Universidade Estadual de Campinas - UNICAMP
Faculdade de Engenharia Elétrica - FEE
Departamento de Engenharia da Computação e Automação Industrial

AEsp: Um Assistente de Especificação para Administração Rural

Silvia Maria Fonseca Silveira Massruhá

Orientador: Prof. Mário Dias Ferraretto
Co-Orientador: Prof. Mário Jino

Dissertação de Tese apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, como requisito parcial para obtenção do título de Mestre.

Janeiro de 1996

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

M388a

Massruhá, Silvia Maria Fonseca Silveira
AEsp: um assistente de especificação para administração rural / Silvia Maria Fonseca Silveira Massruhá.--Campinas, SP: [s.n.], 1996.

Orientadores: Mário Dias Ferraretto, Mario Jino.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica.

1. Análise de sistemas (Agricultura). 2. Aquisição de conhecimento (Sistemas especialistas). 3. Programas de computador - Reutilização. 4. Administração rural. I. Ferraretto, Mário Dias. II. Jino, Mario. III.Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica. IV. Título.

A Francisco, Nezita,
Ério e "Karina"

Agradecimentos

À Empresa Brasileira de Pesquisa Agropecuária - EMBRAPA, especialmente ao Álvaro Seixas Neto, chefe do Centro Nacional de Pesquisa Tecnológica em Informática para Agricultura - CNPTIA, pelo apoio fundamental à realização deste trabalho.

Ao Prof. Dr. Mário Jino pelo importante incentivo, colaboração e orientação durante todo o decorrer deste trabalho.

Ao Prof. Dr. Mário Dias Ferraretto pela valiosa orientação, enriquecida de sua confiança, apoio e amizade, e pela sua grande contribuição para meu aperfeiçoamento e amadurecimento profissional.

A todos os colegas do CNPTIA, em especial a Maria Angélica, Carla Geovana e Paulo César, pelo apoio e amizade.

A todo o grupo do Projeto FMS pela ajuda, em especial ao Carlos, Visoli e Sérgio que ajudaram diretamente no enriquecimento do trabalho do AEsp.

Aos meus irmãos, Luiz Fernando, Vania/Carlos , Júnior/Adriane e Beto e tias, Dinha, Julinha e Ana Maria que com muito carinho e amizade colaboraram para que este trabalho chegasse ao final.

À minha filha "Karina" e aos meus sobrinhos, Camila e Caíque que, de alguma forma, gostaria que este trabalho servisse de exemplo.

Ao meu esposo Ério, cuja presença, carinho, amor, incentivo e paciência tornaram a elaboração deste trabalho muito mais fácil.

Aos meus pais, Francisco e Nezita, pelo apoio, afeto e compreensão sem os quais seria impossível a realização de mais este passo.

Sumário

A produção de software na área de administração rural envolve um número crescente de aplicações semelhantes porém não iguais. Das propostas sugeridas pela Engenharia de Software para a captura de especificações de aplicações desta natureza, este trabalho adotou as técnicas de abordagem por domínio. Este método permite equacionar o problema de múltiplas especificações de sistemas parecidos oferecendo algumas oportunidades para reuso e desenvolvimento por prototipação.

Este trabalho descreve uma ferramenta, denominada AEsp, e um método de utilização que visa auxiliar a captura das especificações das aplicações do domínio de administração rural e transformá-las, incrementalmente, em uma representação que pode ser traduzida para um programa operacional. Esta ferramenta está sendo desenvolvida no âmbito do Projeto FMS[1], no CNPTIA/EMBRAPA[2], que visa construir um processo produtivo para a classe de aplicações do domínio de administração rural a um custo decrescente com o tempo.

[1] Projeto FMS : "Farm Management Systems" - Ambiente de Desenvolvimento de Software para o Domínio de Administração Rural

[2] CNPTIA - Centro Nacional de Pesquisa Tecnológica em Informática para Agricultura
EMBRAPA- Empresa Brasileira de Pesquisa Agropecuária

Abstract

The software production for the farm management area involves an increasing number of similar, but not equal, applications. From the suggested application specification aquisition approaches in the Software Engineering field for this kind of applications, the domain oriented analysis technique was adopted in this work. This method allows structuring the problem of multiple specifications of similar systems providing same oportunities for reuse and prototyping.

This dissertation describes a tool, called AEsp, and a tool utilization method that aims to help on capturing farm management applications specifications and converts them, incrementally, to a representation that can be translated to a operacional program. This tool is under development in the FMS[1] Project, at CNPTIA/EMBRAPA[2], and aims to build a productive process for the farm management applications class with a decreasing cost in time.

[1] Projeto FMS : "Farm Management Systems" - Ambiente de Desenvolvimento de Software para o Domínio de Administração Rural

[2] CNPTIA - Centro Nacional de Pesquisa Tecnológica em Informática para Agricultura
EMBRAPA- Empresa Brasileira de Pesquisa Agropecuária

Índice

1) Introdução

1.1) Contexto do trabalho 1

1.2) Abordagem da dissertação 2

1.3) Organização da dissertação 3

2) Revisão Bibliográfica

2.1) Introdução 4

2.2) Assistentes de Especificação 4

2.3) Exemplos de Assistentes de Especificação 6

2.4) Quadro Comparativo entre Assistentes 13

2.5) Análise Geral 20

3) Assistente de Especificação do FMS - AEsp

3.1) Introdução 21

3.2) AEsp no contexto do FMS 21

3.3) Características do AEsp 24

3.4) Considerações sobre o Capítulo 25

4) Modelo Conceitual do AEsp

4.1) Introdução 26

4.2) Abordagem do AEsp 26

4.3) Modelo do Processo de Entrevista do AEsp 27

4.3.1) Metodologia de Decomposição - HOS 27

4.3.2) Caracterização da Entrevista 29

4.4) Um exemplo de nó-primitivo - "Controlar" 36

4.5) Considerações sobre o Capítulo 46

5) Modelo de Implementação do AEsp

5.1) Introdução 47

5.2) Arquitetura Geral do AEsp 47

5.3) Entrevistador 48

5.3.1) Implementação do Entrevistador 49

5.3.2) Base de Conhecimento do AEsp 62

5.4) Outras ferramentas do AEsp 64

5.4.1) Catalogador 64

5.4.2) Reutilizador 65

5.4.3) Prototipador 67

5.4.4) Montador 70

5.5) Plataforma de Suporte Utilizada 73

5.6) Considerações sobre o Capítulo 73

6) Exemplo de uma especificação sob o AEsp

6.1) Introdução 74

6.2) Entrevistador e sua operação 74

6.3) Prototipador e sua operação 89

6.4) Montador e sua operação 93

6.5) Considerações sobre o Capítulo 93

7) Conclusões e Trabalhos Futuros

7.1) Conclusões e Resultados Obtidos 94

7.2) Trabalhos Futuros 95

8) Referências Bibliográficas 96

9) Apêndice A 101

Lista de Figuras

Figura 2.1: Paradigma de desenvolvimento de software automatizado 14

Figura 2.2: Mapeamento das etapas do ciclo de vida no paradigma de desenvolvimento de software automatizado 14

Figura 3.1: Modelo do FMS sob o modelo do Lehman 22

Figura 3.2: Hierarquização do Nível Operacional 22

Figura 3.3: Escopo do AEsp 24

Figura 3.4: Ferramentas do AEsp 25

Figura 4.1: Arquitetura Geral do AEsp 26

Figura 4.2: Árvore de Decomposição HOS - "Control Map" 28

Figura 4.3: "Control Map" - Caracterização da Entrevista 29

Figura 4.4: "Control Map" Instanciáveis - CMI 34

Figura 4.5: Processo de Tradução da Entrevista 35

Figura 4.6: Modelo do Nó "Controlar" 36

Figura 4.7(a): CMI do "Controlar" 41

Figura 4.7(b): CMI do "Controlar" (continuação) 42

Figura 4.7(c): CMI do "Controlar" (continuação) 43

Figura 4.8(a): CMO - Criar Nível de Representação 44

Figura 4.8(b): CMO - Criar Nível de Representação (continuação) 45

Figura 4.9: "Control Map" Instanciado - CMP 46

Figura 5.1: Arquitetura Geral do AEsp 47

Figura 5.2: Interface do AEsp 48

Figura 5.3: Entrevistador 48

Figura 5.4: Representação Gráfica da Classe Funções em Nexpert 50

Figura 5.5: Classe Objetos do Dominio 51

Figura 5.6: Representação Gráfica da Classe Forms em Nexpert 51

Figura 5.7: Especificação em HOS do CMO - Popular CMP 53

Figura 5.8: Exemplo de uma Regra em Nexpert pertencente à implementação do CMO - Popular CMP 54

Figura 5.9: Exemplo de uma Regra em Nexpert referente a uma estrutura de controle da Metodologia HOS 55

Figura 5.10(a): CMI - Criar CMP Instanciado.....56

Figura 5.10(b): Criar CMP da Função (continuação do CMI - Criar CMP Instanciado) 56

Figura 5.10(c): Editar CMP da Função (continuação do CMI - Criar CMP Instanciado) 57

Figura 5.10(d): CMO - Decompor Função em Subfunções.....58

Figura 5.10(e): CMO - Consultar Base < Nome _Base >.....59

Figura 5.11: Exemplo do CMI "Controlar".....60

Figura 5.12: Criar _Nó_H - Exemplo de regra em Nexpert.....61

Figura 5.13: Características da Base Funções.....63

Figura 5.14: Catalogador.....64

Figura 5.15: Primitivas e Sinônimos.....65

Figura 5.16: Reutilizador.....66

Figura 5.17: Prototipador.....67

Figura 5.18: Interface do Prototipador.....68

Figura 5.19: Propriedades de Campos.....69

Figura 5.20: Propriedades de Formulários.....69

Figura 5.21: Montador.....70

Figura 6.1: Tela Inicial do Entrevistador.....74

Figura 6.2: CMI - Criar CMP Instanciado.....75

Figura 6.3(a): Editar CMP da Função (continuação do CMI - Criar CMP Instanciado).76

Figura 6.3(b): Tela apresentando as opções Atualizar, Deletar e Prosseguir 76

Figura 6.4(a): Criar CMP da Função (continuação do CMI - Criar CMP Instanciado)..77

Figura 6.4(b): CMO - Consultar _Base_ Primitivas 78

Figura 6.5: Control Map Instanciado 84

Figura 6.6: Decomposição de Dados 84
Figura 6.7(a): Representação em Nexpert da classe Forms 85
Figura 6.7(b): Representação em Nexpert da classe Campos 85
Figura 6.7(c): Representação em Nexpert da classe Textos 86
Figura 6.7(d): Representação em Nexpert da classe Transições 86
Figura 6.7(e): Representação em Nexpert da classe Relatórios 87
Figura 6.7(f): Representação em Nexpert da classe Eventos 87
Figura 6.8: Formato da base Forms.nxp 88
Figura 6.9: Tela Inicial do Prototipador 89
Figura 6.10: Formulário "Menu0"e sua tela de propriedades 90
Figura 6.11: "Menu1" com as propriedades da transição "Controlar Rebanho da Fazenda" 91
Figura 6.12(a): "Menu2" com tela de propriedades do formulário 91
Figura 6.12(b): "Menu2" com tela de propriedades do formulário alterada 92
Figura 6.13: Tela "Perfil" e propriedades do campo "Brinco" 92
Figura 6.14: Exemplo de especificações geradas pelo Montador 93

Lista de Tabelas

Figura 4.1: Suporte à modelagem da Entrevista.....31

Figura 4.2: Operadores de "Control Maps".....32

Figura 4.3: Exemplos de Coletivos.....38

CAPÍTULO 1

INTRODUÇÃO

O objetivo deste capítulo é delimitar o problema de desenvolvimento de software para o domínio de administração rural, objetivo último do AEsp, e apresentar como esta ferramenta pode auxiliar na captura e especificação das aplicações deste domínio. Em seguida, está descrita a organização deste trabalho.

1.1) Contexto do trabalho

O projeto FMS - "Farm Mangement Systems", onde está sendo desenvolvido o AEsp, visa a construção de um Ambiente de Desenvolvimento de Software para o Domínio de Administração Rural. Este projeto tem como objetivo mais específico o desenvolvimento e integração de metodologias e ferramentas para a produção automatizada de aplicativos para o domínio de administração rural, buscando sempre a qualidade e produtividade na produção de software, a custos mínimos.

A produção de programas de computador a baixo custo é tão importante economicamente que muitas panacéias foram propostas e sonoramente refutadas nos últimos 30 anos. A depuração dos resultados mostra que ganhos de produtividade têm sido lentos e que os resultados mais promissores foram alcançados quando se localizou o esforço numa área pequena e bem delimitada [FER93].

Os sistemas de apoio a gestão da pequena ou média propriedade agroindustrial têm características que os descreveriam, sob a ótica de engenharia de software, como pequenos, com pouca sofisticação de funções, baixa complexidade de dados e demandando uma interface com usuários simples e padronizada.

Por outro lado, a importância deste tipo de apoio não pode ser minimizado - é inconcebível que o administrador de uma pequena propriedade agrícola, básica na agricultura brasileira, não tome decisões amparado em informação quantificada e agregada sobre sua produção. Além disso, as informações sobre suas atividades devem ser atualizadas e confiáveis de modo que possam ser inseridas nos sistemas de planejamento da agricultura.

O projeto FMS visa construir um processo para essa classe de programas a um custo decrescente com o tempo, transformando a experiência de cada produtor no ponto de partida do próximo sistema. Os sistemas gerados cobrirão algumas classes de aplicações e terão as características descritas acima. Estes programas serão especificados pelo produtor/extensionista, gerados automaticamente e dispensarão a maior parte do ciclo de manutenção.

Gerar esses programas de forma automatizada é essencial. O custo de produção de um elenco enorme de aplicações nos moldes convencionais gera um problema econômico insolúvel. A geração automática de programas, mesmo que estes sejam simples e de aplicação bastante localizada, exige um elenco grande de ferramentas e tecnologias e é um problema técnico bastante atual.

A faceta "gerador de aplicações" do FMS exige a utilização de técnicas de compilação, projeto lógico e físico de banco de dados e projeto de linguagens e metodologias de desenvolvimento de sistemas [CLE88,MEI91]. A faceta "capturador de

especificações" do FMS demanda a integração de gerador de interfaces, bases de conhecimento e objetos, técnicas de análise de domínio e reuso. A agregação dessas facetas tecnológicas na forma de um sistema funcional exige metodologias de integração de sistemas que é uma das correntes atuais, em termos de publicação, em Engenharia de Software [YEH91].

A faceta "capturador de especificações" é o tema abordado neste trabalho. A transformação de especificações do usuário em especificações formais é ainda um problema em aberto. Os trabalhos de Balzer [BAL86] sobre este tópico mostram que processamento de especificações escritas em linguagem natural está longe de ser resolvido. A abordagem do FMS foi evitar o processamento automático de linguagem natural e sim usar um engenheiro de especificação para isso. Assistentes de especificação são programas que apoiam o processo de captura dessas especificações [ARI92,CZU88,GOM92A,KAR88,PUN88,SCH88].

No método preconizado pelo FMS o usuário é entrevistado pelo engenheiro de especificação, que opera o programa assistente de especificação (AEsp) para documentar o processo de decomposição da aplicação e oferecer aos usuários soluções já desenvolvidas e armazenadas durante sessões prévias de especificação.

Este método de reuso é aderente ao modelo de domínio usado pelo FMS [MAS94B]. A metodologia de decomposição de problemas para obtenção da especificação é um problema central da engenharia de software e o FMS adotou uma metodologia já conhecida na literatura, denominda "High-Order" Software - HOS [HAM76A,HAM76B,MAR85A].

A construção de um sistema de apoio à captura de especificações, enquanto diminui muito o trabalho de anotação das informações do usuário, não elimina uma das fontes mais importantes de imprecisões na produção de programas: o entendimento errado, pelo engenheiro de especificação, da demanda do usuário [ASL91]. A inexistência de uma teoria formalizada para atacar esse problema tem levado muitos sistemas a adotarem metodologias de prototipação [JOR89,LUQ88,LUQ89]. O FMS seguiu este caminho e construiu ferramentas que, a qualquer momento do processo de especificação, permitem que o usuário tenha uma visão do que já foi capturado.

Deve-se notar que o almejado pelo projeto FMS é baseado em processos com fundamentação estabelecida na literatura - criar um ambiente para a geração mais automatizada possível de programas para apoio à gestão da propriedade rural. O AEsp é uma das ferramentas deste ambiente com a função de capturar as especificações do usuário e auxiliar na sua tradução para esses programas.

1.2) Abordagem da dissertação

O Assistente de Especificação (AEsp) proposto no projeto FMS é um programa para apoiar o processo para captura, documentação e tradução, para um nível operacional, das especificações das aplicações do domínio de Administração Rural [MAS94B].

O AEsp é baseado em vários modelos de atividades típicas do estágio onde é usado: modelo de decomposição sistemática das funções e dados, modelo de entrevista e um modelo de documentação e reuso da especificação coletada. A premissa básica de construção do AEsp é que ele não é usado para processamento de linguagem natural ou dialetos desta. O entendimento do diálogo entre o especificador (o especialista do domínio ou usuário) e o coletor da informação (o Engenheiro de Especificação que

manipula o AEsp) é atividade tipicamente humana. A remoção das ambiguidades é feita pela experiência do Engenheiro de Especificação apoiado no processo de prototipação imediata oferecida pelo sistema.

Metodologias já estabelecidas de decomposição e entrevista foram estudadas e implementadas em um ambiente comercial de gerenciamento de bases de conhecimento. Nesta etapa também foram identificados e implementados os "nós primitivos" da decomposição. Essa identificação é que torna possível a concretização do processo abstrato de especificação (decomposição) na forma de um programa interpretável em um nível de representação.

O resultado da especificação é a associação de um verbo (ação) e seus objetos (operandos) a um elemento existente no nível de representação. Essa identificação é armazenada perenemente pelo sistema e, através de sinonímia (suportada pelo sistema e pela intuição do Engenheiro de Especificação), é a base do mecanismo de reuso. O resultado final do processo de coleta de especificação é um conjunto de "nós primitivos" que são visitados e são a base de geração da aplicação na forma de um programa escrito em uma linguagem denominada LC-FMS. Alguns nós já identificados, apresentados nos Capítulos 4 e 5, mostram bom potencial de expressividade.

No decorrer desta dissertação, estes conceitos poderão ser vistos mais detalhadamente.

1.3) Organização da dissertação

O Capítulo 2 apresenta uma revisão bibliográfica dos assistentes de especificação existentes para auxiliar na especificação e dimensionamento do AEsp. Para tanto, é feita uma descrição breve de vários assistentes e em seguida, estes são comparados com o AEsp utilizando-se alguns critérios estabelecidos.

O Capítulo 3 descreve o enfoque do AEsp no contexto do ambiente FMS e suas características. Neste capítulo também são mencionadas as ferramentas que compõem a arquitetura do AEsp: Entrevistador, Catalogador, Reutilizador, Prototipador e Montador.

O quarto capítulo descreve o modelo conceitual do AEsp, enfatizando o modelo de entrevista e decomposição, onde está baseada a sua implementação.

O quinto capítulo apresenta o processo de construção do AEsp, destacando o modelo de implementação do Entrevistador, que é o cerne do AEsp.

No capítulo 6 é apresentado um exemplo da especificação de uma aplicação sob o AEsp. Finalmente, no Capítulo 7 estão descritos os resultados obtidos e os trabalhos futuros propostos para utilização desta ferramenta.

CAPÍTULO 2

REVISÃO BIBLIOGRÁFICA

2.1) Introdução

O objetivo deste capítulo é fazer uma revisão bibliográfica sobre alguns assistentes de especificação existentes, considerando os seguintes aspectos:

- o tema central desta dissertação é um Assistente de Especificação para Sistemas de Administração Rural;

- os assistentes, aqui considerados, são aqueles que auxiliam nas primeiras etapas do ciclo de vida do software (assistente de requisitos, assistente de especificação, assistente de projeto) [MAS94A];

- o modelo de ciclo de vida evolucionário aderente a estas ferramentas;

- os métodos, técnicas e ferramentas utilizadas na formalização das etapas do ciclo de vida do software dos assistentes;

- as ferramentas de software que compõem a arquitetura de um assistente;

- a plataforma de suporte para auxiliar na implementação de um assistente;

- as metodologias de engenharia de software utilizadas no desenvolvimento de um assistente, tais como análise de domínio, reuso, componentização e prototipação.

2.2) Assistentes de Especificação

Na literatura sobre Engenharia de Software [PRE92] existem propostas de ferramentas de software, denominadas CASE [FIS92][GOM92B][KAR88][PRE91], cujo objetivo é automatizar o processo de produção de software. Algumas destas ferramentas combinam métodos e técnicas, de engenharia de software e inteligência artificial, em uma infraestrutura comum para suportar todas as atividades do ciclo de vida visando melhorar a qualidade e a produtividade do software.

Uma das etapas mais importantes e problemáticas do ciclo de vida do software para ser automatizada é a etapa de especificação dos requisitos [BOR86][GRE84]. Os erros ocorridos nesta etapa tornam-se extremamente caros para serem corrigidos nas etapas posteriores do ciclo de vida [BOE81].

Segundo Aslett [ASL91] a dificuldade de comunicação entre o usuário e o desenvolvedor de sistema é um fator limitante na etapa de especificação de requisitos que compromete a confiabilidade do software produzido. O usuário é um especialista do domínio da aplicação e não conhece a tecnologia e a terminologia utilizada pelo desenvolvedor do sistema e vice-versa.

Além disso, as etapas do ciclo de vida (conversão de requisitos em especificação, especificação em implementação) são mal documentadas, fazendo com que as informações e o raciocínio que está por trás de cada passo não estejam disponíveis nas etapas seguintes. Estes problemas de comunicação e documentação tornam a especificação de requisitos errônea, incompleta e ambígua.

Waters e Rubenstein [WAT91] falam sobre a transformação da especificação informal de requisitos em especificação formal e apontam seis informalidades típicas na comunicação entre o usuário e o analista: abreviação, ambiguidade, ordenação, contradição, incompleteza e baixa confiabilidade.

Desta forma, a transformação dos requisitos dos usuários em especificações formais tem sido um problema em aberto. Os trabalhos de Balzer [BAL86] sobre este tópico mostram que o processamento de especificações escritas em linguagem natural está longe de ser resolvido.

Assistentes de Especificação são ferramentas que compõem ambientes de software automatizados (CASE) para auxiliar na formalização da etapa de captura das especificações .

As abordagens nestas ferramentas utilizam-se de técnicas para extração de requisitos (entrevistas) [GAU89][LIN88][ZUC89], técnicas para decomposição de problemas (DFD,MER,SADT) [PRE92] e estratégias de reuso [BIG86][GIA91][MOO91][NEI86] para minimizar as informalidades típicas na comunicação entre usuário e analista.

Estes sistemas, enquanto eliminam o trabalho de anotações das informações do usuário, não eliminam a ambiguidade na comunicação do usuário com o analista. A inexistência de uma teoria formal para atacar este problema tem levado muitos ambientes de software a adotar a metodologia de prototipação [JOR89][LUQ88][LUQ89][TAN89] como uma alternativa pois, através desta técnica, o usuário e o analista podem, juntos, desenvolver e validar a especificação do sistema.

O modelo destes ambientes de software interativos tem as seguintes características [PRE92]:

- possibilitar a um analista criar interativamente uma especificação baseada em uma linguagem;
- invocar ferramentas automatizadas que traduzem especificações baseadas em linguagem para sistemas executáveis;
- permitir que o usuário utilize protótipos para refinar requisitos anotados.

O modelo de geradores de aplicação, para algumas classes de aplicações, baseia-se neste processo não convencional de desenvolvimento de software, onde é possível obter-se um sistema executável (aplicativo) gerado a partir de uma especificação refinada ao longo do processo de interação.

A abordagem de geradores de aplicação é bastante conhecida na literatura [CLE88][MEI91]. Associadas a estes geradores, estão as linguagens que são usadas para exprimir as aplicações e essas linguagens normalmente são conhecidas como linguagens de quarta geração - 4GL [MAR85B]. O processo de tradução dessas linguagens pode ser diretamente para código executável ou para uma linguagem alvo para a qual já existe um compilador disponível ("Source to source translation") [AHO86].

Nos trabalhos de Prieto Diaz e Júlio Leite [DIA90][LEI90] sobre Linguagens de Domínio e Linguagens de Aplicação existem propostas de como sintetizar essas linguagens.

A especificação de uma aplicação nessas linguagens necessita de uma tradução do nível informal (nível do usuário) para um nível formal (nível da linguagem da aplicação). Assistentes de Especificação são ferramentas de apoio a esta atividade.

Alguns Assistentes de Especificação operam sob a perspectiva de um domínio de aplicações, permitindo o desenvolvimento de família de sistemas com uma infraestrutura comum.

Um processo, denominado Análise de Domínio [ARA89,DIA87,FRE87], é usado para capturar as características comuns e as variações dentro de uma família de sistemas de um domínio para definir essa infraestrutura comum.

Segundo Prieto Diaz [DIA87], a análise de domínio visa encontrar técnicas para capturar, estruturar e armazenar informações de um domínio de aplicações de modo que se possa reusá-las na especificação de novas aplicações do domínio.

Assistentes de especificação que se utilizam destes conceitos e operam como uma infraestrutura de reuso são usados para desenvolver várias aplicações do domínio. Estes assistentes permitem que a especificação de uma nova aplicação do domínio possa usar as informações já armazenadas no desenvolvimento de aplicações anteriores [DIA91,MOO91]. Sob este aspecto, assistentes de especificação tornam-se ferramentas de apoio à atividade de Análise de Domínio.

2.3) Exemplos de Assistentes de Especificação

Nesta secção estão descritos alguns assistentes citados na literatura e, posteriormente, estes serão comparados para auxiliar na especificação do Assistente de Especificação do FMS - AEsp.

2.3.1) KBRET

a) Descrição:

O KBRET (Knowledge-Based Requirements Elicitation Tool) é uma ferramenta baseada em conhecimento que visa gerar especificações do sistema alvo a partir da modelagem do domínio. A ferramenta KBRET tem sido desenvolvida como parte de um protótipo de um ambiente de Engenharia de Software, desenvolvido pela George Mason University, para demonstrar os conceitos de modelagem de domínio e reuso. Este ambiente, independente de um domínio de aplicações, é usado para suportar modelos de domínios e gerar a especificação do sistema alvo a partir destes modelos [GOM92A][GOM92B].

b) Modelo de Ciclo de Vida:

O modelo de ciclo de vida proposto para este ambiente é o ciclo de vida baseado em reuso, denominado ciclo de vida evolucionário e orientado a domínio. Este ciclo de vida é altamente interativo e opera na perspectiva de um domínio de aplicações permitindo o desenvolvimento de família de sistemas.

c) Metodologia:

O processo de gerar especificações do sistema alvo utilizado no KBRET consiste em coletar os requisitos em termos das características do domínio, recuperar os componentes correspondentes do modelo do domínio para suportar aquelas características e garantir a consistência entre elas.

d) Arquitetura Geral:

A arquitetura do KBRET possui duas fontes de conhecimento:

Conhecimento Independente do Domínio: fornece conhecimento de controle para as várias funções suportadas pelo KBRET. Estas funções incluem ferramentas tais como:

- "Dialog Manager": é responsável por conduzir o diálogo com o engenheiro do sistema alvo e elicitar os requisitos para o sistema alvo;
- "Domain Browser": fornece regras para consultar as fontes de conhecimento do domínio para o engenheiro de sistema poder modelar o sistema alvo;
- "Feature and Object Selection/Deletion": mantêm a trilha de seleção e remoção das características para o sistema alvo e os tipos de objetos correspondentes;
- "Dependency Checker": é responsável pela consistência das dependências intra-características e inter características dos objetos;
- "Target System Generator": responsável por montar o sistema alvo depois que as suas características estão completas;

Conhecimento Dependente do Domínio: representa as visões múltiplas de um modelo de domínio da aplicação. Ferramentas que fazem parte desta fonte são:

- "Features and Object Types Knowledge Source": contêm uma lista de todos os tipos de objetos e suas características incorporados no modelo do domínio;
- "Inter-Feature and Feature-Object Dependencies Knowledge Source": captura os vários relacionamentos e dependências entre as características e entre os objetos.
- "Multiple Views": contém visões diferentes do modelo do domínio tais como hierarquia de agregação, generalização e especialização. Este conhecimento é derivado pelo "Domain Dependent Knowledge Base Extractor Tool", que estrutura o conhecimento como fatos em CLIPS, do Repositório de Objetos (onde ficam armazenados os tipos de objetos do domínio, suas características e seus relacionamentos para poder modelar um domínio).

e) Plataforma de suporte:

O KBRET foi implementado em CLIPS (C Language Integrated Production System) - Shell para desenvolvimento de sistemas especialistas da NASA. Outro software utilizado no ambiente onde está inserido o KBRET é o STP (IDE'S Software Through Pictures CASE) para representar as visões múltiplas do modelo do domínio. O STP armazena os dados em uma base de dados relacional denominada TROLL.

f) Metodologias de Análise de Domínio:

O KBRET pode ser caracterizado como uma ferramenta de apoio a atividade de análise de domínio, pois permite gerar especificações de aplicações a partir de modelos de domínios. O ambiente, no qual o KBRET está inserido, é independente de um domínio de aplicações, isto é, ele não é voltado para um domínio específico mas é utilizado na perspectiva de

domínios de aplicações. A especificação e a implementação da infraestrutura deste ambiente é construído de modo a permitir modelagem de domínios de aplicações e reuso das informações destes domínios.

2.3.2) KBRA

a) Descrição:

O KBRA (Knowledge-Based Requirements Assistant) é uma ferramenta para auxiliar a etapa de análise de requisitos do ambiente KBSA (Knowledge-Based Software Assistant). O KBSA é proposta de um ambiente que tem como objetivo formalizar todo o processo de produção de software e fornecer suporte, baseado em conhecimento, para todos aspectos do ciclo de vida da produção de software [GRE86,CZU88].

b) Modelo de Ciclo de Vida:

O modelo de ciclo de vida do KBSA é baseado no paradigma de desenvolvimento incremental de software. Divide o ciclo de vida em três etapas principais: Desenvolvimento, Validação da Especificação e Gerenciamento de Projeto.

Segundo o modelo de ciclo de vida proposto neste ambiente, a atividade de evolução torna-se a atividade central no processo de desenvolvimento de software e adequa-se ao modelo de ciclo de vida evolucionário.

c) Metodologia:

A base metodológica do KBSA é capturar e formalizar todo o processo de desenvolvimento de software (a identificação de requisitos, o projeto da especificação, a implementação daquela especificação, e sua manutenção). O KBRA é uma ferramenta que auxilia a etapa de captura de requisitos cobrindo três fases principais: aquisição, análise e comunicação. Para tal, o KBRA apresenta as seguintes propriedades: (a) uso do computador desde o ínicio do projeto; (b) capacidade de fornecer versões múltiplas do sistema que está sendo especificado; (c) suporte para captura e auxílio em decisões; (d) oportunidades para manusear reuso na etapa de requisitos; (e) capacidade de consistir e completar alguns requisitos automaticamente.

d) Arquitetura Geral:

O protótipo do KBRA auxilia a etapa de análise e aquisição de requisitos desde a fase inicial até a preparação do documento final. Para suportar esta funcionalidade, o KBRA possui uma biblioteca de componentes reusáveis e formas de apresentação das informações. As formas de apresentação suportadas pelo KBRA são:

- "Notepad" inteligente: um editor de texto estruturado para o usuário rascunhar as idéias iniciais do sistema.
- Diagrama de contexto: editor de diagramas utilizado para descrever a interface entre o sistema e o hardware, software e operadores do ambiente externo.
- Diagrama de Estados: editor gráfico utilizado para descrever os estados do sistema e eventos que provocam uma mudança no estado.

- Planilha de cálculos: é um objeto que armazena a descrição da evolução do sistema mostrado na forma de planilha.
- Documento de Requisitos: KBRA gera automaticamente um documento de requisitos para o sistema, na mesma maneira apresentada no "notepad".

A partir das informações obtidas por estas formas de apresentação, o KBRA propaga decisões para um repositório central de requisitos e checa a consistência dos resultados.

e) Plataforma de suporte:

O KBRA utiliza técnicas de representação de conhecimento associadas com a etapa de análise e aquisição de requisitos: apresentação, texto estruturado e descrição do sistema envolvido. A plataforma de suporte do KBRA também é baseada em técnicas de inteligência artificial, tais como: herança de propriedades de tipos de objetos genéricos, classificação automática baseada nos discriminadores que indicam como especializar instâncias, e propagação de restrições para processamento de ramificações de decisões de requisitos e para suporte à mudanças.

f) Metodologias de Análise de Domínio:

O KBRA é uma ferramenta de software independente de um domínio de aplicações. O KBSA, do qual o KBRA faz parte, é um assistente especialista nas metodologias envolvidas nas etapas do ciclo de vida do software.

2.3.3) ASPIS

a) Descrição:

O ASPIS (A Knowledge Based CASE Environment) é um ambiente que visa melhorar a qualidade e produtividade das primeiras fases do ciclo de vida do software (Análise de Requisitos e Projeto) combinando técnicas de Inteligência Artificial e Engenharia de Software [ASL91][PUN88].

b) Modelo de Ciclo de Vida:

Modelo de ciclo de vida evolucionário é adotado pelo ASPIS para fazer a ligação entre as fases de Análise e Projeto. O ciclo de vida do ASPIS suporta a interação entre o projetista e o analista com um processo chamado síntese. Neste processo a informação é coletada da fase de análise e colocada de forma disponível na fase de projeto, permitindo a comunicação entre analista e projetista. É um processo contínuo onde as inconsistências são checadas e então modificadas para atender as necessidades dos usuários.

c) Metodologia:

O ASPIS tem como objetivo melhorar a qualidade do software suavizando a transição entre as necessidades dos usuários, análise e projeto através de um conjunto de ferramentas integradas. Na fase de Análise de Requisitos é utilizada SAL (Structural Analysis Language) para representar funções e esquemas entidade-relacionamento para representar dados. O RSL (Reasoning Support Logic) é utilizado para otimizar os diagramas SA e para checar a consistência e a corretude das especificações. O RSL permite especificar propriedades de um sistema com um conjunto de axiomas em uma linguagem baseada em lógica. Usando o RSL, o analista pode escrever axiomas em termos de funções, predicados e posições em um diagrama SA.

Estes axiomas são transformados em código PROLOG para serem executados. A fase de Projeto é dividida em duas etapas:

- projeto do sistema: descreve o sistema nos termos globais de hardware e software;
- projeto do software: descreve as funções do software em termos de processos e transições.

d) Arquitetura Geral:

O ASPIS possui um conjunto de ferramentas baseadas em conhecimento, denominadas assistentes, e a definição de um formalismo baseado em lógica para especificações. O ASPIS possui quatro assistentes: assistente de análise, projeto, protótipo e reuso.

- Assistente de Análise e Assistente de Projeto são caracterizados como assistentes baseados em conhecimento, pois incorporam o conhecimento sobre o método e sobre o domínio da aplicação. Eles são utilizados por desenvolvedores de uma aplicação particular aderindo a uma metodologia particular. O assistente de análise possui um conjunto de ferramentas que deixa o usuário e o sistema gerenciar documentos de análise.
- Assistente de Protótipo e Assistente de Reuso são caracterizados como assistentes de suporte, pois o primeiro ajuda na verificação das propriedades do sistema e o segundo ajuda os desenvolvedores a reusar especificações e projetos.

e) Plataforma de suporte:

Estes assistentes do ASPIS foram implementados utilizando a linguagem PROLOG.

f) Metodologias de Análise de Domínio

O ASPIS é um ambiente independente de um domínio de aplicações, mas ele é utilizado na perspectiva de domínios de aplicações permitindo reusar informações de um domínio específico na especificação de novas aplicações deste domínio. O ASPIS incorpora conhecimento sobre o método e sobre os domínios de aplicações particulares.

2.3.4) Requirements Apprentice - RA

a) Descrição:

O Requirements Apprentice (RA) é um assistente automatizado para aquisição de requisitos. Ele ajuda o analista na criação e modificação dos requisitos de software. Focaliza a transição entre especificações informais e formais. O RA está sendo desenvolvido no contexto do projeto Programmer's Apprentice, cujo objetivo é a criação de um assistente inteligente para todos os aspectos do desenvolvimento de software [WAT91].

b) Modelo de Ciclo de Vida:

O modelo de ciclo de vida adotado no RA é o modelo evolucionário que é aderente a abordagem de desenvolvimento incremental utilizada neste tipo de assistente. O ciclo de vida utilizado é divido em três partes principais:

Análise de Requisitos Projeto Implementação
|----------|----------|

A ferramenta RA é usada na etapa de Análise de Requisitos.

c) Metodologia:

O Requirements Apprentice atua na fase de aquisição de requisitos. Esta fase pode ser divida em três etapas:

- elicitação: que tem como produto final requisitos informais;
- formalização: etapa que constrói a especificação formal a partir dos requisitos informais;
- validação: etapa para validar a especificação formal.

O RA atua mais especificamente na etapa de formalização que faz a ponte entre a especificação informal e a especificação formal. Ele tenta resolver seis informalidades típicas da comunicação entre o usuário e o analista: abreviação, ambiguidade, ordenação, contradição, incompleteza e inconfiabilidade.

Para atingir seus objetivos o RA tem três tipos de saída para interagir com o analista:

- saída interativa: notifica o analista de conclusões e inconsistências detectadas enquanto a informação de requisitos está sendo fornecida;
- uma base de conhecimento de requisitos;
- documentos contendo o resumo da base de conhecimento de requisitos.

d) Arquitetura Geral:

A arquitetura do RA é composta por três módulos:

- CAKE: é um sistema para raciocínio e representação do conhecimento, que suporta as habilidades de raciocínio do RA;
- EXECUTIVE: este sistema manuseia a interação com o analista e fornece controle em alto nível do raciocínio executado pelo CAKE;
- Biblioteca de "Clichés": é um repositório declarativo de informações relevantes para requisitos gerais e requisitos de domínios particulares. Esta biblioteca que possibilita inferir grandes quantidades de informação a partir dos enunciados do analista.

A interface do RA é dividida em três janelas:

- janela principal: é usada para mostrar informações sobre os requisitos;

- janela de diálogo: é usada para digitar os comandos para o RA e para mostrar as respostas do RA imediatamente;

- janela de saídas pendentes: é usada para mostrar uma lista de saídas pendentes que necessitam ser resolvidas.

e) Plataforma de suporte:

O Requirements Apprentice - RA foi implementado em Commom LISP, portanto o diálogo entre o analista e o RA é feito através de expressões descritas em LISP.

f) Metodologias de Análise de Domínio:

O RA pode ser utilizado como uma ferramenta de suporte a atividade de Análise de Domínio. Este não é voltado para um domínio específico mas ele é utilizado na perspectiva de domínios de aplicações. O RA permite reusar informações de domínios particulares através de uma estrutura de clichés.

2.3.5) Assistente Especialista para Especificação de Requisitos - AR

a) Descrição:

O AR, sigla utilizada para fins de simplificação, é um assistente especialista para especificação de requisitos. Este assistente possibilita a especificação dos requisitos através de respostas dadas pelo usuário, em linguagem natural, a um questionário proposto pelo assistente. A partir destas respostas um modelo semi-formal, baseado no conhecimento sobre o modelo entidade-relacionamento (MER) é gerado [ARI92].

b) Modelo de Ciclo de Vida:

O modelo de ciclo de vida adotado neste trabalho é baseado no paradigma de programação automática proposto por Balzer [BAL86]. Neste conceito é possível perceber que a partir dos conceitos informais obtém-se os requisitos do sistema até conseguir uma especificação formal, que funciona como um protótipo sobre o qual é feita a validação.

c) Metodologia:

Este assistente se caracteriza na classe de assistentes especialistas em metodologias, cuja filosofia consiste em observar como os peritos em metodologias atuam na aplicação das mesmas. As características básicas de um assistente especialista em metodologias:

- são ferramentas baseadas em conhecimento;

- servem para comparar ou mesmo compelir a utilização da metodologia;

- proporcionam uma interface amigável com o usuário.

A metodologia é expressa através de regras que definem seus príncipios e regras que restrigem práticas contrárias ao enfoque dado.

Este assistente de requisitos utiliza o MER como método de especificação que é gerado a partir de um questionário que é aplicado ao usuário.

d) Arquitetura Geral:

A arquitetura deste assistente especialista em metodologias consta das seguintes partes:

- base de conhecimentos: onde é especificado o conhecimento sobre a metodologia (neste caso é o MER);
- base de dados: onde é armazenado o conhecimento obtido através do diálogo com o usuário, e de onde se obterá, posteriormente, o modelo textual;
- máquina de inferência: é a parte do assistente que consulta a base de conhecimentos para validar a especificação segundo o modelo entidade-relacionamento;
- interface em linguagem natural: é um ambiente no qual o usuário fornece os conceitos informais da aplicação, através de um diálogo em um sub-conjunto da linguagem natural, guiado pelo assistente.

e) Plataforma de suporte:

Este assistente foi implementado utilizando o ambiente PROLOG.

f) Metodologias de Análise de Domínio

O AR se enquadra na classe de assistentes especialistas em metodologias. É uma ferramenta independente de um domínio de aplicações e também não é utilizado na perspectiva de domínios de aplicações.

2.4) Quadro Comparativo entre Assistentes

Na seção anterior pôde-se observar que os assistentes são baseados no modelo de ciclo de vida evolucionário [GOM92A][GOM92B][LUQ88][LUQ89] e suportam o desenvolvimento de software de forma automática como descrito na Figura 2.1 [PRE92].

O modelo evolucionário prevê que a fase de desenvolvimento englobe a fase de evolução e manutenção, de modo que o desenvolvimento seja incremental e haja uma validação da especificação antes da geração do código. Desta forma, o modelo evolucionário se encaixa no paradigma de desenvolvimento de software automatizado.

Neste trabalho dividiu-se o ciclo de vida no qual se baseiam estes Assistentes de Especificação em 3 etapas principais: análise de requisitos/especificação, validação da especificação e desenvolvimento.

O mapeamento destas etapas do ciclo de vida no paradigma de desenvolvimento automatizado de software pode ser representado como na Figura 2.2.

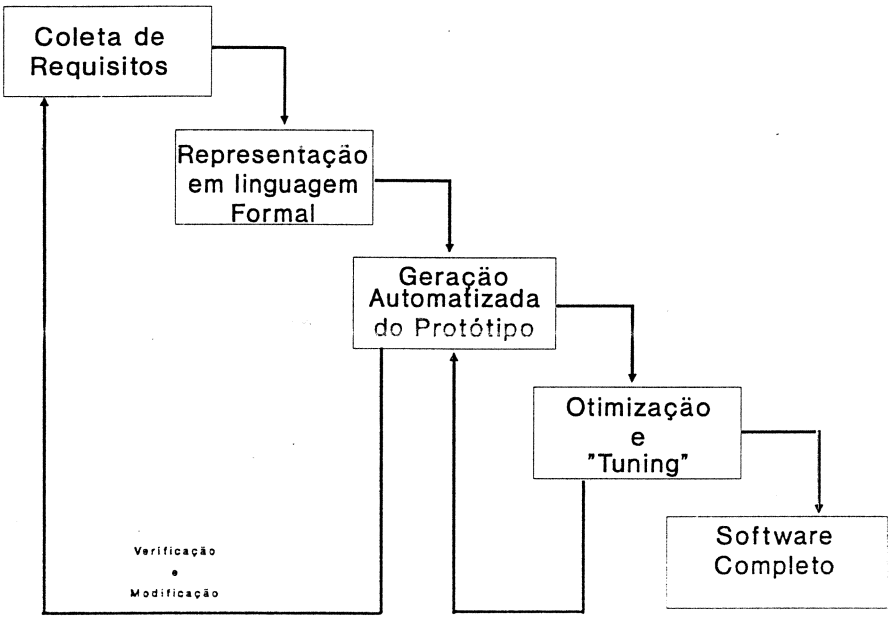


Figura 2.1: Paradigma de desenvolvimento de software automatizado

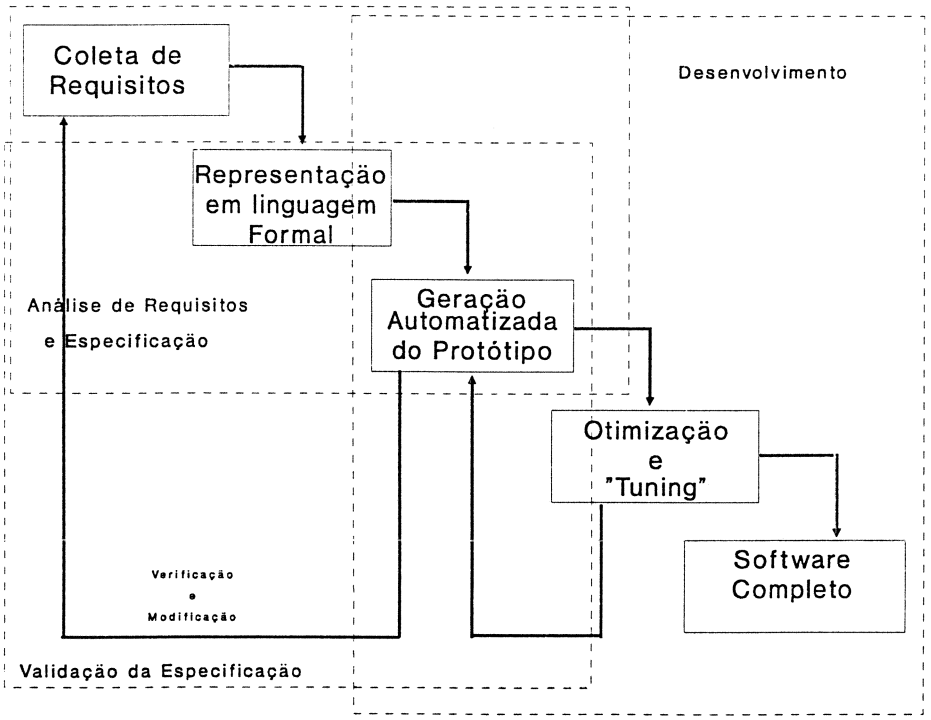


Figura 2.2: Mapeamento das etapas do ciclo de vida no paradigma de desenvolvimento de software automatizado

A análise de requisitos e especificação é a etapa responsável por capturar as informações do usuário, decompor e refinar o problema até se obter uma especificação.

A etapa de validação da especificação é responsável por transformar a especificação em um protótipo executável para que o usuário possa validá-la.

A etapa de desenvolvimento engloba a implementação e manutenção. A implementação sendo realizada como resultado de várias transformações das especificações e a manutenção sendo executada pela alteração da especificação.

Neste capítulo é mostrado que Assistentes de Especificação auxiliam na primeira e segunda etapa diretamente e através da especificação auxiliam a fase de desenvolvimento. Assistentes de Especificação auxiliam a fase de desenvolvimento invocando ferramentas automatizadas que traduzem as especificações baseadas em linguagem para sistema executável.

Além das atividades descritas acima, referentes às etapas do ciclo de vida, existem outras atividades, tais como Desempenho e Gerenciamento, que não serão abordadas neste trabalho, embora alguns assistentes as suportem.

Neste trabalho, de modo a tornar mais objetiva a comparação entre assistentes, selecionaram-se algumas características relativas às três etapas do ciclo de vida, como descrito abaixo:

a) Análise de Requisitos e Especificação

- Forma de Aquisição das Especificações: meio pelo qual o assistente se comunica com seu usuário alvo;
- Técnica de Aquisição de Conhecimento: técnicas utilizadas para extrair as informações do usuário;
- Linguagem de Manipulação dos Requisitos: são linguagens específicas suportadas pelo assistente para especificar os requisitos;
- Metodologia de Refinamento e Decomposição do Problema: metodologias para refinar o problema até obter a especificação;
- Estratégia de Reuso dos Requisitos: um sistema de reuso que permita reutilizar especificações de aplicações anteriores;
- Domínio de Aplicações: domínio específico suportado pelo assistente, se este for orientado a domínio;
- Classes de Usuário: o usuário alvo do assistente.

b) Validação da Especificação

- Modelo de Representação da Especificação: modo pelo qual são representadas as especificações dos requisitos, formal ou informalmente;
- Linguagem de Especificação: forma pela qual a especificação é representada - em uma linguagem que pode ser executável ou interpretável;

- Consistência da Especificação: mecanismos e processos que permitam garantir a consistência interna da especificação;
- Teste da Especificação: mecanismos e processos que permitam que a especificação possa ser testada para ser validada pelo usuário antes de gerar o código da aplicação;
- Parafraseador de Especificação: capacidade para parafrasear uma especificação formal em linguagem natural com vocabulário restrito;
- Prototipação Rápida: capacidade para automaticamente compilar uma especificação para um nível que permita gerar um protótipo;
- "Tracer" de Requisitos: mecanismos e processos que permitam atendimento aos requisitos do usuário em qualquer ponto do ciclo de vida;
- Descritor de comportamento: mecanismos com capacidade de explicar em linguagem natural o comportamento de uma especificação.

c) Desenvolvimento

- Modelo de Implementação: o processo subjacente de desenvolvimento dos sistemas - incremental ou convencional;
- Linguagem de Amplo Espectro: existência de uma linguagem capaz de representar o projeto de um sistema em todos os estágios do ciclo de vida, da especificação formal até a implementação;
- Linguagem de Transformações: se há disponibilidade de uma linguagem capaz de descrever as transformações de abstratas para concretas;
- Linguagem de Propriedades: disponibilidade de uma linguagem capaz de descrever as propriedades de objetos dos programas (variáveis, módulos , relações etc.);
- Documentação Interativa: se há disponibilidade de um sistema capaz de documentar passos selecionados pelo usuário e as decisões que o levaram a tal;
- Prova Automática de Propriedades: mecanismo de inferência para automaticamente provar propriedades de partes do sistema em desenvolvimento;
- Desenvolvimento Automatizado: disponibilidade de um sistema capaz de gerar a aplicação automaticamente a partir de uma especificação;
- Reuso Automatizado: capacidade de adaptar um desenvolvimento prévio para uma especificação alterada.

Abaixo, comparam-se os assistentes descritos no ítem 2.3 segundo as características específicas de cada etapa do ciclo de vida descritas acima. O AEsp será inserido nos quadros comparativos para se ter uma visão de como se encaixa nesta classe de sistemas. O AEsp será descrito detalhadamente no decorrer desta dissertação.

a) Análise de Requisitos e Especificação

Assistentes/ Características[1]	KBRET	KBRA	ASPIS	RA	AR	AEsp
Forma de Aquisição das Especificações	Diálogo	Diálogo	Diálogo	Diálogo	Diálogo	Diálogo
Técnicas de Aquisição de Conhecimento	Entrevista	Entrevista	Entrevista	Entrevista	Entrevista	Entrevista
Linguagem de manipulação de requisitos	Gráfica	Textual/ Gráfica	Textual	Textual/ Gráfica	Textual/ Gráfica	Textual/ Gráfica
Metodologia de refinamento do problema	Refinamento através de 5 visões múltiplas	Prevê uma metodologia	SAL	Estrutura de Cliches	MER	HOS
Estratégia de reuso	Reuso dos objetos do repositório de objetos	Biblioteca de componentes reusáveis	Reuso de análise/ projeto	Reuso de Cliches	*	Reuso das especificações
Orientado a domínio	Perspectivas do domínio	Independente do domínio	Perspectivas do domínio	Perspectivas do domínio	Independente do domínio	Domínio de Administração Rural
Classes de Usuário	Engenheiro de Requisitos do Sistema	Usuário Final	Desenvolvedor da Aplicação	Analista de Sistemas	Engenheiro de Sistemas	Engenheiro de Especificação

Tabela 2.1: Características dos Assistentes na etapa de Análise de Requisitos e Especificação

[1] As colunas preenchidas com (*) asteriscos indicam que esses assistentes não possuem as características assinaladas ou que estas características não puderam ser identificadas nos artigos referenciados.

b) Validação da Especificação

Assistentes/ Características[1]	KBRET	KBRA	ASPIS	RA	AR	AEsp
Modelo de Representação	Formal	Formal	Formal	Formal	Formal	Formal
Linguagem de Especificação	Interpretável	Executável	Executável	Executável	Interpretável	Interpretável
Consistência da Especificação	Disponível	Disponível	Disponível	Disponível	Disponível	Disponível
Teste da Especificação	*	Protótipo Executável	Protótipo Executável	Protótipo Executável	Validação do MER	Protótipo incompleto
Parafraseador de Especificação	*	Disponível	*	Disponível	*	*
Prototipação Rápida	*	Disponível	Disponível	Disponível	*	Disponível
"Tracer" de Requisitos	Disponível	Disponível	*	Disponível	*	*
Descritor de Comportamento	*	Disponível	*	Disponível	*	*

Tabela 2.2: Características dos Assistentes na etapa de Validação da Especificação

[1] As colunas preenchidas com (*) asteriscos indicam que esses assistentes não possuem as características assinaladas ou que estas características não puderam ser identificadas nos artigos referenciados.

c) Desenvolvimento

Assistentes/ Características[1]	KBRET	KBRA	ASPIS	RA	AR	AEsp
Modelo de Implementação	Incremental	Incremental	Incremental	Incremental	Incremental	Incremental
Linguagem de Amplo Espectro	*	Disponível	*	Disponível	*	Disponível
Linguagem de Transformações	*	Disponível	*	Disponível	*	Disponível
Linguagem de Propriedades	*	Disponível	*	*	*	Disponível
Documentação Interativa	Disponível	Disponível	Disponível	Disponível	Disponível	Disponível
Prova Automática de Propriedades	*	Disponível	*	Disponível	*	*
Desenvolvimento Automatizado	*	Disponível	*	Disponível	*	Disponível
Reuso Automatizado	Disponível	Disponível	Disponível	Disponível	*	Disponível

Tabela 2.3: Características dos Assistentes na etapa de Desenvolvimento

[1] As colunas preenchidas com (*) asteriscos indicam que esses assistentes não possuem as características assinaladas ou que estas características não puderam ser identificadas nos artigos referenciados.

2.5) Análise Geral

Os trabalhos citados não foram escritos tendo em mente os critérios de comparação aqui estabelecidos. A comparação aqui sugerida visa auxiliar no dimensionamento e especificação do AEsp [MAS94A].

É notável que os modelos de ciclo de vida para ambientes desta classe sejam muito parecidos. Isto é evidenciado pelas Tabelas 2.1 a 2.3 quando se examinam atributos ligados a esse modelo - forma de aquisição do conhecimento, orientação a domínio, modelo de representação, testes e linguagens usadas. Esta semelhança se deve, ao que parece, mais ao tipo de ferramentas e métodos usados (Análise de Domínio e Engenharia de Conhecimento) do que as propriedades específicas do objeto alvo do sistema.

A homogeneidade da arquitetura dos assistentes é evidenciada pelas ferramentas que os compõem: ferramentas para elicitação de requisitos; especificação formal; decomposição de problemas; prototipação rápida e geração automática de código.

As plataformas de suporte utilizadas para implementação dos assistentes também são similares. Utilizam-se de técnicas de análise de sistemas (orientação a objetos), linguagens de quarta geração, processos de inteligência artificial e técnicas de engenharia de software.

Outro ponto de observação é que alguns destes assistentes operam na perspectiva de domínios de aplicações, permitindo o desenvolvimento de família de sistemas com uma infraestrutura comum. Estes assistentes tornam-se ferramentas de apoio a atividade de Análise de Domínio [DIA90].

Independente das metas dos assistentes acima citados, será possível verificar no decorrer desta dissertação, que a proposta do AEsp, no espectro de suas aplicações, corresponde às expectativas de um assistente.

No Capítulo 3 será descrito o escopo do AEsp e nos Capítulos 4 e 5 será descrita detalhadamente a base conceitual do AEsp.

CAPÍTULO 3

ASSISTENTE DE ESPECIFICAÇÃO DO FMS - AESP

3.1) Introdução

O AEsp está limitado, por um lado, pelo conjunto de sublinguagens de especificação do FMS e, por outro, por um modelo de entrevista e correspondente elenco de objetos - CMs na terminologia deste trabalho - que conduzem essa entrevista. O objetivo deste capítulo é descrever o escopo do AEsp no contexto do ambiente FMS.

3.2) AEsp no contexto do FMS

A abordagem utilizada pelo FMS [FER94A][FER94B] prevê que, durante a fase de especificação do processo de desenvolvimento de software, as informações de uma aplicação sejam capturadas, decompostas e armazenadas para posterior reutilização. As informações devem ser decompostas até um nível de representação para o qual estará disponível um gerador de código fonte, dispensando a fase de implementação.

Segundo Neighbors [NEI89], cada problema tem sua própria linguagem que é denominada Linguagem do Domínio (LD) e a identificação desta linguagem é resultado de um processo chamado Análise de Domínio.

A análise de domínio, para Neighbors, produz os operandos e operadores de uma classe de aplicações e, com base nesta análise, uma linguagem é construída e usada para especificar outros problemas do domínio em questão [NEI89].

A análise de domínio, no escopo do Projeto FMS, foi realizada em duas etapas. A primeira etapa identificou o elenco de funções básicas e o modelo de fluxo de controle e de dados típicos das aplicações do domínio de administração rural. A segunda etapa agregou esse conjunto de funções na forma de uma Linguagem de Composição do FMS (LC-FMS) e modelou as aplicações na forma de programas da classe reativa [FER94A].

No modelo do FMS, os requisitos do usuário são transformados em uma especificação descrita em LC-FMS e a aplicação, gerada automaticamente a partir da especificação em LC-FMS, comporta-se como uma máquina de estados estendida [WAS85]. Para suportar esta abordagem, a metodologia subjacente ao FMS é baseada em um modelo de níveis.

3.2.1) Modelo de Níveis

O modelo de níveis do FMS propõe uma hierarquização entre os níveis do processo de transformação dos requisitos do usuário, basicamente devido aos métodos disponíveis para essa tarefa [MAS94B].

O modelo utilizado é o modelo do Lehman [LEH84], no qual o nível não formalizado é descrito como nível conceitual e o nível onde já há uma descrição formal da aplicação é denominado, neste trabalho, nível operacional. Na Figura 3.1 é mostrado o modelo do FMS sob o modelo do Lehman.

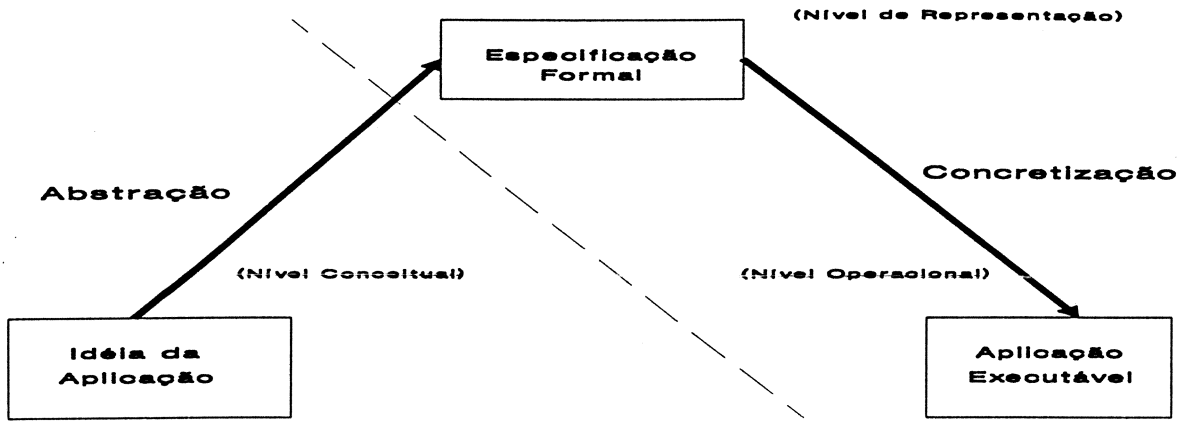


Figura 3.1: Modelo do FMS sob o modelo do Lehman

3.2.1.1) Nível Conceitual

O processo de captura dos requisitos do usuário (informal) e sua tradução para uma representação formal é um problema em aberto [BAL86], pois envolve todos os problemas citados no Capítulo 2 e em [MAS94A].

Este trabalho descreve um processo de apoio a essa captura e transformação, denominado AEsp, para o domínio de aplicações citado no Capítulo 1, de forma semelhante à descrita em várias publicações [GOM92A] [KAR88] [PRE91] [PUN88] [SPC88] [WAT91].

Para tal, o AEsp suporta o processo de entrevista e documentação dos requisitos dos usuários, sua decomposição sistematizada sob uma metodologia formal e a coleta do jargão através da instanciação de componentes pré-existentes. O resultado de uma sessão sob o AEsp, quando bem sucedida, é a aplicação do usuário e sua linguagem da aplicação (LA) subjacente.

3.2.1.2) Nível Operacional

O processo de transformação dos requisitos do usuário, já descritos formalmente para um programa executável, foi reduzido ao processo de tradução entre os níveis da aplicação, representação e implementação [MAS94B]. A Figura 3.2 abaixo mostra essa hierarquização.

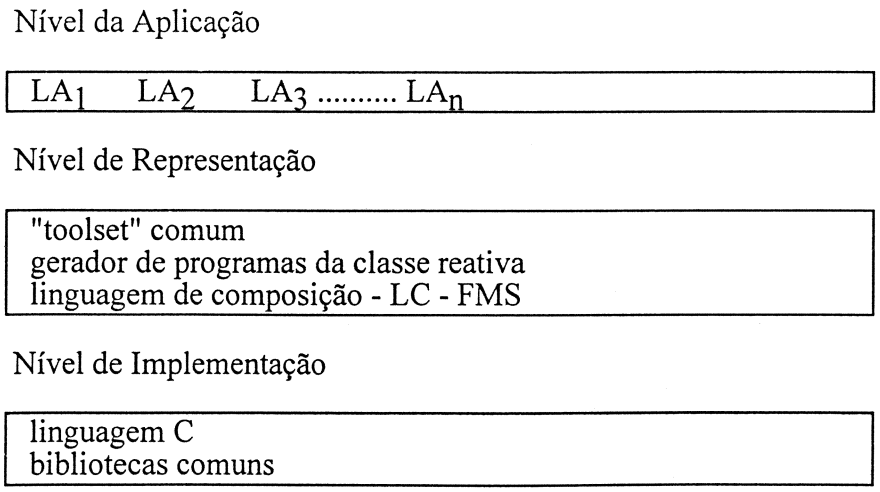


Figura 3.2: Hierarquização do nível operacional

A) Nível da Aplicação

Usando a terminologia de Leite [LEI90], uma aplicação do usuário, neste nível, é descrita em uma linguagem da aplicação (LA). Segundo Leite, as linguagens da aplicação (LAs) são linguagens que têm todas as características das linguagens do domínio (LD), só que tratam de uma ou duas instâncias daquele domínio. As LAs, no modelo do FMS, são expressas em LC-FMS. Esta metáfora suporta:

a) versões diferentes da mesma aplicação, o que representa o seu desenvolvimento incremental;

b) aplicações diferentes para problemas semelhantes, que é amparado pelo modelo de Linguagens de Domínio [NEI86].

B) Nível de Representação

O nível de representação é descrito pela sintaxe e semântica da LC-FMS, que é composta por várias sublinguagens: linguagem de formulários, linguagem de ações (operações relacionais), linguagem de eventos, linguagem de síntese de campos, linguagem de consistências, linguagem de helps e linguagem de relatórios. Através dessas linguagens, a aplicação incorpora os vários componentes do domínio:

a) componentes de alta especificidade que aparecem como operandos e operadores dessas linguagens;

b) componentes de alto índice de reutilização, que são "templates" ou "clichés" [CAL91,MEI91,WAT91], cuja expansão sintática materializa funções específicas da aplicação em desenvolvimento.

C) Nível de Implementação

O nível de implementação é gerado pela tradução automática da LC-FMS (nível de representação) para uma implementação em linguagem C através de um gerador de código fonte (GFMS), que usa as técnicas convencionais de compilação [AHO86] e ferramentas de geração de código [MAS93].

O processo de tradução entre os níveis de aplicação, representação e implementação é conhecido na literatura e pode ser modelado, na abordagem de análise de domínio, como um processo de tradução suportado por uma rede de domínios [NEI86].

O processo de tradução dos requisitos do usuário para uma representação formal, que é suportado pelo AEsp, está descrito nas próximas seções.

A Figura 3.3 mostra o escopo do AEsp através do mapeamento do FMS sob o modelo de Lehman [LEH84] que permite separar as etapas de abstração (coleta de especificação, incluindo reuso e prototipação) da etapa de concretização (geração automatizada de código).

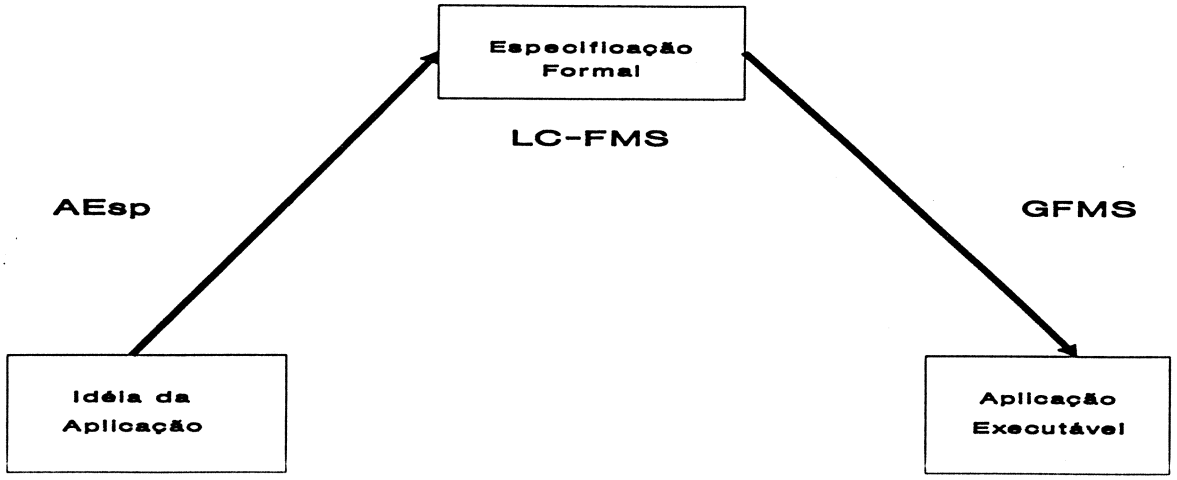


Figura 3.3: Escopo do AEsp

3.3) Características do AEsp

O enfoque utilizado pelo AEsp para síntese da aplicação descrita em LC-FMS é:

a) considerar o usuário como especialista que fornece a informação sobre a aplicação;

b) elicitar incrementalmente, através de uma entrevista, os conceitos da aplicação sob um processo formal de decomposição;

c) não tentar automatizar a etapa de especificação, isto é, evitar o processamento automático da linguagem natural e, sim, utilizar um Engenheiro de Especificação para conduzir a entrevista e reduzir o vocabulário envolvido;

d) oferecer ao usuário uma infraestrutura para entrevista similar à aplicação final;

e) catalogar as abstrações de dados e ações de forma a possibilitar seu reuso em outras aplicações e, portanto, potencializar a transformação de uma LA em LD.

Para suportar este processo o AEsp foi construído com as seguintes características:

a) estrutura de controle de entrevista com o usuário, permitindo capturar as especificações das aplicações;

b) estrutura de prototipação para auxiliar na entrevista e validar a especificação em qualquer momento;

c) esquema de catalogação de componentes (operandos e operadores) de modo a incrementar permanentemente o acervo do domínio. A transformação de componentes específicos em componentes instanciáveis é "off-line";

d) esquema de reuso das informações do domínio;

e) apresentação da decomposição da aplicação, sob várias formas, para facilitar "Backtracking".

O protótipo do AEsp suporta essas funcionalidades através de um conjunto de ferramentas integradas por uma base de conhecimento cuja conceituação está apresentada na Figura 3.4. Estas ferramentas são: Entrevistador, Prototipador, Catalogador de Componentes, Montador e Reutilizador.

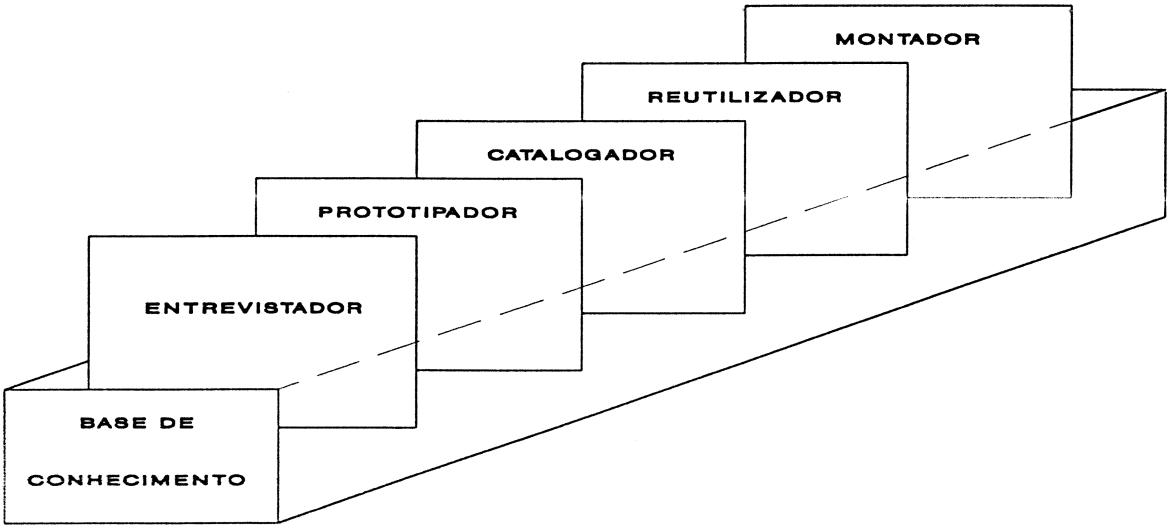


Figura 3.4: Ferramentas do AEsp

O Entrevistador é o modulo central do AEsp, pois o modelo de entrevista no qual se baseia é o cerne do AEsp. Todas as informações capturadas pelo entrevistador durante o processo de entrevista, tanto no nível da aplicação quanto no nível de representação, ficam armazenadas em uma base de conhecimento. Estas informações podem ser recuperadas pelas outras ferramentas do AEsp em qualquer momento da especificação da aplicação. A descrição detalhada dessas ferramentas está no Capítulo 5.

3.3.1) Engenheiro de Especificação

O **Engenheiro de Especificação** é um técnico treinado a operar o AEsp com o objetivo de conduzir a entrevista para extrair a informação do Especialista do Domínio (usuário). Suas atividades são:

a) auxiliar no mapeamento da linguagem natural para o vocabulário controlado do ambiente AEsp;

b) oferecer o protótipo da aplicação durante a entrevista;

c) apoiar a modelagem dos dados e decomposição das ações;

d) apoiar o "backtracking" quando necessário.

3.4) Considerações sobre o Capítulo

Neste capítulo foi descrito o AEsp no contexto do ambiente FMS, mostrando os modelos em que se baseia a sua construção e suas funções, que serão detalhadas no Capítulo 5. Antes de descrever o modelo de implementação do AEsp será descrita sua modelagem conceitual no próximo capítulo.

CAPÍTULO 4

MODELO CONCEITUAL DO AEsp

4.1) Introdução

O objetivo deste capítulo é descrever a abordagem do AEsp e alguns conceitos no qual está baseada sua implementação. É dada enfâse ao modelo do processo de entrevista do Entrevistador que é o cerne do AEsp.

4.2) Abordagem do AEsp

O AEsp baseia-se na premissa de que o processo para especificação de uma aplicação é direcionado por um roteiro de entrevista. O processo inicia-se com a descrição dos requisitos pelo *Especialista do Domínio* utilizando o seu próprio jargão. O diagrama da arquitetura geral do AEsp, representado na forma de actigrama em "SADT" [PRE92] na Figura 4.1, ilustra esta abordagem.

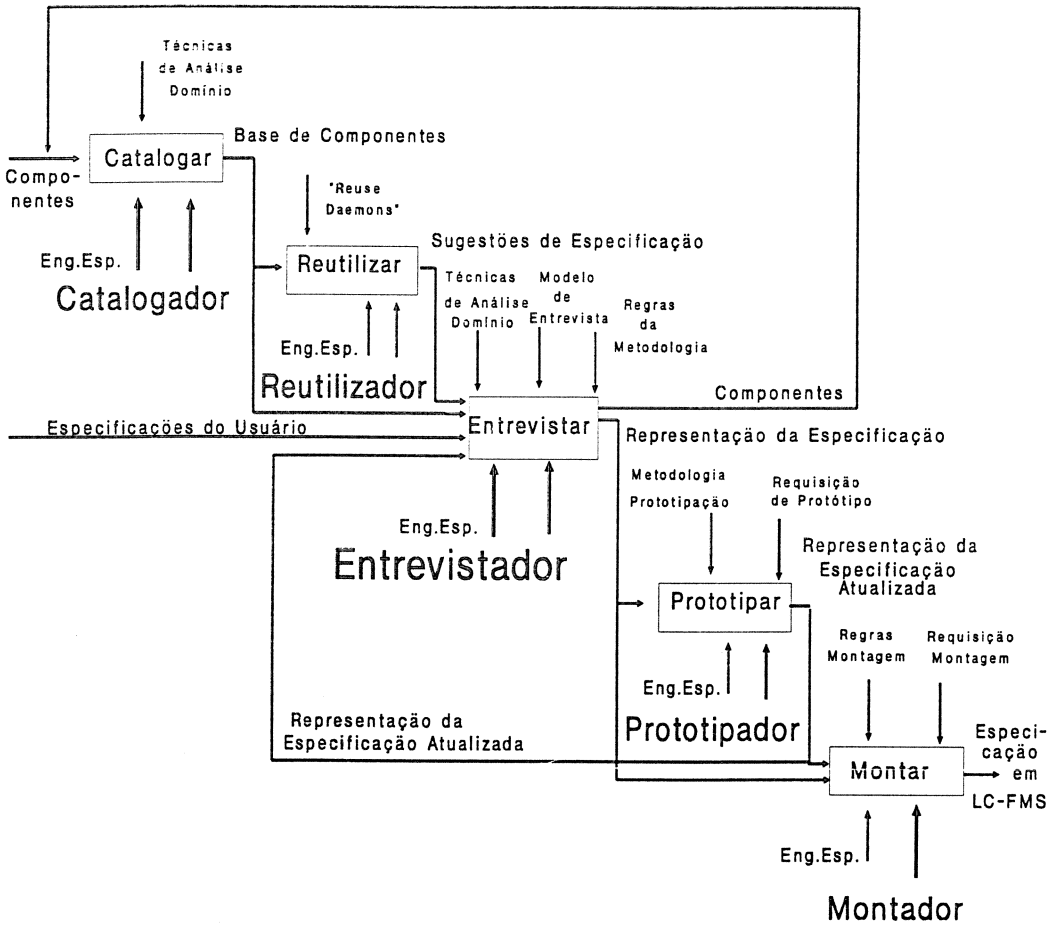


Figura 4.1: Arquitetura Geral do AEsp

O *Engenheiro de Especificação* apóia o mapeamento dos requisitos do Especialista do Domínio em uma linguagem formal suportada pelo Entrevistador.

O *Entrevistador* é uma ferramenta que, baseada em um modelo de entrevista, permite especificar as aplicações. A especificação das aplicações é realizada segundo uma metodologia de decomposição até chegar aos componentes da aplicação. Estes componentes, identificados durante a entrevista, são armazenados "off-line" em uma base de conhecimento através de uma ferramenta, denominada *Catalogador de Componentes* [CAL91].

O *Reutilizador* é uma ferramenta que permite reusar esses componentes, armazenados na base de conhecimento, na especificação de uma nova aplicação do domínio. O modelo de domínio do FMS é aderente a este método de reuso. Durante o processo de entrevista o Engenheiro de Especificação pode utilizar sugestões de especificação vindas do Reutilizador.

O Engenheiro de Especificação também pode oferecer o protótipo da aplicação, em qualquer momento da entrevista, enviando uma requisição ao prototipador.

O *Prototipador* é uma ferramenta para auxiliar na validação da especificação durante a entrevista. Esta ferramenta gera uma primeira versão incompleta do protótipo que mostra a interface e o comportamento da aplicação. O protótipo completo da aplicação será obtido após a entrevista, quando o Engenheiro de Especificação enviar uma requisição de montagem ao Montador.

O *Montador* gera a especificação da aplicação em LC-FMS a partir da varredura da árvore gerada pelo entrevistador e das bases de conhecimento correlatas. Essa especificação descrita em LC-FMS é automaticamente traduzida pelo GFMS.

Na Figura 4.1, pode-se notar a centralização do Entrevistador na arquitetura geral do AEsp. Esta ferramenta tem uma função gerencial no processo de transformação dos requisitos do usuário. O modelo de entrevista do Entrevistador é onde está centrada a base conceitual do AEsp.

4.3) Modelo do Processo de Entrevista do AEsp

O modelo do processo de entrevista do AEsp é baseado em perguntas direcionadas ao usuário que permite especificar o problema segundo uma metodologia de decomposição. A estrutura do entrevistador depende da metodologia de decomposição e, neste trabalho, está sendo utilizada a metodologia "High-Order" Software - HOS [HAM76A, HAM76B, MAR85A].

4.3.1) Metodologia de Decomposição - HOS

A metodologia HOS permite decompor as funções e os dados de uma aplicação em uma especificação na forma de uma estrutura de árvore, denominada "Control Map" - CM. Os nós da árvore representam níveis hierárquicos da decomposição. A Figura 4.2 mostra o exemplo do "Control Map" de uma Função "F" que tem um ou mais objetos (dados) de entrada (x) e saída (y).

A função F corresponde ao nó raiz que é decomposta em subfunções até chegar aos nós folhas. Os nós folhas, também denominados "Nós Primitivos", podem ser divididos em 4 tipos: P: Operação primitiva, OP: Operação já definida, R: Operação recursiva, XO: Operação externa.

Os objetos de entrada (x) e saída (y) das funções são, frequentemente, item de dados, registros, arquivos ou base de dados.

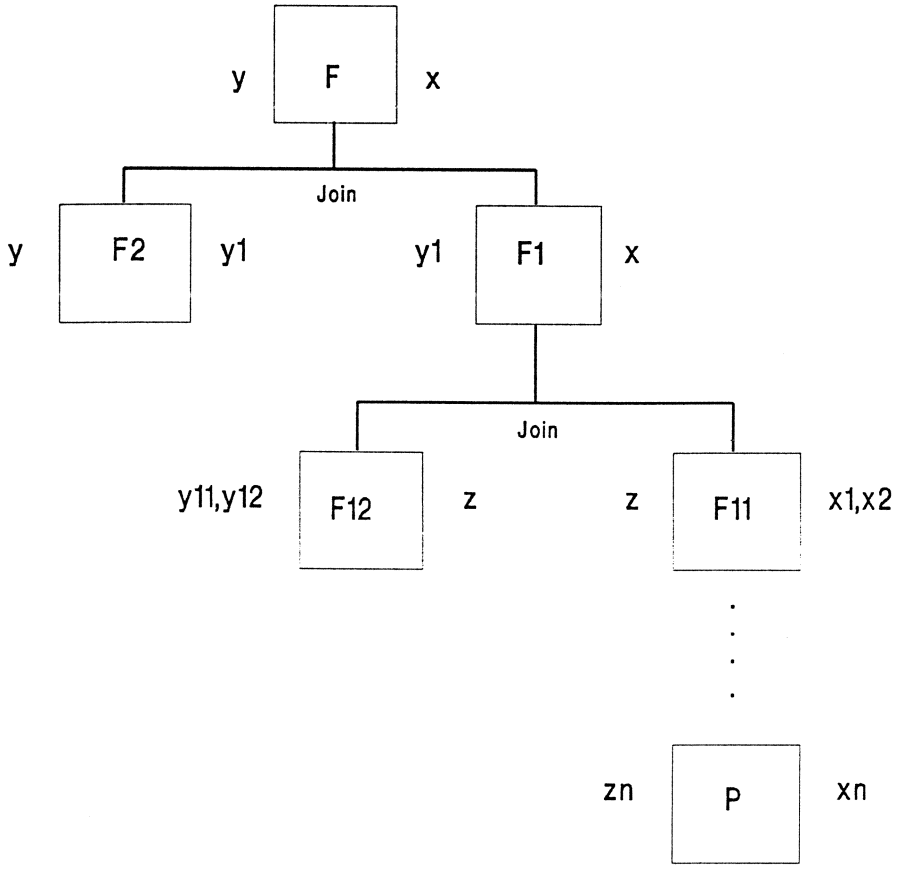


Figura 4.2: Árvore de Decomposição HOS - "Control Map"

A decomposição em HOS de uma função em subfunções é matematicamente precisa, conforme Hamilton e Zeldin [HAM76A,HAM76B], devido a suas estruturas de controle que, baseadas em axiomas matemáticos, permitem consistir o relacionamento de cada nó com os seus nós filhos.

Existem três estruturas de controle primitivas, denominadas JOIN, INCLUDE e OR. Outras estruturas de controle podem ser definidas a partir destas, denominadas Estruturas de Controle Derivadas. Neste trabalho, também são consideradas quatro estruturas de controle derivadas, apresentadas em Martin [MAR85A]: COJOIN, COINCLUDE, COOR e CONCUR. As estruturas de controle derivadas aumentam a flexibilidade da decomposição.

O formalismo associado aos "Control Maps" prevê um conjunto de regras de decomposição que torna o resultado deste passo uma representação correta da especificação. As estruturas de controle aqui citadas e suas regras estão resumidas em Hamilton [HAM76A, HAM76B] e Martin [MAR85A].

4.3.2) Caracterização da Entrevista

O modelo do processo de entrevista do AEsp, conforme descrito anteriormente, é baseado em perguntas direcionadas ao Especialista do Domínio que permitem especificar o problema segundo a metodologia de decomposição HOS.

O resultado do processo de entrevista sob o Entrevistador do AEsp é um "Control Map" correspondente à especificação do Especialista do Domínio [FER94B]. O modelo de entrevista do Entrevistador pode ser representado como na Figura 4.3.

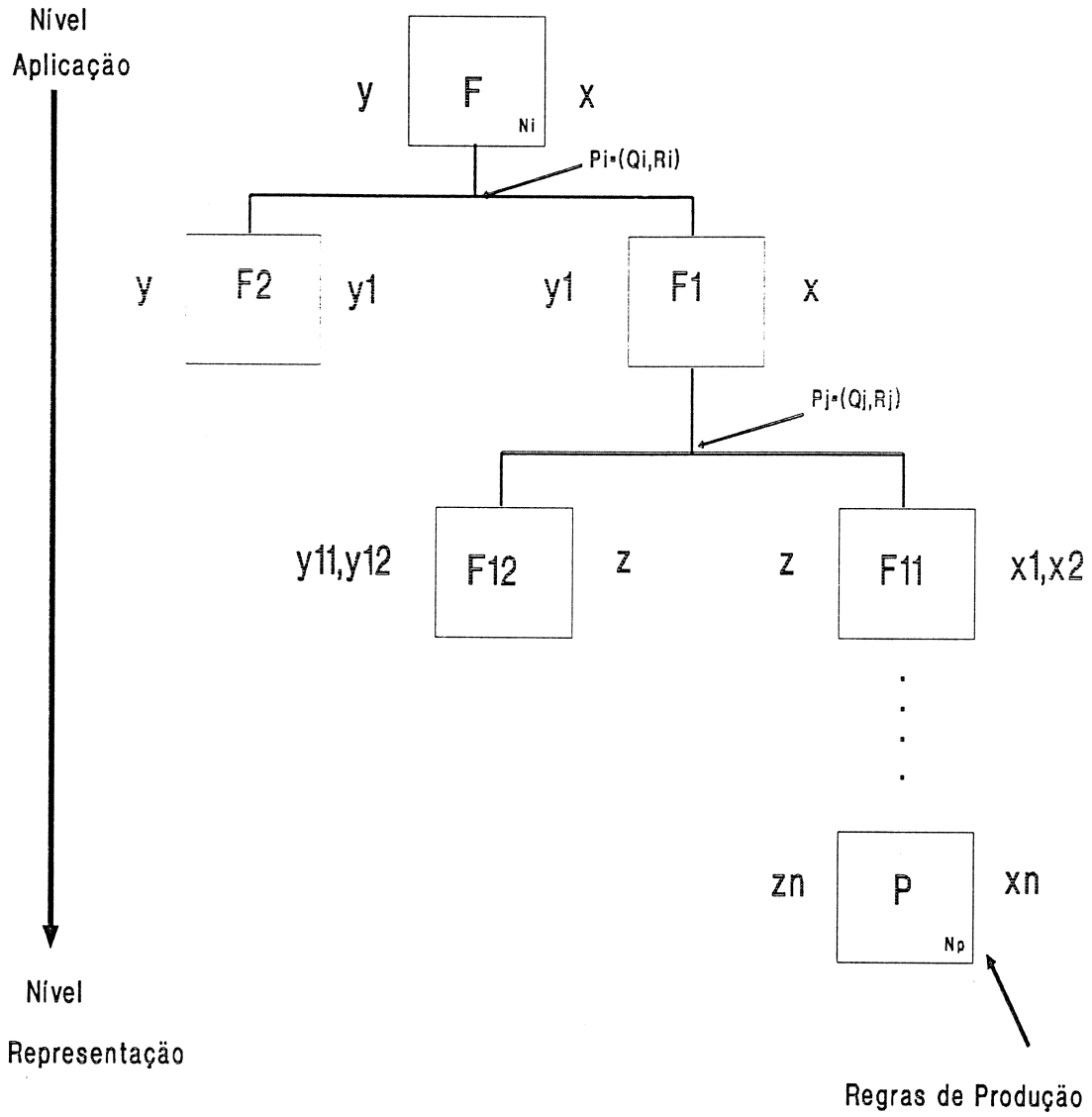


Figura 4.3: "Control Map" - Caracterização da Entrevista

Na Figura 4.3, em cada ponto da ramificação ocorre um par $P_i = (Q_i,R_i)$ correspondente à uma questão Q_i e uma resposta R_i. A especificação do usuário, na sua versão final seria, portanto, uma sequência $D_f = \{P_i\}$, i = 1,n.

Naturalmente a sequência $D_r = \{P_k\}$, k = 1,m com m >= n é a sequência real que levou a vários pontos de estrangulamento e que foi "backtracked" para D_f durante a interação com o usuário.

O processo de entrevista pode ser modelado como o processo de reconhecimento das sentenças {Ri} que leva do nó inicial (Ni) ao nó primitivo alvo (Np).

Pode-se descrever a entrevista, portanto, como o reconhecimento de uma linguagem descrita pela tripla D = (N,F,T)

onde,

N = conjunto de frases, compostas por verbos e substantivos, para os quais há regras de decomposição;

F = conjunto de regras que suportam a decomposição.

T = conjunto de verbos e substantivos redutíveis a operadores e operandos existentes no domínio;

O processo de reconhecimento desta linguagem pode ser explicado contextualizando-o dentro do processo de síntese das aplicações do domínio de administração rural, proposto na Metodologia de Desenvolvimento Rápido de Aplicações - MEDRA [FER94B].

A MEDRA prevê a existência de um conjunto de classes e subclasses que sustente o processo de síntese das aplicações. Esse conjunto foi definido na fase de pré-análise do domínio e está resumido na Tabela 4.1, onde indicam-se as principais classes e subclasses com o objetivo de esclarecer o modelo do processo de entrevista.

Na Tabela 4.1, além das principais classes e subclasses com suas respectivas funções, também são apresentadas as linguagens de descrição e os processos de tradução correspondentes a cada classe.

A primeira classe identificada, durante a atividade de pré-análise do domínio, foi a Modelagem da Aplicação correspondente ao Nível de Representação [MAS94B] do modelo de níveis do FMS. Esta classe é caracterizada pela representação dos elementos típicos das aplicações do domínio e seu comportamento. Dentre os elementos típicos levantados temos: Formulários, Relatórios, Eventos e Tabelas. A partir da definição destas subclasses, foi sintetizada a linguagem de descrição correspondente à classe Modelagem da Aplicação, denominada LC-FMS.

Na sequência, foram levantadas as subclasses da classe Representação para implementar os elementos do domínio, da classe Modelagem da Aplicação, sob a forma operacional. Algumas das subclasses identificadas foram: Base de Dados, "Windowing" e Help. A linguagem para descrição desta classe é composta por templates em código C, bibliotecas e trechos de programas.

A classe Operacionalização é composta pelas subclasses que permitem gerar na forma executável a classe Representação. A linguagem de descrição desta classe é a aplicação executável.

Classes	Decomposição	Domínio	Modelagem da Aplicação	Representação	Operacionalização
Subclasses	Dados Funções	Controlar Planejar Relatar	Forms Reports Eventos MEF Tabelas	Base de Dados "Windowing" Help	Bibliotecas Make Compiladores Linkers
Função	Transformação, sob a metodologia, de uma estrutura de dados ou função em outra mais adequada	Representar as demandas do usuário, expressas no seu jargão	Representar os elementos típicos das aplicações do domínio, seu comporta-mento	Implementar os elementos do domínio sob a forma operacional	Implementar uma forma executável
Linguagem de Descrição	Operadores de Control Map (CMO)	Control Maps Instanciáveis (CMI)	Linguagens (LC-FMS)	Templates em código C Bibliotecas Trechos de programas	Executáveis
Processo de Tradução	Control Map Matching (CMM)	Control Map Matching (CMM)	Compilador LC-FMS Macro-Expansor sintático	Compilador C Make language Link Language	--

Tabela 4.1 - Suporte à modelagem da Entrevista

Após a identificação das classes correspondentes aos níveis, representação e implementação, do modelo de níveis do FMS [MAS94B], foram definidas as classes correspondentes ao nível da aplicação.

A classe Domínio é caracterizada pelos termos que representam as demandas do usuário, expressas no seu jargão, tais como, Controlar, Planejar e Relatar. A linguagem utilizada para descrever esta classe é o próprio "Control Map". Os "Control Maps" que especificam os objetos do domínio são denominados "Control Maps " Instanciáveis - CMI.

Além dos "Control Maps", citados na Seção 4.3.1, serem utilizados como forma de representação da especificação gerada durante a entrevista e da especificação dos objetos do domínio (CMIs), eles também são utilizados para representar os operadores que implementam os CMIs, denominados operadores de "Control Map" - CMOs. Portanto, o termo "Control Map", neste trabalho, é utilizado em vários níveis.

A classe Decomposição permite transformar uma estrutura de funções e dados em outra mais adequada sob a metodologia de decomposição HOS. Os CMOs correspondem a linguagem de descrição desta classe.

Os operadores de "Control Map "- CMO, citados na Tabela 4.1, implementam as funções que permitem gerar um "Control Map" (CM ← F(CM)) onde F pertence às classes descritas na tabela abaixo:

F	**Descrição**	**Regras de Produção**
F_{deaf}	operador de decomposição de funções	regras HOS - Join, Include, Or e derivadas
F_{dead}	operador de decomposição de dados	funções morfológicas
F_{eval}	operador para instanciação de CM parametrizável	F(CMI)→ CM
F_{daco}	operador de criação de atributos do domínio	funções de propriedades classes e subclasses de atributos

Tabela 4.2: Operadores de "Control Maps"

O F_{deaf} é o operador de decomposição de funções, isto é, este operador implementa as regras das estruturas de controle do HOS, tais como, Join, Include, Or e derivadas. Estas regras garantem a consistência entre os níveis hierárquicos do "Control Map" a ser gerado.

O F_{dead} é o operador de decomposição de dados. Este operador permite, durante a decomposição em HOS, transformar um tipo de dado em outro.

O F_{eval} é o operador que implementa as funções necessárias para que, durante a entrevista, sejam instanciados os "Control Maps" Instanciáveis.

O F_{daco} é o operador que implementa as funções de propriedades de classes e subclasses de atributos do domínio que permitem a tradução dos objetos do Domínio em objetos da classe Modelagem da Aplicação.

Este elenco de operadores implementa a linguagem para descrição dos objetos do domínio, suas características morfo-funcionais e todos os atributos que suportam a sua instanciação para a especificação do usuário.

Para que a síntese das aplicações seja possível, é necessário construir um mecanismo de tradução entre os vários objetos e classes descritas acima. O processo de tradução inicia-se na classe Decomposição, desta para a classe Domínio e assim sucessivamente, até obter a classe Operacionalização (da esquerda para a direita na Tabela 4.1).

O modelo de entrevista do AEsp realiza parte deste mecanismo de tradução para síntese das aplicações. O modelo de entrevista proposto abrange:

a) reconhecimento, a partir do discurso do entrevistado, de objetos da classe Domínio;

b) instanciação ou redução dos objetos da classe Domínio a objetos da classe Modelagem da Aplicação.

c) Processamento dos objetos da classe Modelagem da Aplicação, segundo a MEDRA, para gerar a aplicação do usuário.

As seguintes premissas são assumidas durante o processo de reconhecimento e tradução inerente à entrevista:

- o vocabulário é finito e controlado;

- a função de sinonímia é suportada pelo Engenheiro de Especificação, usando o seu próprio conhecimento e o léxico ampliado [LEI90] existente para o domínio;

- o discurso é não metafórico ou poético. As metáforas, ao invés de introduzirem imagens no discurso, representam provavelmente elementos do domínio sem taxonomia adequada e deverão ser incorporadas ao Léxico Ampliado da Aplicação.

4.3.2.1) Processo de Tradução associado à Entrevista

O processo de tradução citado na Tabela 4.1, que é pertinente ao Entrevistador, denominado "Control Map Matching" - CMM, consiste em identificar, na base de objetos do domínio aqueles que, instanciados durante o processo de entrevista, sejam equivalentes à especificação do usuário. O processo é descrito como de "matching" para reforçar a metáfora do processo de tradução de linguagens.

A primeira etapa do processo de tradução da entrevista é o reconhecimento, a partir do discurso do entrevistado, dos objetos do domínio que foram identificados durante a atividade de pré-análise do domínio. Após a identificação, estes objetos do domínio foram especificados em HOS e armazenados em uma base de objetos do domínio. Portanto, durante o processo de entrevista o Engenheiro de Especificação deve reconhecer no discurso do entrevistado as funções e os dados que podem ser decompostos até reduzí-los aos objetos do domínio já existentes.

É importante lembrar que, durante o processo de entrevista, a decomposição termina quando se chega aos nós primitivos. Os nós primitivos, nesta fase, são os objetos da classe domínio com seus CMIs correspondentes, e os nós primitivos dos CMIs especificados são os CMOs. Portanto, os CMIs são a forma básica de reuso durante a entrevista.

Após o reconhecimento dos componentes do domínio, inicia-se o processo de instanciação dos CMIs correspondentes. Os CMIs, exemplificados na Figura 4.4, possuem nós onde,

x é um lado direito

y é um lado esquerdo

f é uma função arbitrária

p são predicados

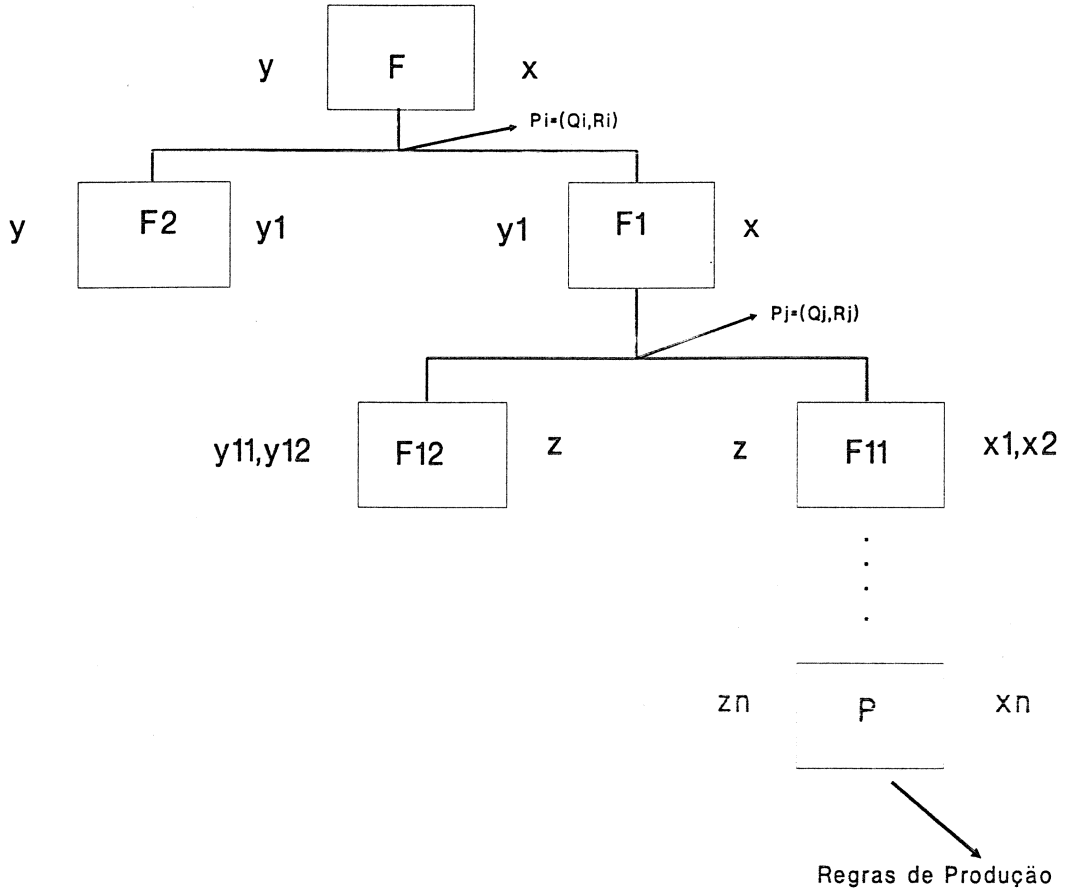


Figura 4.4: Control Map Instanciáveis - CMI

O processo de instanciação consiste em :

a) substituir x,y e f por valores (nomes) fornecidos no momento da avaliação e aplicar recursivamente a regra de instanciação;

b) executar as questões relativas a P durante a instanciação;

c) aplicar, para cada reavaliação da regra de instanciação, as regras Fdeaf e Fdead, isto é, executar as regras de produção que são aplicadas a partir dos CMOs (Tabela 4.2).

Os "Control Maps" foram estendidos para propiciar as regras de produção dos CMOs, apresentadas na Tabela 4.2. É necessário, para fins de preservar o alcance da metodologia em uso, provar que as extensões não afetam os teoremas que garantem o corretismo descrito em Hamilton e Zeldin [HAM76A,HAM76B].

O processo de tradução da entrevista do AEsp, apresentado na Figura 4.5, pode ser resumido no processo de instanciação dos CMIs. Este processo permite, a partir das questões dos CMIs e respostas dadas, gerar o "Control Map" Instanciado - CMP correspondente à especificação da aplicação do usuário.

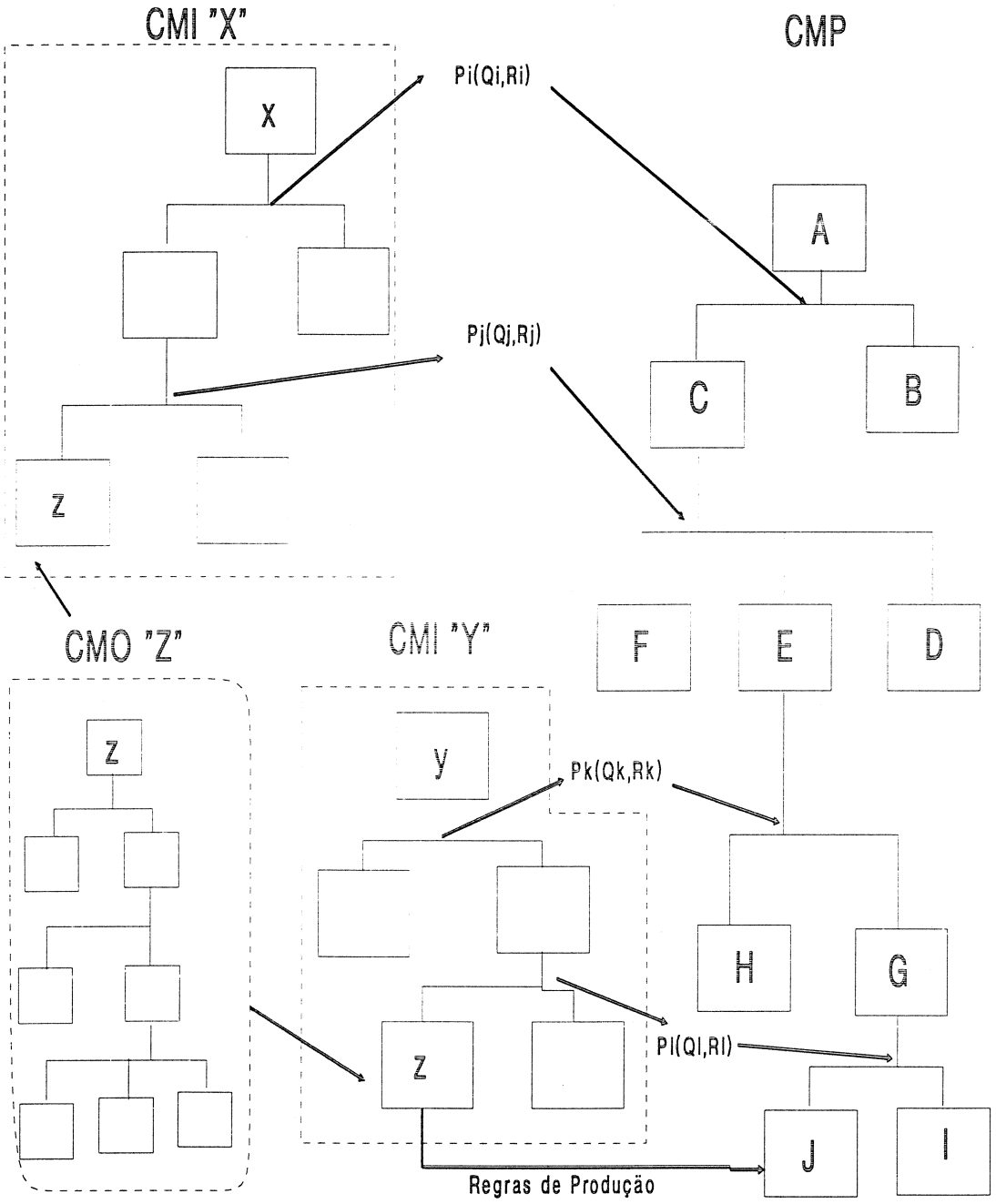


Figura 4.5: Processo de tradução da Entrevista

O corretismo global do "Control Map" Instanciado - CMP depende da implementação final dos nós primitivos e a interpretação do que se denomina, aqui, "Regras de Produção" dos nós primitivos. Essas regras, além de garantir a consistência do CMP, permitem reduzir os objetos da classe domínio a objetos da classe modelagem da aplicação, concluindo a segunda etapa do processo de tradução da entrevista.

Posteriormente, a classe modelagem da aplicação é processada utilizando os compiladores das sublinguagens da LC-FMS que a traduzem para a linguagem C. A síntese das aplicações é concluída com a compilação destes programas em C, que gera a aplicação executável [MEI92].

4.4) Um exemplo de nó-primitivo - "Controlar"

Nesta seção é dado um exemplo da subclasse "Controlar". É mostrada a análise local feita para identificar os objetos do domínio e regras de produção, e daí é mostrado o CMI do "Controlar". Por último, é apresentado um exemplo de um "Control Map" Instanciado - CMP gerado a partir da instanciação do CMI do "Controlar".

A classe controlar descreve os objetos do domínio representados pelos nós "Controlar <x> de <y>". Esses componentes foram identificados na pré-análise do domínio descrita na MEDRA [FER94B]. Eles são particulamente importantes porque introduzem o conceito de eventos para as aplicações do domínio em questão. A frase/cliché, sintetizada durante a atividade de pré-análise de domínio, está descrita abaixo:

Frase/Cliché	
controlar <X> de <Y>	
onde	X é o atributo a ser controlado Y é o objeto de controle caracterizado por um coletivo genérico

A Figura 4.6 apresenta o modelo do nó primitivo correspondente ao "Controlar <x> de <y>", sintetizado a partir da atividade de pré-análise do domínio.

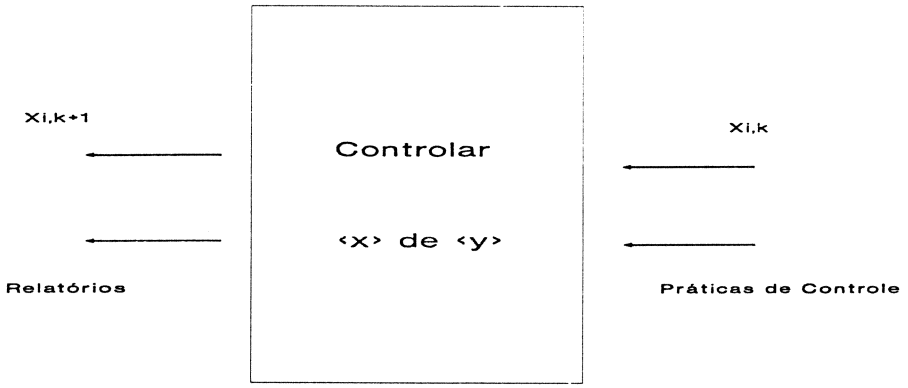


Figura 4.6: Modelo do nó "Controlar"

Portanto, as questões da entrevista foram elicitadas no sentido de obter as seguintes características do nó "Controlar", apresentado na Figura 4.6:

- Identificar as variáveis de estado e suas transformações
  $x_i^k \Rightarrow x_i^{k+1}$
- Coletar a máquina de estados finita (MEF) que implementa o controle
  Práticas de Controle

- Escolher os atributos e formato de dados
  Relatórios

- Identificar as ligações entre o estado e práticas de controle

  - Decomposição até que x_i seja um escalar monovalorado

  - Elicitação dos valores de x_i até a obtenção das tabelas de valores e intervalos
    $x_i^0 \quad x_i^1 \quad x_i^2 \quad \ldots \quad x_i^n$

  - Levantamento das tabelas de transição /ação
    $x_i^k \overset{\text{transição}}{\underset{\text{ação}}{\Rightarrow}} x_i^{k+1}$

  - Associação da transição a um evento (regras de disparo/funções de ação)

A) Síntese das questões

A síntese das questões do nó "Controlar", para elicitar as características definidas na atividade de pré-análise do domínio, seguiu a seguinte premissa:

(a) Estabelecer que <X> é um atributo de controle, e as propriedades do atributo de controle são:

a.1) <X> deve ter um número finito de alternativas (cardinalidade) (valores ou intervalos não sobreponíveis);

a.2) deve ser possível construir a função
$(x_i,y_j) \Rightarrow (x_k,y_j)$
onde $\{ x_i \mid i \in \# X\}$
$\{xj \mid j \in \# Y\}$
e,
Y - (cardinalidade de Y) é o número de membros ou componentes do coletivo Y.

X - (cardinalidade de X) é o número de diferentes valores que a propriedade X pode assumir.

$(x_i,y_j) \in \{X * Y\}$
$(x_k,y_j) \in \{X * Y\}$

$(x_i,y_j) \Rightarrow$ não $\exists (x_k,y_j)$
$k = 1, \# X$
$k <> i$

portanto, cada y_j tem, a cada instante, um único valor da propriedade x.

A partir desta premissa, definiram-se as seguintes questões:

Questão 1: (estabelecer a cardinalidade de y)

(Q1) <Y> é um coletivo de que indíviduo?

(Exemplo: (animais ou rebanho) é coletivo de (vacas,bezerros)

Note-se:

(a) Esta questão, que parece ser expletiva nesta altura da entrevista, é essencial para permitir o "join" dos arquivos sobre Y - ou seja - juntar todos os atributos de Y numa mesma base - o que é importante para desempenho e simplicidade.

(b) Numa mesma aplicação poderão aparecer diferentes coletivos para representar subconjuntos (do coletivo mais geral) que têm nome especial por um atributo (valor) que exibem. Exemplos:

	subconjunto	**atributo**
Novilhas/bezerros	animais/rebanho	idade
Livros atrasados	acervo	tempo de empréstimo
Materiais de Alto Custo	estoque	valor unitário

Tabela 4.3: Exemplos de coletivos

Esses subconjuntos devem ser tratados, durante a entrevista pelo Engenheiro de Especificação, no momento em que são citados pelo Especialista do Domínio porque podem exigir códigos de escape (aumento da cardinalidade de X).

(c) Se não se puder identificar a pertinência de y_i a Y, isto é uma indicação de que o coletivo (conjunto ou subconjunto) foi mal escolhido. Provavelmente é um impasse da entrevista e será necessário "backtracking".

(d) A metodologia aplicada conduz na direção de separar o estado global E em $E_1, ..., E_n$ onde $E = U\ E_i$ p/ i= 1,n; o que é altamente recomendável do ponto de vista funcional. Porém essa separação também implica em intersecção de $E_iE_j = 0$ para i diferente de j (o que é garantido pela metodologia HOS) e haverá, caso não sejam tomados os cuidados previstos em (a), uma multitude de pequenas relações representando E_i.

Questão 2: (estabelecer a cardinalidade de X).

(Q2) Quais são os possíveis valores de <X>?

A resposta a esta questão deverá ser:

(a) um conjunto de valores (X é discreta);

(b) um conjunto de cadeias (X é discreta);

(c) uma tabela de intervalos de (X é contínua);

(x_i , x_j)
(x_{j+1} , x_k)
(x_{k+1} , x_l)
.
.
.
(x_{n+1} , x_n)

(d) negativa - o que implica em mudança de estratégia, descrita no item A.1 abaixo.

Caso a resposta não possa ser dada sob uma das 3 formas acima, isto é uma indicação de que <X> não é um atributo (é provavelmente multivalorado).

Nos três casos acima é necessário criar uma representação para o estado de Y no que concerne ao atributo X. Provavelmente, a anotação de enumeração é a melhor. Na base de conhecimento as alternativas são armazenadas como uma enumeração. No caso (c) é necessário criar duas representações para o atributo X - um para o valor de X (contínuo) e outro para I_X, discreto, que designa o estado. A tabela de $X \Rightarrow I_X$ deverá ser armazenada para permitir a avaliação do estado durante a execução do programa.

Note-se que os valores serão usados com o objetivo de:

(a) representar o estado;
(b) permitir as consistências;
(c) gerar a lista de "choices" para entrada de dados (na LC-FMS, somente é possível se for um número pequeno).

Neste ponto, caso tenha sido possível construir a cardinalidade de X e Y e demonstrar que cada y_i tem apenas um x_j (função), tem-se a seguinte configuração da MEF (incompleta).

x_1	x_2	x_3	x_4	x_5

Esse diagrama representa o estado em que cada y_i pode estar. Cabe agora levantar as transições e as transformações associadas a essa MEF.

Questão 3: (levantar as possíveis transições)

Para todos os x_i, efetuar a seguinte questão:

Questão 3.1: (versão espacial)

(Q3.1) Como cada y_i pode passar de x_i para x_j (com i diferente de j) ?

Questão 3.2: (versão temporal)

(Q3.2) Quando cada y_i pode passar de x_i para x_j (i diferente de j) ?

A versão temporal ou espacial da questão dará origem à seguinte MEF:

$x_1 \underset{tr_1}{\Rightarrow}$ $x_2 \underset{tr_2}{\Rightarrow}$ $x_3 \underset{tr_3}{\Rightarrow}$ $x_4 \underset{tr_4}{\Rightarrow}$ x_5

Essa MEF (ainda incompleta) deverá ser armazenada.

E_t	E_{t+1}	Tr_i
x_1	x_1	regra de passagem de x_i -> x_i
x_1	x_2	regra de passagem de x_1 -> x_2
x_2	x_n	regra de passagem de x_2 -> x_n
.		
.		
.		
x_n	x_n	regras de passagem de x_n -> x_n

Essas regras deverão ser armazenadas textualmente (ou formalizadas se for possível).

Caso fique patente durante esse processo que há algum estado não catalogado, ou que é impossível criar uma regra de transição, deve-se voltar ao estado anterior e propor uma reanálise do atributo.

Esse conjunto de regras reflete as práticas de Controle pelo usuário para controlar a variável <X> de <Y>. Na atual implementação do FMS, a transição é escalada por um evento e realizada pela confirmação do evento. Há, portanto, uma dualidade (evento, regra Tr_i).

Essa dualidade permite associar a cada transição um evento e, portanto, colecionar as ações (último componente da MEF) ligadas aos eventos.

Transição	evento	ação
tr_i	v_i	a_i
.	.	.
.	.	.
.	.	.
tr_n	v_n	a_n

As ações denotam as transformações efetuadas sobre Ei. Essas transformações são de 2 naturezas:

(a) mudança do valor do atributo e de outras variáveis associadas (histórico, por exemplo)

(b) escalação dos eventos futuros.

As transformações da classe (a) acima são convencionais. Sua captura pode ser na forma textual ou pode obedecer as mesmas regras da metodologia HOS.

As transformações da classe (b) - escalação dos eventos futuros, permite um controle centralizado, isto é, após a catalogação do evento que o programou devem ser catalogados os parâmetros deste novo evento. Este novo evento disparado pode aparecer como uma nova transição ou não. Caso não apareça como uma nova transição, no final da sessão devem ser catalogados os seus parâmetros.

A.1) Mudança de estratégia

Esta mudança é causada pela impossibilidade, durante o processo de entrevista, de elucidar quais são os possíveis valores de uma variável de estado.

A Questão 2 exige que o usuário tenha abstraído qual a varíavel de estado e quais seus valores. Caso não consiga obter esses valores, a entrevista deverá entrar no modo de simulação [LUH94].

Questão 4: **O que é feito hoje para Controlar o <X> de <Y>? (Q4)**

No modo de simulação não é necessário identificar inicialmente os valores de <X>. O caminho natural é identificar os eventos correspondentes às ações do usuário. Como existe uma dualidade entre evento e transição, e como cada transição ocorre entre um x_i e x_j, a identificação de todos os eventos implica na identificação de todas as transições e, por conseguinte, no levantamento de cardinalidade (X).

A notação complementar mais adequada para recolher esse conhecimento é possivelmente um DFD/SADT onde são anotadas as ações (verbos) descritas pelos usuários e os documentos (dados) usados para controlar são anotados nos arcos. A anotação mais indicada é a que permite a idéia de sequência. Talvez usando os modelos correntes de Engenharia de Processos [SCH93].

4.4.1) "Control Map" Instanciável do "Controlar"

As Figuras 4.7(a), 4.7(b) e 4.7(c) mostram parte do "Control Map" Instanciável do "Controlar" para ilustrar a aplicabilidade das questões definidas acima. É mostrado também, nas Figuras 4.8(a) e 4.8(b), a especificação de um CMO utilizado na especificação do CMI do "Controlar".

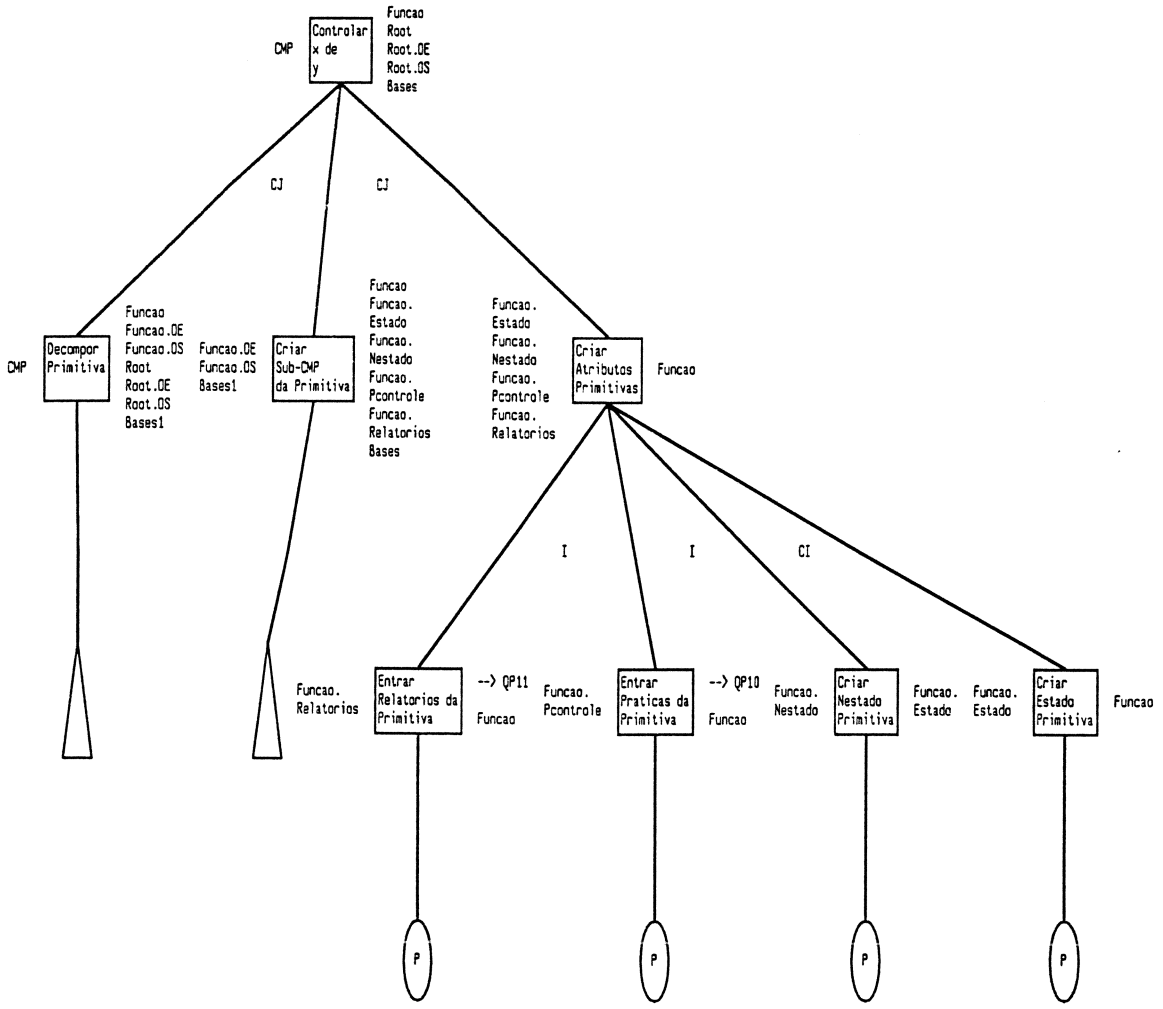


Figura 4.7(a): CMI do "Controlar"

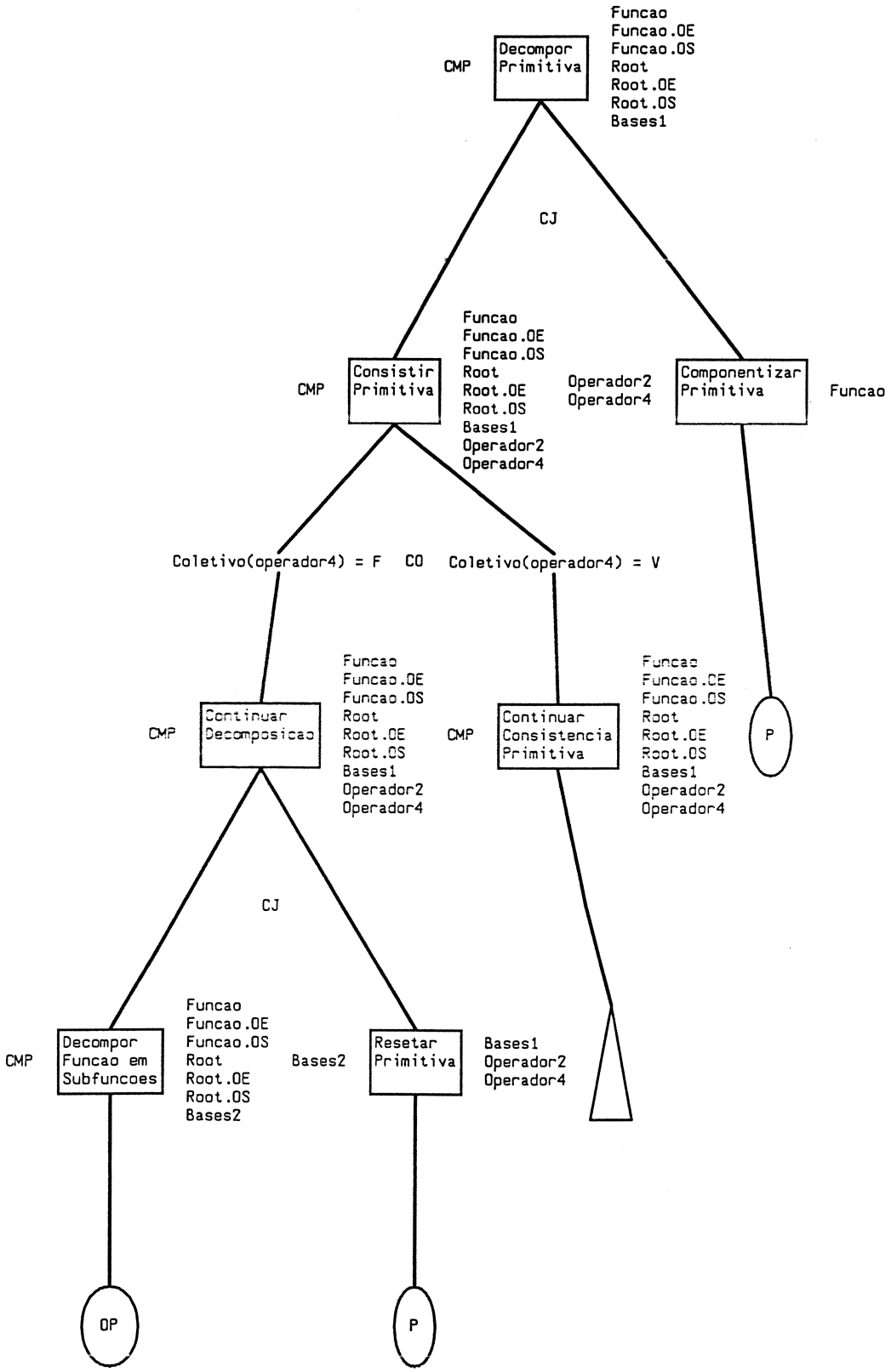


Figura 4.7(b): CMI do "Controlar" (continuação)

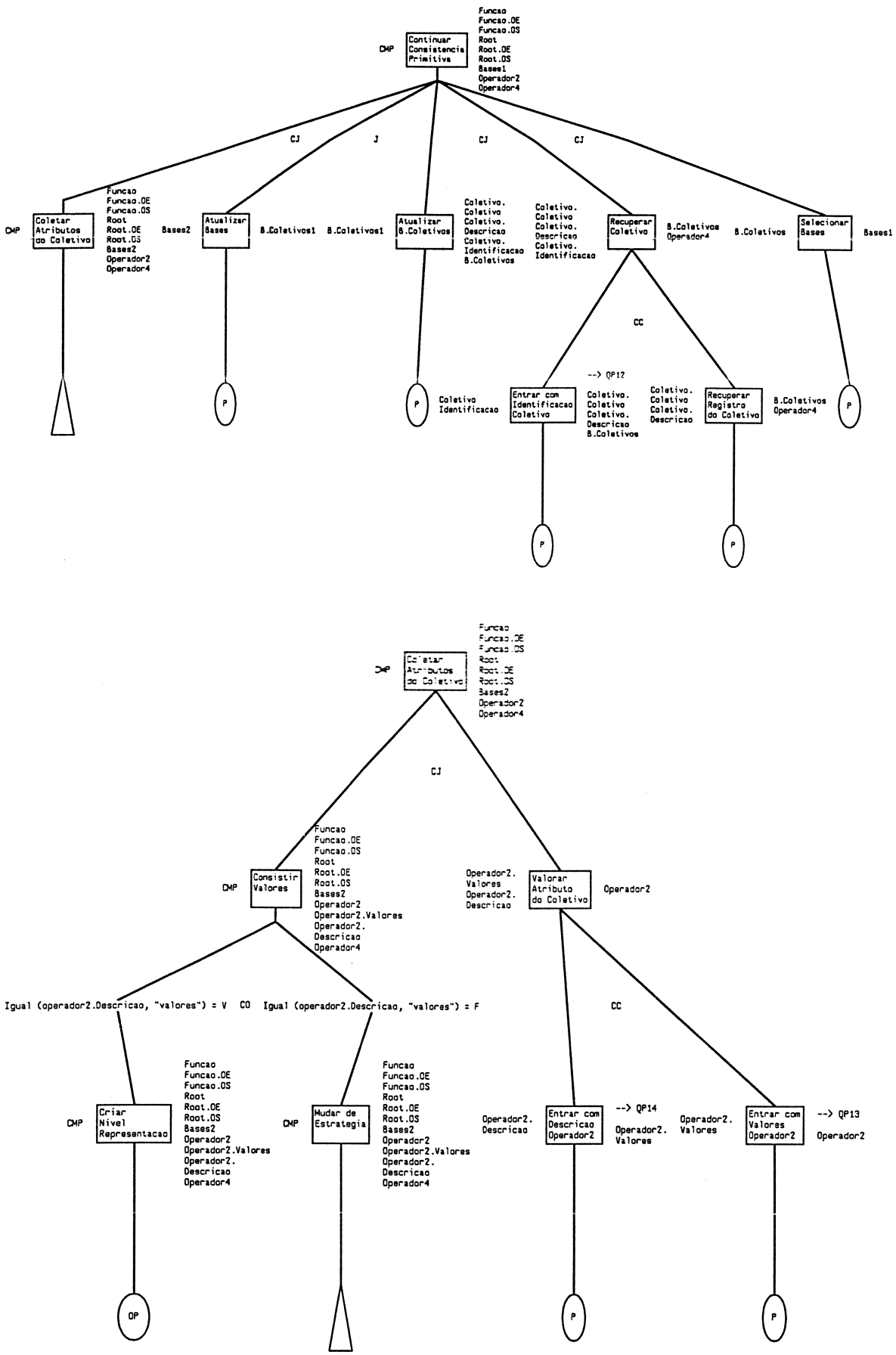


Figura 4.7(c) : CMI do "Controlar" (continuação)

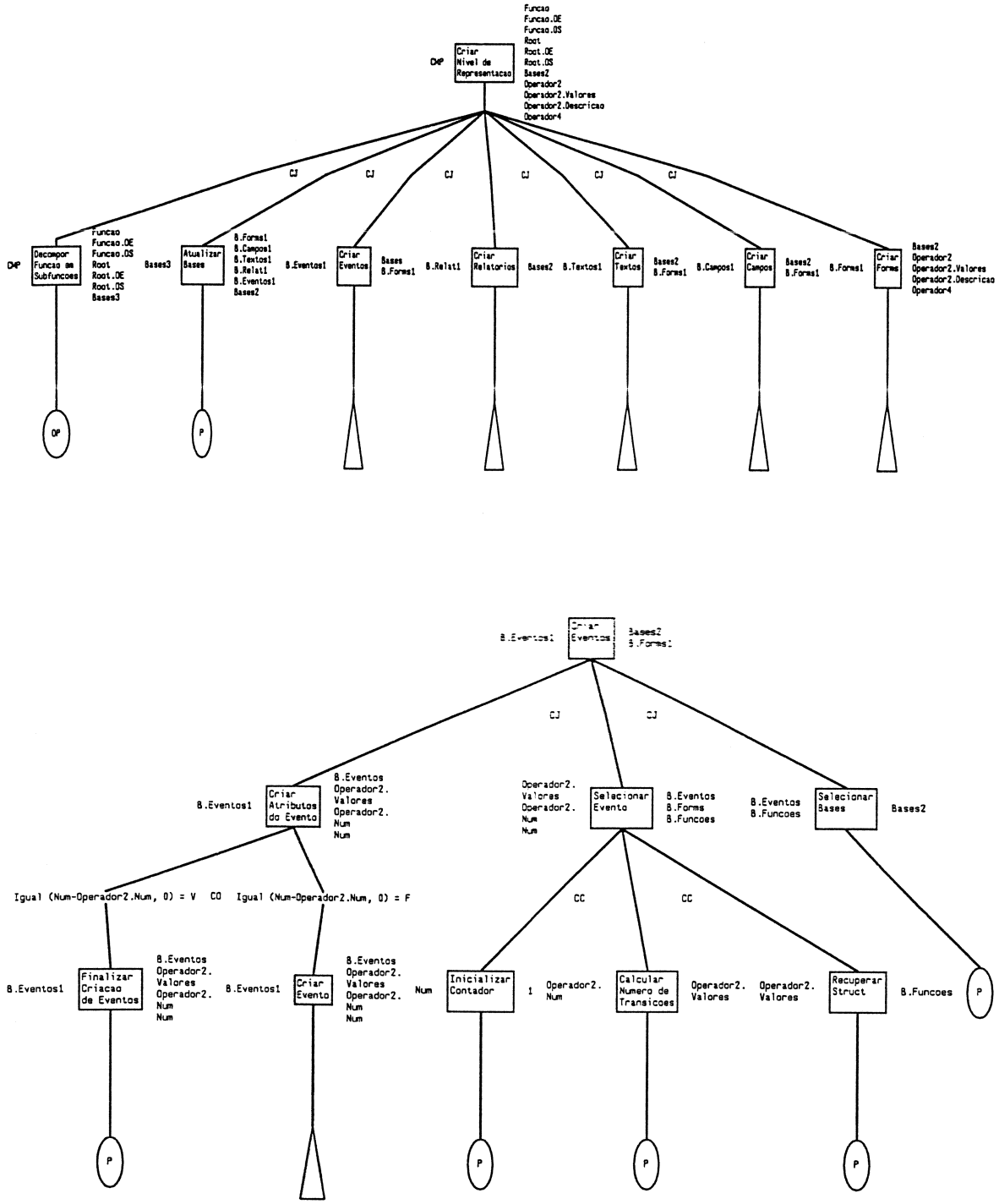


Figura 4.8(a): CMO - Criar Nível de Representação

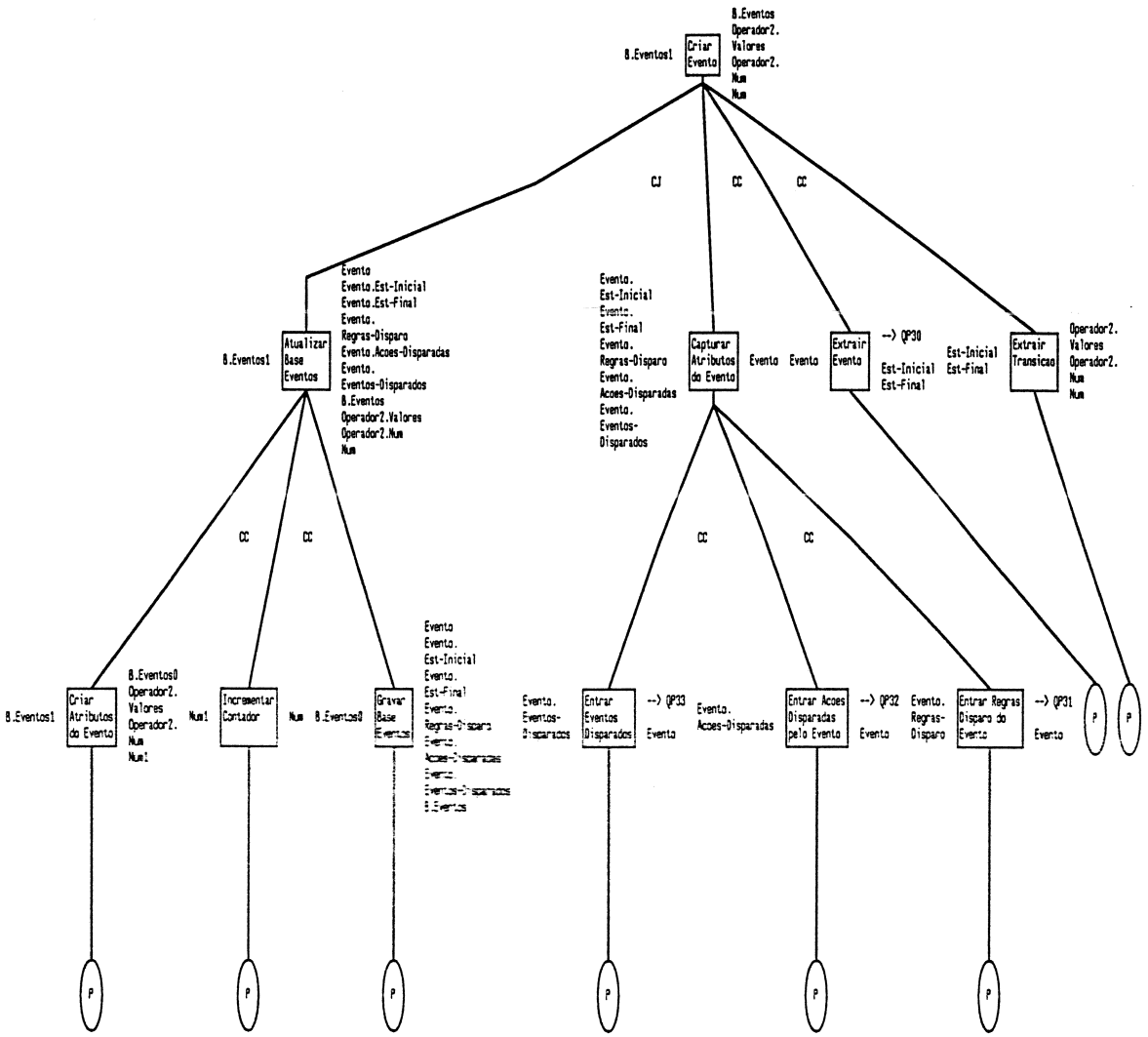


Figura 4.8 (b): CMO - Criar Nível de Representação (continuação)

Note que as questões (QP10, QP11, QP12, QP13, QP14, QP30, QP31, QP32, QP33) que aparecem nos "Control Maps" correspondem a algumas das questões necessárias para elicitar as questões (Q1, Q2, Q3) identificadas no processo de análise acima descrito. As respostas dadas a estas questões geram um "Control Map" Instanciado - CMP. No apêndice A está descrita a especificação de outros CMIs e CMOs identificados com a lista de questões correspondentes.

4.4.2) "Control Map" Instanciado do "Controlar"

A Figura 4.9 mostra um exemplo de um "Control Map" Instanciado - CMP, correspondente à especificação de uma aplicação do usuário, gerado a partir da instanciação do CMI do "Controlar".

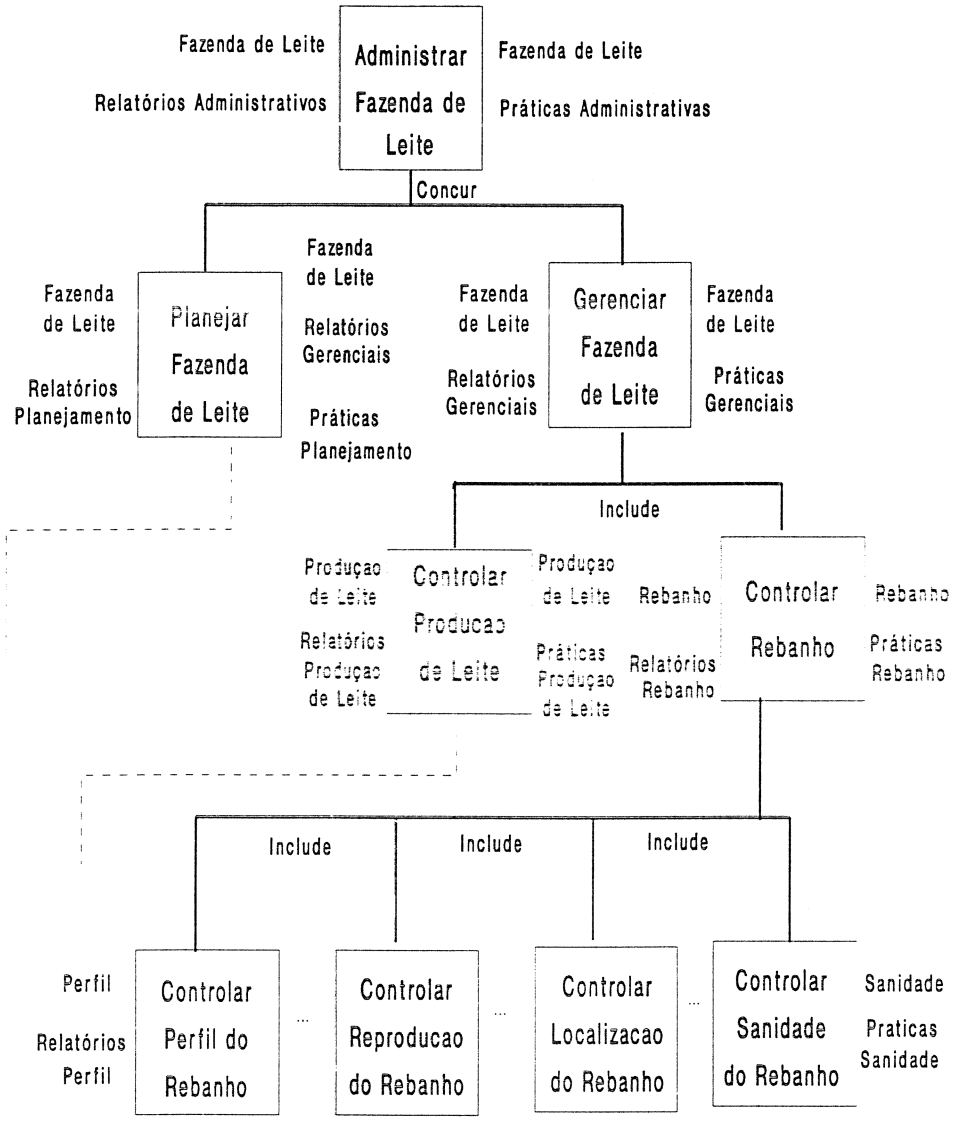


Figura 4.9: "Control Map" Instanciado - CMP

4.5) Considerações sobre o Capítulo

Neste capítulo foi descrito detalhadamente o modelo conceitual do processo de entrevista do Entrevistador que é a parte central do AEsp e onde está baseada a sua implementação que será descrita no próximo capítulo.

CAPÍTULO 5

MODELO DE IMPLEMENTAÇÃO DO AEsp

5.1) Introdução

Este capítulo tem como objetivo descrever o processo de construção do AEsp para implementar os conceitos apresentados no Capítulo 4. Devido ao papel central do Entrevistador na arquitetura do AEsp, a implementação desta ferramenta será apresentada primeiramente e, posteriormente, será dada uma descrição mais sucinta das outras ferramentas do AEsp.

5.2) Arquitetura Geral do AEsp

Conforme já descrito, o Entrevistador é o módulo central do AEsp. Ele tem como função gerenciar a coleta ordenada das especificações do usuário, sua documentação e transformação para uma representação operacional. As funções e relações do Entrevistador com as outras ferramentas do AEsp, apresentadas na Figura 5.1 no formato SADT, são:

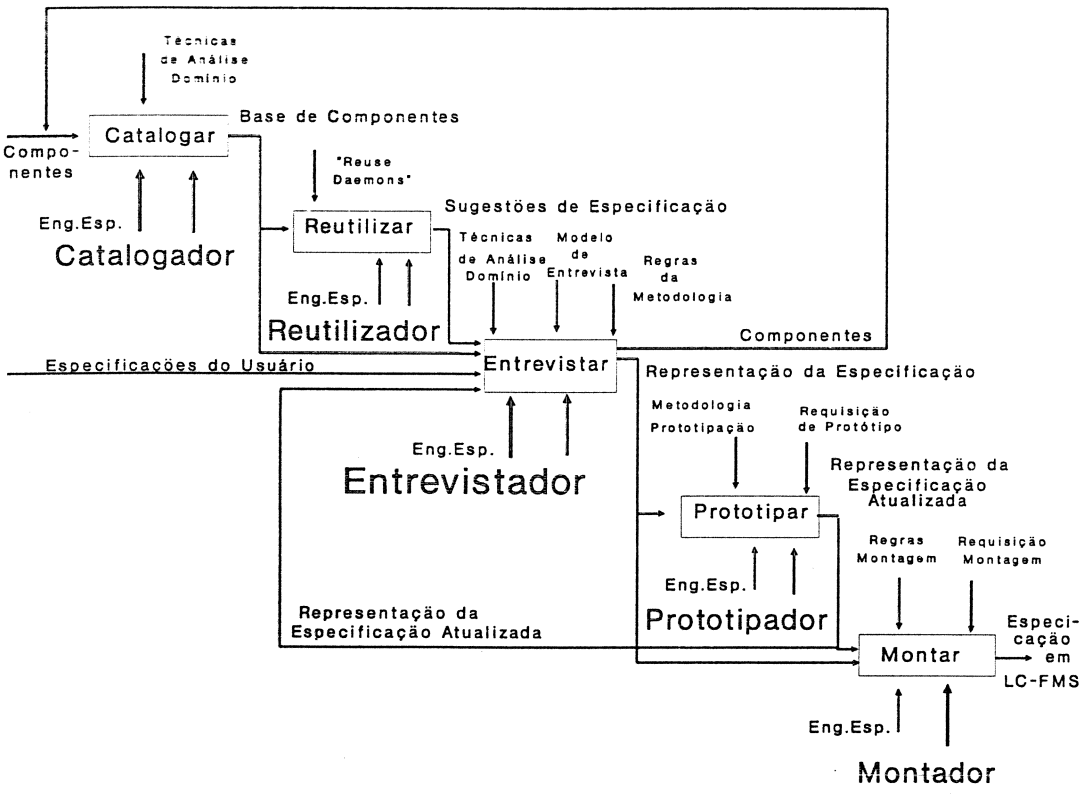


Figura 5.1: Arquitetura Geral do Aesp

a) Manter uma especificação corrente, seja pelo processo de decomposição ou pelo processo de prototipação, subsidiando o Prototipador e Montador, conforme

necessidade do Engenheiro de Especificação ou do Especialista do Domínio (usuário);

b) Selecionar, segundo as regras da entrevista, os componentes do Catalogador;

c) Exibir as sugestões da especificação oriundas do Reutilizador e validá-las, perante o usuário, usando as regras da entrevista e o Prototipador.

A Figura 5.2 mostra a interface do AEsp, padrão Openlook [DAN90], onde cada botão corresponde a uma das ferramentas da arquitetura do AEsp, que serão descritas neste capítulo.

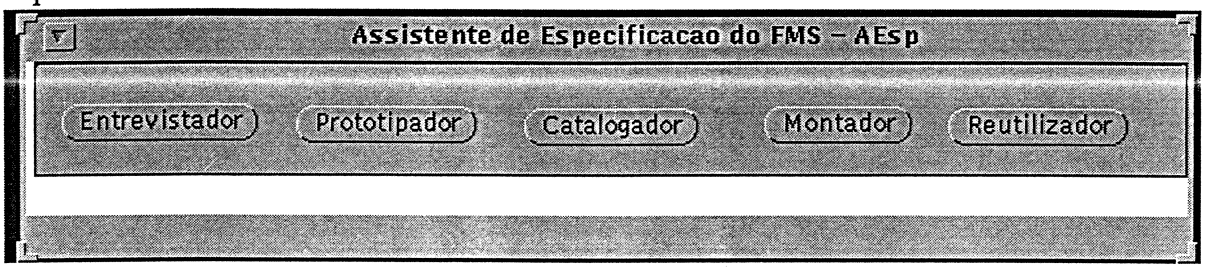


Figura 5.2: Interface do AEsp

5.3) Entrevistador

O Entrevistador é o **gerenciador da entrevista**, que captura os requisitos do especialista do domínio seguindo o modelo de entrevista proposto no Capítulo 4. A Figura 5.3 apresenta, no formato SADT [PRE92], a especificação das funções do Entrevistador para desempenhar seu papel gerencial.

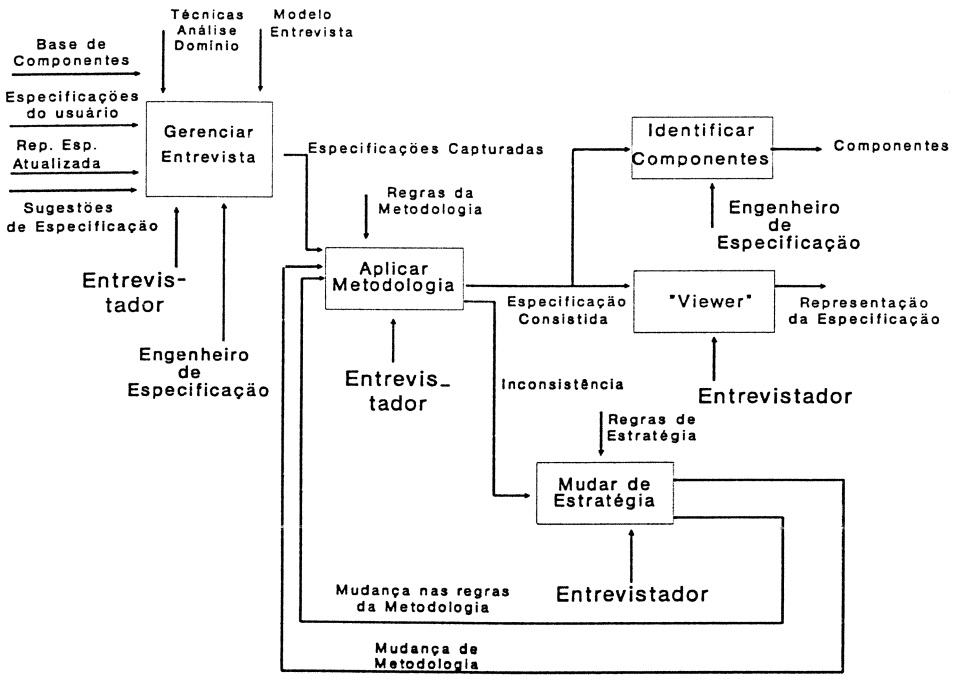


Figura 5.3: Entrevistador

O Entrevistador é também o **aplicador das regras da metodologia** de decomposição HOS, durante a entrevista. O Entrevistador conduz a entrevista, gerando a árvore de decomposição em HOS, até chegar aos nós primitivos e permite que as regras de produção, correspondentes a esses nós, sejam aplicadas para obter a especificação da aplicação descrita em LC-FMS.

O Entrevistador também funciona como um "**Viewer**", isto é, em qualquer momento da entrevista o Engenheiro de Especificação pode ver graficamente a árvore de decomposição que está sendo gerada e identificar componentes do domínio a partir da especificação da aplicação. O processo de decomposição pode ser iniciado de qualquer nó da árvore.

O modelo do Entrevistador prevê que caso a decomposição chegue a um "dead end" ou uma inconsistência, um "**aconselhador de estratégia**" possa sugerir uma reorientação da decomposição.

Todas as informações capturadas pelo Entrevistador ficam armazenadas em uma base de conhecimento de modo que possam ser reusadas na especificação de uma nova aplicação do domínio.

A partir do conjunto de funções descritas na Figura 5.3 e as relações descritas na Seção 5.2, pode-se sintetizar as seguintes propriedades do Entrevistador a serem implementadas:

a) modelo de entrevista para refinamento do problema;

b) suporte à decomposição do problema segundo a metodologia HOS e aplicação das regras previstas pela metodologia;

c) suporte à reutilização dos objetos do domínio;

d) suporte a mudanças de estratégia na decomposição quando necessário;

e) suporte à visão global do processo de decomposição.

5.3.1) Implementação do Entrevistador

Dadas as propriedades sintetizadas acima e as considerações teóricas mencionadas no Capítulo 4, a implementação do Entrevistador foi dividida em duas etapas: Implementação dos CMOs e Especificação e Implementação dos CMIs.

A primeira etapa visa implementar as propriedades (b) (d) e (e) e a segunda etapa visa abranger as propriedades (a) e (c) descritas na Seção 5.3.

Na versão corrente, o Entrevistador foi implementado usando uma base de conhecimento em Nexpert Object [NEX91]. O Nexpert é um sistema híbrido que possui uma máquina de inferência e uma representação orientada a objetos. Esta ferramenta foi utilizada devido a essas características que foram reconhecidas como apropriadas para atividades típicas do papel do Entrevistador. Na descrição de cada uma das fases de implementação do Entrevistador, essas características do Nexpert poderão ser melhor visualizadas.

5.3.1.1) Implementação dos CMOs

O macro objetivo do Entrevistador é auxiliar na tradução entre as três primeiras classes da Tabela 4.1. Primeiramente, na implementação do Entrevistador, foram definidas as classes e subclasses/objetos apresentadas na Tabela 4.1, tanto no nível da aplicação

(Decomposição, Domínio) quanto no nível de representação (Modelagem da Aplicação). Em uma segunda etapa, foram implementadas as regras em Nexpert que, manuseando estas classes, fazem a tradução entre elas.

As subclasses da classe Decomposição definidas foram: *Funções e Dados*. Na classe Domínio foi definida a subclasse *Objetos do Domínio*. Referente à classe Modelagem da Aplicação foram definidas as seguintes subclasses: *Forms, Campos, Textos, Transições, Relatórios, Eventos.*

A representação orientada a objetos do Nexpert foi utilizada para especificação destas classes e subclasses. O Nexpert suporta os conceitos de modelagem orientada a objetos: classes, objetos, atributos, hierarquia, especialização, instanciação. Além disso, a representação gráfica destas classes e objetos, suportada pelo Nexpert, auxiliam na visualização do processo de decomposição, facilitando o Entrevistador a desempenhar sua função de "Viewer".

A Figura 5.4 mostra um exemplo da representação gráfica em Nexpert da subclasse *Funções,* pertencente a classe Decomposição.

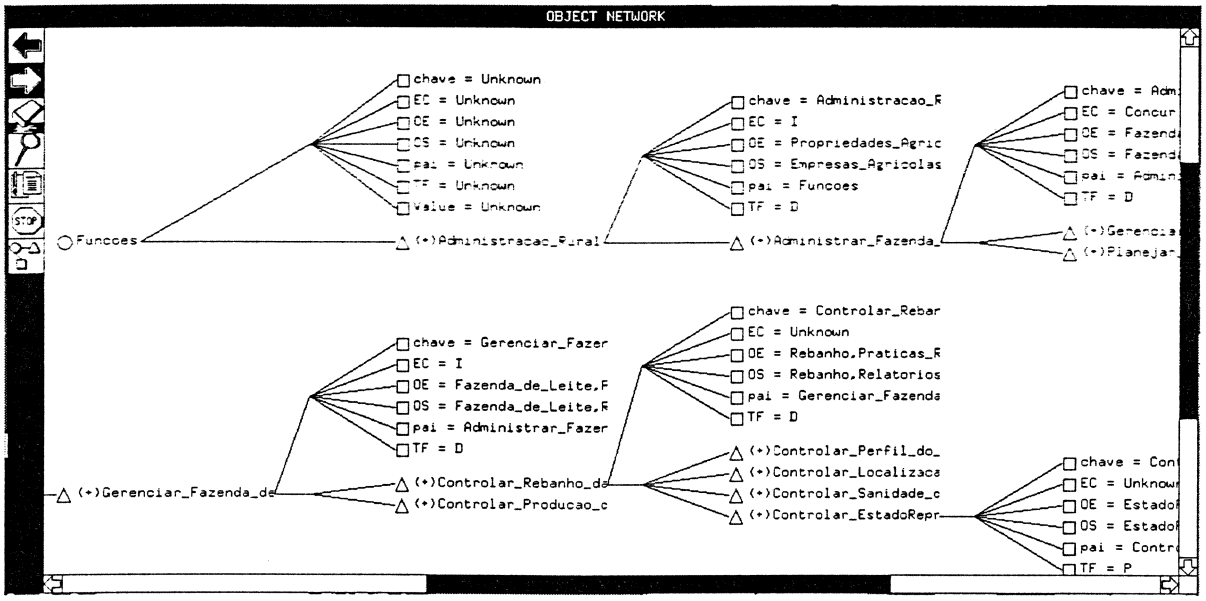


Figura 5.4: Representação Gráfica da classe *Funções* em Nexpert

O ○ (círculo) representa a classe , o □ (quadrado) representa os atributos desta classe e o Δ (triângulo) representa um objeto desta classe.

Os atributos da subclasse Funções foram baseados na decomposição em HOS. Estes atributos são:

- OE: Objetos de Entrada
- OS: Objetos de Saída
- EC: Estruturas de Controle (Join, Include, Or, Cojoin, Coinclude, Coor, Concur)
- TF: Tipo da Função (Primitiva (P) ou Decomponível ((D))

O "Control Map" Instanciado, correspondente à especificação da aplicação do usuário, é gerado sob a classe Funções. O "Control Map" Instanciado pode ser visualizado, durante a entrevista, conforme a Figura 5.4.

Os atributos da subclasse Dados, também pertencente à classe Decomposição, foram definidos de modo a permitir a decomposição de dados durante a entrevista.

A Figura 5.5 mostra o exemplo da subclasse Objetos do Domínio com seus respectivos atributos. Os atributos desta classe foram definidos de modo a permitir o cadastramento e recuperação destes objetos.

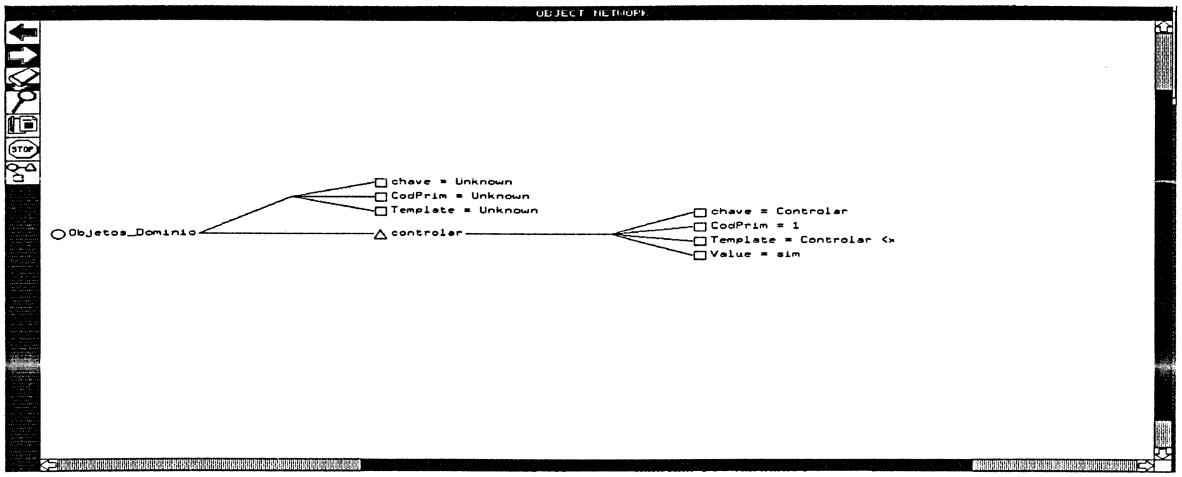


Figura 5.5: Classe Objetos do Domínio

Os atributos correspondentes à classe Objetos do Domínio são:

- Chave: Nome do Objeto do Domínio
- CodPrim: Código do Objeto do Domínio. Este código sequencial permite a instanciação durante a entrevista.
- Template: Atributo que contém a frase/cliché, sintetizada na atividade de pré-análise do domínio, isto é, o atributo que contém os parâmetros do objeto.

As subclasses da classe Modelagem da Aplicação foram baseadas nas sublinguagens da LC-FMS, descritas no Capítulo 3. A Figura 5.6 apresenta, como um exemplo das subclasses desta classe, a subclasse Forms.

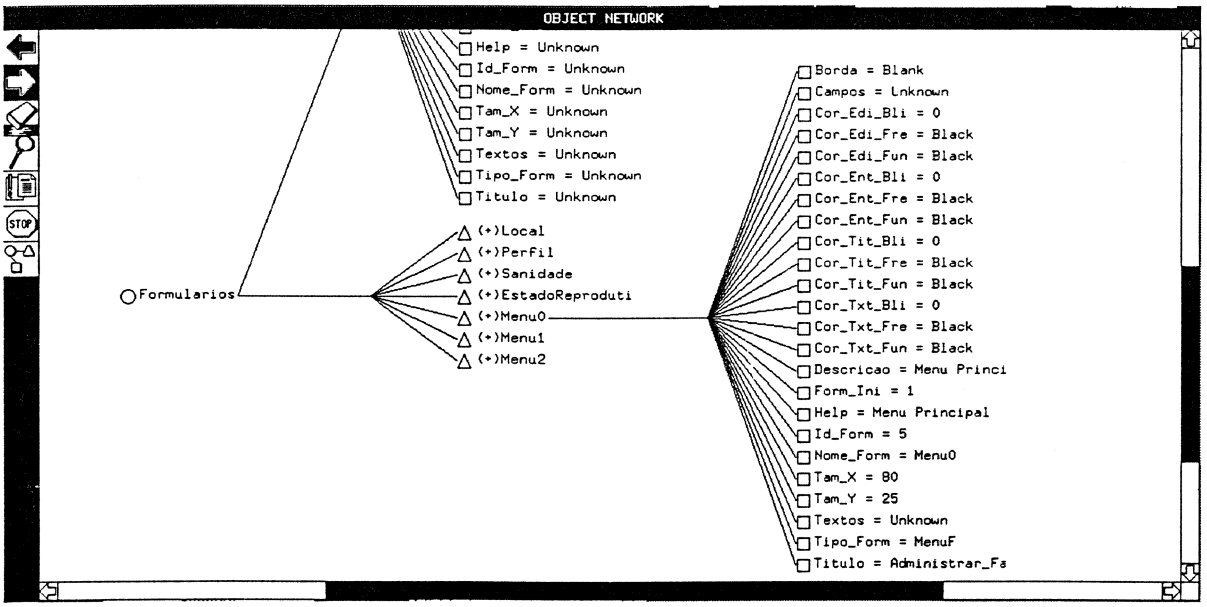


Figura 5.6: Representação gráfica da classe *Forms* em Nexpert

Os atributos da classe Forms foram definidos a partir da sublinguagem de Formulários. Estes atributos, descritos em ordem alfabética, são:

- Borda: Tipo da borda da tela (Blank, Single, Double)
- Campos: Campos pertencente àquele formulário
- Cor_Edi_Fre: Cor de Edição do campo do Formulário (Frente)
- Cor_Edi_Fun: Cor de Edição do campo do Formulário (Fundo)
- Cor_Edi_Bli: Especifica se a cor de edição do campo do Formulário pisca ou não (Blink)
- Cor_Ent_Fre: Cor de Entrada do Formulário (Frente)
- Cor_Ent_Fun: Cor de Entrada do Formulário (Fundo)
- Cor_Ent_Bli: Especifica se a cor de entrada do campo do Formulário pisca ou não (Blink)
- Cor_Tit_Fre: Cor do Título do Formulário (Frente)
- Cor_Tit_Fun: Cor do Título do Formulário (Fundo)
- Cor_Tit_Bli: Especifica se a cor do Título do Formulário pisca ou não (Blink)
- Cor_Txt_Fre: Cor do Texto do Formulário (Frente)
- Cor_Txt_Fun: Cor do Texto do Formulário (Fundo)
- Cor_Txt_Bli: Especifica se a cor do Texto do Formulário pisca ou não (Blink)
- Descrição: Descrição do Formulário
- Form_Ini: Caracterização do formulário (Inicial ou não)
- Help: Texto explicativo sobre o formulário
- Id_Form: Número de identificação do Formulário
- Nome_Form: Nome do Formulário
- Tam_X: Número de colunas da tela (máximo 80)
- Tam_Y: Número de linhas na tela (máximo 25)
- Textos: Textos do Formulário
- Tipo_Form: Tipo do Formulário (Edit, NoEdit, Browse, Config, Notice, Ev_Entry, Ev_Confirm)
- Título: Título do Formulário

Os termos em inglês que aparecem nos atributos dos formulários devem-se à especificação da sublinguagem de formulários. As sublinguagens da LC-FMS foram baseadas nas ferramentas de software utilizadas para compor a classe Representação (Tabela 4.1).

Após a implementação destas classes em Nexpert (Funções, Dados, Forms, Campos, Textos, Transições, Relatórios, Eventos e Objetos do Domínio), foram definidas as regras em Nexpert para implementar os operadores de "Control Map" - CMOs, isto é, as regras que manuseiam estas classes/objetos para auxiliar na tradução do processo de entrevista.

Os CMOs (Fdeaf, Fdead,Feval, Fdaco) são as funções que permitem gerar um "Control Map" Instanciado e seu nível de representação. Estas funções foram especificadas em HOS e foram implementadas utilizando as regras do sistema de inferência do Nexpert. O formato de uma regra em Nexpert é expressa como:

If ... then ... and do ...

onde

if	*then ...*	*and do ...*
(condiçoes)	(hipótese)	(ações)

A Figura 5.7 mostra a especificação do CMO - Popular CMP. Este operador permite criar níveis hierárquicos do "Control Map" Instanciado. Na implementação em Nexpert, cada operador pode ser representado por uma ou várias regras. A Figura 5.8 mostra uma regra implementada em Nexpert, denominada Decompor_Filhos, pertencente ao conjunto de regras desenvolvidas para suportar este operador.

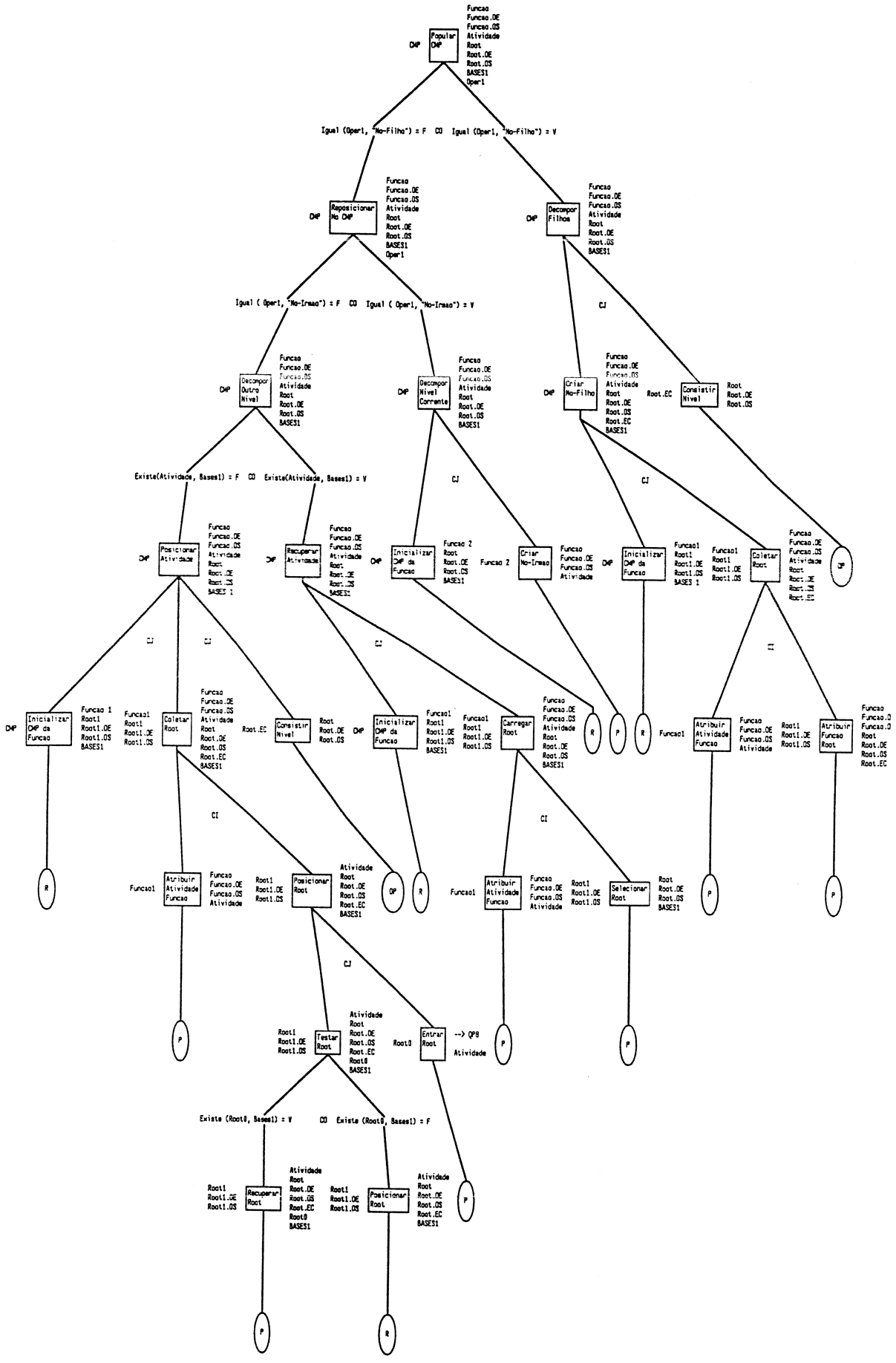


Figura 5.7: Especificação em HOS do CMO - Popular CMP

```
(@RULE= regra 37
        (@LHS=
                (Is     (Criar_Nos1)    ("sim"))
                (=      (\Funcao\.TF)    ("D"))
                (Is     (oper1) ("NO_FILHO"))
        )
        (@HYPO= DECOMPOR_FILHOS)
        (@RHS=
                (Do     (TRUE)   (Nivel_Concluido))
                (Reset  (CONSISTIR_NIVEL_H)
                (Do     (CONSISTIR_NIVEL_H)  (CONSISTIR_NIVEL_H))
                (Do     (FALSE)   (Nivel_Concluido))
                (Do     (df+1)       (df))
                (Do     (Funcao)     (root))
                (Do     (Funcao.chave)  (root.chave))
                (Do     (Funcao.pai)     (root.pai))
                (Do     (Funcao.OE)    (root.OE))
                (Do     (Funcao.OS)    (root.OS))
                (Let    ((Funcao.Ec)    ("I"))
                (Do     (Funcao.Ec)     (root.Ec))
                (Do     (Funcao.TF)     (root.TF))
                (Execute                ("GetRelatives")                @ATOMID=\Funcao\;
@STRING="@OBJECTS, @CLASSES, @CHILDREN, @ONELEVEL, @RETURN=
"Funcao.Filhos";\
))
                (Do     (Funcao.Filhos)     (root.Filhos))
                (Do     (root)    ((ROOT_TMP))
                (Do     (Funcao.Estado)   (Estados))
                (Do     (Funcao.Praticas_Controle)   (Praticas))
                (Do     (Funcao.Relatorios)   (Relat))
                (Execute ("ResetFrame")  (@ATOMID=Funcao;))
                (Do   (atividade)   (Funcao))
                (Let   (Ini_Vars)   ("sim")
                (Reset   (Inicializar_Vars_H))
                (Do (Inicializar_Vars_H)   (Inicializar_Vars_H))
                (Let      (Inicio)    ("sim"))
                (Reset    (Criar_Funcao_h))
        )
)
```

Figura 5.8: Exemplo de uma regra em Nexpert pertencente à implementação do CMO - Popular CMP

Note que a regra Decompor_Filhos infere outra regra, denominada Consistir_Nivel_H. Esta regra permite consistir os níveis hierárquicos do "Control Map" Instanciado. Portanto, a regra Consistir_Nivel_H infere as regras de consistência das estruturas de controle do HOS. A Figura 5.9 apresenta a regra Join_H que foi implementada para suportar as regras da estrutura de Controle JOIN.

```
(@RULE= regra 51
        (@LHS=
                (Is         (Est_Cont)          ("sim"))
                (Equal      (root_temp.OE)      (Funcao1.OE))
                (Equal      (Funcao1.OS)        (Funcao2.OE))
                (Equal      (Funcao2.OS)        (root_temp.OS))
```

```
)
(@HYPO= Join_H)
(@RHS=
        (Execute        ("SetMultiValue")        @ATOMID=\root\;
        @STRING="@ADD=Join,@DUPLICATE,@COMP=STRING";))
        (Do   (\root\.EC)   ((root.EC))
        (Let  (Est_Cont)  ("nao"))
    )
)
```

Figura 5.9: Exemplo de uma regra em Nexpert referente a uma estrutura de controle da metodologia HOS

As regras em Nexpert foram utilizadas para implementar as funções necessárias para suportar a decomposição segundo a metodologia HOS, isto é, as regras previstas pela metodologia, regras de produção que devem ser aplicadas a partir dos nós primitivos e as regras para mudanças de estratégia na decomposição quando necessário.

Até este ponto da implementação do Entrevistador, as regras permitem manusear as classes definidas, gerando a especificação em HOS, e aplicar as regras de produção para gerar o nível de representação. Porém, as questões estão baseadas na decomposição em HOS.

O próximo passo é identificar os pares pi=(qi,ri), isto é, o conjunto de questões e respostas que auxiliam na tradução dos requisitos do usuário em uma especificação em HOS e, consequentemente, na aplicação das regras de produção para gerar o nível de representação. Esta foi a segunda fase da implementação do Entrevistador.

5.3.1.2) Especificação e Implementação dos CMIs

A segunda fase da implementação do Entrevistador refere-se à especificação e implementação dos objetos do domínio. Durante esta especificação foram utilizados os CMOs já cadastrados. Portanto, nesta fase foram criados os "Control Maps" dos componentes do domínio, denominados "Control Maps" Instanciáveis - CMIs.

Para cada CMI foram levantados os pares Pi=(Qi,Ri), isto é, o conjunto de questões e respostas que auxiliam na aplicação das regras de produção para gerar o nível de representação.

O primeiro CMI especificado foi o "Control Map" de inicialização da entrevista, denominado Criar CMP Instanciado. Na verdade, tanto este CMI como o CMI do Finalizar apresentado no apêndice A, não correspondem aos objetos do domínio, mas são essenciais para o processo de entrevista. O CMI, apresentado na Figura 5.10(a), é o primeiro "Control Map" que deve ser instanciado para iniciar todo o processo de instanciação da entrevista. As Figuras 5.10(b) e 5.10(c) apresentam os "Control Maps" denominados Criar CMP da Função e Editar CMP da Função que são continuação do CMI Criar CMP Instanciado. As figuras 5.10(d) e 5.10(e) apresentam os CMOs denominados Decompor Função em Subfunções e Consultar Base <Nome-Base> que são utilizados na descrição do CMI da Figura 5.10(a).

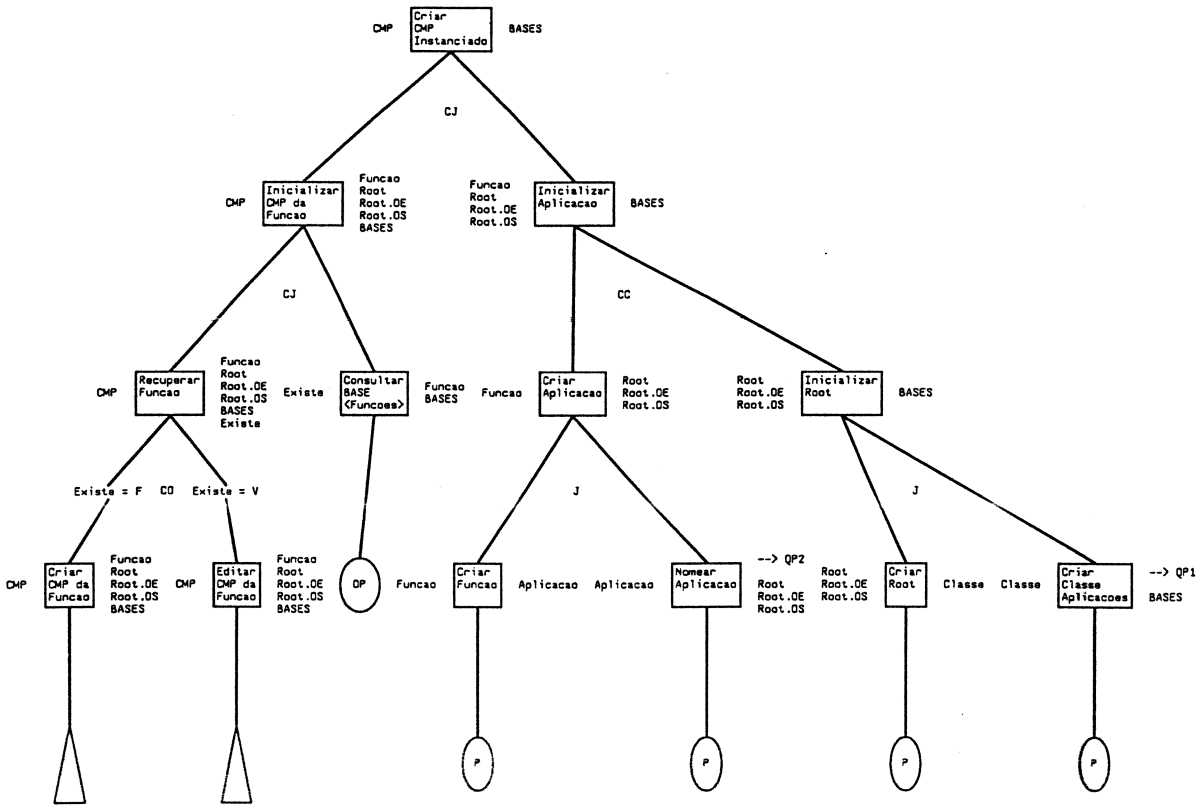


Figura 5.10 (a): CMI - Criar CMP Instanciado

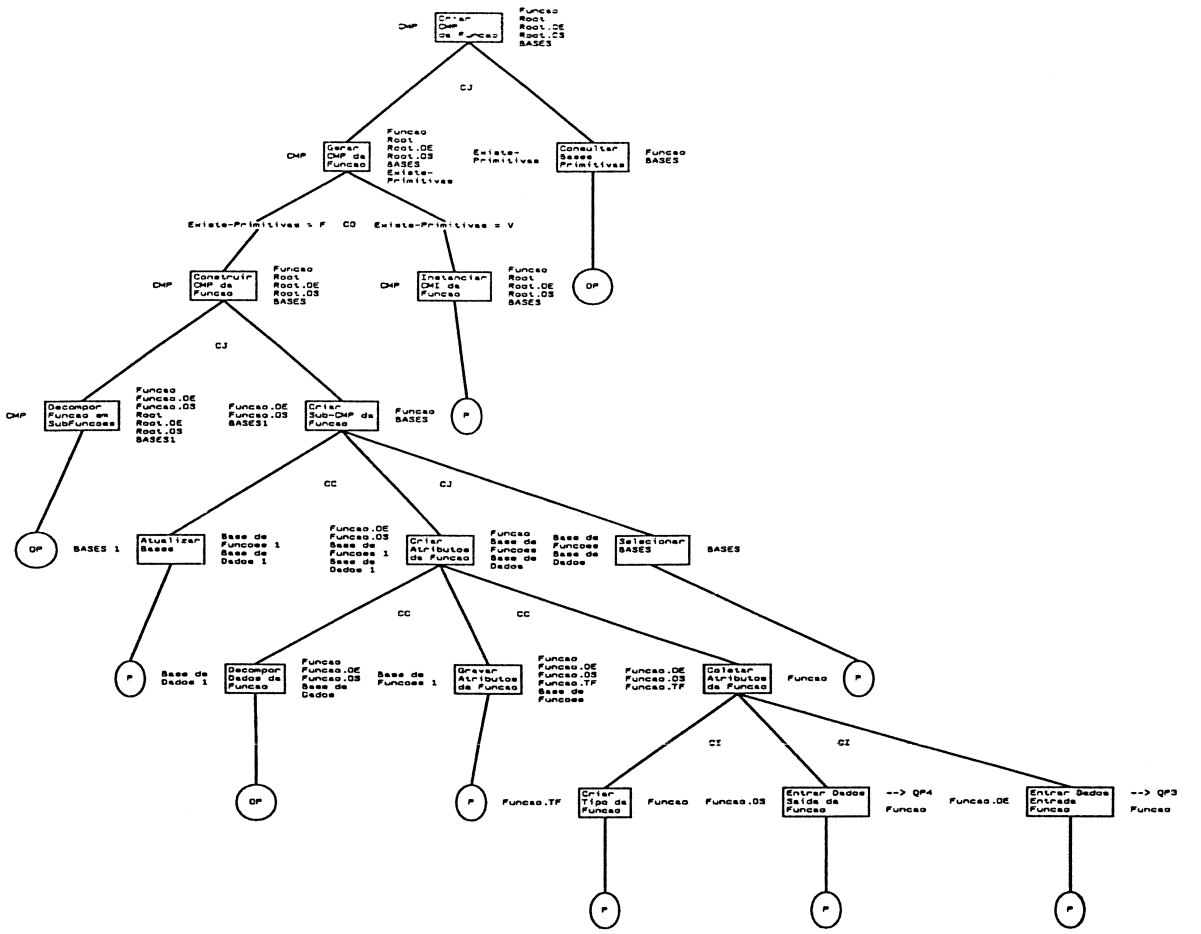


Figura 5.10(b): Criar CMP da Função (continuação do CMI Criar CMP Instanciado)

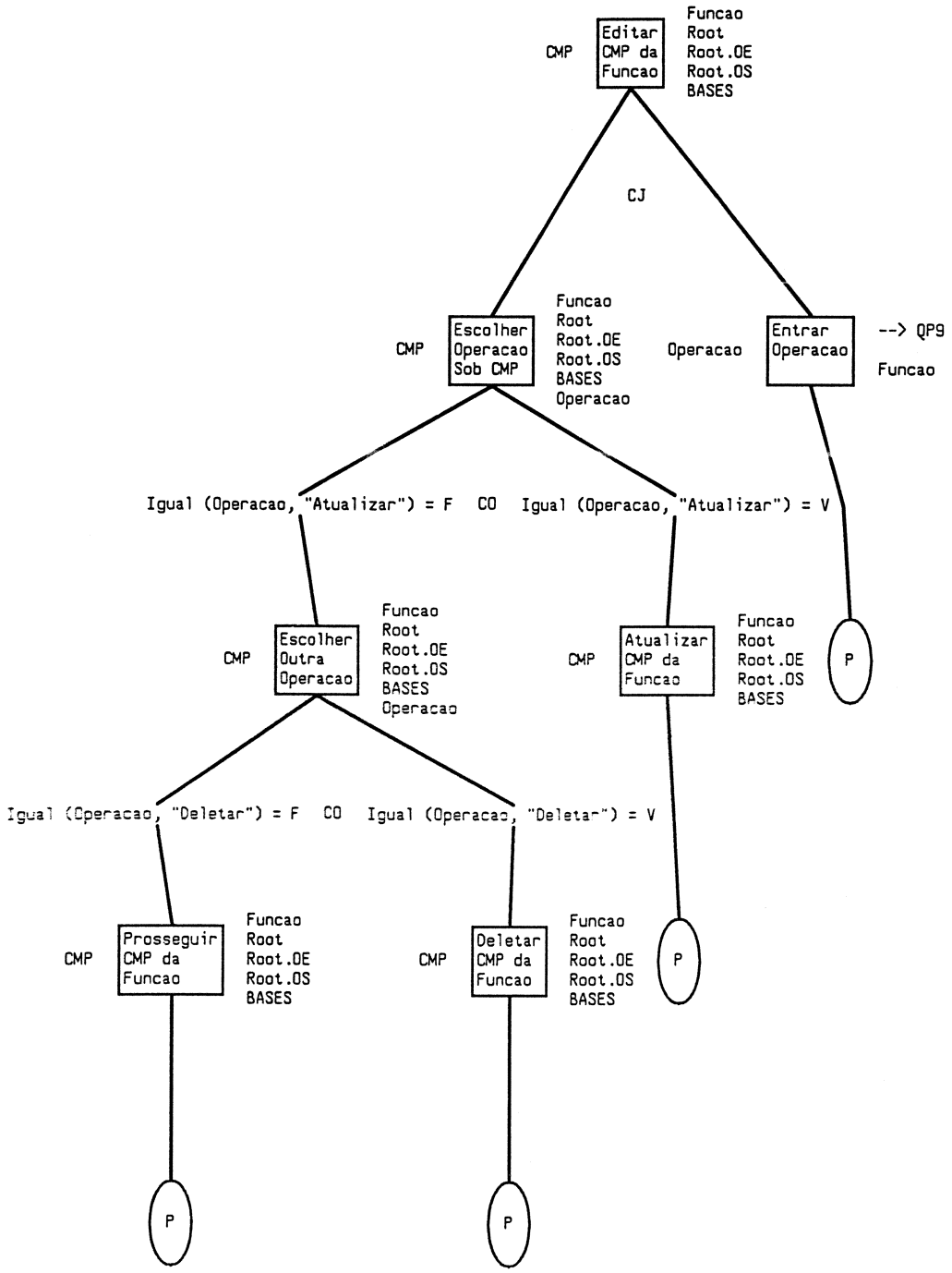


Figura 5.10(c): Editar CMP da Função (continuação do CMI Criar CMP Instanciado)

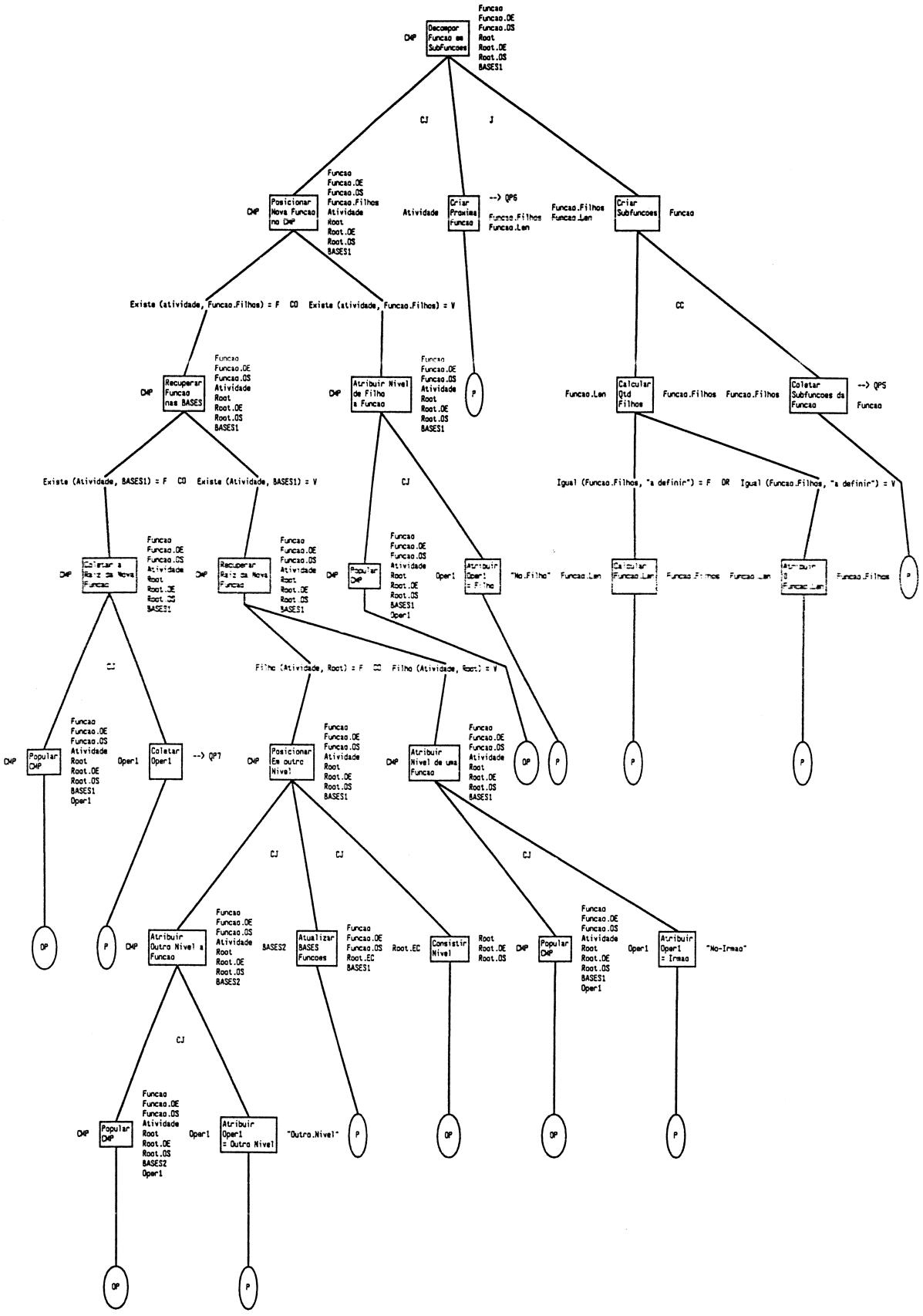


Figura 5.10(d): CMO - Decompor Função em Subfunções

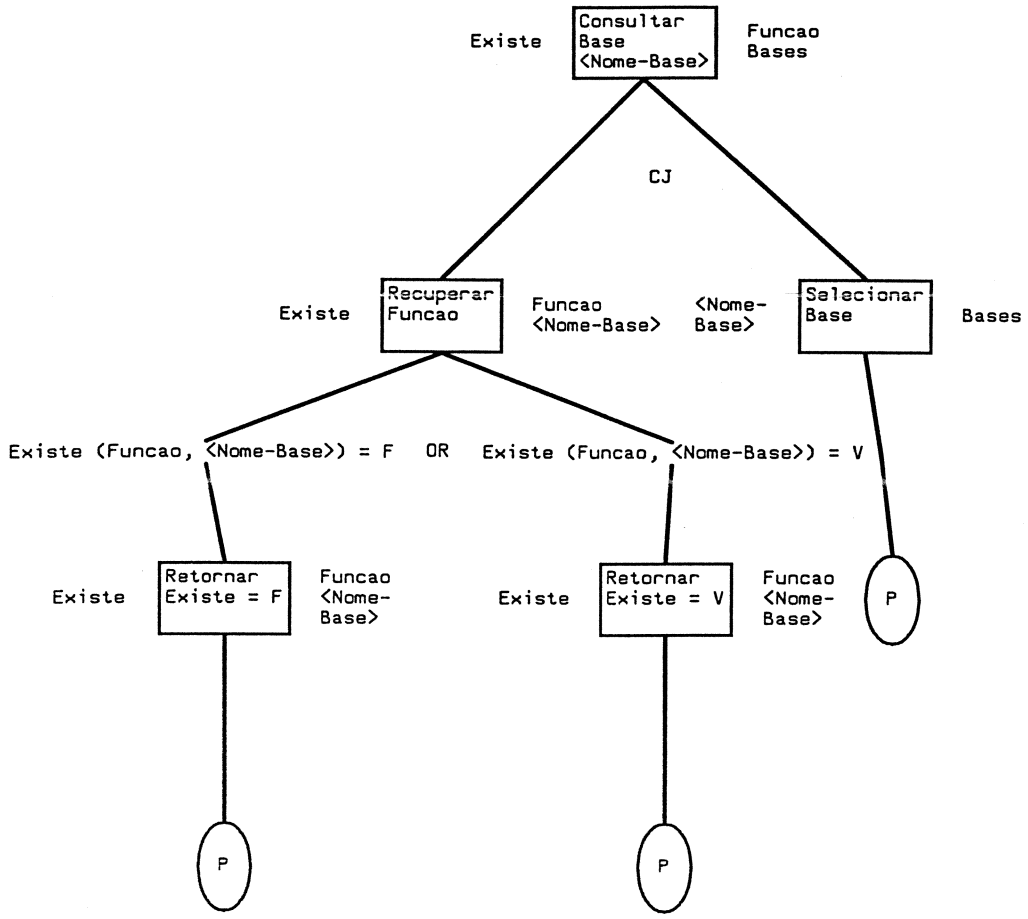


Figura 5.10(e): CMO - Consultar Base <Nome-Base>

Algumas questões definidas para o processo de entrevista do Entrevistador, durante a especificação dos CMIs e CMOs, estão descritas abaixo:

QP1) Você está no ambiente FMS. As classes de aplicações conhecidas estão descritas abaixo. Entre com a sua opção.
R: *Administração Rural* ←

QP2) Que aplicação deseja desenvolver?
R: <*F*>

QP3) Quais as informações necessárias para < *F*>?
R: <*x*>

QP4) O que resulta de < *F*>?
R: <*y*>

QP5) Descreva usando verbos e substantivos o que é <*F*>.
R: <*F1,F2*>

QP6) Você descreveu <*F*>. em <*F1,F2*>. Entre com a nova atividade que deseja descrever.
R: <*F1*>

QP9) <*F1*> já existe. escolha uma das opções abaixo:
R: *Atualizar*
Deletar
Prosseguir ←

QP10) Quais as práticas de controle de <*F1*>?
R: <*x2*>

QP11) Quais os relatórios necessários para <*F1*>?
R: <y2>

.
.
.

QP6) Você descreveu <*Fn*>. Entre com a nova atividade que deseja descrever.
R: *Finalizar*

.
.
.

As questões QP1, QP2, QP3, QP4 e QP9 correspondem ao CMI - Criar CMP Instanciado que está representado nas Figuras 5.10 (a), 5.10 (b) e 5.10 (c) e 5.10 (e). As questões QP5, QP6 correspondem ao CMO - Decompor Função em Subfunções que está apresentado na Figura 5.10 (d).

A Figura 5.11 apresenta o exemplo do CMI do "Controlar", que é central no domínio de aplicações de Administração Rural: a este CMI as questões QP10, QP11 estão vinculadas. As especificações dos outros CMIs e CMOs, com outras questões correspondentes, estão no Apêndice A.

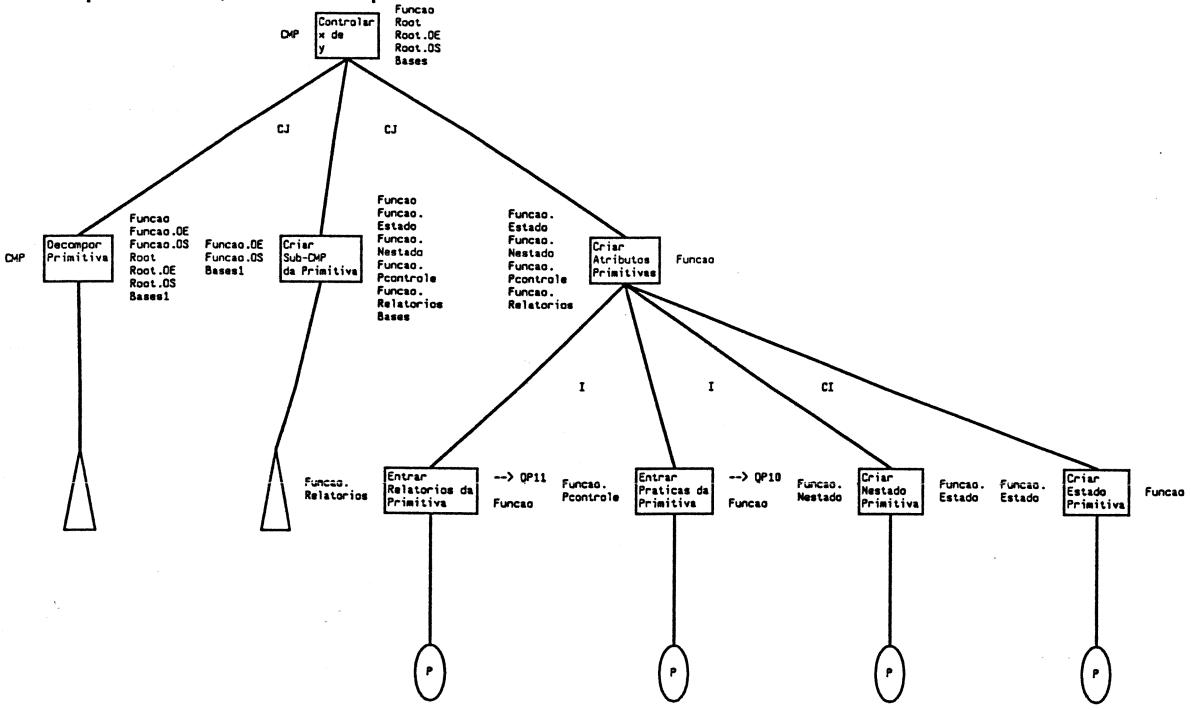


Figura 5.11: Exemplo do CMI do "Controlar"

Na sequência, foram implementadas algumas regras em Nexpert correspondentes a novos CMOs identificados durante a especificação dos CMIs. Por exemplo, as questões QP10 e QP11 que são específicas para o nó primitivo "Controlar" deram origem a uma regra Criar_No_H específica, como descrito na Figura 5.12.

```
(@RULE= regra 128
        (@LHS=
                (Is    (Criar_Prim1)    ("sim"))
                (Is    (dummy_funcao.dummy_cursor)    (0))
                (=     (primitiva.CodPrim)  (1))
        )
        (@HYPO= Criar_No_H)
        (@RHS=
                (Do    (1)   (testcont))
                (CreateObject  (\Funcao\)  (\ROOT_TMP))
                (Do    (Funcao.Estado)    (Funcao.Estado))
                (Do    (Funcao.Praticas_Controle)    (Funcao.Praticas_Controle))
                (Do    (Funcao.Novo_Estado)    (Funcao.Novo_Estado))
                (Do    (Funcao.Relatorios)    (Funcao.Relatorios))
                (Execute              ("SetMultiValue")              @ATOMID=\Funcao\.OE;
                @STRING="@ADD=@V(Funcao.Estado),@NODUPLICATE,\
                @COMP=STRING";))
                (Execute              ("SetMultiValue")              @ATOMID=\Funcao\.OE;
                @STRING="@ADD=@V(Funcao.Praticas_Controle),@NODUPLICATE,\
                @COMP=STRING";))
                (Execute              ("SetMultiValue")              @ATOMID=\Funcao\.OS;
                @STRING="@ADD=@V(Funcao.Novo_Estado),@NODUPLICATE,\
                @COMP=STRING";))
                (Execute              ("SetMultiValue")              @ATOMID=\Funcao\.OS;
                @STRING="@ADD=@V(Funcao.Relatorios),@NODUPLICATE,\
                @COMP=STRING";))
                (Do    (ROOT_TMP)    (Funcao.pai))
                (Do    (Funcao)      (Funcao.chave))
                (Do    (\Funcao\.OE)    (Funcao.OE))
                (Do    (\Funcao\.OS)    (Funcao.OS))
                (Let   ((Funcao.Ec)   ("I"))
                (Reset  (Decompor_Primitiva_H))
                (Do     (Decompor_Primitiva_H)  (Decompor_Primitiva_H))
                (Reset  (Criar_TF_H))
                (Do     (Criar_TF_H)  (Criar_TF_H))
                (Reset  (DD))
                (Reset  (Decompor_Dados_H))
                (Do     (Decompor_Dados_H)  (Decompor_Dados_H))
                (Reset  (Criar_Nivel_Representacao_H))
                (Do     (Criar_Nivel_Representacao_H) (Criar_Nivel_Representacao_H))
                ( Do    (1)   (nos))
                (Reset   (Criar_Nos_H))
                (Do      (Criar_Nos_H)  (Criar_Nos_H))
        )
)
```

Figura 5.12: Criar_No_H - Exemplo de regra em Nexpert

Observe que a quarta e a sexta linha, da parte de ações (RHS), correspondem às questões QP10,QP11. Estas questões capturam os objetos de entrada e saída do "Controlar".

Os CMIs devem ser especificados somente utilizando os CMOs já existentes. Para tal, os CMOs devem ser generalizados o máximo possível durante a atividade de análise do domínio. Assim, na incorporação de novos CMIs, a base de conhecimento estaria estável a tal ponto que não seria necessário a implementação em Nexpert de novos operadores. A implementação do Entrevistador está evoluindo neste sentido.

As regras do entrevistador, especificamente, estão na base denominada *Entrevistador.* Este arquivo, no formato ASCII, contém todas as classes/objetos e regras referentes ao Entrevistador.

Durante uma sessão de entrevista, esta base é carregada. As regras nela definidas manuseiam os objetos e classes para gerar a especificação em HOS. Durante este processo de inferência várias bases são consultadas e atualizadas. Estas bases compõem a base de conhecimento do AEsp, apresentada na Figura 3.4.

5.3.2) Base de conhecimento do AEsp

Nesta seção estão descritas as bases que compõem a base de conhecimento (Figura 3.4) manipulada pelo Entrevistador durante o processo de entrevista, e que posteriormente, é utilizada pelas outras ferramentas do AEsp.

Nas Figuras 5.10(a) e 5.10(b), pode-se observar as primeiras bases consultadas pelo Entrevistador. O Entrevistador consulta a *base de nós primitivos* toda vez que uma atividade é referenciada. Caso não seja encontrada, ele procura o sinônimo da atividade em uma *base de sinônimos.*. Estas bases estão no formato "flat file" e são denominadas *Primitivas* e *Sinônimos,* respectivamente.

As bases *Primitivas* e *Sinônimos* correspondem à classe Domínio da Tabela 4.1. A inserção de registros nestas bases é via o Catalogador, que será descrito na Seção 5.4.1.

Durante uma sessão do Entrevistador são atualizados os arquivos no formato "flat-file" correspondentes a outras classes definidas na Seção 5.3.1.1. Os arquivos *Funcoes.nxp* e *Dados.nxp* correspondem à classe Decomposição. Os arquivos *Forms.nxp, Campos.nxp, Textos.nxp, Transicoes.nxp, Relatorios.nxp* e *Eventos.nxp* correspondem à classe Modelagem da Aplicação. Estes últimos arquivos são lidos pelo Prototipador e Montador para simular o comportamento da aplicação e gerar a especificação em LC-FMS, respectivamente. A Figura 5.13 mostra um exemplo do formato destes arquivos. O exemplo dado é da base Funcoes.nxp.

Além das bases descritas acima, existem bases que são identificadas durante a especificação dos CMIs, denominadas bases de predicados. Como exemplo podemos citar a *base de coletivos*, denominada *Coletivos.nxp*, identificada na especificação do nó "Controlar". Estas bases aparecem relacionadas a um predicado do "Control Map" que está sendo processado. A criação destas bases é posterior à especificação, mas devem ser criadas automaticamente no momento da especificação via Catalogador.

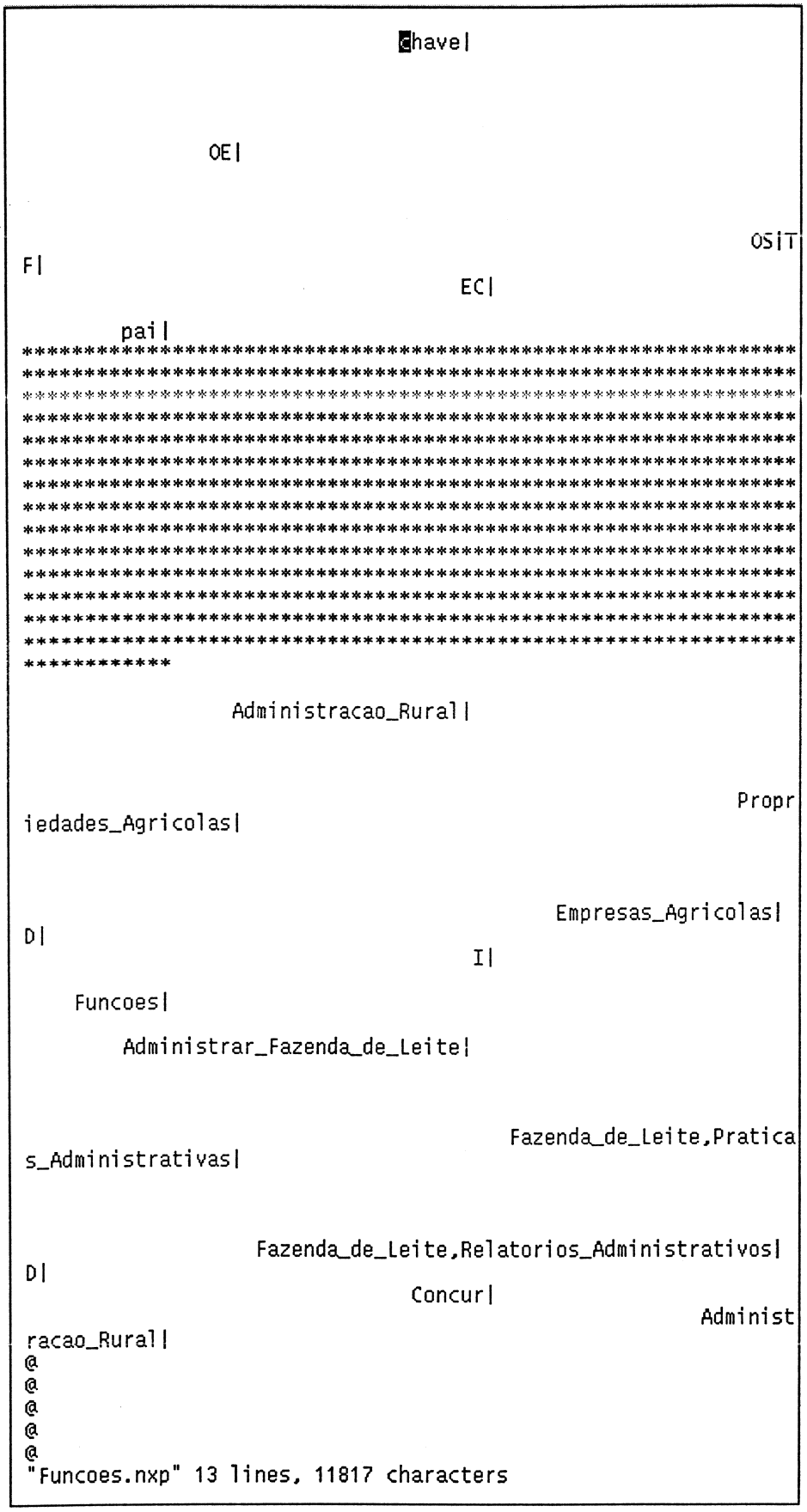


Figura 5.13: Características da base Funções

5.4) Outras Ferramentas do AEsp

5.4.1) Catalogador

Na análise do domínio das aplicações do domínio de Administração Rural, foram identificados alguns nós primitivos, tais como: Planejar, Controlar, Relatar e Finalizar. Estes nós primitivos, exceto "Planejar", foram cadastrados no Catalogador de Componentes, que será descrito nesta seção.

O *Catalogador* é uma ferramenta para auxiliar no cadastramentos dos componentes do domínio. Esta ferramenta implementa duas funções básicas (Figura 5.14):

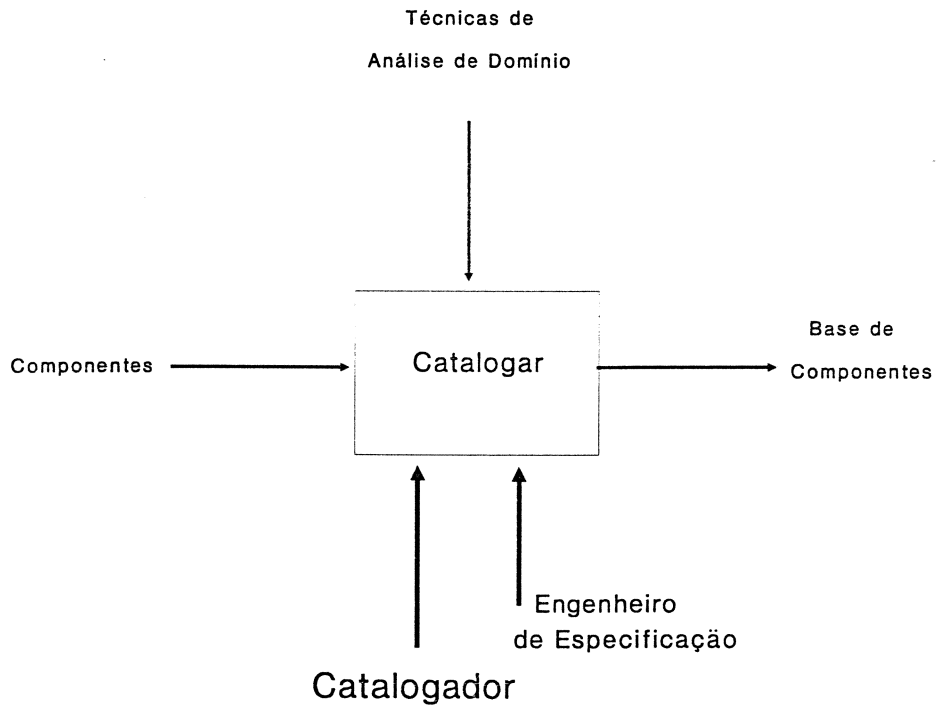


Figura 5.14: Catalogador

a) salvar parte de uma sessão da entrevista que o Engenheiro de Especificação perceba ser potencialmente reutilizável - normalmente são "Control Maps".

b) incluir componentes instanciáveis, que são a forma básica de reutilização de informações do domínio.

A entrevista é, em última análise, o conjunto de perguntas e respostas que permite instanciar esses componentes para o "Control Map" específico da aplicação do usuário. As técnicas de reanálise do domínio, que permitem examinar um conjunto de componentes específicos e transformá-los em um componente instanciável, ainda não estão formalizadas. Componentes instanciáveis podem ser identificados tanto no nível da aplicação quanto no nível de representação.

5.4.1.1) Implementação do Catalogador

O catalogador também foi implementado em Nexpert. A estrutura do Catalogador é basicamente o cadastramento da base de nós primitivos, isto é, da base de componentes e base de sinônimos. A tela inicial tem as seguintes opções:

1. Base de Componentes
2. Base de Sinônimos
3. Base de Coletivos

Para as duas bases são permitidas as funções básicas de cadastramento:

1. Inserir
2. Deletar
3. Alterar
4. Consultar

Essas bases estão no formato "flat-file" conforme é mostrado na Figura 5.15.

```
                          chave|        CodPrim|                          Temp
late|
*************************************************************************
*****
                      Controlar|             1|          Controlar <x> de
 <y>|
                       Imprimir|             2|           Imprimir <x> de
 <y>|
                      Finalizar|             3|                       Final
izar|
*************************************************************************
*****
```

```
                          chave|                          sinonimo|
*****************************************************************
                      Gerenciar|                        Controlar|
*****************************************************************
~
~
~
~
```

Figura: 5.15: Primitivas e Sinônimos

Quando é inserido um componente na base de nós primitivos, é permitido entrar com o "Control Map" específico daquele componente. Atualmente, após este cadastramento, são implementadas as regras correspondentes a novos operadores identificados na especificação. Os trabalhos futuros, na implementação do Catalogador, serão na incorporação automática dos "Control Map" Instanciáveis. Os "CMIs já cadastrados estão especificados no Apêndice A.

5.4.2) Reutilizador

O Reutilizador, apresentado na Figura 5.16, é uma ferramenta que permite usar as informações já armazenadas na base de conhecimento durante o desenvolvimento de aplicações anteriores para a especificação de uma nova aplicação do domínio.

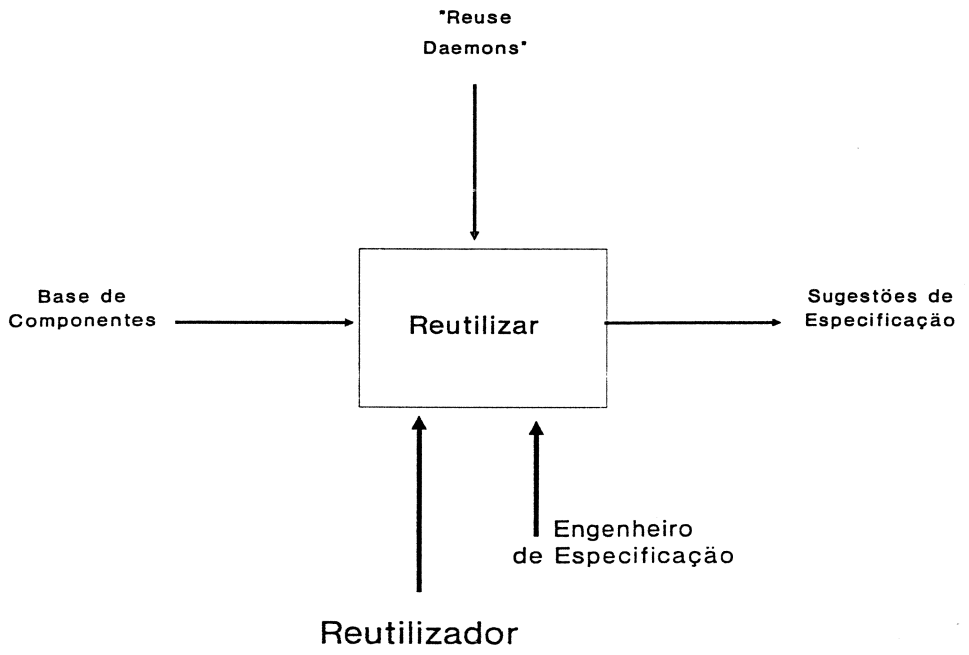


Figura 5.16: Reutilizador

A base de conhecimento mantida pelo Catalogador é continuamente varrida durante o transcorrer da entrevista pelo que se convencionou chamar de "reuse daemons". Estes processos são disparados sempre que:

a) o usuário fornece uma ação expressa por um verbo que, colocado na forma de infinitivo pelo engenheiro de especificação, fornece a chave de busca na base de componentes;

b) o usuário fornece um objeto da ação que conste em uma das classes existentes.

Nestas ocasiões é disparada a instanciação dos componentes envolvidos (a maior parte das vezes pela troca dos identificadores formais) e oferecidas as possíveis sugestões de especificações previamente armazenadas.

Esta abordagem de reuso é aderente ao domínio agropecuário baseado nas seguintes premissas:

a) propriedades com a mesma atividade de produção têm necessidade de informações muito semelhantes, mas não iguais;

b) processos gerenciais são particulares do proprietário e devem ser incorporados na aplicação.

5.4.2.1) Implementação do Reutilizador

Na atual implementação, os processos disparados pelo Reutilizador para sugerir as especificações são via Entrevistador ou Prototipador.

As funções de reuso até agora implementadas na estrutura do Entrevistador são funções que disparam a instanciação dos componentes instanciáveis ou funções que permitem recuperar os "Control Map" Instanciados.

O prototipador permite que sejam reusadas as especificações de aplicações desenvolvidas anteriormente, apresentando telas da aplicação já definidas via o Prototipador.

Desta forma, o botão correspondente ao Reutilizador no AEsp (Figura 5.2) ainda não está disponível. Para tal, são necessários estudos e pesquisas nesta área de reuso para otimização do AEsp e implementação do Reutilizador, efetivamente.

5.4.3) Prototipador

O *Prototipador* é uma ferramenta para auxiliar na especificação da aplicação permitindo a validação incremental da aplicação durante a entrevista. O prototipador do FMS, mostrado na Figura 5.17, tem como funções básicas:

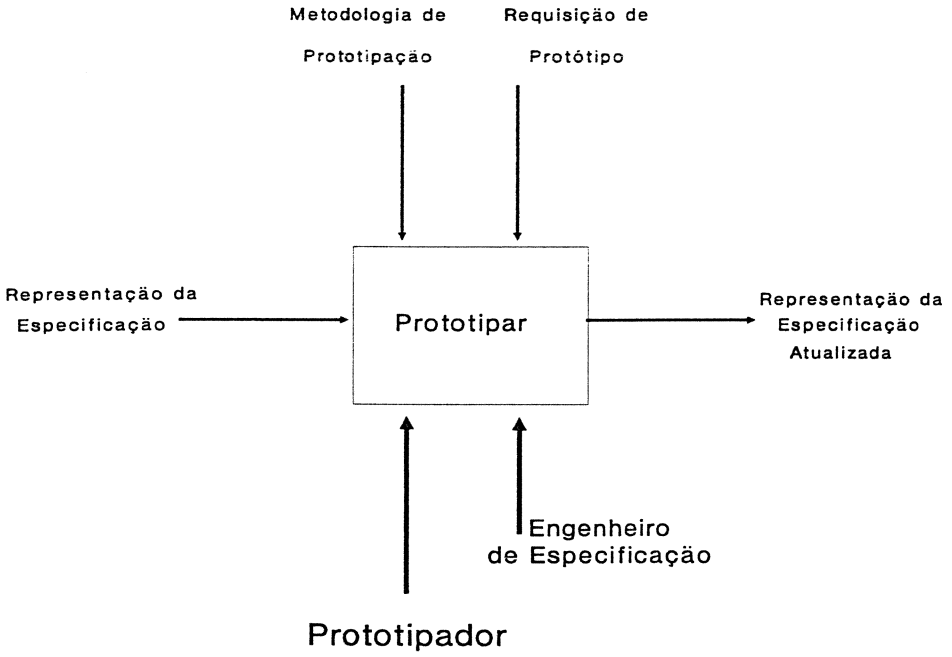


Figura 5.17: Prototipador

a) suportar a metodologia de prototipação incremental;
b) ajuste da interface com o usuário;
c) interface com o montador.

O Prototipador pode ser dividido em duas partes: Editor de formulários e Simulador de ações.

O Editor de Formulários é uma ferramenta onde podem ser desenhadas as telas da aplicação. O Editor recupera as informações da base de conhecimento e gera as telas correspondentes. A partir daí, com apoio de uma interface amigável, o usuário pode alterar as telas da aplicação da maneira desejada.

O prototipador permite também, a partir das informações da base de conhecimento, uma simulação do comportamento da aplicação. O usuário pode alterar substancialmente a

interface, não apenas no que concerne o valor dos atributos e posicionamento dos campos das telas, mas também no encadeamento das telas e ações. Todas as mudanças no protótipo são incorporadas à base de conhecimento original.

Após a validação da interface pelo usuário pode ser requisitada uma montagem da aplicação ao Montador para completar o protótipo e transmití-lo a um microcomputador IBM PC compatível, que pode mostrar o comportamento completo da aplicação.

5.4.3.1) Implementação do Prototipador

Na implementação corrente da ferramenta [PAS94] está sendo utilizado o SILK - Encap (gerador de interfaces gráficas para padrão OpenLook) [SIL93]. Na Figura 5.18 é mostrada a interface do Prototipador.

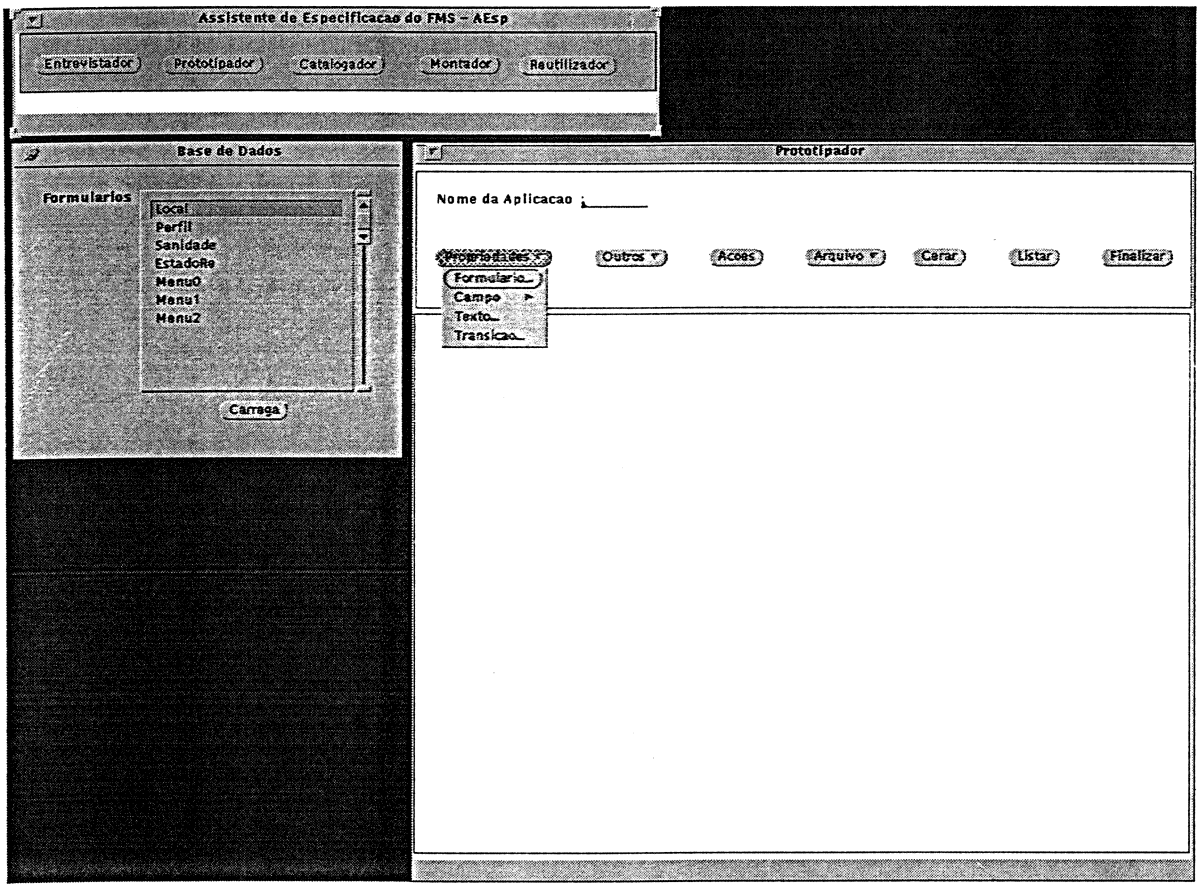


Figura 5.18: Interface do Prototipador

Dividiu-se essa interface em duas regiões: área de controle e área de visualização. A área de controle é área dos menus: menus de propriedades (formulários, campos, relatórios, textos, transições e eventos) e os menus para carregar e salvar um formulário da base. A área de visualização é onde o usuário pode visualizar as telas (formulários) da aplicação e manuseá-las.

Quando um usuário seleciona um formulário ou campo deste formulário, ele pode visualizar as propriedades referentes ao formulário ou propriedades dos campos, textos ou transições do formulário.

A Figura 5.19 mostra um exemplo da interface dos painéis de propriedades de campos e a Figura 5.20 mostra um exemplo da interface dos painéis de propriedades de formulários.

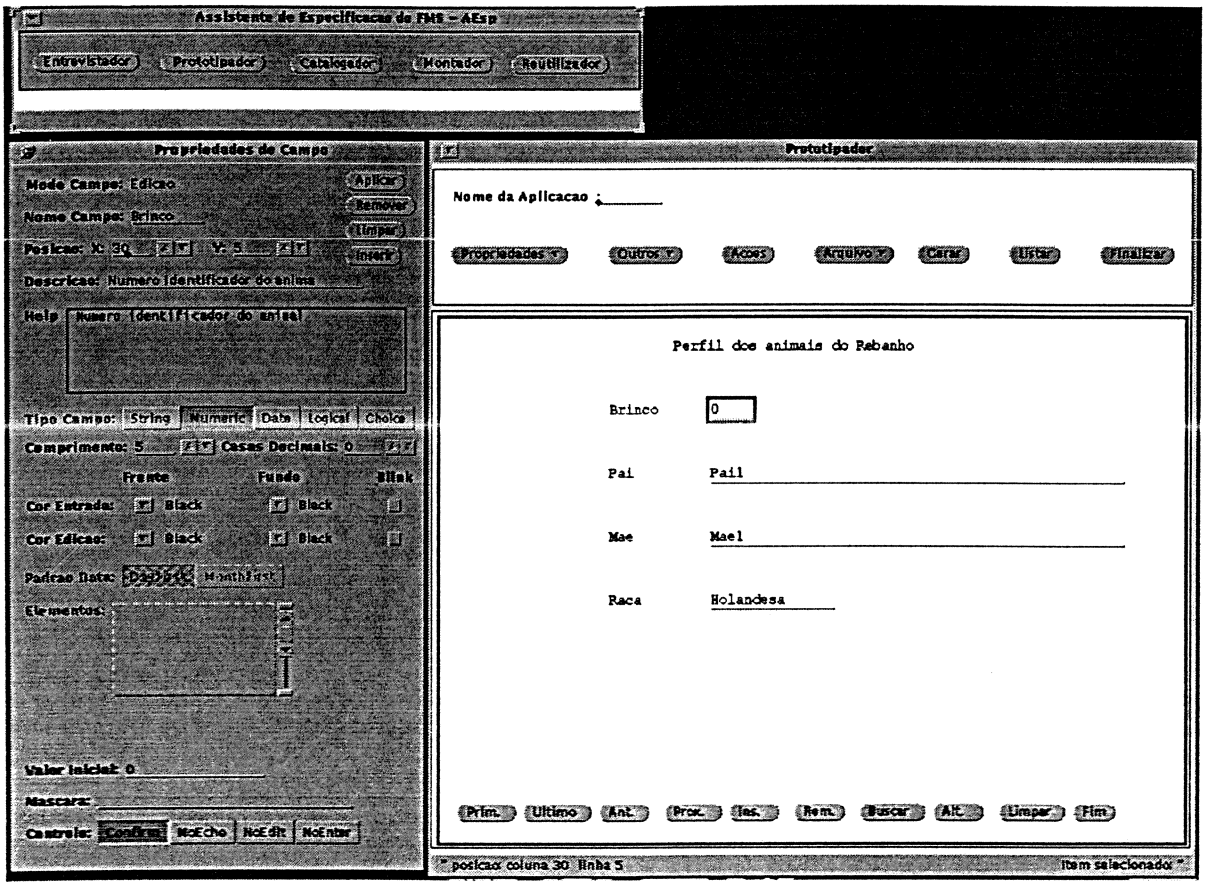


Figura 5.19: Propriedades de Campos

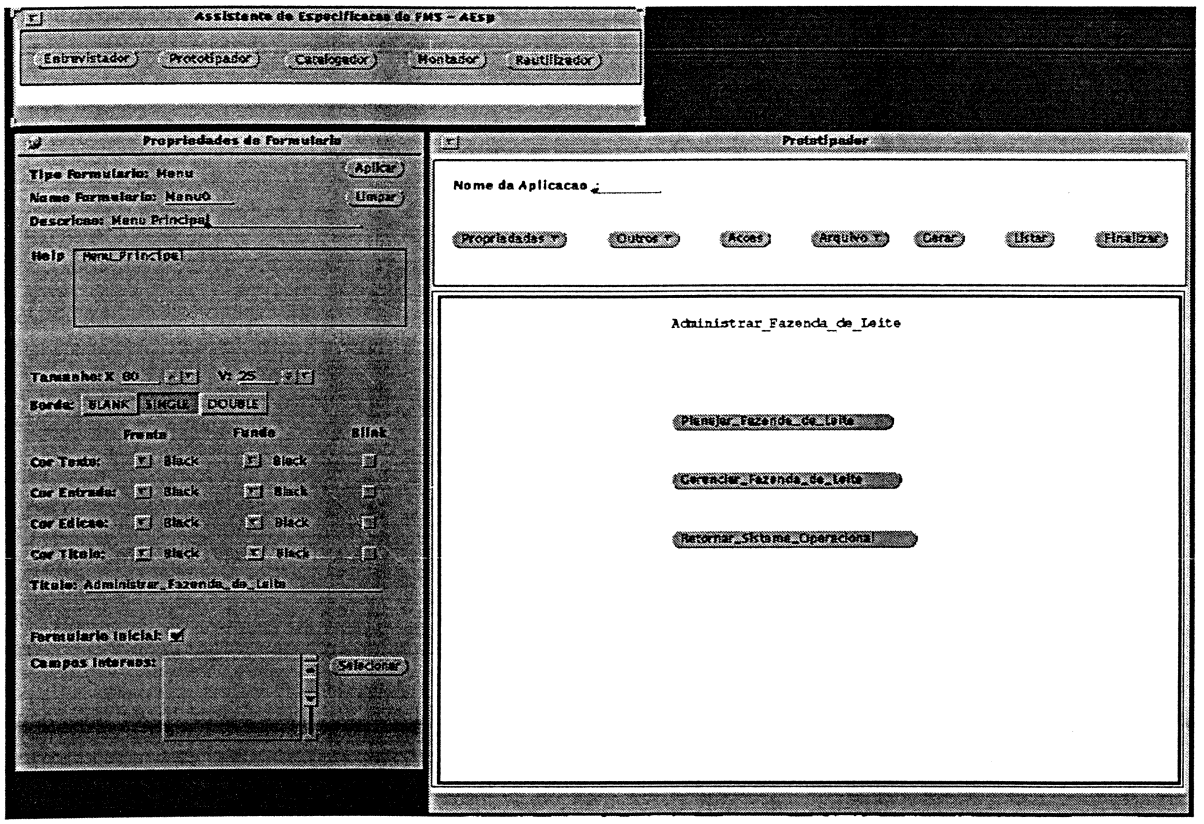


Figura 5.20: Propriedades de Formulários

Note que as propriedades de formulários e campos são as mesmas definidas nas bases Forms.nxp e Campos.nxp, geradas pelo Entrevistador.

O Prototipador, na versão corrente, não suporta o protótipo completo da aplicação devido à plataforma de hardware do AEsp ser diferente da plataforma de hardware do sistema alvo. O Prototipador apenas gera a interface da aplicação que pode ser refinada. Após a validação da interface pelo usuário, pode ser requisitado uma montagem da aplicação ao Montador para completar o protótipo.

5.4.4) Montador

O **Montador**, a partir da varredura da árvore gerada pelo Entrevistador e da base de conhecimento correlata, gera o programa descrito em LC-FMS que será a entrada para o GFMS gerar a aplicação do usuário, como é mostrado na Figura 5.21.

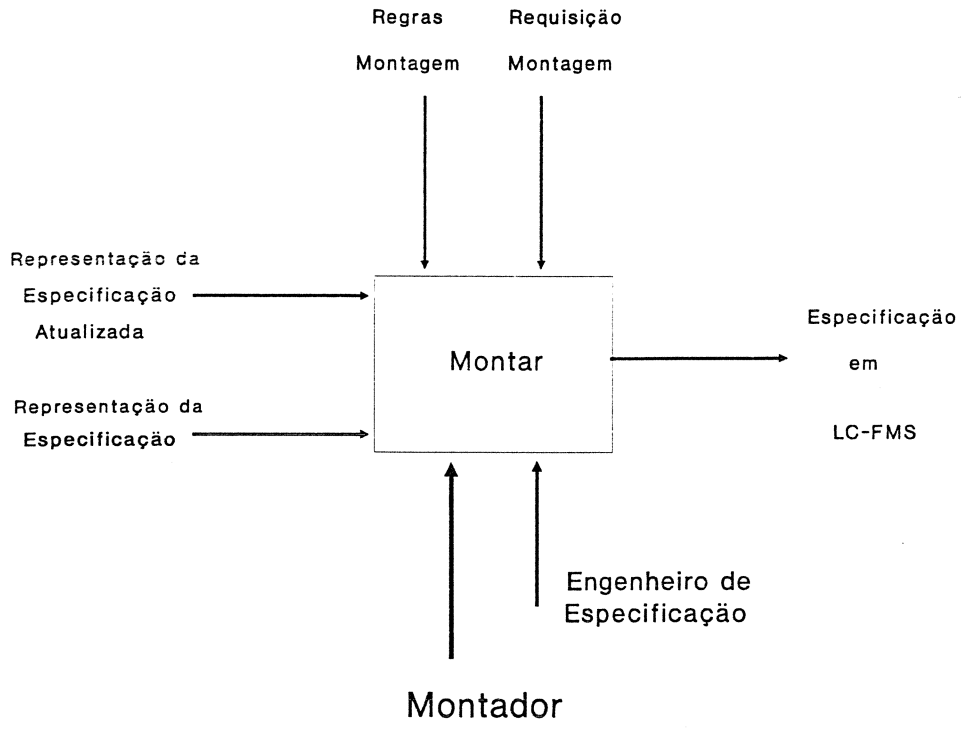


Figura 5.21 : Montador

A LC-FMS, como já descrito anteriormente, é composta por várias sublinguagens: sublinguagem de especificação de formulários, sublinguagem de especificação de base de dados, sublinguagem de especificação de consistências, sublinguagem de especificação de ações, helps e relatórios. A sintaxe e semântica destas linguagens estão definidas em [MEI92].

Como um exemplo da LC-FMS, abaixo é dada a especificação de duas telas de um Sistema de Controle de Rebanho Leiteiro (SISCOREB) que está sendo desenvolvido para validar os conceitos do AEsp.

Exemplo da especificação de uma tela do tipo menu (NOEDIT):

```
%%EXT_FORM
ATTRIBUTE = Noedit

%%FORM
!==================================================
! MENU0.CUF
!==================================================
! Definicao do formulario do menu principal
! Historico: Agosto - 1995
!==================================================
COLOUR     = White/Black
INPUT      = White/Cyan
TCOLOUR    = White/Black
EDITCOLOUR = White/Cyan
TYPE   = DOUBLE
SIZE   = 46,12
TITLE = "Menu Principal"

%%TRANSITIONS
INITIAL

BUTTON BUT_1 TO menu_reb; POSITION = @6,3; &
" Controle do Rebanho          -> "
BUTTON BUT_2 TO menu_pro; POSITION = @6,5; &
" Controle da Producao de Leite -> "
BUTTON BUT_3 TO menu_cfg; POSITION = @6,7; &
" Configuracao do Sistema      -> "
BUTTON BUT_4 EXIT; POSITION = @6,9; &
" Fim                          "
```

Exemplo da especificação de uma tela para edição dos parâmetros do evento cadastramento do rebanho:

```
%%EXT_FORM
ATTRIBUTE = Ev_Entry
SOURCE = Scheduled

%%FORM
!==================================================
! CADAST.CUF
!==================================================
! Definicao do formulario para a edicao dos parametros do evento cadastramento
! Historico: Agosto 1995
!==================================================
COLOUR     = White /Black
INPUT      = White/Cyan
TCOLOUR    = White/Black
EDITCOLOUR = White/Cyan

TYPE   = DOUBLE
SIZE   = 80,25
TITLE = "Cadastramento"

DATABASE_FIELD; NAME=EV_ID; STRING(15)
TEXT  = @3,2;  "BRINCO.....:"
TEXT  = @3,3;  "NOME COMPLETO.....:"
TEXT  = @3,5;  "DATA DE NASCIMENTO.....:"
TEXT  = @3,6;  "PAI.....:"
```

```
TEXT  = @3,7;  "MAE.....:"
TEXT  = @3,8;  "RACA.....:"
TEXT  = @3,9;  "GRAU DE SANGUE.....:"
TEXT  = @44,9; COLOUR=White/Cyan; ":"
TEXT  = @3,10; "REGISTRO GENEALOGICO.....:"
TEXT  = @3,11; "CATEGORIA.....:"
FIELD = @39,2; NAME=BRINCO; STRING(5)
FIELD = @27,3; NAME=NOME; STRING(50)
FIELD = @39,5; NAME=DTA_NASC; DATE
FIELD = @27,6; NAME=PAI; STRING(50)
FIELD = @27,7; NAME=MAE; STRING(50)
FIELD = @39,8; NAME=RACA; &
CHOICE(12) = "Holandesa PB|Holandesa VB|Jersey|Pardo
Suica|Simental|Gir|Caracu|Guzera|Girolanda|Guernsey|Mestica"
FIELD = @39,9; NAME=GSANGUE; &
CHOICE(5) = "1/2|3/4|7/8|15/16|31/32|GC-1|GC-2|GC-3|GC-4|GC-5|GC-6|GC-7|GC-8|GC-9|GC-
10|GHB|PO|POI"
FIELD = @45,9;  NAME=GSANGUEC; STRING(32)
FIELD = @39,10; NAME=REGISTRO; STRING(15)
FIELD = @39,11; NAME=TIPO_ANI; &
CHOICE(7)="Bezerra|Novilha|Vaca|Bezerro|Garrote|Touro|Rufiao"

TEXT  = @3,12;  "ESTADO PRODUTIVO.....:"
TEXT  = @3,13;  "NUMERO DE LACTACOES.....:"
TEXT  = @3,14;  "DATA DE INICIO DA ULTIMA LACTACAO.:"
TEXT  = @3,15;  "ESTADO REPRODUTIVO.....:"
TEXT  = @3,17;  "DATA DA ULTIMA COBERTURA.....:"
TEXT  = @3,18;  "TOURO.....:"
FIELD = @39,12; NAME=EST_PROD; CHOICE(13) = "Em Lactacao|Seca|Nao se aplica"
FIELD = @39,13; NAME=LACTACAO; NUMERIC(2)
FIELD = @39,14; NAME=DTA_LACT; DATE
FIELD = @39,15; NAME=EST_REPR; &
CHOICE(12)="Nao Coberta|Coberta|Vazia|Prenhe|Nao Servindo|Servindo"
FIELD = @39,17; NAME=DTA_COBT; DATE
FIELD = @27,18; NAME=TOURO; STRING(20); &
TEMPLATE="!!!!!!!!!!!!!!!!!!!"

TEXT  = @3,20;  "DATA DO CADASTRAMENTO.....:"
TEXT  = @3,21; "COMENTARIO:"
FIELD = @39,20; NAME=DATA_EV; DATE
FIELD = @15,21; NAME=COMENT0;STRING(62)
FIELD = @3,22;  NAME=COMENT;STRING(74)

! Define os buttons para forms de entrada de eventos
! -------------------------------------------------
BUTTON = @3,23; BUT_1 = "CONFIRMAR"; COLOUR=WHITE/BLACK
BUTTON = @13,23; BUT_2 = "FIM"; COLOUR=WHITE/BLACK

%%SCHEMA
#FILE CADAST
#KEY EV_ID
#KEY NRO_PART
```

A forma LC-FMS, textual na atual implementação, permite tradução automática para outras especificações híbridas ("resources" do Windows ou "widgets" do Xview).

5.4.4.1) Implementação do Montador

Na atual implementação do Montador [PAS94] foram desenvolvidas, em linguagem C, as funções necessárias para geração da sublinguagem de formulários (formulários, campos, textos e transições). Estas funções podem ser classificadas como: funções de varredura, recuperação, especificação e geração.

Basicamente, estas funções manipulam as bases geradas pelo Entrevistador e geram os arquivos, também no formato ASCII, com a descrição textual das sublinguagens. Estes arquivos, contendo a especificação em LC-FMS, são as entradas para o GFMS.

Na atual implementação do Montador e do Prototipador estão sendo desenvolvidas funções de conversão do formato. nxp para o formato dbf pois estas ferramentas necessitam de um gerenciador de base de dados para implementar suas funcionalidades.

5.5) Plataforma de Suporte Utilizada

A implementação do AEsp exigiu a integração de geradores de interfaces, bases de conhecimento, base de objetos, gerenciadores de base de dados, técnicas de análise de domínio e reuso.

Desta forma, a plataforma de software definida para implementar o AEsp foi:

- Nexpert [NEX91]: funciona como administrador dos dados do AEsp e tem como características principais:
  - Ferramenta baseada em conhecimento com um gerenciador de objetos e classes com estrutura adequada para armazenar componentes do domínio ;
  - Máquina de inferência de complexidade suficiente para exprimir os "reuse daemons";
  - Interface com Linguagem C.
- Silk [SIL93]: gerador de interfaces gráficas para openwindows com recursos de "Link dinâmico".
- Codebase [COD92]: Sistema gerenciador de base de dados (formato .dbf) utilizado na importação dos dados dos arquivos "flat-file" do Nexpert para facilitar a implementação do Prototipador e Montador.
- Linguagem C.

A plataforma de Hardware para desenvolvimento do AEsp, na versão corrente, é a estação de trabalho (SUN). Entretanto a aplicação final é gerada no ambiente PC para ser mais acessível aos possíveis usuários do FMS.

5.6) Considerações sobre o Capítulo

Neste capítulo foi descrita a implementação utilizada no AEsp, baseada na metodologia descrita no capítulo 4, para captura, documentação e tradução para o nível operacional das especificações das aplicações do domínio de Administração Rural. No capítulo 6 será dado um exemplo de uma especificação sob o AEsp.

CAPÍTULO 6

UM EXEMPLO DE UMA ESPECIFICAÇÃO SOB O AEsp

6.1) Introdução

Neste capítulo está descrito um exemplo de uma especificação sob o AEsp. O exemplo dado é a especificação de uma aplicação para Administrar Fazenda de Leite. Está baseado em uma pequena parte do Sistema de Controle de Rebanho Leiteiro - SisCoReb, que está sendo desenvolvido para validar os conceitos do ambiente FMS. É um exemplo simplificado e extraído do SisCoReb para ilustrar o ciclo de uma especificação sob o AEsp.

6.2) Entrevistador e sua operação

Nesta seção é apresentado o comportamento do Entrevistador. Esta deve ser a primeira ferramenta a ser diparada para iniciar a especificação de uma aplicação sob o AEsp (Figura 6.1). A execução do Entrevistador envolve as bases mantidas pelo Catalogador e funções disparadas pelo Reutilizador.

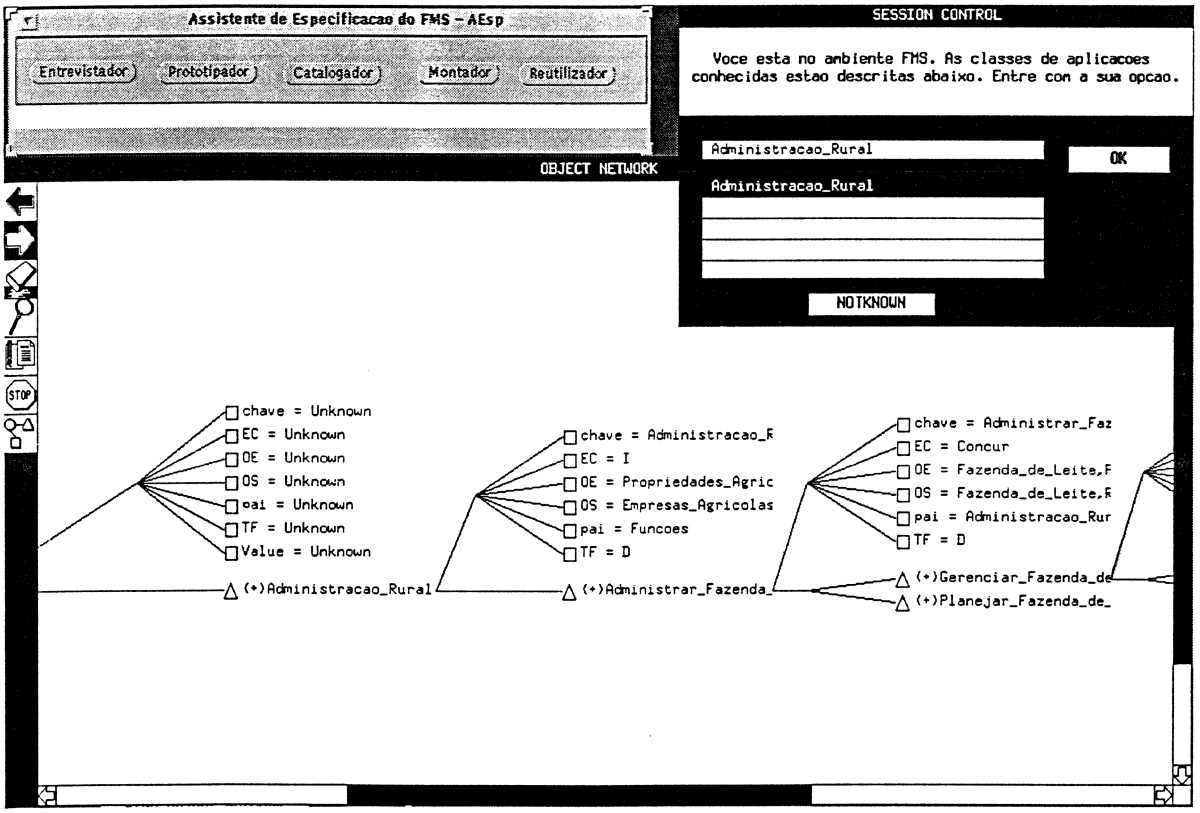


Figura 6.1: Tela Inicial

Primeiramente, ao clicar o botão do Entrevistador, é carregado o Nexpert. Na sequência , utilizando o shell do Nexpert, é carregada a base de conhecimento referente

ao Entrevistador, denominada Entrevistador. Ao iniciar a sessão, aparece uma janela com a pergunta inicial para o processo de entrevista (Figura 6.1). A partir da resposta aparece uma nova pergunta e assim sucessivamente.

A Figura 6.2 apresenta a especificação em HOS do CMI - Criar_CMP_Instanciado que mostra como o Entrevistador opera. Este é o primeiro "Control Map" a ser instanciado.

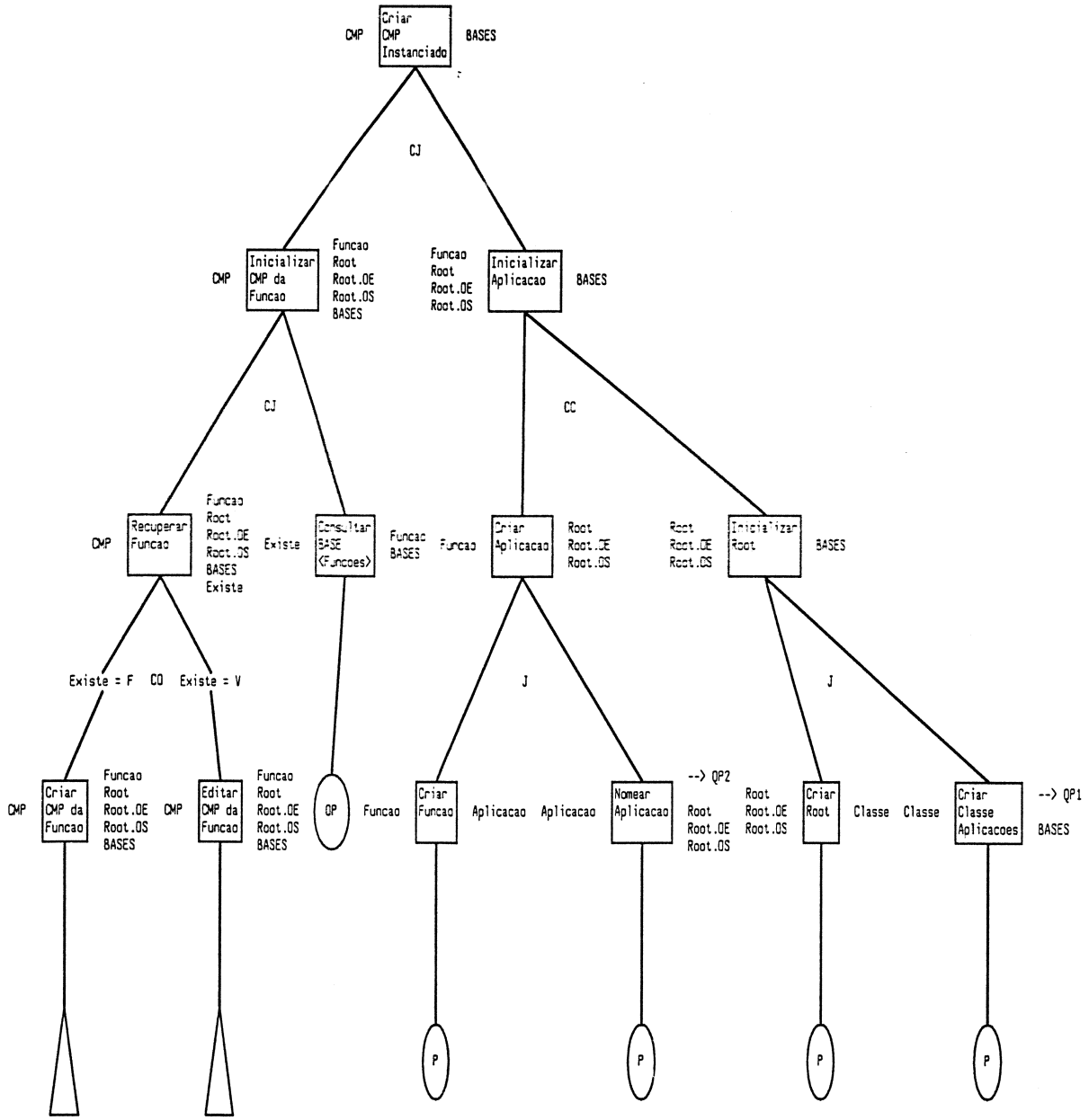


Figura 6.2: CMI - Criar_CMP_Instanciado

O Entrevistador, para realizar o processo de entrevista, considera cada função descrita pelo Engenheiro, na resposta às suas questões, como uma atividade. Toda vez que uma atividade é citada, o Entrevistador consulta a base Funções.nxp para ver se a função já existe (figura 6.2). Caso exista, ele executa o nó Editar CMP da Função, mostrado na figura 6.3 (a). A instanciação deste nó apresenta as opções de Atualizar, Deletar ou Prosseguir a entrevista, conforme Figura 6.3 (b).

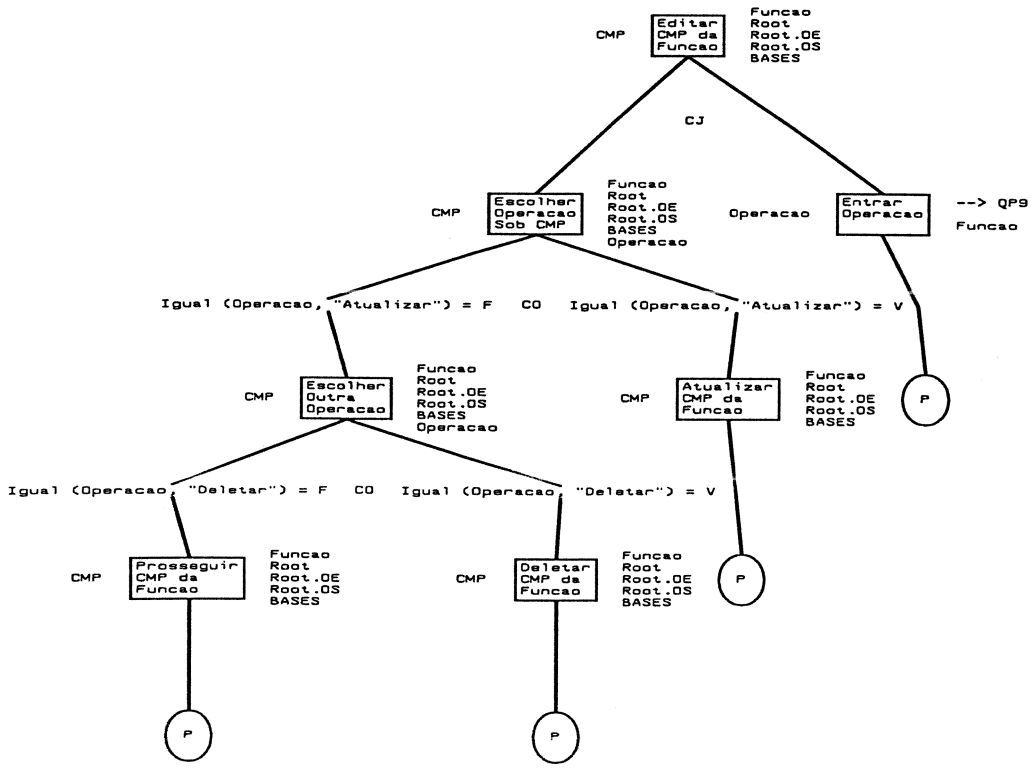


Figura 6.3 (a): Editar CMP da Função (pertence ao CMI Criar_CMP_Instanciado)

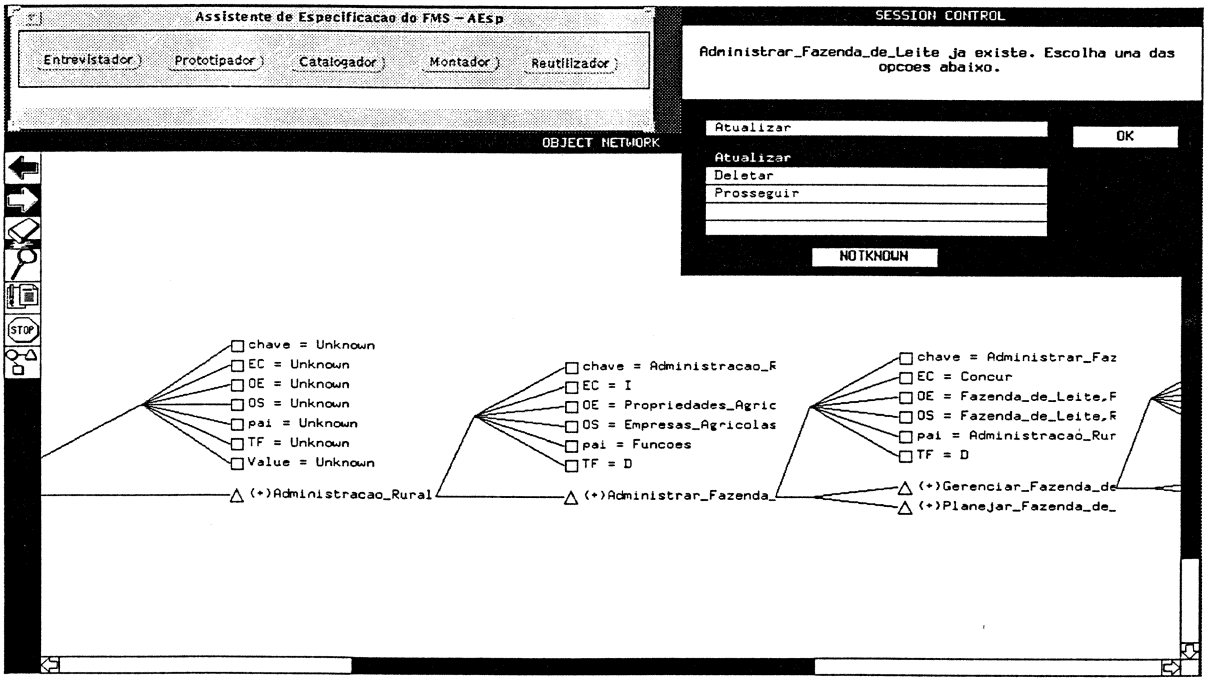


Figura 6.3 (b): Tela apresentando as opções Atualizar, Deletar e Prosseguir

Seguindo as Figuras 6.4 (a) e 6.4 (b), pode-se observar que o Entrevistador também consulta a base Primitivas.nxp e Sinônimos.nxp, que são mantidas via

Catalogador,toda vez que uma atividade é referenciada . Caso a atividade não seja encontrada, na base de nós primitivos ou na base de sinônimos, é seguida a entrevista com as questões que mapeiam para a decomposição em HOS. Se a atividade é encontrada na base de nós primitivos as perguntas são direcionadas segundo as regras referentes àquele nó primitivo, isto é, segundo o CMI do nó primitivo.

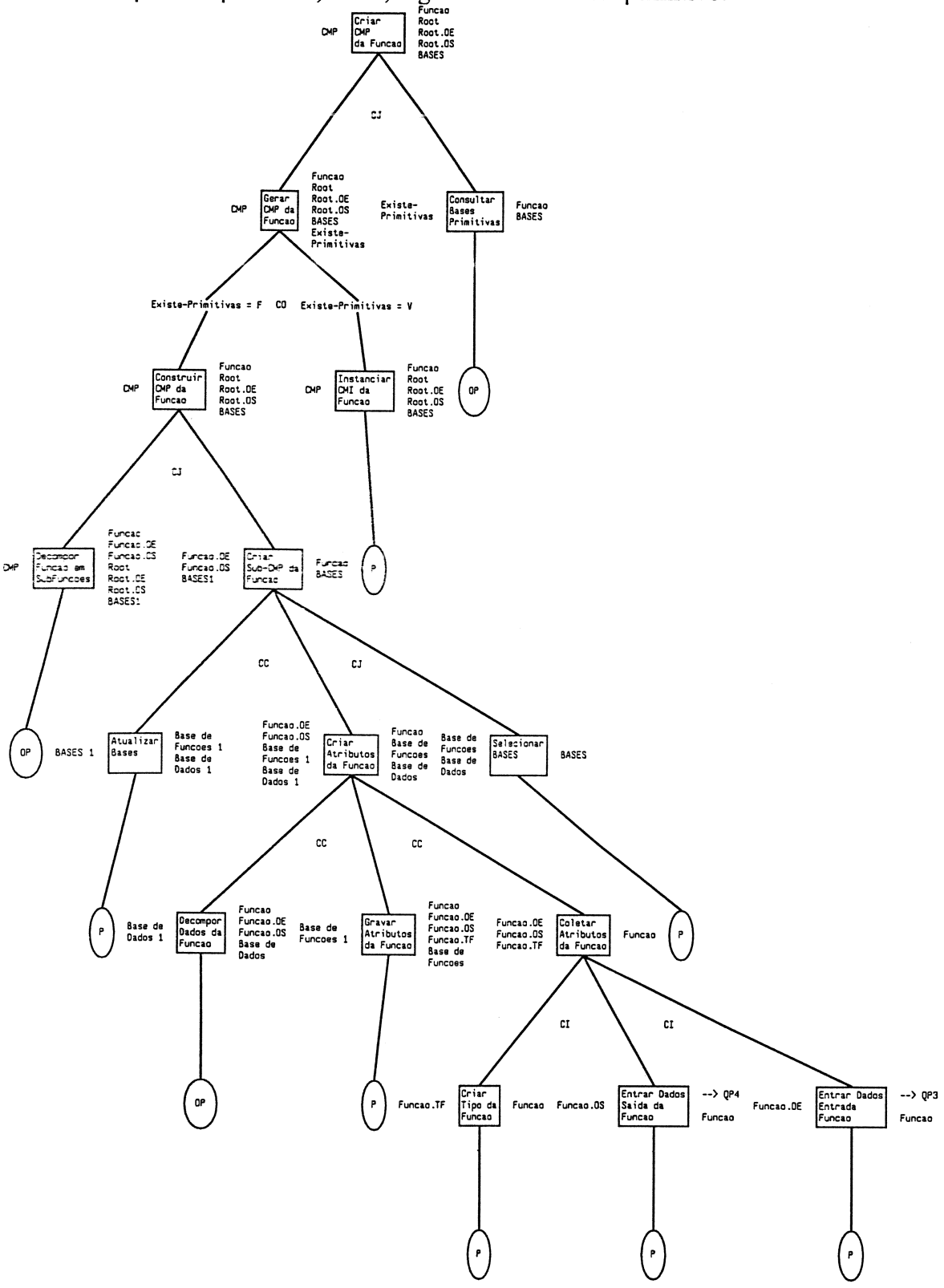


Figura 6.4 (a): Criar_Cmp_da _Função (pertence ao CMI - Criar_CMP_Instanciado)

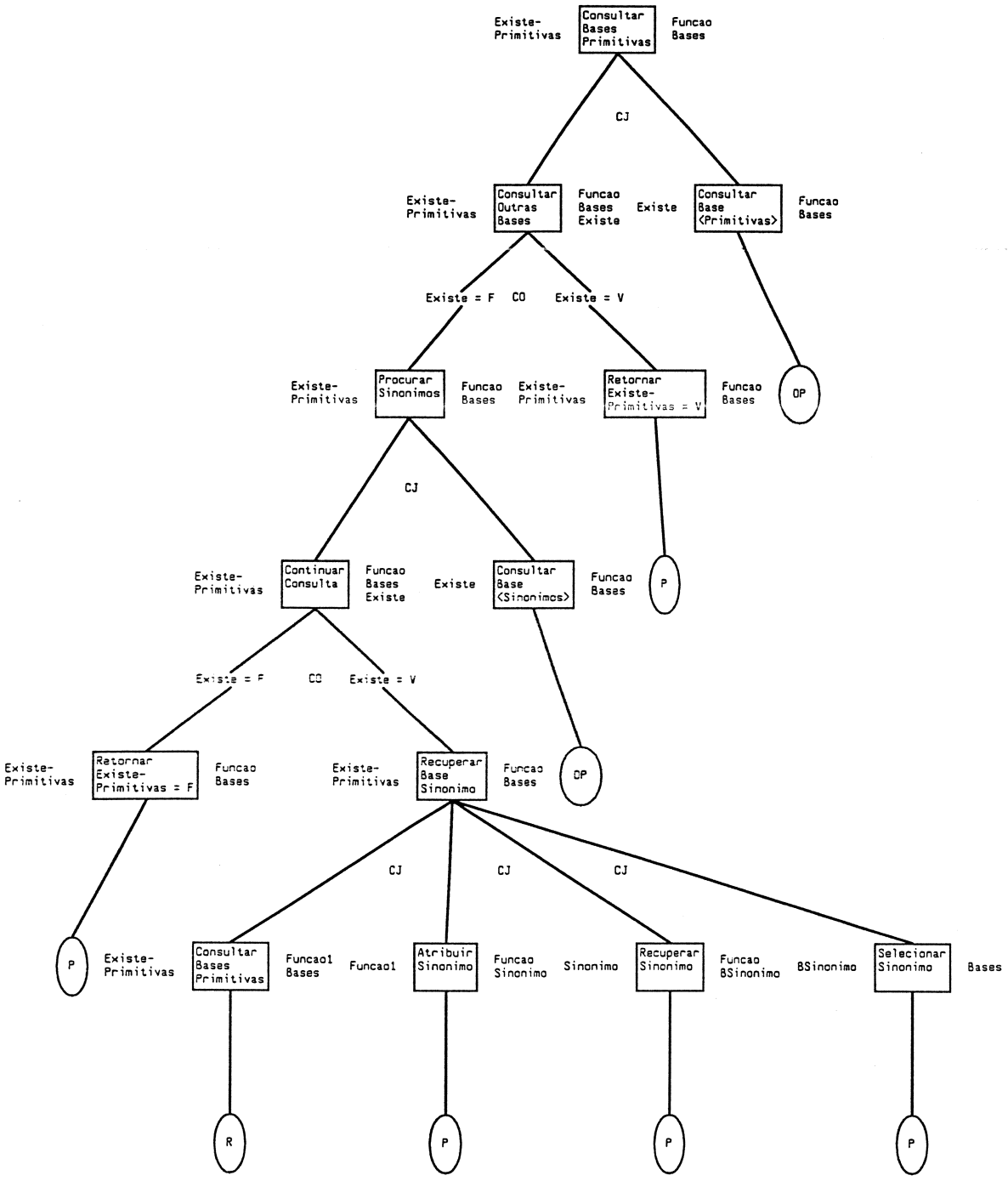


Figura 6.4 (b): CMO - Consultar_Base_Primitivas

Durante uma sessão sob o AEsp, esses CMIs que são executados e a instanciação destes "Control Maps" geram o "Control Map " instanciado , isto é, a especificação da aplicação em HOS e seus objetos do nível de representação, conforme descrito no Capítulo 4.

Abaixo está descrita parte de uma sequência de questões apresentadas, durante um processo de entrevista, para especificar a aplicação de Administrar Fazenda de Leite. Acompanhe o roteiro da entrevista seguindo a árvore de decomposição gerada na Figura 6.5. As Figuras, 6.5 e 6.6, mostram a tradução das funções e dados de Administrar_Fazenda_de_Leite em objetos do domínio (Controlar, Relatar) e a tradução destes em objetos da classe Modelagem da Aplicação é mostrado nas Figuras 6.7 (a), 6.7 (b), 6.7 (c), 6.7 (d), 6.7(e), 6.7(f).

- Roteiro da Entrevista:

Q1) Você está no ambiente FMS. As classes de aplicações conhecidas estão descritas abaixo. Entre com a sua opção.
R: *Administração Rural* ←

Q2) Que aplicação deseja desenvolver?
R: *Administrar_Fazenda_de_Leite*

Neste momento foi criado o nó Administrar_Fazenda de _Leite (Figura 6.5)

Q3) Quais as informações necessárias para <*Administrar_Fazenda_de_Leite*>?
R: *Fazenda_de_Leite,Praticas_Administrativas*

Q4) O que resulta de <*Administrar_Fazenda_de_Leite*>?
R: *Fazenda_de_Leite,Relatórios_Administrativos*

Q5) Descreva usando verbos e substantivos o que é <*Administrar_Fazenda_de_Leite*>.
R: *Gerenciar_Fazenda_de_Leite, Planejar_Fazenda_de_Leite*

Neste momento foram criados os nós:
Gerenciar_Fazenda de _Leite e Planejar_Fazenda_de_Leite (Figura 6.5)

Q6) Você descreveu <*Administrar_Fazenda_de_Leite*>. em <*Gerenciar_Fazenda_de_Leite, Planejar_Fazenda_de_Leite*>. Entre com a nova atividade que deseja descrever.
R: Planejar_Fazenda_de_Leite

Q7) <Planejar_Fazenda_de_Leite> já existe. Escolha uma das opções abaixo:
R: *Atualizar*
Deletar
Prosseguir ←

Note que Planejar_Fazenda_ de_Leite já foi criada (Figura 6.5) mas seus dados ainda não foram descritos, portanto, ele consulta a base de primitivas e sinônimos para prosseguir.

(consulta as bases: primitivas e sinônimos)

Neste momento é consultada a base de nós-primitivos:
Planejar ainda não foi cadastrado como nó primitivo,
portanto, é aplicado a decomposição HOS.

Q3) Quais as informações necessárias para <*Planejar_Fazenda_de_Leite*>?
R: *Fazenda_de_Leite,Praticas_Planejamento, Relatorios_Gerenciais*

Q4) O que resulta de <*Planejar_Fazenda_de_Leite*>?
R: *Fazenda_de_Leite,Relatórios_Planejamento*

Q5) Descreva usando verbos e substantivos o que é <*Planejar_Fazenda_de_Leite*>.
R: *a definir*

Neste exemplo a função Planejar_Fazenda_de_Leite não vai ser decomposta, portanto,
é tratado como um "stub"

Q6) Você descreveu *<Planejar_Fazenda_de_Leite>*. em *<a definir>*. Entre com a nova atividade que deseja descrever.
R: *Gerenciar_Fazenda_de_Leite*

Q7) *<Gerenciar_Fazenda_de_Leite>* já existe. Escolha uma das opções abaixo:
R: *Atualizar*
Deletar
Prosseguir ←

Note que Gerenciar_Fazenda_ de_Leite já foi criada (Figura 6.5) mas seus dados ainda não foram descritos, portanto, ele consulta a base de primitivas e sinônimos para prosseguir.

(consulta as bases: primitivas e sinônimos)

Neste momento é encontrado na base de sinônimos :
Gerenciar é sinônimo de Controlar,
portanto, é aplicado o CMI do "Controlar" cadastrado na base de nós -primitivos.

Q9) Quais as práticas de controle de *<Gerenciar_Fazenda_de_Leite>*?
R: *Práticas_Gerenciais*

Q10) Quais os relatórios necessários para *<Gerenciar_Fazenda_de_Leite>*?
R: *Relatórios_Gerenciais*

(consulta a base de coletivos)

Neste momento é consultada a base de coletivos :
Leite não é coletivo,
portanto, segue a decomposição HOS (Decompor Função em Subfunções).

Q5) Descreva usando verbos e substantivos o que é *<Gerenciar_Fazenda_de_Leite>*.
R: *Controlar_Rebanho_da_Fazenda, Controlar_Produção_da_Fazenda*

Neste momento há uma decomposição de dados:
Fazenda_de_Leite (ou Fazenda) é decomposto em Rebanho e Produção (Figura 6.6).

Q6) Você descreveu *<Gerenciar_Fazenda_de_Leite>*. em *<Controlar_Rebanho_da_Fazenda, Controlar_Produção_da_Fazenda>*. Entre com a nova atividade que deseja descrever.
R: *Controlar_Rebanho_da_Fazenda*

Neste momento é consistido o nível anterior,
isto é,
Os objetos de entrada e saída da função Administrar_Fazenda_de_Leite são consistidos com os objetos de seus filhos Planejar_Fazenda_de_Leite e Gerenciar_Fazenda_de_Leite:
⇒ Ec= Concur (consistente).

.
.
.

Q5) Descreva usando verbos e substantivos o que é *<Controlar_Rebanho_da_Fazenda>*.
R:*Controlar_Perfil_do_Rebanho,Controlar_Localização_do_Rebanho, Controlar_Sanidade_do_Rebanho, Controlar_EstadoReprodutivo_do_Rebanho.*

Neste momento há uma decomposição de dados:
Rebanho é decomposto em Perfil,Localizacao,Sanidade,EstadoReprodutivo (Figura 6.6).

Q6) Você descreveu *<Controlar_Rebanho_da_Fazenda>*. em *<Controlar_Perfil_do_Rebanho,Controlar_Localização_do_Rebanho, Controlar_Sanidade_do_Rebanho,Controlar_EstadoReprodutivo_do_Rebanho.* > Entre com a nova atividade que deseja descrever.
R: *Controlar_EstadoReprodutivo_do_Rebanho.*

Neste momento é consistido o nível anterior,
isto é,
Os objetos de entrada e saída da função Gerenciar_Fazenda_de_Leite é consistida com os objetos de seus filhos Controlar_Rebanho_da_Fazenda e Controlar_Produção_da_Fazenda:
⇒ Ec= I (Inconsistente, pois a atividade de Controlar_Produção_da_Fazenda ainda não foi descrita).

.
.
.

⇒ Aplicado CMI do "Controlar"

Q9) Quais as práticas de controle de *<Controlar_EstadoReprodutivo_do_Rebanho>*?
R: *Práticas_EstadoReprodutivo*

Q10) Quais os relatórios necessários para *<Controlar_EstadoReprodutivo_do_Rebanho>*?
R: *Relatórios_EstadoReprodutivo*

(consulta a base de coletivos)

Neste momento é consultada a base de coletivos:
Rebanho é coletivo de animais,
portanto, segue o CMI do "Controlar".

Q11) O que identifica cada *<animal>* do *<Rebanho>*?
R: *Brinco*

Q12) Quais os possíveis valores de *<EstadoReprodutivo>*?

R: *Vazia,Inseminada,Prenha,Parida.*

Q13) <*Vazia,Inseminada,Prenha,Parida*> é(são) uma descrição ou o(s) valor(es) de <*EstadoReprodutivo*>?
R: *Descrição*
Valores ←

.
.
.

(Processo de Inferência - Formulários)

.
.
.

Q14) Entre com o título para o formulário <*EstadoReprodutivo*>.
R: *Controle do estado reprodutivo do animal*

Q15) Entre com um help para o formulário <*EstadoReprodutivo*>.
R: *Este formulário controla os estados reprodutivos de cada animal do rebanho*

Q16) Entre com o tipo do formulário <*EstadoReprodutivo*>.
R: *Edit*
NoEdit
Browse
Config
Notice
Ev_Entry ←
Ev_Confirm

Q17) Foi identificado o(s) seguintes campo(s) para o formulário <*EstadoReprodutivo*>: <*Brinco, EstadoReprodutivo*>. Entre com uma das opções abaixo.
R: *Confirmar* ←
Inserir

Q18) Entre com os textos para o formulário <*EstadoReprodutivo*>.
R: *No.Brinco,EstadoReprodutivo*

.
.
.

(Processo de Inferência - Campos)

.
.
.

Q19) Entre com uma descrição para o campo <*Brinco*>.
R: *Número para identificar cada animal do rebanho*

Q20) Entre com o tipo do campo <*Brinco*>.

R: *String*
Numeric ←
Choice
Date
Logical

Q19) Entre com uma descrição para o campo <*EstadoReprodutivo*>.
R: *Estado de cada animal do rebanho*

Neste momento,
O tipo do campo <*estadoReprodutivo*> *já foi inferido como "Choice".*

.
.
.

(Processo de Inferência - Eventos)

.
.
.

Q21) Quando ou como passar de <*Vazia*> para <*Inseminada*>?
R: Quando há uma <*Inseminação*>

Q22) Quais as condições para que ocorra um(a) <*Inseminação*>?
R: A vaca estar no estado *Vazia* ou *Parida*

Q23: Descreva as próximas ocorrências quando há um(a) <*Inseminação*>?
R: Marcar um *toque* e uma *transferência de local*

Q21) Quando ou como passar de <*Inseminada*> para <*Prenha*>?
R: Quando é confirmada uma *Prenhês*

.
.
.

Q21) Quando ou como passar de <*Prenha*> para <*Parida*>?
R: Quando há um(a*) Parto*

.
.
.

Q6) Você descreveu <*Controlar_Sanidade_do_Rebanho*>. Entre com a nova atividade que deseja descrever.
R: *Finalizar*

Neste momento, é aplicado o CMI do Finalizar que consiste todo o "Control Map"Instanciado (Figura 6.5) e gera os formulários do tipo "menus" da aplicação correspondentes aos nïveis hierarquicos do "CMP" Instanciado.

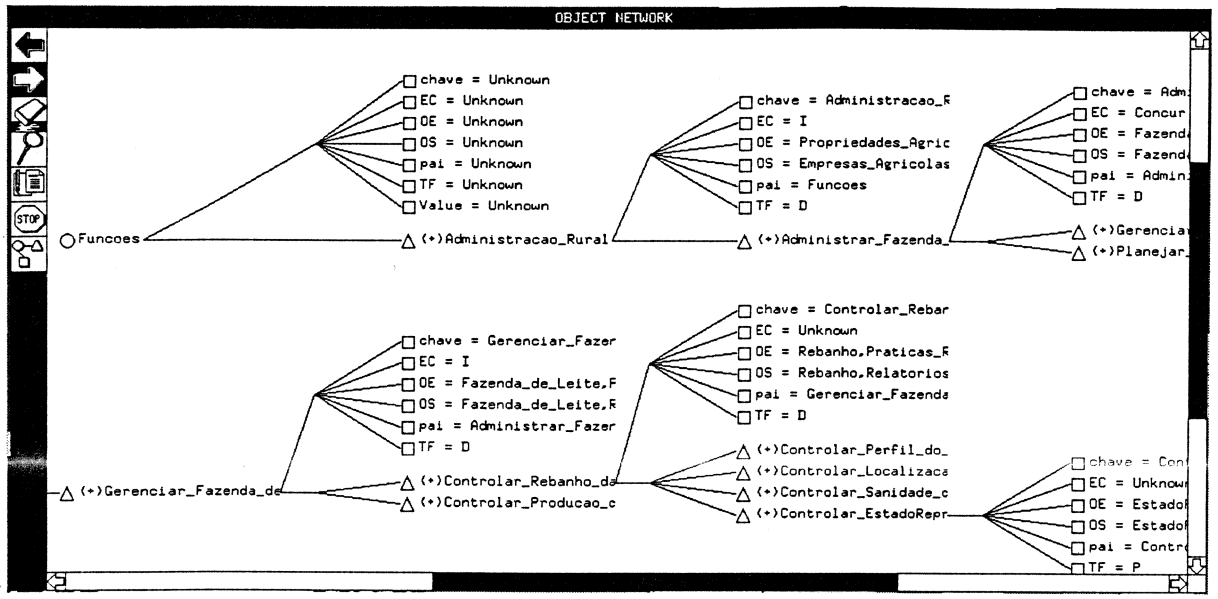


Figura 6.5: Control Map Instanciado

Na Figura 6.6 é mostrado um exemplo da decomposição de dados gerada durante o processo de decomposição acima.

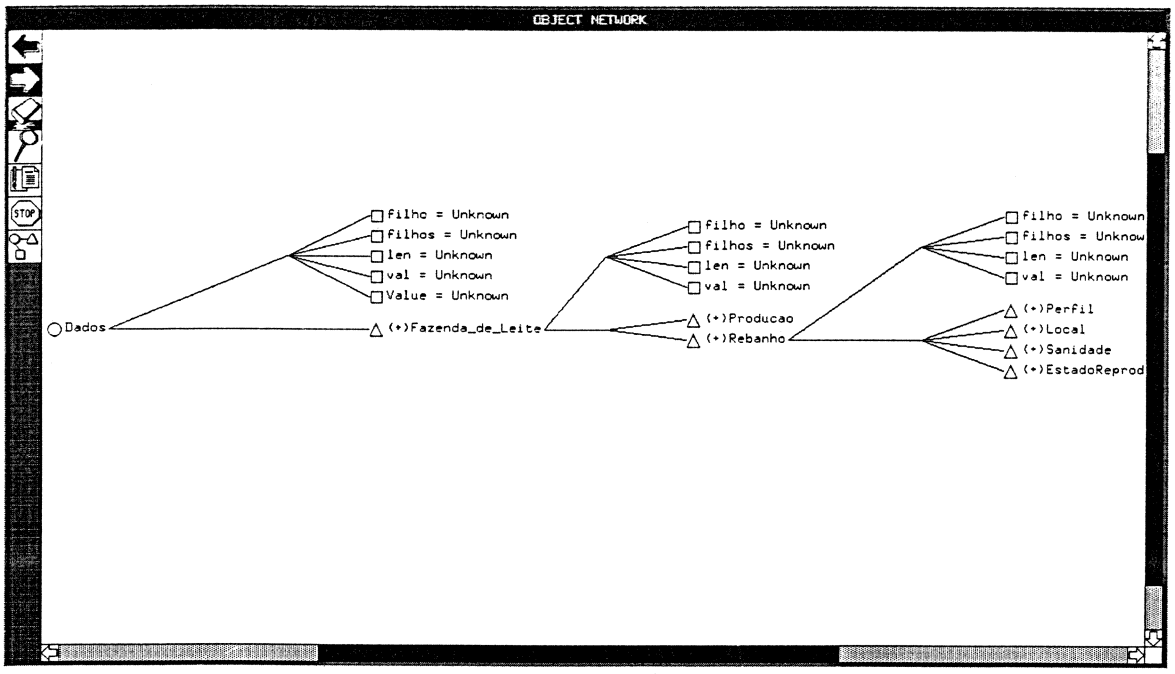


Figura 6.6: Decomposição de dados

Na Figuras 6.7 (a), 6.7 (b), 6.7 (c), 6.7 (d), 6.7 (e), 6.7 (f) é mostrado exemplos de objetos gerados nas classes do nível de representação: Forms, Campos, Textos, Transições, Relatórios e Eventos, respectivamente.

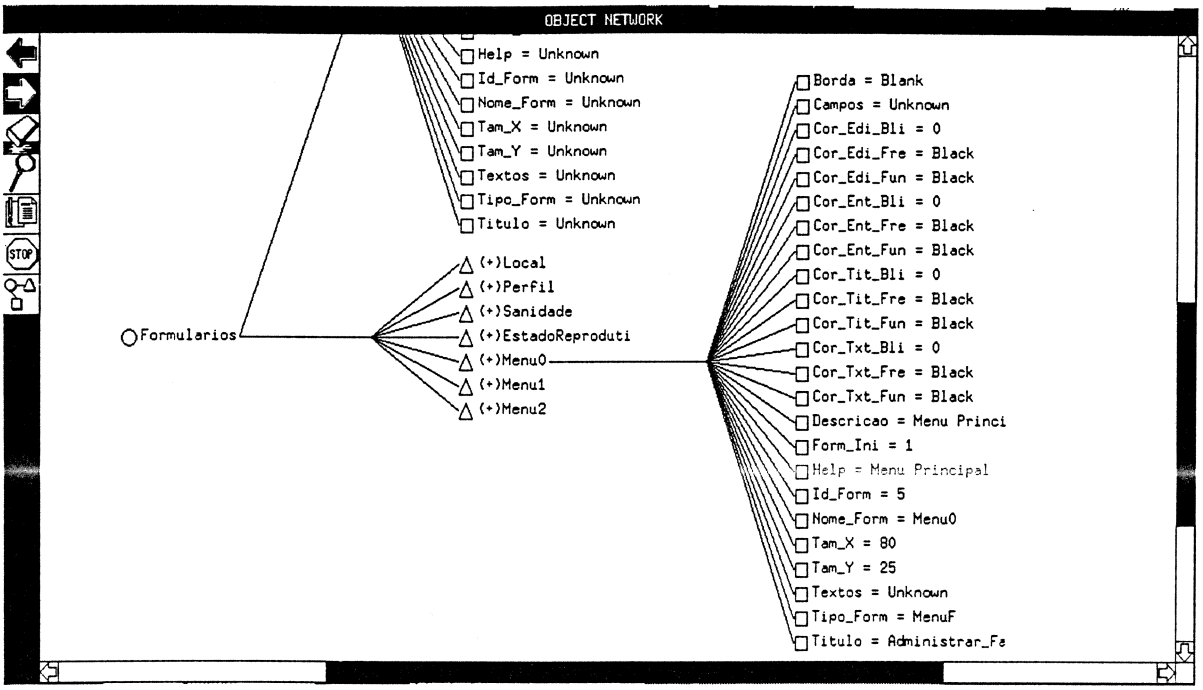


Figura 6.7 (a): Representação em Nexpert da classe Forms

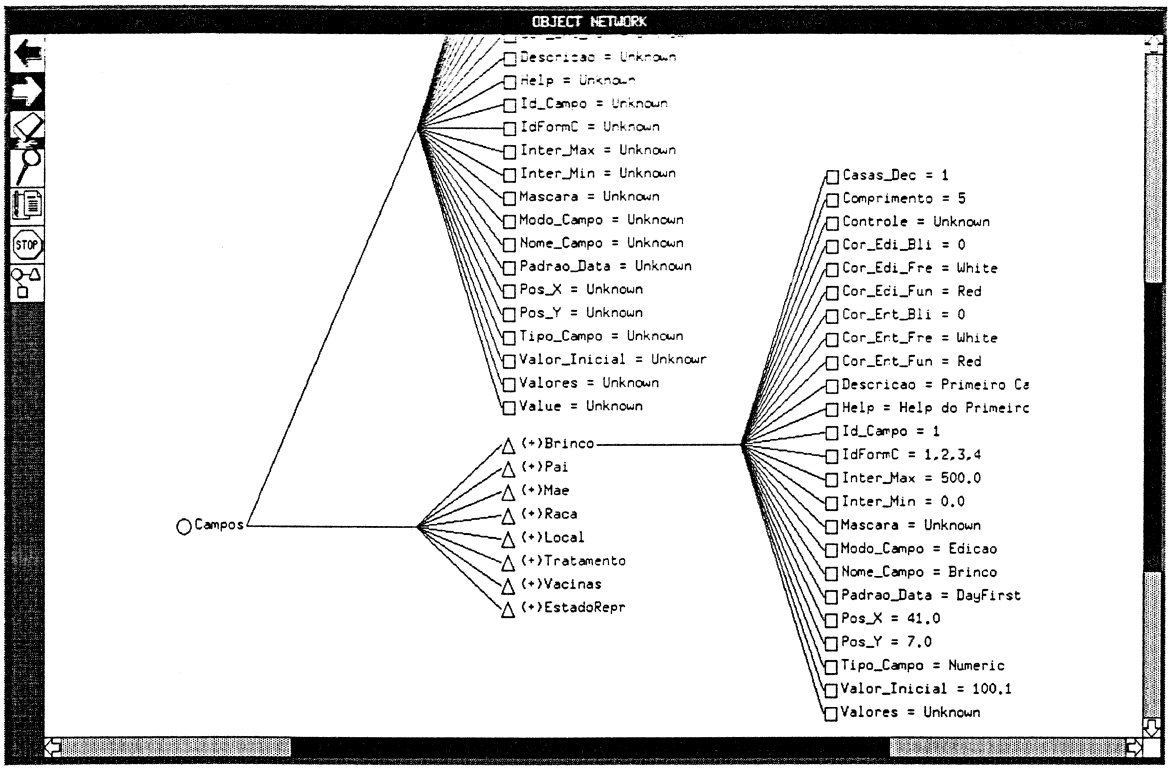


Figura 6.7 (b): Representação em Nexpert da classe Campos

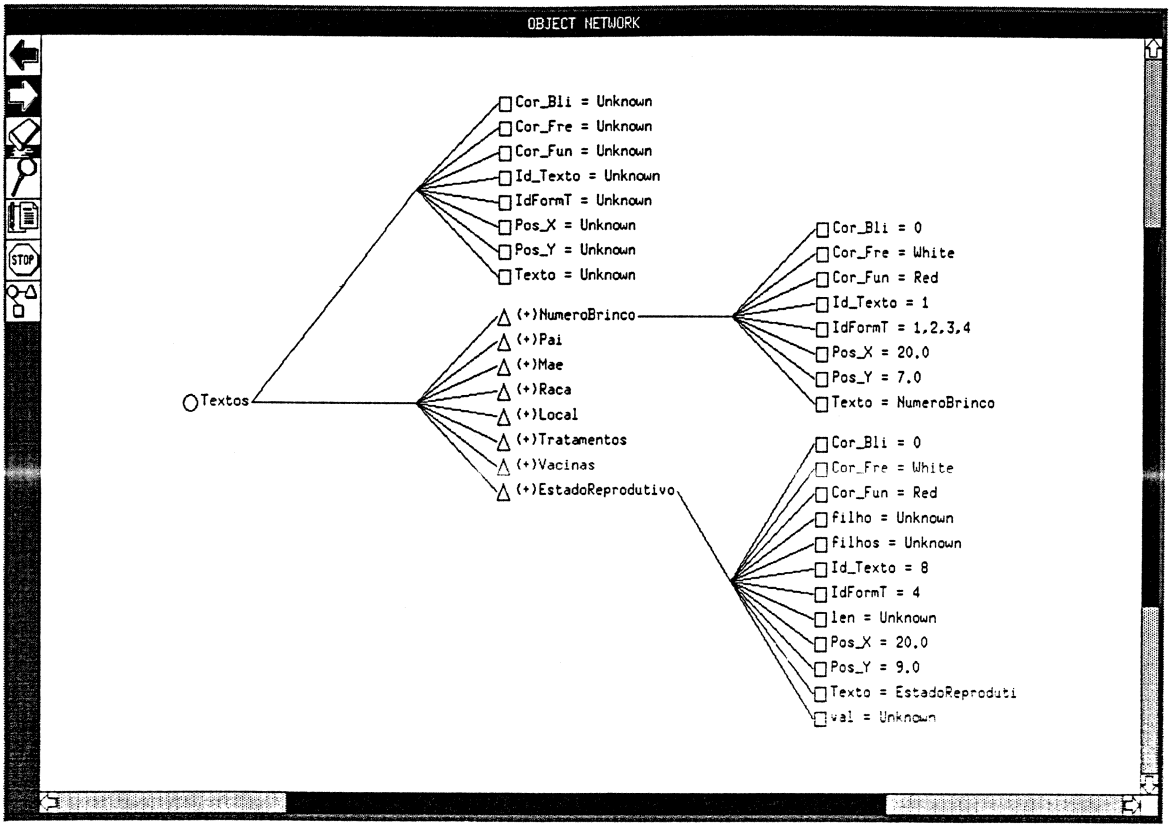


Figura 6.7 (c): Representação em Nexpert da classe Textos

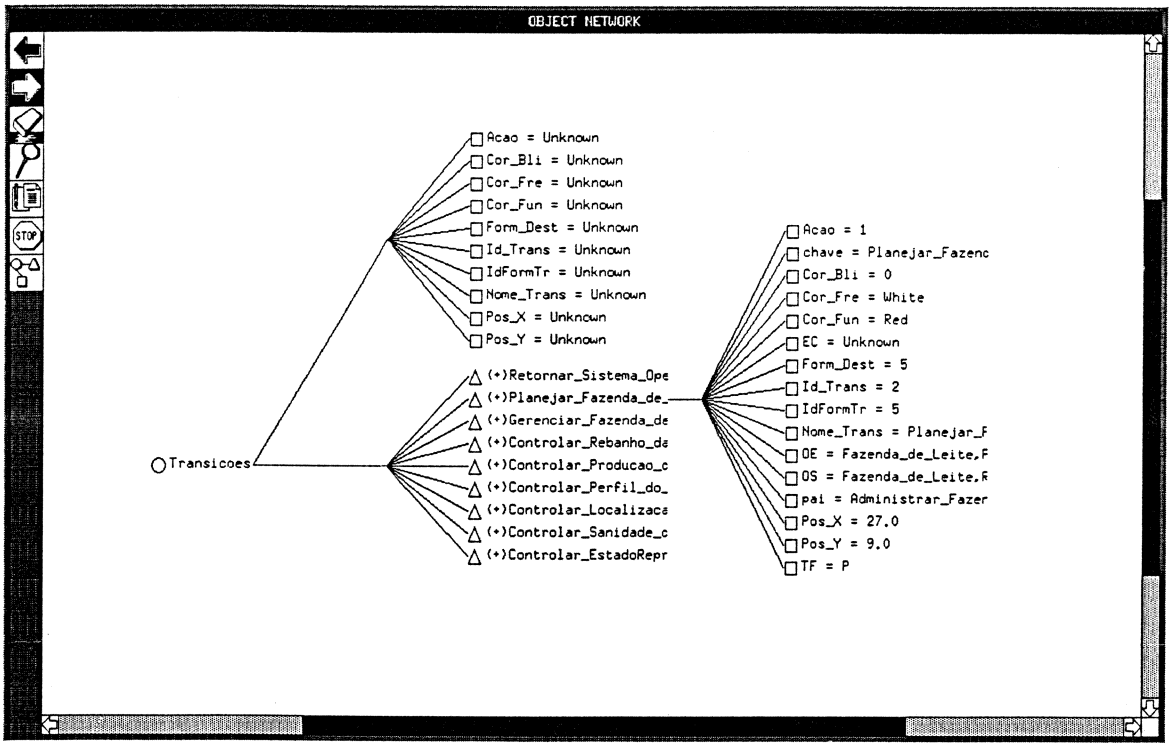


Figura 6.7 (d): Representação em Nexpert da classe Transições

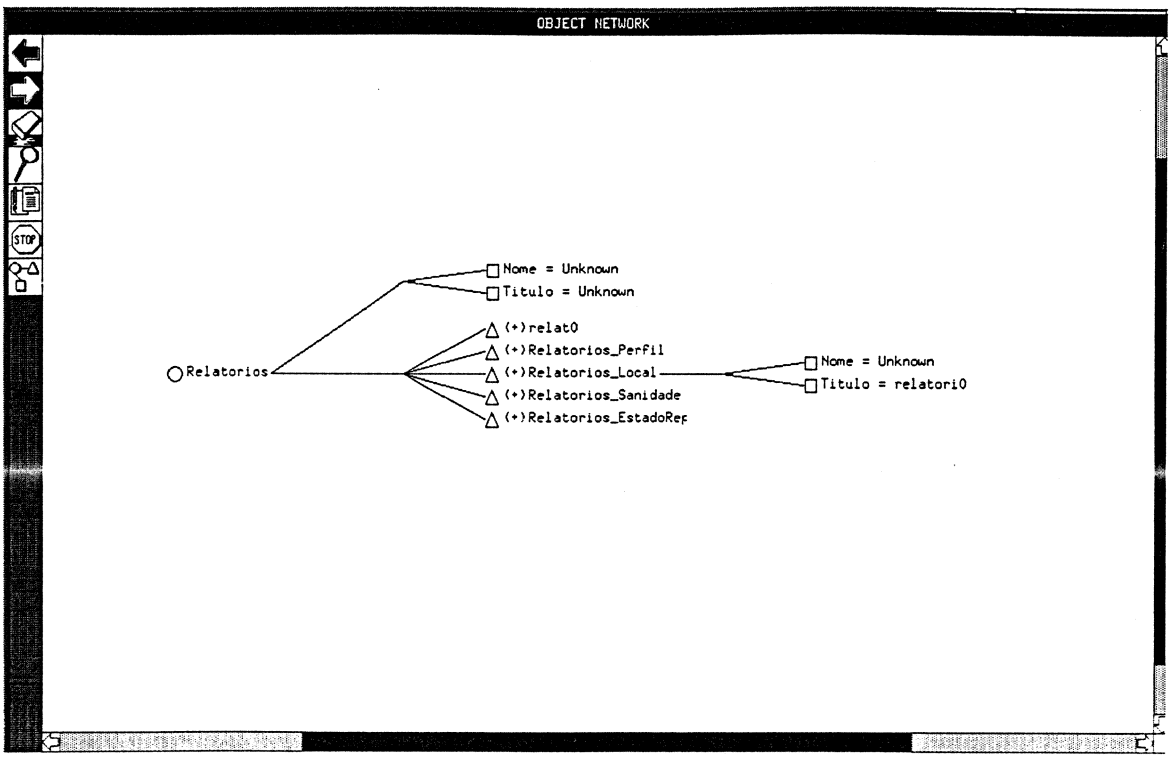


Figura 6.7 (e): Representação em Nexpert da classe Relatórios

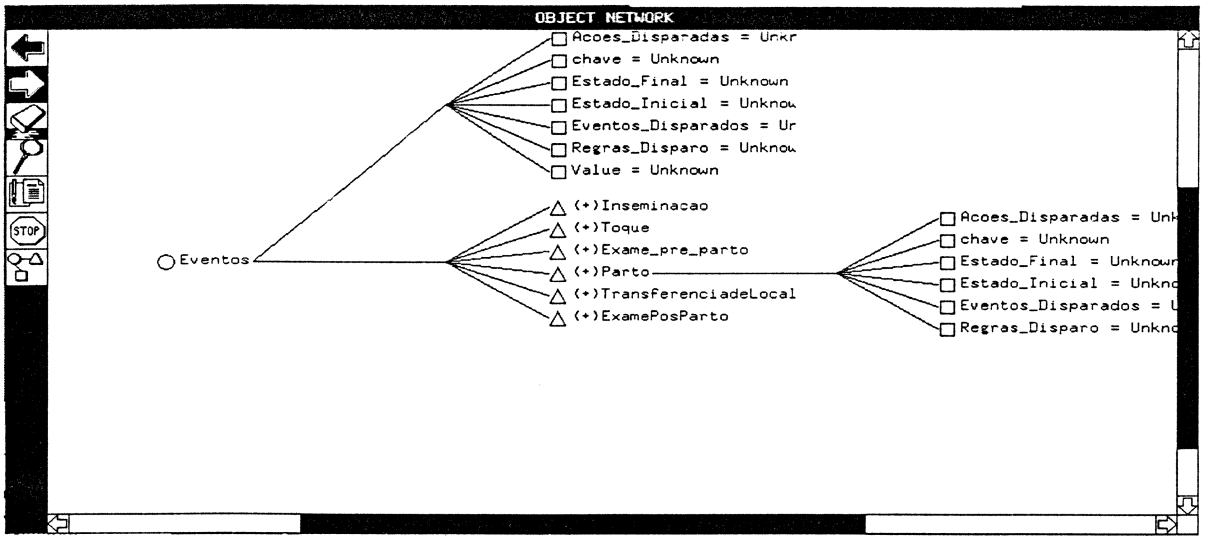


Figura 6.7 (f): Representação em Nexpert da classe Eventos

Após uma sessão do Entrevistador são atualizados os arquivos correspondentes às classes acima representadas. Os arquivos Funcoes.nxp e Dados.nxp da classe Decomposição. Os arquivos Forms.nxp, Campos.nxp, Textos.nxp, Transicoes.nxp, Relatorios.nxp e Eventos.nxp correspondentes a classe modelagem da aplicação.

Estes arquivos serão lidos pelo Prototipador e Montador para simular o comportamento da aplicação e gerar a especificação em LC-FMS respectivamente. A Figura 6.8 mostra um exemplo do formato destes arquivos. O exemplo dado é a base Forms.nxp.

```
 Id_Form|      Nome_Form|   Tipo_Form|                    Descric
ao|
                                                              Help|
   Tam_X|   Tam_Y|   Borda| Cor_Txt_Fre| Cor_Txt_Fun|Cor_Txt_Bli|
Cor_Ent_Fre| Cor_Ent_Fun|Cor_Ent_Bli| Cor_Edi_Fre| Cor_Edi_Fun|Cor
_Edi_Bli| Cor_Tit_Fre| Cor_Tit_Fun|Cor_Tit_Bli|
                                    Titulo|Form_Ini|
*****************************************************************
*****************************************************************
*****************************************************************
*****************************************************************
*****************************************************************
*****************************************************************
************************************************
       1|          Local|      Edicao|Os numero de piquetes varia
de|
                                                 Entre com os piquetes|
      80|      25|  Single|       White|         Red|          0|
      White|         Red|          0|       White|         Red|
       0|       White|         Red|          0|
                  Cadastro de Localizacao|       0|
       2|         Perfil|       MenuF| Perfil dos animais do reban
ho|
                       Formulario para edicao do perfil dos animais|
      80|      25|  Single|       White|         Red|          0|
      White|         Red|          0|       White|         Red|
       0|       White|         Red|          0|
            Perfil dos animais do Rebanho|       0|
       3|       Sanidade|      Edicao|            Sanidade do anim
al|
                                                          Sanidade|
      80|      25|  Single|       White|         Red|          0|
      White|         Red|          0|       White|         Red|
       0|       White|         Red|          0|
                     Sanidade do animal|       0|
       4|       EstadoRe|      Edicao| Estado reprodutivo do reban
ho|
                         Estado reprodutivo dos animais do Rebanho|
      80|      25|  Single|       White|         Red|          0|
      White|         Red|          0|       White|         Red|
       0|       White|         Red|          0|
              Estado Reprodutivo do animal|       0|
       5|           menu5|       MenuF|                 Menu Princip
al|
                                                    Menu Principal|
      80|      25|  Double|       White|         Red|          0|
      White|         Red|          0|       White|         Red|
       0|       White|         Red|          0|
             Administrar_Fazenda_de_Leite|       1|
@
@
"Forms.nxp" 10 lines, 4480 characters
```

Figura 6.8: Formato da base Forms.nxp

6.3) Prototipador e sua operação

Nesta seção é apresentada a execução do Prototipador a partir das bases geradas pelo Entrevistador. A tela inicial do Prototipador é apresentada na Figura 6.9:

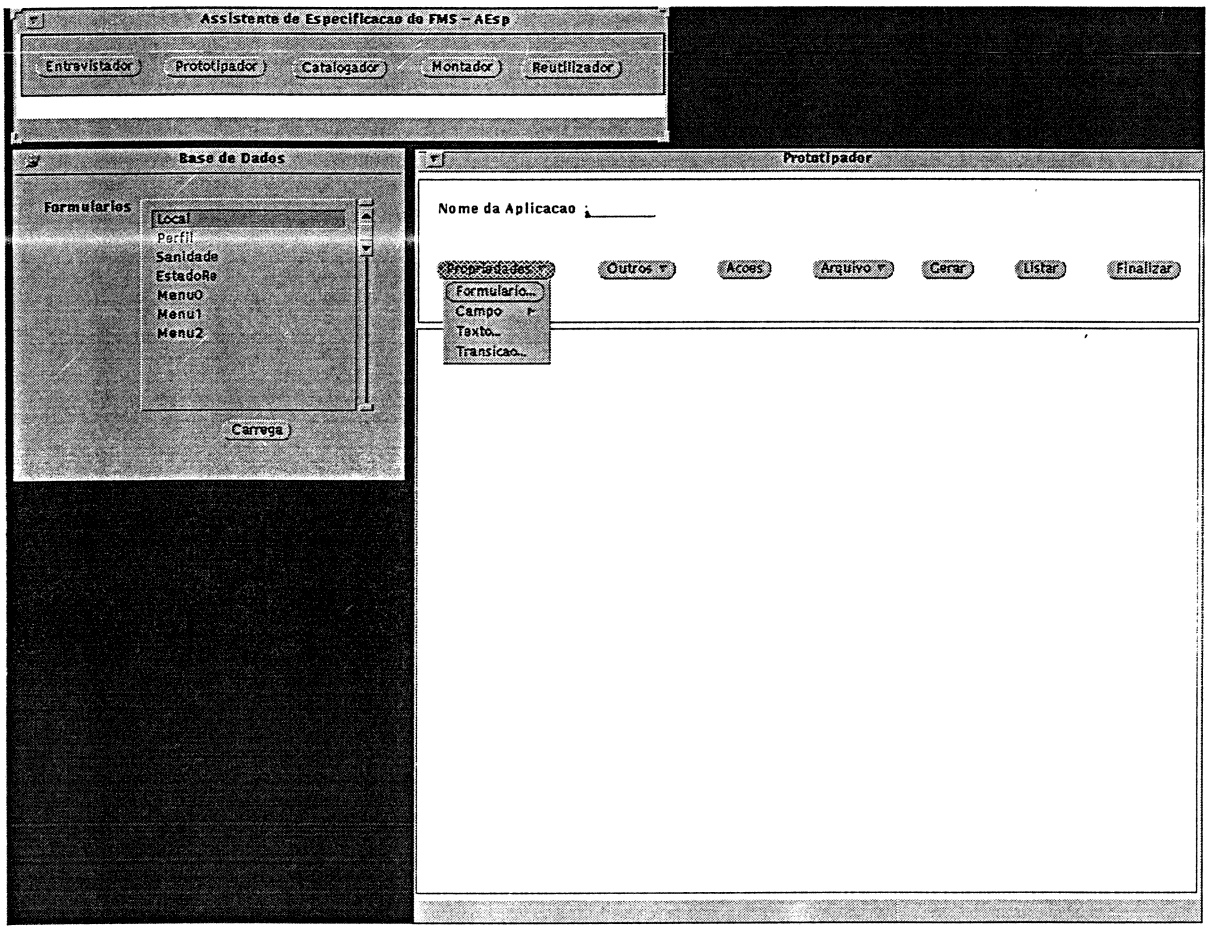


Figura 6.9: Tela Inicial do Prototipador

Na Figura 6.9, ao clicar o botão "Arquivo" aparece um menu com as opções: "Carregar" e "Salvar". Ao clicar "Carregar" aparece uma janela com os formulários que podem ser carregados. Note que são mostrados nesta janela os formulários tipo "Edit", gerados na aplicação das regras de produção aos nós primitivos da árvore da Figura 6.5, e os formulários tipo "NoEdit", que são os menus do sistema alvo, gerados na aplicação das regras de produção do CMI do "Finalizar".

Quando é carregado o formulário "Menu0", aparece na área de visualização este menu. Se clicarmos o botão de propriedades, pode-se ver as propriedades deste menu, conforme é mostrado na Figura 6.10.

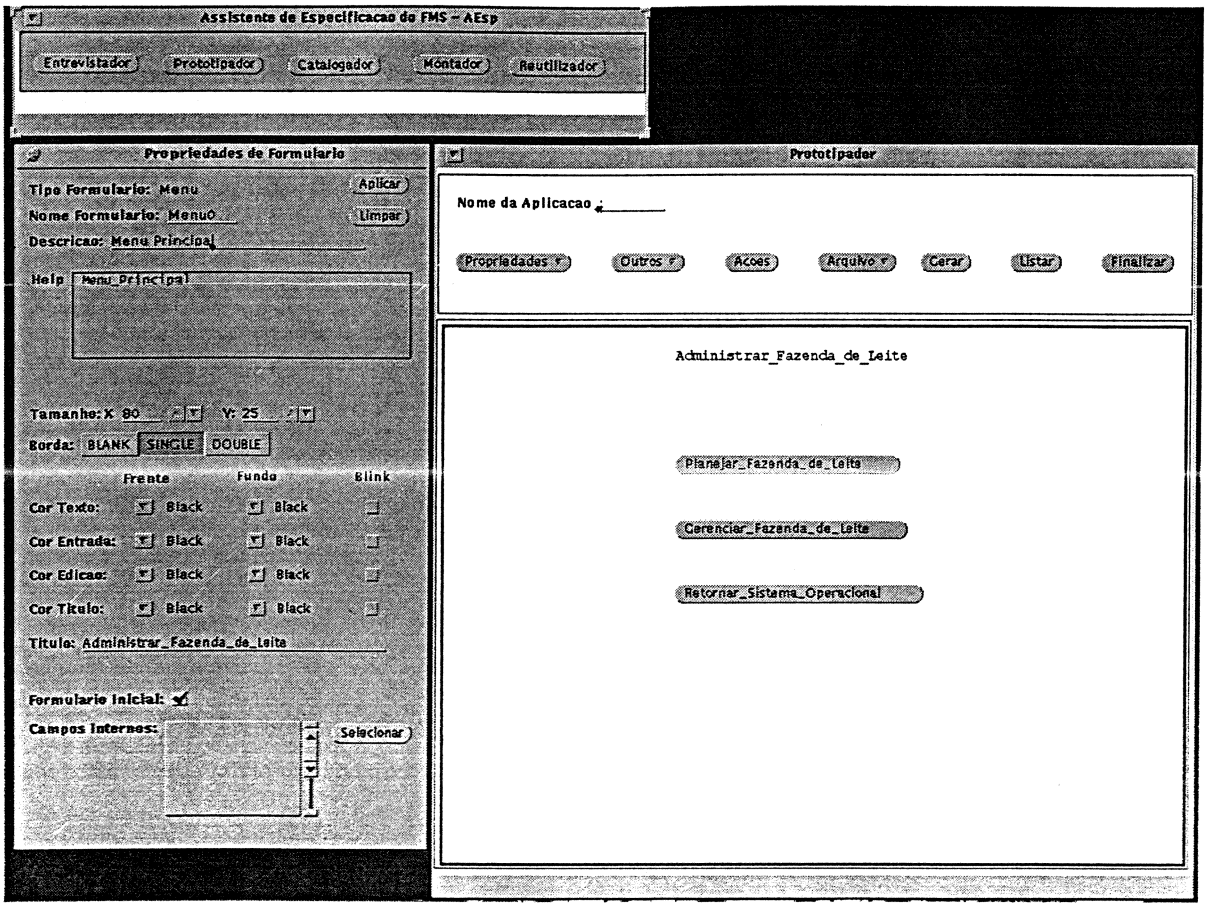


Figura 6.10: Formulário "Menu0" e sua tela de propriedades

Na sequência, ao clicar a opção de menu Gerenciar_Fazenda_de_Leite aparecerá o "Menu1". Este menu também pode ser carregado via o botão "Ärquivo" mas, aqui, é mostrado como o prototipador permite navegar nas telas da aplicação, simulando o comportamento da aplicação.

A Figura 6.11 mostra a tela "Menu1" carregada, onde é marcado a opção "Controlar_Rebanho_da_Fazenda" do menu e é mostrado as propriedades desta transição na tela ao lado. A Figura 6.12 (a) mostra o "Menu2" que aparece quando esta opção é clicada.

A propriedade "Borda" do "Menu2" está marcado "blank", isto é, sem borda. Se mudarmos este atributo para "single", uma borda simples aparecerá na tela como é apresentado na Figura 6.12 (b).

Neste exemplo, das Figuras 6.12 (a), 6.12 (b), é mostrado que o Especialista do Domínio e o Engenheiro de Especificação podem manipular os atributos da interface conforme desejarem.

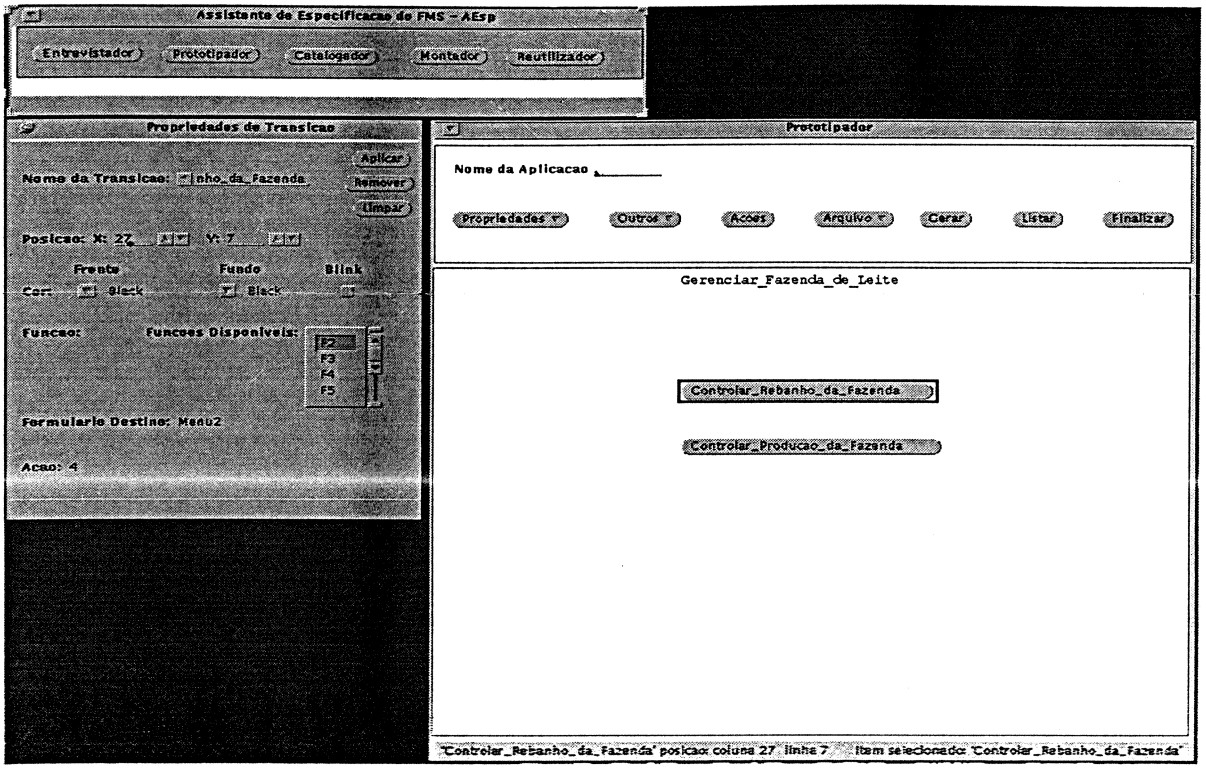


Figura 6.11: "Menu1" com as propriedades da transição "Controlar_Rebanho_da_Fazenda"

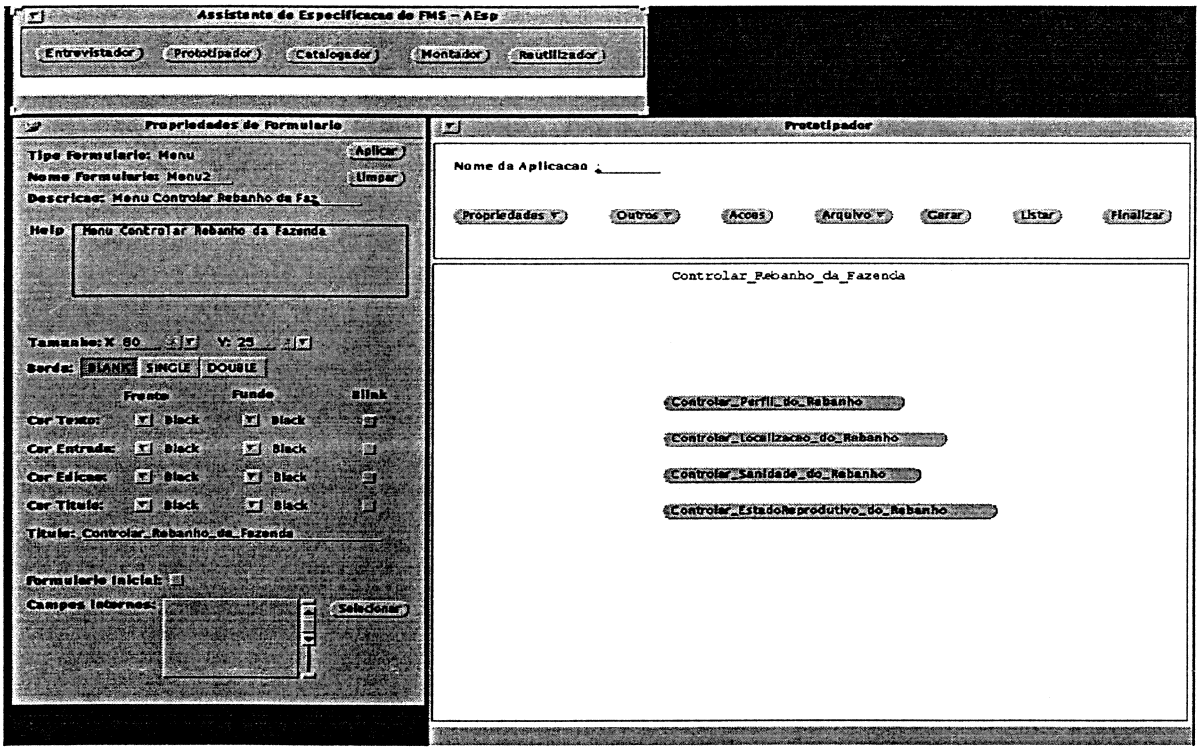


Figura 6.12 (a): "Menu2" com tela de propriedades do formulário

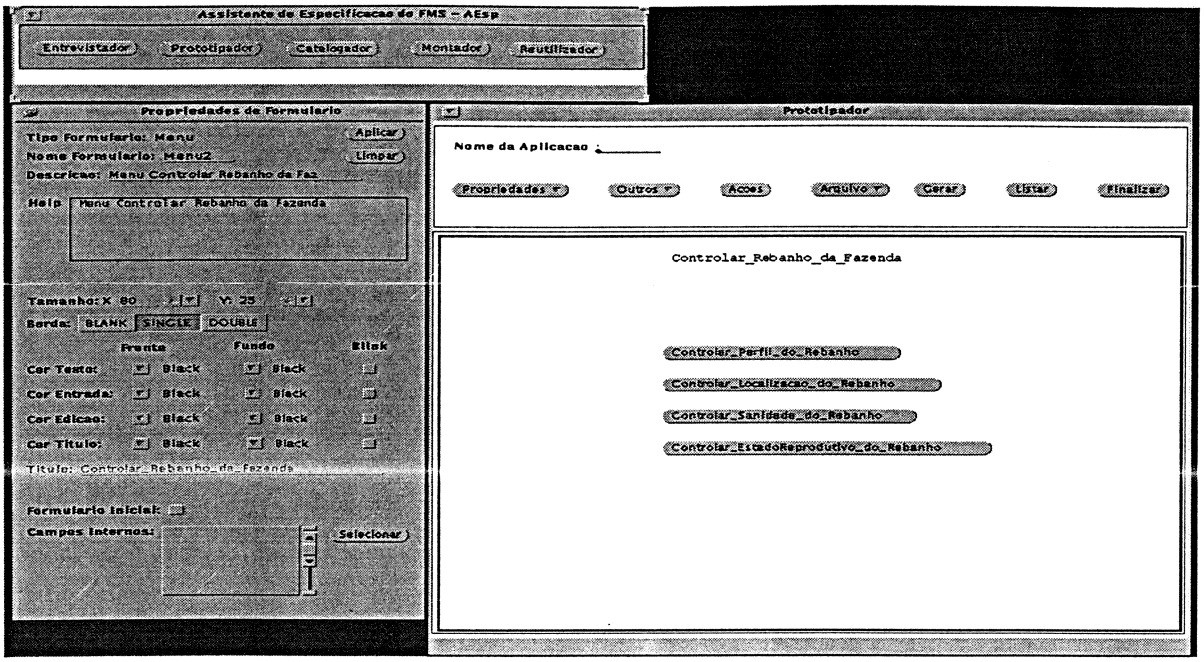


Figura 6.12 (b): "Menu2" com tela de propriedades do formulário alterada

Quando é escolhida a opção "Controlar_Perfil_do_Rebanho" do "Menu2" (Figura 6.12b) aparece a tela "Perfil" que é mostrada na Figura 6.13. Nesta figura, também são mostradas as propriedades de campos do campo "Brinco".

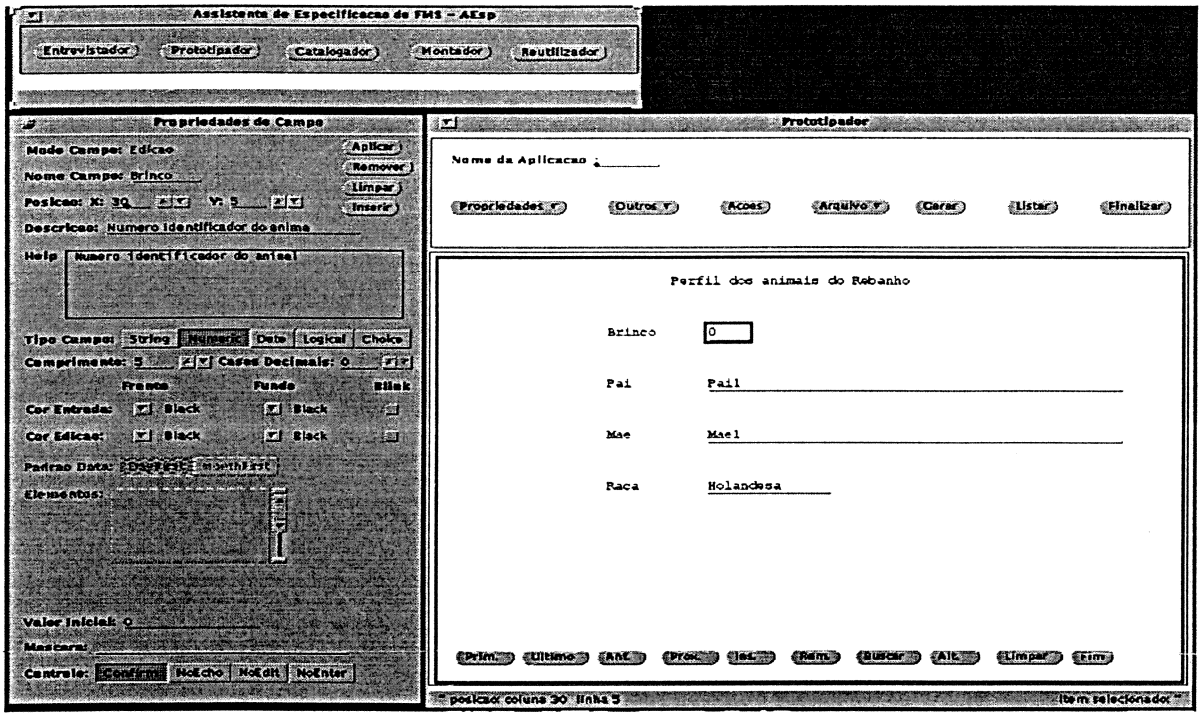


Figura 6.13: Tela "Perfil"e propriedades do campo "Brinco"

6.4) Montador e sua operação

A execução do Montador mostra na Figura 6.14, a especificação em LC-FMS de duas telas: Menu0 e Perfil. As especificações em LC-FMS são geradas ao clicar o botão "Montador" no AEsp. Neste momento, o Montador, a partir das bases geradas pelo Entrevistador e atualizadas pelo Prototipador, gera a especificação em LC-FMS.

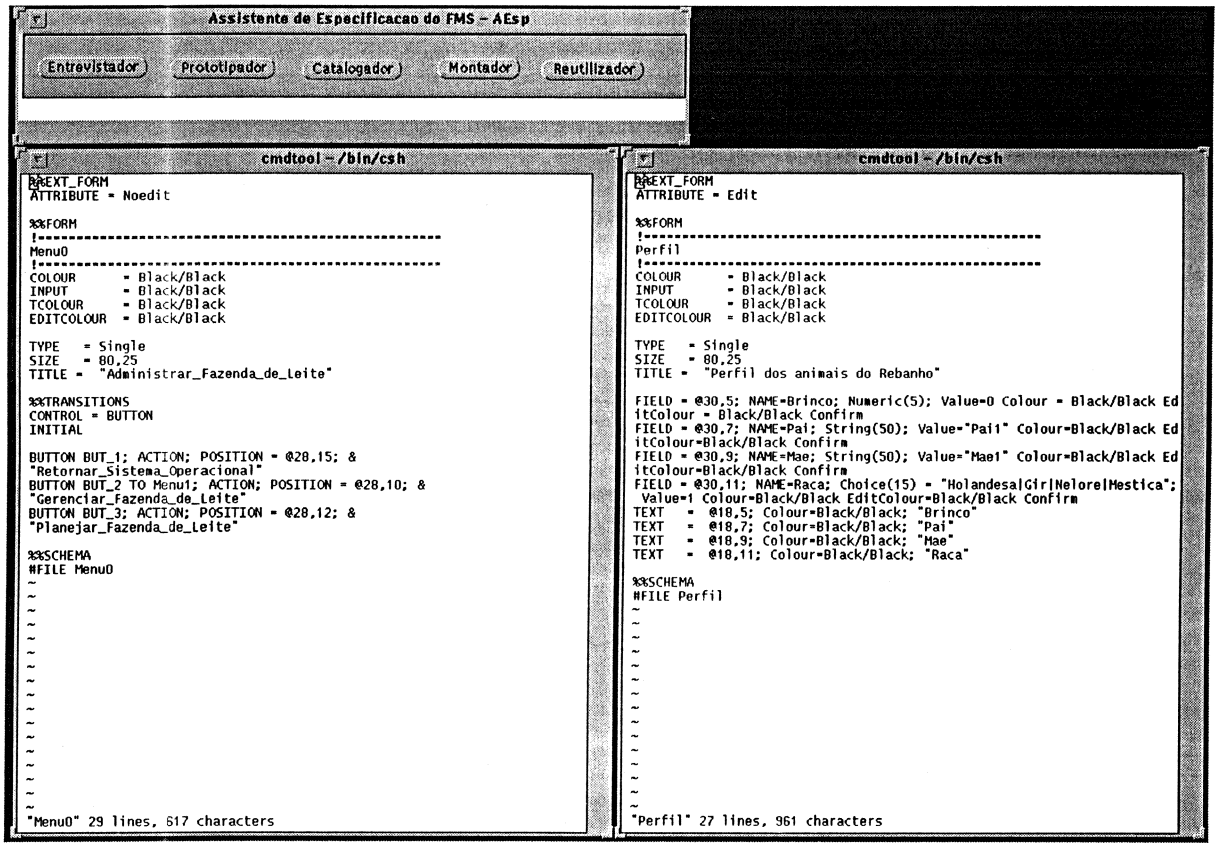


Figura 6.14: Exemplo de especificações geradas pelo Montador

As especificações em LC-FMS, geradas pelo Montador, que são os arquivos de entrada para o GFMS gerar os arquivos em linguagem C. Estes arquivos em C podem ser gerados na SUN, onde o AEsp é executado, ou no PC. Posteriormente, estes arquivos em C são compilados no PC, onde deve rodar a aplicação final do usuário.

6.5) Considerações sobre o capítulo

Neste capítulo foi mostrado um pequeno exemplo de uma especificação sob o AEsp. O exemplo mostrou desde a entrevista com o especialista do domínio até a geração da especificação em LC-FMS. A operacionalização do Catalogador e do Reutilizador, que têm suas funcionalidades parcialmente implementadas, não foram apresentadas diretamente. Porém, a operacionalização destas ferramentas não interferem no ciclo de especificação apresentado.

CAPÍTULO 7

CONCLUSÕES E TRABALHOS FUTUROS

7.1) Conclusões e resultados obtidos

Neste trabalho foi apresentada a base conceitual da arquitetura do AEsp e sua implementação. O AEsp é uma ferramenta para auxiliar a captura das especificações das aplicações do domínio de administração rural, e transformá-las incrementalmente em uma representação que pode ser traduzida para um programa operacional.

A arquitetura do AEsp está razoavelmente estabilizada. A implementação do Entrevistador em Nexpert já inclui um avaliador genérico de "Control Maps" e sua instanciação com um número de aproximadamente 250 regras. O Catalogador está parcialmente implementado e já permite o cadastramento dos componentes do domínio com seus respectivos CMIs. O Reutilizador também tem algumas funções implementadas que são disparadas pelo Entrevistador mas ainda é bastante primário. O Prototipador e Montador estão implementados e geram, respectivamente o protótipo da interface da aplicação e parte substancial das especificações em LC-FMS. As sublinguagens em LC-FMS, que já são geradas, são aquelas que tem seus compiladores operacionais no GFMS, excluindo a sublinguagem de eventos e ações.

As características do AEsp, como implementado, são compatíveis com os outros projetos deste tipo [MAS94A, MAS94B]. O AEsp cobre uma classe de aplicações simples, portanto, o seu escopo é mais limitado que os assistentes apresentados no Capítulo 2 [ASL91], [CZU88], [GOM92A] e [WAT91]. Porém, pressupõe-se que essa focalização aumente a viabilidade do AEsp.

O AEsp tem dificuldades operacionais por exigir uma plataforma de hardware heterogênea. Há estudos de viabilidade que mostram que a sua migração completa para a plataforma alvo (PC) é possível pela substituição do SILK por um gerador de interfaces para Windows e a execução do Nexpert neste hardware.

O AEsp , bem como as outras ferramentas do ambiente FMS (GFMS e ferramentas auxiliares), também está sendo usado para a definição e geração de um Sistema de Controle de Rebanho Leiteiro -SisCoReb.

O componente do domínio "Controlar", essencial para essa aplicação, está bem definido e estudado, e já há uma descrição deste objeto (CMI) com a correspondente linguagem de operadores (CMO) para sua catalogação. O SisCoReb já incorpora os conceitos de eventos e decomposição de estados suportados pela metodologia de entrevista usada no AEsp. A versão alfa do sistema está sendo validada por dois usuários. O refinamento e evolução do AEsp será feita a partir desta validação.

Portanto, o modelo preconizado pelo AEsp apresenta um bom potencial de expressividade, utilidade e exequibilidade com os componentes instanciáveis já identificados. Estes componentes abrangem a parte de gerência da propriedade agrícola. A incorporação do componente "Planejar" ampliará esta abrangência dentro do domínio de Administração Rural.

7.2) Trabalhos Futuros

No âmbito da implementação do AEsp visa-se:

a) Implementação de uma interface para o Entrevistador independente do Nexpert, isto é, o Entrevistador é a "cara" do AEsp para os usuários. Na atual implementação, a interface do Entrevistador está muito limitada. Portanto, necessita-se do desenvolvimento de uma interface mais "amigável" para esta ferramenta;

b) Implementação de algumas funcionalidades no Catalogador, tais como:

- Linguagem de descrição de CMO mais estável para incorporação automática de novos CMIs, isto é, os CMOs devem ser o mais parametrizáveis possível para que na identificação de novos CMIs não se tenha que desenvolver novos CMOs. Consequentemente, aumentaria o reuso de CMOs na especificação de novos objetos do domínio;

- Criador genérico de bases de predicados, isto é, as bases, que identificadas durante a especificação dos CMIs, sejam incorporadas automaticamente na sua catalogação;

c) Especificação e implementação dos "reuse daemons" do Reutilizador para serem utilizados durante a entrevista;

d) Evolução do Entrevistador, Prototipador e Montador para suportar todas sublinguagens que já estão operacionais pelo GFMS;

e) Incorporação do componente do domínio "Planejar", já identificado na atividade de pré-análise do domínio, no modelo de Entrevista do AEsp.

Para a incorporação de novos objetos do domínio no AEsp, identificados na atividade de pré-análise do domínio, deve ser reaplicada a metodologia apresentada neste trabalho, que pode ser resumida em 2 passos:

a) Especificação e Implementação do processo de tradução dos novos objetos do domínio para os novos objetos do nível de representação correspondentes (CMIs e CMOs);

b) Incorporar na entrevista as perguntas que foram elicitadas acima.

A identificação e incorporação de novos componentes instanciáveis dependerá da utilização do AEsp em larga escala.

REFERÊNCIAS BIBLIOGRÁFICAS

[AHO86] AHO, A. V.; SETHI, R.; ULLMAN, J. D. **Compilers - Principles, Techniques, and Tools**, Addison Wesley, 1986.

[ARA89] ARANGO , G. "Domain Analysis - from art form to engineering discipline", **ACM SigSoft Software Engineering Notes**, v. 14, n. 3, May 1989.

[ARI92] ARIAS, C. **Um assistente especialista para especificação de requisitos**, Campinas: UNICAMP/IMECC, 1992. (Tese de Mestrado)

[ASL91] ASLETT, M. J. **A knowledge based approach to software development**: ESPRIT Project ASPIS. Amsterdam: North Holland, 1991.

[BAL86] BALZER, R.; GOLDMAN, N.; WILE, D. "Informality in Program Specifications". In: RICH, C.; WATERS R. C. **Readings in artificial intelligence and software engineering**. Los Altos, CA: Morgan Kaufmann, 1986. p. 223-232.

[BIG86] BIGGERSTAF , T.J.; RICHTER, C. Reusability framework, assessment, and directions. Austin: MCC, 1986. (Technical report)

[BOE81] BOEHM, B. **Software engineering economics**. Englewood Cliffs: Prentice Hall, 1981.

[BOR86] BORGIDA, A.; GREENSPAN, S.; MYLOPOULOS, J. Knowledge representation as the basis for requirements specifications. In: RICH, C.;WATERS R. C. **Readings in artificial intelligence and software engineering**. Los Altos, CA: Morgan Kaufmann, 1986. p. 561-570.

[CAL91] CALDIERA,G; BASILI, V.R. Identifying and qualifying reusable software components, **IEEE Computer**, p. 61-69, Feb. 1991.

[CLE88] CLEAVELAND, J.C. Building application generators, **IEEE Transactions on Software Engineering**, v. 5, n. 4, p. 25-33, Jul. 1988.

[COD92] **CODEBASE 5.0 User's Guide:** the C and C++ library for database management. Canadá: Sequiter Software Inc.,1993.

[CZU88] CZUCHRY, A.J.; HARRIS, D.R. KBRA: A new paradigm for requirements engineering, **IEEE Expert**, v. 3, n. 4, p. 21-35, winter, 1988.

[DAN90] DAN H. **Xview Programming Manual**,. O'Reilly & Associates, vol. 7 , 1990.

[DIA87] DIAZ P.R. Domain Analysis for Reusability, **Procedings of COMPSAC'87**, p. 23-29, 1987.

[DIA90] DIAZ, P.R. Domain analysis: an introduction. **ACM Sigsoft Software Engineering Notes**, v. 15 n. 2, p. 47-54, Apr. 1990.

[DIA91] DIAZ, P.R.; ARANGO, G. **Domain analysis and software systems modeling.** Los Alamitos, CA: IEEE Computer Society, 1991.

[FER93] FERRARETTO, M.D.; MASSRUHÁ, S.M.F.S. **Projeto: ambiente de desenvolvimento de software para domínio de administração rural - FMS.** Campinas/SP: EMBRAPA-CNPTIA, 1993. (Documento interno apresentado ao Sistema EMBRAPA de Planejamento - SEP)

[FER94A]FERRARETTO, M.D.; MASSRUHÁ, S.M.F.S. **Projeto: ambiente de desenvolvimento de software para domínio de administração rural - FMS.** Campinas/SP: EMBRAPA-CNPTIA, 1994. (Documento interno apresentado ao Sistema EMBRAPA de Planejamento - SEP)

[FER94B] FERRARETTO, M.D. **Metodologia para desenvolvimento rápido de aplicações - MEDRA**. Campinas/SP: EMBRAPA-CNPTIA, 1994. (Documento interno)

[FIS92] FISCHER, G.; GIRGENSOHN, A.; NAKAKOJI, K.; REDMILES, D. *Supporting Software Designers with Integrated Domain-Oriented Design Environmemts,* **IEEE Transactions on Software Engineering**, v. 18, n. 6, Jun. 1992.

[FRE87] FREEMAN, P. A Conceptual analysis of the Draco approach to constructing software systems. **IEEE Transactions on Software Engineering**, v. 13. n. 7, p. 830-844, Jul. 1987.

[GAU89] GAUSE, D. C.; WEINBERG, G. M. **Exploring requirements: quality before design**. Dorset House, 1989.

[GIA91] CALDIERA, G.; BASILI, V. R. Identifying and qualifying reusable software components. **IEEE Computer**, v. 24, n. 2, p. 61-69, Feb. 1991.

[GOM92A] GOMAA, H.; KERSCHBERG L.; SUGURUMAN V. A knowledge-based approach to generating target system specifications from domain model. In: **IFIP Congress**, Madrid, Spain, 1992.

[GOM92B] GOMAA, H.; KERSCHBERG L.; BOSCH, C.; SUGURUMAN V.; TAVAKOLI, I. **A prototype software engineering environment for domain modeling and reuse**. Fairfax/VI: George Mason University, 1992. (Technical report)

[GRE84] GREENSPAN, S.; MYLOPOULOS, J. Capturing more world knowledge in the requirements specification. In: FREEMAN, P.; WASSERMAN,A. I. **Tutorial on Software Design Techniques**, 1984. p. 231-240.

[GRE86] GREEN, C.; LUCKMAN,D.; BALZER, R.; CHEATHAM, T.; RICH, C. Report on a Knowledge-Based Software Assistant. In: RICH, C.;WATERS R. C. **Readings in artificial intelligence and software engineering**. Los Altos, CA: Morgan Kaufmann, 1986. p. 525-535.

[HAM76A] HAMILTON, M.; ZELDIN, S. Higher Order Software - A Methodology for Defining Software. **IEEE Transactions Software Engineering**, v. 2, n. 1, p. 09-32, Mar. 1976.

[HAM76B] HAMILTON, M.; ZELDIN, S. **Integrated software development system / Higher Order Software conceptual description**, version 1, Cambridge, nov. 1976. (Research and Development Technical Report, ECOM - 76 - 0329 - f)

[JOR89] JORDAN, P.W.; KELLER, K.; TUCKER; VOGEL, D. Software storming - combining rapid prototyping and knowledge engineering. **IEEE Computer**, v.22, n.5, p. 39-48, May 1989.

[KAR88] KARIMI, J.; KONSYNSKI, B.R. An Automated Software Design Assistant. **IEEE Transactions on Software engineering**, v. 14, n. 2, p. 194-210, Feb. 1988.

[LEH84] LEHMAN, M.M. A futher model of coherent programming processes. **Software Process Workshop**, p. 27-35, 1984.

[LEI90] LEITE, J.C.S.P. O uso de hipertexto na elicitação de linguagens de aplicação. Simpósio Brasileiro de Engenharia de Software, 4, Águas de São Pedro, 24-26 de outubro de 1990. **Anais**. São Paulo: USP/CCS, 1990. p. 134-149.

[LUH94] LUH, C. A modelling and simulation in object-oriented environment, Information and Software Technology, 1994, 36 (6), 343 - 352.

[LIN88] LINSTER, M. A Critical Look at Kriton. In: AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop, 3., Banff, Canadá, Nov. 1988. **Proceedings**

[LUQ88] LUQI. Knowledge-based support for rapid software prototyping. **IEEE Expert**, v.3, n.4, p.9-15, winter, 1988.

[LUQ89] LUQI. Software evolution through rapid prototyping. **IEEE Computer**, v.22, n.5, p. 13-25, May 1989.

[MAR85A] MARTIN, J. **System design from provably correct constructs**. Englewood Cliffs: Prentice-Hall, 1985.

[MAR85B] MARTIN, J. **Fourth Generation Languages**. New Jersey: Prentice-Hall, 1985. v. 1, p. 319-335.

[MAS93] MASIERO, P.C.; MEIRA, C.A.A. Development and instantiation of a generic application generator. **The Journal of Systems and Software,** v. 23, n.1, p. 27-38, Oct. 1993.

[MAS94A] MASSRUHÁ, S.M.F.S. **Um estudo sobre assistentes de especificação**. Campinas/SP: EMBRAPA-CNPTIA, 1994. (Relatório Técnico, submetido a publicação)

[MAS94B] MASSRUHÁ, S.M.F.S et alli. **AEsp: Um Assistente de Especificação**. Simpósio Brasileiro de Engenharia de Software, 8, Curitiba, 25-28 de outubro de 1994. **Anais**. Paraná: PUC/PR, 1994. p. 311-324.

[MEI91] MEIRA, C.A.A. **"Sobre Geradores de Aplicação"**, São Carlos: USP/ICMSC, Setembro, 1991. (Tese de mestrado)

[MEI92] MEIRA, C.A.A. **Relatório de Atividades - RHAE/CNPQ,** Período: 01/11/91 - 31/10/92. Campinas: EMBRAPA-CNPTIA, 1992. (Documento Interno)

[MOO91] MOORE, J.M., BAILIN, S.C.. Domain Analysis: Framework for Reuse, **Domain analysis and software systems modeling**. Los Alamitos, CA: IEEE Computer Society Press, p. 179-203, 1991.

[NEI86] NEIGHBORS, J.M. The Draco approach to constructing software from reusable components. In: RICH, C.; WATERS R. C. **Readings in artificial intelligence and software engineering**. Los Altos, CA: Morgan Kaufmann, 1986. p.525-535.

[NEI89] NEIGHBORS, J.M. **Draco: a method for engineering reusable software systems, software reusability, concepts and models**. 1989 (ACM Press Frontier Series, 1)

[NEX91] **NEXPERT OBJECT; version 2.0; introduction manual**. Palo Alto/CA: Neuron Data, 1991.

[PAS94] PASSOS, S.L.Z. **Relatório de Atividades - RHAE/CNPQ**, Período: 01/04/93 - 31/03/94. Campinas: EMBRAPA-CNPTIA, 1992. (Documento Interno)

[PRE91] PREMKUMAR D.; Ronald J.; BRACHMAN; SELFRIDGE, P.G.; BALLARD, B. Lassie: A Knowledge-Based Software Information System. **Communications of the ACM**, v. 34, n. 5, p. 34-49, May 1991.

[PRE92] PRESSMAN, R. **Software engineering: a practitioner's approach**. 3.ed. New York: McGraw-Hill, 1992.

[PUN88] PUNCELLO, P.; TORRIGIANI, P.; PIETRI, F.; BURLON, R.; CARDILE, B.; CONTI, M. ASPIS: A knowledge-based CASE environment. **IEEE Software**, v.5, n.2, p. 58-65, Mar. 1988.

[SCH88] SCHOEN, E.; SMITH, R.; BUCHANAN, B. Design of Knowledge-Based Systems with a Knowledge-Based Assistant, **IEEE Transactions on Software Engineering**, v. 14, n. 12, Dec. 1988.

[SCH93] SCHLEMER, R.A. **Integrated Process Support for large and Complex Development Projects**, Austin/Texas: ISSI, sep. 1993. (Documento Interno)

[SIL93] **SILK - Encap: user's manual**, release 1.0. Austin/TX: ISSI, 1993.

[SPC88] **SOFTWARE PRODUCTIVITY CONSORTIUM**. Function and Structure of an Expert System for Constructing Formal Specifications: A Pratical Example Using Formal Specifications, Apr. 1988.

[TAN89] TANIK, M.M.; YEH, R.T. Guest editor's introduction: Rapid prototyping in software development. **IEEE Computer**, v. 22, n.5, p.9-12, May 1989.

[WAS85] WASSERMAN, A. I. Extending state transition diagrams for the specification of human-computer interaction. **IEEE Transactions on Software Engineering**, v. 11, n. 8, Aug. 1985.

[WAT91] WATERS, R. The requirements apprentice: automated assistance for requirements acquisition. **IEEE Transactions on Software Engineering**, v. 17 n. 3, p. 226-240, Mar. 1991.

[YEH91] YEH, R.T et alli. A Comomsense management model, **IEEE Software**, p. 23-33, Nov. 1991.

[ZUC89] ZUCCONI, L. Techniques and experiences capturing requirements for several real-time aplications. **ACM SIGSOFT Software Engineering Notes**, v. 14, n. 6, p. 51-55, Oct. 1989.

Apêndice A

1. Lista de Questões dos CMIs e CMOs
2. Especificação de CMIs e CMOs

Apêndice A

1. Lista de questões dos CMOs e CMIs

QP1) Você está no ambiente FMS. As classes de aplicações conhecidas estão descritas abaixo. Entre com a sua opção.
R. <Classe> (Classe = Administração Rural)

QP2) Que aplicação você deseja desenvolver?
R. <Funcao>

QP3) Quais as informações necessárias para <Função>?
R. <Função.OE>

QP4) O que resulta de <Função>?
R. <Função.OS>

QP5) Descreva usando verbos e substantivos o que é <Função>.
R. <Função.Filhos>

QP6) Entre com a nova atividade que deseja descrever.
R. <Atividade>

QP7) Você estava descrevendo <Função> em <Função.Filhos>. A atividade <Atividade> ainda não foi descrita. Posicione-se na árvore de decomposição.
R. <Oper1>

QP8) Redirecionamento da entrevista. Entre com a atividade no qual <Atividade> faz parte.
R. <Root0>

QP9) <Função> já existe. Escolha uma das opções abaixo.
R. <operação> (operação = Atualizar, Deletar, Prosseguir)

QP10) Quais as práticas de controle de <Função>?
R. <Função.PControle>

QP11) Quais os relatórios necessários para <Função>?
R.. <Função.Relatórios>

QP12) O que identifica cada um dos (as) <Coletivo.Descrição> do <Coletivo.Coletivo>?
R. <Coletivo.Identificação>

QP13) Quais os possíveis valores ou componentes de <Operador2>?
R. <Operador2.Valores>

QP14) <Operador2.Valores> é(são) valor(es) ou componente(s) de <Operador2>?
R. <Operador2.Descrição>

QP15) O valor de <Val.Value> pode ser mudado para um mesmo <Coletivo.Identificação>?
R. <Val.Valores> (val.Valores = sim, não)

QP16) Entre com o título para o formulário <Form>.
R. <Form.Titulo>

QP17) Entre com um help para o formulário <Form>.
R. <Form.Help>

QP18) Entre com o tipo do formulário <Form>.
R. <Form.Tipo>

QP19) Quais os campos do formulário <Form>?
R. <Form.Campos >

ou

QP19) Foi identificado o(s) seguinte(s) campo(s) para o formulário <Form>: <Operador2.Campos>. Entre com uma das opções abaixo.
R. <confirm> (confirm = Inserir, Confirmar)

QP20) Quais os textos do formulário <Form>?
R. <Form.Textos >

ou

QP20) Foi identificado o(s) seguinte(s) texto(s) para o formulário <Form>: <Operador2.Campos>. Entre com uma das opções abaixo.
R. <confirm1> (confirm1 = Inserir, Confirmar)

QP21) Entre com uma descrição para o campo <Campo>.
R. <Campo.Descr>

QP22) Entre com o modo do campo <Campo>.
R. <Campo.Modo>

QP23) Entre com o tipo do campo <Campo>.
R. <Campo.Tipo>

QP24) Entre com um help para o campo <Campo>.
R. <Campo.Help>

QP25) Entre com o valor mínimo para este campo <Campo>.
R. <Campo.Inter_Min>

QP26) Entre com o valor máximo para este campo <Campo>.
R. <Campo.Inter_Max>

QP27) Quais os valores do choice <Campo>?
R. <Campo.Lista>

QP28) Quantas casas decimais tem o campo <Campo>?
R. <Campo.Casas_Dec>

QP29) Qual o comprimento do campo <Campo>?
R. <Campo.Comprim>

QP30) Quando ou como passar de <Est_Inicial> para <Est_Final>?
R. <Evento>

QP31) Quais as condições necessárias para que ocorra um(a) <Evento>?
R. <Evento.Regras_Disparo>

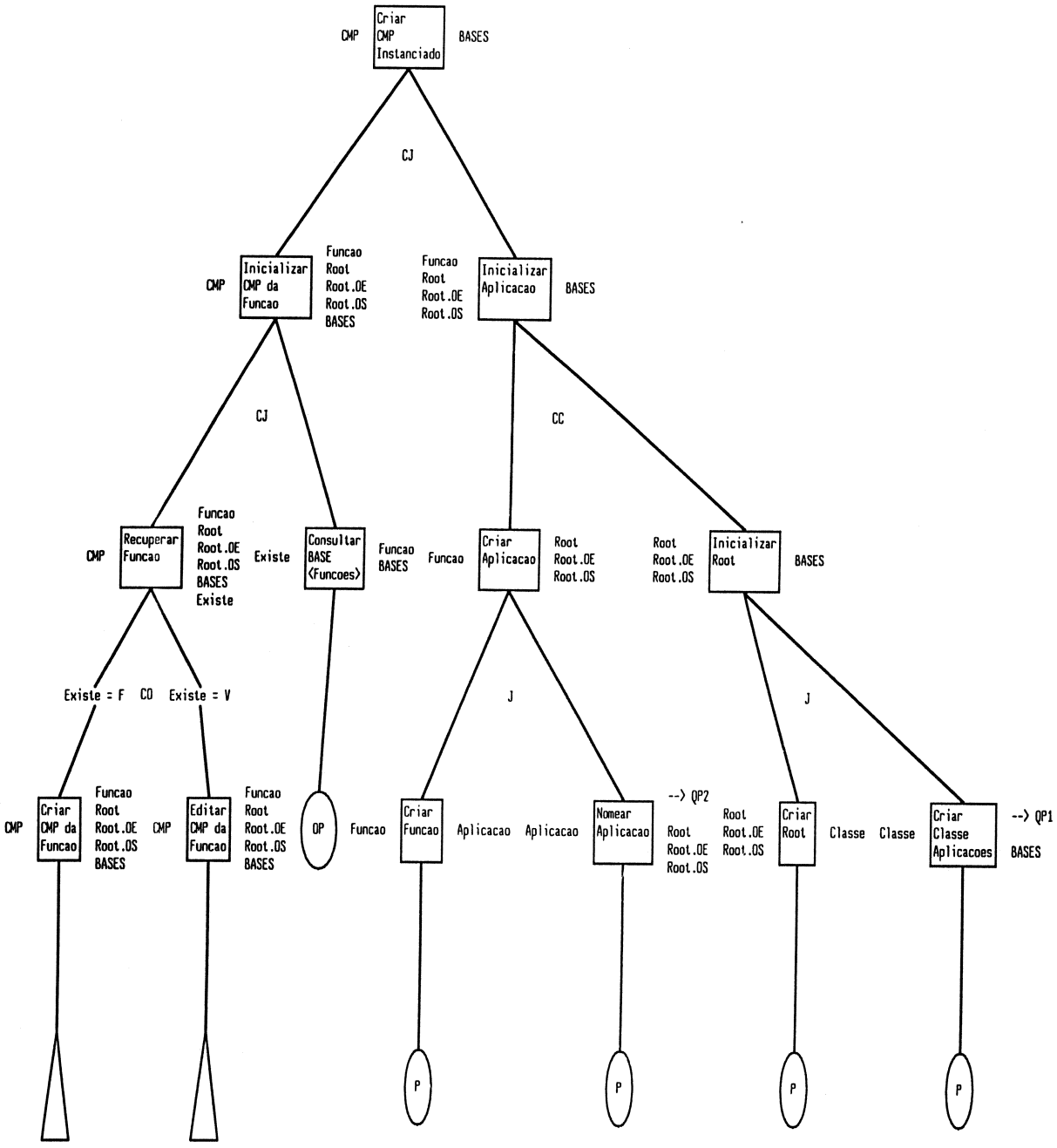
QP32) Quais as ações disparadas pelo(a) <Evento>?
R. <Evento.Acoes_Disparadas>

QP33) Descreva as próximas ocorrências quando um <Evento> é confirmado.
R. <Evento.Eventos_Disparados>

QD1) Entre com o nome no qual <Dado> faz parte.
R. <Dado.Pai>

2. Especificação de CMIs e CMOs

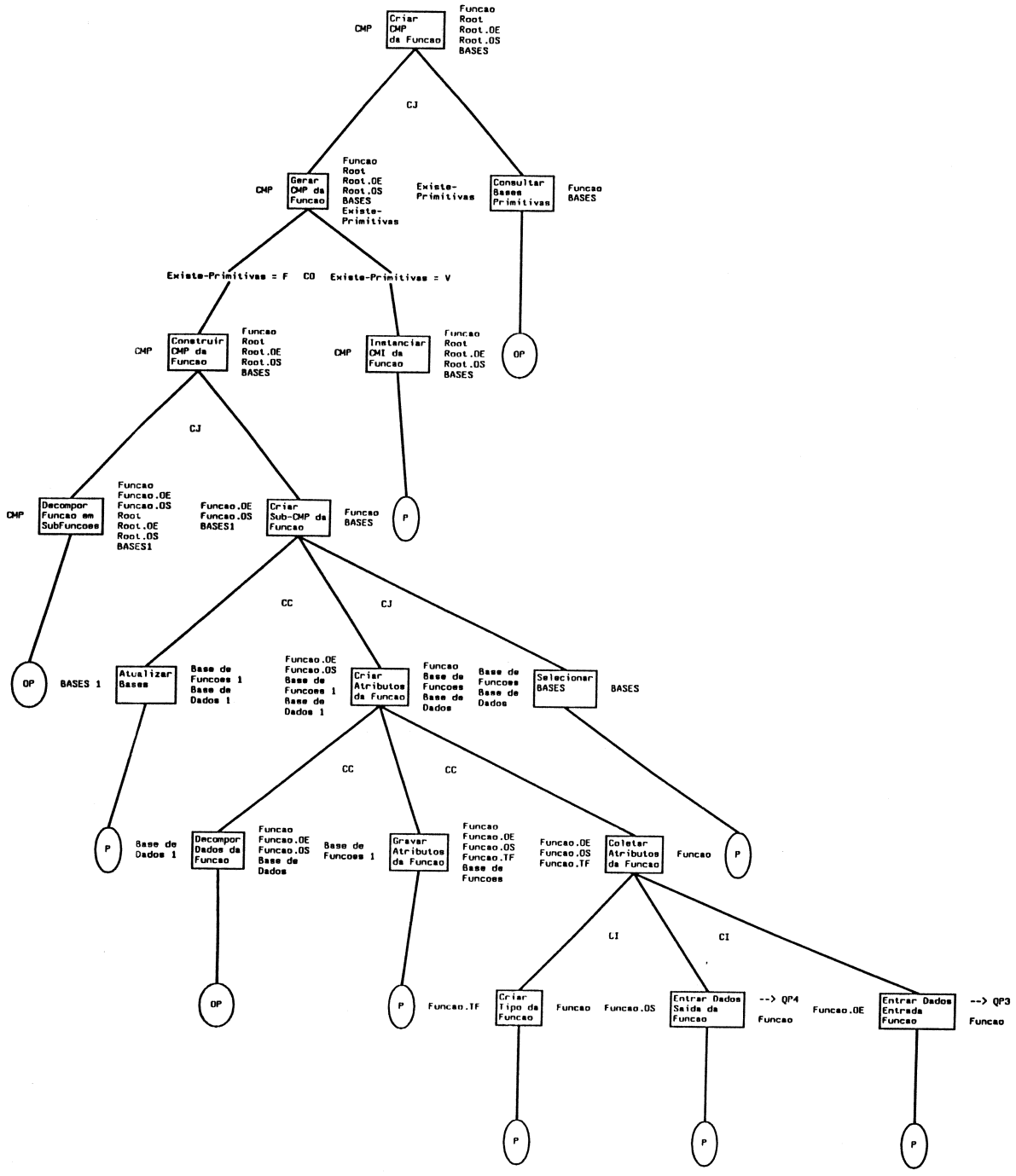
CMI - Criar CMP Instanciado



QP1) Você está no ambiente FMS. As classes de aplicações conhecidas estão descritas abaixo. Entre com a sua opção.
R. <Classe> (Classe = Administração Rural)

QP2) Que aplicação você deseja desenvolver?
R. <Funcao>

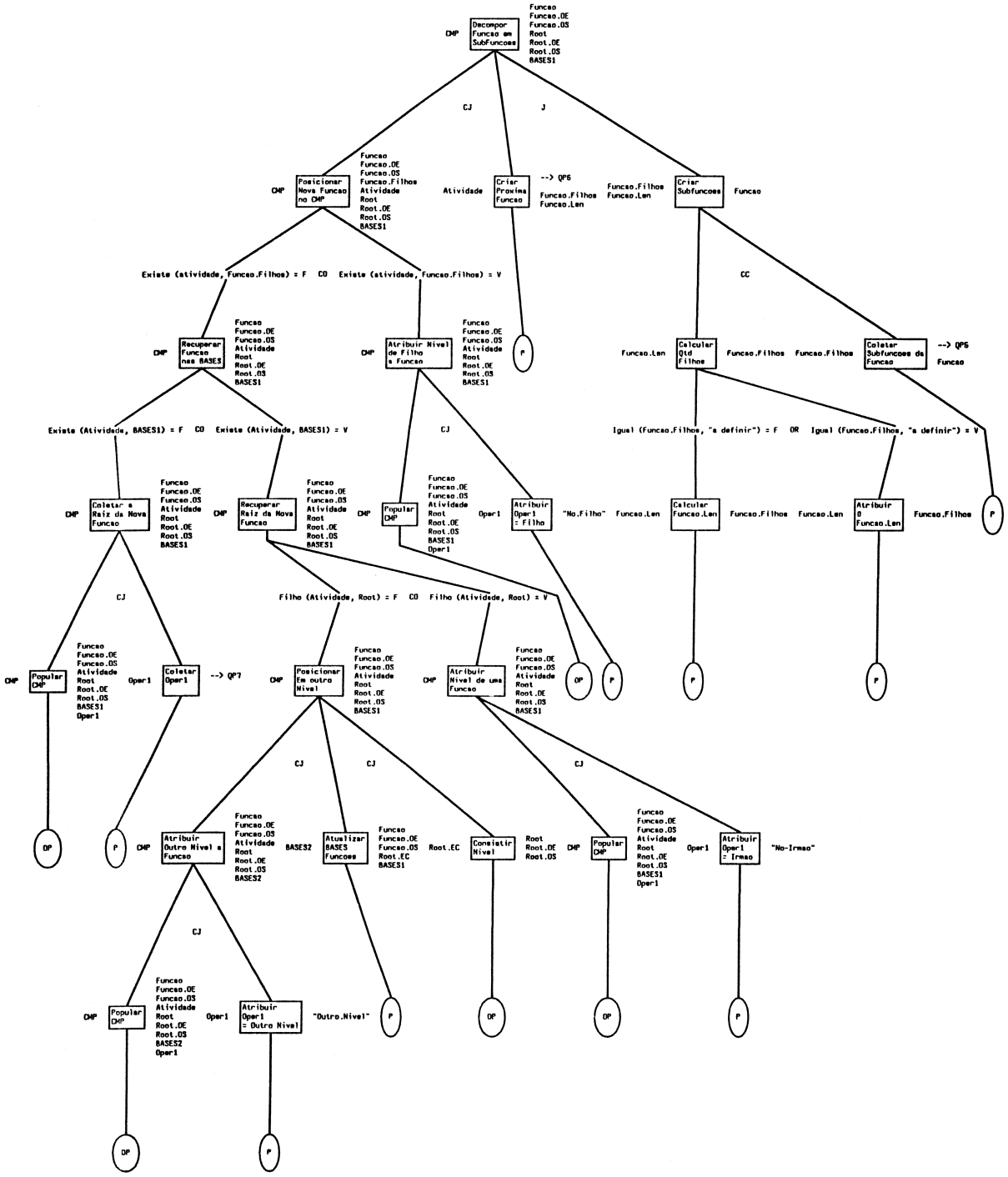
Criar CMP da Função
(continuação do CMI - Criar CMP Instanciado)



QP3) Quais as informações necessárias para <Função>?
R. <Função.OE>

QP4) O que resulta de <Função>?
R. <Função.OS>

CMO - Decompor Função em Subfunções

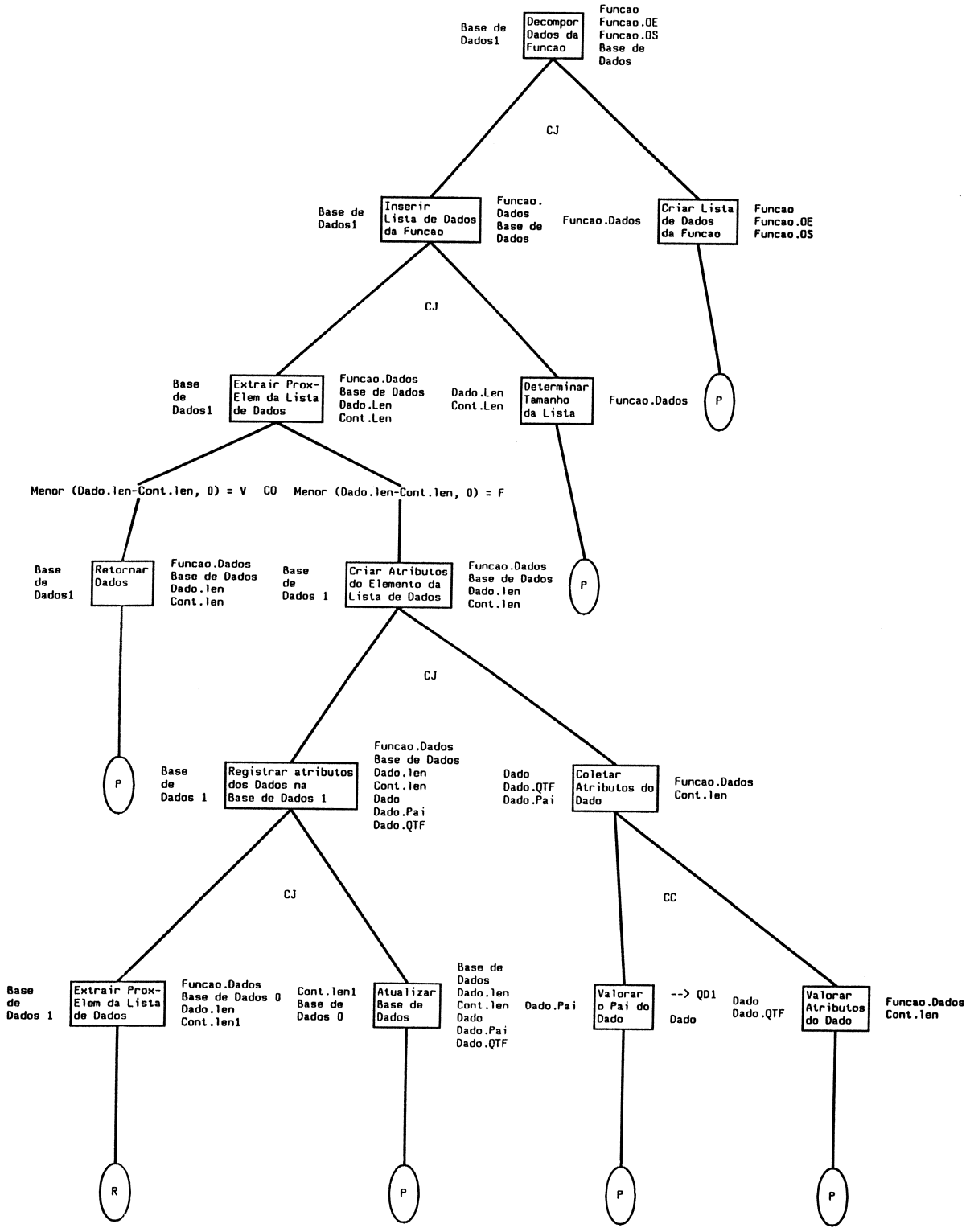


QP5) Descreva usando verbos e substantivos o que é <Função>.
R. <Função.Filhos>

QP6) Entre com a nova atividade que deseja descrever.
R. <Atividade>

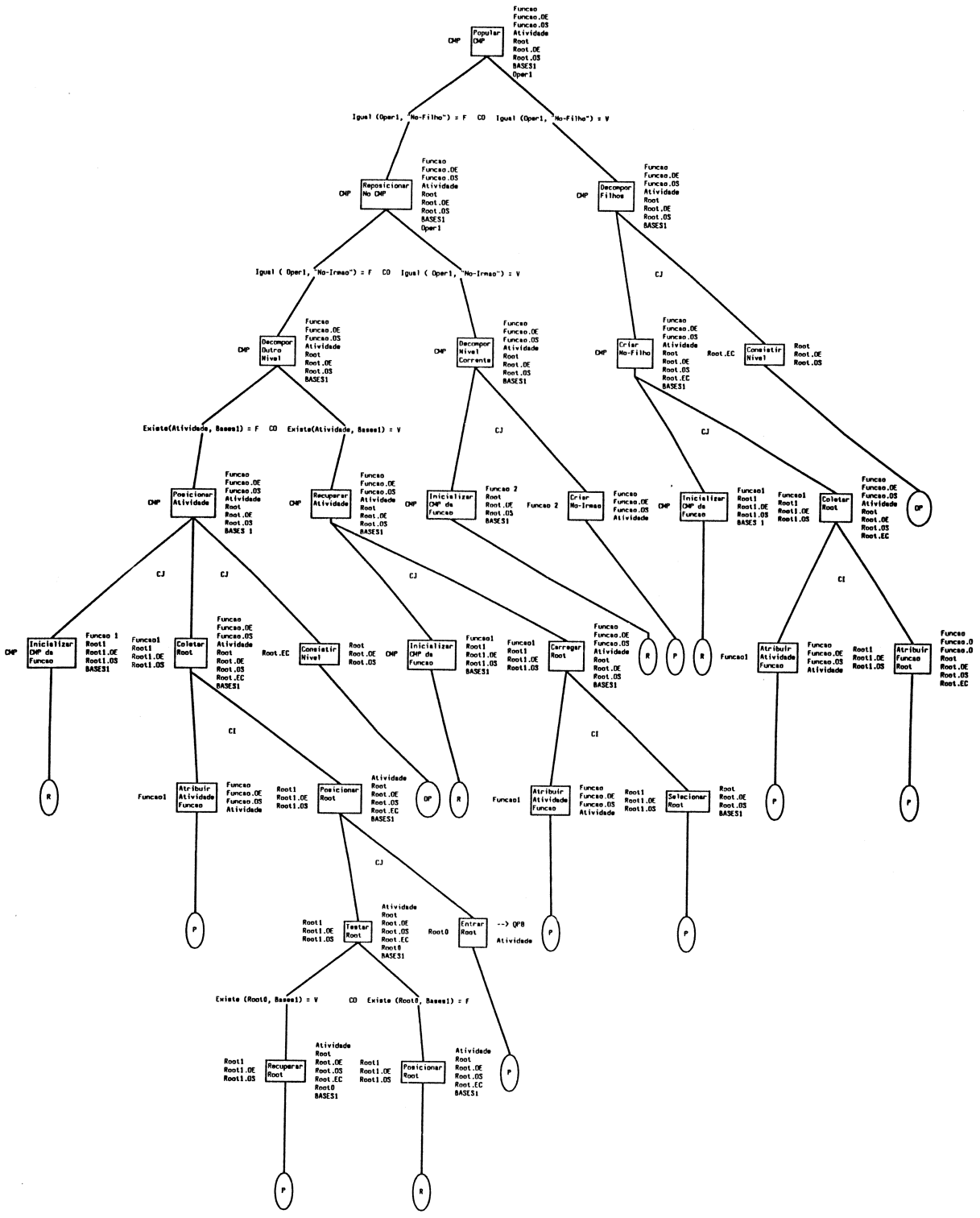
QP7) Você estava descrevendo <Função> em <Função.Filhos>. A atividade <Atividade> ainda não foi descrita. Posicione-se na árvore de decomposição.
R. <Oper1>

CMO - Decompor Dados da Função



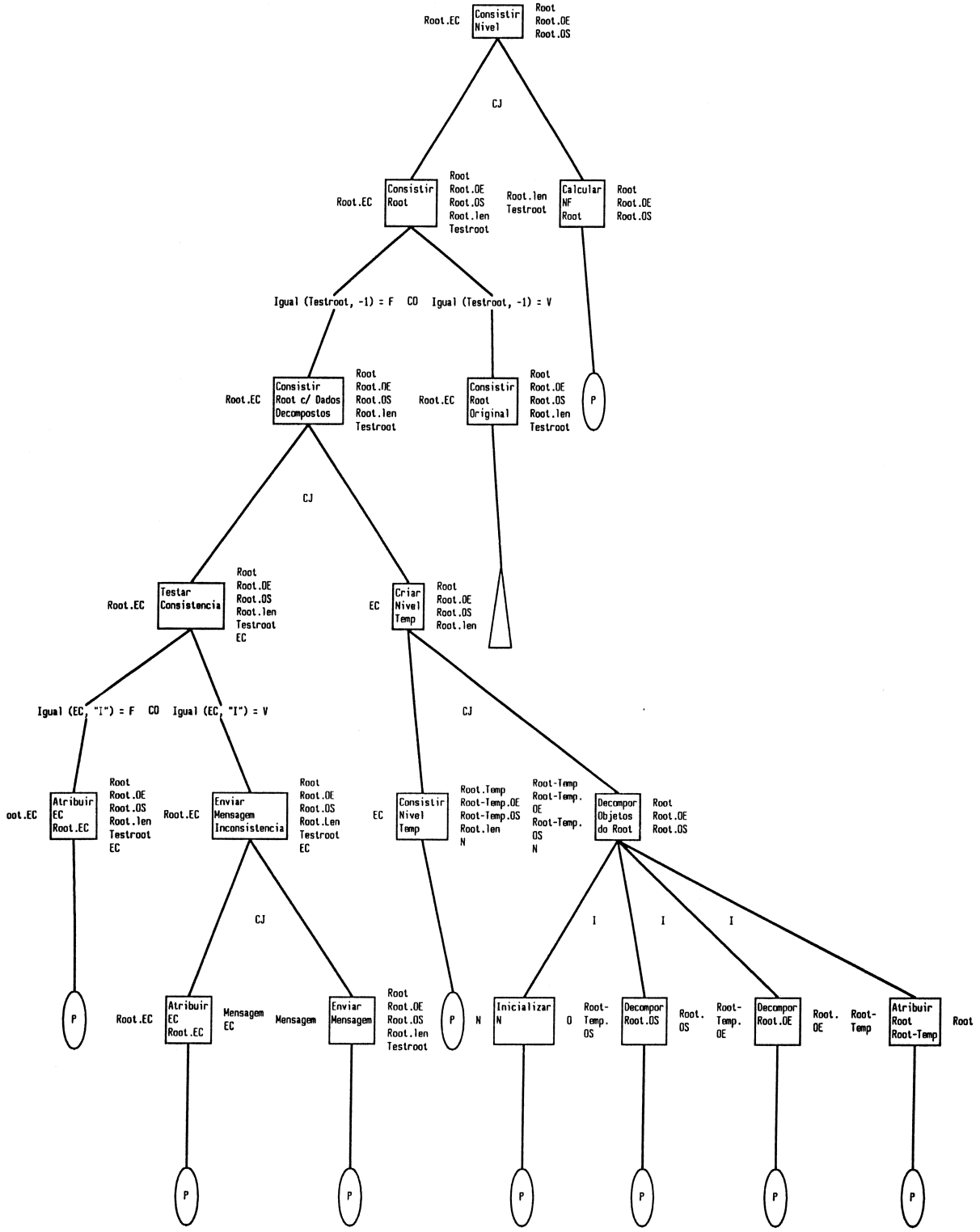
QD1) Entre com o nome no qual <Dado> faz parte.
R. <Dado.Pai>

CMO - Popular CMP



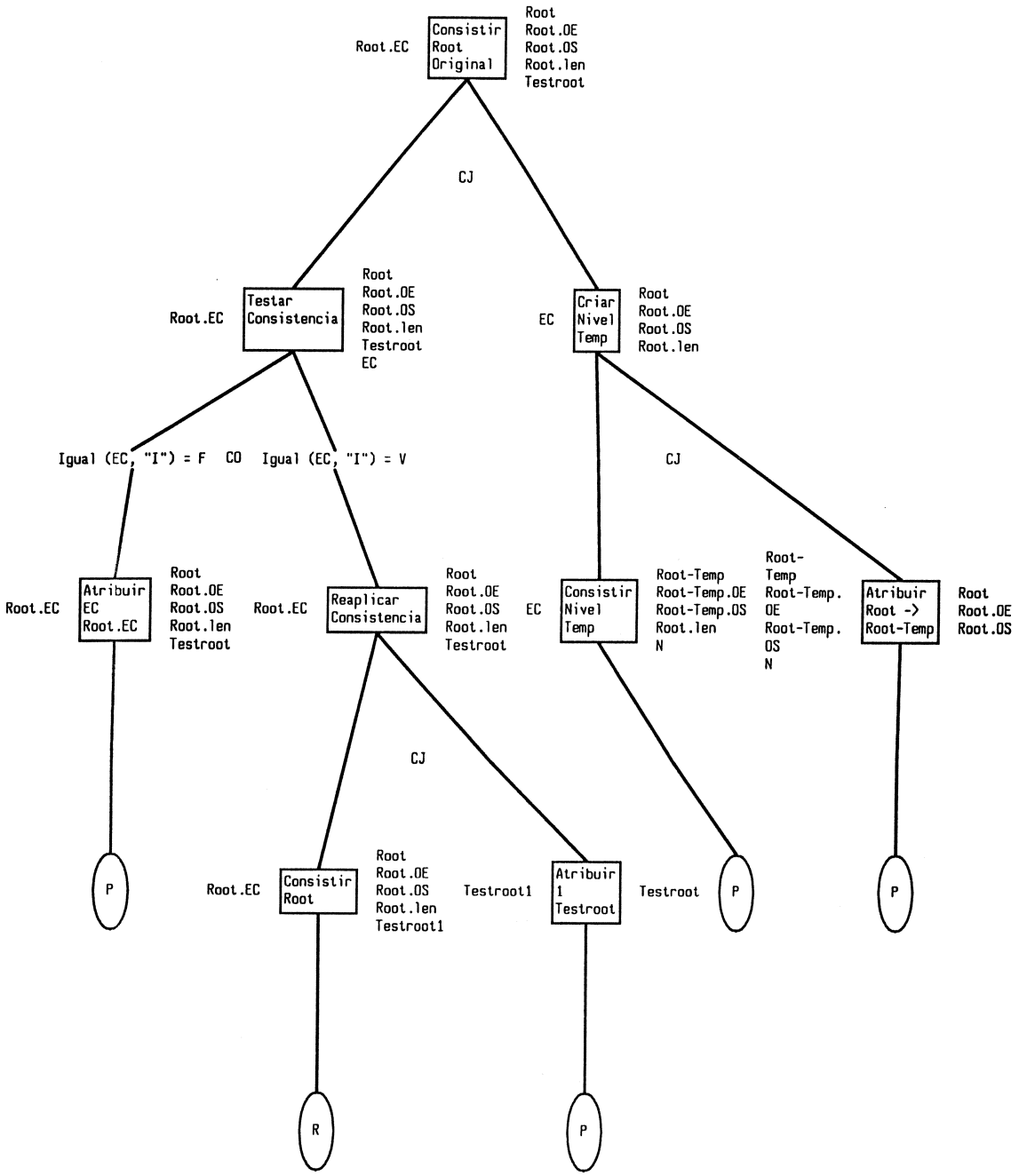
QP8) Redirecionamento da entrevista. Entre com a atividade no qual <Atividade> faz parte.
R. <Root0>

CMO - Consistir Nível

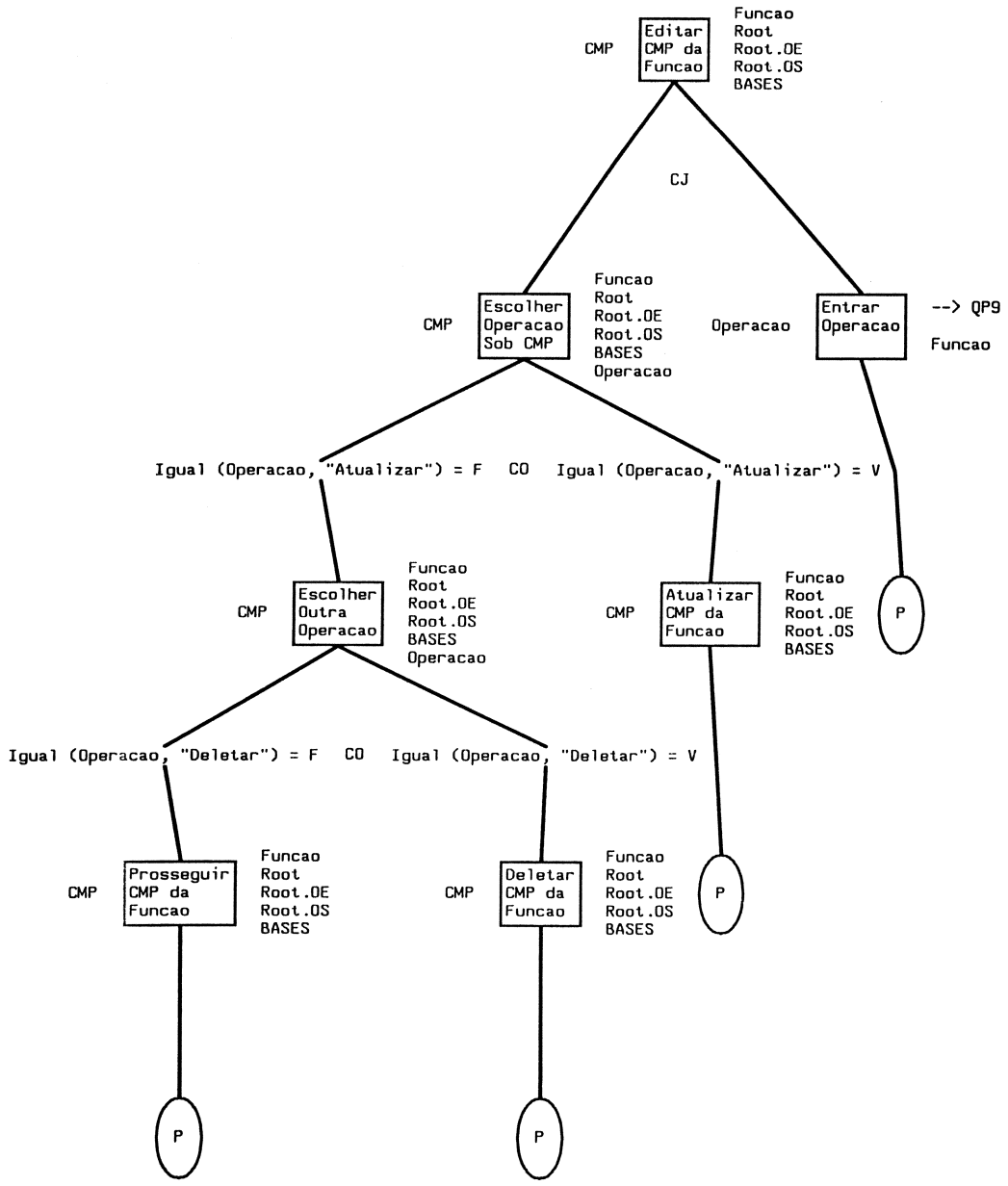


Consistir Root Original

(continuação do CMO - Consistir Nível)

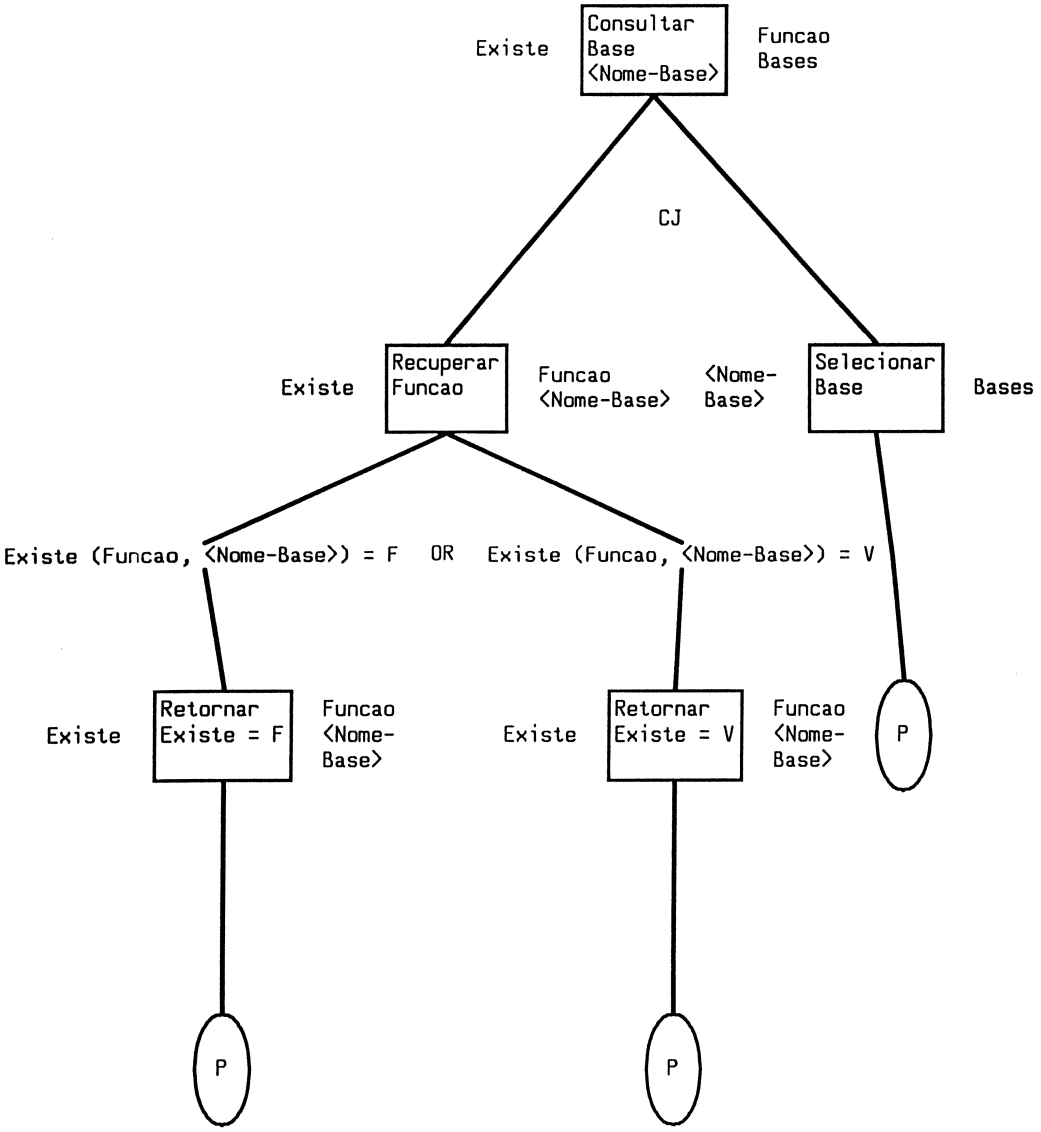


Editar CMP da Função
(continuação do CMI - Criar CMP Instanciado)

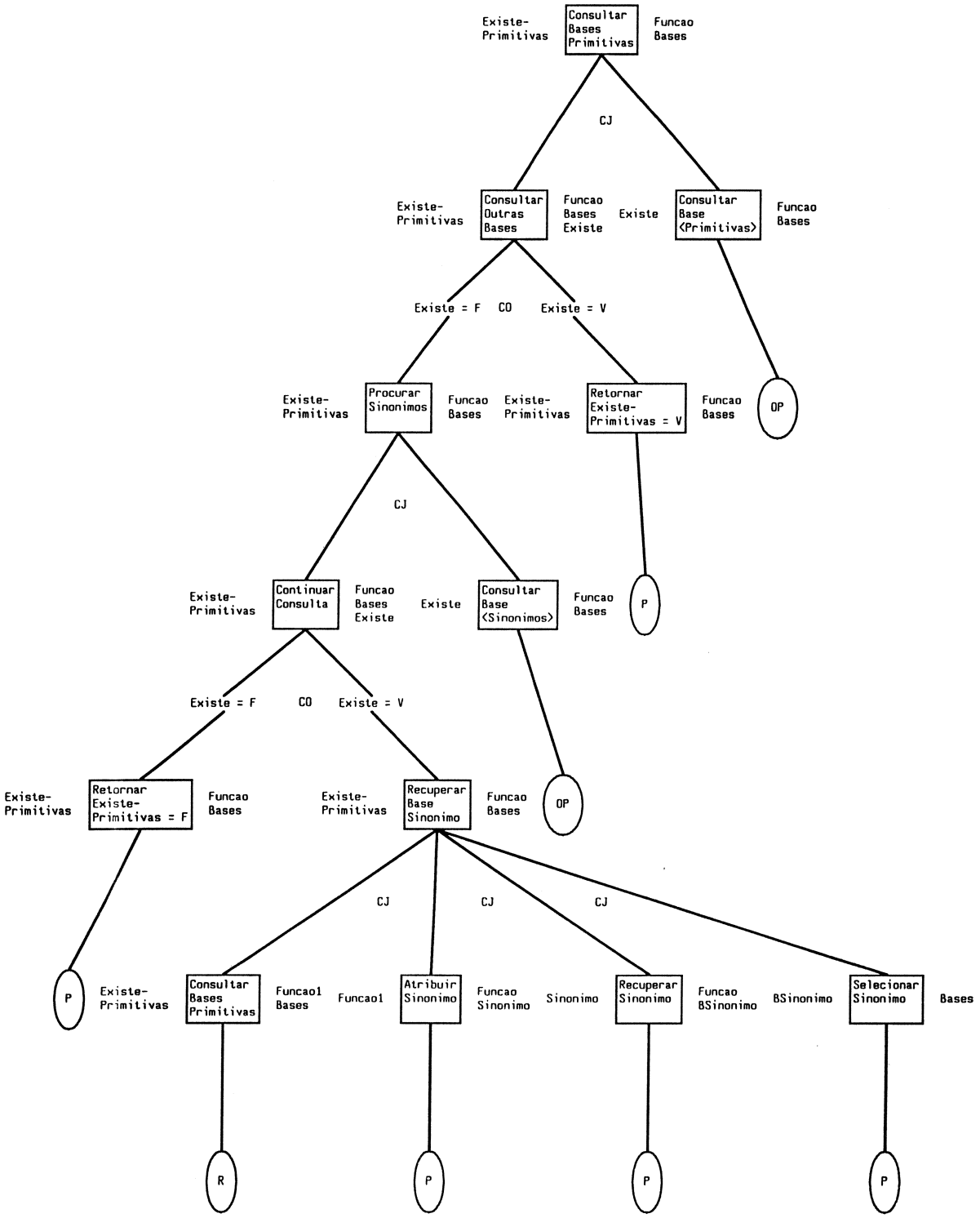


QP9) <Função> já existe. Escolha uma das opções abaixo.
R. <operação> (operação = Atualizar, Deletar, Prosseguir)

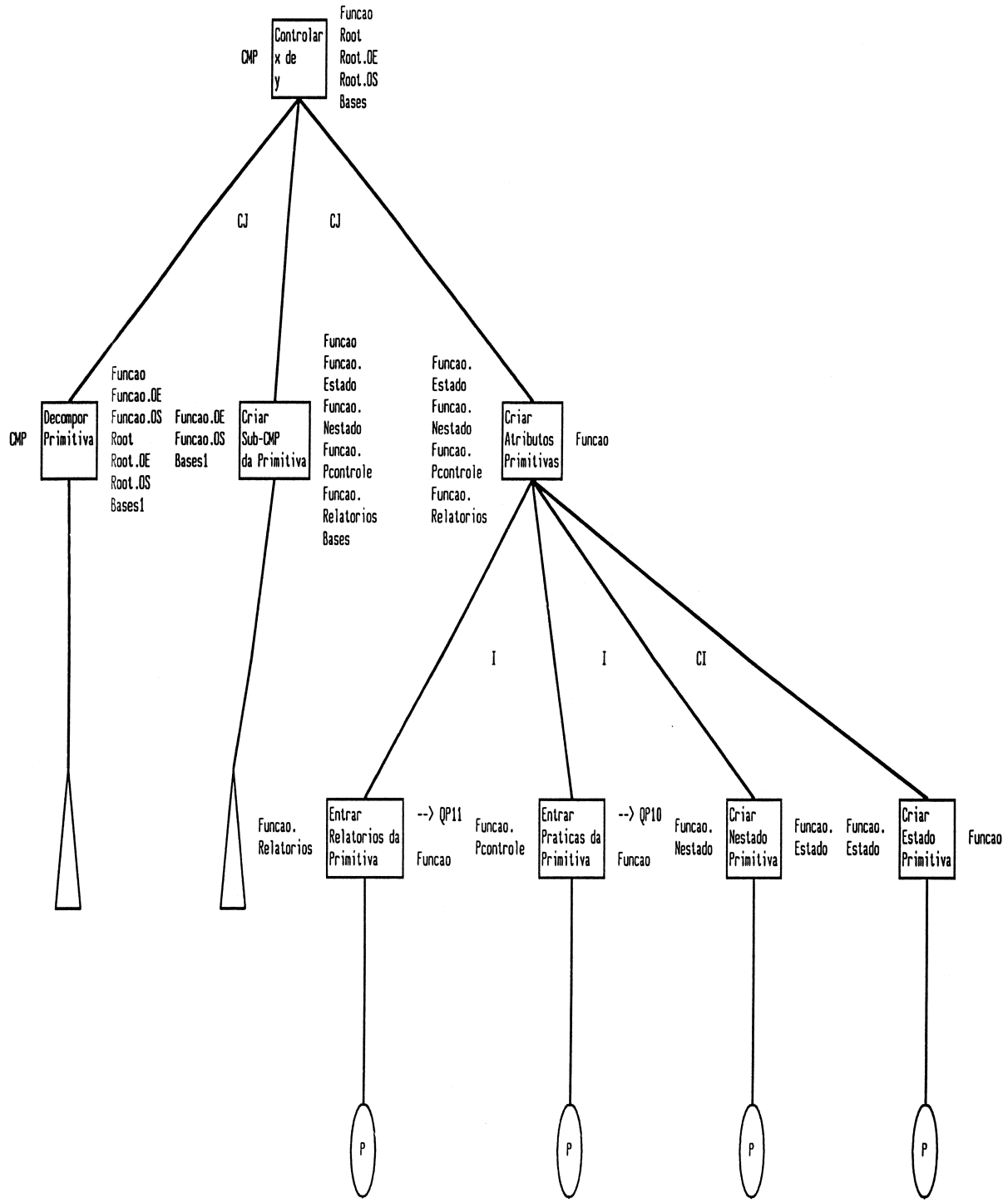
CMO - Consultar Base <Nome-Base>



CMO - Consultar Bases Primitivas



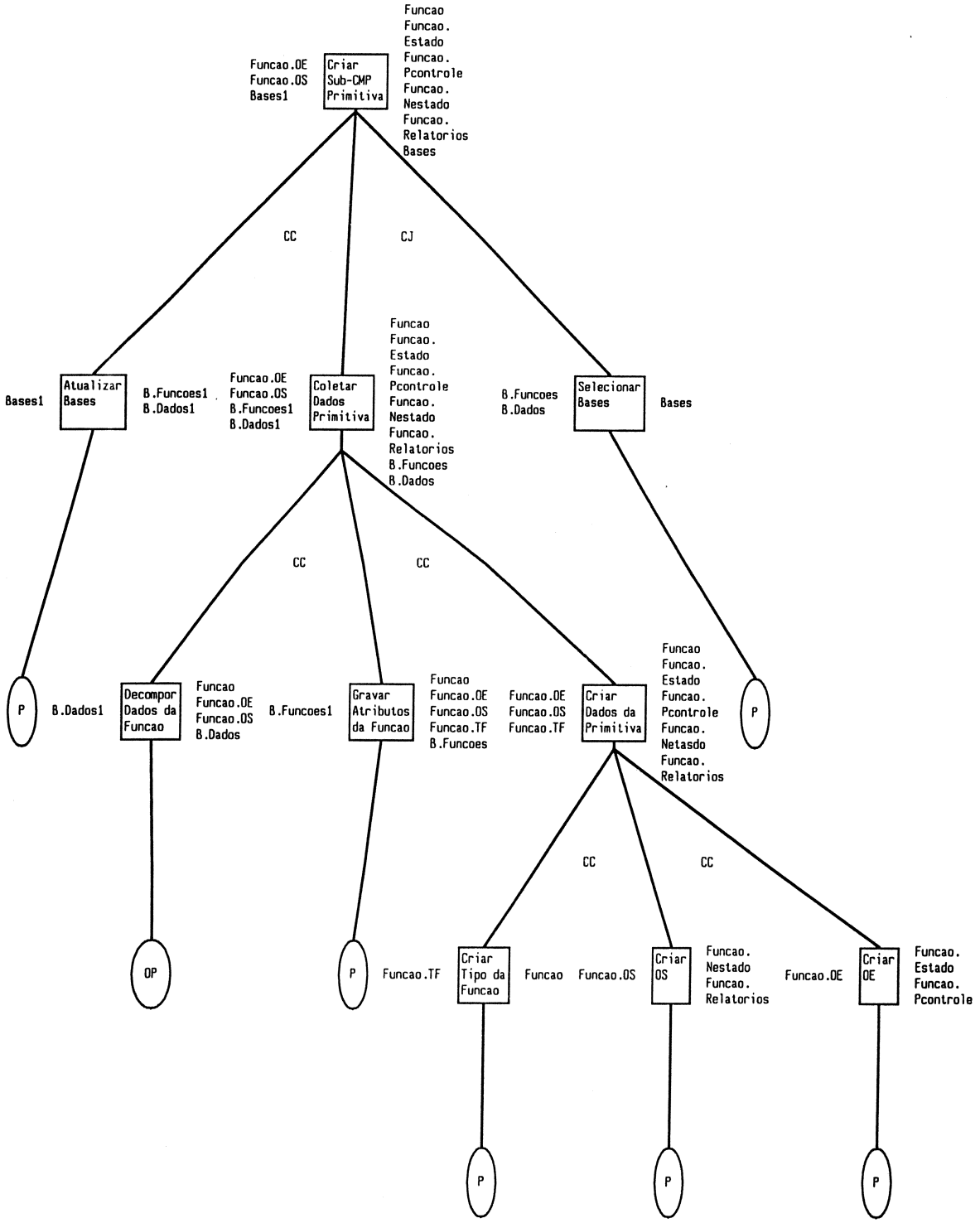
CMI - "Controlar"



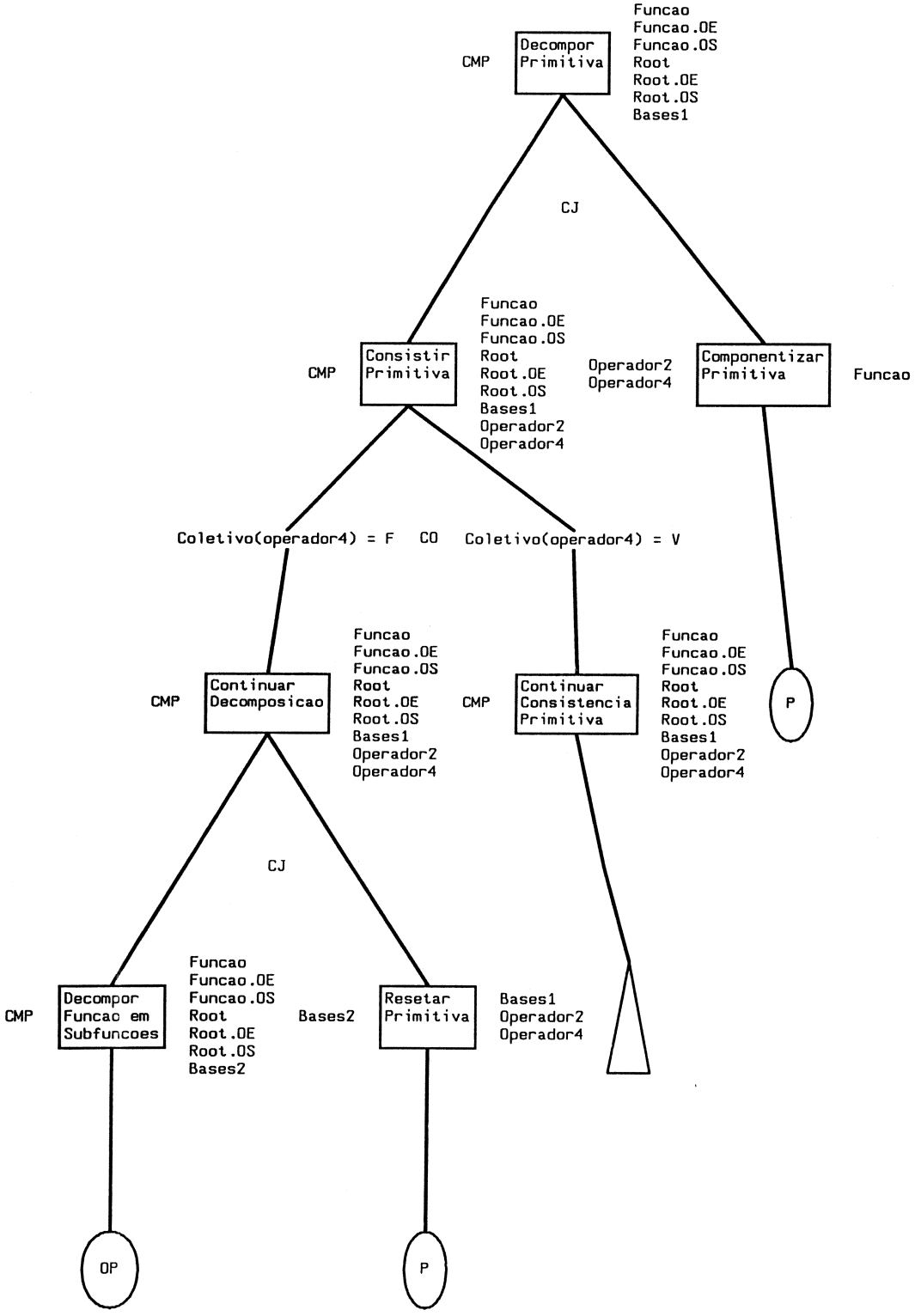
QP10) Quais as práticas de controle de <Função>?
R. <Função.PControle>

QP11) Quais os relatórios necessários para <Função>?
R.. <Função.Relatórios>

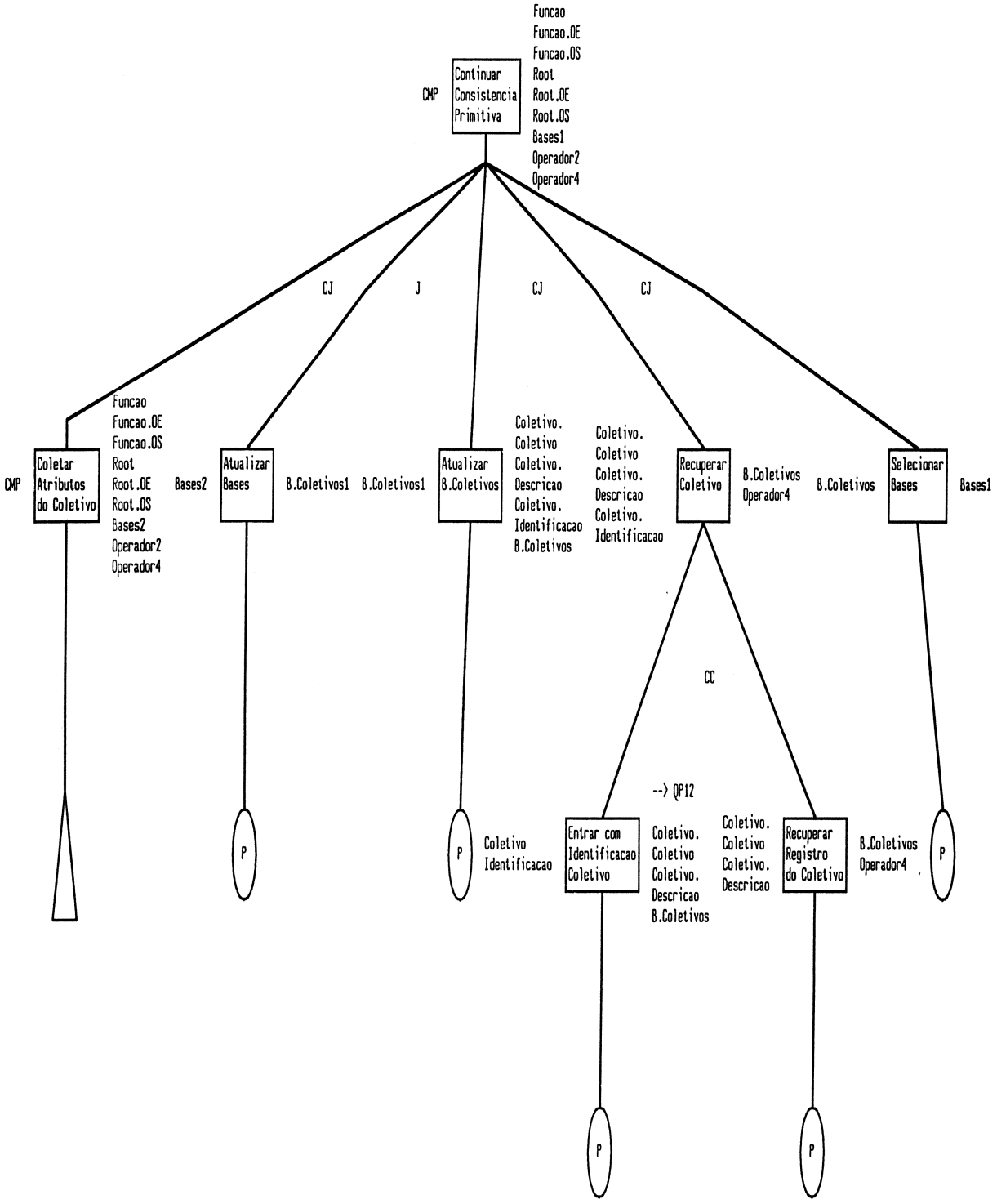
Criar Sub-CMP da Primitiva
(continuação do CMI - "Controlar")



Decompor Primitiva
(continuação do CMI - "Controlar")

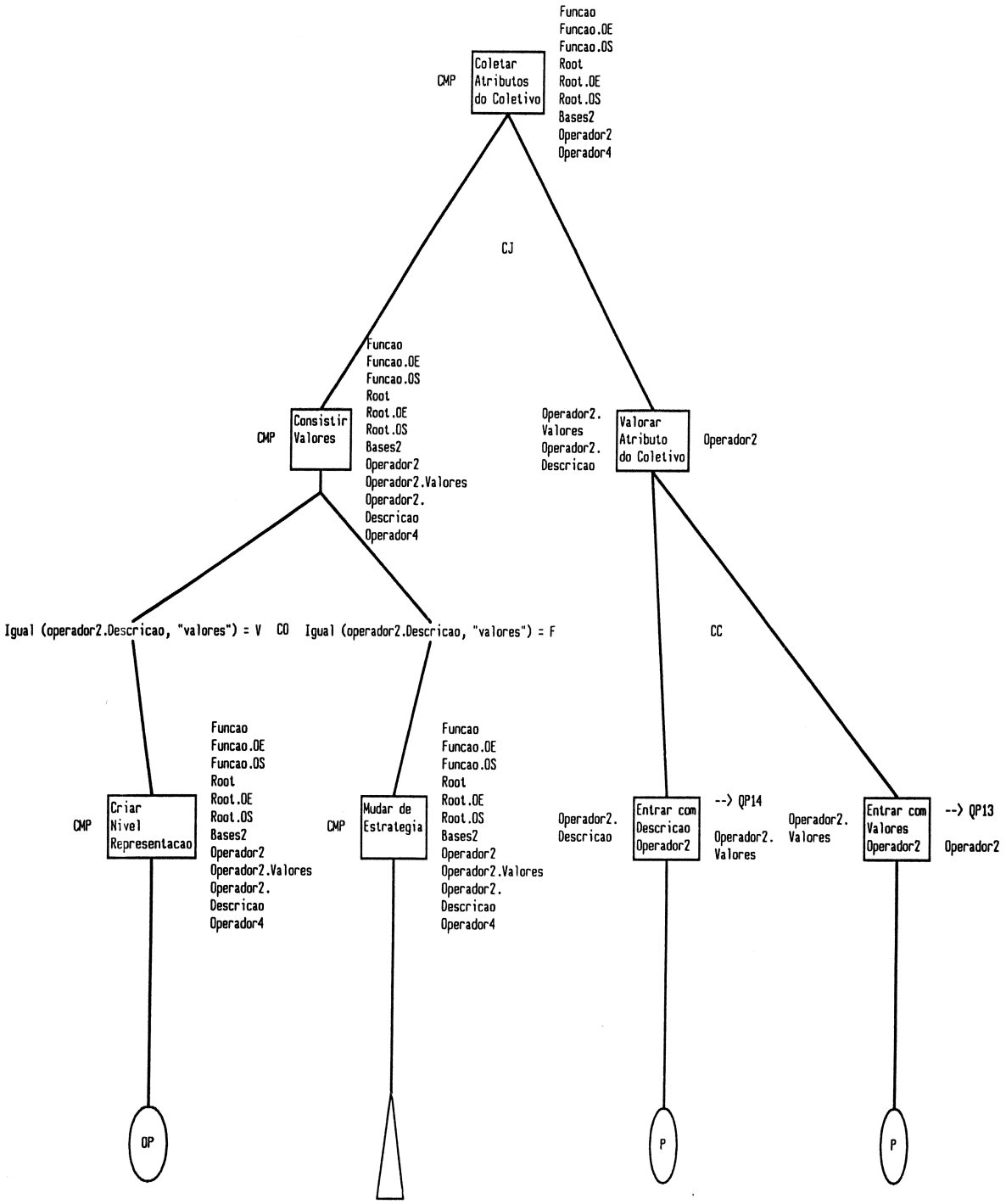


Continuar Consistência Primitiva
(continuação do CMI - "Controlar")



QP12) O que identifica cada um dos (as) <Coletivo.Descrição> do <Coletivo.Coletivo>?
R. <Coletivo.Identificação>

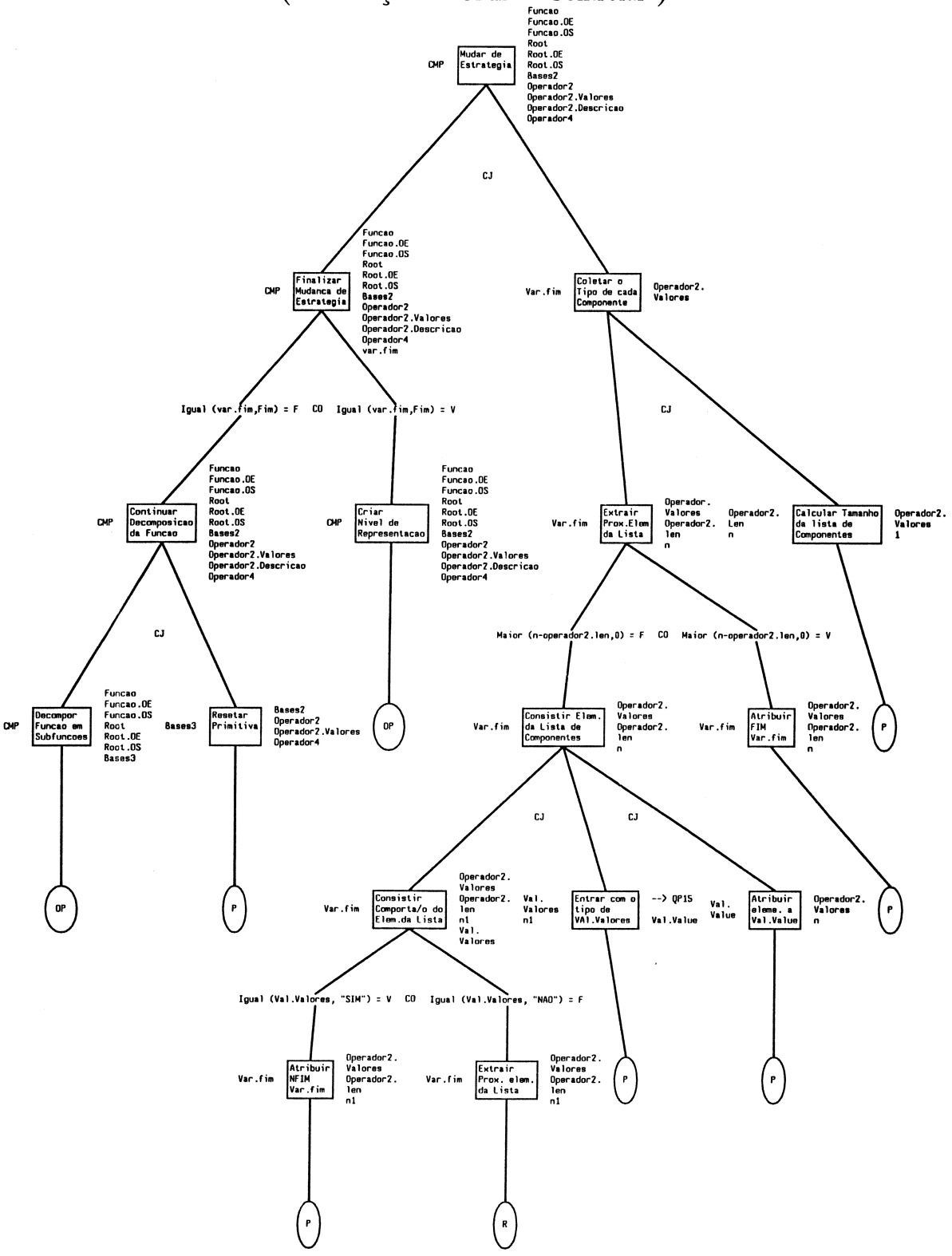
Coletar Atributos do Coletivo
(continuação do CMI - "Controlar")



QP13) Quais os possíveis valores ou componentes de <Operador2>?
R. <Operador2.Valores>

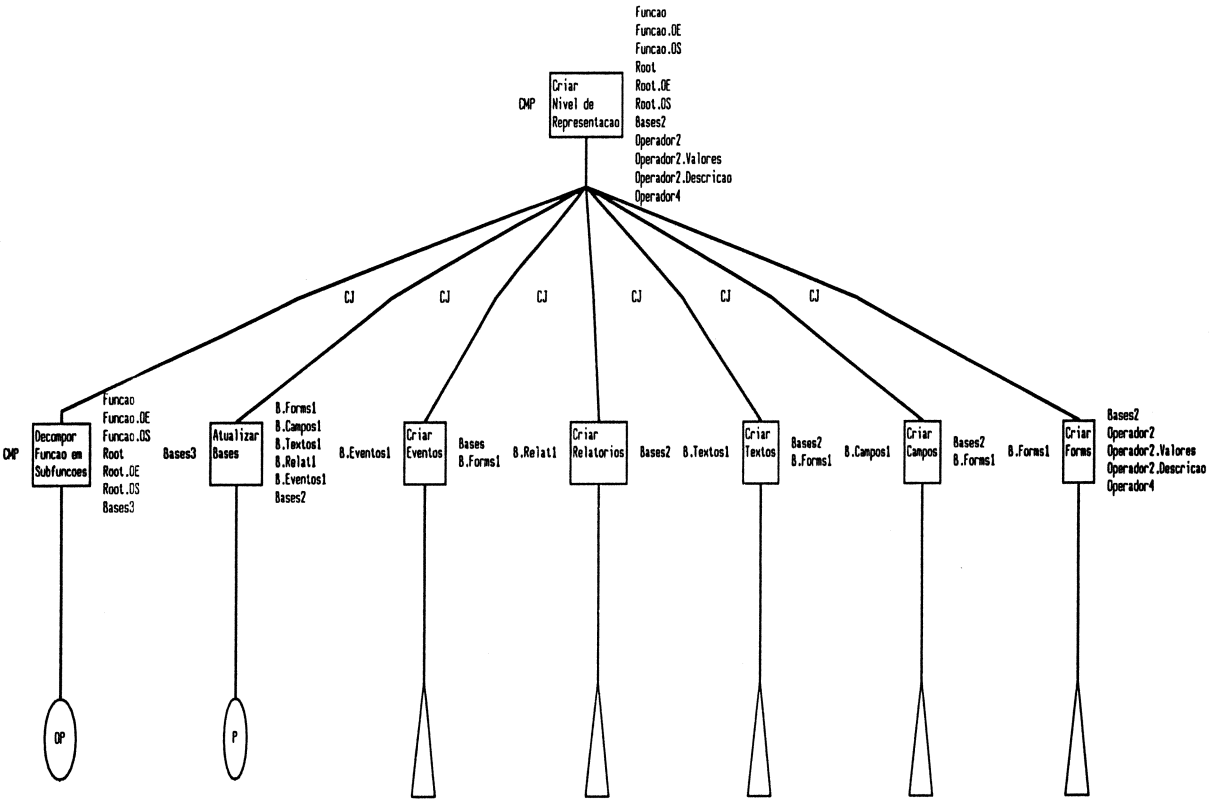
QP14) <Operador2.Valores> é(são) valor(es) ou componente(s) de <Operador2>?
R. <Operador2.Descrição>

Mudar de Estratégia
(continuação do CMI - "Controlar")

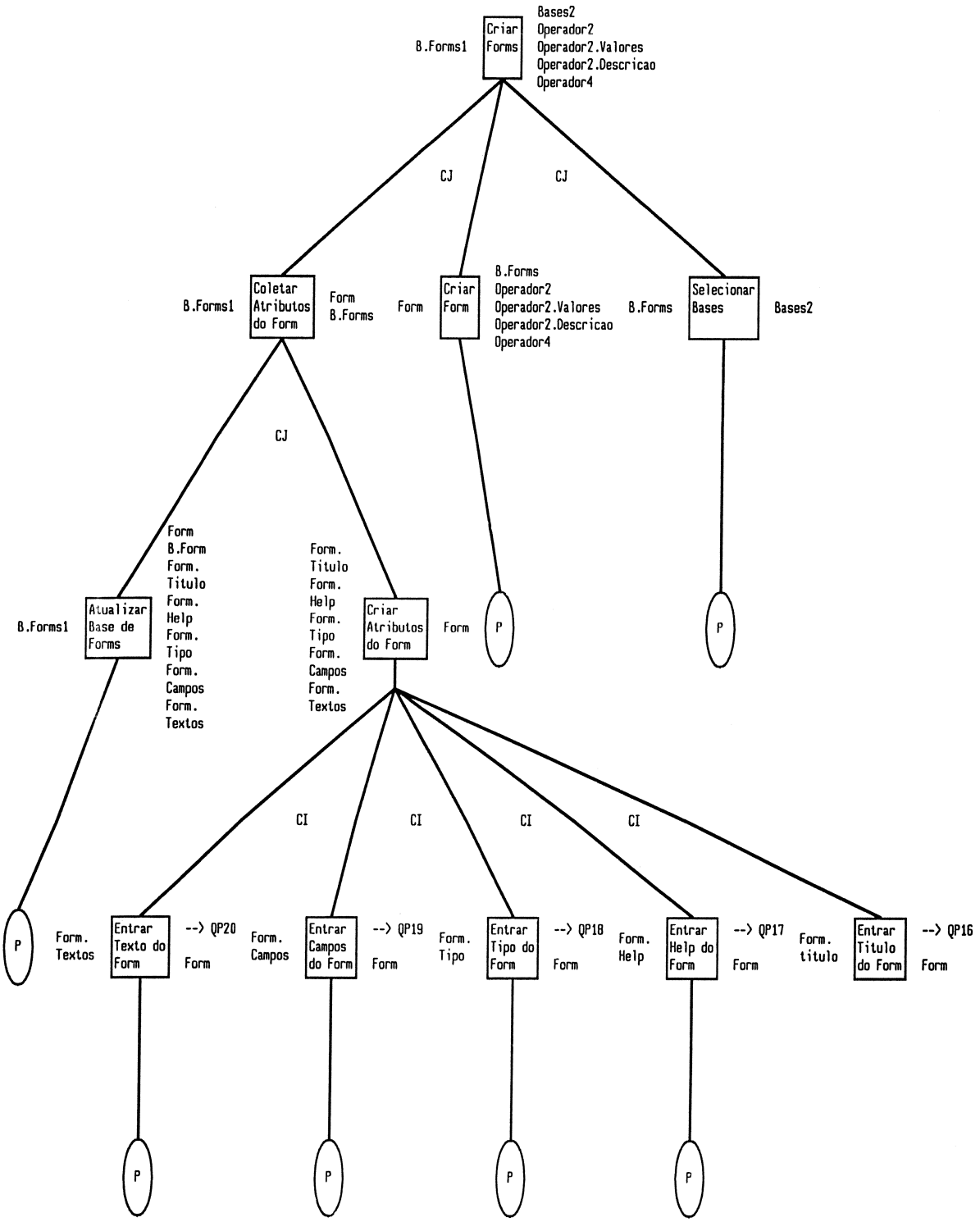


QP15) O valor de <Val.Value> pode ser mudado para um mesmo <Coletivo.Identificação>?
R. <Val.Valores> (val.Valores = sim, não)

CMO - Criar Nível de Representação



Criar Forms
(continuação do CMO - Criar Nível de Representação)



QP16) Entre com o título para o formulário <Form>.
R. <Form.Titulo>

QP17) Entre com um help para o formulário <Form>.
R. <Form.Help>

QP18) Entre com o tipo do formulário <Form>.
R. <Form.Tipo>

QP19) Quais os campos do formulário <Form>?
R. <Form.Campos >

ou

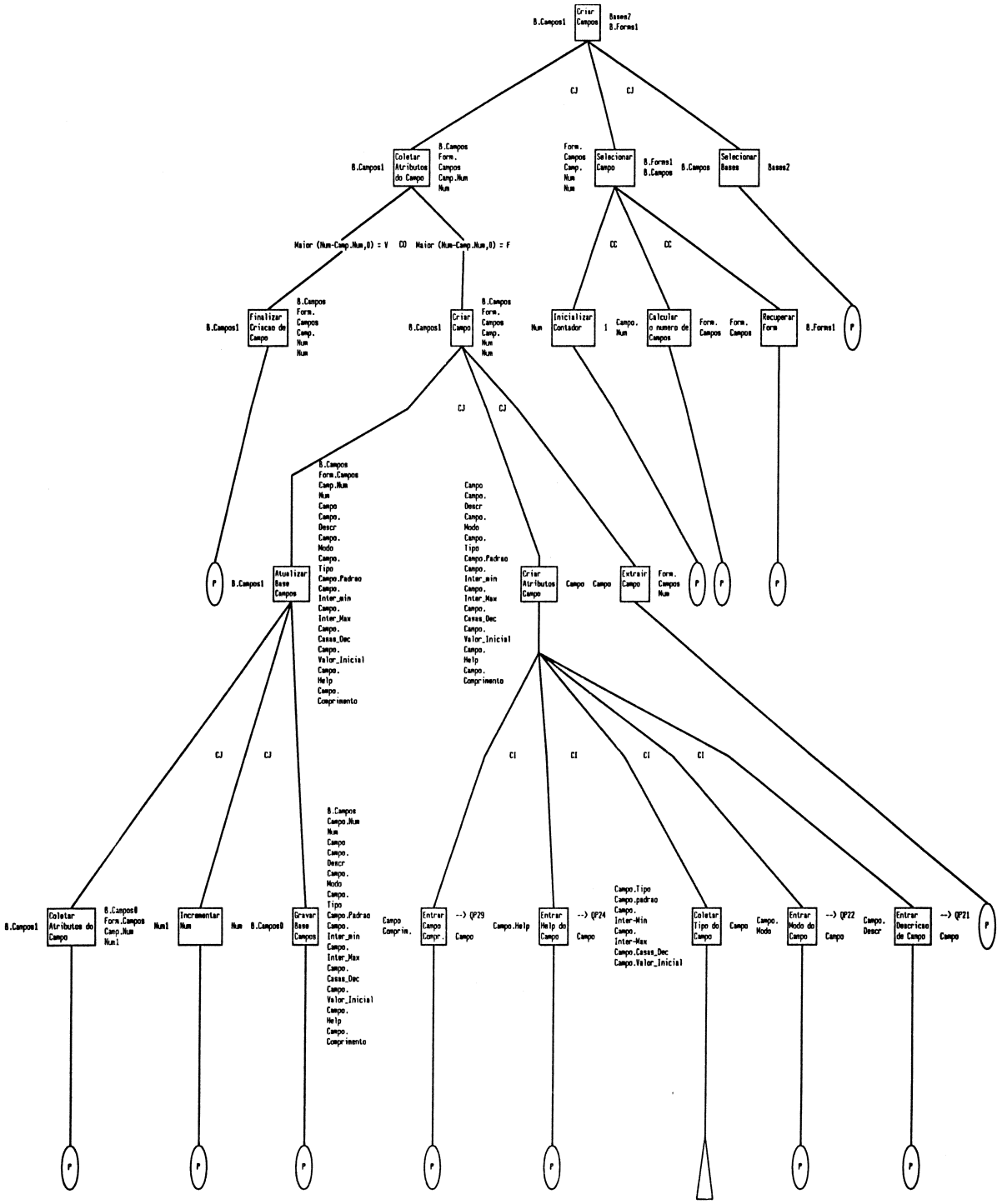
QP19) Foi identificado o(s) seguinte(s) campo(s) para o formulário <Form>:
<Operador2.Campos>. Entre com uma das opções abaixo.
R. <confirm> (confirm = Inserir, Confirmar)

QP20) Quais os textos do formulário <Form>?
R. <Form.Textos >

ou

QP20) Foi identificado o(s) seguinte(s) texto(s) para o formulário <Form>:
<Operador2.Campos>. Entre com uma das opções abaixo.
R. <confirm1> (confirm1 = Inserir, Confirmar)

Criar Campos
(continuação do CMO - Criar Nível de Representação)



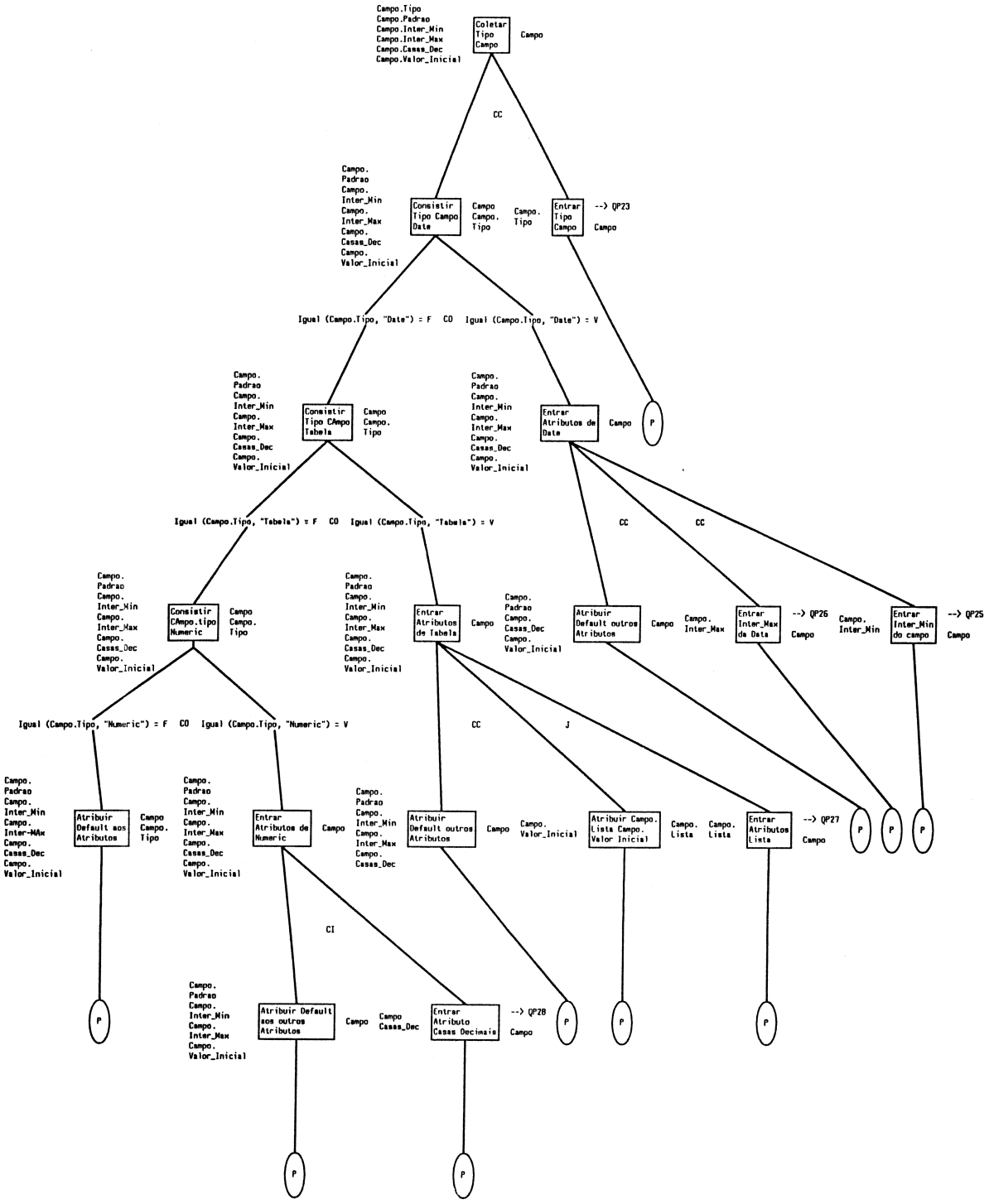
QP21) Entre com uma descrição para o campo <Campo>.
R. <Campo.Descr>

QP22) Entre com o modo do campo <Campo>.
R. <Campo.Modo>

QP24) Entre com um help para o campo <Campo>.
R. <Campo.Help>

QP29) Qual o comprimento do campo <Campo>?
R. <Campo.Comprim>

Coletar Tipo Campo
(continuação do CMO - Criar Nível de Representação)



QP23) Entre com o tipo do campo <Campo>.
R. <Campo.Tipo>

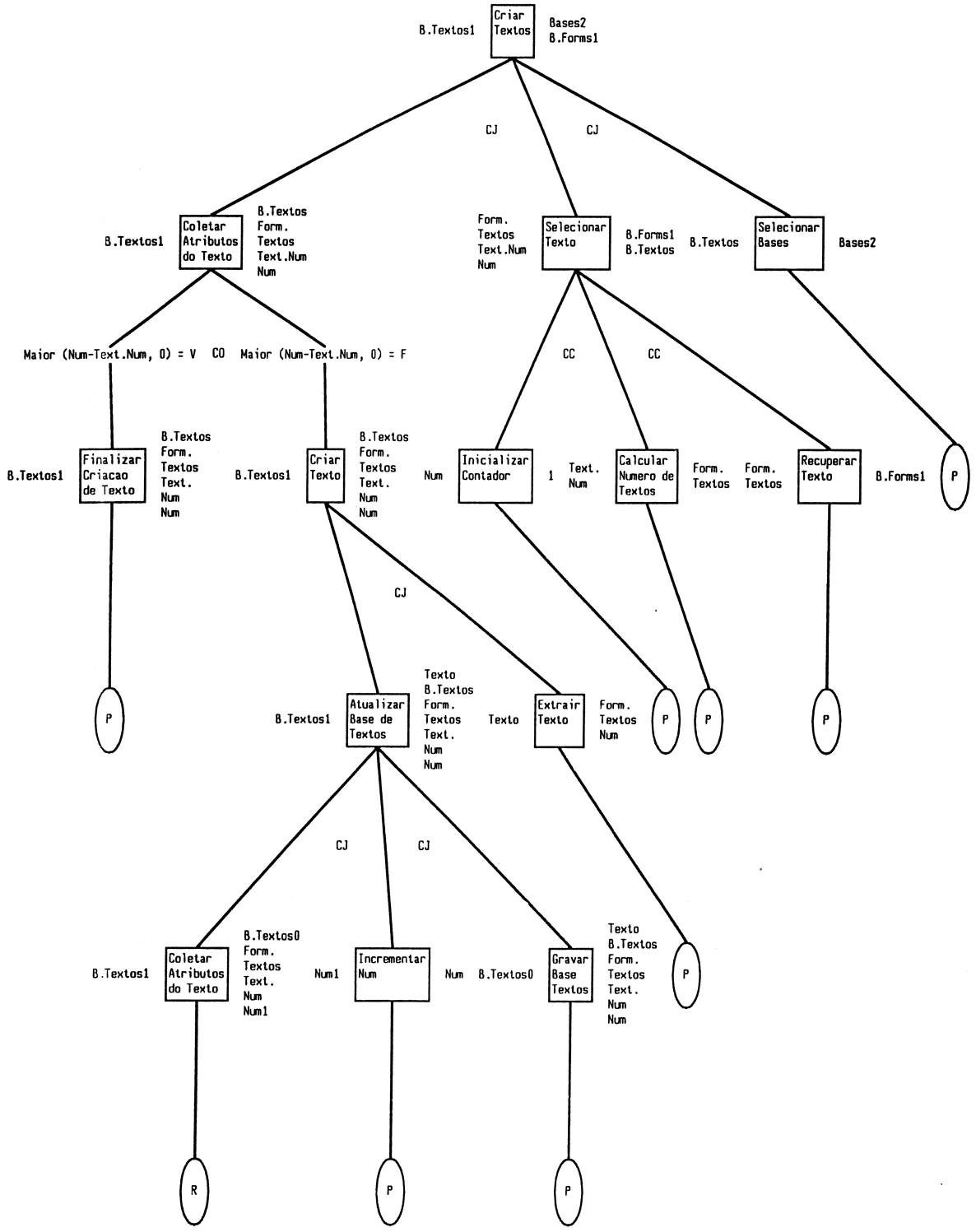
QP25) Entre com o valor mínimo para este campo <Campo>.
R. <Campo.Inter_Min>

QP26) Entre com o valor máximo para este campo <Campo>.
R. <Campo.Inter_Max>

QP27) Quais os valores do choice <Campo>?
R. <Campo.Lista>

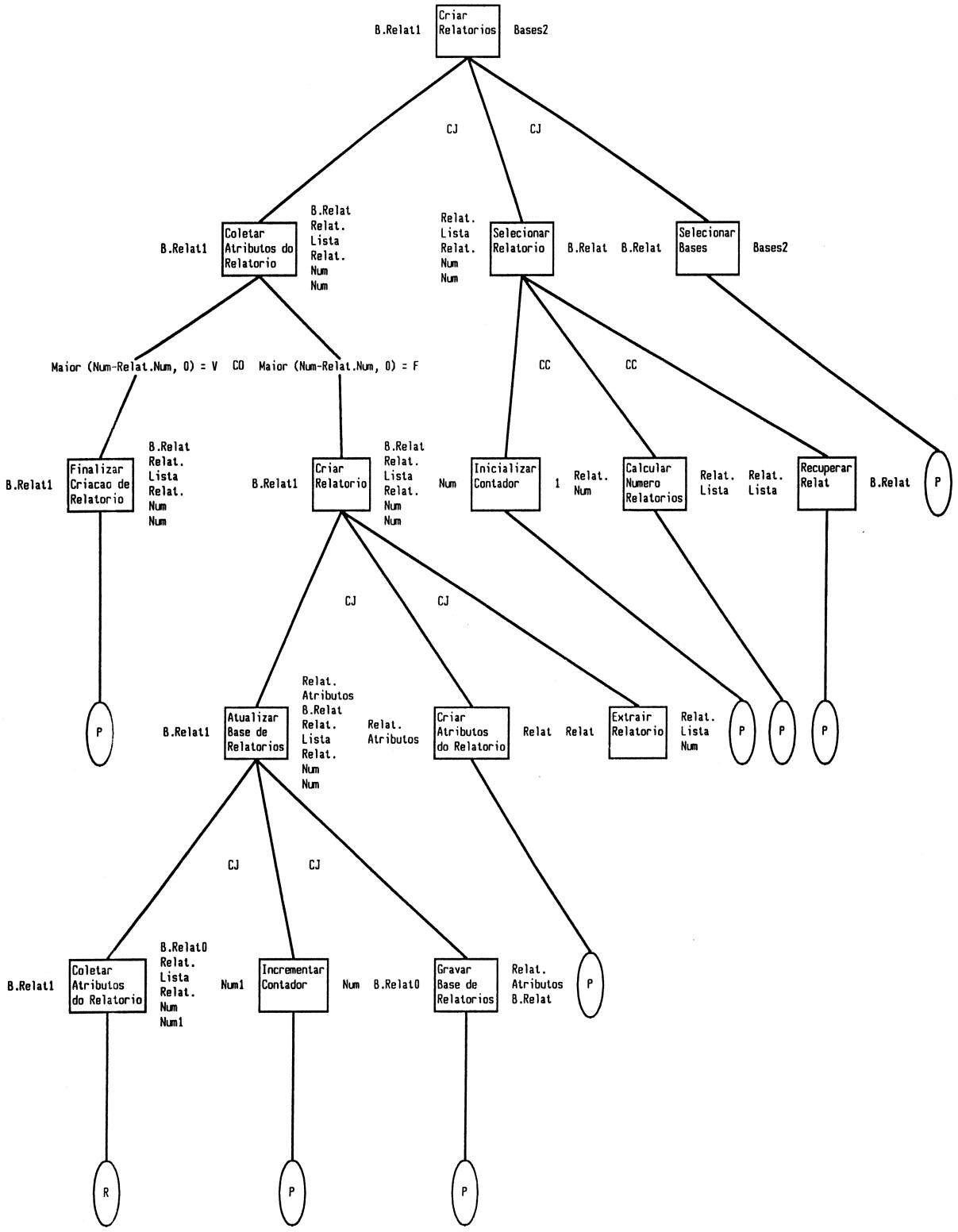
QP28) Quantas casas decimais tem o campo <Campo>?
R. <Campo.Casas_Dec>

Criar Textos
(continuação do CMO - Criar Nível de Representação)



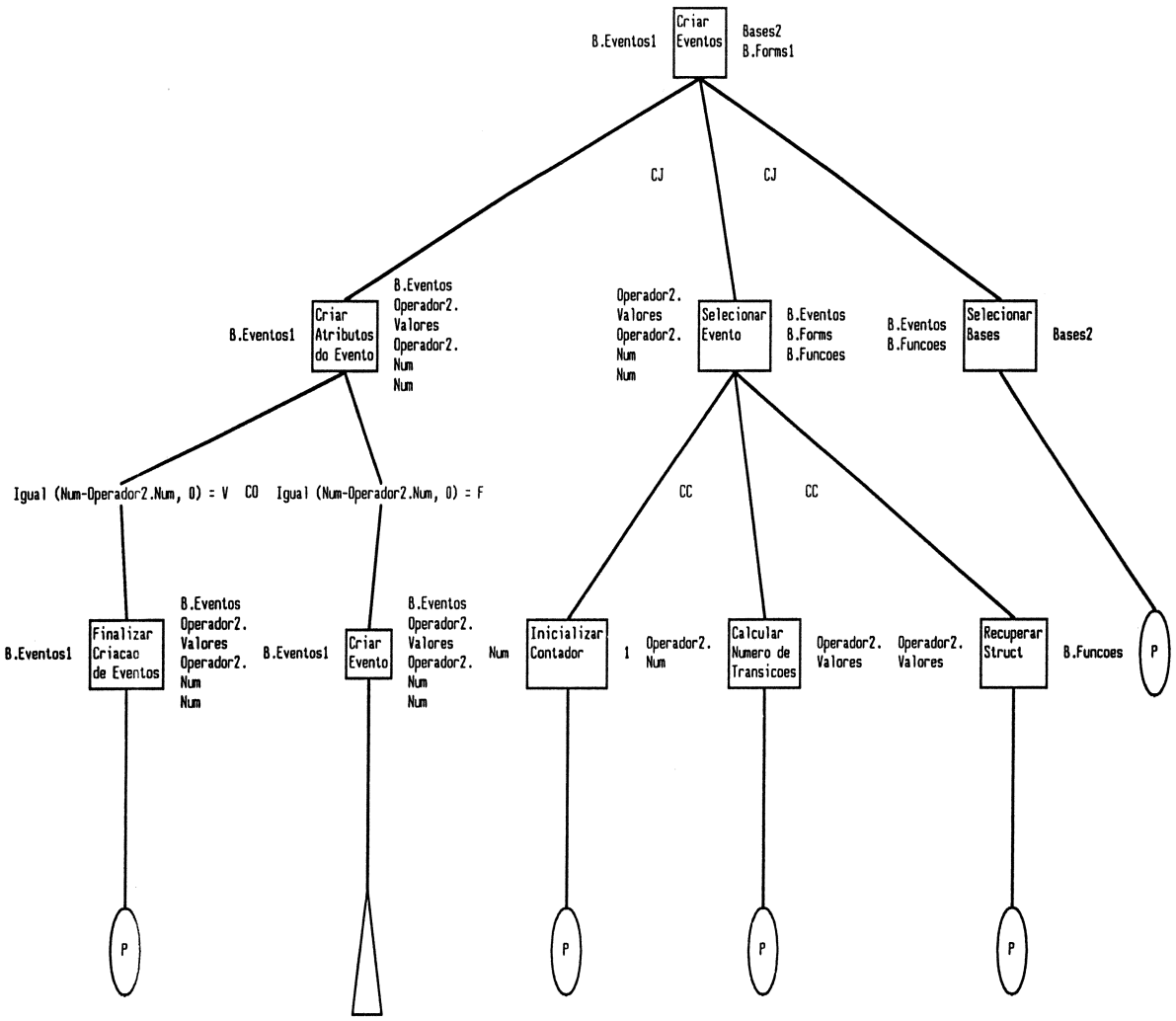
Criar Relatórios

(continuação do CMO - Criar Nível de Representação)

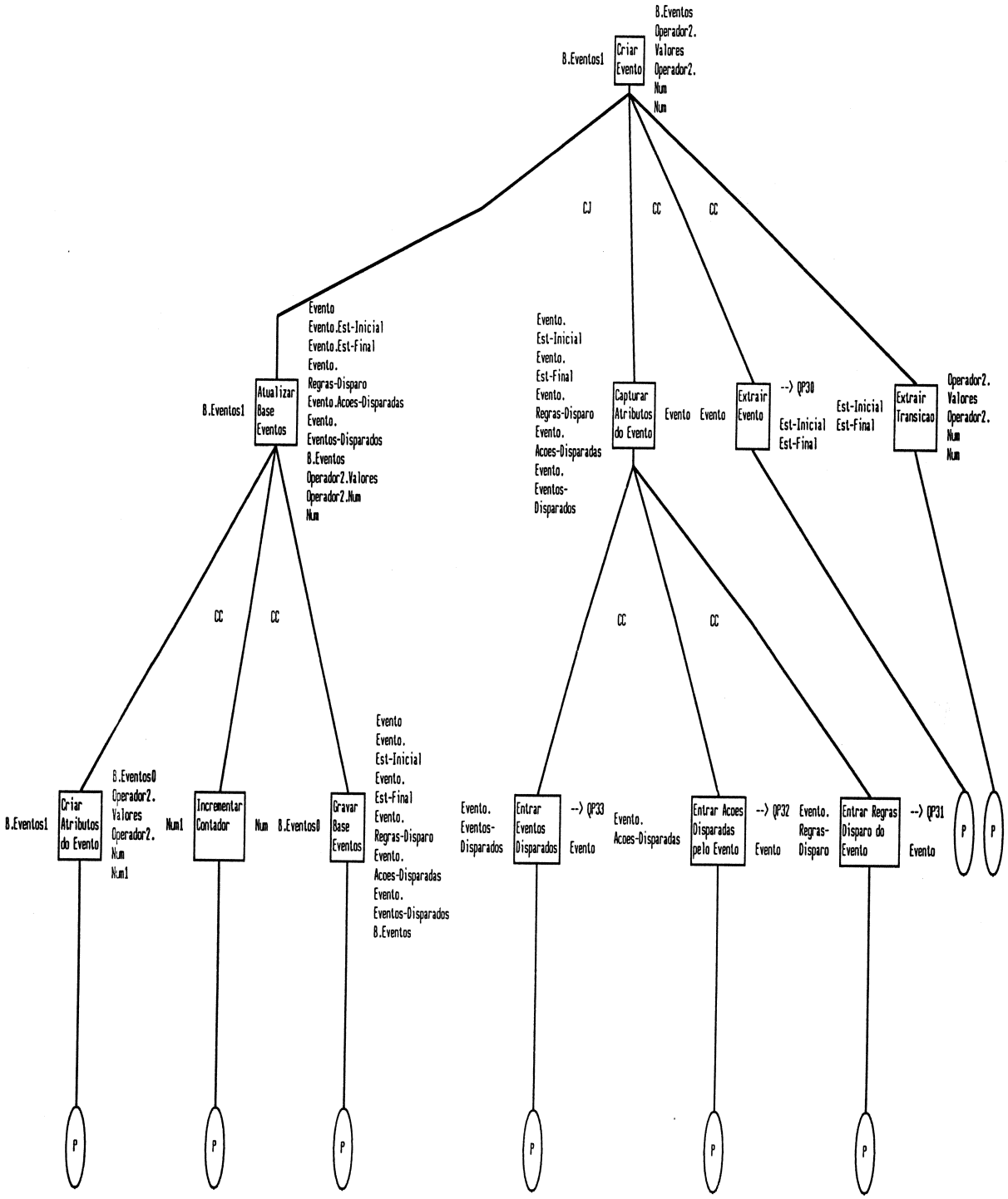


Criar Eventos

(continuação do CMO - Criar Nível de Representação)



Criar Evento
(continuação do CMO - Criar Nível de Representação)



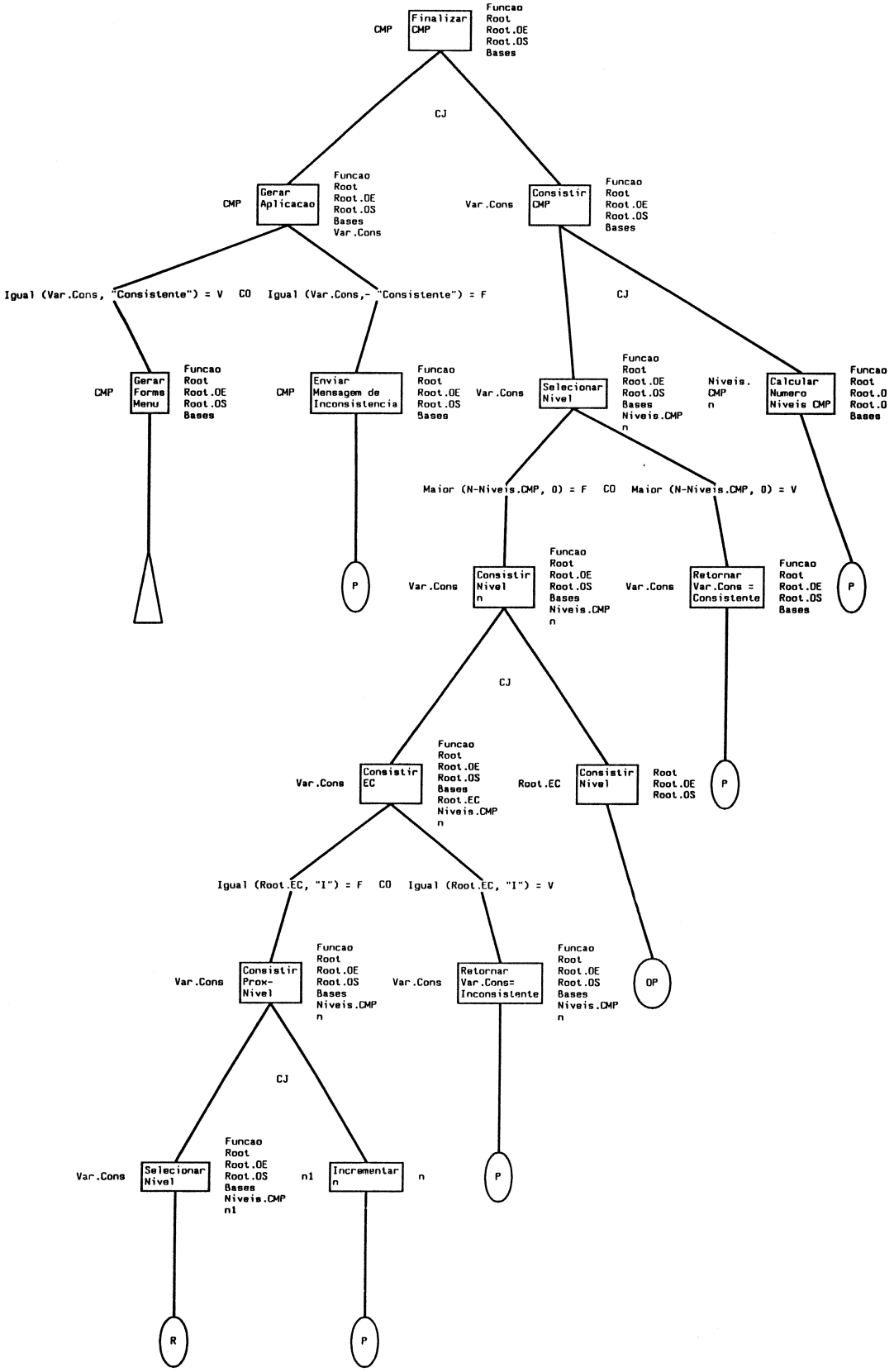
QP30) Quando ou como passar de <Est_Inicial> para <Est_Final>?
R. <Evento>

QP31) Quais as condições necessárias para que ocorra um(a) <Evento>?
R. <Evento.Regras_Disparo>

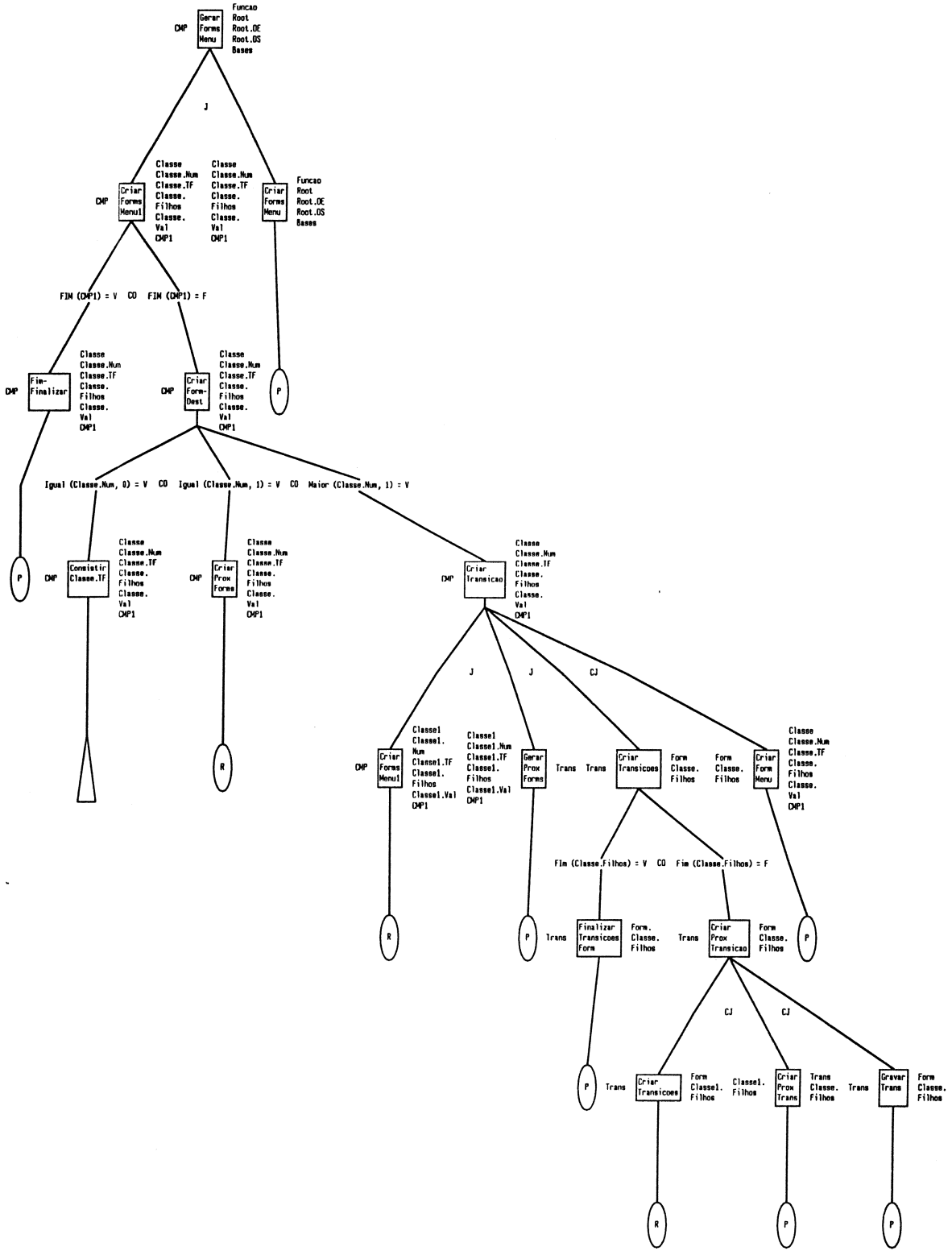
QP32) Quais as ações disparadas pelo(a) <Evento>?
R. <Evento.Acoes_Disparadas>

QP33) Descreva as próximas ocorrências quando um <Evento> é confirmado.
R. <Evento.Eventos_Disparados>

CMI - "Finalizar"



Gerar Forms Menu
(continuaçãodo CMI - Finalizar)



Consistir Classe.TF
(continuaçãodo CMI - Finalizar)