UM SISTEMA BASEADO EM CONHECIMENTO DE APOIO À MÉTODOS DE DESENVOLVIMENTO DE SOFTWARE

João Fernando Diniz Falcão

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.Sc.) EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovado por:

Profa. Ana Regina C. da Rocha, D.Sc.
(Presidente)

Profa. Teresa Cristina Aguiar, D.Sc.

Profa. Riva Roitman, Doutor em Educação

RIO DE JANEIRO, RJ - BRASIL

MAIO DE 1992

"Se um homem começa com certezas, ele findará com dúvidas; mas se ficar satisfeito por começar com dúvidas, certamente findará com certezas".

Francis Bacon

Aos meus pais...

Aos meus irmãos...

A DEUS...

AGRADECIMENTOS

À Professora Ana Regina Cavalcanti da Rocha pela carinhosa orientação acadêmica, pela sua eterna paciência e por acreditar nos resultados deste trabalho.

A Teresa Cristina Aguiar pelas críticas e sugestões desde o princípio deste trabalho.

À Professora Riva Roitman pela presença na banca examinadora e por suas sugestões ao trabalho.

Aos amigos da COPPE pelas mais diversas colaborações.

Ao CNPq pela ajuda financeira.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

UM SISTEMA BASEADO EM CONHECIMENTO DE APOIO À MÉTODOS DE DESENVOLVIMENTO DE SOFTWARE

João Fernando Diniz Falcão

Maio de 1992

Orientadora: Ana Regina Cavalcanti da Rocha

Programa: Engenharia de Sistemas e Computação

Neste trabalho desenvolve-se um estudo sobre Sistemas Tutoriais Inteligentes e Assistentes Inteligentes com o intuito de criar subsídios para a caracterização de assistentes baseados em conhecimento, que auxiliem nas diversas tarefas do processo de desenvolvimento de software.

O objetivo é desenvolver um modelo de como os assistentes baseados em conhecimento devem interagir com os usuários de ferramentas para desenvolvimento de software, como devem ser projetados e implementados, suas características e arquitetura.

Um protótipo é construido para demonstrar a exequibilidade da assistência baseada em conhecimento para o domínio da Engenharia de Software. O protótipo apoia o usuário na manipulação e controle de uma ferramenta genérica de edição de diagramas de fluxo de dados.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

A KNOWLEDGE BASED SYSTEM
TO ASSIST SOFTWARE DEVELOPMENT METHODS

João Fernando Diniz Falcão

May, 1992

Chairman: Ana Regina Cavalcanti da Rocha
Department: Engenharia de Sistemas e Computação

In this work, we developed a research about Intelligent Tutorial Systems and Intelligent Help Systems. Our aim is to create a background for a characterization of Intelligent Assistants that can help in the various tasks in the process of software development.

The goal is to develop a model of how the knowledge based assistant may to interact with the users of tools for the software development, how they must be designed and implemented, their characteristics and architecture.

A prototype is built to demonstrate the factibilid of the knowledge based assistance for the Software Engineering domain. The prototype help the user in the manipulation and control of a generic tool for edition of data flow diagram.

ÍNDICE

DEDICATÓRIA iii

AGRADECIMENTOS iv

SUMÁRIO v

ABSTRACT vi

LISTA DE ILUSTRAÇÕES x

1. INTRODUÇÃO

1.1. Motivação 1

1.2. Objetivos da Tese 8

1.3. Organização da Tese 11

2. SISTEMAS INTELIGENTES

2.1. Inteligência Artificial 13

2.2. Engenharia de Software &
Inteligência Artificial 21

2.3. Sistemas Baseados em Conhecimento 26

2.4. Desenvolvimento de Sistemas Baseados em
Conhecimento 31

3. SISTEMAS TUTORIAIS INTELIGENTES

3.1. Conceituação 40

3.2. Componentes dos Sistemas de Comunicação de Conhecimento 44

3.2.1. Base do Conhecimento 47

3.2.2. Modelo do Usuário 49

3.2.3. Estratégias de Comunicação 51

3.2.4. Interface 53

3.3. Sistemas Tutoriais Inteligentes com Orientação 55

3.3.1. Projeto WEST 59

3.3.2. Projeto WUSOR 63

3.3.3. Tutor de Geometria 69

3.3.4. Tutor LISP 74

4. ASSISTENTES INTELIGENTES

4.1. Conceituação 79

4.2. Assistentes Inteligentes Presentes na Literatura 85

4.2.1. EUROHELP 86

4.2.2. Projeto Aprendiz de Programador: KBEmacs 95

4.2.3. IN-EDITO: Interface Inteligente para um Editor de Texto 104

5. ARQUITETURA DE UM ASSISTENTE BASEADO EM CONHECIMENTO PARA APOIO A AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE

5.1. Apresentação 108

5.2. Arquitetura do Assistente 116

5.2.1. Modo Orientador 120

5.2.2. Modo Especialista 145

5.2.3. Modo Tutor 148

5.2.4. Modo Meta-Usuário 150

6. UM ASSISTENTE BASEADO EM CONHECIMENTO PARA UM EDITOR DE DIAGRAMAS DE FLUXO DE DADOS

6.1. Descrição do Protótipo 151

7. CONCLUSÃO 168

8. REFERÊNCIAS BIBLIOGRÁFICAS 171

LISTA DE ILUSTRAÇÕES

figura 1. Engenharia do Conhecimento 37
figura 2. As Cinco Fases do Desenvolvimento de um Sistema Baseado em Conhecimento 39
figura 3. Modelo de um Sistema Tutorial Inteligente 45
figura 4. Algumas Questões Básicas na Comunicação de Conhecimento 54
figura 5. Os Principais Componentes de um Sistema Tutorial Inteligente com Orientação 58
figura 6. Diagramas do Fluxo de Dados do Tutor WEST 61
figura 7. Arquitetura do WUSOR-II 66
figura 8. Arquitetura do EUROHELP 87
figura 9. Entradas, Saídas e Processos do Módulo Estratégias de Comunicação 90
figura 10. Filosofia do Assistente de Programação ... 98
figura 11. Visão Geral da Arquitetura do KBEmacs 102
figura 12. Estrutura da Interface 107
figura 13. Arquitetura do Modo Orientador 123
figura 14. Paradigma de Planos e Questões 125
figura 15. Exemplos de Regras de Produção do Modelo Ideal 130
figura 16. Exemplos de Regras de Produção do Modelo de Erros 133

figura 17. Arquitetura do Modo Especialista 147

figura 18. As questões de um Editor de Diagramas de Fluxo de Dados 154

figura 19. Diagramas de Fluxo de Dados 156

figura 20. Representação em Prolog dos Diagramas de Fluxo de Dados da Figura 19 157

figura 21. Modelo Lógico do Protótipo 162

figura 22. Modelo Lógico do Protótipo 163

figura 23. Modelo Lógico do Protótipo 164

figura 24. Modelo Lógico do Protótipo 165

1 INTRODUÇÃO

1.1. Motivação

Atualmente nos deparamos com a proliferação do uso de sistemas de informação pelas mais diversas áreas da sociedade, com a consequente ampliação da complexidade destes sistemas. Por outro lado, os usuários queixam-se frequentemente da baixa qualidade dos produtos oferecidos, quase sempre abaixo de suas expectativas. Este é o resultado de estarmos ainda incipientes nos conceitos da Engenharia de Software, tornando-se necessário pesquisar ferramentas que aumentem a produtividade e a qualidade dos produtos.

No início da década passada, comprovou-se que o software era responsável por uma parcela significativa do custo total do sistema. Fairley [FAIRLEY 85] apontou como causa desta situação o aparecimento do hardware de terceira geração e os consequentes problemas da utilização destas tecnologias, tais como a quebra de custos e prazos pré-estabelecidos, falta de confiabilidade, ineficiência e a consequente rejeição do usuário. Como os sistemas tornaram-se mais complexos, ficou evidente que a demanda de software cresceria mais rápido que a nossa habilidade para produzi-lo.

Estima-se que a demanda de software multiplicará por dez na próxima década, necessitando de grande quantidade de recursos humanos e financeiros. Esta situação caótica foi denominada Crise do Software [JENSEN 79].

O termo Engenharia de Software foi pela primeira vez introduzido em um *"workshop"* em Garmisch, Alemanha Ocidental, para analisar tal situação. Existem muitas definições propostas para Engenharia de Software:

> *"A aplicação prática do conhecimento científico para o projeto e construção de sistemas e a documentação associada necessária para desenvolvê-lo, operá-lo e mantê-lo", Boehm [BOEHM 76].*
>
> *"A técnica sistemática para desenvolver, operar, manter e retirar o software", IEEE Standard Glossary of Software Engineering.*
>
> *"O estabelecimento e o uso de técnicas de engenharia a fim de obter software de baixo custo que seja confiável e opere em uma máquina real", Bauer [BAUER 72].*

O objetivo primário da Engenharia de Software é melhorar a qualidade dos produtos de software e aumentar a produtividade e a satisfação no trabalho dos engenheiros de software.

A fim de obter o rigor utilizado nas demais áreas da engenharia, automatizaram-se os processos de projeto e implementação do software. Em um primeiro instante, buscou-se automatizar a aplicação dos métodos

desenvolvidos. Uma dificuldade nova tornou-se transparente: *"o modelo de ciclo de vida do software e os métodos a ele associados foram idealizados tendo-se em mente a produção artesanal de software" [LUCENA 87].* Baseado nesta premissa e incentivado pelo projeto japonês de quinta geração, buscaram-se soluções na área da Inteligência Artificial. A partir destes estudos, linguagens e métodos de representação do conhecimento encontraram aplicações na Engenharia de Software. A futura geração de ambientes de desenvolvimento de software poderá não diferir em termos de recursos técnicos, mas passará a incorporar modelos explícitos e adaptáveis dos processos que eles são capazes de representar.

As tentativas da Inteligência Artificial em abrir novas fronteiras na Engenharia de Software estão concentradas em três campos de trabalho [MOSTOW 85]:

a. Ambientes de programação de Inteligência Artificial;

b. Estudos dos processos de desenvolvimento de software;

c. Sistemas baseados em conhecimento.

A primeira categoria lida com o aparato experimental desenvolvido para suportar a ciência. A segunda categoria lida com a descoberta de tipos de conhecimento usados na tarefa de criação do software. A última categoria lida com a incorporação deste conhecimento em sistemas práticos.

a. *Ambientes de programação de Inteligência Artificial*:

A Inteligência Artificial encorajou a criação de ferramentas sofisticadas para suportar o rápido desenvolvimento dos programas para aplicações mal-entendidas, característica típica da Inteligência Artificial.

No processo de desenvolvimento, comumente denominado Programação Exploratória, a implementação precede a especificação. Os programas de Inteligência Artificial são parecidos com a maioria dos softwares, apenas com um detalhe: o desenvolvimento de um programa de Inteligência Artificial pode ser visualizado como uma forma acelerada de manutenção envolvendo frequentes mudanças na especificação. Tradicionalmente, métodos da Engenharia de Software trabalham mal em situações instáveis. Já a Inteligência Artificial requer o desenvolvimento de técnicas alternativas para suportar a evolução rápida do programa.

A dificuldade do desenvolvimento de um programa de Inteligência Artificial encontra-se em como expressar o conhecimento sobre o domínio de aplicação em termos permitidos pela linguagem de

programação - LISP e PROLOG são as linguagens mais populares. Estas linguagens fornecem paradigmas alternativos tais como regras, lógica e orientação a objetos, pelos quais determinados tipos de conhecimento do domínio podem ser expressos facilmente.

b. *Estudos dos processos de desenvolvimento de software:*

Com intuito de descobrir os tipos de conhecimento utilizados na tarefa de criação de software, a Inteligência Artificial tem-se aproveitado da psicologia cognitiva. Pelo lado da psicologia, uma maneira prática de investigar o modelo de como o ser humano executa certas tarefas é implementar o modelo em um programa. Pelo lado da Inteligência Artificial, uma maneira prática para automatizar uma tarefa é descobrir como as pessoas executam uma tarefa.

c. *Sistemas baseados em conhecimento:*

A expressão *'baseados em conhecimento'* caracteriza sistemas com conhecimento explícito sobre o processo da Engenharia de Software, em oposição a ferramentas como editores que não podem raciocinar sobre as decisões estabelecidas para a implementação.

Estes sistemas podem variar em muitas dimensões, a saber:

a. *Escopo:* a que tipo de software é destinado?

b. *Poder:* o quanto é automatizado?

c. *Nível:* a que ciclo de vida é destinado?

d. *Propósito:* a que partes do ciclo de vida é destinado?

e. *Conhecimento:* que tipos de conhecimento são utilizados explicitamente pela máquina?

Estudos experimentais do desenvolvimento de software demonstraram como ainda é misterioso o processo executado pelo homem. Este processo deveria ser executado ou assistido pela máquina. As pesquisas na área da Inteligência Artificial prometem incrementos na produtividade, confiança e flexibilidade do software.

Pode-se estimar alguns impactos na Engenharia de Software caso estas pesquisas mostrem resultados positivos:

a. Surgimento de novos paradigmas na Engenharia de Software. Mudança do próprio ciclo de vida, com a utilização de protótipos rápidos a fim de reduzir significativamente as incertezas sobre o produto final e a reutilização dos históricos dos projetos;

b. Se um sistema de programação automática estiver disponível, pode-se esperar uma redução drástica no esforço requerido para a decomposição, implementação, otimização e teste e uma redução razoável no esforço para a validação. Por outro lado, mais esforço é requerido durante a especificação. Espera-se um fator de redução no esforço em torno de três.

1.2. Objetivos da Tese

Este trabalho está dentro do âmbito do Projeto Taba [ROCHA 88], cujo objetivo é a construção de uma estação de trabalho configurável para o desenvolvimento de software, que especifique e gere ambientes adequados às características dos produtos a serem desenvolvidos.

O Projeto Taba emergiu a partir da consciência de que *"aplicações diferentes têm perfis diferentes e estas diferenças tem influência sobre seus processos de desenvolvimento"*. Consequentemente, pode-se afirmar que as características de uma certa aplicação determinam as características de seu ambiente de desenvolvimento.

A estação de trabalho a ser construida no Projeto Taba poderá ser usada para [COPPE 87]:

a. Auxiliar o engenheiro de software na configuração (especificação) do ambiente mais adequado ao desenvolvimento de um produto específico;

b. Auxiliar o engenheiro de software a implementar as ferramentas necessárias ao ambiente definido em (a);

c. Permitir aos que desenvolvem software utilizar a estação através do ambiente especificado em (a) e produzido em (b);

d. Executar o software, dado que, eventualmente, o software-produto poderá ser

executado na própria estação para ele configurada (isso é sempre verdade, pelo menos na fase de testes).

O objetivo desta tese é fornecer subsídios para a construção de Assistentes Baseados em Conhecimento que auxiliem os usuários da Estação Taba nas diversas tarefas do processo de desenvolvimento de software. O assistente deve apoiar o usuário na manipulação e controle das ferramentas responsáveis pela automatização das atividades da Engenharia de Software.

Um assistente inteligente não apenas responde questões do usuário, mas também *'olha sobre seus ombros'* e o interrompe quando necessário. O assistente deve comportar-se como um orientador humano, que interrompe quando as atividades não estão caminhando bem ou quando existe uma oportunidade de ampliar o conhecimento do usuário. Além disso, deve ser capaz de responder questões no contexto da atividade.

Uma consequência natural ao acoplar um assistente baseado em conhecimento a uma determinada ferramenta é a sua transformação em uma ferramenta que realmente *'entende'* com profundidade o *'que'*, para *'quem'* e *'como'* deve auxiliar. O intuito é criar um novo agente no processo de desenvolvimento, com as características de uma ferramenta inteligente e interagindo com o desenvolvedor como um assistente humano.

Os resultados apresentados neste trabalho foram calcados em estudos detalhados de Sistemas Tutoriais

Inteligentes e Assistentes Inteligentes de propósito geral presentes na literatura. A partir de análises das características destes sistemas, construi um modelo para Assistentes Baseados em Conhecimento cujo propósito é auxiliar o Engenheiro de Software no desenvolvimento de suas atividades.

1.3. Organização da Tese

Durante a elaboração da tese, houve uma preocupação constante em prepará-la de forma completa. Consequentemente, algumas seções poderiam estar menos detalhadas, mas a intenção é que o leitor não interrompa a leitura para buscar informações em outras referências.

Os capítulos concernentes aos Sistemas Tutoriais Inteligentes com Orientação e aos Assistentes Inteligentes representam a espinha dorsal da tese. Estes capítulos fundamentam o propósito crucial do trabalho: a caracterização de assistentes baseados em conhecimento de apoio a Ambientes de Desenvolvimento de Software.

O capítulo 1 (Introdução) expõe as principais motivações para o desenvolvimento da tese.

O capítulo 2 (Sistemas Inteligentes) apresenta o segmento da ciência da computação denominado Inteligência Artificial, a busca de soluções neste segmento para resolver dificuldades da Engenharia de Software, e, finalizando, descreve um caminho resultante desta interação: sistemas baseados em conhecimento para auxiliar as atividades da Engenharia de Software.

O capítulo 3 (Sistemas Tutoriais Inteligentes) descreve as propriedades de sistemas de instrução baseados em técnicas da Inteligência Artificial, de onde foi possível retirar uma grande quantidade de conceitos

úteis para o projeto do assistente. O paradigma de Sistemas Tutoriais Inteligentes com Orientação inspirou estudos cuidadosos, devido à sua filosofia de prover assistência não intrusa.

O capítulo 4 (Assistentes Inteligentes) exibe as características de assistentes baseados em conhecimento que estão definidos na literatura.

O capítulo 5 (Arquitetura de um Assistente Baseado em Conhecimento para Apoio a Ambientes de Desenvolvimento de Software) recolhe os conceitos importantes descritos anteriormente e define as características específicas de um assistente baseado em conhecimento para auxiliar na manipulação e controle das ferramentas responsáveis pela automatização das atividades do processo de desenvolvimento de software.

O capítulo 6 (Um Assistente Baseado em Conhecimento para um Editor de Diagramas de Fluxo de Dados) descreve um pequeno protótipo desenvolvido para demonstrar a exequibilidade da técnica de assistência baseada em conhecimento para o domínio da Engenharia de Software.

O capítulo 7 (Conclusão) apresenta algumas considerações finais sobre o tema e aponta para alternativas possíveis de futuros trabalhos.

2. SISTEMAS INTELIGENTES

2.1. Inteligência Artificial

O termo Inteligência Artificial tem sofrido uma contínua crise de identidade desde que foi mencionado por John McCarthy. Segundo Rich [RICH 88], o ramo Inteligência Artificial é o responsável pelo estudo de como fazer os computadores realizarem tarefas que ,no momento, são melhor realizadas por pessoas.

As pessoas executam com superioridade as atividades que envolvem inteligência. Elas fazem bem mais do que processar informação; elas *'entendem'* a informação. Então o objetivo da Inteligência Artificial é tornar o computador mais inteligente. Este conceito forma a base para uma segunda definição [BARR 82]:

> *"Inteligência Artificial é a parte da ciência computacional concentrada no projeto de sistemas inteligentes, isto é, sistemas que exibem características que nós associamos com inteligência no comportamento humano".*

Uma definição exata de inteligência mostra-se extremamente evasiva. As seguintes características são sugeridas como uma lista de habilidades essenciais para inteligência [MISHKOFF 86]:

a. *Responder a situações com flexibilidade:*

Nós podemos emitir respostas diferentes para uma mesma situação. Assim, não há necessidade de respostas idênticas a cada vez que o sistema for confrontado com um mesmo problema.

b. *Ser coerente mesmo com mensagens ambíguas ou contraditórias:*

Nós somos capazes de entender muitas afirmações que parecem ser ambíguas ou contraditórias porque nosso conhecimento e experiência permite colocá-las no contexto;

c. *Reconhecer a importância relativa de diferentes elementos de uma situação:*

Embora sejamos bombardeados com uma quantidade esmagadora de informação, a cada dia, determinamos diferentes níveis de importância para diferentes eventos;

d. *Encontrar similaridades entre situações apesar das diferenças que podem separá-las:*

Pelo reconhecimento de similaridades, podemos basear nossas futuras ações naquilo que foi aprendido no passado;

e. Descobrir distinções entre situações apesar das equivalências que podem ligá-las:

Embora duas situações possam parecer similares, somos capazes de notar as diferenças que podem conduzir-nos a ajustar nossas reações.

As duas definições apresentadas anteriormente para o termo Inteligência Artificial concentram-se na comparação entre as respectivas habilidades dos seres humanos e dos computadores. As definições apresentadas a seguir estão focalizando as diferenças entre as técnicas de programação utilizadas na Inteligência Artificial e os métodos convencionais de programação:

"Inteligência Artificial é o ramo da ciência computacional que lida com métodos simbólicos e não algoritmicos de solução de problemas", Shortliffe [SHORTLIFFE 84].

"Inteligência Artificial é o ramo da ciência computacional que lida com maneiras de representar o conhecimento utilizando símbolos e regras práticas - heurísticas, métodos para o processamento da informação", Buchanan [BUCHANAN 85].

"Em termos simplificados, Inteligência Artificial trabalha com métodos de reconhecimento de padrões que tentam descrever objetos, eventos ou processos em termos de suas características qualitativas e relações lógicas e computacionais", Battle Research Corporation [MISHKOFF 86].

A natureza dos computadores no futuro e o impacto que terão sobre o nosso dia-a-dia serão determinados pelo que acontecer no campo da Inteligência Artificial na próxima década [CHILDERS 84]. Sendo um ramo relativamente novo, a Inteligência Artificial continua a experimentar rápidas mudanças no seu escopo.

A Inteligência Artificial redefine continuamente o que os computadores podem fazer e empurra as tecnologias existentes para o seu limite. Quando um problema particular de Inteligência Artificial está substancialmente solucionado, os pesquisadores deslocam-se para novos desafios. Isto resulta em uma definição muito fluida de Inteligência Artificial como um escopo de uma ciência continuamente deslocando-se em novas direções. Childers resume muito bem esta situação [CHILDERS 84]:

> *"É talvez mais difícil colocar a Inteligência Artificial em perspectiva do que o próprio computador. Inteligência Artificial tende a ser uma disciplina evasiva. Inteligência Artificial é melhor entendida como uma evolução do que uma revolução".*

Existem diversos sub-campos de pesquisa na área da Inteligência Artificial, a saber [MISHKOFF 86]:

a. *Sistemas Especialistas:*

Um sistema especialista é um programa projetado para agir como um especialista em um domínio particular. Um sistema especialista inclui uma base de conhecimento, consistindo de fatos sobre o domínio, e heurísticas para a aplicação destes fatos. Sistemas especialistas foram projetados para assistir o especialista, ao invés de substituí-lo. Eles tornaram-se bastante úteis em áreas como diagnóstico médico, análise química, exploração geológica e configuração de sistemas. Tornou-se a primeira tecnologia da Inteligência Artificial a ter um grande impacto na indústria e nos negócios;

b. *Processamento de Linguagem Natural:*

A utilidade dos computadores é frequentemente limitada pelas dificuldades de comunicação. O uso efetivo de um computador envolve a utilização de uma linguagem de programação ou um conjunto de comandos que tem que ser utilizados para a comunicação com o computador. O objetivo do processamento de linguagem natural é permitir às pessoas e computadores a comunicação em uma linguagem humana natural;

c. *Reconhecimento da Fala:*

O objetivo da pesquisa do reconhecimento da fala é permitir aos computadores entender a fala humana de tal maneira que possam ouvir nossas vozes e reconhecer as palavras que estamos falando, simplificando o processo de comunicação interativa entre pessoas e computadores;

d. *Visão por Computador:*

é uma tarefa simples unir uma câmera ao computador de tal maneira que possa receber as imagens visuais. Entretanto, provou-se ser uma tarefa dificílima interpretar estas imagens para que o computador possa entender exatamente o que está vendo. As pessoas geralmente utilizam a visão como meio primário de percepção do ambiente. O objetivo da pesquisa nesta área é dotar o computador com a mesma facilidade poderosa para entender o seu ambiente. Uma das primeiras aplicações é na área da robótica.

e. *Robótica:*

Um robô é um instrumento eletro-mecânico que pode ser programado para executar tarefas manuais. A Associação das Indústrias de Robótica definiram um robô como *"um manipulador multi-funcional programável projetado para mover materiais, ferramentas ou instrumentos especializados através de movimentos programados para executar uma variedade de tarefas"*. Nem todo robô é considerado como parte da Inteligência Artificial, um robô inteligente inclui algum tipo de

aparelho sensorial, tal como uma câmera, que permita responder às mudanças do ambiente.

f. *Instrução Inteligente Assistida por Computador:*

O poder do computador tem sido utilizado a alguns anos para influenciar o processo educacional. Agora, os métodos da Inteligência Artificial estão sendo aplicados para o desenvolvimento de sistemas de instrução inteligente com o intuito de criar tutores computadorizados que modelam suas técnicas de ensino de acordo com os padrões de aprendizado de cada estudante;

g. *Ferramentas Inteligentes para o Desenvolvimento do Software:*

São programas da Inteligência Artificial especialmente projetados para ajudar os programadores a executar várias tarefas do desenvolvimento do software, incrementando desta maneira a sua produtividade. Embora estas ferramentas possam simplificar vários aspectos do desenvolvimento do software, elas ainda requerem um esforço considerável do programador;

h. *Programação Automática:*

As ferramentas inteligentes para o desenvolvimento do software são apenas um estado intermediário do objetivo final do desenvolvimento do software: a Programação Automática. Se os programas fossem capazes de *'escrever a si próprios'*, não existiria a necessidade de programadores e muito menos de ferramentas para aumentar sua eficiência;

i. *Planejamento e Suporte à Decisão:*

O desenvolvimento de planos pode necessitar de uma coleção e uma avaliação de quantidades significativas de informação. Programas inteligentes de planejamento são projetados para prover uma assistência ativa no processo de planejamento e espera-se que sejam particularmente úteis na tomada de decisões.

2.2. Engenharia de Software e Inteligência Artificial

A simbiose sempre beneficia as duas entidades relacionadas: a Inteligência Artificial e a Engenharia de Software rumam para um casamento frutífero. Este relacionamento está em um estágio incipiente, mas muitos trabalhos acadêmicos e industriais já incluem tecnologias baseadas em conhecimento.

A Engenharia de Software é uma atividade de intenso conhecimento, exigindo conhecimento extensivo do domínio de aplicação e do próprio software alvo. Muitos dos custos da Engenharia de Software podem ser atribuídos à ineficiência das técnicas correntes para gerenciar este conhecimento. As técnicas da Inteligência Artificial podem amenizar esta situação [BARSTOW 87]. Uma vez que a Engenharia de Software requer inteligência, parece natural a utilização de técnicas da Inteligência Artificial para a construção de sistemas que executem e assistam o processo de desenvolvimento do software.

Segundo Simon [SIMON 86], a verdadeira questão não é apenas querer usar os métodos da Inteligência Artificial na Engenharia de Software. Deve-se questionar se a Inteligência Artificial é suficientemente poderosa, se existe conhecimento o suficiente sobre esta área e se ela está avançada o suficiente para realmente prestar auxílio.

Por natureza, a Engenharia de Software é mal-estruturada, uma característica emprestada principalmente do processo do projeto combinado com questões de gerência. Por outro lado, os sistemas inteligentes são úteis para solucionar tipos bem formulados de problemas mal-estruturados que possuam um método forte para a sua solução. Estes métodos fortes consistem de heurísticas que especialistas humanos acumularam durante anos de experiência [TSAI 86].

Grande parte do conhecimento da Engenharia de Software não está armazenado explicitamente. Conhecimento sobre o domínio de aplicação é refletido nos documentos de especificação e requisitos, mas raramente são armazenadas as razões por trás destes documentos. Da mesma maneira, a maioria das decisões do projeto e implementação não está estabelecida e a documentação usualmente não inclui nenhuma discussão sobre as motivações das decisões. Sendo assim, os desenvolvedores do software trabalham em um contexto de grande incerteza sobre estas duas importantes áreas do conhecimento. Estas incertezas podem ser removidas apenas com suporte automatizado para gerenciar este conhecimento.

Os argumentos básicos para aplicar técnicas da Inteligência Artificial nas atividades da Engenharia de Software podem ser assim sumarizadas [BARSTOW 87]:

a. As atividades da Engenharia de Software têm intensa parcela de conhecimento embutido;

b. Técnicas correntes para a gerência do conhecimento lidam com grande incerteza;

c. Remover a incerteza requer suporte automatizado para representar explicitamente o conhecimento relevante;

d. O suporte automatizado para representar e usar efetivamente o conhecimento requer técnicas da Inteligência Artificial.

Entretanto, pode-se ser mais específico sobre o uso das técnicas da Inteligência Artificial:

a. O paradigma de pesquisa heurística pode ser utilizado para modelar as atividades do projeto e implementação;

b. Sistemas de inferência formal são necessários para analisar e inferir propriedades sobre o domínio de aplicação e o software alvo;

c. Técnicas de representação do conhecimento, em geral, e sistemas baseados em regras, em particular, são necessários para representar e usar conhecimento sobre métodos da Engenharia de Software;

d. Técnicas de representação do conhecimento são necessárias para armazenar e recuperar conhecimento sobre o domínio de aplicação;

e. Técnicas de representação do conhecimento são necessárias para armazenar e recuperar conhecimento sobre a história do software;

f. Técnicas de aquisição de conhecimento são necessárias durante a especificação e requisitos;

g. Técnicas de explanação do conhecimento são necessárias durante a evolução e manutenção.

Evidentemente que as técnicas da Inteligência Artificial não são suficientes para solucionar todos os problemas da Engenharia de Software. Uma razão é que sistemas práticos necessitam mais do que técnicas de Inteligência Artificial (um caso prático de sistema inteligente - *dipmeter advisor*: 30% do código representava a parte da Inteligência Artificial; 13% algoritmos de processamento; 42% de interface com o usuário e 15% de ambiente de suporte). A segunda razão é que outras técnicas são necessárias para suportar atividades da Engenharia de Software, mesmo sem a presença de técnicas da Inteligência Artificial.Em outras palavras, as técnicas da Inteligência Artificial serão mais efetivas se forem aplicadas de maneira coordenada e coerente.

Infelizmente, durante duas décadas de pesquisas não houve demonstração da utilidade das técnicas da Inteligência Artificial em situações práticas na Engenharia de Software. A principal lição dos trabalhos pioneiros nesta área pode ser sumarizada em quatro palavras: **desempenho inteligente requer conhecimento**. Embora as primeiras tentativas de programas inteligentes

de propósito geral (como solucionador de problemas gerais) apareceram como promissoras no começo, provaram-se limitadas devido a falta de conhecimento do domínio de aplicação.

O software é desenvolvido para solucionar problemas existentes em determinado domínio de aplicação. Conhecimento do domínio de aplicação desempenha papel fundamental no processo de desenvolvimento do software porque ele provê o contexto tanto para a definição do problema quanto para a avaliação dos sistemas propostos como solução. O conhecimento do domínio de aplicação é um dos ramos do conhecimento necessários para apoiar a Engenharia de Software. As vantagens do enfoque baseado em conhecimento podem ser estendidas para outros aspectos, como o conhecimento sobre programação e o conhecimento sobre gerência de projeto. Cada um desses temas correspondem a um domínio independente, para os quais os mesmos métodos e ferramentas podem ser aplicados. Se todas as descrições, inclusive aquelas do próprio software, são representadas em bases de conhecimento, consequentemente os mesmos métodos e ferramentas podem ser utilizados uniformemente para todos os aspectos da Engenharia de Software [LUCENA 87].

Baseado nestas observações, as pesquisas em Inteligência Artificial estão focalizadas em como adquirir, representar, organizar e aplicar tais conhecimentos para uma variedade de aplicações.

2.3. Sistemas Baseados em Conhecimento

A quantidade de publicações sobre sistemas baseados em conhecimento é bem relevante. Existem artigos de pesquisas em periódicos científicos, em revistas populares de computação e na imprensa em geral. Ou seja, o potencial comercial destes sistemas começa a ser reconhecido.

Feigenbaum define um sistema baseado em conhecimento como um *"programa inteligente que usa conhecimento e procedimentos de inferência para solucionar problemas que requerem conhecimento humano significativo para a sua solução"* [HARMON 85].

Todo programa contém conhecimento sobre algum problema. A diferença entre os sistemas baseados em conhecimento e os programas convencionais é que os primeiros representam o conhecimento em alto nível, armazenando-o em uma base de conhecimento de fatos e heurísticas que imitam a maneira como as pessoas raciocinam sobre um problema. Além do mais, estes programas podem explicar seu conhecimento ou responder questões sobre seu uso.

Em resumo, um sistema baseado em conhecimento deve ser capaz de realizar as seguintes tarefas [WALKER 87]:

a. Solucionar ou ajudar a solucionar problemas importantes que de outra maneira exigiriam os serviços de um especialista humano;

b. Integrar novos conhecimentos junto à base de conhecimento;

c. Auxiliar o projetista a deduzir, organizar e transferir conhecimento;

d. Apresentar o conhecimento de uma forma fácil de ser entendida pelas pessoas;

e. Prover justificativas de seus conselhos;

f. Raciocinar com 'bom senso' ou conhecimento inexato sobre a natureza de uma tarefa ou métodos de realizá-la eficientemente;

g. Raciocinar com conhecimento declarativo;

h. Suportar uma interface legível e natural.

Algumas destas características podem faltar em determinados sistemas, portanto elas podem ser utilizadas como *'pesos de sofisticação'* de um sistema baseado em conhecimento. Para suportar estas características, estes sistemas possuem, tipicamente, os seguintes componentes:

a. Uma *base de conhecimento* codificada na forma de fatos e heurísticas;

b. Uma *máquina de inferência* para raciocinar com fatos e heurísticas e controlar a busca de soluções;

c. *Métodos* para adquirir novos conhecimentos e codificá-los na base de conhecimento;

d. Um *manipulador de diálogos* que pode variar em complexidade desde um menu até um processador de linguagem natural.

O desenvolvimento de sistemas baseados em conhecimento difere das técnicas *"top-down"* utilizadas no software convencional. Os sistemas baseados em conhecimento devem codificar formas heurísticas complexas de conhecimento incluindo símbolos, estratégias e relações. Representar tais conceitos é extremamente difícil em linguagens convencionais como Pascal ou C. Mas o importante é que ninguém sabe exatamente como os especialistas chegam às conclusões - nem eles próprios. Portanto, o desenvolvimento de sistemas baseados em conhecimento nunca é um processo linear. Baseado em repetidas entrevistas com especialistas, o sistema tenta aumentar seu conhecimento em iterações sucessivas. Projeto e codificação estão integrados dentro de um processo iterativo simples [WILLIAMS 86].

Os sistemas baseados em conhecimento tentam copiar o comportamento de um especialista na solução de problemas em um domínio de conhecimento particular. O conhecimento do especialista reúne os fatos aplicados a uma área particular e o conhecimento de como e quando utilizar estes fatos para solucionar um problema neste domínio.

Capturar e utilizar o conhecimento do domínio requer [IEEE SOFTWARE 88]:

a. *Aquisição do conhecimento* é o maior obstáculo para a construção de sistemas baseados em conhecimento. Este obstáculo deriva do fato que o especialista não está treinado para expressar seu conhecimento de uma maneira que possa ser codificado;

b. *Representação do conhecimento* é um conjunto de convenções sintáticas e semânticas que nos permitem descrever e manipular objetos abstratos do domínio;

c. *Manipulação do conhecimento* requer um conjunto combinado de métodos de raciocínio e mecanismos de inferência.

Partindo-se da premissa que o desenvolvimento de software é uma atividade de intenso conhecimento, exigindo conhecimento extensivo do domínio de aplicação, parece natural a utilização de sistemas baseados em conhecimento que executem e assistam o processo de desenvolvimento de software.

Lucena [LUCENA 87] aponta estes sistemas como *"objetivos intermediários naturais antes de alcançar a programação automática"*, ou seja, não se pode começar construindo um sistema automático. Deve-se então construir sistemas interativos que permitam a transferência gradual de tarefas inteligentes para a máquina.

Logo, uma das aplicações mais imediatas dos sistemas baseados em conhecimento consiste em apoiar a atividade do engenheiro de software nas diversas fases do desenvolvimento do software. Como o conhecimento sobre especificação de requisitos, projeto e programação é mal estruturado (muito pouco está formalizado e o que se sabe está baseado na experiência de especialistas) faz sentido a construção de ferramentas isoladas ou integradas em ambientes que são assistentes inteligentes em tarefas especializadas de concepção ou implementação de software. Tais ferramentas, ao longo do ciclo de vida, podem tornar-se guias especializados no uso de ambientes de desenvolvimento de software, cumprindo uma função integradora.

2.4. Desenvolvimento de Sistemas Baseados em Conhecimento

Um típico sistema baseado em conhecimento simula o processo de raciocínio humano pela aplicação de conhecimento e inferências específicos. A intenção é solucionar problemas que requerem um extenso conhecimento humano.

A idéia da tecnologia *'baseado em conhecimento'* é igualar o desempenho destes sistemas com a performance dos peritos humanos. Os peritos são pessoas que possuem a habilidade de decidir corretamente em situações que estão carregadas de incertezas e riscos. Esta qualidade é o resultado de treinamento, experiência e prática profissional [HU 87].

O sucesso do projeto de um sistema baseado em conhecimento depende de três fatores: A definição cuidadosa do seu escopo, a seleção de uma boa área de aplicação e a escolha das técnicas de representação do conhecimento apropriadas.

A utilização da tecnologia de sistemas baseados em conhecimento para qualquer aplicação de maneira indiscriminada é uma técnica usual e errônea. A atitude correta é identificar o problema que necessita de solução e então, caso faça sentido, aplicar a tecnologia, porque apenas determinadas categorias de problemas podem ser

satisfatoriamente solucionados com um sistema baseado em conhecimento.

A primeira etapa na construção de um sistema baseado em conhecimento é a identificação do problema [FRENZEL 87] [WEITZEL 89]. No caso do trabalho corrente, a intenção é criar subsídios para a construção de assistentes que auxiliem na manipulação e controle das ferramentas responsáveis pelas atividades envolvidas no processo de desenvolvimento do software. Com o intuito de testar as idéias apresentadas, optou-se pela construção de um pequeno protótipo para assistir uma ferramenta genérica de edição de diagramas de fluxo de dados.

O processo de decisão da adequação da aplicação é uma das fases mais críticas no desenvolvimento de sistemas baseados em conhecimento. Um conjunto de diretrizes devem ser observadas com cuidado antes do início da construção do sistema. Ao estabelecer a conveniência da construção de um sistema baseado em conhecimento, é imprescendível considerar se o desenvolvimento é possível, justificado e apropriado.

O desenvolvimento do sistema é possível, caso [WATERMAN 86] [ROLSTON 88] [KELLER 87]:

a. A tarefa não exija intuição e senso comum;

b. A tarefa exija apenas habilidades cognitivas;

c. Existam peritos genuínos capazes de solucionar o problema no domínio em questão;

d. Os peritos estejam disponíveis;

e. Os peritos sejam capazes de articular seus métodos de maneira razoável;

f. Os peritos desejem fornecer o conhecimento;

g. Os peritos gozem de credibilidade junto aos usuários potenciais do sistema;

h. Os peritos concordem com as soluções;

i. A tarefa não seja muito complexa;

j. A tarefa não esteja mal entendida;

k. Exista uma disponibilidade razoável de tempo para desenvolver o sistema.

Entretanto, apenas porque é possível desenvolver um sistema baseado em conhecimento para uma tarefa particular, não significa que é desejável construi-lo. Existem alguns motivos que justificam o esforço do desenvolvimento [WATERMAN 86] [ROLSTON 88] [KELLER 87]:

a. Os peritos estão sumindo, portanto existe a necessidade de preservar o conhecimento que está se perdendo;

b. Os peritos estão escassos;

c. A necessidade da presença do perito em muitos lugares;

d. A necessidade da presença do perito em ambiente hostil;

e. A existência de um desnível muito grande entre a performance do perito e do praticante comum;

f. A necessidade de diversas pessoas para solucionar o problema, uma vez que não existe uma pessoa que detenha todo o conhecimento;

g. A necessidade de um grande volume de conhecimento, dificultando a organização e manipulação das informações;

h. A existência de uma dependência enorme em relação ao perito;

i. A solução do problema apresenta um alto custo.

A natureza, complexidade e escopo do problema a ser resolvido determinam quando é apropriado desenvolver um sistema baseado em conhecimento [WATERMAN 86] [ROLSTON 88] [KELLER 87] [FRENZEL 87]:

a. A solução requer o uso de conhecimento de especialistas;

b. A tarefa requer manipulação simbólica;

c. A tarefa requer soluções heurísticas;

d. A tarefa exije decisões baseadas em dados muitas vezes incompletos, não confiáveis, ambíguos e dinâmicos;

e. A tarefa lida com certo grau de incerteza;

f. Ocasionalmente, é aceitável para o sistema falhar na busca da solução ou produzir uma solução *'não-ótima'* em certos casos;

g. O problema possui um nível moderado de dificuldade;

h. O problema possui um número mínimo de soluções;

i. A tarefa tem um valor prático.

Outro aspecto importante, ao decidir pela construção de um sistema baseado em conhecimento, é determinar a

exequibilidade econômica através da elaboração de uma análise de custo/benefício do projeto. Os custos incluem o tempo gasto pelo perito, o tempo gasto pelo engenheiro do conhecimento e os custos relacionados com hardware e software necessários. Os benefícios do sistema incluem a rapidez e confiança com que a tarefa será executada, a oportunidade de servir a diversos usuários, a eliminação de treinamento e a segurança ao preservar conhecimento crítico.

A Engenharia de Software, devido a natureza de suas atividades, tem intensa parcela de conhecimento embutido. Um agravante é o fato que grande parte deste conhecimento não está armazenado explicitamente, porque consiste de heurísticas que peritos acumularam durante anos de experiência. Portanto, parece uma decisão natural projetar sistemas baseados em conhecimento que executem e assistam o processo de desenvolvimento do software.

O desenvolvimento é disparado com a Engenharia do Conhecimento, ou seja, adquirindo o conhecimento [FRENZEL 87]. O conhecimento tem origem em várias fontes. Livros, artigos e outras referências podem ser fontes valiosas, entretanto listas de conceitos e regras, tais como encontradas nestes documentos, não são suficientes. O verdadeiro conhecimento é obtido através de indivíduos que são especialistas no domínio específico.

O engenheiro do conhecimento é responsável pelo processo de transferência do conhecimento do perito para o sistema. Ele extrai dos peritos seus procedimentos, estratégias e regras práticas para a solução do problema.

A figura 1 ilustra este mecanismo. Abaixo, estão descritas as principais tarefas do engenheiro do conhecimento [HU 87]:

a. Adquirir conhecimento do domínio de aplicação em questão;

b. Auxiliar os peritos na definição e modelagem do problema;

c. Entrevistar os peritos com o intuito de obter o conhecimento;

d. Estruturar e compilar o conhecimento;

e. Validar o conhecimento;

f. Implementar o sistema.

Existem diversas técnicas utilizadas para extrair o conhecimento do perito [HU 87] [KELLER 87] [FRENZEL 87]. Esta é uma atividade de extrema importância, porque o poder de solução de um sistema baseado em conhecimento emana do corpo do conhecimento que acumula durante a construção.

A evolução de um sistema baseado em conhecimento dirige-se de tarefas simples para tarefas mais complexas pelo incremento da organização e representação do conhecimento do sistema [WEITZEL 89]. Esta técnica incremental significa que o próprio sistema pode auxiliar no esforço de desenvolvimento. Logo que o desenvolvedor

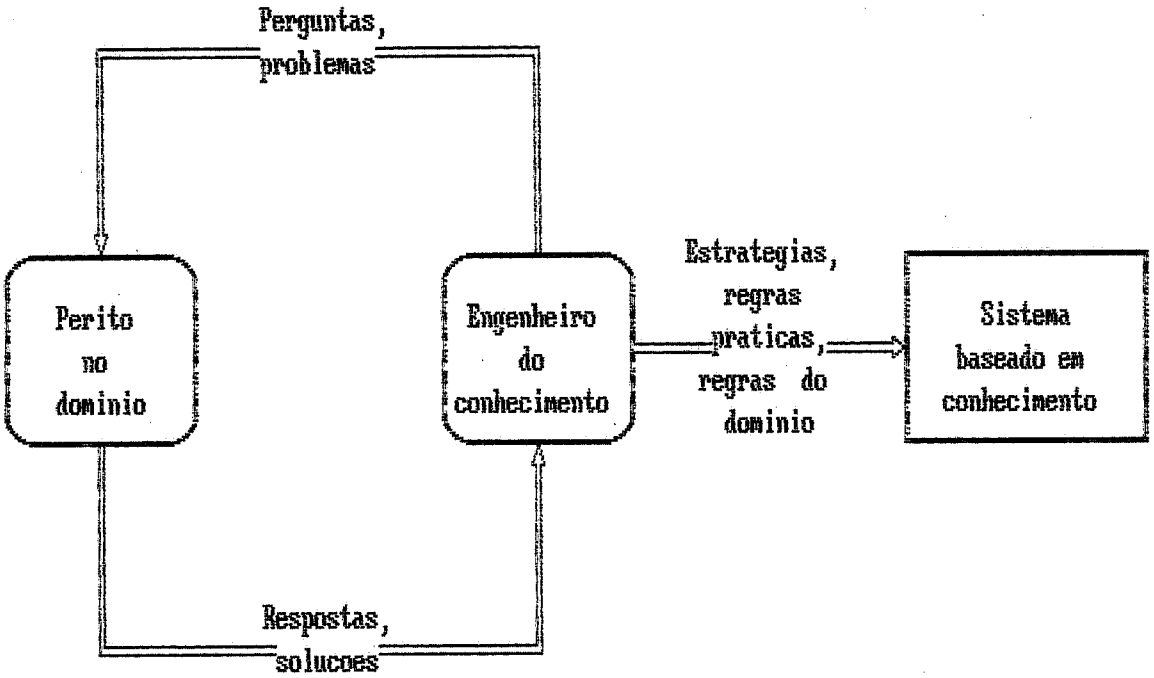


figura 1. Engenharia do Conhecimento

[WATERMAN 86]

adquire conhecimento suficiente para construir um sistema simples, ele o faz e usa o *"feedback"* do modelo executável para direcionar o esforço.

O desenvolvimento de um sistema baseado em conhecimento apresenta cinco fases altamente interdependentes [WATERMAN 86]: identificação, conceituação, formalização, implementação e testes. A figura 2 apresenta as cinco fases e as suas interdependências.

A escolha da linguagem adequada para a construção do sistema é uma decisão difícil. Qualquer linguagem pode ser utilizada para desenvolver o código executável de um sistema baseado em conhecimento. Uma linguagem de Inteligência Artificial não é necessariamente obrigatória. Entretanto, elas reduzem de maneira significativa o tempo de desenvolvimento. Mas estas linguagens são fracas em fatores como portabilidade e desempenho. Uma boa estratégia é desenvolver o protótipo em linguagens de Inteligência Artificial e, então, traduzí-lo para uma linguagem de propósito geral. Isto é mais rápido do que construir o sistema direto na linguagem de propósito geral. Atualmente, Lisp e Prolog são as linguagens de Inteligência Artificial disponíveis e apropriadas.

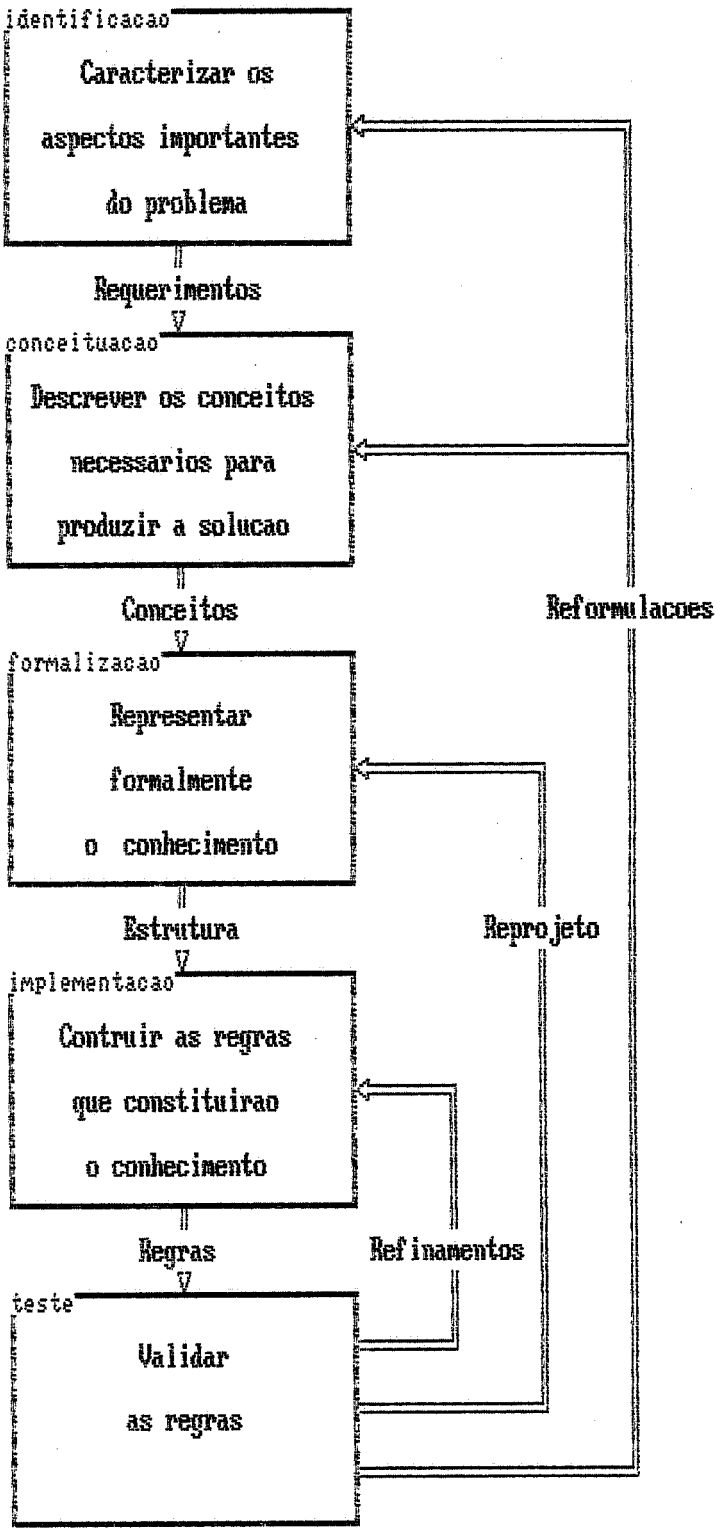


figura 2. As Cinco Fases do Desenvolvimento de um Sistema Baseado em Conhecimento

[WATERMAN 86]

3 SISTEMAS TUTORIAIS INTELIGENTES

3.1. Conceituação

O objetivo principal deste trabalho é buscar subsídios para a construção de um Assistente Baseado em Conhecimento. Após um estudo da literatura atual, observou-se que os Assistentes Inteligentes apresentavam muitas semelhanças com os Sistemas Tutoriais Inteligentes, de onde foi possível retirar uma grande gama de conceitos e propriedades úteis para o projeto do assistente.

A expressão Sistema Tutorial Inteligente reflete a própria história da pesquisa nesta área: os pioneiros nestes estudos (em particular John Seely Brown, Alan Collins e Ira Goldstein [LAMSWEERDE 87]), com intuito de contrastar seus trabalhos com os tradicionais sistemas de instrução apoiados por computador, chamaram seus programas baseados em técnicas da Inteligência Artificial de *"Intelligent Computer-Assisted Instruction" (ICAI)*. A expressão *"Intelligent Tutoring Systems" (ITS)* possui o mesmo significado.

Os Sistemas Tutoriais alcançaram muita importância quando alguns estudos [WOOLF 88] demostraram que a tutoria individual apresenta uma eficácia surpreendentemente maior que a instrução convencional de sala de aula. Estes estudos mostraram a melhoria no desempenho do aluno em cerca de 98% comparado com o treinamento tradicional. Este resultado coloca os Sistemas Tutoriais Inteligentes como um dos métodos educacionais mais eficientes.

Os tutores inteligentes são uma tentativa de combinar técnicas da Inteligência Artificial e uma teoria de psicologia de aquisição de conhecimento dentro de um plano efetivo de ensino. Talvez a melhor razão para chamar estes sistemas de *'inteligentes'* seja a sua habilidade de solucionar os mesmos problemas que apresentam ao usuário do sistema. Esta capacidade provê uma base eficiente para a explanação de todos os detalhes da solução do problema. Outra característica dos Sistemas Tutoriais Inteligentes é o fornecimento de um grau de instrução individualizado, graças à elaboração de um modelo distinto do que o usuário sabe, através da interpretação de como ele se comporta.

Oliveira [OLIVEIRA 88] argumenta que os mecanismos educacionais que não incorporam alguma técnica da Inteligência Artificial podem incluir modelos muito sofisticados de conceitos e operações em seu domínio da matéria, mas não *'entendem'* nada sobre o usuário, matéria ou abordagem pedagógica, ou seja, os Sistemas Tutoriais Inteligentes devem *'entender'* o que, para quem e como

ensinar, e devem adaptar conteúdo e método às necessidades individuais de cada usuário, sem serem limitados a um repertório de respostas pré-especificadas.

Com o intuito de entender melhor os Sistemas Tutoriais Inteligentes, enumera-se algumas de suas propriedades desejáveis [OLIVEIRA 88]:

a. O tutor faz com que as heurísticas de solução de problemas do usuário convirjam para as dele;

b. O tutor deve aprender e adotar métodos de solução do usuário se eles forem superiores aos seus;

c. O tutor escolhe exemplos e problemas apropriados para o usuário;

d. Quando o usuário precisa de ajuda, o tutor pode recomendar esquemas de solução e demonstrar como aplicar técnicas;

e. O tutor pode apresentar exemplos arbitrários escolhidos pelo usuário;

f. O tutor está apto a adaptar-se a diferentes níveis de conhecimento dos usuários;

g. O tutor está apto a medir o progresso dos usuários;

h. O tutor pode rever com o usuário o material aprendido anteriormente quando surge a necessidade;

i. O tutor deverá apontar imediatamente os erros quando estes surgirem, enquanto permite ao usuário decidir livremente como resolver um problema;

j. Após o usuário resolver um problema, o tutor pode mostrar soluções mais diretas que utilizem técnicas ou teoremas aprendidos mais recentemente.

As propriedades apresentadas, acima, são comuns aos Assistentes Inteligentes, reforçando o fato da existência de conceitos comuns. Demonstra-se, assim, a idéia de que os Sistemas Tutoriais Inteligentes e os Assistentes Inteligentes estão ocupando um mesmo grupo de estudo, denominado por Wenger [WENGER 87] de Sistemas de Comunicação de Conhecimento.

Wenger [WENGER 87] concebeu os chamados Sistemas de Comunicação de Conhecimento, onde *'comunicação de conhecimento'* é definida como a *"habilidade para causar e/ou suportar a aquisição de conhecimento de alguém por outra pessoa, via um conjunto restrito de operações de comunicação"*.

3.2. Componentes dos Sistemas de Comunicação de Conhecimento

Em consequência à natureza experimental dos trabalhos neste campo, ainda não se identificou uma arquitetura padrão para estes sistemas. A figura 3 apresenta um exemplo de Sistema Tutorial Inteligente com seus módulos e suas interconexões. Entretanto, existe um conjunto de componentes típicos:

a. A *base de conhecimento*, que contém a representação do conhecimento a ser comunicado. Este módulo é capaz de solucionar os problemas nos quais os usuários estão trabalhando;

b. O *modelo do usuário* representa o estado de conhecimento do usuário (aptidões e deficiências). Contém todos os aspectos do comportamento do usuário;

c. As *estratégias de comunicação*, que incluem as técnicas de orientação na execução das tarefas. Escolhe as estratégias para assistir o usuário baseado no estado corrente do modelo do usuário;

d. A *interface*, responsável pela conversão da representação interna em uma linguagem entendida pelo usuário e vice-versa.

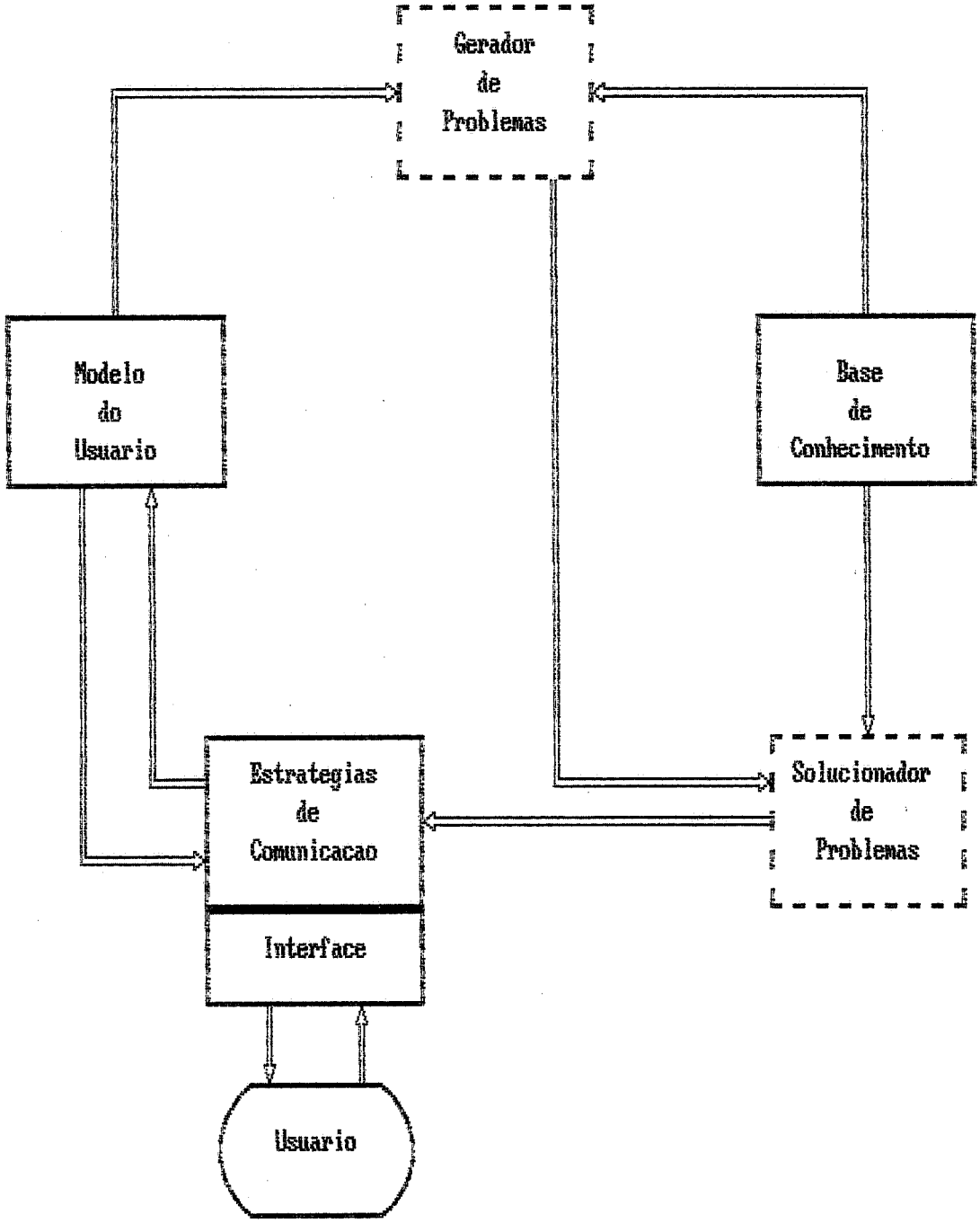


figura 3. Modelo de um Sistema Tutorial Inteligente
[OLIVEIRA 88]

Existem muitas vantagens decorrentes da separação clara dos componentes. Primeiro, do ponto de vista da Engenharia de Software, torna-se possível a construção e teste de cada componente independentemente. Segundo, os componentes podem ser otimizados por suas características distintas. Terceiro, o desenvolvimento de um domínio livre da teoria de instrução.

Nas próximas seções, os conceitos dos componentes são apresentados em caráter preliminar, uma vez que eles estão detalhados no decorrer da tese. Ao final, a figura 4 sumariza as principais questões na comunicação de conhecimento.

3.2.1. Base de Conhecimento

Nos Sistemas de Comunicação de Conhecimento existe um módulo especial, também denominado inteligente, que contém a representação do conhecimento a ser comunicado. Em muitos casos, esta representação do conhecimento não é apenas uma descrição dos vários conceitos e habilidades que o usuário deve adquirir, como em um *"curriculum"*, mas um modelo atual que pode ser executado em um domínio particular e conseqüentemente fornece ao sistema uma forma dinâmica de inteligência. O conhecimento pode ser adquirido em muitas fontes - livros, artigos, base de dados, mas principalmente na experiência profissional.

O módulo inteligente compreende uma dupla função. Por um lado, atua como fonte para o conhecimento a ser apresentado. Isto inclui geração de explicações e respostas para o usuário. Por outro lado, o módulo inteligente serve como um padrão para avaliar o desempenho do usuário.

A estrutura interna do módulo inteligente pode ser aberta para a inspeção e capaz de suportar explicações de suas ações e conclusões. Módulos inteligentes podem ser classificados ao longo de um espectro, desde 'completamente opaco ou *'caixa-preta'*, onde apenas os resultados finais estão disponíveis, até

completamente transparente ou *'caixa-branca'*, onde cada etapa do raciocínio pode ser inspecionada e interpretada.

O módulo inteligente, por necessidade, corporifica uma visão específica do domínio. A linguagem de representação utilizada e os conceitos estabelecidos como primitivos são escolhas feitas pelo projetista que influenciam a apresentação inteira.

O conhecimento incorreto deve ser representado a fim de refletir o processo pelo qual aparecem os erros. Extensões para acomodar o conhecimento incorreto podem tomar várias formas. Uma solução é juntar uma grande quantidade de informações sobre erros prováveis para um dado domínio e uma dada população de usuários e incluir estas distorções observadas como primitivas da linguagem modelada. O sistema seleciona destes modelos de conhecimento correto e incorreto, elementos que justificam o comportamento do usuário. Para aplicações correntes, o desenvolvimento laborioso de tais Catálogos de Erros para vários domínios é uma atividade importante dos projetistas.

3.2.2. Modelo do Usuário

Nenhuma comunicação pode existir sem um certo entendimento do receptor. Desta forma, equivalente à idéia de representar explicitamente o conhecimento a ser transmitido veio a idéia de fazer a mesma coisa com o usuário, na forma de um **Modelo do Usuário**. Este modelo deve incluir todos os aspectos do comportamento do usuário e o conhecimento que tem repercussões no seu desempenho. De qualquer maneira, a tarefa de construir tal modelo não é simples para um sistema baseado em computador.

A adaptabilidade do sistema é determinada pelo alcance e exatidão das informações contidas no modelo do usuário. Com as técnicas da Inteligência Artificial torna-se possível inferir aspectos não observáveis do comportamento do usuário a fim de produzir uma interpretação de suas ações e reconstruir o conhecimento que deu origem a estas ações.

O modelo do conhecimento do usuário deve estar presente fora da representação do conhecimento do sistema. Assim, o **estado de conhecimento do usuário** pode ser comparado com o conhecimento do módulo inteligente e a instrução adaptada para exercitar porções de conhecimento que o modelo demonstra estarem mal entendidas.

O comportamento incorreto nem sempre origina-se de um conhecimento incompleto. Portanto, um modelo do usuário mais informativo deve ser capaz de prover uma representação explícita de versões incorretas do conhecimento do usuário e conseqüentemente as ações remediáveis que podem ser tomadas. Além disso, o modelo do usuário pode conter informações explicatórias sobre o aparecimento dessas porções incorretas de conhecimento.

Uma característica importante de um modelo do usuário é ser *'executável'*. Um modelo do usuário executável pode fornecer prognósticos sobre o comportamento de um usuário particular em um contexto particular. Esta simulação permite a verificação de hipóteses precisas como a natureza dos erros do usuário.

O processo de formar e atualizar o modelo do usuário pela análise dos dados disponíveis ao sistema é denominado **diagnóstico**. Valores numéricos que avaliam o desempenho podem ser atualizados com simples computações estatísticas. Ao contrário, o diagnóstico detalhado requer usualmente a reconstrução da estrutura de objetivos para interpretar o comportamento do usuário. Uma abordagem para modelar o conhecimento do usuário - denominada **modelagem diferencial** - visualiza o usuário como um subconjunto do conhecimento representado no sistema. Assim, a representação interna do modelo torna-se um conjunto de hipóteses do conhecimento do usuário sobre o módulo inteligente.

3.2.3. Estratégias de Comunicação

Até recentemente, a idéia que as estratégias de comunicação poderiam ser representadas explicitamente teve pouca atenção em comparação com a representação do domínio. Entretanto, concluiu-se que somente *'inteligência'* não era suficiente para suportar as estratégias de comunicação. O processo de comunicação é organizado através da integração de novos materiais e novas experiências dentro do conhecimento que detém o usuário. Isto requer pedaços de informação que são utilizados especificamente para as estratégias de comunicação, embora por natureza pertençam ao conhecimento do domínio. Por exemplo, relações e medidas da dificuldade relativa são cruciais para a flexibilidade na sequência instrucional.

Em um sistema transparente ideal, este conhecimento pode ser armazenado na forma de princípios gerais expressos declarativamente e interpretado dentro de decisões atuais. Representações explícitas das estratégias de comunicação geram o potencial para sistemas adaptarem e melhorarem suas estratégias a todo momento e criam a oportunidade de reutilizar componentes em outros domínios.

Este módulo determina quando uma intervenção é necessária, se o usuário pode ser interrompido em uma atividade, e o que pode e deve ser dito ou apresentado em um dado momento. Em suma, especifica a orientação na execução das tarefas, as explicações do fenômeno e do processo e as medidas remediáveis.

3.2.4. Interface

Enquanto o módulo estratégias de comunicação decide o oportunismo e o conteúdo das ações, o módulo interface elabora a sua forma final. Mais genericamente, traduz a representação interna do sistema para a linguagem de interface entendida pelo usuário e vice-versa. Embora o módulo interface opere em cooperação com o módulo modelo do usuário e o módulo estratégias de comunicação, suas decisões são de natureza diferente e requerem um tipo diferente de conhecimento. é, portanto, útil identificar a interface como um componente distinto.

Um aspecto da questão da interface tradicionalmente ligado à pesquisa de Inteligência Artificial é o processamento da linguagem natural. Embora o uso livre da linguagem natural ainda pertença ao futuro, sistemas atuais variam na habilidade de lidar com expressões verbais. Interfaces variam no uso de menus fixos com perguntas de múltipla escolha até um tratamento razoavelmente livre de uma linguagem pseudo-natural.

Base de Conhecimento

. Funcao:

modelo X curriculum
fonte e padroes
solucoes multiplas

. Comunicabilidade:

informacao curricular
transparencia
visoes particulares

Modelo do Usuario

. Informacao:

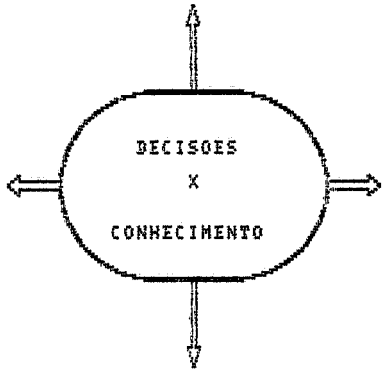
interpretacao do comportamento
estado do conhecimento
conceitos erroneos

. Representacao:

primitivas neutras
primitivas de erros
executavel

. Processo de diagnostico:

dirigido por modelos e dados
"top-down" ou "botton-up"
robusto
passivo ou ativo
inferencial ou interativo



Estrategias de Comunicacao

. Processo Didatico:

regras X principios
decisoes globais: sequencia
decisoes locais: intervencoes
guias
explicacoes
remediacoes

. Grau de Controle:

monitoracao estrita
iniciativa mista
orientacao

Interface

. Funcao:

capacidade conversacional
processamento de linguagem
representacao grafica

. Qualidades:

clareza na apresentacao
facil de usar e atrativa
capacidades explicitas

figura 4. Algumas Questoes Basicas na Comunicacao de Conhecimento

[WENGER 87]

3.3. Sistemas Tutoriais Inteligentes com Orientação

Segundo o trabalho de Oliveira [OLIVEIRA 88], identificam-se cinco paradigmas que constituem o domínio dos Sistemas Tutoriais Inteligentes:

a. *Sistemas Tutoriais Inteligentes com diálogo de iniciativa mista:*

Neste caso, tanto o usuário como o tutor podem tomar a iniciativa do diálogo. O tutor tenta ensinar ao usuário por meio de um diálogo socrático de descoberta guiada;

b. *Sistemas Tutoriais Inteligentes com orientação:*

O tutor observa o desempenho do usuário e dá conselhos que o auxiliem a melhorar esse desempenho;

c. *Sistemas Tutoriais Inteligentes de diagnóstico:*

O tutor é dirigido por um catálogo de erros que identifica os erros que o usuário possa cometer na solução de um dado problema;

d. *Micromundos:*

Compreendem o desenvolvimento de ferramentas computacionais que permitem ao usuário explorar uma matéria, tal como geometria, física ou música. O exemplo mais conhecido deste paradigma é a linguagem LOGO;

e. *Sistemas Especialistas articulados:*

Um sistema especialista com capacidade de explicação refinada pode ser utilizado para prover o usuário de prática em solução de problemas e em tomadas de decisão.

No nosso trabalho, os Sistemas Tutoriais Inteligentes com orientação inspiram estudos mais profundos, devido à sua filosofia de prover assistência não intrusa, liberando assim o usuário para o desenvolvimento de suas atividades.

Partindo desta premissa, dois sistemas são apresentados preliminarmente dentro desta linha: WEST [WENGER 87] [OLIVEIRA 88] e WUSOR [WENGER 87]. Em ambos a atividade envolvida é um jogo. De qualquer maneira, ambientes de jogos foram escolhidos simplesmente por sua simplicidade conceitual e a sua motivação intrínseca - os princípios apresentados são aplicáveis a qualquer atividade.

Em seguida, são apresentados dois Sistemas Tutoriais com orientação, sem características de um ambiente de jogos: Tutor de Geometria [ANDERSON 85a] e Tutor LISP [ANDERSON 85b] [ANDERSON 89] - onde o objetivo é, respectivamente, o ensino de provas de teoremas em geometria e o ensino da linguagem de programação LISP.

As idéias que baseiam muitas das pesquisas em Sistemas Tutoriais com orientação podem ser caracterizadas como aprendizagem guiada por descoberta. Assume-se que o usuário constrói seu entendimento de uma situação baseado em sua experiência. De acordo com esta

teoria, a noção de concepção errônea tem papel central no processo de aprendizagem. Se o usuário tem bastante informação para determinar o que causou o erro e pode corrigí-lo, o erro é dito como construtivo [OLIVEIRA 88].

Contudo, a arte da orientação é sutil. Intervenções devem ser efetivas na prevenção de erros sem destruir o interesse na atividade. Orientação difere de outras situações tutoriais, pois o usuário está sempre com o controle total da atividade. A tarefa de diagnóstico é dificultada porque a mesma toma lugar atrás das cenas, mas em certas situações duvidosas o tutor pode adotar uma atitude conservadora e simplesmente ficar em silêncio.

A figura 5 apresenta um típico Sistema Tutorial Inteligente com orientação. O módulo conhecimento do domínio, que armazena o conhecimento do perito, é capaz de solucionar os mesmos problemas manipulados pelo usuário. Outra função é responder às questões do usuário.

Comparando as soluções do usuário com as soluções geradas pelo conhecimento do perito, o diagnosticador constrói hipóteses sobre as aptidões e deficiências do usuário. Estas hipóteses refletirão no estado corrente do modelo do usuário. O trabalho das estratégias de comunicação é manter um processo individual de comunicação baseando-se no estado de conhecimento do usuário.

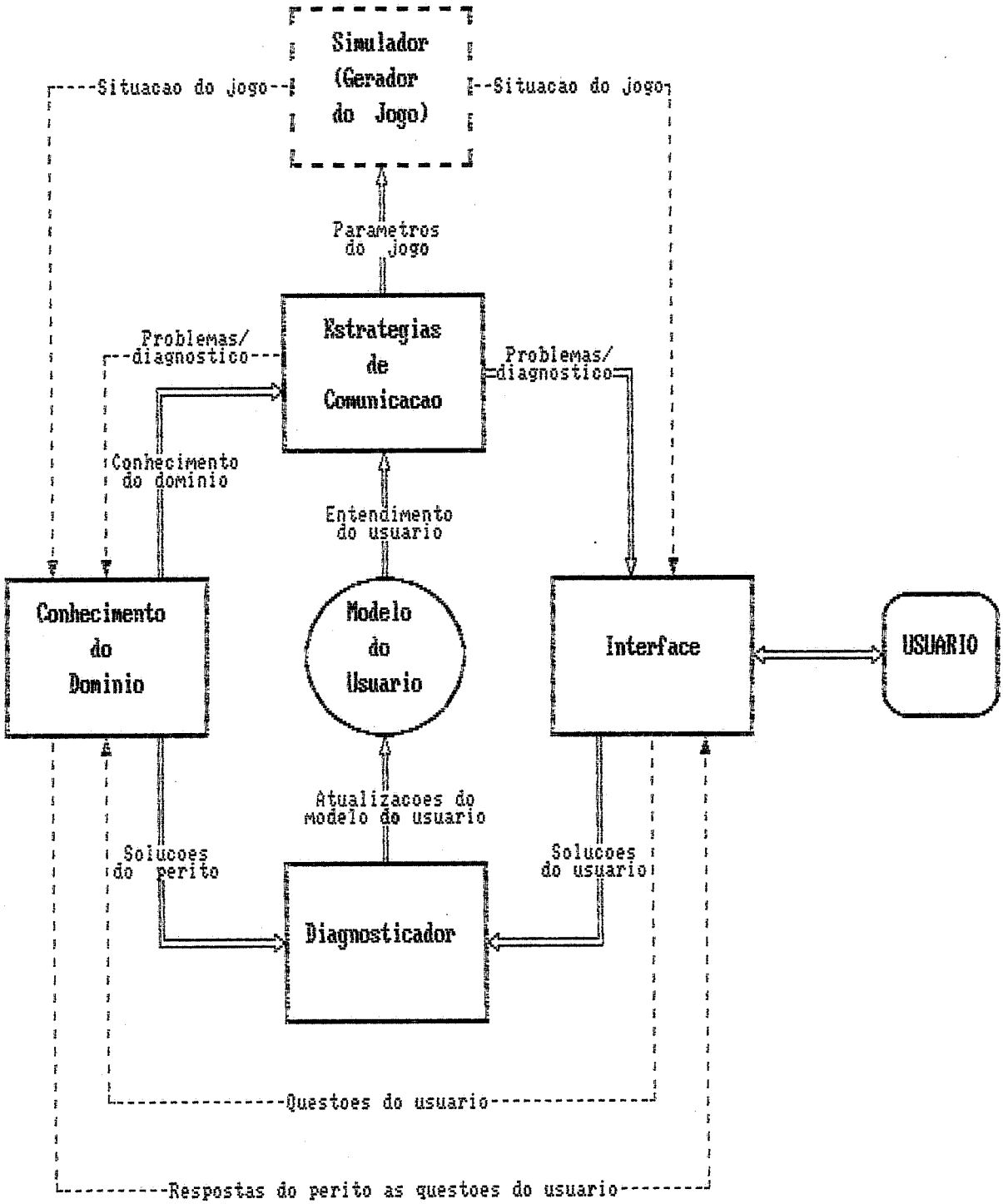


figura 5. Os Principais Componentes de um Sistema Tutorial Inteligente com Orientacao [KEARSLEY 87]

3.3.1. Projeto WEST

O projeto WEST [WENGER 87] [OLIVEIRA 88] teve início no contexto do projeto SOPHIE (tutor que simula como um circuito elétrico particular comporta-se sob diferentes cargas) [LAMSWEERDE 87], cujo objetivo original era incluir a técnica de orientação. Uma vez que o domínio da eletrônica parecia muito complexo para uma primeira investigação na arte da orientação, o domínio escolhido para este propósito foi um jogo computacional.

O propósito do jogo é exercitar as habilidades aritméticas do usuário. O jogador é envolvido com uma corrida entre cidades. Ele recebe três números randômicos, com os quais deverá compor uma expressão artimética envolvendo dois operadores diferentes e o resultado determina o número de espaços a percorrer. Entretanto, o jogo foi projetado para que a obtenção do maior número nem sempre signifique a melhor estratégia. Logo, o jogador é forçado a explorar diferentes maneiras de combinação dos números com os operadores aritméticos.

O termo orientação descreve um ambiente no qual o usuário é envolvido em uma atividade, na qual o programa instrucional opera, ocasionalmente, fazendo críticas e sugestões para o melhoramento do desempenho do usuário. Esta pesquisa procurou identificar [OLIVEIRA 88]:

a. Estratégias de diagnóstico requeridas para inferir deficiências no entendimento do usuário a partir do seu comportamento esperado;

b. Várias estratégias tutoriais para dirigir o tutor a dizer a coisa certa no tempo certo.

Burton e Brown [WENGER 87] propuseram um paradigma geral para a técnica baseada em orientação, denominada *"questões e exemplos"*, cujos diagramas de fluxo de dados são apresentados na figura 6.

O conhecimento a ser transmitido é descrito como um conjunto de questões, que são apresentadas para o usuário quando tornam-se relevantes para o jogo, através de exemplos concretos pertinentes a movimentação corrente. Encontrar questões relevantes e exemplos apropriados depende da noção de modelagem diferencial, onde o comportamento do usuário é analisado em relação às escolhas de um perito para a mesma circunstância.

Devido à natureza randômica do jogo, é impossível determinar de antemão quais as questões que serão necessárias. Logo, o conhecimento do usuário das várias questões não pode ser determinado por suas

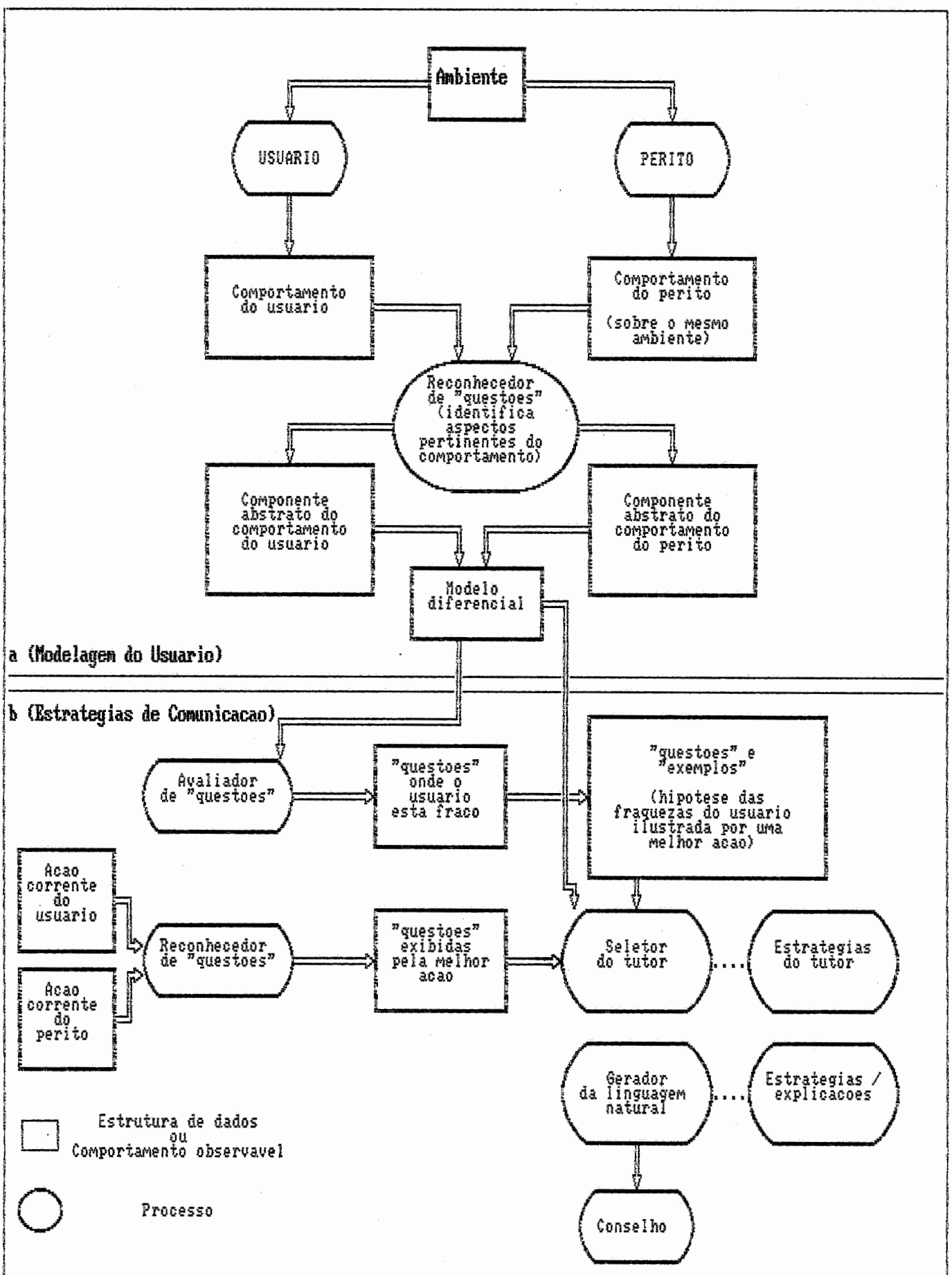


figura 6. Diagramas do Fluxo de dados do Tutor WEST

[WENGER 87]

ocorrências absolutas no jogo, mas apenas pela comparação com as decisões de um perito na mesma ação.

Quando o usuário joga, o módulo inteligente gera uma lista ordenada de todas as ações possíveis. Se a expressão do usuário não fornece a ação ótima do módulo inteligente, o processo de diagnóstico é inicializado. Primeiro, a ação do usuário é analizada pelo reconhecedor de questões para determinar quais as *questões* utilizadas. A seguir, todas as ações do módulo inteligente que são melhores que as ações do usuário são também analisadas pelo reconhecedor de questões, a fim de produzir uma lista de questões onde o usuário falhou. As diferenças entre as listas de questões aplicadas pelo usuário e pelo perito fornecem informações para atualizar o modelo do usuário. No WEST, o modelo diferencial armazena como cada questão foi empregada apropriadamente e como cada questão não foi percebida. Não existe a noção de questões erradas. O WEST adota uma atitude cautelosa e interrompe apenas quando existe uma boa evidência de deficiências no usuário.

O paradigma de questões e exemplos propõe um mecanismo para determinar quando e como o conhecimento pode ser apresentado pelo orientador para motivar o aprendizado no desempenho da atividade. Se existe uma evidência da deficiência, são mencionadas as questões críticas utilizadas erroneamente. Então são introduzidos exemplos concretos na forma de soluções melhores para os movimentos correntes.

3.3.2. Projeto WUSOR

O projeto WUSOR [WENGER 87] começou com um curso de tecnologia educacional no MIT. A principal inspiração foi aparentemente fornecida pelo trabalho com o WEST. Ira Goldstein [LAMSWEERDE 87] desenvolveu um Sistema Tutorial Inteligente com orientação para o jogo WUMPUS, adaptando-o para o ensino de raciocínio probabilístico. De uma maneira similar ao WEST, Goldstein utilizou regras de produção para estabelecer os princípios de aprendizagem.

Este jogo leva o jogador através de sucessivas cavernas onde o terrível Wumpus está escondido. Além do Wumpus, outros perigos estão escondidos: buracos mortais e morcegos podem ser encontrados em qualquer caverna. Sempre que o jogador está procurando uma nova caverna, uma lista das cavernas vizinhas é fornecida, juntamente com alguns conselhos: um chiado revela a presença de um buraco ou morcego; o próprio Wumpus pode ser reconhecido por uma diferença de duas cavernas.

Para ganhar o jogo, o jogador deve lançar uma de suas cinco flechas dentro da caverna do Wumpus. O jogador pode perder pela queda em um buraco, por caminhar dentro da caverna do Wumpus ou pelo uso de todas as flechas sem acertar nenhum alvo.

A primeira versão do tutor, denominado WUSOR-I [WENGER 87], consiste apenas de um módulo inteligente e as estratégias de comunicação. No módulo inteligente, o conhecimento do domínio é representado na forma de regras de produção. Estas regras são organizadas dentro de especialidades para cada perigo, que cooperam para classificar todas as ações possíveis e selecionar a melhor. Este módulo consiste de regras heurísticas para a probabilidade aproximada, ao invés de computações cansativas. Isto não se deve apenas a uma questão de eficiência, mas porque esta técnica está mais de acordo com a forma dos seres humanos raciocinarem.

O módulo estratégias de comunicação intervém a cada momento que o usuário não escolhe a ação ótima de acordo com a classificação do módulo inteligente. Associado com cada categoria de ação e com cada regra está um conjunto específico de explicações que são enviadas ao usuário. O WUSOR não mantém um modelo do usuário. Esta forma rudimentar influencia a explicação dada e ocasiona a supressão de algumas regras, a fim de evitar que iniciantes sejam inundados com material avançado.

Da necessidade de diagnosticar o estado de conhecimento do usuário e adaptar suas intervenções, nasceu o WUSOR-II [WENGER 87], onde foi proposta a teoria *"overlay"* para a modelagem do usuário, que tornou-se um paradigma padrão nos Sistemas Tutoriais Inteligentes.

O paradigma *"overlay"* é aplicável sempre que o conhecimento pode ser expresso através de um conjunto de

regras. Basicamente, é quase similar ao modelo diferencial do projeto WEST. Também faz correspondência entre as avaliações individuais e unidades simples de conhecimento através da comparação do comportamento do usuário e o suposto comportamento de um perito na mesma situação. Uma vez que estas unidades de conhecimento são partes do próprio módulo inteligente, o estado de conhecimento do usuário é visualizado como um sub-conjunto do conhecimento de um perito: daí a expressão "overlay". O modelo "overlay" do projeto WUSOR-II tem um módulo simples que determina quando uma habilidade individual pode ser considerada como adquirida. Para esta decisão, este módulo analisa a história das variações na avaliação dos parâmetros para esta habilidade. Assim, caso a história de uma habilidade indique mudanças constantes poderá significar a manifestação de inconsistências e descontinuidades.

O modelo do usuário *"overlay"* está integrado dentro de uma arquitetura completamente baseada em regras para um tutor inteligente, ilustrada no diagrama da figura 7. O módulo inteligente com suas regras de habilidade pode sugerir e analisar movimentos. O módulo diagnóstico constrói e atualiza o modelo do usuário utilizando suas regras de evidência. O módulo estratégias de comunicação, com suas regras de explanação, organiza as intervenções ao usuário de acordo com o modelo mantido

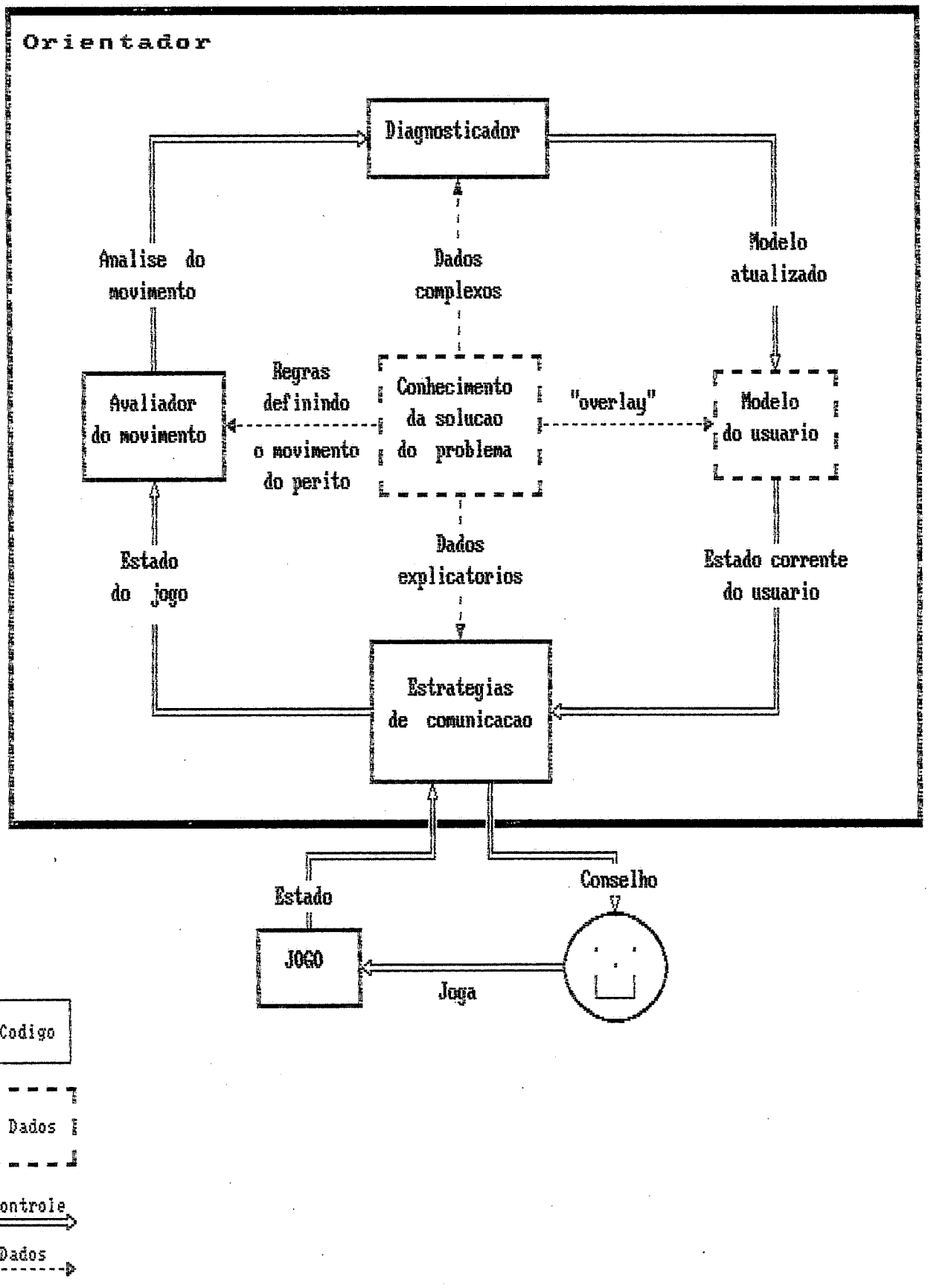


Figura 7. Arquitetura do MUSOR - II

[WENGER 87]

pelo módulo diagnóstico. Finalmente, regras de linguagem são aplicadas para criar uma interface adequada.

Carr e Goldstein [WENGER 87] propuseram um modelo *"overlay"* mais elaborado que inclui a receptividade do usuário às várias estratégias de comunicação. Do mesmo modo que o conhecimento do usuário é representado como um *"overlay"* das regras do perito, as preferências pelos modos de comunicação podem ser descritas como um *"overlay"* das regras das estratégias de comunicação. Para atualizar esta parte do modelo, o módulo diagnóstico deve cotar as várias regras de acordo com a receptividade e efetividade individuais.

Existem duas limitações importantes ao modelo *"overlay"* do usuário. O primeiro é um exemplo do problema geral de representação do conhecimento, ampliado aqui devido ao conhecimento do usuário ser visualizado como um sub-conjunto do conhecimento do perito. O modelo *"overlay"* falhará caso o domínio permita múltiplos paradigmas de solução do problema e o usuário seguir alguma solução não programada, possivelmente realizando bons movimentos que o sistema considerará como *'não ótimos'*.

A segunda dificuldade é específica do paradigma de modelagem *"overlay"*. Este paradigma assume que o comportamento *'não ótimo'* é causado exclusivamente por conhecimento insuficiente e que as distorções das habilidades corretas não ocorrem. Isto é uma simplificação irreal porque, em muitos domínios, a modelagem do usuário requer que as regras incorretas e as aplicações não apropriadas das regras corretas sejam descritas especificamente.

3.3.3. Tutor de Geometria

O Tutor para provas em Geometria [ANDERSON 85a] consiste de três componentes básicos:

a. Um módulo com um conjunto de regras ideais e regras de erros (*"Ideal and Buggy Rules"* ou IBR), a *base de conhecimento*;

b. Um módulo de *diagnóstico*;

c. Um módulo de *interface*.

A estratégia empregada envolve o acompanhamento do comportamento do usuário em termos das regras instanciadas, corrigindo o usuário quando seu comportamento desvia-se um mínimo do estabelecido e auxilia o usuário a superar os obstáculos.

A efetividade do tutor requer um modelo interno de como as tarefas podem ser executadas e a capacidade de acessar este modelo para propósitos de instrução. Tal modelo é denominado **modelo ideal**. O tutor necessita suplementar este modelo com vários erros que um usuário pode cometer no desvio do modelo ideal. Este modelo é *denominado modelo de erros*.

Essencialmente, o tutor esboça o comportamento do usuário através de seu modelo ideal e de erros. Em qualquer ponto existem produções no tutor que podem ser aplicadas. O módulo diagnóstico infere qual regra o usuário executou pela determinação de qual regra

corresponde ao resultado do usuário. Se é uma produção correta, o tutor permanece quieto e continua a esboçar a solução do problema pelo usuário. Se uma produção incorreta foi aplicada, o tutor interrompe com a instrução de reparo adequada. As possibilidades finais podem ser quando o usuário não sabe o que fazer em seguida ou o comportamento do usuário não corresponde a nenhuma produção correta ou incorreta. Usualmente, isto ocorre quando o usuário está muito confuso. A melhor coisa a fazer, em tais situações, é comunicar ao usuário a próxima etapa. Se isto é explicado adequadamente, o usuário frequentemente é capaz de voltar ao caminho correto.

Os conjuntos de regras ideais e regras de erros estão baseados em análise teórica considerável das características do domínio do problema e uma grande observação empírica do comportamento do usuário. É razoável supor que elas são mais sofisticadas do que as possuídas pelos tutores humanos. Também, tem-se maior acesso ao conhecimento codificado nestas regras do que em tutores humanos, onde o modelo ideal é frequentemente sub-entendido e incapaz de ser diretamente descrito para o usuário.

O componente central é o módulo diagnóstico. Ele define a estratégia de tutoria corrente. Este módulo utiliza informações das regras do IBR e do usuário (via interface) para controlar a interação tutorial. Existem dois meios de interação com o IBR:

a. Pode-se olhar quais regras ideais e regras de erros estão sendo instanciadas e utilizar isto para interpretar o comportamento do usuário. Ele permite traçar a solução do usuário pela seleção de uma das regras de produção.

b. Pode-se solicitar ao modelo ideal se uma afirmação pode ser provada, sujeita a certas restrições. Isto levará o IBR a tentativa de uma prova e retornará informações tais como a existência da prova, o tamanho da prova, quais regras envolve, etc.

O Usuário deve galgar três estágios para completar uma prova de geometria:

a. Selecionar um conjunto de assertivas que farão a inferência;

b. Especificar as regras de inferência que se aplicarão a estas assertivas;

c. Especificar as assertivas que resultam da aplicação das regras de inferência.

Na etapa de seleção de assertivas o usuário pode selecionar um conjunto de assertivas para aplicar uma regra de inferência ou solicitar auxílio. Caso o usuário solicite auxílio, o tutor escolhe as assertivas mais cotadas do módulo IBR, sujeitas às restrições que a

inferência produziu pela regra que é atualmente parte da prova. Isto provê o usuário de assertivas que são partes da regra. Caso o usuário selecione um conjunto de assertivas e elas representem a instanciação de qualquer regra correta o sistema aceita, mesmo se está pouco cotada ou que não faça parte da prova.

Se o usuário seleciona um conjunto de assertivas que não fazem parte de nenhuma regra, o tutor comunica este detalhe ao usuário, e o usuário volta a selecionar. Se o usuário, novamente, escolher um conjunto incorreto, o tutor apresenta o mesmo conjunto de assertivas caso o usuário tivesse solicitado auxílio. O tutor trata a etapa de seleção de regras e a etapa de resultados de aplicação das regras de maneira análoga.

O tutor tenta guiar o usuário para a solução com uma intervenção mínima e aceita qualquer inferência legal que o usuário pretenda realizar. Entretanto, deve-se prevenir para que o usuário não se perca em uma confusão de inferências legais, mas inúteis para a solução do problema. A implicação desta capacidade é a habilidade do tutor questionar o IBR quando determinada inferência não esperada é parte de uma prova aceitável.

O último componente é a interface. O maior esforço no projeto da interface é comunicar ao usuário a estrutura lógica de uma prova e a estrutura do processo de solução do problema pelo qual a prova é gerada. Na fase inicial do estabelecimento do problema, o enunciado a ser provado fica no topo da tela, os dados na base e o diagrama na lateral esquerda superior. O usuário pode

raciocinar *'para frente'* a partir dos dados e *'para trás'* a partir do enunciado a ser provado. Cada etapa da inferência envolve um conjunto de premissas, uma justificativa e uma conclusão. No raciocício *'para frente'*, o usuário aponta para as premissas, estabelece a justificativa, e aponta para a conclusão ou estabelece esta conclusão. No raciocínio *'para trás'*, o usuário aponta para a conclusão, estabelece a justificativa e então prova as premissas.

3.3.4. Tutor LISP

O artigo de Anderson [ANDERSON 85b] aborda o desenvolvimento de um Tutor para a linguagem LISP. LISP é uma das principais linguagens de programação da Inteligência Artificial.

O Tutor LISP ostenta os seguintes componentes:

a. Uma *base de conhecimento*, que pode solucionar os problemas correntes;

b. Um *catálogo de erros*, que contém todos os desvios possíveis que um usuário pode cometer em um comportamento ideal;

c. As *estratégias de comunicação*, que decidem quando interromper o processo de solução do problema e o que dizer, e estimam quais os problemas que o usuário deve fazer e quando deve avançar para novo material;

d. Um *modelo do estado de conhecimento do usuário*, que determina o conhecimento e os erros do usuário a partir do seu comportamento, permitindo estabelecer uma instrução individualizada;

e. Uma *interface* para a comunicação com o usuário. Sua construção requer decisões de engenharia humana em relação a como apresentar as informações de maneira inteligível, como questionar o usuário, como o usuário deve colocar suas perguntas, quais informações devem ser mantidas no vídeo.

A fundamentação da construção do Tutor LISP é simples [ANDERSON 89]: *"O usuário deve ser capaz de trabalhar em um problema em um ambiente amigável. Entretanto, sempre que cometer um erro ou solicitar auxílio, o tutor deve providenciar informações úteis que guiem o usuário pelo caminho correto da solução do problema"*.

A fim de monitorar o progresso do usuário, descobrir e instruir sobre erros, o tutor deve ser capaz de solucionar o problema que o usuário está trabalhando. Por este motivo, o primeiro componente do tutor - a base de conhecimentos - representa o modelo ideal, uma simulação do conhecimento de programação ideal utilizado na solução dos problemas.

Cada regra de solução de problema é representada no sistema como uma regra de produção. Cada regra de produção contém uma parte IF, que é um conjunto de condições utilizadas para determinar se a regra é aplicável e uma parte THEN, que especifica o que fazer nesta situação. Abaixo, apresenta-se um exemplo de uma regra de produção:

IF *o objetivo é combinar a LISTA-1 e LISTA-2 em uma simples lista*

THEN *use a função APPEND e estabeleça como sub-objetivo a codificação de LISTA-1 e LISTA-2*

Esta regra de produção representa o uso da função APPEND, cujo objetivo é construir uma lista a partir de duas listas. Quando o modelo ideal codifica uma função LISP, ele aplica muitas regras de produção como a apresentada acima para planejar e escrever o código. Ele também contém um catálogo de erros, que representa conceitos errôneos de programadores novatos desenvolvidos durante a aprendizagem.

O modelo ideal representa o conhecimento que o usuário deve adquirir. Mas o tutor tem que representar o que o usuário sabe ou não sabe e sua técnica para cada problema particular. Ele tenta interpretar o que o usuário está fazendo no estado corrente do problema, e tenta construir um modelo com o estado de conhecimento do usuário. Isto é feito através de um modelo estatístico, ou seja, para cada produção o tutor mantém uma estimativa de seu conhecimento.

O tutor segue o usuário enquanto ele digita seu código, símbolo por símbolo, e tenta calcular qual regra de produção correta ou incorreta conduziria a entrada. Se a regra encontrada está correta, então o tutor permanece em silêncio e espera por mais entradas. Por outro lado, caso a entrada esteja errada, o tutor interrompe com conselhos.

Quando o tutor descobre que o usuário está tendo muita dificuldade na codificação do problema, ele utiliza um exemplo, trabalhando-o através do algoritmo com o usuário, etapa por etapa. Depois de construir o algoritmo, o usuário pode retornar para a codificação, presumivelmente, com uma idéia melhor do que ele pode fazer para que seu código trabalhe satisfatoriamente.

O tutor foi projetado com um grande compromisso com o *"feedback"* imediato. Tão logo o usuário cometa um erro, o tutor responde com uma mensagem do diagnóstico apropriado. O tutor também fornece guia por insinuações da solução correta, caso o usuário apresente dificuldades. Estas insinuações tomam a forma de perguntas e recados sobre os objetivos correntes. Se for necessário, o tutor pode fornecer a próxima porção de código para que o usuário possa continuar. Em resumo, o Tutor LISP pode ser particionado em ciclos [ANDERSON 89]:

a. O tutor estabelece um objetivo de codificação;

b. O usuário especifica a unidade de código correspondente;

c. O tutor seleciona a entrada como correta ou incorreta (ou como um pedido de ajuda) e responde de acordo. Se a resposta é correta, o tutor estabelece um novo objetivo. Se a resposta está incorreta, o tutor imediatamente notifica o usuário, provê *"feedback"* e estabelece o mesmo objetivo no próximo ciclo. Se o usuário solicita uma explanação ou está em dificuldades naquele objetivo, o tutor provê a resposta correta e estabelece um novo objetivo no próximo ciclo.

O Tutor LISP contém aproximadamente 325 regras de produção sobre o planejamento e codificação de programas LISP e 475 versões de erros dessas regras, que mostraram-se efetivos no diagnóstico de 45 a 80% dos erros do usuário, dependendo da complexidade da lição.

4 ASSISTENTES INTELIGENTES

4.1. Conceituação

Os Assistentes Inteligentes combinam ferramentas com inteligência. Das técnicas de ferramentas, ganham-se anos de experiência de outros cientistas que construíram e usaram ferramentas e ambientes particulares. De técnicas baseadas em conhecimento, ganha-se estrutura desejável para encorajar ou forçar a utilização das regras e diretrizes que constituem um determinado método. O encorajamento da aplicação do método pode ser *'não ativo'*, facilitando o acompanhamento de regras e diretrizes; ou pode ser *'ativo'*, fornecendo sugestões úteis ao usuário.

O estado da arte dos assistentes não mudou muito na última década. As técnicas atuais para fornecer auxílio a usuários inexperientes e não usuais falham por duas razões principais:

a. Os assistentes correntes não possuem um entendimento real de quem ele está tentando auxiliar (isto é, o que um usuário particular sabe, quais os objetivos prováveis dos usuários), o tópico que ele está fornecendo informações ou as informações que pode prover;

b. Os assistentes não são interativos, o processo de obter e fornecer com sucesso as informações das necessidades do usuário depende se o usuário e o assistente são agentes ativos, ou seja, em qualquer ponto as duas partes devem estar aptas a tomar a iniciativa.

O sucesso de um assistente inteligente dependerá da profundidade do seu entendimento sobre o domínio de aplicação, sobre o sistema do qual fornece informações e sobre o conhecimento do usuário sobre este sistema.

Um assistente apóia o usuário na manipulação e controle de uma atividade automatizada. Um assistente eficiente não apenas responde questões do usuário, mas também *'olha sobre seus ombros'* e o interrompe quando necessário. Logo, a necessidade de auxílio pode ser expressa pelo próprio usuário ou ser inferida pelo assistente. Isto significa que um assistente inteligente deve comportar-se como um orientador humano, que interrompe quando as coisas não estão caminhando bem ou quando existe uma oportunidade de ampliar o conhecimento do usuário, e que é capaz de responder questões no contexto da atividade. A última característica é crucial, pois muitos usuários - em particular usuários iniciantes - não estão advertidos sobre os problemas, e se estão, não sabem como descrevê-lo. Esta é uma razão importante de porque assistentes inteligentes, que não interpretam o desempenho do usuário, possuem uma funcionalidade limitada. A monitoração *"on-line"* do desempenho do usuário envolve muitos problemas

conceituais e computacionais, mas não existem qualitativamente diferenças desses mesmos problemas em Sistemas Tutoriais Inteligentes com orientação.

Finin [FININ 82] esboçou o processo de auxílio em um assistente inteligente, visualizando seis etapas a fim de fornecer ao usuário as informações que necessita:

a. *O usuário necessita de auxílio?*

Deve-se saber quando o usuário necessita de auxílio. O usuário pode descobrir por si próprio e comunicar isto ao assistente. O assistente pode descobrir isto pela observação que o usuário está deparando com erros inesperados, pela observação que o usuário está cometendo um erro catalogado ou pela observação da inatividade inesperada do usuário.

b. *Qual a informação que o usuário necessita?*

O assistente e o usuário devem estabelecer uma descrição inicial das informações que são necessárias. O usuário deve tentar estabelecer isto pela exploração da informação que está disponível no assistente de uma maneira *"top-down"* através do pedido de auxílio mais geral e então refinar a questão. O assistente pode utilizar o modelo do usuário e o contexto corrente para fazer uma suposição do que o usuário provavelmente necessita saber. Outra estratégia que o assistente pode utilizar é fornecer ao usuário um menu, ou um sistema de menus que enumerem todos os tópicos relevantes.

c. *Qual a informação que o assistente pode fornecer?*

O assistente e o usuário necessitam ter algum modelo descrevendo as informações que podem ser dirigidas pelo assistente. O usuário pode descobri-lo por si próprio se o assistente provê facilidades como o *'meta-auxílio'* que forneceria informações sobre o próprio assistente. O assistente pode informar ao usuário sobre a amplitude das informações que pode prover, apresentando sumários de auxílios disponíveis ou enumerando apenas os tópicos que ache relevantes.

d. *Quais as propostas que correspondem às necessidades?*

Uma vez que possuímos a descrição das informações que o usuário necessita e a descrição das informações que o assistente pode prover, necessita-se identificar as melhores correspondências. Em geral, não existirá correspondência *'um-para-um'*. O usuário pode utilizar suas próprias facilidades para comparar as descrições ou o assistente pode utilizar alguma espécie de processo de descrição de correspondências para sugerir estas correspondências.

e. *Como alguém solicita um fragmento particular de informação?*

Uma vez que o usuário decidiu qual a informação que ele necessita e qual das informações oferecidas correspondem às suas necessidades, ele tem que saber como especificar isto para o assistente. Um grande número de modelos é possível para permitir ao usuário descrever as informações que ele quer: nome de tópicos, palavras-chaves, palavras-chaves com qualificadores, linguagens naturais simples e linguagens naturais não restritas. Por outro lado, o assistente pode tomar a iniciativa nesta etapa apresentando um menu das escolhas prováveis ou selecionando a candidata mais provável e simplesmente apresentando-a ao usuário.

f. *O usuário compreendeu o auxílio?*

Uma vez que o usuário recebeu a informação, ele tem que entendê-la. Pode ser um objetivo do assistente dimensionar, quando possível, as informações apresentadas ao usuário tendo em vista o nível de experiência do usuário e a experiência anterior. O assistente deve verificar se o usuário entendeu corretamente as informações fornecidas. O usuário deve estar apto a pedir explanações e exemplos. O assistente deve estar preparado para oferecer informações de segundo plano (por exemplo, definição de termos) e apresentar alternativas da mesma informação.

Dois tipos principais de assistentes estão descritos na literatura [LUCENA 87]: No primeiro tipo, o assistente

é ativo permanentemente e tenta reconhecer na sequência de ações do usuário um plano típico, ou seja, procura inferir qual o objetivo a que ele se propõe. Quando uma sequência de comandos diverge do plano reconhecido, o assistente intervém espontaneamente para propor uma estratégia mais adequada à situação. No segundo tipo, o assistente não fica ativo a não ser sob demanda do usuário, ou seja, a todo instante no curso de seu trabalho, este pode solicitar ao assistente a sua interferência para apoio na obtenção de um certo resultado.

O presente trabalho propõe-se a estudar o primeiro tipo de assistente inteligente - assistente ativo -, denominado por Goldstein [WINSTON 79] de *"Intelligent Computer Coaches"*, devido às grandes semelhanças conceituais com os Sistemas Tutoriais Inteligentes com orientação - *"Coaching Systems"*.

4.2. Assistentes Inteligentes Presentes na Literatura

A seguir apresentam-se três exemplos de assistentes inteligentes que aplicam o paradigma da orientação: EUROHELP, KBEmacs e IN-EDITO.

O EUROHELP [BREUKER 87], um projeto conduzido pelo programa europeu ESPRIT, contando com 100 homens/ano por um período de 5 anos, é um assistente que apóia a manipulação e controle de uma ferramenta de domínio independente.

O KBEmacs, um projeto do MIT Artificial Intelligence Laboratory [WALTERS 86] [WALTERS 88], apresenta um assistente baseado em *"clichés"* cujo domínio de aplicação é a fase de programação. Ele acompanha os detalhes e dá assistência às partes fáceis do processo de programação enquanto o programador preocupa-se com as partes difíceis do processo.

O IN-EDITO, um protótipo produto de uma tese de mestrado [GUARANYS 87], é um software inteligente que acompanha a ação do usuário, alertando, criticando, ensinando e relembrando, durante a edição de textos.

4.2.1. EUROHELP

O projeto EUROHELP [BREUKER 87] concentra seus esforços na construção de um assistente inteligente que auxilie o usuário na manipulação e aprendizado de uma ferramenta, independente do domínio. A figura 8 apresenta a arquitetura do assistente EUROHELP.

Devido ao fato que as necessidades do usuário podem ser identificadas pelo próprio usuário ou pelo sistema, o assistente contém, respectivamente, um interpretador de questões e um interpretador do desempenho.

O problema da interpretação das questões pode ser reduzido pela confiança na competência linguística do usuário, o problema da interpretação do desempenho apresenta-se como preocupante. Na orientação normal, o sistema apresenta uma tarefa específica para o usuário. Entretanto, na interpretação do desempenho em assistentes inteligentes, as tarefas podem variar muito. Descobrir que algo está incorreto é altamente dependente da identificação das intenções do usuário. Por esta razão, o interpretador do desempenho contém um reconhecedor de planos e um planejador, que cooperam de tal maneira que o primeiro trabalha de maneira *"bottom-up"* e fornece restrições para o último na geração de planos exequíveis. Os planos não apenas podem ser impossíveis ou estar

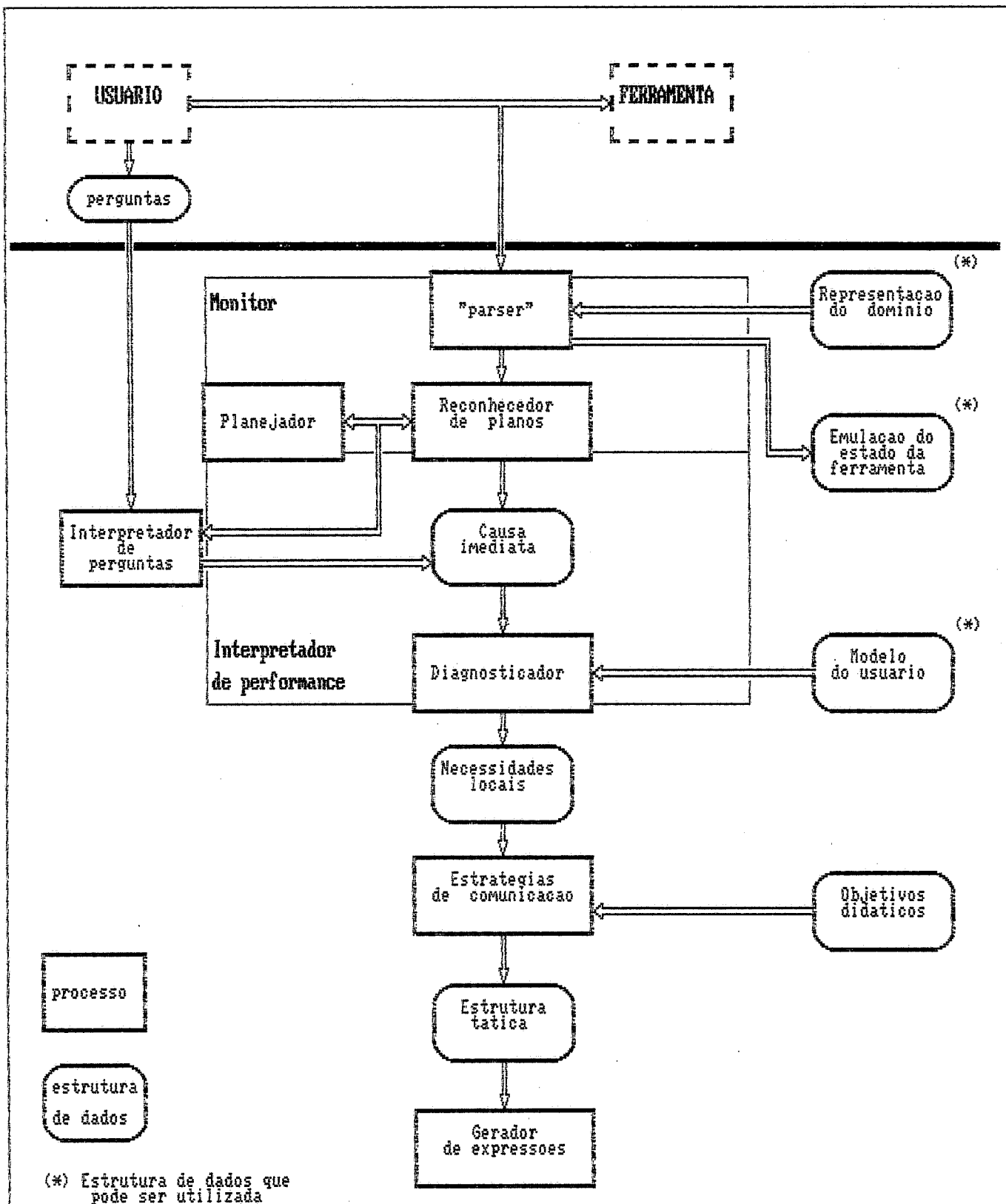


figura 8. Arquitetura do EUROHELP

[BREUKER 87]

incorretos, como eles podem estar muito ineficientes. Por exemplo, em um editor de texto, uma linha pode ser destruída por um simples comando ao invés de fazê-lo caracter por caracter.

A localização dos erros é simplificada em certo grau pelo fato de que muitas ações erradas do usuário não são executáveis pela ferramenta acoplada ao assistente. Entretanto, a variedade de erros executáveis é tão complicada quanto em qualquer domínio natural, porque os usuários podem adquirir todos os tipos de erros conceituais. Embora o diagnóstico de erros conceituais por pertubações sistemáticas do conhecimento correto do domínio esteja certamente além do estado da arte para domínios de qualquer complexidade, a identificação dos erros e os conceitos envolvidos pelo diagnosticador é forçado pelo objetivo corrente do usuário e pelo modelo do usuário detalhado.

No EUROHELP, a orientação tem duas funções:

a. Assistir ao usuário com um problema corrente;

b. Ensinar ao usuário sobre a ferramenta acoplada.

Estas duas funções estão fortemente ligadas. O desempenho correto facilita o aprendizado e o conhecimento sobre a ferramenta acoplada permite melhor desempenho. Entretanto, o escopo destas funções é diferente. Objetivos de aprendizado são a longo prazo, enquanto a função de auxiliar é um objetivo local. Por isso a distinção entre necessidades globais (conhecimento

a ser adquirido sobre a ferramenta em particular) e necessidades locais (que estabelecem o problema corrente do usuário).

A apresentação da informação consiste de uma sequência de atos de discurso ou táticas. Esta sequência é resultado de um processo de planejamento tomando em consideração o que dizer, quando e como, dado o problema identificado do usuário. Sistemas inteligentes de ensino contém mais ou menos estratégias de ensino pré-estabelecidas. Neste sistema, as estratégias de orientação estão também pré-estabelecidas na forma de *"frames"* fixos, no qual tópicos podem ser inseridos. Entretanto, a grande variedade de necessidades locais requer uma técnica de geração. As estratégias de orientação devem ser geradas como uma função do problema corrente e o estado do conhecimento do usuário.

A estrutura do módulo estratégias de comunicação, ilustrada na figura 9, consiste dos três níveis seguintes:

a. *Gerador de objetivos didáticos:*

Os objetivos didáticos são *"overlays"* dos conceitos do domínio que fornecem uma visão didática do domínio;

b. *Planejador de estratégias:*

O segundo nível contém um planejador que constrói as estratégias de orientação;

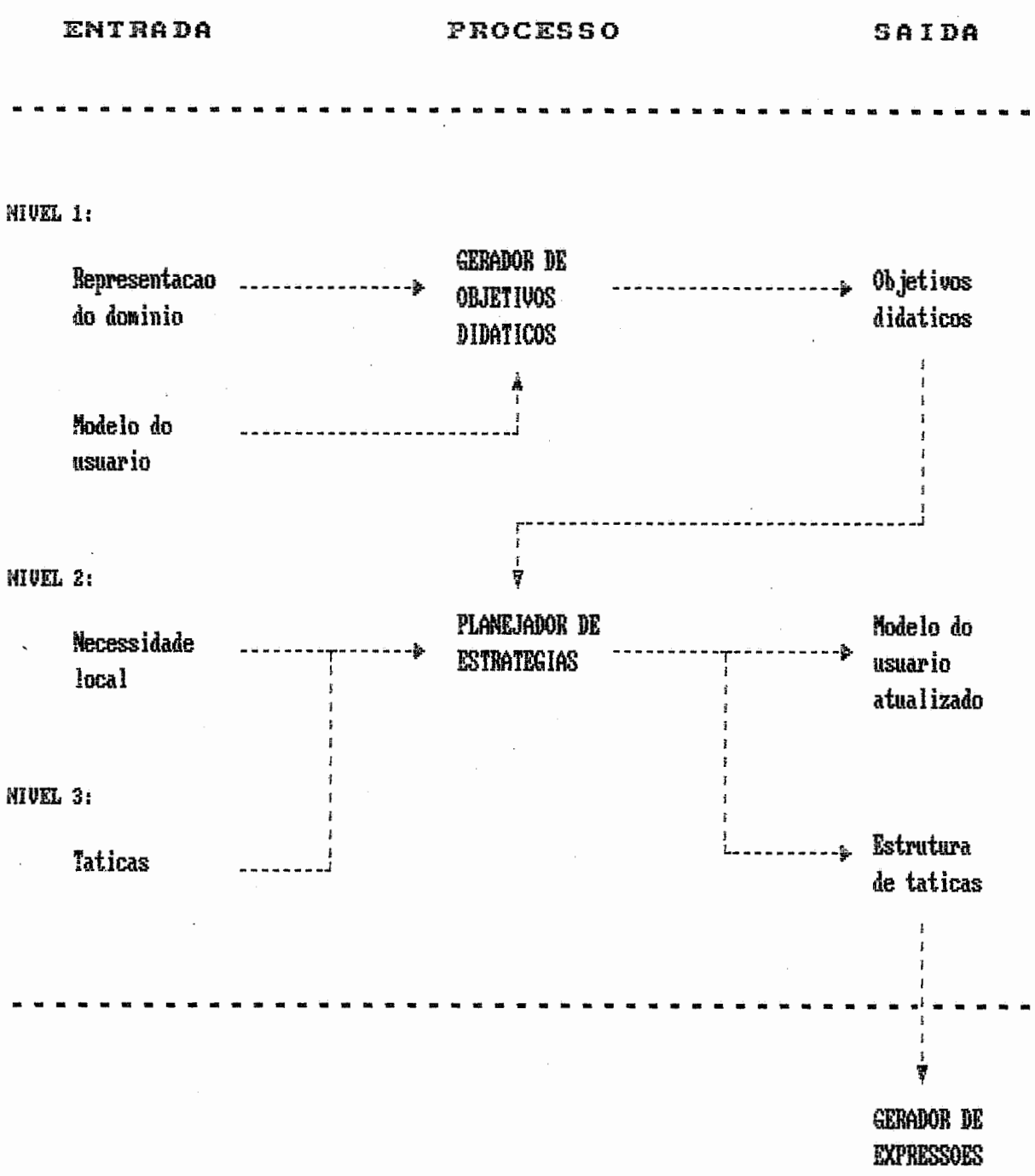


figura 9. Entradas, Saidas e Processos do Modulo Estrategias de Comunicacao
[BREUKER 87]

c. *Táticas:*

O terceiro nível contém as táticas, que representam uma estrutura de dados. As táticas são os elementos terminais das estratégias. Elas são atos do discurso, consistindo de um ato de comunicação.

Um objetivo didático refere-se ao conceito do domínio ou a parte deste domínio. Os objetivos didáticos indicam os princípios didáticos que estão envolvidos no aprendizado de um novo conceito, dado que outros conceitos já estão fundamentados. Os objetivos didáticos não precisam ser especificados pelo desenvolvedor de um assistente para um domínio particular, mas são derivados da representação do domínio (e do modelo do usuário) pelo gerador de objetivos didáticos.

Uma vez identificada a necessidade local do problema corrente do usuário, esta é enviada para o módulo estratégias de comunicação. Três tipos de necessidades locais correspondem a três funções da orientação, a saber:

a. *Erro:*

Quando o usuário emite um comando não executável ou executa de uma maneira diagnosticada como não intencional ou não ótima. A função de orientação é remediar o problema;

b. *Ocasião para expansão:*

Quando o desempenho do usuário fornece uma ocasião para introduzir um novo conceito (um objetivo didático). A função de orientação é expandir novo conhecimento;

c. *Falta de "feedback":*

Quando o *"feedback"* da ferramenta é insuficiente para o usuário. A função de orientação é prover "feedback".

Uma questão colocada pelo usuário pode, em combinação com o desempenho, causar uma das três necessidades locais descritas acima.

A necessidade local, também, contém a causa imediata (isto é, o responsável pelo problema) e um diagnóstico da falta específica de conhecimento ou erro de conceito que explique a necessidade local. O diagnóstico fornece assim os tópicos da necessidade local, fornece alguma informação sobre o estado de conhecimento que o usuário tem a respeito destes tópicos (proveniente do modelo do usuário) e apresenta o conhecimento necessário para evitar aquele erro no futuro.

A primeira decisão que o módulo estratégias de comunicação tem que tomar é decidir se fornece auxílio puro ou orientação. A orientação é aconselhável, caso exista a oportunidade de expandir o conhecimento do usuário. Caso contrário, pode assumir que o usuário não é capaz de reter nova informação e decide fornecer auxílio,

que sob este ponto de vista é uma versão despojada de orientação.

A seguir o planejador de estratégias escolherá uma estratégia de alto nível para enfrentar a necessidade local. As estratégias de alto nível são: Remediar, Expandir e Prover *"feedback"*. Não existe diferença fundamental entre as estratégias de alto nível e sub-estratégias. A primeira estratégia escolhida é denominada de alto nível. De fato, a mesma estratégia pode ser mais tarde uma sub-estratégia. Por exemplo, muitos remendos implicam em uma expansão do conhecimento do usuário. Neste ponto um ciclo recursivo de seleção de estratégias e refinamento de estratégias começa. As estratégias de alto nível apontam para um conjunto pré-estabelecido de estratégias mínimas.

As estratégias podem ser gerais ou específicas. As estratégias gerais não são imediatamente aplicáveis. Elas necessitam refinamentos sucessivos para acomodar-se à situação corrente. As estratégias específicas são aplicáveis como elas são, mas requerem uma situação corrente correspondente. Se a situação corrente não corresponde a uma estratégia específica disponível, uma estratégia geral será escolhida.

Depois de um plano ser selecionado, começa o processo de refinamento. Toda a estratégia será refinada até ser mapeada diretamente para uma tática (o elemento terminal que pode ser executado imediatamente). Certas vezes será necessário voltar para consultar outros módulos do assistente, por exemplo o planejador de

desempenho para construir um plano para alcançar algum objetivo (por exemplo, reparar para desfazer uma situação corrente) ou ao modelo do usuário para descobrir se um tópico é conhecido ou não.

Finalmente, se tem uma estrutura complexa com táticas como elementos terminais. Pode existir um conjunto de redundâncias ou uso ineficiente de táticas e, portanto, algumas supressões precisam ser feitas. A tática resultante será alimentada através do gerador de expressões. Este fornece processamento texto-semântico, sintático e léxico. Agora, o processo de geração de saídas consiste em preencher os *"frames"* de textos que estão associados com as táticas.

Breuker apresenta, de forma detalhada, o mecanismo utilizado para identificação das táticas em um domínio particular. O domínio escolhido para a investigação é a edição de texto. O *"Unix Vi"* foi escolhido pois os conceitos envolvidos são moderadamente complexos e o desempenho eficiente depende do planejamento elaborado.

4.2.2. Projeto Aprendiz de Programador: KBEmacs

O *"Knowledge-Based Editor in Emacs"* (KBEmacs) é um sistema de demonstração implementado como parte do Projeto Aprendiz de Programador [WALTERS 86] [WALTERS 88]. KBEmacs é capaz de atuar como um assistente semi-inteligente para uma pessoa que está escrevendo um programa - tomando para si algumas tarefas da programação. Utilizando KBEmacs, é possível construir um programa pela introdução de uma série de comandos de alto nível. KBEmacs é capaz de operar em programas ADA e LISP de tamanho e complexidade reais.

As técnicas básicas utilizadas pelo KBEmacs transcendem o domínio da programação. KBEmacs é um exemplo de uma classe geral de sistemas de Inteligência Artificial, uma vez que tem o intuito de imitar o comportamento inteligente.

KBEmacs recebe como entrada um conjunto de dados e gera uma resposta complexa - um projeto completo para o artefato desejado. A base de conhecimento deve conter vários pedaços de conhecimento sobre como projetar determinados artefatos.

O objetivo do projeto Aprendiz de Programador é desenvolver uma teoria de como os programadores experientes analisam, sintetizam, modificam, explicam, especificam, verificam e documentam programas. Esta é a pesquisa básica de intercessão da Inteligência Artificial e Engenharia de Software. A intenção é que o aprendiz atue como um parceiro e crítico, acompanhando os detalhes e assistindo com as partes fáceis do processo de programação enquanto o programador preocupa-se com as partes difíceis do processo.

Uma virtude importante do KBEmacs é que a sua implementação é independente da linguagem de programação.

Existem dois princípios básicos que fundamentam este projeto: a técnica de assistência e métodos de inspeção (*"clichés"*).

a. *Técnica de Assistência:*

Uma técnica para solucionar os problemas correntes do software é eliminar os programadores através da programação automática. Como foi concebida, a programação automática exige que o usuário final escreva uma especificação completa do que deseja, um sistema automático gera então um programa que satisfaça a especificação. Geradores de programas deste tipo foram desenvolvidos para um número bem limitado de aplicações.

Uma técnica alternativa é assistir o programador ao invés de substituí-lo. Um exemplo provocativo de uma técnica de assistência foi proposta por Harlan Mills da IBM em 1970. Ele

sugeriu a criação de grupos de programadores chefes compostos por programadores experientes que assistiriam outras pessoas menos experientes (programador júnior, documentador, etc). A produtividade aumentou porque o programador chefe podia aplicar todo o esforço nas partes mais difíceis. A experiência demonstrou que esta divisão de trabalho era eficiente.

Pensou-se no aprendiz como um novo agente no processo de software, tal como uma ferramenta, interagindo com o programador como um assistente humano. Além do mais, o programador e o aprendiz têm o acesso a todas as facilidades do ambiente de programação. A figura 10 ilustra esta interação.

O segredo para esta técnica é a comunicação entre o programador e o aprendiz. Precisa basear-se em um corpo substancial de conhecimento compartilhado de técnicas de programação. Se o programador tem que explicar tudo para o aprendiz, é mais fácil fazer o trabalho sozinho.

A técnica de àssistência teve um progresso incremental. Inicialmente, o aprendiz era capaz de tomar conta das tarefas simples de programação. Com os avanços da tecnologia, as porções das tarefas manipuladas pelo aprendiz cresceram.

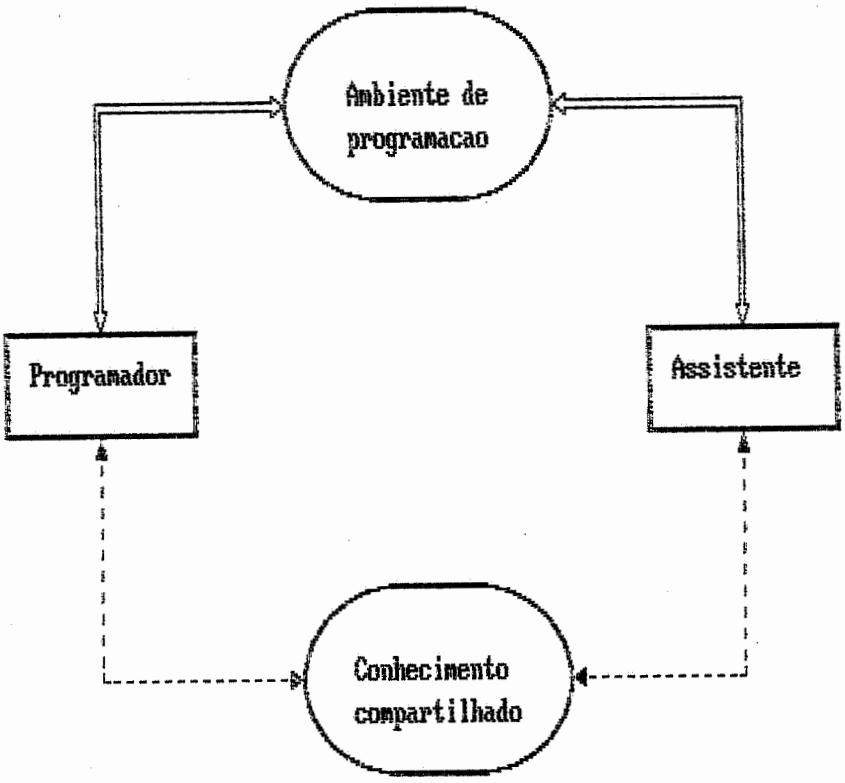


figura 10. Filosofia do Assistente de Programacao

[WALTERS 86]

b. Métodos de Inspeção (Clichés):

O termo *"clichê"* é utilizado para referir-se a um método padrão de lidar com uma tarefa. No uso normal, a palavra *"clichê"* tem um sentido pejorativo com conotação de falta de criatividade.

Em geral, um *"clichê"* consiste de papéis e regras. Os papéis do *"clichê"* são as partes que variam de uma ocorrência de um *"clichê"* para o próximo. As regras são utilizadas para especificar os elementos fixos da estrutura (partes presentes em todas as estruturas), para verificar se as partes que preenchem os papéis em uma ocorrência particular são consistentes e para determinar como preencher totalmente os papéis em uma ocorrência parcialmente especificada de um *"clichê"*.

Uma propriedade essencial de *"clichés"* é a sua relação com outros *"clichés"*. Por exemplo, um *"clichê"* pode ser um caso especial ou uma extensão de outro *"clichê"*. *"Clichés"* de algoritmos e estruturas de dados podem estar relacionados como possíveis implementações de *"clichés"* de especificação.

Dada uma biblioteca de *"clichés"*, é possível realizar muitas tarefas por inspeção. Por exemplo, na análise por inspeção, as propriedades de um programa podem ser deduzidas pelo reconhecimento de ocorrências de *"clichés"* e referindo-se às suas propriedades conhecidas. Na síntese por inspeção, decisões de implementação são feitas pelo reconhecimento de *"clichés"* na especificação e então

escolhendo-se entre vários *"clichés"* de implementação.

Uma parte essencial do conceito de *"clichés"* é a reutilização. Uma vez que alguma coisa tenha sido concebida e recebeu um nome, então poderá ser reutilizado como um componente em planos futuros.

No KBEmacs, uma grande porção do conhecimento compartilhado entre a máquina e o homem está na forma de uma biblioteca de *"clichés"*.

"Clichés" e métodos de inspeção são conceitos teóricos. Para aplicar estas idéias, *"clichés"* precisam ser representados em uma forma concreta utilizável pela máquina. A representação formal de programas *"clichés"* de programação é denominada de *"Plan Calculus"*.

A seleção de uma representação apropriada do conhecimento é o segredo da aplicação da Inteligência Artificial em qualquer tarefa. Como um problema prático, a única maneira de realizar uma operação complexa é encontrar uma representação do conhecimento pelo qual a operação possa ser executada de uma forma relativamente simples. Com esta finalidade, muitos sistemas da Inteligência Artificial utilizam a idéia de um plano: uma representação que deliberadamente ignora alguns aspectos de um problema a fim de facilitar o raciocínio sobre os aspectos restantes do problema.

O formalismo do plano utilizado pelo KBEmacs é projetado para representar dois tipos básicos de

informação: a estrutura de programas particulares e conhecimento sobre *"clichés"*.

A estrutura de um programa é expresso essencialmente como um diagrama de fluxos onde fluxo de dados e fluxo de controle são representados por arcos explícitos. A fim de representar os *"clichés"*, é fornecido suporte adicional para representar papéis e regras.

Com a mesma importância, uma representação do conhecimento deve facilitar as operações a serem executadas. As principais operações realizadas pelo KBEmacs são o raciocínio simples sobre programas e a combinação entre *"clichés"* para a criação de programas. O formalismo do plano é projetado essencialmente para suportar estas operações.

Um aspecto importante do formalismo do plano é a abstração das características sintáticas de linguagens de programação e a representação das características semânticas de um programa diretamente. Além de facilitar a manipulação de programas, tem a vantagem de tornar as operações internas do KBEmacs independentes da linguagem de programação.

KBEmacs foi implementado na *"Symbolics LISP Machines"*. A figura 11 mostra a arquitetura do sistema. O KBEmacs mantém duas representações para o programa: o

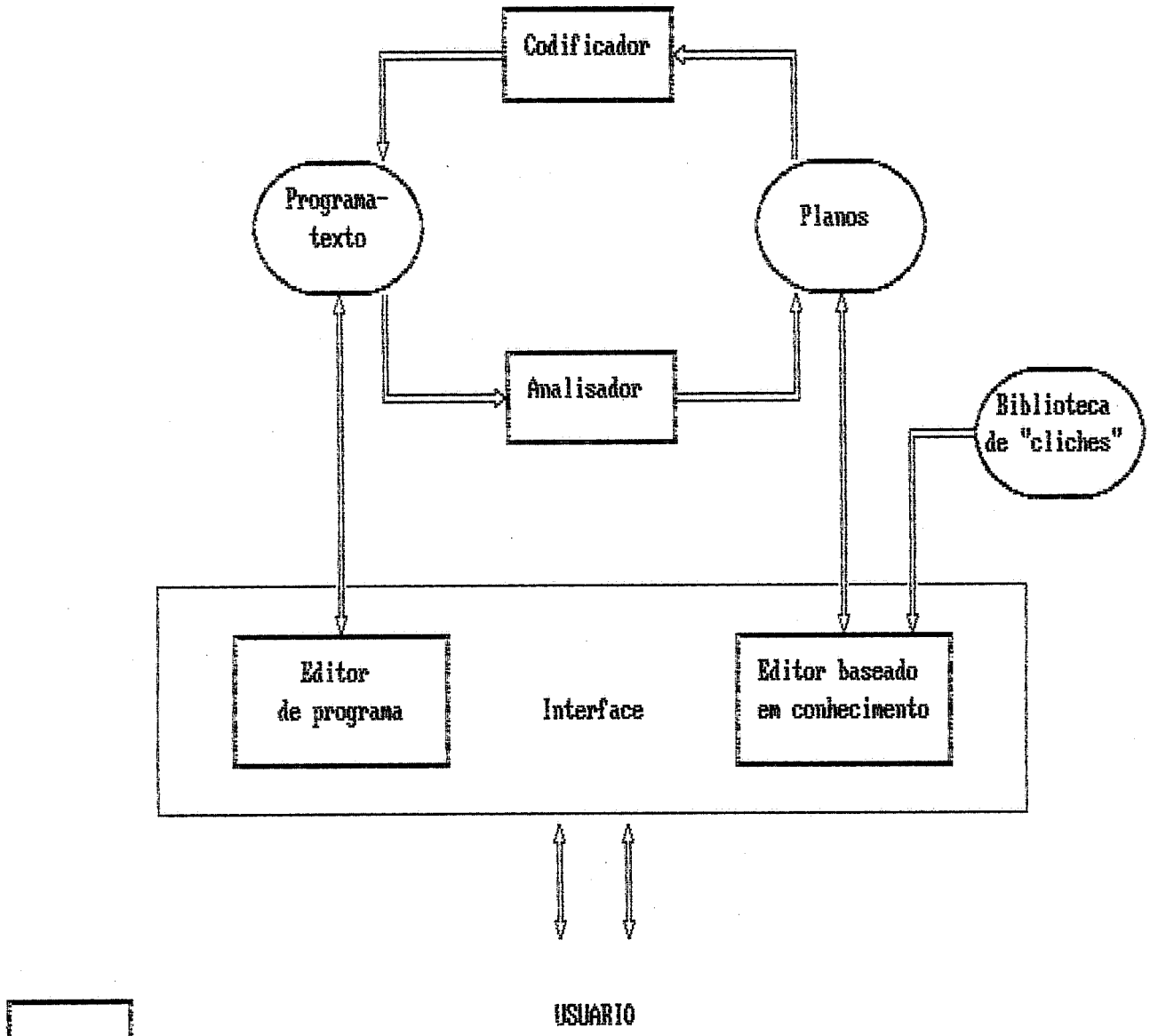


figura 11. Visao Geral da Arquitetura do KBEmacs
[WALTERS 86]

programa-texto e o plano. A qualquer momento, o programador pode modificar as duas representações. Uma interface unifica a edição do programa e a edição baseada em conhecimento. Assim sendo, ambos podem ser acessados através do editor padrão da máquina LISP.

Um importante acessório do editor baseado em conhecimento é a biblioteca de *"clichés"*. A biblioteca age como um simples repositório de "clichés" (representados por planos) que podem ser referidos pelos seus nomes.

O objetivo do protótipo implementado foi demonstrar a exequibilidade da técnica de assistência baseada em *"clichés"* para o domínio da programação. Atualmente, a equipe do projeto está empenhada na construção do Aprendiz de Requerimentos e do Aprendiz do Projeto, que suportarão as primeiras fases do processo de programação: a aquisição e análise dos requisitos e o projeto detalhado.

4.2.3. IN-EDITO - Uma Interface Inteligente para um Editor de Texto

IN-EDITO [GUARANYS 87] é uma interface inteligente específica para um editor de texto. O resultado deste trabalho foi a implementação, em Turbo Prolog, de um pequeno protótipo da interface para um editor semelhante ao Wordstar.

O sistema apresenta ao usuário as possibilidades de requisitar uma ajuda ou explicação. Além disto, acompanha a utilização do editor pelo usuário. Consequentemente, o sistema possui três módulos principais: ajuda, explicação (após requisição por parte do usuário) e inteligente (fornecida pelo sistema no momento oportuno).

A ajuda fornecida pela interface é uma consulta às capacidades do editor, isto é, às funções disponíveis e aos comandos a elas associados. Tal comando ocasionará o aparecimento de uma tela contendo os objetos sobre os quais o usuário pode estar com um problema ou desejando explicação. A ajuda é fornecida em sucessivas telas, que vão aparecendo e superpondo-se à medida que o usuário requisita maiores detalhes ou detecta mais precisamente a consulta desejada.

A explicação é a apresentação ao usuário de um histórico de sua interação com o editor. Após o editor

receber o comando de explicação, é aberta uma janela com a sintaxe e a semântica do último comando fornecido. Caso o usuário deseje, poderá ver o comando fornecido anteriormente ao mostrado, e assim sucessivamente. A razão principal para a utilidade desses recursos da interface é assistir o usuário quando o editor produz uma ação inesperada ou diferente da desejada.

O acompanhamento do usuário é a parte inteligente, pois contém capacidade de inferência. Ele consiste no fornecimento, sempre que possível, de uma maneira mais eficiente de obter o mesmo resultado final desejado pelo usuário.

Para o desenvolvimento do módulo inteligente foi necessário um estudo dos comandos existentes. No editor de texto tratado, os comandos dividem-se em três grupos quando classificados quanto às suas definições: os que dizem respeito a manipulação de blocos, os relacionados com a criação de linha e os que são definidos como uma sequência de repetições de outros mais simples. Em razão de dificuldades, apenas a parte inteligente dos últimos dois grupos foi elaborada inicialmente.

Uma vez com os comandos e suas definições, deve-se a seguir estudar o comportamento dos mesmos para verificar que grupos são equivalentes e quais os mais eficientes. Por exemplo, quando o usuário do sistema está percorrendo uma mesma linha da tela, caracter a caracter, e passa pelo menos uma vez pelo começo de alguma palavra, ele é comunicado que existe um comando que o leva

diretamente ao começo da próxima palavra. Da mesma forma, se ele chegar ao fim da linha, saberá que há outro comando que o leva diretamente ao final da linha.

Como o sistema tem o conhecimento de todas as capacidades do editor, bem como quais comandos são definidos a partir de combinações de outros, pode-se otimizar o trabalho do usuário. Além disso, pode-se ensinar-lhe outras formas de obter o mesmo efeito final.

A estrutura da interface, ilustrada na figura 12, reflete o controle do aplicativo. O aplicativo é visto como um conjunto de regras, no qual regra-1 é encarregada de chamar *'leitura-ação'*, que estabelecerá a comunicação com o usuário e após isto, chamará regra-2 para o prosseguimento natural de sua utilização. A regra *'leitura-ação'* controla as três regras referentes aos três módulos que compõem a interface.

O trabalho permitiu a formulação de um método de projeto de interfaces que pode ser adotado na construção de software aplicativo de qualquer domínio.

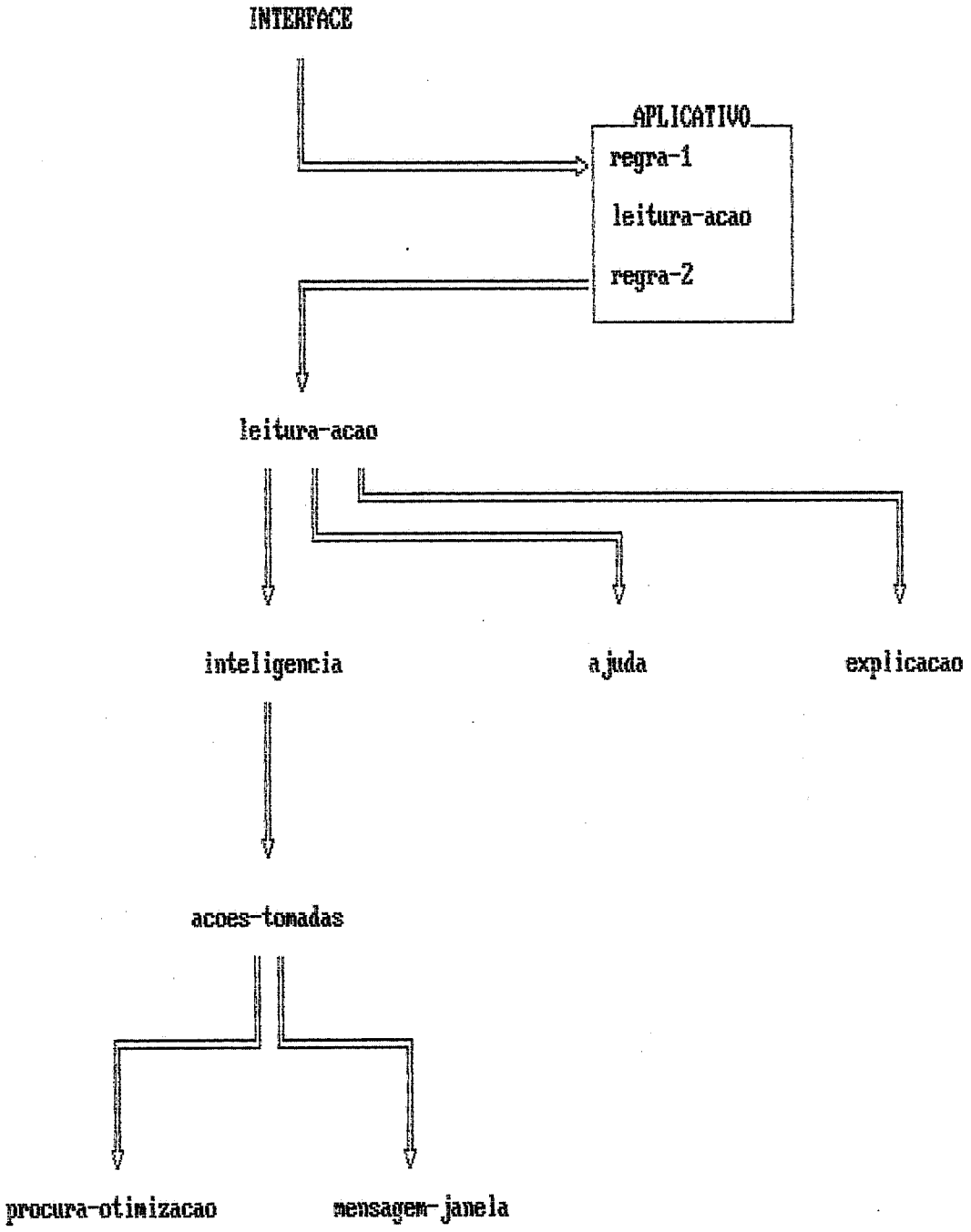


figura 12. Estrutura da Interface
[GUARANYS 87]

5 ARQUITETURA DE UM ASSISTENTE BASEADO EM CONHECIMENTO PARA APOIO A AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE

5.1. Apresentação

O objetivo deste capítulo é definir as características específicas de um assistente baseado em conhecimento de apoio a Ambientes de Desenvolvimento de Software, cujo propósito é auxiliar na manipulação e controle das ferramentas responsáveis pela automatização das atividades do processo de desenvolvimento do software.

O objetivo é desenvolver um modelo de como os assistentes baseados em conhecimento devem interagir com os usuários das ferramentas, como devem ser projetados e implementados, suas características e arquitetura. Esta é uma pesquisa de intercessão da Inteligência Artificial e Engenharia de Software. Da perspectiva da Inteligência Artificial, aplica-se o processo de desenvolvimento do software como um domínio pelo qual deve-se estudar questões fundamentais na representação e entendimento do conhecimento. Da perspectiva da Engenharia de Software, aplicam-se técnicas da Inteligência Artificial para auxiliar o processo de desenvolvimento do software.

A tese está no contexto do Projeto Taba [ROCHA 90], cujo objetivo é a construção de uma estação de trabalho configurável para o desenvolvimento de software, que especifique e gere ambientes adequados às características dos produtos a serem desenvolvidos. Ao configurar um ambiente apropriado, a idéia é gerar automaticamente um assistente baseado em conhecimento específico, que auxilie os usuários da estação de trabalho, no uso do ambiente.

Assim sendo a intenção final desta tese é criar subsídios para a construção deste assistente baseado em conhecimento para apoio ao uso de Ambientes de Desenvolvimento de Software gerados na Estação TABA. Uma vez que um ambiente é composto de métodos, instrumentos e ferramentas que são utilizados ao longo do ciclo de vida, nada mais natural do que imaginar um assistente de apoio a uma determinada ferramenta e, posteriormente, ampliar os conceitos a nível de um ambiente completo. Embora todas as características descritas neste trabalho estejam direcionadas ao nível de ferramentas, podemos considerar que um assistente de apoio ao uso de um Ambiente de Desenvolvimento de Software será um conjunto consistente dos diversos assistentes de apoio às ferramentas que compõem o ambiente.

A necessidade da construção de assistentes baseados em conhecimento para apoio a Ambientes de Desenvolvimento de Software deriva do fato que a Programação Automática ainda é um objetivo a longo prazo [BALZER 85], consequentemente o estado da arte está em auxiliar o

engenheiro de software ao invés de substituí-lo. Deve-se imaginar um assistente como um novo agente no processo de desenvolvimento do software, interagindo com o usuário do Ambiente como um *'assistente humano'*.

Os assistentes descritos na literatura podem ser classificados em ativos ou passivos. Os assistentes ativos caracterizam-se por estabelecer contato com o usuário quando as atividades não estão caminhando muito bem ou quando existe a oportunidade de ampliar o conhecimento do usuário. Os assistentes passivos caracterizam-se por produzir mensagens de erros específicas, ficam ativos apenas sob demanda do usuário. Estes assistentes não fornecem nenhuma informação sobre o domínio da aplicação, abrangendo as informações exclusivas da utilização dos comandos das ferramentas que compõem o ambiente. Na concepção da arquitetura do assistente proposto nesta tese, optou-se pelos assistentes ativos.

Um assistente ideal apóia o desenvolvedor na manipulação e controle de uma atividade automatizada. Um assistente não apenas responde questões do usuário, mas também *'olha sobre seus ombros'* e o interrompe quando necessário. Consequentemente, a necessidade de auxílio pode ser expressa pelo próprio usuário ou ser inferida pelo assistente. Os assistentes que não interpretam o desempenho do usuário possuem uma funcionalidade limitada. A inferência da necessidade de auxílio é elaborada através do reconhecimento na sequência de ações do usuário de um *'plano típico'*, ou seja, procura inferir

qual o objetivo a que ele se propõe. Quando uma sequência diverge do plano reconhecido, o assistente intervém espontaneamente para propor uma estratégia mais adequada àquela situação.

Uma consequência natural ao integrar um assistente a uma determinada ferramenta é a transformação desta ferramenta antes sem qualquer inteligência em uma ferramenta que *'entende'* o domínio de aplicação, seu próprio funcionamento e características e que é capaz de perceber o estado de conhecimento do usuário sobre si mesma e sobre o domínio da aplicação.

O conhecimento da ferramenta contém as características específicas da ferramenta à qual está integrado o assistente. Inclui os comandos válidos, os comandos incorretos, os comandos que podem ser substituídos por comandos mais eficientes e as regras práticas oriundas da experiência dos especialistas na ferramenta.

O conhecimento do domínio de aplicação contém o conhecimento necessário ao usuário para realizar determinada atividade do processo de desenvolvimento do software com o melhor desempenho possível. Este conhecimento pode ser decomposto em três áreas distintas, a saber [ROCHA 87] [ROCHA 88] [WERNECK 88]:

a. *Conhecimento do ciclo de vida:*

Contém as fases do processo de desenvolvimento e as atividades a serem realizadas em cada fase;

b. *Conhecimento do método:*

Contém as técnicas utilizadas para organizar o pensamento e o trabalho do desenvolvedor. Deve conter as técnicas construtivas, normativas e gerenciais;

c. *Conhecimento da área de aplicação:*

Contém as características específicas da área para a qual está se desenvolvendo o software. Um assistente realmente eficaz deve possuir conhecimento da área de aplicação, ou seja, deve *'conhecer'* as características específicas da área de aplicacão em questão.

Devido às características específicas das ferramentas utilizadas no processo de desenvolvimento do software, o assistente deve visualizar o usuário de duas maneiras distintas: Quanto ao seu conhecimento sobre o domínio de aplicação e quanto ao seu conhecimento sobre a utilização da ferramenta. Esta modelagem do conhecimento do usuário é feita através da análise do seu desempenho. Apoiado nesta análise, pode-se construir hipóteses sobre

qual conhecimento o usuário adquiriu ou quais conceitos errôneos possui. Estas hipóteses refletem o estado corrente de conhecimento do usuário. Baseado na modelagem do usuário, pode-se prover uma assistência individualizada.

A interação do assistente com o usuário é ativada de diversas maneiras. Abaixo, são apresentadas algumas das situações possíveis:

a. O usuário solicita a interferência do assistente para dirimir dúvidas em relação à utilização da ferramenta;

b. O usuário solicita a interferência do assistente para dirimir dúvidas em relação ao domínio de aplicação;

c. O usuário solicita a interferência do assistente para apoiar na obtenção de um certo resultado, no que se refere à utilização da ferramenta;

d. O usuário solicita a interferência do assistente para apoiar na obtenção de um certo resultado, no que se refere ao domínio de aplicação;

e. O assistente interrompe o usuário quando infere a necessidade de prover auxílio em caso de erro na utilização da ferramenta;

f. O assistente interrompe o usuário quando infere a necessidade de prover auxílio em caso de erro no universo do domínio de aplicação;

g. O assistente interrompe o usuário quando infere a existência de uma estratégia mais adequada

à situação, no que se refere à utilização da ferramenta;

h. O assistente interrompe o usuário quando infere a existência de uma estratégia mais adequada à situação, no que se refere ao domínio de aplicação;

i. O assistente infere a necessidade de auxiliar o usuário através do fornecimento de alguns procedimentos pré-armazenados, com intuito de facilitar a execução de determinadas tarefas;

j. O usuário solicita a interferência do assistente para apoiar no aprendizado, no que se refere à utilização da ferramenta;

k. O usuário solicita a interferência do assistente para apoiar no aprendizado, no que se refere ao domínio de aplicação;

l. O assistente infere a necessidade de apoiar no aprendizado, no que se refere à utilização da ferramenta;

m. O assistente infere a necessidade de apoiar no aprendizado, no que se refere ao domínio de aplicação;

n. O *'meta-usuário'* solicita os mecanismos responsáveis pela manutenção dos diversos tipos de conhecimento.

A localização dos erros (*ítem .e.*) é simplificada em certo grau pelo fato que muitas ações erradas do usuário não são executáveis pela ferramenta acoplada ao assistente.

5.2. Arquitetura do Assistente

A minha proposta a ser apresentada da arquitetura de um Assistente Baseado em Conhecimento para Ambientes de Desenvolvimento de Software, tendo-se em mente o estado da arte nos campos da Inteligência Artificial e Engenharia de Software, foi calcada em análises detalhadas dos seguintes sistemas:

a. *Representando a classe dos Sistemas Tutoriais Inteligentes com orientação:*

WEST [WENGER 87] [OLIVEIRA 88]; WUSOR [WENGER 87]; Tutor de Geometria [ANDERSON 85a]; Tutor LISP [ANDERSON 85b] [ANDERSON 89]; OFTALMO [CAMELO 88]; [CLANCEX 87]; [KEARSLEY 87]; [LAWLER 87]; [O'SHEA 83].

b. *Representando a classe dos assistentes inteligentes:*

EUROHELP [BREUKER 87]; KBEmacs [WALTERS 86] [WALTERS 88]; IN-edito [GUARANYS 87]; Sistema de Auxílio para Talisman [RONDON 88]; [FIKES 81]; [FININ 82]; [KAISER 87]; [PUNCELLO 88]; [LUCENA 87]; [RIDGE 88]; [SCHOEN 88]; [STEPHENS 85].

Foram estudados os sistemas listados acima, retirando os conceitos que mostrassem utilidade para fundamentar o assistente baseado em conhecimento para o processo de desenvolvimento do software. A partir deste

estudo, pode-se listar algumas das propriedades desejáveis para o assistente:

a. O assistente emula um acompanhamento humano eficiente;

b. O assistente é capaz de reconhecer os *'planos de solução'* do usuário;

c. O assistente faz com que os *'planos de solução'* do usuário convirjam para os dele;

d. O assistente aprende e adota *'planos de solução'* do usuário caso eles sejam superiores aos seus;

e. O assistente é capaz de responder questões do usuário de acordo com o contexto da atividade;

f. O assistente intervém imediatamente quando surgem erros ou quando existe uma estratégia mais adequada no contexto da atividade;

g. O assistente é capaz de auxiliar no aprendizado do usuário, através da escolha de problemas que serão solucionados juntamente com o mesmo;

h. O assistente está apto a adaptar-se a diferentes níveis de conhecimento e estilos dos usuários;

i. O estado de conhecimento do usuário é modelado através do conhecimento do perito embutido no assistente;

O conhecimento do assistente é representado por um conjunto de regras de produção. A arquitetura deve ser altamente modular, oferecendo uma grande distinção entre os diversos componentes. A modularidade facilita a

construção de sistemas independentes do domínio e agiliza a sua manutenibilidade.

Do ponto de vista do usuário, o assistente trabalha em quatro modos distintos, a saber:

a. *Modo Orientador:*

é o *'cérebro'* do assistente. Caracteriza-se por estabelecer contato com o usuário do ambiente toda vez que for necessário (conceitos errôneos, estratégias mais adequadas no contexto da atividade, procedimentos pré-armazenados para facilitar a execução da atividade). A necessidade de intervenção é inferida através do reconhecimento dos objetivos do usuário (planos de solução);

b. *Modo Especialista:*

Caracteriza-se pela solicitação por parte do usuário da interferência do assistente para dirimir dúvidas e apoiar na obtenção de um certo resultado no contexto da atividade;

c. *Modo Tutor:*

Caracteriza-se pela solicitação por parte do usuário ou pela inferência do assistente da necessidade de auxiliar no aprendizado;

d. *Modo Meta-Usuário*:

Caracteriza-se pela solicitação por parte do *'meta-usuário'* da inclusão de conhecimento nos diversos componentes do assistente - modelo ideal, modelo de erros, estratégias de comunicação e outros. Ou seja, deve prover facilidades para a manutenção do conhecimento embutido no assistente.

Cabe salientar que a arquitetura dos quatro modos de trabalho não são disjuntas, pelo contrário, um modo de trabalho utiliza as facilidades presentes nos componentes dos outros modos. Do ponto de vista do usuário da ferramenta, isto passa despercebido.

Nas próximas seções, os modos de trabalho são apresentados em detalhes, explicando-se cada módulo e suas características. A arquitetura foi elaborada para qualquer ferramenta do domínio da Engenharia de Software, entretanto os exemplos estão calcados no protótipo descrito no capítulo 6, cujo propósito é assistir uma ferramenta genérica de edição de diagramas de fluxo de dados.

5.2.1. Modo Orientador

Representa o cerne do assistente, fundamentado no **paradigma da orientação**. O termo orientação descreve um ambiente em que o usuário está envolvido em uma atividade, na qual o assistente opera, ocasionalmente, criticando e sugerindo com o intuito de incrementar o desempenho do usuário.

A intervenção do assistente é estabelecida quando a atividade não está caminhando muito bem ou quando existe uma oportunidade de apresentar uma estratégia de trabalho mais adequada no contexto da atividade.

Uma outra alternativa de intervenção é o fornecimento de procedimentos pré-armazenados que facilitem a execução da atividade. A idéia é a utilização de uma biblioteca de *"clichés"* [WALTERS 86]. Ua vez que o usuário deseje executar um determinado procedimento, o assistente verifica a sua existência na biblioteca. Em caso afirmativo, apresenta-o ao usuário facilitando, assim, o desenvolvimento daquela tarefa. Para exemplificar, utiliza-se a edição de Diagramas de Fluxo de Dados (DFD) e a tarefa a ser desenvolvida é a análise da manutenção de um determinado depósito de dados [GANE 83]. Certamente este procedimento já foi utilizado em situações passadas, consequentemente, ao surgir a oportunidade de reutilizá-lo, o usuário solicita-o, ou

caso passe despercebido, o assistente informa a sua presença na biblioteca de *"clichés"*.

A necessidade de intervenção é inferida através da interpretação do desempenho do usuário, ou seja, o assistente tenta reconhecer na sequência de ações do usuário qual o objetivo a que ele se propõe. Quando uma sequência diverge do *'plano reconhecido'*, o assistente intervém espontaneamente para propor uma estratégia mais adequada.

A localização dos erros é simplificada em certo grau pelo fato que muitas ações erradas não são executáveis pela ferramenta acoplada ao assistente. Entretanto, a variedade de erros executáveis é muito extensa.

A fim de diagnosticar o desempenho do usuário, o assistente deve ser capaz de solucionar o problema no qual o usuário está trabalhando. Com este propósito, o assistente contém um **modelo ideal**, uma simulação do conhecimento de um perito no uso da ferramenta e no domínio de aplicação correspondente. O assistente necessita suplementar este modelo com vários erros que um usuário pode cometer no desvio do modelo ideal. Este modelo é denominado modelo de erros.

Com o objetivo de adaptar as intervenções do assistente às peculiaridades de cada usuário, o assistente procura diagnosticar o **estado de conhecimento do usuário**. A modelagem do usuário é elaborada através da comparação do comportamento do usuário e o suposto

comportamento de um perito na mesma situação, ou seja, o estado de conhecimento do usuário é visualizado como um sub-conjunto do conhecimento de um perito.

A partir do **diagnóstico** do desempenho do usuário, pode-se determinar a natureza exata das interações entre o assistente e o usuário. Pode-se verificar quando uma intervenção é necessária, se o usuário pode ser interrompido em uma atividade, o que pode e deve ser apresentado em um dado momento dentro do contexto. Em suma, pode-se determinar a orientação na execução das tarefas, explicações do fenômeno e do processo e as medidas remediáveis.

A arquitetura do Modo Orientador é apresentada na figura 13. Os componentes deste modo de trabalho estão descritos com detalhes nas próximas seções.

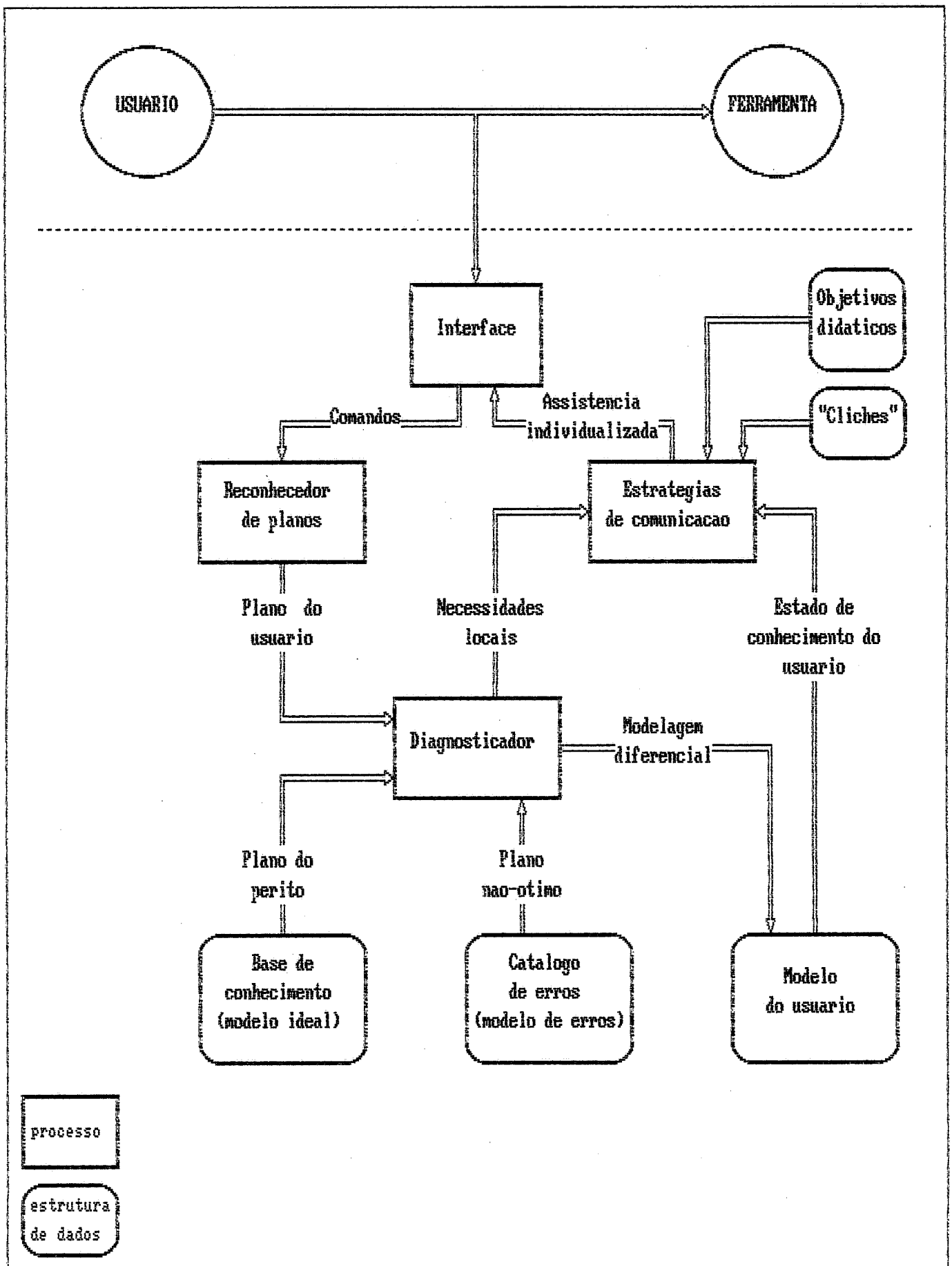


figura 13. Arquitetura do Modo Orientador

(A) Reconhecedor de 'Planos'

As ações dos usuários são analisadas pelo reconhecedor de planos com o intuito de determinar as *'questões'* e os *'planos'* envolvidos. A figura 14 apresenta o paradigma de *'planos e questões'* no contexto do editor de diagramas de fluxo de dados. O conceito de *questões* (em analogia ao discutido no Tutor WEST [WENGER 87]) é um elemento cuja participação nas decisões pode ser reconhecida e discutida, ou seja, uma *questão* é a unidade mais primitiva do conhecimento. Desta forma, uma *questão* pode ser qualquer unidade do conhecimento de interesse estratégico para o assistente.

Lucena [LUCENA 87] denomina as *questões* de conceitos primitivos ou conceitos atômicos de uma representação, ou seja, é a menor unidade conceitual utilizada pela representação para descrever um fenômeno do mundo real.

A decomposição do conhecimento em unidades comunicáveis, as chamadas *questões*, essencialmente reflete a intuição dos autores do sistema sobre os contituintes primitivos do domínio. Entretanto, as primitivas que correspondem às unidades perceptíveis utilizadas pelos usuários não são deduzidas obviamente ou fáceis de determinar a priori. A dificuldade para aplicar esta definição formal para domínios complexos deriva do

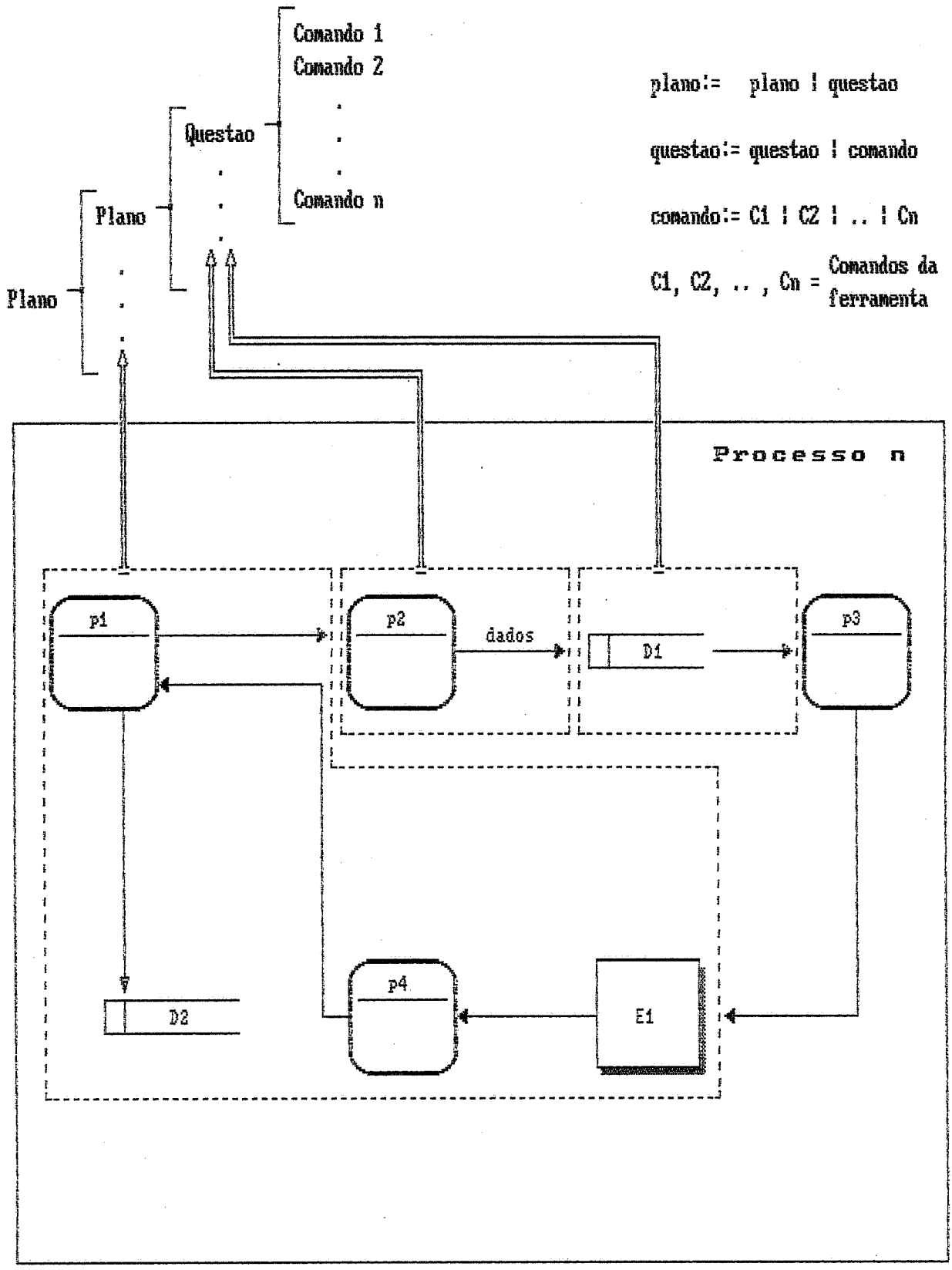


figura 14. Paradigma de Planos e Questoes
(no Contexto do Editor de Diagramas de Fluxo de Dados)

fato que as habilidades envolvidas em tal domínio envolvem múltiplas formas de conhecimento.

No caso de um editor de diagramas de fluxo de dados, uma unidade do conhecimento representa um processo, uma entidade externa ou um depósito de dados tendo um fluxo de dados como origem ou destino. Ou seja, uma questão representa os objetos e o relacionamento entre eles.

Paralelamente ao reconhecimento das *questões*, este módulo encarrega-se de compor o *plano* do usuário a partir das diversas *questões*. Os *planos* são representados como um conjunto bem definido de *questões*. O objetivo da construção deste *plano* é confrontá-lo ao *plano* do perito utilizado para a solução do mesmo problema. Um *plano* representa, no contexto do editor de diagramas de fluxo de dados, uma função imprescindível para modelar um problema específico na área de aplicação ou mesmo um conceito errôneo, como a ligação de duas entidades externas.

(B) Base de Conhecimento (Modelo Ideal)

Segundo Waterman [WATERMAN 86], o coração de um sistema baseado em conhecimento é o poder do corpo do conhecimento acumulado durante a construção do sistema. Este conhecimento deve representar a ação dos melhores peritos na área, solucionando o problema de forma criativa, acurada e eficiente.

Este módulo contém a representação do conhecimento do perito na solução de determinados problemas. A fim de diagnosticar o desempenho do usuário é necessário ter a simulação do *'desempenho ideal'*, ou seja, o assistente deve ser capaz de solucionar o mesmo problema que o usuário está trabalhando. A partir destes dados, o desempenho do usuário é comparado com o comportamento do modelo ideal. Consequentemente, o modelo ideal compreende uma dupla função. Por um lado, atua como fonte para o conhecimento a ser transmitido ao usuário. Por outro lado, age como um padrão para avaliar o desempenho do usuário.

A implementação do modelo do perito é uma restrição importante para a definição do domínio a ser explorado. Para um modelo ser operacional, o conhecimento não pode ser simplesmente enunciado e explicado. Ele deve ser colocado para trabalhar. O fato é que este modelo tem que ser capaz de solucionar o mesmo problema que o usuário está solucionando. Porém, as experiências na

Engenharia do Conhecimento tem revelado muitas dificuldades em extrair informações exatas dos peritos de uma forma que possam torná-las operacionais.

A representação do conhecimento comumente utilizada para formalizar a especialização do sistema é a de regras de produção. Cada regra de produção contém uma parte IF, que representa um conjunto de condições utilizadas para determinar se a regra é aplicável e uma parte THEN, que especifica o que fazer nesta situação. As principais vantagens desta representação são:

a. a *modularidade*, facilitando a manutenção da base de regras;

b. a *uniformidade* com que forma a base;

c. a *naturalidade* com que expressa os tipos de conhecimento mais utilizados por especialistas humanos para explicar como eles realizam suas conclusões ao executarem uma tarefa.

A inclusão de uma lista de conceitos e regras, tal como encontrado em livros, não é suficiente. O conhecimento do domínio requer uma investigação do entendimento do perito no domínio específico, com o intuito de absorver três tipos de conhecimento:

a. *Conhecimento declarativo:* Representa os fatos;

b. *Conhecimento procedural:* Inclui o raciocínio utilizado pelo sistema para solucionar os problemas do domínio;

c. *Conhecimento heurístico:* Descreve as ações tomadas por um perito para executar transformações

no domínio, ou seja, as ações que fazem parte do conhecimento experimental do perito sobre como realizar soluções no domínio. Este conhecimento não descreve conceitos que são partes do domínio.

Devido às características peculiares das ferramentas empregadas no processo de desenvolvimento do software, este módulo deve conter dois tipos distintos de conhecimento: O conhecimento sobre a manipulação da ferramenta e o conhecimento sobre o domínio de aplicação.

```
/* Regras que verificam a consistencia dos niveis superiores */

nivel_superior_consistente:-

   entra_processo (Pai,Nivel,Processo,Fluxo),
   Nivel > 0,
   Nivel_superior = Nivel - 1,
   alguma_inconsistencia (Nivel_superior),
   !,
   necessidade_local (550,argumento (Pai,Processo,Fluxo,_)),
   retract (entra_processo (Pai,Nivel,Processo,Fluxo)).




/* Regras auxiliares */

alguma_inconsistencia (Nivel):-

   sai_entidade (_,Nivel,_,Fluxo),
   not (entra_processo (_,Nivel,_,Fluxo)),
   !.

alguma_inconsistencia (Nivel):-

   entra_entidade (_,Nivel,_,Fluxo),
   not (sai_processo (_,Nivel,_,Fluxo)),
   !.

alguma_inconsistencia (Nivel):-

   sai_deposito (_,Nivel,_,Fluxo),
   not (entra_processo (_,Nivel,_,Fluxo)),
   !.

alguma_inconsistencia (Nivel):-

   entra_deposito (_,Nivel,_,Fluxo),
   not (sai_processo (_,Nivel,_,Fluxo)),
   !.

alguma_inconsistencia (Nivel):-

   entra_processo (_,Nivel,_,Fluxo),
   not (sai_entidade (_,Nivel,_,Fluxo)),
   not (sai_deposito (_,Nivel,_,Fluxo)),
   not (sai_processo (_,Nivel,_,Fluxo)),
   !.

alguma_inconsistencia (Nivel):-

   sai_processo (_,Nivel,_,Fluxo),
   not (entra_entIdade (_,Nivel,_,Fluxo)),
   not (entra_deposito (_,Nivel,_,Fluxo)),
   not (entra_processo (_,Nivel,_,Fluxo)),
   !.

alguma_inconsistencia (Nivel):-

   entra_entidade (_,Nivel,_,_),
   not (sai_entidade (_,Nivel,_,_)),
   !.

alguma_inconsistencia (Nivel):-

   sai_entidade (_,Nivel,_,_),
   not (entra_entIdade (_,Nivel,_,_)),
   !.
```

figura 15. Exemplos de Regras de Producao do Modelo Ideal

(no Contexto do Editor de Diagramas de Fluxo de Dados)

(C) Catálogo de Erros (Modelo de Erros)

Quando um usuário executa uma solução do problema *'não ótima'* (talvez uma ação correta, mas estrategicamente fraca) ou incorreta, um bom assistente deve ter alguma noção da causa do erro. Consequentemente, o sistema deve possuir produções de reconhecimento de erros.

O estudo dos desvios observados em um domínio específico é usualmente denominado Teoria dos Erros. A teoria mais comum é a enumerativa, que toma a forma de catálogos ou bibliotecas dos desvios observáveis, que são individualmente reconhecidos no comportamento do usuário.

A existência de um catálogo de erros que contenha todos os desvios possíveis que um usuário pode cometer em um comportamento ideal, tem um papel fundamental na habilidade do assistente em diagnosticar o desempenho do usuário.

O diagnóstico dos erros é uma das tarefas mais difíceis de serem concretizadas. Neste caso, o problema é amenizado porque o sistema concentra-se na discussão de fatos que são conclusões das regras dos peritos. Assim, localizar a causa de uma ação errônea do usuário resume-se a buscar esta causa nas regras que tem como conclusão aquela do usuário.

Importante salientar que determinada ação do usuário que não encontre uma correspondência no modelo ideal e no modelo de erros, não justifica a descoberta de uma solução 'não ótima' ou incorreta, talvez signifique um modelo ideal incompleto.

```
/* Regras que identificam duas entidades externas conectadas */

entidade_entidade:-

   sai_entidade   (_,_,Entidade1,Fluxo),
   entra_entidade (_,_,Entidade2,Fluxo),
   Entidade1 <> Entidade2,
   not (fluxo_por_processo (Fluxo)),
   necessidade_local (531,argumento (Entidade1,Entidade2,_,_)).




/* Regras que identificam uma entidade externa conectada
   a um deposito de dados                                  */

entidade_deposito:-

   sai_entidade   (_,_,Entidade,Fluxo),
   entra_deposito (_,_,Deposito,Fluxo),
   not (fluxo_por_processo (Fluxo)),
   necessidade_local (533,argumento (Entidade,Deposito,_,_)).




/* Regras auxiliares */

fluxo_por_processo (Fluxo):-

   entra_processo (_,Nivel,_,Fluxo),
   sai_processo   (_,Nivel,_,Fluxo).
```

figura 16. Exemplos de Regras de Producao do Modelo de Erros (no Contexto do Editor de Diagramas de Fluxo de Dados)

(D) Diagnosticador

O módulo diagnosticador é responsável por obter e inferir as informações sobre o usuário e suas ações. Como esta tarefa envolve a elaboração do **modelo do usuário**, esta atividade também é chamada de **modelagem do usuário**.

O processo de diagnóstico é composto de duas fases distintas. A primeira atividade é determinar as unidades do conhecimento que estão diretamente envolvidas na explicação do comportamento. A outra atividade é a atualização do modelo do usuário, em consonância com as novas informações.

O propósito da primeira atividade é determinar que unidades do conhecimento, corretas ou incorretas, foram utilizadas para produzir um comportamento específico e que unidades do conhecimento relevantes não estão sendo usadas. Para definir estas informações, as unidades do conhecimento empregadas pelo usuário são comparadas com as unidades que seriam empregadas pelo perito na mesma situação (modelagem diferencial). A associação entre os erros diagnosticados com os conceitos que fundamentaram estas falhas é identificada através do catálogo de erros.

A partir da modelagem diferencial, pode-se identificar a necessidade local do usuário no problema

corrente. A necessidade local contém uma causa imediata (responsável pelo problema) e um diagnóstico da falta específica de conhecimento ou do erro de conceito que explique a necessidade local.

A segunda atividade representa a atualização do estado de conhecimento do usuário. Esta tarefa é realizada no contexto de uma série contínua de observações do comportamento do usuário através das quais novas informações diagnosticadas devem ser integradas ao modelo do usuário. O processo de diagnóstico tem que desempenhar requisitos contraditórios, pois tem que ser suficientemente sensível para adaptar a atitude do assistente, sem demora, e suficientemente estável para não ser facilmente perturbado por variações locais de desempenho.

O método de atualização mais comum é o uso de atributos escalares, tais como pesos numéricos, associados às unidades do conhecimento. Estes pesos são atualizados por cómputações pseudo-estatísticas representando manifestações do conhecimento no comportamento do usuário. O método de atualização modifica o peso antigo de uma unidade do conhecimento adicionando ou subtraindo um número fixo para um sucesso ou uma falha respectivamente. Para o modelo permanecer sensível às variações recentes, o número fixo pode ser somado a (ou subtraido de) uma fração do valor existente naquela unidade do conhecimento (no Tutor LISP, o novo peso é definido como 2/3 de um peso existente + ou - 1). Este tipo de modelagem assume a independência entre as

várias unidades do conhecimento, em oposição a teoria da hierarquia das unidades do conhecimento (Grafos Genéticos [WENGER 87]).

(E) Modelo do Usuário

Nenhuma assistência é realmente efetiva sem um certo entendimento do receptor. Desta forma, são incluidos todos os aspectos do comportamento do usuário e o conhecimento que tem repercurssões no seu desempenho na forma de um **modelo do usuário**. Entretanto, a tarefa de construir este modelo não é simples para um sistema baseado em computador.

Este módulo é responsável pela avaliação das aptidões e deficiências do usuário. O módulo **estratégias de comunicação** utiliza a modelagem do usuário como guia para a escolha da estratégia mais conveniente a ser seguida nas intervenções do assistente, individualizando assim a comunicação.

Basicamente, a modelagem do usuário é feita pelo método diferencial: o conhecimento do usuário é considerado um sub-conjunto do conhecimento do perito. Assim, a representação interna do modelo torna-se um conjunto de hipóteses do domínio do usuário sobre o modelo ideal.

Baseado nas características específicas das ferramentas utilizadas no processo de desenvolvimento do software, o conhecimento do usuário é modelado de duas maneiras distintas:

a. Quanto ao seu *conhecimento sobre o domínio de aplicação*;

b. Quanto ao seu *conhecimento sobre a manipulação da ferramenta*.

Esta distinção é necessária pois determinado usuário pode ser um perito no domínio de aplicação e não possuir familiaridade com a ferramenta. O contrário também pode ser verdadeiro. Ou seja, os níveis de aptidão para os dois tipos de conhecimento serão diferentes.

Uma vez que o usuário adquire o conhecimento incrementalmente, o escopo de seu conhecimento crescerá progressivamente. Tais diferenças no escopo estimulam a representação do modelo como um *"overlay"*. O conhecimento é decomposto em unidades independentes que transformam-se em dimensões da variabilidade do conhecimento. Um sistema de pesos numéricos é utilizado para indicar o nível de desempenho na manipulação das unidades individuais do conhecimento.

Para determinar quando uma habilidade individual pode ser considerada como *'adquirida'*, este módulo analisa a história das variações dos pesos associados àquela habilidade. Assim, caso a história de uma habilidade indique mudanças constantes poderá significar a manifestação de inconsistências e descontinuidades.

A necessidade de colocar o modelo do usuário em um contexto conceitual mais amplo, obriga a consideração de um tipo adicional de variação entre estados de conhecimento, que não podem ser representados como um sub-conjunto do conhecimento do perito. Esta informação adicional é denominada *'visão'* do usuário. A complexidade do modelo do usuário pode variar de escalas simples que descrevam o desempenho do usuário até a existência de múltiplas *'visões'*.

Um exemplo de *'visão'* poderia ser a receptividade do usuário quanto às estratégias de comunicação do assistente, onde seriam armazenadas as preferências individuais, baseadas na eficiência de cada modo de comunicação. Outro exemplo é a adoção pelo usuário de uma estratégia específica para atacar o problema, pois cada estratégia fornece um contexto diferente para a avaliação da ação do usuário. Um outro exemplo de uma possível *'visão'* reflete a familiaridade do usuário com o linguajar próprio do domínio de aplicação.

(F) Estratégias de Comunicação

é o responsável direto pela condução do diálogo com o usuário. Deve orientar a ação do usuário, aconselhar o usuário sobre erros e antecipar futuras ações baseado nas inferências sobre as atividades correntes do usuário. Deve monitorar as perguntas e respostas do usuário. Este módulo determina a natureza exata das interações entre o assistente e o usuário pelo exame do estado de conhecimento do usuário e pela utilização das facilidades de entrada/saída providas pela interface.

As estratégias são um conjunto de regras descrevendo quando uma intervenção é necessária, se um usuário pode ser interrompido em uma atividade, os tipos de interações e a natureza do *"feedback"* oferecido durante o trabalho do usuário.

Com o intuito de condicionar o nível de conteúdo e detalhe a cada usuário, as operações deste módulo utilizam como entrada a necessidade local oriunda da atividade de diagnóstico, que estabelece a natureza dos problemas correntes do usuário. Além disso, o modelo do usuário fornece informações sobre o nível de conhecimento do usuário.

Quatro tipos distintos de necessidades locais podem ser especificadas:

a. *Erro:*

Quando o usuário emite um comando não executável ou executa de uma maneira diagnosticada como não intencional. Quando o usuário infringe as regras do domínio de aplicação;

b. *Ocasião para expansão:*

Quando o desempenho do usuário introduz uma ocasião para introduzir um novo conceito (um objetivo didático);

c. *Prover "feedback":*

Quando o usuário necessita da obtenção de determinados resultados. Quando o usuário deseja saber os próximos passos possíveis;

d. *Ocasião para "clichés":*

Quando existe uma oportunidade de fornecer certos procedimentos pré-armazenados (*"clichés"*).

Normalmente, o controle deste módulo é determinado por uma construção de regras de produção. De acordo com a necessidade local e o estado de conhecimento do usuário aplica-se um *'conjunto ativo'* de regras. A execução de uma regra tem o efeito de formatar e enviar uma saída para o usuário ou solicitar uma entrada do usuário via interface.

Em um sistema transparente ideal, as representações explícitas das estratégias de comunicação suscitam o potencial de adaptar e melhorar suas estratégias a todo momento e cria a oportunidade de reutilizar componentes em outros domínios de aplicação.

(G) Interface

A interface é o módulo mais sensível ao usuário-alvo de um assistente. Este módulo traduz a representação interna do sistema para a linguagem de interface entendida pelo usuário, e vice-versa. A interface não pode prescindir de um tratamento de uma linguagem pseudo-natural, pois um assistente ao impor regras rígidas de sintaxe ao interagir com o usuário não pode ser considerado inteligente. E com o intuito de encobrir a fragilidade do processador de linguagem, pode-se acrescer outros atrativos para o usuário, tais como menus, capacidade gráfica, animação, etc

O processamento de uma linguagem pseudo-natural pode ser feita por técnicas conhecidas como reconhecimento de padrões linguísticos ou palavras chaves. Segundo Oliveira [OLIVEIRA 88], as principais vantagens destas técnicas são a facilidade de sua implementação e a agilização do tempo de processamento. O fato de que se deve ter um conhecimento prévio dos padrões esperados de respostas do usuário e não se levar em conta se as construções estão gramaticalmente corretas constituem as principais desvantagens.

O projeto da interface pode quebrar a eficácia do assistente, independente da engenhosidade do restante do sistema. Cinco características são apontadas como indispensáveis [KEARSLEY 87]:

a. Ser fácil de usar. Isto significa a minimização do número de ações necessárias na comunicação com o assistente;

b. Ter uma estrutura ou representação que seja consistente com a ferramenta;

c. Ser muito interativa. A interação deve ser um fato motivador nas atividades;

d. Ter a habilidade de notar erros imediatamente quando eles ocorrem;

e. Prover acesso imediato às informações relevantes.

5.2.2. Modo Especialista

Representa a válvula de comunicação do usuário com o assistente. Caracteriza-se pela solicitação, através de perguntas do usuário, da intervenção do assistente com o intuito de dirimir as dúvidas em relação à utilização da ferramenta e ao domínio de aplicação. Neste modo de trabalho reside uma diferença fundamental em comparação aos sistemas de auxílio tradicionais, uma vez que responde às perguntas de acordo com o contexto das atividades, ou seja, baseado no plano do usuário.

Uma outra alternativa é a solicitação da interferência do assistente caso o usuário deseje obter determinado resultado baseado no plano estabelecido até aquele momento, ou seja, o assistente deve fornecer os próximos passos possíveis.

O Modo Especialista usufrui dos componentes do Modo Orientador apresentados na seção anterior. A única diferença é a inclusão do módulo denominado **interpretador de perguntas**, que identifica os objetivos das perguntas colocadas pelo usuário.

O interpretador de perguntas recebe a dúvida do usuário de forma manipulável. A partir desta informação, infere o objetivo da pergunta. O objetivo da pergunta é enviado ao diagnosticador como uma causa imediata, onde é

identificada a necessidade local do usuário no contexto da atividade.

A arquitetura do Modo Especialista é apresentada na figura 17.

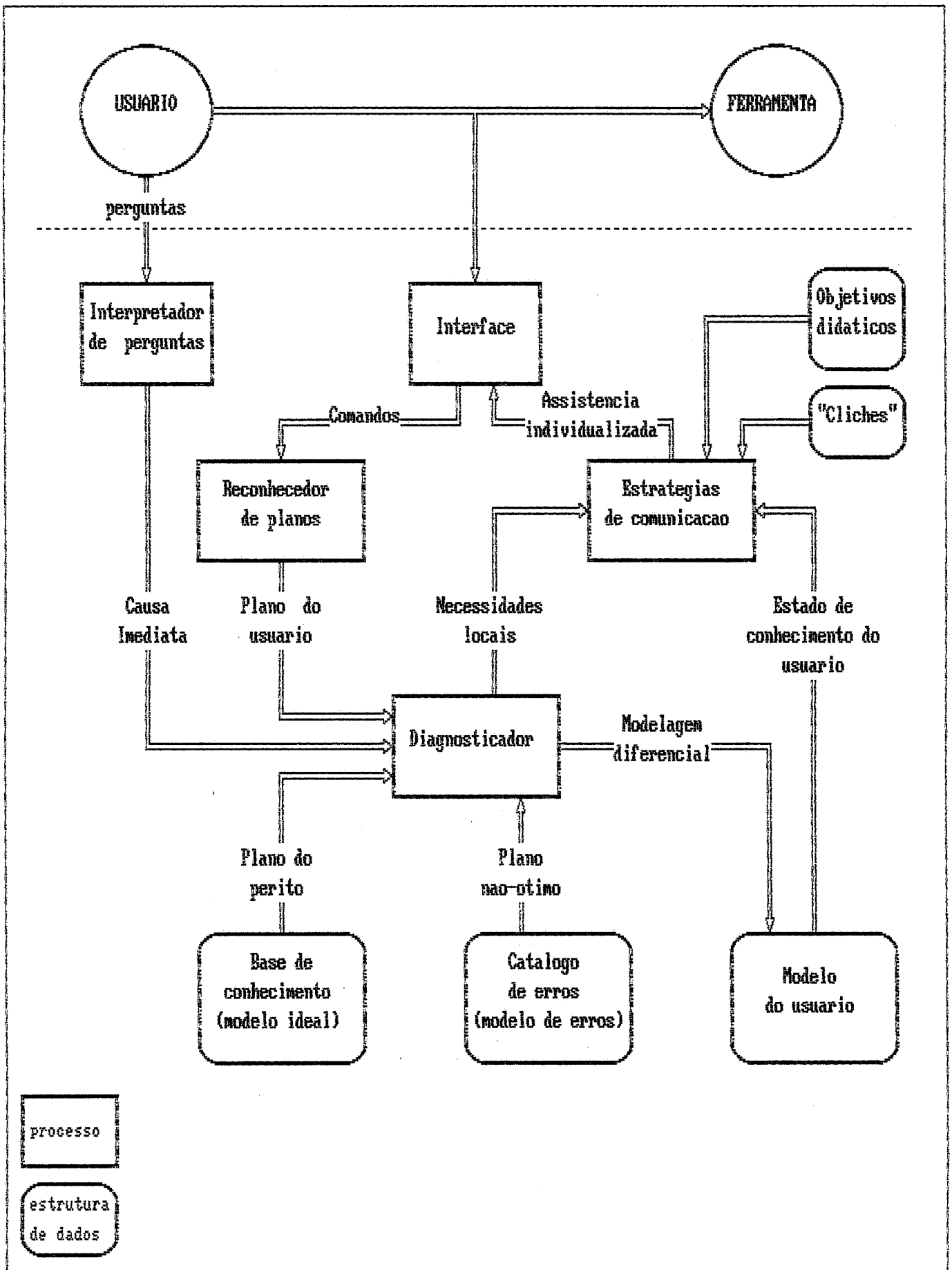


Figura 17. Arquitetura do Modo Especialista

5.2.3. Modo Tutor

Representa o processo de aprendizado dos conteúdos do modelo ideal. é um modo de trabalho para o usuário iniciante ao nível da utilização da ferramenta e/ou ao nível do domínio de aplicação.

O usuário pode solicitar a transferência para o Modo Tutor, caso encontre muitas dificuldades no desenvolvimento da atividade. A outra alternativa é a inferência pelo assistente da necessidade de auxiliar o usuário, caso o desenvolvimento do trabalho esteja muito comprometido. Esta necessidade é percebida através de algumas reações do usuário, a saber:

a. O usuário é repetitivo em seus erros (demonstrando que está inseguro na solução do problema);

b. As informações fornecidas pelo usuário estão fora do contexto da atividade;

c. O usuário é contraditório nas suas ações, ou seja, ele fornece informações incoerentes em ocasiões distintas.

A interação do Modo Tutor com o usuário é estabelecida através da apresentação de uma situação particular. As situações são escolhidas de acordo com o desempenho em suas interações prévias com o assistente (registradas no modelo do usuário). Uma vez escolhida a situação, o usuário desenvolve a solução do problema de

maneira semelhante à interação com o Modo Orientador, mas as explicações são simples e didáticas.

O Modo Tutor utiliza as facilidades dos componentes dos modos anteriores. A diferença resulta da inclusão de um componente simples denominado **selecionador de situações**. A função deste módulo é escolher e apresentar situações específicas para o nível do usuário ativo.

5.2.4. Modo Meta-Usuário

Representa a interação do *'meta-usuário'* com o assistente. A denominação *'meta-usuário'* deriva do fato que o nível de interação com o assistente é diferente do usuário comum. O *'meta-usuário'* é o responsável pela transferência do conhecimento dos peritos para o assistente, processo conhecido como Engenharia do Conhecimento. O Engenheiro do Conhecimento extrai dos peritos seus procedimentos, estratégias e regras práticas para a solução do problema. Uma tarefa de suma importância, uma vez que o poder de auxílio de um assistente emana do conhecimento que possui, e não dos formalismos e esquemas de inferência que empregam.

Este modo de trabalho provê mecanismos que facilitam a manutenção das bases de conhecimento que integram o assistente. A manutenção do conhecimento é uma atividade preciosa, uma vez que o conhecimento está em constante mutação. Devido a grande variedade de áreas de conhecimento que influenciam o projeto do assistente, o *'meta-usuário'* deve interagir com diferentes tipos de peritos.

6. UM ASSISTENTE BASEADO EM CONHECIMENTO PARA UM EDITOR DE DIAGRAMAS DE FLUXO DE DADOS

6.2. Descrição do Protótipo

Um protótipo é um pequeno programa de demonstração que manipula uma porção do problema a ser resolvido. O objetivo principal do desenvolvimento deste protótipo é demonstrar a exequibilidade da técnica de assistência baseada em conhecimento para o domínio da Engenharia de Software. O propósito específico deste protótipo é exemplificar a assistência a uma ferramenta genérica de edição de diagramas de fluxo de dados.

A escolha do editor de diagramas de fluxo de dados não foi ocasional. Devido à grande difusão no Brasil da análise estruturada, existem muitos livros e principalmente, especialistas na modelagem de sistemas através de diagramas de fluxo de dados, facilitando a aquisição do conhecimento.

Três tipos distintos de conhecimento foram levantados no desenvolvimento do protótipo:

a. *Conhecimento da ferramenta:*

Contém as características específicas do editor de diagramas de fluxo de dados. Inclui os comandos válidos; os comandos que podem substituir um conjunto de comandos (ou seja, comandos mais eficientes); os comandos incorretos; as regras práticas oriundas da experiência dos especialistas na ferramenta;

b. *Conhecimento do método:*

Contém as técnicas utilizadas para organizar o pensamento e trabalho do desenvolvedor da aplicação. Inclui técnicas da análise estruturada; regras sintáticas dos diagramas de fluxo de dados; regras práticas oriundas de livros, artigos e da experiência dos especialistas no uso do método.

c. *Conhecimento da área de aplicação:*

Contém as características específicas da área para a qual está se desenvolvendo o software. Inclui as regras práticas oriundas da experiência dos peritos no desenvolvimento de software na área de aplicação em questão.

O ideal seria a colaboração de especialistas na ferramenta, no método e na área de aplicação na construção do protótipo. Entretanto, devido a escassez de tempo para extrair o conhecimento de especialistas, o protótipo apresenta conhecimento oriundo da minha própria

experiência e adquirido em livros e artigos. Este fato limita bastante a efetividade do assistente.

O primeiro passo na construção do protótipo foi identificar as *'questões'*, ou seja, os conceitos atômicos da representação dos diagramas de fluxo de dados. Esta decomposição do conhecimento em unidades comunicáveis reflete a intuição do desenvolvedor do assistente sobre os constituintes primitivos de um domínio específico. Neste protótipo, as questões são um processo, uma entidade externa ou um depósito de dados tendo um fluxo de dados como origem ou destino. As possíveis questões de um editor de diagramas de fluxo de dados estão ilustradas na figura 18.

Em Prolog [ROBINSON 88], as questões do modelo de diagramas de fluxo de dados, apresentadas na figura 18, são representadas da seguinte maneira:

```
sai_processo    (PAI, NÍVEL HIERÁRQUICO,
                 NOME DO PROCESSO, NOME DO FLUXO).
entra_processo (PAI, NÍVEL HIERÁRQUICO,
                 NOME DO PROCESSO, NOME DO FLUXO).
sai_depósito    (PAI, NÍVEL HIERÁRQUICO,
                 NOME DO DEPÓSITO, NOME DO FLUXO).
entra_depósito (PAI, NÍVEL HIERÁRQUICO,
                 NOME DO DEPÓSITO, NOME DO FLUXO).
sai_entidade    (PAI, NÍVEL HIERÁRQUICO,
                 NOME DA ENTIDADE, NOME DO FLUXO).
entra_entidade (PAI, NÍVEL HIERÁRQUICO,
                 NOME DA ENTIDADE, NOME DO FLUXO).
```

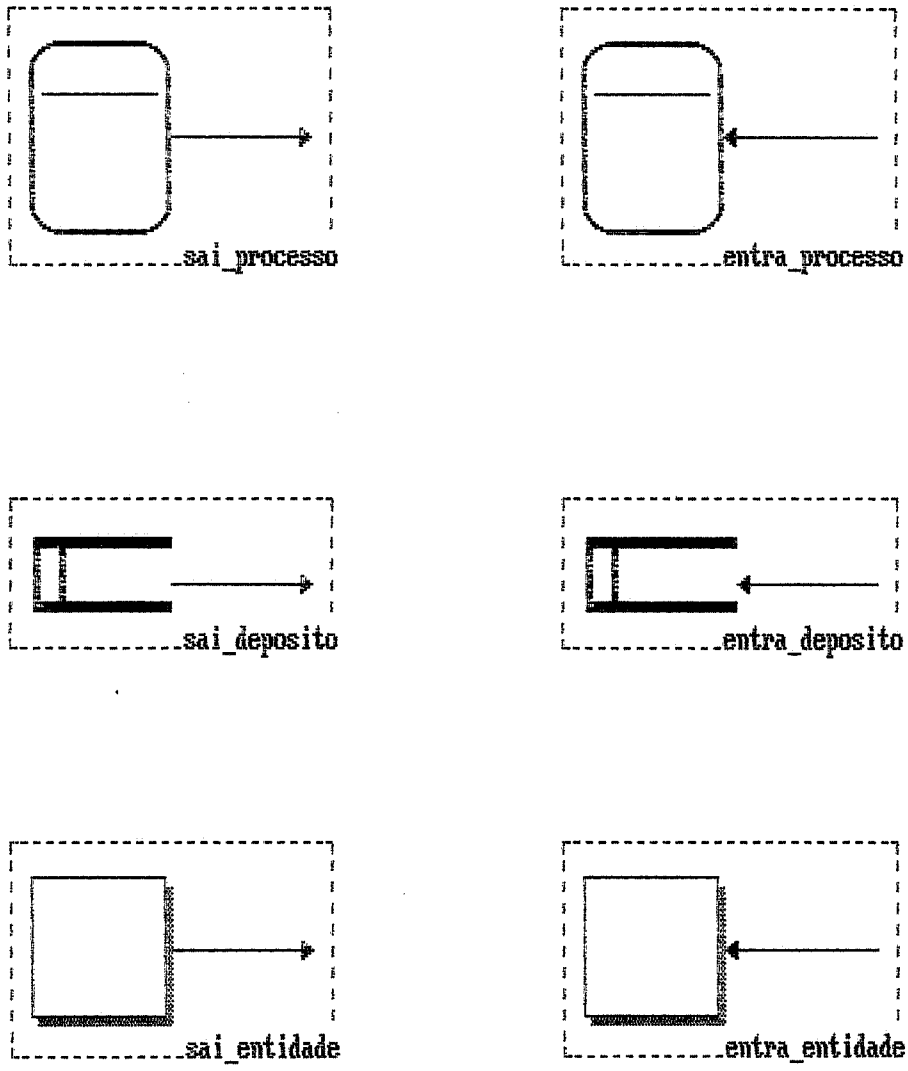


Figura 18. As "Questoes" de um Editor de Diagramas de Fluxo de Dados

Para sedimentar a caracterização das questões, expõe-se na figura 20 a representação em Prolog dos diagramas de fluxo de dados da figura 19.

Um *'plano'* é um conjunto bem definido de questões, que contenha alguma característica relevante no contexto do assistente. No âmbito do editor de diagramas de fluxo de dados, um plano pode representar um conceito errôneo, como a união de duas entidades externas. Um plano pode representar uma função imprescendível para modelar um problema específico na área de aplicação.

O primeiro caso ilustra um desvio nas regras sintáticas dos diagramas de fluxo de dados, uma vez que a ligação entre duas entidades externas não é permitida. Este fato, proveniente do conhecimento do método, está representado no modelo de erros.

O segundo caso apresenta um comportamento ideal na área de aplicação. Este conhecimento, proveniente da experiência dos peritos, está representado no modelo ideal.

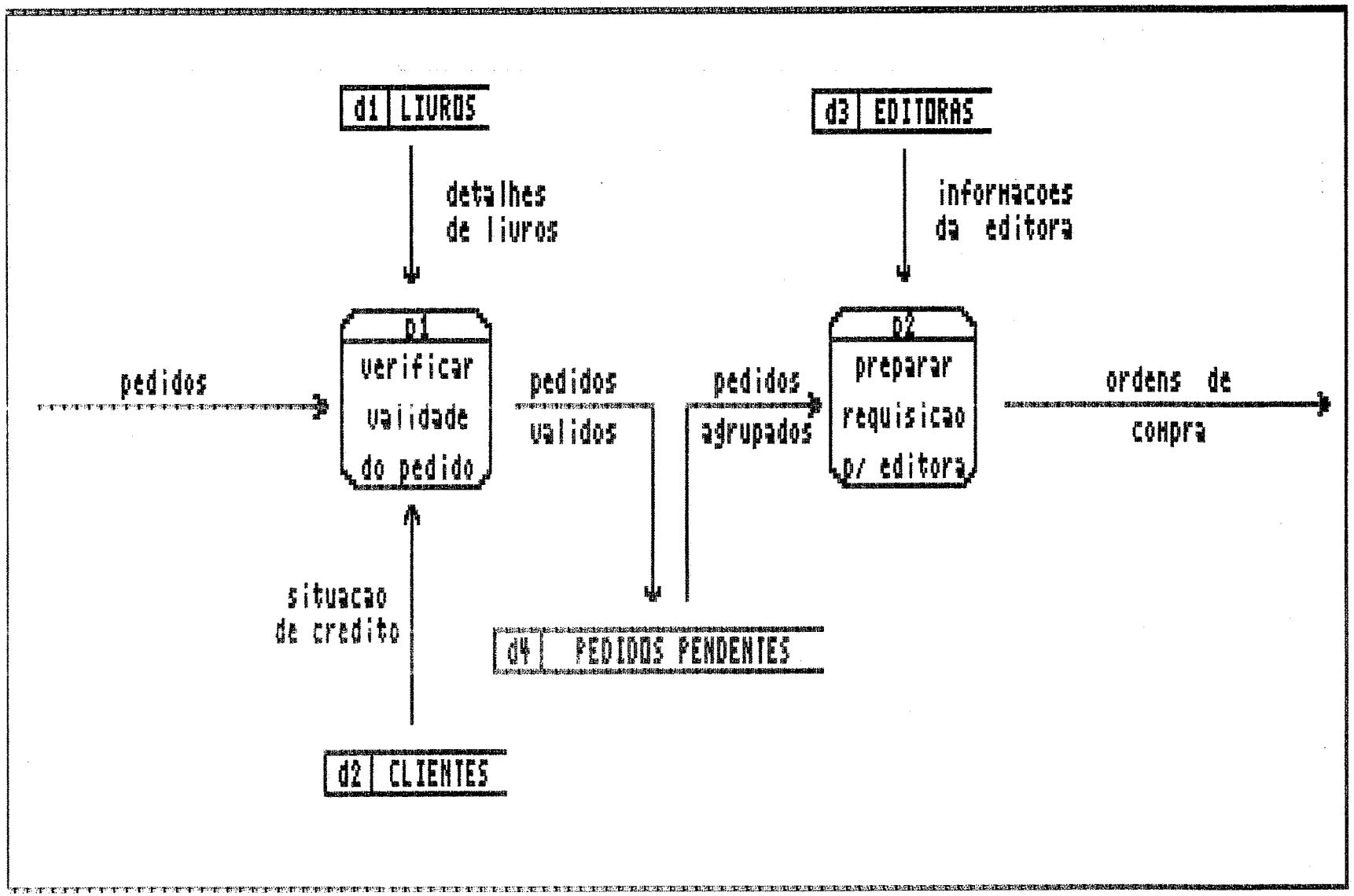


Figura 19. Diagramas de Fluxo de Dados

```
sai_processo ("preparar pedidos",1,"verificar validade do pedido","pedidos validos")

sai_processo ("preparar pedidos",1,"preparar requisicao p/ editora","ordens de compra")


sai_deposito ("preparar pedidos",1,"livros","detalhes de livros")

sai_deposito ("preparar pedidos",1,"clientes","situacao de credito")

sai_deposito ("preparar pedidos",1,"pedidos pendentes","pedidos agrupados")

sai_deposito ("preparar pedidos",1,"editoras","informacoes da editora")



entra_processo ("preparar pedidos",1,"verificar validade do pedido","pedidos")

entra_processo ("preparar pedidos",1,"verificar validade do pedido","detalhes de livros")

entra_processo ("preparar pedidos",1,"verificar validade de pedido","situacao de credito")

entra_processo ("preparar pedidos",1,"preparar requisicao p/ editora","pedidos agrupados")

entra_processo ("preparar pedidos",1,"preparar requisicao p/ editora","informacoes da editora")


entra_deposito ("preparar pedidos",1,"pedidos pendentes","pedidos validos")
```

figura 20. Representacao em Prolog dos Diagramas de Fluxo de Dados da Figura 19

O protótipo do assistente está implementado em Turbo Prolog, versão 1.1, da Borland International. O equipamento necessário para processar o protótipo é um PC/XT ou AT com pelo menos 640kb de memória RAM e um acionador de disco 5 1/4". Trabalha sobre o DOS 3.3 ou mais avançado. A escolha da linguagem Prolog é derivada das seguintes vantagens:

a. O código fonte de um programa escrito em Prolog é cerca de um terço do código fonte deste mesmo programa escrito em Lisp [OLIVEIRA 88];

b. A sintaxe da linguagem Prolog é mais legível que a sintaxe da linguagem Lisp;

c. Restrições de equipamentos e software.

Uma deficiência visível é a ausência de um editor gráfico de diagramas de fluxo de dados (fato oriundo das limitações do Turbo Prolog). Este detalhe não é muito importante, uma vez que a idéia é testar assistentes baseados em conhecimento que serão acoplados à determinadas ferramentas de desenvolvimento de software, consequentemente, estas ferramentas já possuem a sua própria interface gráfica.

Ao acionar o assistente, o usuário poderá interagir em quatro modos de trabalho: Modo Orientador, Modo Especialista, Modo Tutor e Modo Meta-Usuário. Cabe salientar que o Modo Especialista e o Modo Meta-Usuário não foram implementados.

Devido à ausência do editor gráfico, os diagramas são manipulados através de *"menus"*. De acordo com a escolha, as seguintes situações são geradas:

a. *Uma inclusão de determinado elemento no diagrama*. O item "INCLUIR ENTIDADE COM FLUXO DE DADOS SAINDO" representa a inclusão de uma entidade externa em conjunto com um fluxo que sai da mesma. O item "INCLUIR PROCESSO COM FLUXO DE DADOS ENTRANDO" representa a inclusão de um processo em conjunto com um fluxo que chega ao mesmo. Ao escolher um item de inclusão, o usuário recebe uma tela específica, que deve ser preenchida com os valores correspondentes: o nome do pai do elemento (nome do processo que desencadeou a explosão), o nível hierárquico do elemento, o nome do elemento (entidade externa, processo ou depósito de dados) e o nome do fluxo (que sai ou entra do elemento).

b. *Uma exclusão de determinado elemento no diagrama*. O item "EXCLUIR DEPÓSITO COM FLUXO DE DADOS SAINDO" representa a exclusão de um depósito de dados em conjunto com um fluxo que sai do mesmo. O item "EXCUIR PROCESSO COM FLUXO DE DADOS ENTRANDO" representa a exclusão de um processo em conjunto com um fluxo que chega ao mesmo. Ao escolher um item de exclusão, o usuário recebe uma tela específica, que deve ser preenchida com os valores correspondentes: o nome do pai do elemento (nome do processo que

desencadeou a explosão), o nível hierárquico do elemento, o nome do elemento (entidade externa, processo ou depósito de dados) e o nome do fluxo (que sai ou entra do elemento).

c. *Uma análise dos diagramas em construção*. Com a escolha do item "AGENDA", o assistente gera um *"raio-X"* dos diagramas, com o intuito de descobrir possíveis inconsistências. Nesta análise estão descritas as transformações dos fluxos (pode-se averiguar os ciclos, transformações inconsistentes, diagramas com entradas e sem saídas, etc). Estão descritas as pendências dos diagramas (fluxos soltos, processos fontes e/ou sorvedouros, depósitos fontes e/ou sorvedouros). Está descrita uma pequena análise dos fluxos que entram e saem dos depósitos de dados.

d. *Um auxílio sensível ao contexto*. Com a escolha do item "HELP", o assistente apresenta sugestões sensíveis ao contexto da edição dos diagramas.

A qualquer momento, caso o usuário apresente algum desvio no **comportamento ideal**, o assistente intervém apresentando uma mensagem correspondente. No momento da intervenção, o assistente verifica o *'estado de conhecimento do usuário'*. Caso apresente um grau de conhecimento limitado, apresenta-se uma mensagem completa e didática. Caso contrário, apresenta-se uma mensagem compacta.

No Modo Tutor, implementado com mais detalhes, o assistente segue o caminho de solução do usuário e vai explicando os próximos passos a serem seguidos. O usuário interage com o assistente através de simples *"menus"*. A qualquer instante, caso o usuário solicite mais auxílio, o assistente fornece um *"help"* sensível ao contexto. O Modo Tutor foi direcionado para auxiliar usuários completamente leigos na construção de diagramas de fluxo de dados.

O Modo Orientador foi desenvolvido em paralelo com o Modo Tutor, uma vez que todo o aparato e facilidades dos módulos do primeiro são utilizados para fudamentar o segundo. Caso o usuário já apresente alguma familiaridade com a edição de diagramas de fluxo de dados poderá trabalhar diretamente no Modo Orientador. Este modo de trabalho caracteriza-se por estabelecer contato com o usuário apenas quando for necessário.

As figuras 21, 22, 23 e 24 apresentam o modelo lógico do protótipo. O comportamento do perito na solução do mesmo problema enfrentado pelo usuário está representado no modelo ideal. Caso a solução do usuário não esteja condizente com o conhecimento do modelo ideal, o usuário buscou caminhos incorretos ou *'não ótimos'* durante a tentativa de solução do problema. Este conhecimento está representado no modelo de erros. As estratégias de comunicação são moldadas para os padrões específicos de cada usuário de acordo com o respectivo estado de conhecimento do modelo do usuário.

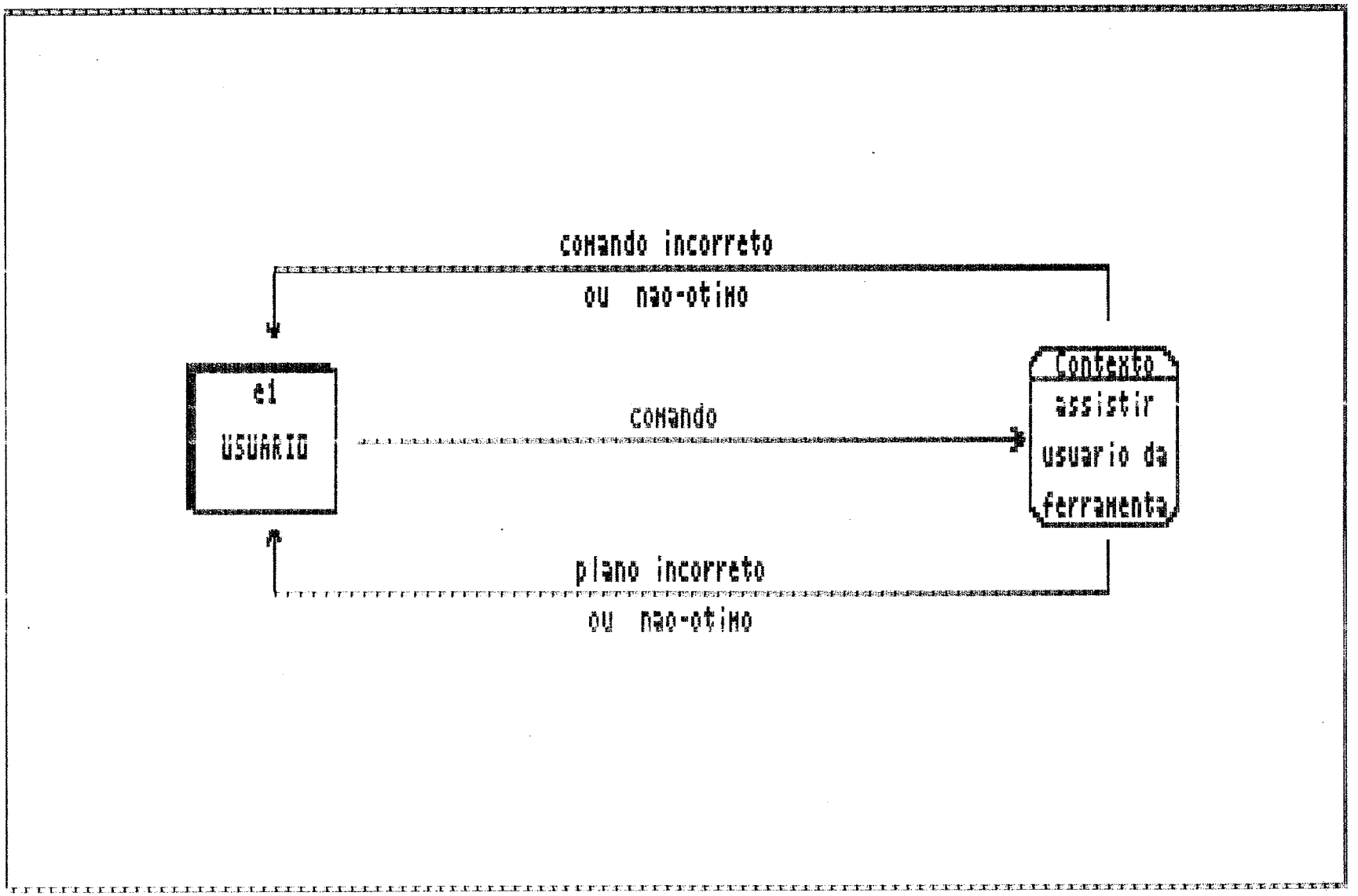


Figura 21. Modelo Logico do Prototipo

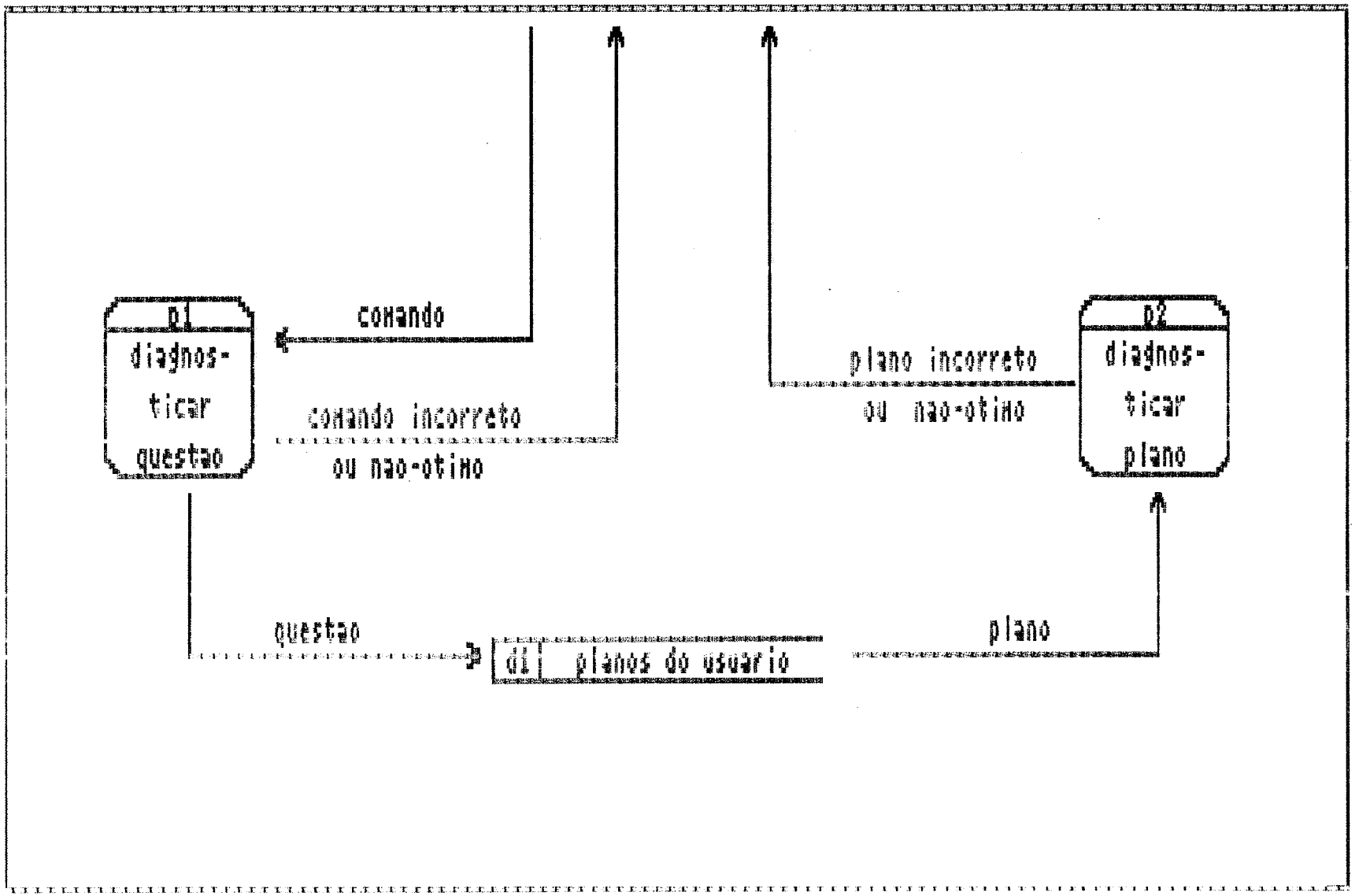


Figura 22. Modelo Logico do Prototipo

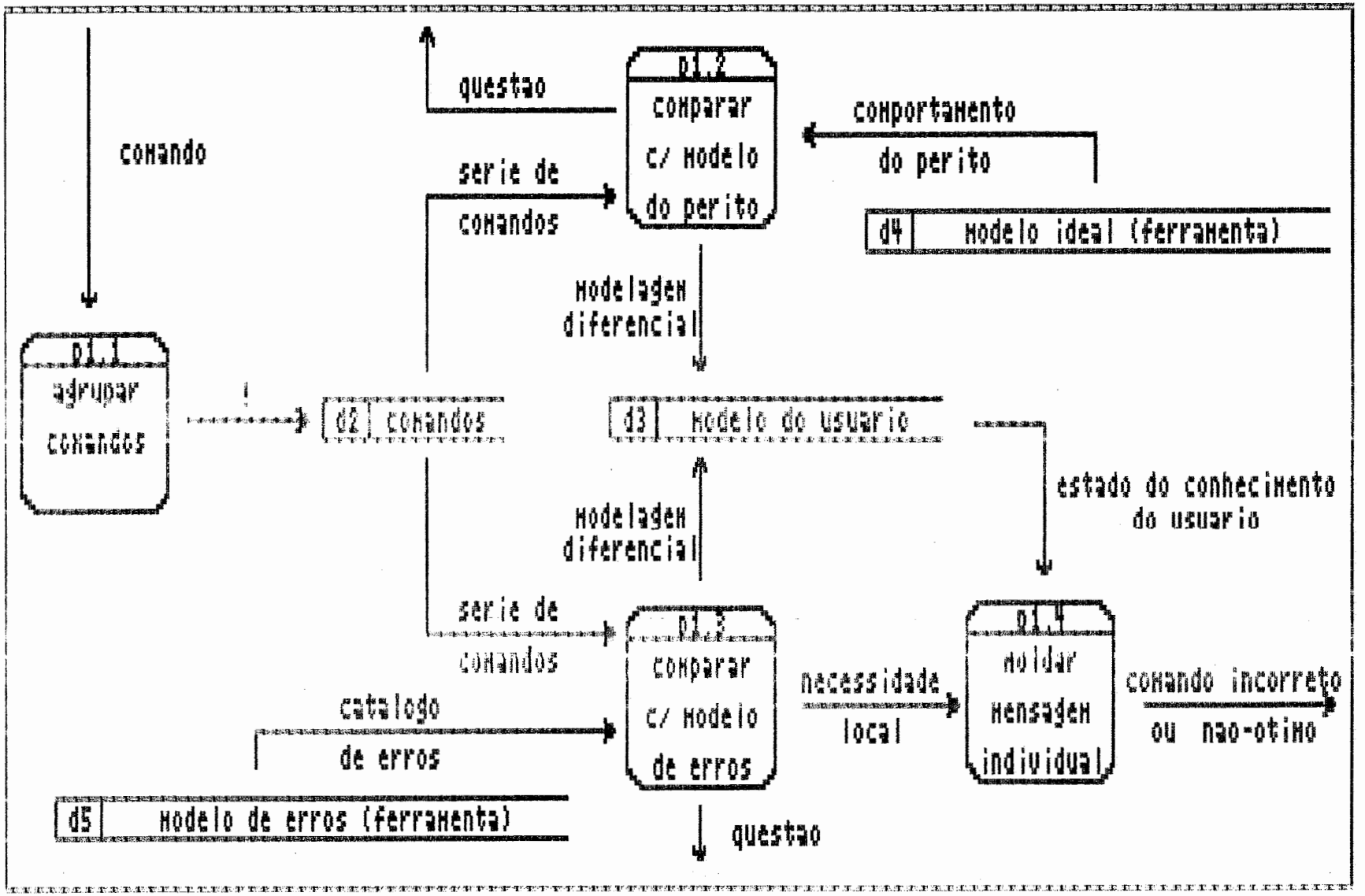


Figura 33. Modelo Logico do Protótipo

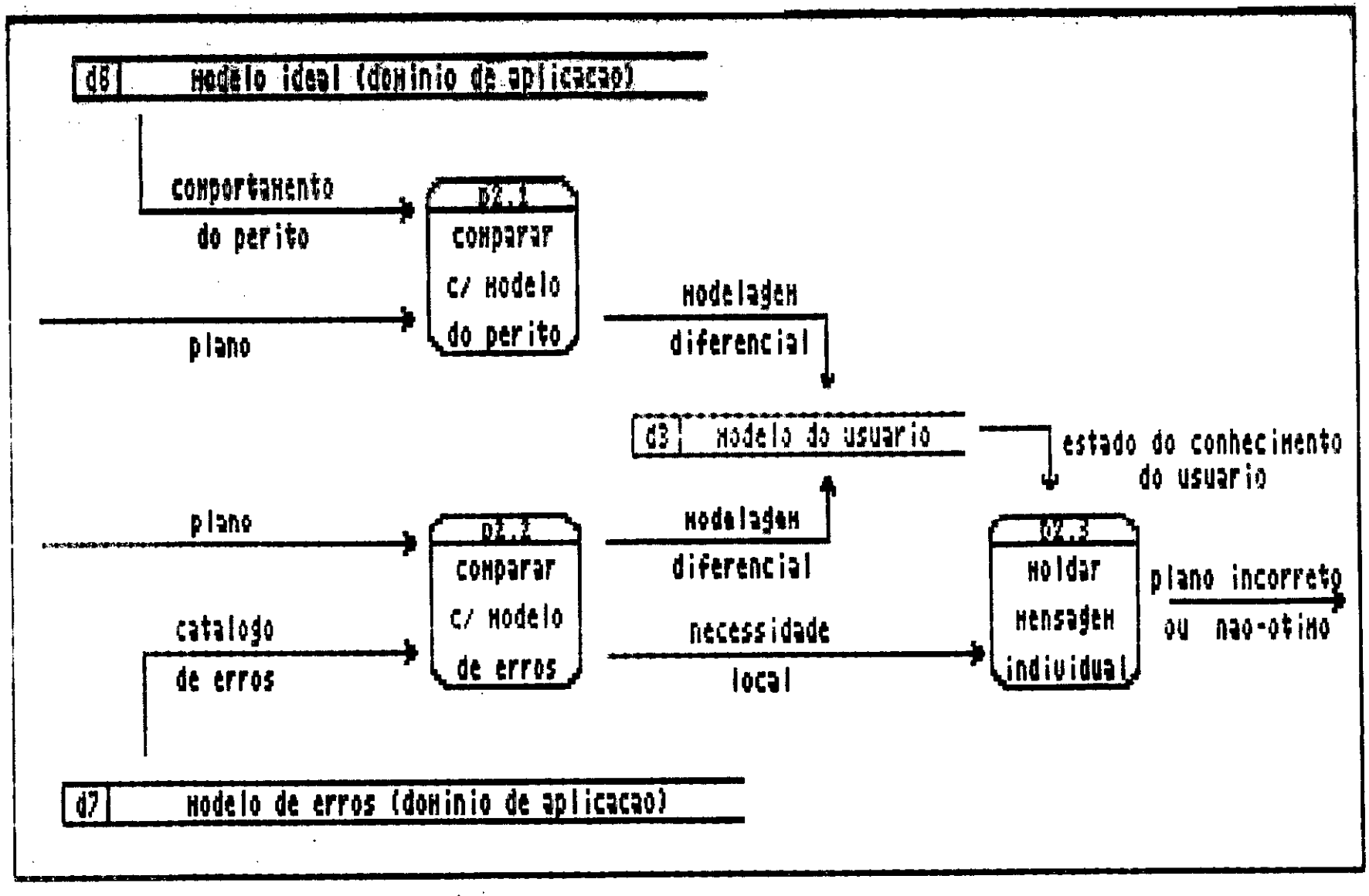


Figura 24. Modelo Logico do Prototipo

O corpo do conhecimento do protótipo retrata uma fatia insignificante comparado ao conhecimento passível de representação. Fato compreensível, uma vez que a intenção do protótipo é apenas testar o paradigma da assistência baseada em conhecimento. Este fato proporciona uma noção da variedade e complexidade do conhecimento necessário para uma assistência eficaz. A maior parcela deste conhecimento nasce da experiência dos especialistas na ferramenta, no método e na área de aplicação.

Um detalhe importante a salientar é a total despreocupação com que foi desenvolvido o protótipo com relação a possíveis futuras utilizações. A única idéia que norteou a construção deste protótipo foi testar os conceitos de assistentes baseados em conhecimento.

Os diskettes com o código objeto e o código fonte do protótipo do assistente baseado em conhecimento para um editor de diagramas de fluxo de dados estão sob a guarda da Prof. Ana Regina, na COPPE/Sistemas - UFRJ.

Para acionar o assistente basta digitar <assist>. Neste momento, o usuário recebe um conjunto de opções de modos de trabalho. O usuário deve optar pelo Modo Tutor, por estar completa a sua implementação. A partir deste ponto, o usuário segue seu plano de solução do problema e, paralelamente, é bombardeado com auxílios sensíveis ao contexto da atividade.

O código fonte do protótipo está dividido em cinco módulos, a saber:

a. *ASSIST.PRO*: Responsável pelas regras do Modo Orientador;

b. *TUTOR.PRO*: Responsável pelas regras do Modo Tutor;

c. *EDIÇÃO.PRO*: Responsável pela edição dos diagramas;

d. *AGENDA.PRO*: Responsável pelas regras da Agenda;

e. *ENTRADA.PRO*: Responsável pela construção da tela de entrada.

7 CONCLUSÃO

O objetivo deste trabalho foi criar subsídios para a construção de assistentes baseados em conhecimento que auxiliem os usuários da estação TABA nas diversas tarefas do processo de desenvolvimento de software. Dentro deste objetivo, nada mais natural do que pensar em um assistente específico para um método e, posteriormente, ampliar os conceitos à nível de Ambientes de Desenvolvimento de Software.

Para conceber uma arquitetura viável para o assistente foram colhidos conceitos nos Sistemas Tutoriais Inteligentes com Orientação e em alguns Assistentes Inteligentes presentes na literatura. Diante das características de ambos os sistemas, várias alternativas foram abertas. As decisões foram tomadas tendo-se em mente a concepção de um assistente eficiente no escopo da Engenharia de Software. A partir deste momento, foi desenvolvido um pequeno protótipo com o intuito de demonstrar a exiquibilidade da técnica de assistência baseada em conhecimento para o domínio da Engenharia de Software.

A construção do protótipo provou a imensa dificuldade de adquirir o conhecimento necessário para torná-lo eficaz. Outro fato importante a salientar é que a escolha de um método mais formal, em oposição a análise estruturada (que representa um método semi-formal), facilitaria a tarefa de caracterização, conceituação e representação das heurísticas do assistente.

A próxima etapa na continuação deste trabalho seria a construção de um protótipo mais apurado, em uma linguagem de Inteligência Artificial que contenha facilidades de interface gráfica. Outra tarefa que merece mais atenção é a extração das heurísticas dos especialistas no método específico. Esta é uma atividade de extrema importância, uma vez que o poder de solução de um sistema baseado em conhecimento emana do corpo do conhecimento que acumula durante a construção.

Neste momento, a ampliação dos conceitos à nível do ambiente TABA ficaria resumida à manipulação da base de conhecimento de acordo com a atividade em questão.

Cabe salientar que este trabalho não apresenta um modelo definitivo. Todos os aspectos devem ser analisados e criticados, uma vez que este estudo abre uma nova frente de trabalho. A idéia de construir assistentes

inteligentes para as atividades da Engenharia de Software é bastante interessante e poderá reduzir significativamente o esforço de construção de software. Finalizando, é um novo agente na busca da essência da Engenharia de Software: "Melhorar a qualidade dos produtos desenvolvidos e aumentar a produtividade e satisfação no trabalho dos Engenheiros de Software".

8 REFERÊNCIAS BIBLIOGRÁFICAS

[ANDERSON 83] Anderson, J. R. *"The Architecture of Cognition"*, Harvard University Press, Cambridge, 1983.

[ANDERSON 85a] Anderson, J. R.; Boyle, C. F.; Yost, G. *"The Geometry Tutor"*, Proceedings of IJCAI-85, Los Angeles, 1985, pp. 1-7.

[ANDERSON 85b] Anderson, J. R.; Reiser, B. J. *"The LISP Tutor"*, Byte, abril 1985, pp. 159-175.

[ANDERSON 89] Anderson, J. R.; Conrad, F. G.; Corbett, A. T. *"Skill Acquisition and the LISP Tutor"*, Cognitive Science, vol. 13, no. 4, New Jersey, outubro-dezembro 1989, pp. 467-505.

[BALZER 85] Balzer, R. *"A 15 Year Perspective on Automatic Programming"*, IEEE Transactions on Software Engineering, vol. SE-11, no. 11, novembro 1985, pp. 1257-1267.

[BARR 82] Barr, A.; Feigenbaum, E. A. *"The Handbook of Artificial Intelligence"*, William Kaufman, vols. 1-2, Los Altos, 1982.

[BARSTOW 87] Barstow, D. *"Artificial Intelligence and Software Engineering"*, 9th International Conference on Software Engineering, Monterey, 30 março-2 abril 1987, pp. 200-211.

[BASILI 86] Basili, V. R.; Selby, R. W.; Hutchens, D. W. *"Experimentation in Software Engineering"*, IEEE Transactions on Software Engineering, vol. SE-12, no. 7, julho 1986, pp. 733-744.

[BAUER 72] Bauer, F. L. *"Software Engineering"*, Information Processing 71, North Holland Publishing Co., Amsterdam, 1972.

[BOBROW 85] Bobrow, D. G. *"If Prolog Is the Answer, What Is the Question? Or What It Takes to Support AI Programming Paradigms?"*, IEEE Transactions on Software Engineering, vol. SE-11, no. 11, novembro 1985, pp. 1401-1408.

[BOEHM 76] Boehm, B. W. *"Software Engineering"*, IEEE Transactions on Computers, vol. C-25, no. 12, dezembro 1976.

[BREUKER 87] Breuker, J.; Winkels, R.; Sandberg, J. *"A Shell for Intelligent Help Systems"*, Proceedings of IJCAI-87, 1987, pp. 167-173.

[BUCHANAN 85] Buchanan, B. G. *"Artificial Intelligence: Towards Machines that Think"*, Encyclopedia Britannica, Yearbook of Science and the Future, 1985.

[CAMELO 88] Camelo, M. F.; Junior, A. M. *"Um Tutor Inteligente em Oftalmologia"*, 5o. Simpósio Brasileiro de Inteligência Artificial, Natal, 7-11 novembro 1988, pp. 151-160.

[CASANOVA 87] Casanova, M. A.; Giorno, F. A. C.; Furtado, A. L. *"Programação em Lógica e a Linguagem Prolog"*, Editora Edgard Blucher Ltda, São Paulo, 1987.

[CHARETTE 86] Charette, R. N.; *"Software Engineering Environments - Concepts and Technology"*, McGraw-Hill Inc., New York, 1986.

[CHILDERS 84] Childers, P. G.; Schank, R. C. *"The Cognitive Computer on Language, Learning and Artificial Intelligence"*, Addison-Wesley Publishing Company, Massachusetts, 1984.

[CLANCEX 87] Clancex, W. J. *"Knowledge-Based Tutoring - The GUIDON Program"*, The MIT Press, Massachusetts, 1987.

[COPPE 87] COPPE/UFRJ, UFRGS e outros. *"Uma Estação de Trabalho Heurística para o Engenheiro de Software"*, Proposta de participação no Projeto ETHOS, Florianópolis, setembro 1987.

[ELIOT 86] Eliot, L. B.; Scácchi, W. *"Towards a Knowledge-Based System Factory: Issues and Implementation"*, IEEE Expert, vol. 1, no. 4, 1986, pp. 51-58.

[FAIRLEY 85] Fairley, R. E. *"Software Enginnering Concepts"*, International Student Edition, McGraw-Hill Company, Singapore, 1985.

[FIKES 81] Fikes, R. E. *"ODYSSEY: A Knowledge-Based Assistant"*, Artificial Intelligence, vol. 16, 1981, pp. 331-361.

[FININ 82] Finin, T. W. *"Providing Help and Advice in Task Oriented Systems"*, Report MS-CIS-1982-22, Computer and Info. Science, University of Pennsylvania, Philadelphia, 1982.

[FRENZEL 87] Frenzel, L. E. *"Crash Course in Artificial Intelligence and Expert Systems"*, Howard W. Sams Co., Indianapolis, 1987.

[GANE 83] Gane, C.; Sarson, T. *"Análise Estruturada de Sistemas"*, LTC - Livros Técnicos e Científicos Editora S.A., Rio de Janeiro, 1983.

[GUARANYS 87] Guaranys, P. Y. *"IN-EDITO - Interface Inteligente para um Editor de Texto"*, Tese Msc., Departamento de Informática, PUC/RJ, Rio de Janeiro, março 1987.

[HARMON 87] Harmon, P. *"Artificial Intelligence and Instruction - Applications and Methods"* (Editado por Greg Kearsley), Addison-Wesley Publishing Company, Massachusetts, 1987, pp. 165-190.

[HARMON 88] Harmon, P.; King, D. *"Sistemas Especialistas - A Inteligência Artificial Chega ao Mercado"*, Editora Campus Ltda, Rio de Janeiro, 1988.

[HU 87] Hu, S. D. *"Expert Systems for Software Engineers and Managers"*, Chapman and Hall, London, 1987.

[IEEE SOFTWARE 88] IEEE Software, novembro 1988, pp. 15-16.

[JENSEN 79] Jensen, R. W.; Tonies, C. C. *"Software Engineering"*, Prentice-Hall Inc., New Jersey, 1979.

[KAISER 87] Kaiser, G. E.; Feiler, P. H. *"An Architecture for Intelligent Assistance in Software Development"*, 9th International Conference on Software Engineering, Monterey, 30 março-2 abril 1987, pp. 200-211.

[KEARSLEY 87] Kearsley, G. *"Artificial Intelligence and Instruction - Applications and Methods"*, Addison-Wesley Publishing Company, Massachusetts, 1987.

[KELLER 87] Keller, R. *"Expert System Technology - Development and Application"*, Prentice-Hall Inc., New Jersey, 1987.

[LAMSWEERDE 87] Lamsweerde, A. V.; Dufour, P. *"Current Issues in Expert Systems"*, Academic Press Inc., San Diego, 1987.

[LAWLER 87] Lawler, R. W.; Yazdani, M. *"Artificial Intelligence and Education"*, Ablex Publishing, vol. 1, New Jersey, 1987.

[LUCENA 87] Lucena, C. *"Inteligência Artificial e Engenharia de Software"*, Jorge Zahar Editor, Rio de Janeiro, 1987.

[MISHKOFF 86] Mishkoff, H. C. *"Understand Artificial Intelligence"*, Texas Instruments Learning Center, Dallas, 1986.

[MOSTOW 85] Mostow, J. *"What is AI? And What Does it Have to Do with Software Engineering?"*, IEEE Transactions on Software Engineering, vol. SE-11, no. 11, novembro 1985, pp. 1253-1256.

[O'SHEA 83] O'Shea, T.; Self, J. *"Learning and Teaching with Computers - Artificial Intelligence in Education"*, Prentice-Hall Inc., Englewood Cliffs, 1983.

[OLIVEIRA 88] Oliveira, U. S. C. *"Um Sistema Tutorial Inteligente"*, Tese Msc., Instituto Militar de Engenharia, Rio de Janeiro, fevereiro 1988.

[PAGE-JONES 90] Page-Jones, M. *"Gerenciamento de Projetos"*, Editora McGraw-Hill Ltda e Newstec Editora Ltda, São Paulo, 1990.

[PUNCELLO 88] Puncello, P. P. et alli. *"ASPIS: A Knowledge-Based Case Environment"*, IEEE Software, março 1988, pp. 58-65.

[RICH 88] Rich, E. *"Inteligência Artificial"*, Editora McGraw-Hill, São Paulo, 1988.

[RIDGE 88] Ridge, J. C.; Tsai, J. J. P. *"Intelligent Support for Specifications Transformation"*, IEEE Software, novembro 1988, pp. 28-35.

[ROCHA 87] Rocha, A. R. C. *"Análise e Projeto Estruturado de Sistemas"*, Editora Campus Ltda, Rio de Janeiro, 1987.

[ROCHA 88] Rocha, A. R. C.; Souza, J. M. *"Ambientes de Desenvolvimento de Software e o Projeto TABA"*, Seminário do Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Rio de Janeiro, dezembro 1988.

[ROCHA 89] Rocha, A. R. C.; Aguiar, T. C.; Souza, J. M.; D'Ipólito, C. *"O Meta-ambiente da Estação TABA"*, Relatório Técnico do Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Rio de Janeiro, março 1989.

[ROCHA 90] Rocha, A. R. C. et alli. *"Uma Visão Funcional da Estação Taba"*, Relatório Técnico em Preparação do Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Rio de Janeiro, dezembro 1990.

[ROLSTON 88] Rolston, D. W. *"Principles of Artificial Intelligence and Expert Systems Development"*, McGraw-Hill Inc., New York, 1988.

[RONDON 88] Rondon, M. A. *"Um Sistema de Auxílio Baseado em Esteredtipos"*, 5o. Simpósio Brasileiro de Inteligência Artificial, Natal, 7-11 novembro 1988, pp. 200-215.

[SCHOEN 88] Schoen, E.; Smith, R. G.; Buchanan, B. G. *"Design of Knowledge-Based Systems with a Knowledge-Based Assistant"*, IEEE Transactions on Software Engineering, vol. SE-14, no. 12, dezembro 1988, pp. 1771-1791.

[SHORTLIFFE 84] Shortliffe, E. H.; Buchanan, B. G. *"Rule-Based Expert Systems"*, Addison-Wesley Publishing Company, Reading, 1984.

[SIMON 86] Simon, H. A. *"Whether Software Engineering Needs to Be Artificially Intelligent"*, IEEE Transactions on Software Engineering, vol. SE-12, no. 7, julho 1986, pp. 726-732.

[SOMMERVILLE 82] Sommerville, I. *"Software Enginnering"*, Addison-Wesley Publishers, London, 1982.

[STEPHENS 85] Stephens, M.; Whitehead, K. *"The Analyst - A Workstation for Analysis and Design"*, 8th International Conference on Software Engineering, London, 28-30 agosto 1985, pp. 364-371.

[SUBRAHMANYAM 85] Subrahmanyam, P. A. *"The Software Engineering of Expert Systems: Is Prolog Appropriate?"*, IEEE Transactions on Software Engineering, vol. SE-11, no. 11, novembro 1985, pp. 1391-1400.

[TSAI 86] Tsai, W. T.; Volovik, D.; Zualkernan, I. *"Expert Systems and Software Engineering: Ready for Marriage?"*, IEEE Expert, vol. 1, no. 4, 1986, pp. 24-30.

[WALKER 87] Walker, A. et alli. *"Knowledge Systems and PROLOG"*, Addison-Wesley Publishing Company, Massachusetts, 1987.

[WALTERS 86] Walters, R. C. *"The Programmer's Apprentice: A Session with KBEmacs"*, Reading in Artificial Intelligence and Software Engineering, Morgan Kaufmann Publishers Inc., 1986, pp. 351-375.

[WALTERS 88] Walters, R. C.; Rich, C. *"The Programmer's Apprentice Project"*, IEEE Computer, vol. 21, no. 11, novembro 1988, pp. 11-25.

[WATERMAN 86] Waterman, D. A. *"A Guide to Expert Systems"*, Addison-Wesley Publishing Company, Massachusetts, 1986.

[WEITZEL 89] Weitzel, J. R.; Kerschberg, L. *"Development Knowledge-Based Systems: Reorganizing the System Development Lyfe Cicle"*, Communications of ACM, vol. 32, no. 4, abril 1989, pp. 482-488.

[WENGER 87] Wenger, E. *"Artificial Intelligence and Tutoring Systems - Computational and Cognitive Approaches to the Comunication of Knowledge"*, Morgan Kaufmann Publishers Inc., California, 1987.

[WERNECK 88] Werneck, V. M. B.; Assis, S. G.; Matos, J. P.; Aguiar, T. C. *"Especificador de Ambientes da Estação Taba: Fase de Identificação"*, Relatório Técnico do Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Rio de Janeiro, julho 1989.

[WILLIAMS 86] Williams, C. *"Expert Systems, Knowledge Engineering and AI Tools - An Overview"*, IEEE Expert, vol. 1, no. 4, 1986, pp. 66-70.

[WINSTON 79] Winston, P. H.; Brown, R. H. *"Artificial Intelligence: An MIT Perspective"*, The MIT Press, vol. 1, Massachussetts, 1979.

[WOOLF 87] Woolf, B. P. *"Artificial Intelligence and Instruction - Applications and Methods"* (Editado por Greg Kearsley), Addison-Wesley Publishing Company, Massachusetts, 1987, pp. 229-265.

[WOOLF 88] Woolf, B. *"Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence"* (Editado por Howard E. Shrobe), Morgan Kaufmann Publishers Inc., Califórnia, 1988, pp. 1-43.

[YOURDON 89] Yourdon, E. *"Revisões Estruturadas"*, Editora Campus, Série Yourdon Press, Rio de Janeiro, 1989.