69) é também visto como formando a grande parte das invariantes do modelo de simulação, para o qual se faz algum tipo de manipulação, seja concreta ou abstratamente.

7.6 Definição da sintaxe das operações

Neste ponto da especificação, define-se as operações que possibilitam interagir com os objetos do sistema de simulação. Essas operações são definidas no VDM via uma sintaxe abstrata, ou seja, a definição e construção dos domínios envolvidos na sintaxe dos objetos em discussão.

7.6.1 O domínio sintático

No sistema de simulação de um ponto de vista de suas operações, interessa em nível mais interno, a simulação execução dos componentes do SADT atividades, conectivos e fluxos presentes em um diagrama. Em plano mais externo, situa-se a operação de simular. Essa operação parte do sistema e seu estado, chegando a um novo estado após cada ciclo de simulação. Essas operações sobre o sistema, em termos sintáticos são descritas à seguir:

(70)	$OPER$	$=$	$INIC \bigcup EXECUTE \bigcup SIMULAR$
(71)	$INIC$	$=$	$INIC \times SISTEMA$
(72)	$EXUCUTE$	$::$	$COMPONENTE \times ESTADO$
(73)	$SIMULAR$	$::$	$SISTEMA \times ESTADO$

Observações

A função *inic* inicializa o sistema. Ela gera os objetos do sistema de simulação, deixando o sistema pronto para ser operado. Uma operação no sistema de simulação, ou a simulação do sistema como um todo.

A função *execute* simula o funcionamento de cada componente de um diagrama SADT. Essa simulação é feita em modo discreto, sendo em cada instante obtido um componente e seu estado, o qual no próximo instante atinge outro estado no ciclo de simulação. *Execute* é uma operação que parte de um determinado componente no diagrama SADT e seu estado, e leva-o a outro estado. Na finalização dessa operação, o componente ativado deverá produzir sempre suas saídas.

A função *simular* simula o funcionamento do sistema como um todo

descrito no diagrama SADT. Esse processo de simulação é feito em um nível acima em relação a função execute.

A simulação do sistema descrito em SADT, considera o estado do sistema como um todo e deverá levá-lo a outro estado, após cada ciclo de simulação.

7.7 Definição das assinaturas das operações

As assinaturas em uma especificação formal representam os "sorts" do problema sendo especificado [GUT 78] e os relacionamentos para a realização das operações [CLE 86],[COH 86]. Essas assinaturas ou "signatures" são fundamentais para uma especificação formal. Representam papel semelhante ao de tipo em linguagem de programação. No problema aqui tratado usa-se os "sorts" SISTEMA, ESTADO, PROCESSO, FILA-set, COMPONENTE, COMPONENTE-set, PRODUTOR CONSUMIDOR e BOOLEANO. Isso significa que esses "sorts" não necessitam ser definidos são tratados apartir de agora como elementos primitivos na especificação VDM. A figura 7.2 apresenta um diagrama ADJ das assinaturas, como proposto em [JON 80]. Os números inteiros próximo a cada arco, correspondem aos números para cada operação das assinaturas na página seguinte.

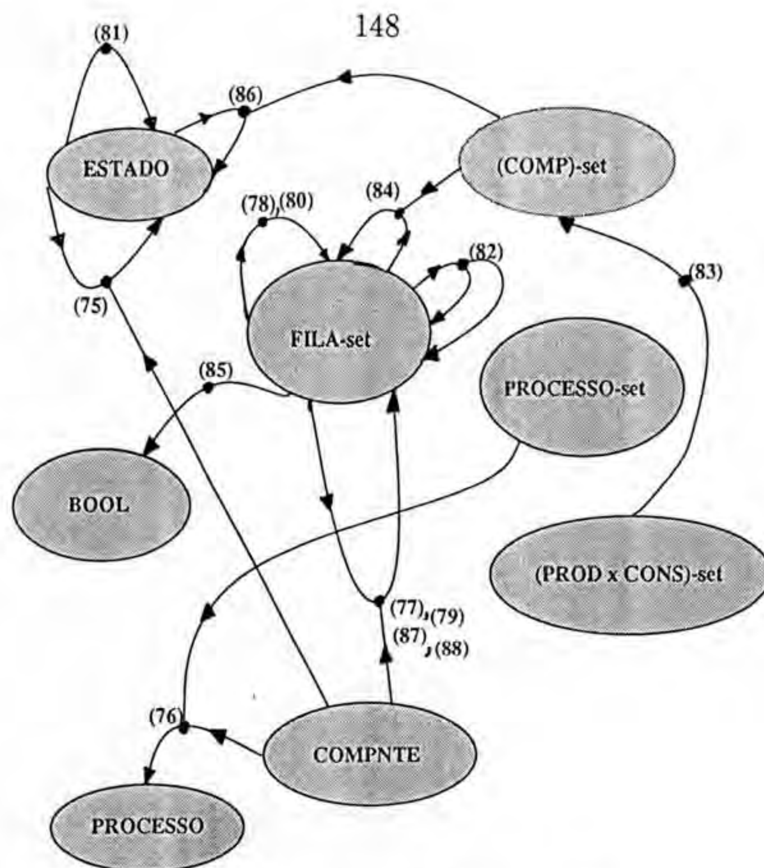


Fig. 7.2 Diagrama de assinaturas

7.7.1 Assinaturas

(74) *type*: *INIC*: { } $\rightarrow$ *SISTEMA*
(75) *type*: *EXECUTE*: *COMPNTE* $\times$ *ESTADO* $\rightarrow$ *ESTADO*
(76) *type*: *PROC_PROCESSO*: *COMPNTE* $\times$ *PROCESSO*-set $\rightarrow$ *PROCESSO*
(77) *type*: *PROC_INT_ENT*: *COMPNTE* $\times$ *FILA*-set $\rightarrow$ *FILA*-set
(78) *type*: *ATLZA_INT_ENT*: *FILA*-set $\rightarrow$ *FILA*-set
(79) *type*: *BUSCA_INT_SAIDA*: *COMPNTE* $\times$ *FILA*-set $\rightarrow$ *FILA*-set
(80) *type*: *ATUALIZA_INT_SAIDA*: *FILA*-set $\rightarrow$ *FILA*-set
(81) *type*: *SIMULAR*: *SISTEMA* $\times$ *ESTADO* $\rightarrow$ *ESTADO*
(82) *type*: *ATUALIZA_FILAS*: (*FILA*-set $\rightarrow$ *FILA*-set) $\rightarrow$ *FILA*-set
(83) *type*: *PRODUTORES*: (*PROD* $\times$ *CONS*)-set $\rightarrow$ (*COMP*)-set
(84) *type*: *HABILITADOS*: *COMP*-set $\times$ *FILA*-set $\rightarrow$ *FILA*-set
(85) *type*: *TESTE*: *FILA*-set $\rightarrow$ *BOOL*
(86) *type*: *PROCESSAR*: *COMP*-set $\times$ *ESTADO* $\rightarrow$ *ESTADO*
(87) *type*: *BUSC_INT_ENT_RAND*: *COMPNTE* $\times$ *FILA*-set $\rightarrow$ *FILA*-set
(88) *type*: *BUSC_INT_SAIDA_RAND*: *COMPNTE* $\times$ *FILA*-set $\rightarrow$ *FILA*-set

7.8 Definição das funções semânticas

A definição das funções semânticas em abordagem denotacional permitem oferecer uma visão em alto nível de abstração, do significado dos construtos do SADT e da execução de uma especificação por simulação. Dentro dessa ótica, considera-se o programa como especificando uma função de um tipo apropriado. Assim sendo, o significado de uma especificação em abordagem denotacional é dado não em termos de seqüência de estados, mas simplesmente como uma função de estado para estado. Nessa abordagem, ainda que se possa considerar as equações como definindo uma sequência, também consideram-se as definições como fornecendo funções abstratas sobre estados com seqüenciamento explícito ou não.

As funções semânticas a seguir ilustram explícitamente os procedimentos necessários, para a execução por simulação de uma especificação feita em diagramas SADT. Nesta dissertação apresenta-se as funções semânticas do problema em estudo, no seu primeiro nível de abstração. Essa especificação poderá ser sucessivamente refinada, em direção a uma concreta linguagem de programação. Esse não é o objetivo deste trabalho; a principal preocupação é, no alto nível de uma abstração VDM, especificar completamente a semântica(comportamento) dos construtos do método SADT e a simulação destes.

7.8.1 Inicialização do sistema

Essa função é responsável pela inicialização do sistema de simulação. Ela cria os objetos consumidores e produtores de informações (atividades, conectivos, fluxos etc..) necessários para a realização das operações para executar sistema, através de uma simulação.

74.0 $inic: \{\} \rightarrow SISTEMA$

7.8.2 Função execute

A função *execute* parte de um componente e seu estado e retorna um outro estado. Para o SADT os componentes que serão executados são por ordem de apresentação na especificação: *ATIVIDADE*,*BRANCH*, *SPREAD*, *ORBRANCH*, *JOIN*, *BUNDLE* e *ORJOIN*.

A execução de cada atividade do diagrama tomará por base algumas premissas estabelecidas no capítulo 4, onde foi feita uma abordagem de semântica operacional para atividade e conectivos. No caso da atividade relembrando tem-se o seguinte:

```
se tempo_simul = 0

  entao Verificar se atividade pode ser disparada,
        caso positivo consumir entrada e controle
        e sortear um tempo para simulação  para a
        atividade.


se tempo_simul = 1

   entao Produzir saída, fazer tempo de simulação da
        atividade nulo, verificar  se  a  atividade
        pode ser disparada, caso positivo  consumir
        entradas e controles e sortear um novo tempo
        para simulação da atividade.


se tempo_simul > 1

  entao Decrementar uma unidade do tempo de simulação
        da atividade.
```

Todos os conectivos do SADT de composição de dados(*JOIN*,*BUNDLE* e *ORJOIN*), tanto quanto os de decomposição de dados(*BRANCH*, *SPREAD*, *ORBRANCH*), são executados de maneira muito semelhante entre si. As funções semânticas desses construtos são claras o suficiente, não necessitando de maiores detalhamentos. E mesmo uma especificação formal para ser entendida é como uma prova matemática, deve ser explorada o suficiente pela pessoa que vai trabalhar com ela [MAC 87].

```
75.0  Elab_Execute(comp, estado) △
  .1  Cases comp
  .2          mk-Compte(a, ATIVIDADE, _, _) →
  .3              let mk-Est(fs, ps) = estado
  .4              let mk-Proc(_, tmin, tmax, ts, s) =
  .5                  procura_processo(comp, ps) in
  .6              if ts > 1
  .7              then mk-Est(fs, ps − {mk-Proc(a, tmin, tmax, ts, s)}
  .8                                       ∪
  .9                  {mk-Proc(a, tmin, tmax, ts − 1, s)})
 .10              else if ts = 0
 .11                  then let fs' = (fs − busc_int_ent(comp, fs))
 .12                                               ∪
 .13                  atlza_int_ent(busc_int_ent(comp, fs))
 .14                  let ts' = if s = Poisson
 .15                               then Poisson(tmin, tmax)
 .16                               else Normal(tmin, tmax)
 .17                  let ps' = (ps − {mk-Proc(a, tmin, tmax, ts', s)}
 .18                                           ∪
 .19                      {mk-Proc(a, tmin, tmax, ts', s)})
 .20                  let fs'' = (fs' − busc_int_sai(comp, fs'))
 .21                                           ∪
 .22                      atlza_int_sai(busc_int_sai(comp, fs')) in
 .23                  mk-Est(fs'', ps')
 .24                  else let fs' = (fs − busc_int_ent(comp, fs))
 .25                                           ∪
 .26                      atlza_int_ent(busc_int_ent(comp, fs))
 .27                      let ts' = 0
 .28                      let ps' = (ps − {mk-Proc(a, tmin, tmax, ts, s)}
 .29                                           ∪
 .30                      {mk-Proc(a, tmin, tmax, ts', s)})
 .31                      let fs'' = (fs' − busc_int_sai(comp, fs'))
 .32                                           ∪
 .33                      atlza_int_sai(busc_int_sai(comp, fs')) in
 .34                      mk-Est(fs'', ps')
```

A função semântica *execute* embora apareça aqui fragmentada com uma parte para cada construto, deve ser vista como compondo uma unidade de especificação.

A função *execute* tem como domínio componente e estado, como imagem (o retorno) estado. A elaboração de *execute* foi definida como uma estrutura de caso. Todo componente no domínio semântico é um objeto composto, formado por nome, tipo de construto, tipo de interface e a ordem em que estas aparecem. Caso o componente seja o construto atividade constroi-se seu estado apartir do conjunto de filas e do conjunto de processos. Feito isso, constroi-se o processo partindo do domínio de nome do processo, o que não interessa neste nível de especificação, os tempos máximo e mínimo de execução da atividade, o tempo de simulação e o sorteio da distribuição de probabilidades Normal ou Poisson. Se o tempo de simulação na atividade for superior a unidade, isso significa a existencia de uma atividade em processamento. Será construído então um novo estado com o conjunto de filas e o conjunto de processos retirando deste, ele próprio, unido a processo com tempo de simulação atualizado. Mas se o tempo de simulação for zero, de acordo com a secção 7.82 deve-se verificar se a atividade pode ser disparada, caso seja verdade serão consumidos os "tokens" da(s) entrada(s) e controle(s) da atividade sendo então sorteado um tempo de simulação para a atividade. Isso é feito da seguinte maneira: o novo estado do conjunto de filas fs' é fornecido pelo conjunto de filas fs retirando-se deste, o conjunto de filas que chegam ao componente unido às filas atualizadas do componente. Feito isso, sorteia-se um tempo para a execução da atividade na distribuição de probabilidades escolhida. Atualiza-se o processo ps' e o novo estado do conjunto de filas representado por fs", retornando assim um novo estado.

No caso de o tempo de simulação da atividade ser zero examina-se a habilitação da atividade. Se esta puder ser disparada serão consumidos os "tokens" de entrada(s) e controle(s) e sorteado um tempo para simulação da atividade.

A especificação da semântica dos conectivos *BRANCH*, *JOIN*, *SPREAD*,*BUNDLE*,*ORJOIN* e *ORBRANCH* é mais simples de ser especificada em VDM. Essas especificações são exatamente como proposto no capítulo 4 desta dissertação. Essa simplicidade deve-se ao fato de que considera-se conectivos como micro-atividades de execução instantânea.

```
.0     mk-Compte(a, BRANCH, _, _) →
.1         let mk-Est(fs, _) = estado
.2         f ∈ procura_int_entrada(comp, fs)
.3         let mk-Fila(o, d, ci, cf) = f in
.4         if ci = 0
.5              then mk-Est(fs, _)
.6              else let fs' = (fs − busc_int_ent(comp, fs))
.7                                ∪
.8                       atlza_int_ent(busc_int_ent(comp, fs))
.9                       let fs'' = (fs' − busc_int_sai(comp, fs'))
.10                               ∪
.11                      atlza_int_sai(busc_int_sai(comp, fs')) in
.12                      elab_Execute(comp, mk-Est(fs'', _))
```

A semântica do conectivo BRANCH é especificada da seguinte maneira:

- É construído o objeto Componente do tipo BRANCH. As informações de interface e ordem, não interessam neste momento.
- É construído o Estado que para os propósitos do construto BRANCH, só interessa o conjunto de filas representado por fs.
- A fila f pertence ao conjunto das filas de entrada do conectivo
- A fila f é construída possuindo origem, destino, comprimento inicial "ci" e comprimento final.
- Se o comprimento inicial da fila for zero (inexistência de "tokens" na fila de entrada).
- Então retorna o estado
- Senão é definida a fila fs' como fs retirando-se desta, o conjunto de filas do componente unido com a fila atualizada.

- Define-se então a fila fs" de maneira similar a fs', unida com a atualização da fila de saída, chamando-se recursivamente a função semântica *execute*.

As semânticas dos conectivos *SPREAD*,*ORBRANCH*,*JOIN*,*BUNDLE* e *ORJOIN* são definidas em modo muito similar ao de *BRANCH*.

```
.0      mk-Compte(a, SPREAD, _, _) →
.1          let mk-Est(fs, _) = estado
.2          f ∈ procura_int_entrada(comp, fs)
.3          let mk-Fila(o, d, ci, cf) = f in
.4          if ci = 0
.5                  then mk-Est(fs, _)
.6                  else let fs' = (fs − busc_int_ent(comp, fs))
.7                                 ∪
.8                          atlza_int_ent(busc_int_ent(comp, fs))
.9                          let fs'' = (fs' − busc_int_sai(comp, fs'))
.10                                ∪
.11                         atlza_int_sai(busc_int_sai(comp, fs')) in
.12                         elab_Execute(comp, mk-Est(fs'', _))
```

```
.0     mk-Compte(a, ORBRANCH, _, _) →
.1         let mk-Est(fs, _) = estado
.2         f ∈ procura_int_entrada(comp, fs)
.3         let mk-Fila(o, d, ci, cf) = f in
.4         if ci = 0
.5                 then mk-Est(fs, _)
.6                 else let fs' = (fs − busc_int_ent(comp, fs))
.7                                 ∪
.8                         atlza_int_ent(busc_int_ent(comp, fs))
.9                         let fs'' = (fs' − busc_int_sai(comp, fs'))
.10                                ∪
.11                        atlza_int_sai(busc_int_sai_rand(comp, fs')) in
.12                        elab_Execute(comp, mk-Est(fs'', _))
```

```
.0     mk-Compte(a, JOIN, _, _) →
.1         let mk-Est(fs, _) = estado
.2         f ∈ procura_int_entrada(comp, fs)
.3         let mk-Fila(o, d, ci, cf) = f in
.4         if ci = 0
.5                 then mk-Est(fs, _)
.6                 else let fs' = (fs − busc_int_ent(comp, fs))
.7                                 ∪
.8                         atlza_int_ent(busc_int_ent(comp, fs))
.9                         let fs'' = (fs' − busc_int_sai(comp, fs'))
.10                                ∪
.11                        atlza_int_sai(busc_int_sai(comp, fs')) in
.12                        elab_Execute(comp, mk-Est(fs'', _))
```

```
.0      mk-Compte(a, BUNDLE, _, _) →
.1          let mk-Est(fs, _) = estado
.2          f ∈ procura_int_entrada(comp, fs)
.3          let mk-Fila(o, d, ci, cf) = f in
.4          if ci = 0
.5                  then mk-Est(fs, _)
.6                  else let fs' = (fs − busc_int_ent(comp, fs))
.7                                  ∪
.8                          atlza_int_ent(busc_int_ent(comp, fs))
.9                          let fs'' = (fs' − busc_int_sai(comp, fs'))
.10                                 ∪
.11                         atlza_int_sai(busc_int_sai(comp, fs')) in
.12                         elab_Execute(comp, mk-Est(fs'', _))
```

```
.0      mk-Compte(a, ORJOIN, _, _) →
.1          let mk-Est(fs, _) = estado
.2          f ∈ procura_int_entrada(comp, fs)
.3          let mk-Fila(o, d, ci, cf) = f in
.4          if ci = 0
.5                  then mk-Est(fs, _)
.6                  else let fs' = (fs − busc_int_ent(comp, fs))
.7                                  ∪
.8                          atlza_int_ent(busc_int_ent_rand(comp, fs))
.9                          let fs'' = (fs' − busc_int_sai(comp, fs'))
.10                                 ∪
.11                         atlza_int_sai(busc_int_sai(comp, fs')) in
.12                         elab_Execute(comp, mk-Est(fs''))
```

7.8.3 Funções auxiliares da função execute

A função semântica *execute* foi definida fazendo referencias a outras funções. Essa disciplina de refinamento sucessivo é uma das características mais fortes em uma especificação VDM. Assim foram definidas como funções semânticas auxiliares a *execute*, as funções: PROC_PROCESSO, PROC_INT_ENTRADA (procura interface de entrada), ATLZA_INT_ENT (atualiza interface de entrada), BUSC_INT_SAIDA (buscar interface de saída) e ATUALIZA_INT_SAIDA.

Função procura processo

A função *proc_processo* tem como partida o domínio de componente e um conjunto de processos. Ela procura um determinado processo no diagrama SADT retornando esse processo.

76.0	$Elab_Proc_processo(c, p) \triangleq$
.1	let $p' \in p$
.2	let mk-$Componente(a, _, _, _) = c$
.3	let mk-$Processo(a', tmin, tmax, ts, s) = p'$ in
.4	if $a = a'$
.5	then mk-$Processo(a, tmin, tmax, ts, s)$
.6	else $Elab_Proc_processo(c, p - \{$mk-$Processo(a', tmin, tmax, ts, s\})$

Função procura interface de entrada

Essa função verifica se todas as filas do sistema chegam a um determinado processo, como interfaces de entrada e de controle.

A função verifica se o conjunto de filas do sistema é vazio, caso positivo retorna esse conjunto. Caso negativo constrói um componente, constrói uma fila e faz um componente destino d. No teste se a = a' a fila f' é unida com a chamada recursiva da função retirando f' do conjunto, caso contrário, faz uma chamada recursiva da função como no caso anterior com o componente e o conjunto de filas retiranto f'.

77.0	$Elab_Proc_Int_Entrada(comp, fs) \triangleq$
.1	if $fs = \{\ \}$
.2	then fs
.3	else mk-$Componente(a, _, _) = comp$
.4	let $f' \in f$
.5	let mk-$Fila(o, d, c) = f'$
.6	let mk-$Componente(a', _, _) = d$ in
.7	if $a = a'$
.8	then $\{f'\} \bigcup Proc_Int_Entrada(comp, fs - \{f'\})$
.9	else $Elab_Proc_Int_Entrada(comp, fs - \{f'\})$

Função atualiza interface de entrada

Essa função aplica-se da seguinte forma: para cada fila que chaga ao processo (atividade) na forma de entrada(s) e controle(s), é retirado um "token", sendo verificado antes, se o comprimento de cada fila é de pelo menos a unidade. O domínio de partida e o contradomínio é um conjunto de filas.

```
78.0  Elab_Atlza_Int_Entrada(fs) ≜
  .1        if fs = { }
  .2        then fs
  .3        else let f' ∈ f
  .4             let mk-Fila(o, d, ci, cf) = f' in
  .5             {mk-Fila(o, d, ci − 1, cf)}
  .6                  ∪
  .7             Elab_Atlza_Int_Entrada(fs − {f'})
```

Função Busca interface de saída

A função *Busca_int_saida* parte do domínio de componente e um conjunto de filas. Ela procura o conjunto de interfaces de saída a uma determinada atividade. O retorno da função é um conjunto de filas.

```
79.0  Elab_Busca_Int_Saida(comp, fs) ≜
  .1        if fs = { }
  .2        then fs
  .3        else let mk-Componente(a, _, _) = comp
  .4             let f' ∈ fs
  .5             let mk-Fila(o, d, ci, cf) = f'
  .6             let mk-Componente(a', _, _) = d in
  .7             if a = a'
  .8             then {f'} ∪ Elab_Busca_Int_Saida(comp, fs − {f'})
  .9             else Elab_Busca_Int_Saida(comp, fs − {f'})
```

Função atualiza interfaces de saída

Esta função tem como partida um conjunto de filas e como chegada também um conjunto de filas. Ela acrescenta um "token" em cada fluxo de saída de uma determinada atividade.

```
80.0  Elab_Atualiza_Int_Saida(fs) ≜
  .1        if fs = { }
  .2        then fs
  .3        else let f' ∈ fs
  .4              let mk-Fila(o, d, ci, cf) = f' in
  .5              {mk-Fila(o, d, ci − 1, cf + 1)} ∪
  .6              Elab_Atualiza_Int_Saida(fs − {f'})
```

Função busca interface de saída randomicamente

A função parte de um componente e de um conjunto de filas. Ela é utilizada na semântica do conectivo ORBRANCH. Objetiva escolher randomicamente um dos fluxos de saída.

```
88.0  Elab_Busca_Int_Saida_Rand(comp, fs) ≜
  .1          if fs = { }
  .2          then fs
  .3          else let mk-Componente(a, OrBranch, _) = comp
  .4               let f' ∈ Randomico(fs)
  .5               let mk-Fila(o, d, ci, cf) = f'
  .6               let mk-Componente(a', OrBranch, _) = o in
  .7               if a = a'
  .8               then {f'} ∪ Busca_Int_Saida_Rand(comp, fs − {f'})
  .9               else Elab_Busca_Int_Saida_Rand(comp, fs − {f'})
```

Função busca interface de entrada randomicamente

A função parte de componente e um conjunto de filas e chega a um conjunto de filas. Ela é auxiliar na definição semântica do conectivo Or-Join. O objetivo é a escolha randômica de um dos fluxos de entrada do conectivo.

```
87.0  Elab_Busc_Int_Ent_Rand(comp, fs) ≜
  .1          if fs = { }
  .2          then fs
  .3          else mk-Componente(a, OrJoin, _) = comp
  .4               let f' ∈ Randomico(fs)
  .5               let mk-Fila(o, d, ci, cf) = f'
  .6               let mk-Componente(a', OrJoin, _) = d in
  .7          if a = a'
  .8          then {f'} ∪ Busc_Int_Ent_Rand(comp, fs − {f'})
  .9          else Elab_Busc_Int_Ent_Rand(comp, fs − {f'})
```

7.8.4 Função semântica simular

Essa função, na especificação, está um nível acima, em relação à *execute*. Esta é a função de simulação da execução de uma especificação SADT, considerando que nos passos anteriores, já foi definido a semântica (comportamento) de cada componente válido para o método.

```
81.0  Elab_Simular(sistema, estado) ≜
  .1        let mk-Estado(fs, ps) = estado
  .2        let fs' = atualiza_filas(fs)({ })
  .3        let mk-Sistema(rs) = sistema
  .4        let prod = produtores(rs)
  .5        let h = habilitados(prod, fs') in
  .6              if h = 0
  .7                   then mk-Estado(fs', ps)
  .8                   else let e = processar(h, mk-Estado(fs', ps)) in
  .9                        Elab_Simular(sistema, e)
```

7.8.5 Funções auxiliares da função simular

Para a realização da função *simular*, subdividiu-se a sua definição também em um corpo de funções auxiliares. Essas funções todas referenciadas na especificação da função *simular* são as seguintes: função ATUALIZA_FILAS, função PRODUTORES, função HABILITADOS, função TESTE e função PROCESSAR.

Função atualizar filas

A função *atualiza_filas* tem como domínio de partida um conjunto de filas para um conjunto de filas. O contradomínio é representado também como um conjunto de filas. Ela faz a atualização de do conjunto de filas antes e depois de cada ciclo de simulação.

```
82.0  Elab_Atualiza_Filas(fs1)(fs2) ≜
  .1      if fs1 = 0
  .2        then fs2
  .3        else let f ∈ f1
  .4             let fs1' = fs1 − {f}
  .5             let mk-Fila(o, d, ci, cf) = f in
  .6             Elab_Atualiza_Filas(fs1', fs2 ∪ {mk-Fila(o, d, ci + cf, 0)})
```

Função produtor

Essa função tem como ponto de partida o conjunto de uma relação e os produtores de informação no sistema. O ponto de chegada é representado pelo conjunto de componentes. É verificado se essa relação de partida é vazia, caso verdade, é retornada a informação de que o sistema não possui produtores, caso contrário fica estabelecido que produtores e consumidores pertencem a relação *rs* e é feita uma chamada recursivamente da função *Produtores* retirando desta a tupla {(p,c)} unida a {p}.

```
83.0  Elab_Produtores(rs, ps)(cs)≜
  .1      if rs = 0
  .2        then { }
  .3        else let (p, c) ∈ rs in
  .4             Elab_Produtores(rs − {(p, c)}) ∪ {p}
```

Função habilitados

A função tem como domínio de partida um conjunto de componentes e um conjunto de filas. O contra-domínio da função é representado por um conjunto de filas. Essa função semântica verifica quais atividades em um diagrama encontram-se habilitadas para disparo. É verificado se em cada atividade existe pelo menos um "token" para cada fluxo de entrada e de controle.

```
84.0  Elab_Habilitados(cs, fs)≜
  .1      if cs = { }
  .2        then { }
  .3        else let c ∈ cs
  .4             let i = procura_int_entrada(c, fs) in
  .5             if teste(i)
  .6                then Elab_Habilitados(cs − {c}, fs) ∪ {c}
  .7                else Elab_Habilitados(cs − {c}, fs)
```

Função teste

A função semântica *teste*, parte do domínio de um conjunto de filas, e chega a um valor booleano. Ela é função auxiliar de uma função auxiliar. Sua função é verificar se o comprimento inicial de uma fila é vazio; caso se confirme, a função verifica que processos estão habilitados e desconsidera essa interface para o componente em análise.

```
85.0  Elab_Teste(fs) ≜
  .1        if fs = { }
  .2           then True
  .3           else let f ∈ fs
  .4                let mk-Fila(o, d, ci, cf) in
  .5                if ci = 0
  .6                   then False
  .7                   else Elab_Teste(fs − {f}) ∧ True
```

Função processar

Esta função parte de um conjunto de componentes válidos do SADT e o estado, retorna um estado. Os componentes do SADT conectivos ou atividade são processados por essa função. A elaboração semântica é a seguinte: se o componente for nulo retorna o estado, caso contrário faz "c" pertencer ao conjunto de componentes e, chama-se recursivamente a função retirando o componente "c" fornecendo um novo estado através de chamada embutida da *execute*.

86.0	$Elab_Processar(cs, e) \triangleq$
.1	if $cs = 0$
.2	then e
.3	else let $c \in cs$
.4	let $Elab_Processar(cs - \{c\}, Elab_Execute(c, e))$

7.9 O refinamento da especificação em direção à implementação

A partir da especificação definida para a execução de uma especificação SADT por simulação, podem ser feitos refinamentos sucessivos, incluindo em cada etapa mais e mais detalhes de implementação, objetivando assim atingir completamente uma linguagem alvo de programação. A cada novo refinamento de uma especificação, deve haver uma equivalência entre os níveis de refinamento, n e n+1. Esse processo pode ser feito, seguindo as recomendações encontradas em [COH 86] e [JAC 85].

Em [BJO 82] apresenta-se um capítulo assinado por C.B. JONES, onde uma especificação VDM é apresentada em termos de pré e pós-condições, sendo então refinada sucessivamente até chegar a um código "Pascal-like". Nesse mesmo livro, em um capítulo o Prof. BJORNER afirma que o caminho de uma especificação VDM até a sua implementação ainda é uma área para muita pesquisa.

Nesta dissertação o autor entende que um meio termo entre as opiniões emitidas pelos representantes das escolas inglesa e dinamarquesa, pode ser adotado, como uma proposição de implementação. É possivel a implementação dessa simulação, mas não é uma tarefa simples. Caminhar do nível abstrato aqui apresentado, até o código de máquina, envolve tempo e vários refinamentos, provas de obrigação e um processo de reificação sólido. O trabalho de [JON 80] e [BJO 82] são boas referências para isso.

7.10 Considerações sobre a especificação formal apresentada

Uma das principais características de métodos formais como o VDM, para desenvolvimento de software, é a maneira clara e precisa de descrever as várias entidades no processo de criação do sistema, expondo de forma correta e objetiva todos os inter-relacionamentos entre as diversas entidades constituintes do sistema [COH 86] e [GEH 86]. O VDM foi escolhido nesta dissertação como método de especificação, por oferecer um ambiente apropriado para tratar o processo de desenvolvimento em modo disciplinado, confiável e seguro. Por outro lado, o VDM é um método orientado a modelos [RIB 89]. Em métodos dessa categoria, procura-se construir, a partir de objetos já conhecidos, um modelo que reflita as propriedades do sistema em discussão. O VDM é um autêntico método orientado à modelos, cujo embasamento está fundamentado no conceito de denotações [MAR 88]. O desenvolvimento em VDM é feito de forma sistemática, em uma disciplina típica dos métodos tradicionais da engenharia, e o problema de simulação aqui especificado exige esse tratamento. Na especificação apresentada, centralizou-se atenção nos conceitos inerentes de uma simulação de execução de especificações SADT, os conceitos subjacentes desse problema são muito mais importantes do que aspectos de uma sintaxe extremamente detalhada.

O VDM já foi aplicado a muitos problemas em nível industrial em centros europeus de processamento de dados. Na especificação aqui elaborada, o uso do VDM apresentou a vantagem de se permitir construir uma especificação não-ambígua, possível de ser lida e entendida igualmente, por quaisquer pessoas conhecedoras de VDM. Como conseqüência disso, descobriram-se erros, que via um um método não formal só seriam descobertos em fases bem posteriores no ciclo de desenvolvimento. Assim a especificação formal aqui apresentada, serve de base segura para o desenvolvimento e implementação correta dos programas (programa que atende a sua especificação) que permitam "executar" via simulação sistemas especificados em SADT.

8 CONCLUSÕES E EXTENSÕES

A engenharia de software é uma uma área muito nova, tanto em contexto acadêmico quanto industrial. A grande maioria dos métodos de apoio ao desenvolvimento de software surgiram nos anos 70 e 80. A quantidade de problemas em aberto é grande, a quantidade de proposições de solução também é grande.

O universo de aplicações de computadores é tão vasto, que dificilmente surgirá um método ou ferramenta capaz de atender a tudo. Para um único problema isolado, é difícil encontrar uma metodologia que consiga fazer uma varredura em todo o ciclo de vida, convencional ou não.

As ferramentas de desenvolvimento para serem bem utilizadas não dispensam talento, disciplina, objetividade e método de reflexão por parte de seu usuário [STA 83].

O trabalho desenvolvido nesta dissertação fez uma exploração ampla do método SADT, destacando suas origens, o modelo de desenvolvimento, interfaceamento com outros métodos de natureza estruturada, vantagens e limitações. Esse estudo que foi apresentado no Capítulo 2, preenche uma importante lacuna hoje existente, em termos de literatura disponível em língua portuguesa, apresentando conceitos de maneira unificada, que estão distribuídos em várias publicações internacionais. Realça-se esse fato, porque até hoje, devido a inexistência de livros acerca do SADT, a maioria das pessoas, incluindo-se nesse conjunto analistas e projetistas de software, o desconhece. Necessariamente, o capítulo não teve maiores preocupações de caráter didático, mas fornece bastante informação para quem não

conhece o SADT.

A maioria dos métodos de desenvolvimento de software orientados a fluxo de dados é na realidade oriundo da teoria das redes, e esse é um fato que quando é desconhecido por um projetista ou analista, ele deixa de usar o real potencial imbutido nesses métodos para a descrição de sistemas. O SADT tem uma íntima relação com teoria de redes, por esse motivo, o Capítulo 3, apresenta a inter-relação do SADT aos sistemas de fluxo de dados, caracterizando que ao construir uma árvore de diagramas, forma-se uma rede, sendo as ligações estabelecidas via código ICOM. Assim foi possível adicionar ao SADT, toda uma visão conceitual básica de redes, grafos, "dataflow machines", regras para disparo e paralelismo. Isso constitui um ponto muito importante nesta dissertação, porque desta forma, o método adquire condições para ser tratado de uma maneira mais exata e consequentemente formal. Os Capítulos 4 e 5, surgem em decorrência natural das inter-relações estudadas no capítulo 3. A abordagem operacional para conectivos e atividades, bem como a proposição de execução de especificações por simulação, é na verdade possível porque as relações do SADT com o mundo das redes são bem mais profunda do que se imagina.

Nesta dissertação, parte-se do estudo de um método dito semi-formal, e o desenvolvimento deste trabalho, acrescentou ao método, um aspecto formal, visando introduzir uma proposição de experimentação do modelo de sistema construído em SADT.

A implementação de uma ferramenta para o SADT foi realizada no ambiente PROSOFT. Na versão atual deste, o primeiro passo para se construir uma ferramenta é definir uma classe que descreva exatamente a linguagem do método a ser implementado. E essa não é uma tarefa fácil. O SADT é um método complexo, tem um conjunto muito grande de construções gráficas e, projetar uma classe ou domínio semântico, que faça uma correta abstração, prevendo todos os objetos possíveis de serem instanciados no ambiente, foi uma árdua tarefa. Esse é o motivo pelo qual se fez a apresentação da classe e da ferramenta gráfica com suas operações, como um capítulo dentro desta dissertação. Alie-se a isso, o fato de que, todas as observações e análises acerca do inter-realacionamento do método com os sistemas de fluxo de dados, como estudados nos capítulos 3,4 e 5, encontram-se plenamente representados nessa classe. A ferramenta foi concluída, consolidando completamente o projeto da

classe, e abrindo a possibilidade de que se implemente no futuro a experimentação ou prototipagem do modelo, exatamente como discutido nos capítulos anteriores ao 6.

O capítulo 7 apresentou a especificação formal da semântica e da simulação do SADT. Aqui se consolida todo um estudo, que possibilita a execução de especificações, porque neste trabalho, indica-se o caminho para tal. Não é um caminho fácil mas possível, mesmo sendo reconhecido que o caminho entre uma especificação formal e sua implementação ainda é uma área que precisa de muita pesquisa.

O ambiente PROSOFT no desenvolvimento da ferramenta aqui apresentada, possibilitou rapidez de desenvolvimento, reusabilidade de software e criação de uma interface gráfica amigável. Sente-se a necessidade de que o ambiente precisa oferecer ao projetista menus hierárquicos. O ATO SADT apresenta um menu relativamente grande. Mas em sintese, o paradigma do ambiente foi muito vantajoso para o desenvolvimento da ferramenta apresentada.

O método formal VDM foi usado para descrever a simulação, porque esse método fornece uma base matemática que falta à engenharia de software. Esse uso, possibilita a detecção de erros nas fases iniciais do desenvolvimento, com isso é possível a redução de custos.

De um ponto de vista teórico, a principal contribuição desta dissertação é a definição da semântica formal para o método, o que possibilitou a especificação da execução por simulação.

As extensões que podem ser feitas a este trabalho em nível de implementação são as seguintes:

- Implementar uma função de roteamento para permitir as ligações automáticas entre os elementos do diagrama. Em [LEA 88] descreve-se em linhas gerais um algoritmo para tal.

- Implementar a simulação do modelo que está especificada formalmente nesta dissertação. Em [BER 75],[BIR 85],[DAH 87],[DEG 90],[GOR 69],[KIV 67] e [PRI 86] encontram-se estudos bem profundos sobre simulação.

- Implementar uma interface de animação da simulação. Em [KRA 88] apresenta-se um importante estudo realizado na Universidade de Sussex, utilizando o método CORE [MUL 79] na animação de especificações de requisitos. E CORE é de certa forma um método "SADT-like".

- A possibilidade de tornar o uso do SADT distribuído, ou seja várias pessoas construindo a solução de um mesmo problema via diagramas em vários terminais. É discutida em [LIS 87] essa possibilidade. O PROSOFT pode suportar isso quando implementar processamento distribuído.

- A implementação de um dicionário de dados para a ferramenta é uma outra possibilidade. Esse dicionário poderá apresentar as informações em modo gráfico e textual. Uma atividade mecanizada poderia ter o seu funcionamento descrito, via diagramas NSD por exemplo. Os fluxos de dados poderiam ter sua representação estrutural via os objetos fornecidos pelo Ato Jackson. [NUN 89] e [NUN 90] apresenta um panorama geral do PROSOFT e de seus atos.

Uma proposição de extensões a este trabalho, em plano teórico, é fazer um estudo, para se construir uma ferramenta que, a partir da especificação de um sistema em diagramas SADT, possa-se gerar uma corrente especificação algébrica LARCH, e essa especificação ser verificada quanto a completeza e consistência.

[BIR 85] comenta uma remota possibilidade de que a partir de um conjunto de diagramações SADT de um sistema, se derive a especificação formal em VDM do sistema. Mas não são oferecidos maiores detalhes desse caminho.

A conclusão desta dissertação é que foi feito um estudo profundo do método SADT, e verificado o relacionamento do método com os sistemas de fluxo de dados. A semântica dos principais construtos, foi definida inicialmente, em modo operacional.

A classe especificada para a implementação da ferramenta diagramática, importou tipos do VDM implementados no PROSOFT. Esse editor implementa mais de 40 operações. Em [ROS 85] afirma-se que nenhuma automação do SADT foi completamente bem sucedida. O conjunto de operações valida a classe projetada e a ferramenta valida o paradigma do PROSOFT.

A especificação formal em VDM, da semântica e da execução de sistemas descritos em SADT por simulação representa a contribuição maior da dissertação para o desenvolvimento da Engenharia de Software.

A especificação formal apresentada sintetiza todo um trabalho de investigação, acerca do método SADT. O uso do VDM para essa formalização mostrou-se adequado, mesmo considerando que são métodos com filosofias próprias.

Para a realização da especificação, progressivamento foi desenvolvido um estudo abrangente do método. Apartir desse estudo foi determinado, um conjunto de conceitos e relações, que abalizaram a construção do modelo formal para o SADT.

Partindo de uma análise da inter-relação do SADT com sistemas de fluxo de dados, foi possível verificar a estrutura de rede de um modelo em SADT. Essa constatação forneceu elementos para uma definição operacional dos construtos atividade e conectivos.

A proposição de simulação discreta de especificações SADT, surgiu naturalmente dentro do contexto no qual foi estudado o método.

A experiência acumulada no desenvolvimento do trabalho, possibilitou estruturar um conjunto de conhecimento, que possibilitou realizar a especificação formal em VDM da semântica e da execução de sistemas descritos em SADT.

A especificação formal apresentada nesta dissertação, encontra-se em seu primeiro nível de abstração, mas consolida todo um estudo, e alarga fronteiras, no sentido de contribiur para o desenvolvimento na área de engenharia de software.

BIBLIOGRAFIA

[AGE 83] AGERWALA T. Putting petri nets to work. In: *SOFTWARE design techniques.* 4. ed. New York, IEEE, 1983. p. 595-603 Tutorial.

[ANA 79] THE ANALYSIS of user needs. *EDP ANALYZER* , Vista, v. 17, n. 1, p. 1-6, Jan. 1979.

[AND 87] ANDREWS, D. Data reification and program decomposition. In: EUROPEAN SYMPOSIUM ON VDM-A FORMAL METHOD AT WORK, Mar. 23-26, 1987, Brussels *Proceedings...* Berlin, Springer-Verlag, 1987. p. 389-422. (Lecture Notes on Computer Science, v.252)

[AVG 87] AVGEROU, Chrisanthi. The Applicability of software engineering in information systems development. *Information & Management.* New York, v. 13, n. 3, p. 135-42, Oct. 1987.

[BAL 78] BALZER,R. et al. Informality in program specifications. *IEEE Transactions on Software Engineering.* New York, v. 4, n. 2, p. 94-103, Mar. 1978.

[BAR 84] BARSTON, D.R.; SHROBE, E.E. ; SANDEWALL E. (ed) *Interactive programming environments.* New York, McGraw-Hill, 1984.

[BAS 81] BASILI, Victor R. ; WEISS, David M. Evaluation of a software requirements document by analysis of change data. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 5. Apr. 1981. New York. *Proceedings...* IEEE, 1981. p. 387-93.

[BEL 83] BELL, Thomas E. et al. An Extendable approach to computer-aided software requirements engineering. In: *SOFTWARE design techniques.* New York, IEEE, 1983. p. 219-29. tutorial.

[BEL 86] BELKHOUCHE, B. ; URBAN, J.E. Direct implementation of abstract data types from abstract specifications. *IEEE Transactions on Software Engineering*, New York, v. 12, n. 5, p. 649-661, May 1986.

[BIE 79] BIEWALD, J.; GOEHNER, P.; LAUBER, R.; SCHELLING H. EPOS a specification and design technique for computer controlled real time automation systems. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 4., Sept. 17-19, 1979. Munich. *Proceedings...* New York, ACM/IEEE, 1979. p. 245-250.

[BER 75] BERNARD,P.Z. *Theory of modelling and simulation.* New York, John Wiley & Sons, 1975.

[BEL 77] BELL T.E.; BIXLER, D.C. ; DYER, M.E. An Extendable approach to computer-aided software requirements engineering. *IEEE Transactions on Software Engineering* , New York, v. 3, n. 1, p. 60-69, Jan. 1977.

[BIR 85] BIRREL, N.D.; OULD, M.A. *Apractical handbook for software development* . New York, Cambridge University Press, 1985.

[BJO 78] BJORNER, D. ; JONES C.B. (eds.) *The Vienna development method: the meta language.* Berlin, Springer-Verlag, 1978. (Lecture Notes in Computer Science, v. 61)

[BJO 81] BJORNER, D. The VDM principles of software specification and program design. In: INTERNATIONAL COLLOQUIUM, Apr. 19-25, 1981. Peniscola, Spain. *Proceedings...* Berlin, Springer-Verlag, 1981. p. 44-74. (Lecture Notes on Computer Science, v. 107)

[BJO 82] BJORNER, D. ; JONES, C.B. *Formal specification and software development.* Englewood Cliffs, Prentice-Hall, 1982.

[BJO 88] BJORNER, D. *Software architectures and programming systems design : the VDM approach.* Lyngby, Technical University of Denmark, 1988.

[BLA 83] BLANK, J.; KRIJER, M.J. (eds.) *Software engineering: methods and techniques.* New York, John Willey & Sons, 1983. chap. 3. p. 31-204.

[BOE 76] BOEHM, B.W. Software engineering. *IEEE Transactions on Computers*, New York, v. 25, n. 12, p. 1226-40, Dec. 1976.

[BOE 84] BOEHM, B.W. Verifying and validating software requirements and design specifications. *IEEE Software*, New York, v. 1, n. 1, p. 75-88, Jan. 1984.

[BOR 85] BORGIDA, A. et al. Knowledge representation as the basis for requirements specifications. *Computer* , New York, v. 18, n. 4, p. 82-90, Apr. 1985.

[BOY 80] BOYDSTON, Louis E. et al. Computer aided modeling of informations systems. In: *CONFERENCE COMPUTER SOFTWARE & APPLICATIONS* , Oct. 27-31, 1980. Chicago, *Proceedings...* New York, IEEE, 1980. p. 37-41.

[BRA 75] BRATMAN, H. ; COURT, T. The software factory. *Computer*, New York, v. 8, n. 5, p. 28-37, May. 1975.

[BUB 83] BUBENKO, Janis A. Informationm modeling in the context of system development. In: *SOFTWARE design techniques* . New York, IEEE, 1983. p. 156-171. tutorial.

[CAM 86] CAMERON, J.R. An overview of JSD. *IEEE Transactions on Engineering* , New York, v. 12, n. 2, p. 222-240, Feb. 1986.

[CER 86] CERI, S. *Requirements collection and analysis in information systems design.* Amsterdan, North-Holland, 1986, p. 205-214.

[CHA 80] CHAFIN, Roy L. The System analyst and software requirements specifications. In: CONFERENCE COMPUTER SOFTWARE &

APPLICATIONS, Oct. 27-31, 1980. Chicago, *Proceedings...* New York, IEEE, 1980. p. 254-258.

[CLE 86] CLEAVELAND, J.C. *An Introduction to data types.* Reading, Addison-Wesley, 1986.

[COH 84] COHEN, A. T. Data abstraction, data encapsulation and object oriented programming. *SIGPLAN Notices* , New York, v. 19, n. 1, p. 31-35, Jan. 1984.

[COH 86] COHEN, B. ; HARWOOD, W.T. ; JACKSON, M. I. *The Specification of complex systems.* Reading, Addison-Wesley, 1986.

[CON 80] CONN, Alex Paul. Maintenance: a key element in computer requirements specifications. In: CONFERENCE COMPUTER SOFTWARE & APPLICATIONS, Oct. 27-31, 1980, Chicago. *Proceedings...* New York, IEEE, 1980. p. 401-406.

[COX 84] COX, B. Message/Object programming: an evolutionary change in programming technology. *IEEE Software,* New York, v. 1, n. 1, p. 50-61, Jan. 1984.

[CRI 78] THE CRITICAL nature of requirements. In: *STRUCTURED analysis and design* . Maidenhead, Infotech International, 1978. p. 91-135.

[DAH 87] DAHLER, J. et al. A Graphical tool for the design and prototyping of distributed systems. *Soft. Eng. Notes,* New York, v. 12, n. 3, p. 24-36, July. 1987.

[DAV 83] DAVIS, Carl G. ; VICK, Charles R. The Software development system. In: *SOFTWARE design techniques* . New York, IEEE, 1983. p. 625-640. tutorial.

[DAV 88] DAVIS, Alan M. A comparison of techniques for the specification of external system behavior. *Communications of the ACM* , New York, v. 31, n. 9, p. 1098- 115, Sept. 1988.

[DEG 90] DEGL'INNOCENTI, M. et al. RFS: a formalism for executable requirement specifications. *IEEE Transactions on Software Engineering*, New York, v. 16, n. 11, p. 1253-1246, Nov. 1990.

[DeM 78] DeMARCO, T. *Structured analysis and systems specification* . New York, Yourdon Press, 1978.

[DIC 81] DICKOVER, M. E., McGOWN, C. L. ; ROSS, D. T. Software design using SADT. In: *SOFTWARE design strategies* . 2. ed. Berkshire, Pergamon/IEEE, 1981. p. 397-409. Tutorial

[DON 76] DONAHUE, J.E. Complementary definitions of programming language semantics. Berlin, Springer-Verlag, 1976. (Lecture Notes in Computer Science, v. 42)

[FAV 89] FAVERO, E.L. *Um Editor orientado por estrutura para linguagens diagramáticas.* Porto Alegre, CPGCC/UFRGS, 1989. Dissertação de Mestrado.

[FRE 79] FREEMAN, P. A Perspective on requirements analysis and specification. In: *STRUCTURED software development.* Maidenhead, Infotech International, 1979. p. 43-55.

[FRE 81] FREEMAN, P. Software design techniques: currente options. In: *SYSTEM design* . Maidenhead, Infotech International, 1981. p. 199-219.

[FRE 83] FREEMAN, P. Requirements analysis and specification: the first step. In: *SOFTWARE design techniques.* New York, IEEE, 1983. p. 79-113.

[GAN 79] GANE, C. ; SARSON, T. *Structured system analysis: tools and techniques* . Englewood Cliffs, Prentice-Hall, 1979.

[GAN 83] GANE, C. Data design in structured systems analysis. In: *SOFTWARE design techniques.* New York, IEEE, 1983. p. 115-32.

[GAN 88] GANE C. *Desenvolvimento rápido de sistemas.* Rio de Janeiro, LTC, 1988.

[GEH 86] GEHANI, N. ; McGETTRICK, A. D. *Software specification techniques* . Reading, Addison-Wesley P., 1986.

[GOR 69] GORDON, M.G. *System simulation.* Englewood Cliffs, Prentice-Hall, 1969.

[GRA 81] GRAPHICAL design language. In: *SYSTEM design.* Maidenhead, Infotech International, 1981. p. 107-123.

[GRE 83] GREENSPAN, S. J. et al. Capituring more world knowledge in the requirements specification. In: *SOFTWARE design techniques* . New York, IEEE, 1983. p. 231-239. tutorial.

[GRI 78] GRIFFITHS, S. N. Design methodologies-a comparison. In: *STRUCTURED analysis and design* . Maidenhead, Infotech International, 1978. p. 113-166.

[GUT 78] GUTAG, J.V. ; HOROWITZ, E. ; MUSSER, D.R. Abstract data types and software validation. *Communications of the ACM*, New York, v. 21, n. 12, p. 1048-1063, Dec. 1978.

[HAL 87] HALBERT, D. ; O'BRIEN, P. D. Using types and inheritance in object-oriented programming. *IEEE Software*, New York, v. 4, n. 5, p. 71-79, Sept. 1987.

[HAM 74] HAMILTON, M. ; ZELDIN, S. Higher order software techniques applied to a space shuttle prototype program. In: COLLOQUE SUR LA PROGRAMATION, 1. Apr. 9-11, 1974. Paris, *Proceedings...* Berlin, Springer-Verlag, 1974. p. 17-32. (Lecture Notes on Computer Science, v. 19)

[HEE 90] HEERJEE, K.B. et al. The Design, validation and evaluation of a software development environment. *Computers Education*, Oxford, v. 14, n. 3, p. 281-295, Mar. 1990.

[HEI 80] HEININGER, K. Specifying software requirements for complex systems: new techniques and their applications. *IEEE Transactions on Software Engineering*, New York, v. 6, n. 1, p. 189-199, Jan. 1980.

[HEI 83] HEITMEYER, C. L. ; McLEAN, J. D. Abstract requirements specification: a new approach and its application. *IEEE Transactions on Software Engineering*, New York, v. 9, n. 5, p. 580-589, Sept. 1983.

[HEN 80] HENINGER, K.L. Specifying software requirements for complex systems: new techniques and their application. *IEEE Transactions on Software Engineering* , New York, v. 6, n. 1, p. 189-200, Jan. 1980.

[HOR 86] HOROWITZ, E. ; WILLIAMSON, R. C. SODOS: a software documentation support environment-its use. *IEEE Transaction on Software Engineering*, New York, v. 12, n. 11, p. 1076-1087, Nov. 1986.

[JAC 85] JACKSON, M.I. Developing ADA programs using the Vienna Development Method (VDM). *Software Practice and Experience* , Sussex, v. 15, n. 3, p. 305-318, Mar. 1985.

[JAC 87] JACOBSON, I. Object oriented development in industrial environment. SIGPLAN Notices, New York, v. 22, n. 12, Dec. 1987. p. 183-190. Trabalho apresentado no OOPSLA'87 Object Oriented Programming Systems, Languages and Applications Conference, Oct. 1987, Orlando.

[JON 80] JONES, C.B. *Software development: a rigorous approach* . Englewood Cliffs, Prentice-Hall, 1980.

[JON 87] JONES,K.D. A formal semantics for a data flow machines using VDM. In: EUROPEAN SYMPOSIUM ON VDM-A FORMAL METHOD AT WORK, Mar. 23-26, 1987. Brussels, *Proceedings...* Berlin, Springer-Verlag, 1987. p. 321-355, (Lecture Notes on Computer Science, v. 252)

[JON 88] JONATHAN, Miguel. O Impacto das linguagens orientadas para objetos na engenharia de software. In: CONGRESSO NACIONAL DE INFORMÁTICA, 21, 22-26 ago. 1988. Rio de Janeiro, *Anais...* Rio de Janeiro, SUCESU, 1988. p. 237-392.

[KAT 89] KATAYAMA, T. A Hierarchical and functional approach to software process description. In: SOFTWARE PROCESS WORKSHOP.

, May. 11-13, 1988. Maretonhampstead, *Proceedings...* New York, ACM/IEEE, p. 87-92.

[KIV 67] KIVIAT, P.J. *Digital event simulation: modeling concepts.* California, Rand Corporation RM-5378-PQ, 1967.

[KOM 81] KOMODA, Norihisa et al. An Innovative approach to system requirements analysis by using structured modeling method. In: INTERNACIONAL CONFERENCE ON SOFTWARE ENGINEERING, 5., 1981. *Proceedings...* New York, IEEE, 1981. p. 305-313.

[KRA 88] KRAMER, J.; NG, K. Animation of requirements specifications. *Software Practice and Experience* , Sussex, v. 18, n. 8, p. 749-774, Aug. 1988.

[LEA 88] LEANDRO, M.A.C. *Ferramenta para especificação de sistemas baseada no método SADT* . Porto Alegre, CPGCC/UFRGS, 1988. Dissertação de Mestrado.

[LEI 83] LEITE, J.C.S.P. SADT datagrams, a powerfull tool for requirements analysis. In: CONGRESSO NACIONAL DE INFORMÁTICA, 16., Out. 1983. São Paulo, *Anais...* São Paulo, SUCESU, 1983. p. 307-326.

[LEI 87] LEITE, J.C.S.P. *A Survey on requirements analysis.* California, University of California at Irvine, 1987.

[LEI 89] LEITE,J.C.S.P. O Uso de pontos de vista na elicitação de requisitos. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE. 3., Jul. 25-27 out., 1989, Recife, *Anais...* Recife, SBC, 1989, p. 71-85.

[LIS 87] LISSANDRE, M.; VAULX, B. SPECIF-X: a tool for CASE. In: EUROPEAN SOFTWARE ENGINEERING CONFERENCE, Sept. 9-11, 1987. Strasbourg, *Proceedings...* Berlin, Springer-Verlag, 1987. p. 28-55. (Lecture Notes on Computer Science, v. 289)

[LOO 87] LOOMIS, M.E. ; SHAH, A.V. ; RUMBAUGH, J.E. An Object modeling technique for conceptual design. In: EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING ECOOP'87.

Jun. 15-17, 1987, Paris, *Proceedings...* Berlin, Springer-Verlag 1987. p. 192-202. (Lecture Notes on Computer Science, v. 276)

[LUD 86] LUDEWIG, J. Practical methods and tools for specification. In: ADVANCED COURSE ON NEW APPROACHES TO THEIR FORMAL DESCRIPTION AND DESIGN, Mar. 5-7, 1986. Zurich, *Embedded Systems.* Berlin, Springer-Verlag, 1986. p. 174-207. (Lecture Notes on Computer Science, v. 284)

[LUN 83] LUNDENBERG, M. An Approach for involving the users in the specification of informations systems. In: *SOFTWARE design techniques.* New York, IEEE, 1983. p. 113-155.

[LUQ 88] LUQI; BERZINS,V.; YEH, R.T. A prototyping language for real time software. *IEEE Transactions on Software Engineering*, New York, v. 14, n. 10, p. 1409-1423, Oct. 1988.

[MAC 87] MAC, M.A. Introduction to VDM tutorial. In: EUROPEAN SYMPOSIUM ON VDM-A FORMAL METHOD AT WORK, Mar. 23-26, 1987. Brussels, *Proceedings...* Berlin, Springer-Verlag, 1987. p. 356-361 (Lecture Notes on Computer Science, v. 252)

[MAC 87a] MAC, M.A. Specification by data types. In: EUROPEAN SYMPOSIUM ON VDM-A FORMAL METHOD AT WORK, Mar. 23-26, 1987. Brussels, *Proceedings...* Berlin, Springer-Verlag, 1987. p. 362-387, (Lecture Notes on Computer Science, v.252)

[MAN 87] MANNINO, P. V. A Presentation and comparison of four information systems development methodologies. *Software Eng. Notes*, New York, 12 , n. 2, p. 26-29, Apr. 1987.

[MAR 85] MARTIN, J. *System design from provably correct constructs.* Englewood Cliffs, Prentice-Hall, 1985.

[MAR 88] MARTINS R. ; MOURA, A. Desenvolvimento sistemático de programas corretos: a abordagem denotacional. Campinas, *IV Escola de Computação*, 1988.

[MAT 85] MATSUMOTO, Y. ; KAWAKITA, S. A Method to bridge discontinuity requirements specification and design. In: CONFERENCE COMPUTER SOFTWARE & APPLICATIONS, Oct. 27-31, 1980. Chicago, *Proceedings ...* New York, IEEE, 1980. p. 259-66.

[MAT 87] MATHUR, R.N. Methodology for business system development. *IEEE Transactions on Software Engineering* , New York, v. 13, n. 5, p. 593-601, May 1987.

[MAT 87a] MATSUMURA, K. ; MIZUTAMI, H. ; ARAI, M. An Application of structural modeling to software requirements analysis and design. *IEEE Transaction on Software Engineering* , New York, v. 13, n. 4, p. 461-471, Apr. 1987.

[MEN 88] MENDES, S.B.T. ; AGUIAR, T.C. *Métodos para especificação de sistemas.* Curitiba, Kapeluz, 1988.

[MEN 89] MENASCÉ, D. ; FONSECA, N.L.S. Redes de petri estocásticas. Rio de Janeiro, Scientific Center IBM, 1989. *Technical Report*

[MER 87] MERLIN, J.A.; FARBER, D.J. Recoverability of communication protocols-implication of a theorical study. *IEEE Transaction on Communication*, New York, v. 24, n. 3, p. 1036-1043, Sept. 1987.

[MEY 85] MEYER, B. On Formalism in specifications. *IEEE Software*, New York, v. 2, n. 1, p. 6-26, Jan. 1985.

[MUL 79] MULLERY, G.P. CORE-a method for controlled requirement specification. In: INTERNATIONAL COFERENCE ON SOFTWARE ENGINEERING, 4., , Sep. 17-19, 1979. Munich, *Proceedings...* New York, ACM/IEEE, 1979 p. 126-135.

[NAK 80] NAKAO, Kazuo et al. A Structural approach to system requirement analysis of information systems. In: CONFERENCE COMPUTER SOFTWARE & APPLICATIONS, Oct. 27-31, 1980. Chicago, *Proceedings...* New York, IEEE, 1980, p. 207-213.

[NET 88] NETO A.F. ; FURLAN, J.D. ; HIGA, W. *Engenharia da informação: metodologia, técnicas e ferramentas.* Rio de Janeiro, McGraw-Hill, 1988.

[NEU 82] NEUHOLD, E.J. Development methodologies for event and message based application systems. In: CONFERENCE ON OPERATING SYSTEMS, Jan. 23-27, 1982. Visegrad, *Proceedings*. Berlin, Springer-Verlag, 1983. p. 28-55. (Lecture Notes on Computer Science, v. 152)

[NOG 88] NOGUEIRA, D. L. *Ferramentas automatizadas para o projeto estruturado*. Rio de Janeiro, COPPE/UFRJ, 1988. Dissertação de mestrado.

[NUN 85] NUNES, Daltro J. *Ein Verfahren zur Rechnerunterstützten Programnkonstruktion.* Stuttgart, Institut fur Informatik der Universitat Stuttgart, 1985. Tese de Doutorado.

[NUN 86] NUNES, Daltro J. *Desenvolvimento de um ambiente de trabalho para construção de programas*. Porto Alegre, DI/UFRGS, 1986.

[NUN 86a] NUNES, Daltro J. *Aplicação de técnicas de comunicação homem-máquina em ambientes de trabalho para construção de programas.* Porto Alegre, PGCC da UFRGS, 1986.

[NUN 87] NUNES, Daltro J. Requisitos de ambientes de engenharia de software. In: ENCONTRO IBM DE CIÊNCIA E TECNOLOGIA, 1., Nov. 23-25, 1987, Rio de Janeiro, *Anais...* Rio de Janeiro, IBM, 1987. p. 497-508.

[NUN 87a] NUNES, Daltro J. Um Ambiente computacional para construção de software. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 7., jul. 11-19, 1987. Salvador, *Anais...* Salvador, SBC, 1987. p. 497-508.

[NUN 89] NUNES, Daltro J. *PROSOFT-Um Ambiente de desenvolvimento de software extendível.* Porto Alegre, DI/UFRGS, 1989.

[NUN 90] NUNES, Daltro J. Um Ambiente de desenvolvimento de software orientado a modelos-PROSOFT. In: SIMPÓSIO BRASILERIO DE ENGENHARIA DE SOFTWARE, 4., 24-26 Out. 1990, Águas de São Pedro, *Anais...* São Paulo, SBC, 1990, p. 277-280.

[PON 87] PONCET,F. SADL: A Software development environment for software specification, design and programming. In: EUROPEAN SOFTWARE ENGINEERING CONFERENCE, Sept. 9-11, 1987. Strasbourg, *Proceedings...* Berlin, Springer-Verlag, 1987, p. 28-55. (Lecture Notes on Computer Science, v. 289)

[PRE 87] PRESSMAN, R.S. *Software engineering: a practioner's approach.* New York, McGraw-Hill, 1987.

[PRI 84] PRICE, Ana Maria A. *SODA-An Interactive design tool.* Sussex, University of Sussex, 1984. Tese de Doutorado.

[PRI 86] PRITSKER, A.A.B. *Introduction to simulation and SLAM II.* New York, John Wiley & Sons, 1986.

[RAM 78] RAMAMOORTHY, C.V. ; SO, H. H. Software requirements and specifications : status and perspectives. In: IEEE COMPUTER SOCIETY INTERNATIONAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE, 2., Nov. 13-16 1978, Chicago, *Software Methodology: tutorial* . Long Beach, IEEE, 1978. p. 43-163.

[RAM 80] RAMAMOORTHY, C. V.; GARY, J.Ho Performance evaluation of asynchronous concurrent systems using petri nets. *IEEE Transactions on Software Engineering,* New York, v. 6, n. 5, p. 440-449, Sept. 1980.

[RAM 86] RAMAMOORTHY, C.V. et al. Programming in the large. *IEEE Transactions on Software Engineering,* New York, v. 12, n. 7, p.768-83, July 1986.

[RAM 87] RAMOS, Loreta S.O. *Ferramenta para especificação de programas baseada na técnica de Nassi-Shneiderman.* Porto Alegre, CPGCC/UFRGS, 1987. Dissertação de Mestrado.

[REI 82] REISIG, W. *Petri nets-an introduction* . Berlin, Springer-Verlag, 1982.

[RIB 88] RIBEIRO, Leila *O método VDM e sua aplicação na especificação formal de um programa de matrícula.* Porto Alegre, DI/UFRGS, 1988. Trabalho de Diplomação.

[RIB 89] RIBEIRO, Adagenor L. ; EDELWEISS, Nina. *Especificações formais em OBJ.* Porto Alegre, CPGCC da UFRGS, 1989. RP 113

[RIB 89a] RIBEIRO, Adagenor L. ; LACOMBE, Jean C. *Uma Proposta de notação diagramática para análise de requisitos em sistemas de informação.* Porto Alegre, CPGCC da UFRGS, 1989.

[RIB 89b] RIBEIRO, Leila ; NUNES, Daltro J. *VDM um método formal para desenvolvimento de software.* Porto Alegre, DI/UFRGS, 1989.

[RIB 90] RIBEIRO, Adagenor L. *Uma Avaliação dos métodos de definição de requisitos de engenharia de software.* Porto Alegre, CPGCC da UFRGS, 1989. TI 172

[RIB 90a] RIBEIRO, Leila *Estudo comparativo de abordagens para especificação de software.* Porto Alegre, CPGCC da UFRGS, 1990.

[ROM 85] ROMAN, G. A Taxonomia of current issues in requirements engineering. *Computer* , New York, v. 18, n. 4, p. 14-23, Apr. 1985.

[ROS 83] ROSS, D. T. ; SCHOMAN, K. E. Jr. Structured analysis for requirements definition. In: *SOFTWARE design techniques* . New York, IEEE 1983. p. 86-95.

[ROS 83a] ROSS, D. T. Structured analysis (SA): a language for communicating Ideas. In: *SOFTWARE design techniques.* New York, IEEE, 1983. p. 96-114.

[ROS 85] ROSS, D.T. Applications and extensions of SADT. *Computer* , Los Angeles, 18 , n. 4, p.25-35, Apr. 1985.

[ROT 88] ROTENBERG, H. B. ; STAA, A. V. Adequando metodologias de análise estruturada a orientação à objetos. In: CONGRESSO NACIONAL DE INFORMÁTICA, 21. 22-26 ago. 1988. Rio de Janeiro, *Anais...* Rio de Janeiro, SUCESU, 1988. p. 445-455.

[ROU 82] ROUBINE, O. Languages for specification and design: emerging for wide use ?. In: *PROGRAMMING technology.* Berkshire, Pergamon Infotech, 1982. p. 288-297. State of the Art Report. v. 10, n. 02

[SOA 90] SOARES, L.F.G. *Modelagem e simulação discreta de sistemas.* São Paulo, VII ESCOLA DE COMPUTAÇÃO, IME/USP, 1990.

[SHA 75] SHANNON, R.E. *System simulation-the art and science.* Englewood Cliffs, Prentice-Hall,1975.

[SEI 87] SEIDEWITZ, (Ed.) Object oriented programming in SMALLTALK and ADA. SIGPLAN Notices, New York, v. 22, n. 12, Dec. 1987. p. 202-213. Trabalho apresentado no OOPSLA'87 Object Oriented Programming Systems, Languages and Applications Conference, Oct. 1987, Orlando.

[SHO 83] SHOOMAN, M.L. *Software engineering.* Tokio, McGraw-Hill, 1983.

[SIE 85] SIEVERT, G.E. ; MIZELL, T. Specification-based software engineering with TAGS. *Computer,* Los Alamitos, v. 18, n. 4, p. 56-65, Apr. 1985.

[SQU 82] SQUIRES, S.L.; BRANSTAD, M.; ZELKOWITX Special issue on rapid prototyping. *Software Engineering Notes,* New York, v. 7, n. 5, p. 88-92, Dec. 1982.

[STA 83] STAA, A.V. *Engenharia de programas.* Rio de Janeiro, LTC, 1989.

[STR 78] STRUCTURED Analysis and design technique. In: *STRUCTURED analysis and design.* Berkshire, Pergamon Infotech, 1987. p. 151-169.

[TAK 89] TAKAHASHI, T. *O Paradigma de objetos: introdução e tendências.* Uberlandia, SBC/UFV, 1989. Mini curso apresentado na VIII Jornada de Atualização em Informática, Uberlandia 16-21 jul. 1989.

[TAT 85] TATE, G. ; DOCKER, T. W. G. Rapid prototyping system basead on data flow principles. *Software Eng. Notes* , New York, v. 10, n. 2, p. 28-34, Apr. 1985.

[TEI 83] TEICHROEW, D. ; HERSHEY, E. A. PSL/PSA: a computer-aided techniques for structured documentation and analysis on information

processing systems. In: *SOFTWARE Design Techniques* . New York, IEEE, 1983. p. 211-17. tutorial

[TER 86] TERWILLIGER, R.B. ; CAMPBELL, R. H. *PLEASE: executable specifications for incremental software development.* Urbana, University of Illinois. 1986. 24 p. Report No. UIUCDCS-R-86-1295

[THO 78] THOMAS, M. Funcional decomposition: SADT. In: *STRUCTURED analysis and design.* Berkshire, Pergamon Infotech 1978, v. 2., p. 337-354. State of the Art Report, 41.

[THO 79] THOMAS, M. O. *GPSS-simulation made simple.* New York, John Wiley & Sons, 1979.

[TOL 89] TOLEDO, C. M. T. *ANA-RE um método para análise e especificação de requisitos.* Campinas, UNICAMP, 1989. Dissertação de Mestrado.

[TRA 80] TRATTNIG, W. ; KERNER, H. EDDA, A very-hight-level programming and specification language in the style of SADT. In: COMPUTER SOFTWARE & APPLICATIONS-CONFERENCE, 4., Oct. 27-31, 1980, Chicago, *Proceedings...* New York, IEEE, 1980. p.436-443.

[VEL 86] VELOSO, P.A.S. *Tipos(abstratos) de dados: programação, especificação e implementação.* Belo Horizonte, Formato, 1986.

[WAR 73] WARNIER, J. D. *Les Procédures de traitment et leurs donnés.* Paris, Les Éditions D'Organisation, 1979.

[WAR 79] WARNIER, J. D. *Guide des utilisateurs du systeme Informatique.* Paris, Éditions D'Organisation, 1979.

[WAS 81] WASSERMAN, Anthony I. User software engineering and the design of interactive systems. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 5., 1981. *Proceedings...* New York, IEEE, 1981. p. 314-23.

[WAS 83] WASSERMAN, Anthony I. Information system design methodology. In: *SOFTWARE design techniques.* New York, IEEE, 1983. p. 43-61.

[YAD 88] YADAV, S. B. et al. Comparison of analysis techniques for information requirement determination. *Communication of the ACM* , New York, v. 31, n. 9, p. 1090-1097, Sept. 1988.

[YEH 79] YEH, R. T.; ARAYA, A.; MITTERMEIR, R. ; MAO, W. ; EVANS, F. Software requirement engineering: a perspective. In: *STRUCTURED software development.* Berkshire, Pergamon Infotech, 1979. p. 313-341.

[YEH 80] YEH, R. T. ; ZAVE, P. Specifying software requirement. *Proceedings of the IEEE,* New York, v. 68, n. 9, p. 1077-1085, Sept. 1980.

[YEH 84] YEH, R. et al Software requirements: new directions and perspectives. In: *HANDBOOK* of software engineering. New York, Von Nostrand Reinhold, 1984. p. 519-543.

[ZAV 82] ZAVE P. An Operational approach to requirements for embedded systems. *IEEE Transactions on Software Engineering,* New York, v. 8, n. 3, p. 250-269, Mar. 1982.

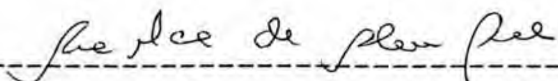
[ZEI 76] ZEIGLER, B.P. *Theory of modeling and simulation.* New York, John Wiley & Sons, 1976.

UFRGS
INSTITUTO DE INFORMÁTICA
BIBLIOTECA

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
CURSO DE PóS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

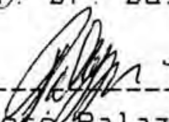
"Semânticas para o método SADT, ferramenta de definição de requisitos e especificação formal em VDM da executabilidade por simulação".

Dissertação apresentada aos Srs.

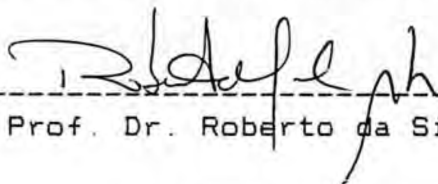


Profa. Dra. Ana Maria de Alencar Price

Prof. Dr. Daltro José Nunes



Prof. Dr. José Palazzo Moreira de Oliveira

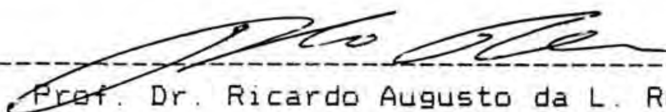


Prof. Dr. Roberto da Silva Bigonha (UFMG)

Visto e permitida a impressão

Porto Alegre, 30 / 10 / 91

Prof. Dr. Daltro José Nunes
Orientador



Prof. Dr. Ricardo Augusto da L. Reis
Coordenador do Curso de Pós-Graduação
em Ciência da Computação

ESR/LJSJr \DIVCPGCC\ASSINA.BAN