UFMG - ICEx
DEPARTAMENTO DE CIÊNCIA DA
C O M P U T A Ç Ã O

UFMG
UNIVERSIDADE FEDERAL DE MINAS GERAIS

Manual de Sistema para Índices e Projeções Baseadas em Índices para o HyperPro Básico

RT DCC 012/99

José de Siqueira
Fabrício Maia Schmidt

Índice Analítico

1. Introdução

O HyperPro Básico é uma ferramenta baseada no editor estruturado Thot para **hiper pro**gramação em lógica com restrições ([5],[6]). Esta ferramenta, desenvolvida para a linguagem de programação Prolog, permite que, em um ambiente homogêneo e integrado, seja feito o desenvolvimento e a documentação de diferentes programas e versões de programas, assim como seja feita a depuração e testes destes programas e versões neste ambiente.

Os documentos criados dentro do sistema Basic HyperPro podem conter diversos programas e suas versões. Estes programas e versões, uma vez que são escritos em CLP, são constituídos por relações.

Todas as relações tem pelo menos uma definição de predicado corrente *(c.p.d),* que é apontada por uma referência hypertexto do Thot, chamada referência de predicado corrente *(c.p.r).* Uma referência de predicado corrente aponta para alguma definição de predicado disponível e pode ser modificada para apontar para qualquer outra definição de predicado que o usuário deseje. Toda relação também possui uma barra de versões que é preenchida com os nomes das versões das quais ela participa.

O usuário define c.p.r's da maneira que lhe aprouver e, em seguida, nomeia a versão que será constituída pelas relações cujos c.p.d's serão aqueles apontados pelos c.p.r's previamente definidos pelo usuário. Assim, toda vez que o usuário modifica o c.p.r de uma relação para apontar para outra definição de predicado, uma nova versão deverá ser definida.

Uma nova versão é definida selecionando-se a primeira relação que compõe a versão e, em seguida, seleciona-se o comando para nomear a versão e digita-se o nome da mesma. Automaticamente, todas as relações participantes da versão terão a sua barra de versões acrescidas do nome da nova versão. Este nome é uma referência hypertexo para o c.p.d da relação, apontado por seu c.p.r no momento em que o comando para se nomear a versão foi disparado.

A figura 1 mostra um conjunto de relações com os seus c.p.r's apontando para as suas respectivas c.p.d's. Esta tarefa é feita pelo usuário do sistema, como dito anteriormente. Assim, é possível ver que na primeira relação optou-se para que o c.p.r apontasse para a primeira definição de predicado, ao invés da segunda. A figura 2 mostra a modificação obtida na figura 1 após o usuário ter selecionado a primeira relação *a/1* e nomeado a versão *V1*. Pode se ver que a barra de versões de todas as relações que participam da cadeia de chamadas iniciada em *a/1* recebem o nome da versão, *V1*, que é uma referência hypertexto para a definição de predicado correntemente apontada pelo c.p.r. da relação.

É importante dizer que, quando uma relação é selecionada para nomear uma versão, a sua definição de predicado naturalmente irá conter predicaçôes que fazem referências a outras relações presentes no documento, cujas definições de predicado também irão conter predicações que fazem referências a outras relações no documento e assim recursivamente. Esta cadeia de referências iniciada com a relação escolhida para nomear a versão é que irá definir quais relações irão fazer parte da versão, e que,

consequentemente, terão a sua barra de versões acrescidas do nome da nova versão nomeada. A maneira como a cadeia anterior pode ser definida e assim percorrida também é tarefa do usuário. Este deve colocar referências hypertextos em todas as predicações no corpo de cada definição de predicado que apontem para a relação que as define. Estas referências são chamadas referências de programas: p.r. A definição das p.r's é de grande importância, pois esta define a relação que define a predicação sem que haja dualidade. De fato, podem haver duas relações com o mesmo nome no documento, como é o caso da relação *a/1* na figura 1. Note-se que predicações dentro do c.p.d. da relação que as define, como no caso de chamadas recursivas ao predicado, não necessitam de p.r's. Na figura 2, as p.r's não são apresentadas para não sobrecarregar demais a figura.

A figura 3 abaixo mostra a situação da figura 2 depois que a relação *a/1* na seção 1.2 foi selecionada e a versão foi nomeada como *V2*. Depois disto, a relação *c/i* terá sua barra de versões acrescida de *V2*.

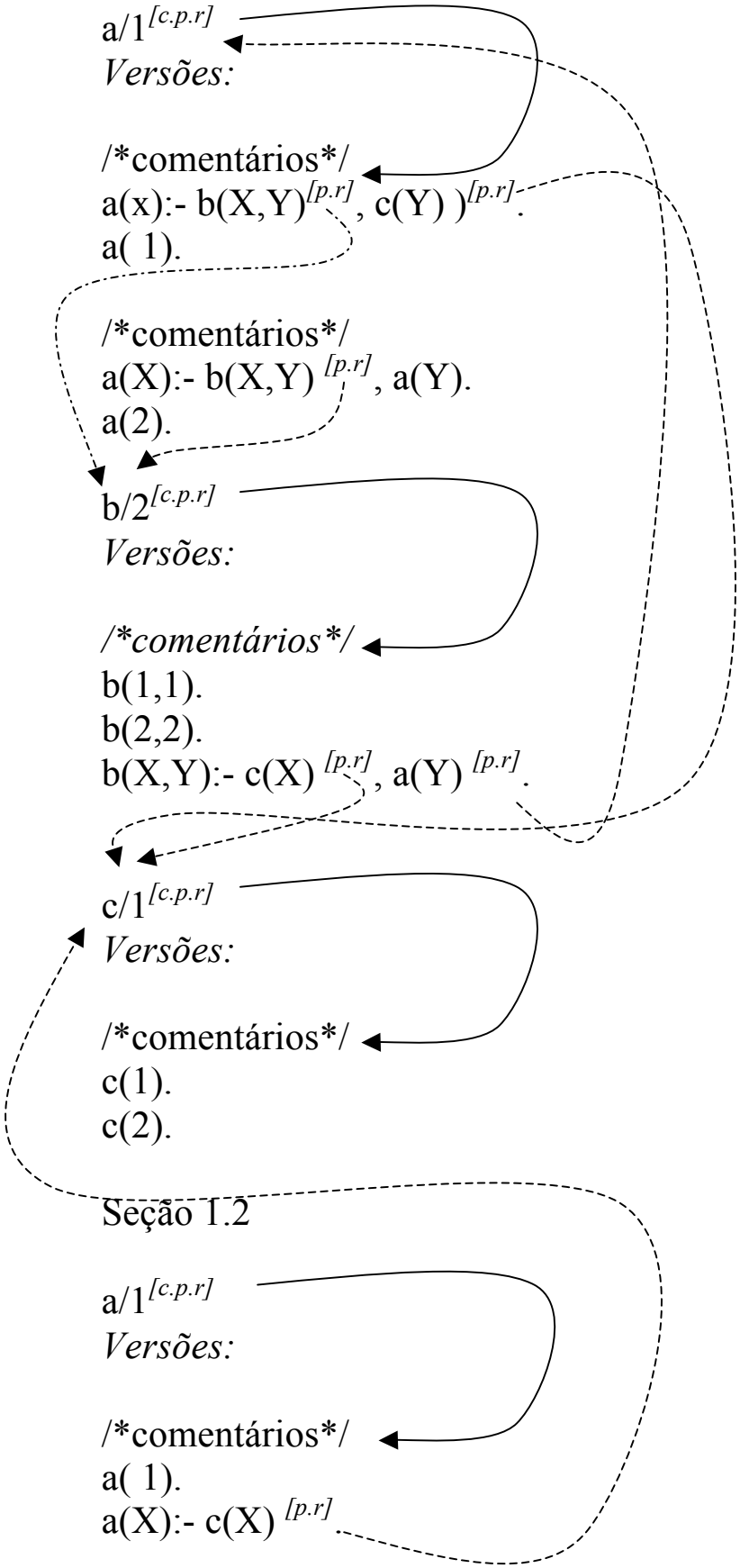

Figura 1: Algumas relações e suas referências de predicado corrente.

Seção 1.1

a/1[c.p.r]
Versões: V1.

/*comentários*/
a(x):- b(X,Y), c(Y).
a(1).

/*comentários*/
a(X):- b(X,Y), a(Y).
a(2).

b/2[c.p.r]
Versões: V1.

*/*comentários*/*
b(1,1).
b(2,2).
b(X,Y):- c(X), a(Y).

c/1[c.p.r]
Versões: V1.

/*comentários*/
c(1).
c(2).

Seção 1.2

a/1[c.p.r]
Versões:

/*comentários*/
a(1).
a(X):- c(X).

Figura 2: Depois de nomear a versão *V1*.

Seção 1.1

a/1[c.p.r]
Versões: V1.

/*comentários*/
a(x):- b(X,Y), c(Y).
a(1).

/*comentários*/
a(X):- b(X,Y), a(Y).
a(2).

b/2[c.p.r]
Versões: V1.

*/*comentários*/*
b(1,1).
b(2,2).
b(X,Y):- c(X), a(Y).

c/1[c.p.r]
Versões: V1, V2.

/*comentários*/
c(1).
c(2).

Seção 1.2

a/1
Versões: V2

/*comentários*/
a(1).
a(X):- c(X).

Figura 3: Depois de nomear a versão *V2*

2. Os Índices

O Thot adota um modelo em que os documentos são definidos por um esquema de estrutura, definido na linguagem S [1], e um esquema de apresentação, definido na linguagem P [1]. Um esquema de estrutura define como um documento é organizado no que diz respeito aos elementos que o compõem, tais como parágrafos, títulos, seções, etc. Todos estes elementos são organizados em uma estrutura de árvore, onde cada um tem uma posição e papel definido dentro do documento. O anexo A mostra a árvore de estrutura do documento HyperPro.

O esquema de apresentação, por sua vez, define, através da liguagem P, como estes elementos serão apresentados, isto é, define a fonte a ser usada em cada parte, seu tamanho, sua posição no texto, sua cor, etc. Uma característica do esquema de apresentação é que este pode definir diferentes vistas para um mesmo documento. Detsa forma, podemos apresentar, em uma vista, apenas os comentários, fórmulas, ou programas presentes no documento, ao invés de apresentarmos este como um todo. As vistas definidas no esquema de apresentação funcionam como um filtro que nos apresenta apenas a parte da estrutura do documento que queremos visualizar em dado momento.

Para construir automaticamente os índices, optou-se por definir novos elementos correspondentes aos índices, ou seja, foram implementados esquemas de estrutura e de apresentação para cada elemento índice (veja o Anexo B). Estes elementos foram adicionados ao esquema de estrutura e apresentação dos documentos HyperPro Básico. Esta forma foi escolhida por que a estrutura dos índices não é condizente com a estrutura do documento para o qual ele será construído. Se as estruturas fossem condizentes, os índices poderiam ser criados como vistas do documento, através do seu esquema de apresentação.

Os índices foram construídos através de aplicações que manipulam os elementos definidos para os índices e o documento para o qual serão construídos os índices.

Para implementar as aplicações que irão criar os índices automaticamente foi utilizada a Linguagem Geradora de Aplicação do Thot (Thot Application Generation Language [2]) e o Kit de API do Thot (Thot Tool Kit API [3]).

A Linguagem Geradora de Aplicação do Thot permite aplicações baseadas no conceito de documento ativo. Um documento ativo é um documento eletrônico que se transforma ou atua no seu próprio ambiente computacional, quando certos comandos de edição são requisitados pelo usuário. Este conceito é usado para construir diferentes tipos de aplicações assim como os índices deste projeto.

A Linguagem Geradora de Aplicação do Thot é usada para definir a interface gráfica (conjunto de menus) para o qual o projetista da aplicação pode associar um conjunto de funções. Estas funções podem ser as funções "standard" do Thot ou novas funções específicas.

Para a construção dos índices, a Linguagem Geradora de Aplicação do Thot foi usada par definir um menu que permite a escolha do índice a ser criado, Índice de Versões ou Índice de Referência Cruzada [4]. Neste menu, os botões referentes a cada índice são associados às funções que criam automaticamente os índices. Estas funções são HP_Menu_VersionIndex e HP_MenuPredCrossRefIndex e foram escritas em C, utilizand várias funções do Kit de API do Thot, como descrito nas seções seguintes.

3. Índice de Versões

Associado ao item de menu relativo à criação do Índice de Versões (IV), foi implementada em C a função HP_Menu_VersionIndex. Esta função usa várias funções do Kit de API do Thot, e constrói o IV quando o usuário seleciona o item de menu relativo à criação do Índice de Versões [4].

Esta função, no primeiro passo, percorre a estrutura do documento HyperPro, para o qual será construído o IV, acessando e armazenando os elementos necessários para a construção deste.

Para armazenar e recuperar estes elementos, foi criado um arquivo de inclusão em C. Neste arquivo se encontra a estrutura de dados que armazena estes elementos, as funções de inserção que posiciona estes elementos na estrutura e as funções de recuperação que obterão os elementos no passo seguinte. Para maiores informações, o texto deste arquivo se encontra no Anexo C.

No primeiro passo, serão acessadas cada relação que participe do documento e, para cada uma dessas relações, são acessadas as versões das quais elas participam e o c.p.d que a relação assume em cada uma dessas versões. As versões são então organizados em uma lista e a relação (nome/aridade e c.p.d) é armazenadada na listas correspondentes às versões das quais ela participa. Esta estrutura é uma lista encadeada de listas encadeadas e tal estrutura é montada para o documento da figura 3, como mostra a figura 4.

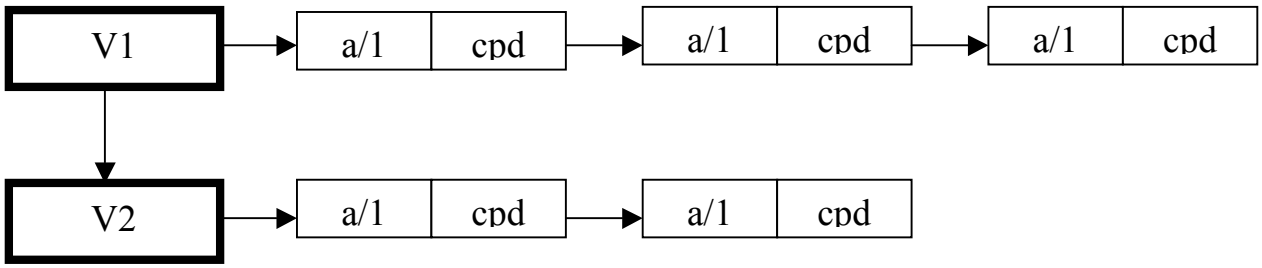

Figura 4. Lista de versões

É importante mencionar que podem existir duas relações com mesmo nome e aridade mas que participem de versões diferente e por isso possuem c.p.d's diferentes. O fato de haver duas relações a/1 não afeta o índice: a/1 participará da entrada referente a V1 e a V2, mas, quando alguma relação a/1 for selecionada no índice, o c.p.d referente àà sua versão é o exibido no documento.

Para a exibição destas informações no IV, foi criado um elemento cujo esquema de estrutura e de apresentação atende às necessidades deste índice. Para maiores informações, o esquema de estrutura e o esquema de apresentação do elemento IV se encontra no Anexo B. O esquema em árvore deste elemento pode ser vista na figura 5 abaixo (a letra L dentro de um círculo, nas figuras 5 e 7, significa que os elementos abaixo do círculo estão organizados em lista).

No segundo passo da nossa função, as informações coletadas no primeiro passo são então utilizadas para a construção do elemento IV. Um elemento IV é criado, e automaticamente são criados os elementos correspondentes a cada uma das versões (entradas do índice). Para cada versão (entrada do índice), os elementos correspondentes as relações participantes da versão são criados e fixados com os nomes das relações, assim como as referências aos c.p.d's são criadas e fixadas para os c.p.d's correspondentes. Estas referências aparecem como o número da página em que aparece os c.p.d referenciado. Clicando-se neste número o c.p.d referenciado é então exibido no documento HyperPro. Para maiores informações sobre como foi implementada esta função o seu código se encontra no Anexo D.

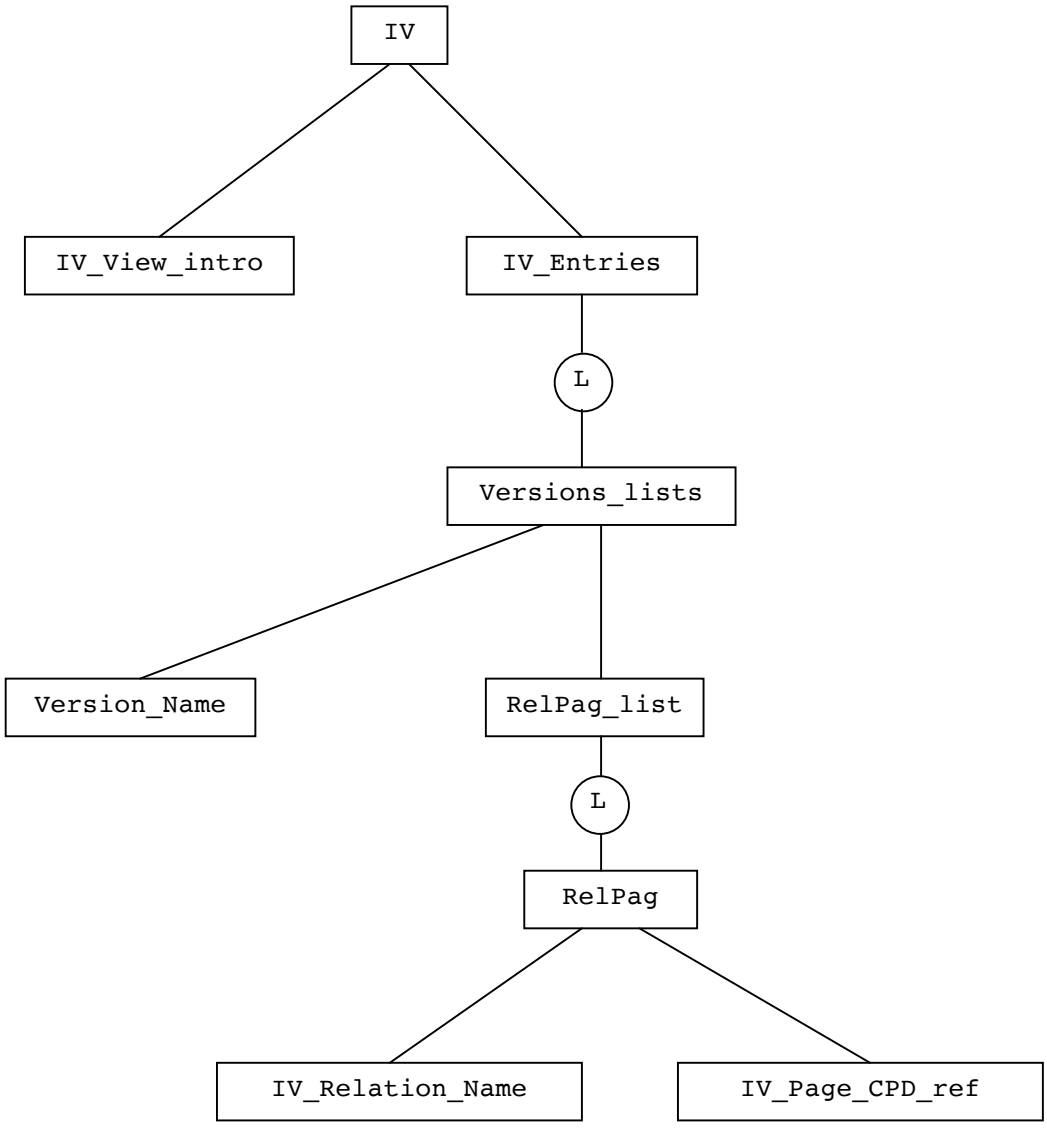

Figura 5: Esquema em árvore da estrutura do Índice de Versões

Em um nível mais detalhado, o que ocorre em cada passo da função para construção do IV pode ser sintetizado nas seguintes operações:

No primeiro passo:

1. Inicialização da estrutura que armazenará os elementos que figurarão no índice, acesso ao elemento raiz do documento Hyperpro Básico e acesso ao primeiro elemento do tipo Relation_title do documento encontrado depois da raiz. O elemento Relation_title é a raiz da árvore que será percorrida para coleta das informações relativas à relação em questão.

   CRInitIndex ⇒ Inicializa a estrutura que armazenará os elementos que participarão do índice.

TtaGetMainRoot ⇒ Acessa a raiz do documento
TtaGetSchema ⇒ Acessa o esquema de estrutura de um elemento.
TtaGiveTypeFromName ⇒ Retorna o tipo de um elemento dado o seu nome.
TtaSearchTypedElement ⇒ Retorna um elemento do documento do mesmo tipo que o fornecido como argumento.

2. Acessa a referência de predicado corrente do elemento Relation_title.

TtaGetFirstChild ⇒ Acessa o primeiro filho de um elemento.
TtaNextSibling ⇒ Acessa o primeiro irmão posterior de um elemento.

3. Acesso ao nome da relação e sua aridade, concatenando-os.

TtaGetTextLength ⇒ Retorna o número de caracteres de um string.
TtaGetMemory ⇒ Aloca um buffer do tamanho do parâmetro fornecido.
TtaGiveTextContent ⇒ Retorna o texto contido em um determinado elemento.

4. Acessa o elemento Versions_bar da relação, este elemento é uma lista das versões das quais a relação participa.

5. Acessa a primeira versão da barra de versões, elemento Version.

6. Acessa o nome da versão.

7. Acessa o atributo Version_ref de Version, este atributo é uma referência para a definição de predicado corrente da relação na versão em questão.

TtaGiveAttributeTypefromName ⇒ Retorna as informações de tipo de um atributo dado o seu nome.
TtaGetAttribute ⇒ Retorna o atributo de um elemento dada as informações de tipo deste atributo.

8. Acessa a definição de predicado corrente referenciada por Version_ref.

TtaGiveReferenceAttributevalue ⇒ Retorna o elemento referenciado por um atributo do tipo referência.

9. Procura a entrada na estrutura do índice correspondente ao nome da versão e se ela não existir a cria.

SearchEntry ⇒ Retorna a entrada do índice correspondente ao nome de versão fornecido, se elea não existir a cria.

10. Insere as informações obtidas nos passos 2 e 7 (nome da relação e definição de predicado corrente na versão em questão) na entrada do índice.

CRIInsertRelCPD ⇒ Insere no índice, na entrada fornecida como parâmetro, o nome de uma relação e seu cpd.

11. Se houver outro elemento Version o acessa e volta ao passo 5.

12. Procura o próximo elemento Relation_title do documento após o anterior, se encontrar volta ao passo 2.

Neste segundo passo são criados os elementos do documento constituintes do índice e as informações coletadas no primeiro passo são usadas para definir o valor destes elementos:

1. Primeiramente, se houver uma vista do IV aberta, esta é fechada.

   TtaGetViewFromName ⇒ Retorna a vista correspondente ao nome fornecido.
   TtaCloseView ⇒ Fecha a vista.

2. A raiz do documento é acessada, e, a partir dela, é procurado o elemento do tipo IVIndex.

3. O antigo elemento IVIndex é apagado e um novo é criado.

   TtaDeleteTree ⇒ Apaga um elemento e todos os seus filhos.
   TtaNewTree ⇒ Cria um novo elemento e todos os seus filhos.

4. O elemento CRIndex é acessado.

5. O elemento IVIndex é posicionado na estrutura do documento à direita de CRIndex.

   TtaInserSibling ⇒ Insere um elemento como irmão de outro elemento.

6. O primeiro elemento Versions_lists do CRIndex é acessado. A primeira entrada da estrutura de índice criada no passo 1 também é acessada.

7. A variável que representa o nome da versão na estrutura correspondente à entrada do índice, é copiada para o elemento Version_Name de Relations_lists.
8. O primeiro elemento RelPag da lista de relações e cpd's de RelPag_list é acessado assim como o primeiro da estrutura correspondente à entrada do índice.

9. O elemento Relation_Name de RelPag é acessado e seu valor é copiado da variável RelationName da lista de relações e cpd's da entrada do índice.

10. O elemento IV_Page_CPD_ref de RelPag é ajustado para referenciar o elemento referenciado pela variável que representa a definição de predicado corrente da relação na lista de chamadas da estrutura correspondente à entrada do índice (CPD).

    TtaSetReference ⇒ Ajusta uma referência para apontar para o mesmo elemento referenciado por outra.

11. Se existir mais chamadas na lista de chamadas da estrutura que representa uma entrada do índice, acessa esta chamada, cria outro elemento Use para armazenar a referência, insere-o ao lado do use anterior e volta ao passo 11.

12. Se existir outra entrada na estrutura do índice, acessa esta estrutura, cria novo elemento Versions_lists, insere-o depois do antigo Versions_lists e volta ao passo 9.

13. Abre-se a vista do Índice de Versões e define-se o título da vista com "Versions index".

    TtaOpenView ⇒ Abre uma vista.
    TtaChangeViewTitle ⇒ Define o título de uma vista.

4. Índice de Referência Cruzada

Associado ao item de menu relativo a criação do Índice de Referência Cruzada (IRC), foi implementada em C a função HP_Menu_PredCrossRefIndex. Esta função usa várias funções do Kit de API do Thot, e constrói o IRC quando o usuário seleciona o item de menu relativo à criação do Índice de Referência cruzada. Esta função, no primeiro passo, percorre a estrutura do documento HyperPro, para o qual será construído o IRC, acessando e armazenando os elementos necessários para a construção do IRC.

Para armazenar e recuperar estes elementos foi criado um arquivo de inclusão em C. Neste arquivo se encontra a estrutura de dados que armazena estes elementos, as funções de inserção que posiciona estes elementos na estrutura e as funções de recuperação que obterão os elementos no passo seguinte. Para maiores informações, este arquivo se encontra no Anexo D.

No primeiro passo, são acessadas cada relação que participe do documento e, para cada uma dessas relações, são acessados os elementos correspondentes ao seu título ao seu c.p.d e às chamadas que a relação possa ter no documento. As relações são então

organizados em uma lista e o seu título c.p.d e chamadas são armazenados na sublista da relação correspondente. Esta estrutura é uma lista encadeada de listas encadeadas e tal estrutura é montada para o documento da figura 3, como mostrado na figura 6 abaixo.

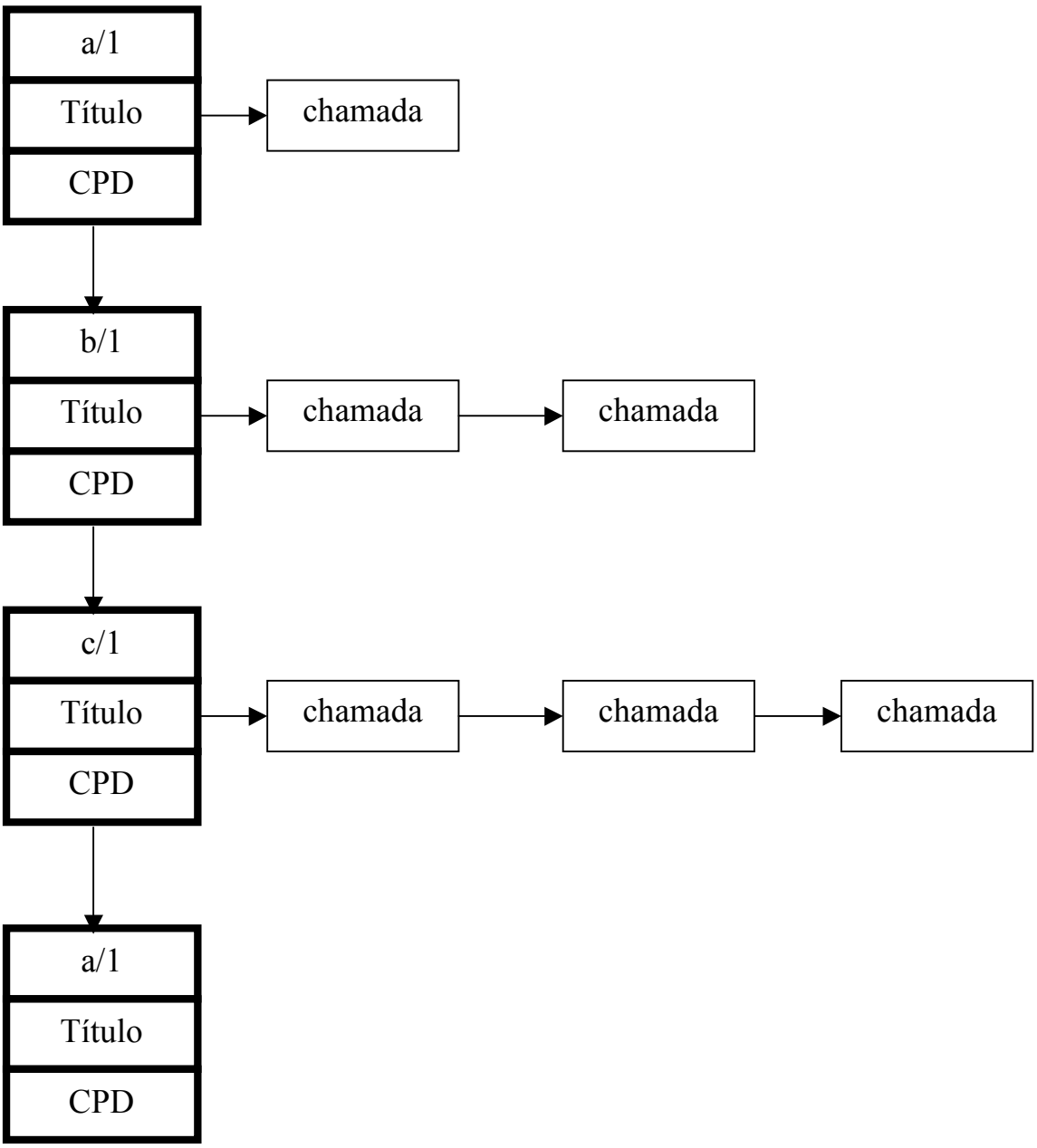

Figura 6 Estrutura para o índice de refrências cruzadas

É importante mencionar que podem existir duas relações com mesmo nome e aridade mas que participem de versões diferente e por isso possuem c.p.d's diferentes. O fato de haver duas relações a/1 não afeta o índice: a/1 terá duas entradas diferentes no índice, uma para cada versão. Isto é possível pois cada relação possui um identificador próprio para o elemento no documento HyperPro que a representa.

Para a exibição destas informações no IRC, foi criado um elemento cujo esquema de estrutura e de apresentação atende às necessidades deste índice. Para maiores informações, o esquema de estrutura e o esquema de apresentação do elemento IRC se encontram no Anexo B. O esquema em árvore da estrutura do elemento IRC é mostrada na figura 7 abaixo.

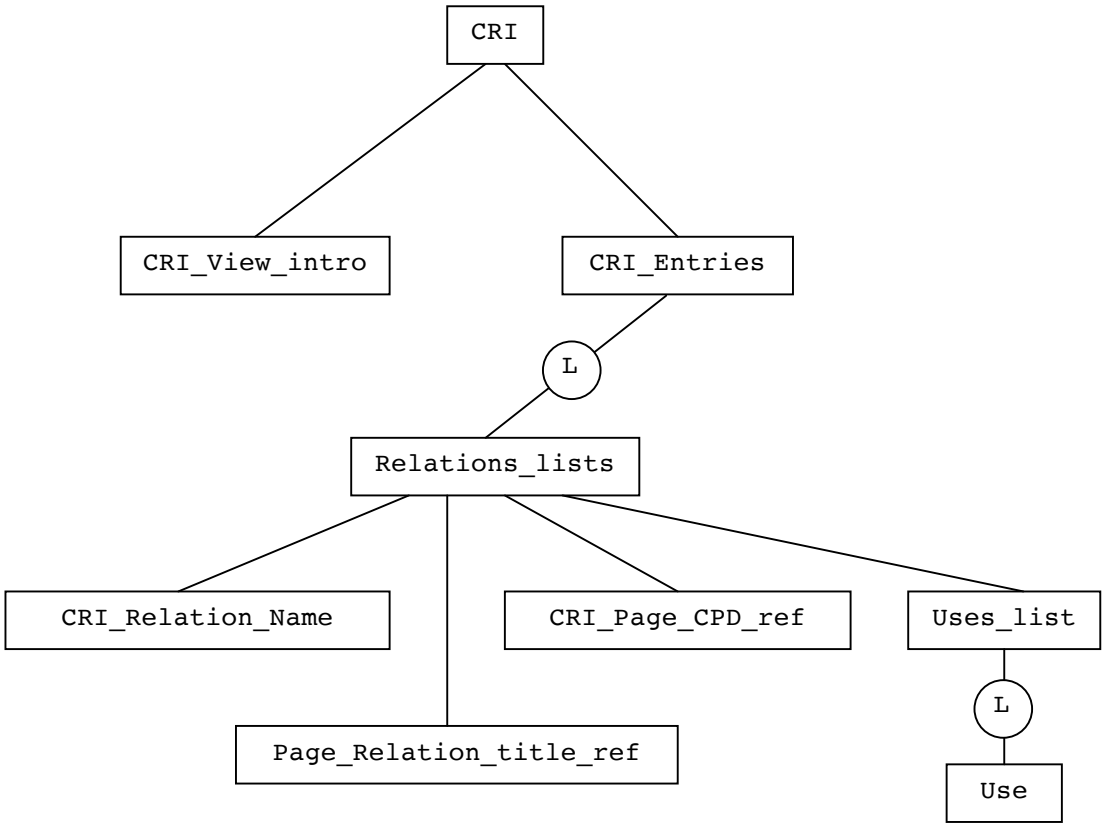

Figura 7: Esquema em árvore da estrutura do Índice de Referência Cruzada

No segundo passo da nossa função para a construção do IRC, as informações coletadas no primeiro passo são utilizadas para a construção do documento correspondente ao IRC, definido pelos esquemas citados anteriormente. Um elemento IRC é criado e automaticamente são criados os elementos correspondentes a cada uma das relações, que correspondem às entradas do índice. Para cada relação, as referências correspondentes ao título, c.p.d e chamadas à relação são criados e definidos para os elementos correspondentes no documento HyperPro, que foram coletados no passo anterior. Estas referências aparecem como o número da página em que aparece os elementos referenciados. Clicando-se neste número, o elemento referenciado é então exibido no documento HyperPro, conforme é mostrado na figura 5. Para maiores informações sobre como foi implementada esta função, o texto do seu código se encontra no Anexo F.

Em um nível mais detalhado, o que ocorre em cada passo da função para construção do IRC pode ser sintetizado nas seguintes operações:

No primeiro passo:

1. Inicialização da estrutura que armazenará os elementos que figurarão no índice, acesso ao elemento raiz do documento Hyperpro Básico e acesso ao primeiro elemento do tipo Relation_title do documento encontrado depois da raiz. O elemento Relation_title é a raiz da árvore que será percorrida para coleta das informações relativas à relação em questão.

   CRInitIndex ⇒ Inicializa a estrutura que armazenará os elementos que participarão no índice.
   TtaGetMainRoot ⇒ Acessa a raiz do documento
   TtaGetSchema ⇒ Acessa o esquema de estrutura de um elemento.
   TtaGiveTypeFromName ⇒ Retorna o tipo de um elemento, dado o seu nome.
   TtaSearchTypedElement ⇒ Retorna um elemento do documento do mesmo tipo que o fornecido como argumento.

2. Acesso à referência de predicado corrente da relação que está sendo processada e, à partir dela, ao elemento que está sendo referenciado.

   TtaGetFirstChild ⇒ Acessa o primeiro filho de um elemento.
   TtaNextSibling ⇒ Acessa o primeiro irmão a direita de um elemento.
   TtaGiveReferredElement ⇒ Retorna o elemento referenciado por um elemento do tipo referência.

3. Acesso ao nome da relação e sua aridade concatenando-os.

   TtaGetTextLength ⇒ Retorna o número de caracteres de um string.
   TtaGetMemory ⇒ Aloca um buffer do tamanho do parâmetro fornecido.
   TtaGiveTextContent ⇒ Retorna o texto contido em um determinado elemento.

4. As informações obtidas nos passos 2 e 3 são armazenadas em uma estrutura correspondente à uma entrada do índice e depois é inserida no mesmo.
   CRIInsertEntry ⇒ Insere no índice uma entrada.

5. Procura o próximo elemento do tipo Relation_title do documento, à partir do anterior. Se for encontrado, volta ao passo2.

6. Novamente o documento será percorrido desta vez para coletar as chamadas às relações e armazená-las nas entradas correspondentes da estrutura do índice. Mais

uma vez a raiz do documento é acessada e, a partir dela, o primeiro elemento do tipo Relation_def_ref é acessado. Este elemento é uma referência ao título da relação que está sendo chamada pelo seu elemento ancestral Other_const.

7. O título da relação apontado pelo elemento Relation_def_ref é acessado assim como o elemento Other_const. É preciso usar duas vezes a função TtaGetParent sobre Relation_def_ref para que este último seja acessado. Ele será o elemento visualizado pelo índice quando a chamada correspondente for selecionada no mesmo.

   TtaGetParent ⇒ Acessa o ancestral de um elemento do tipo fornecido como parâmetro.

8. A entrada correspondente ao título da relação acessada no passo anterior é procurada na estrutura do índice e então Other_const é inserido na lista de chamadas da entrada.

   CRISearchEntry ⇒ Retorna a posição do índice correspondente ao parâmetro fornecido.
   CRIInsertCall ⇒ Insere na lista de chamadas da entrada fornecida como parâmetro uma nova chamada.

9. Procura o próximo elemento do tipo Relation_def_ref do documento à partir do anterior, se for encontardo algum volta ao passo 7.

Neste segundo passo são criados os elementos do documento costituintes do índice e as informações coletadas no primeiro passo são usadas para definir o valor destes elementos:

1. Primeiramente se houver uma vista do IRC aberta esta é fechada.

   TtaGetViewFromName ⇒ Retorna a vista correspondente ao nome fornecido.
   TtaCloseView ⇒ Fecha a vista.

2. A raiz do documento é acessada e, a partir dela, é procurado o elemento do tipo CRIndex.

3. O antigo elemento CRIndex é apagado e um novo é criado.

   TtaDeleteTree ⇒ Apaga um elemento e todos os seus filhos.
   TtaNewTree ⇒ Cria um novo elemento e todos os seus filhos.

4. O elemento Document_title é acessado.

5. O elemento CRIndex é posicionado na estrutura do documento à direita de Document_title.

   TtaInsertSibling ⇒ Insere um elemento como irmão de outro elemento.

6. Primeiro elemento Relations_lists do CRIndex é acessado. A primeira entrada da estrutura de índice criada no passo 1 também é acessada.

7. A variável que representa o nome da relação na estrutura correspondente à entrada do índice é copiada para o elemento CRI_Relation_name de Relations_lists.

8. Elemento Page_relation_title_ref de Relations_lists é ajustado para referenciar o elemento do documento referenciado pela variável que representa o título da relação na estrutura correspondente à entrada do índice.

   TtaSetReference ⇒ Ajusta uma referência para apontar para o mesmo elemento referenciado por outra.

9. Elemento CRI_Page_CPD_ref de Relations_lists é ajustado para referenciar o elemento do documento referenciado pela variável que representa a definição de predicado na estrutura correspondente à entrada do índice.

10. O primeiro elemento Use da lista de chamadas Uses_list é acessado, assim como o primeiro da estrutura correspondente à entrada do índice.

11. Elemento Use é ajustado para referenciar o elemento referenciado pela variável que representa uma chamada da relação na lista de chamadas da estrutura correspondente à entrada do índice.

12. Se existir mais chamadas na lista de chamadas da estrutura, que representa uma entrada do índice, acessa esta chamada, cria outro elemento Use para armazenar a referência, insere-o ao lado do use anterior e volta ao passo 11.

13. Se existir outra entrada na estrutura do índice, acessa esta estrutura, cria novo elemento Relations_lists, insere-o depois do antigo relations_lists e volta ao passo 7.

14. Abre-se a vista do Índice de Referência Cruzada e define-se o título da vista com “Predicate cross-reference index”.

    TtaOpenView ⇒ Abre uma vista.
TtaChangeViewTitle ⇒ Determina o título de uma vista.

5. Projeção Baseada em Índices

Uma projeção mostra partes selecionadas de um documento em uma vista separada. A Projeção Baseada em Índices permite ao usuário selecionar qualquer entrada do índice e mostrar em uma vista separada todas as partes do documento relecionadas com elementos pertencentes à entrada selecionada do índice.

Para criar a Projeção baseada em índices, tivemos que modificar o esquema de apresentação e de estrutura do documento Basic HyperPro. A visibilidade de um elemento qualquer no documento deve ser definida no esquema de apresentação. As regras de apresentação do elemento devem definir uma visibilidade maior que a sensibilidade definida para a vista em que será exibido o documento.

No esquema de estrutura do documento Hyperpro Básico foram incluídos dois atributos de visibilidade, um para os elementos a serem exibidos na projeção baseada no Índice de Versões e outro para os elementos a serem exibidos na projeção baseda no Índice de Referência Cruzada. Abaixo são mostradas as partes do código do esquema de estrutura onde foram acrescentadas as modificações:

```
ATTR
Index_IV_proj_visible   = Yes;
Index_CRI_proj_visible = Yes;
```

No esquema de apresentação do Basic HyperPro, na seção views, foram incluídas as declarações das vistas correspondentes ao Índice de Versões e ao Índice de Referência Cruzada. A visibilidade do HyperPro Básico foi definida como zero para estas vistas e, finalmente, a sensibilidade de cada vista foi definida para conter o valor oito. Para definir a sensibilidade de cada vista foi usada a função da API TtaAttachAttribute. Abaixo são mostradas as partes do código do esquema de apresentação onde forma acrescentadas as modificações:

```
VIEWS
...
        Index_IV_based_projection_view, Index_CRI_based_projection_view;
...
RULES
HyperPro:
BEGIN
...
                IN Index_IV_based_projection_view Visibility: 0;
                IN Index_CRI_based_projection_view Visibility: 0;
                END;
...
ATTRIBUTES
...
Index_IV_proj_visible:
IN Index_IV_based_projection_view Visibility: 8;
Index_CRI_proj_visible:
IN Index_CRI_based_projection_view Visibility: 8;
END
...
```

Depois de executadas estas modificações a função HP_Menu_IndexBasedProjection, que realiza as projeções baseadas no Índice de Versões e no Índice de Referência Cruzada, foi implementada (para ver o código comentado desta função, veja o Anexo G). Esta função, em alto nível, desempenha as seguintes operações:

1. Exibe um cursor para escolher o elemento para o qual será exibida a projeção e retorna o elemento escolhido.

   TtaClickElement ⇒ Retorna o elemento selecionado.

2. O ancestral do elemento escolhido é acessado e se ele for do tipo Versions_lists a projeção baseada no Índice de Versões é executada. Se ele for do tipo Relations_lists, a projeção baseada no Índice de Referência Cruzada é executada.

   TtaGetTypedAncestor ⇒ Retorna o elemento ancestral de um dado elemento que tenha o mesmo tipo que o fornecido como parâmetro.

A seguir, detalhamos as operações necessárias à criação das projeções baseadas no Índice de Versões e no Índice de Refências Cruzadas.

Projeção Baseada no Índice de Versões:

1. Primeiramente é removido do documento todos os atributos de visibilidade do tipo attr_Index_IV_projection_visible para que se crie uma nova projeção baseda no elemento escolhido.

HP_Util_RemoveAttributes ⇒ Remove todos os atributos de um determinado tipo do documento.

2. O elemento RelPagList de Versions_lists é acessado.

   TtaGetLastChild ⇒ Retorna o último filho de um elemento.

3. O primeiro elemento do tipo RelPag de RelPag_list é acessado.

   TtaSearchTypedElement ⇒ Retorna um elemento do documento do mesmo tipo que o fornecido como argumento.

4. A referência de predicado corrente de RelPag é acessada (IV_Page_CPD_ref).

5. O elemento referenciado por IV_Page_CPD_ref é acessado.

TtaGiveReferredElement ⇒ Retorna o elemento referenciado por um elemento do tipo referência.

6. O ancestral do tipo Relation_def do elemento referenciado por IV_Page_CPD_ref é acessado.

   TtaGetTypedAncestor ⇒ Retorna o elemento ancestral de um dado elemento que tenha o mesmo tipo que o fornecido como parâmetro.

7. Se Relation_def não possuir o atributo de visibilidade Index_IV_projection_visible, este último é criado e é associado à Relation_def.

   TtaGetAttribute ⇒ Retorna o atributo de um elemento dada as informações de tipo deste atributo.
   TtaNewAttribute ⇒ Cria novo atributo.
   TtaAttachAttribute ⇒ Associa um atributo a um elemento.

8. Caso haja um outro elemento RelPag em Versions_lists este é acessado e volta ao passo 4.

TtaNextSibling ⇒ Acessa o primeiro irmão posterior de um elemento.

9. A vista da Projeção Baseada no Índice de Versões é aberta.

   HP_Util_OpenProj ⇒ Abre a vista de uma projeção.

Projeção Baseada no Índice de Referência Cruzada:

1. Primeiramente é removido do documento todos os atributos de visibilidade do tipo attr_Index_CRI_projection_visible para que se crie uma nova projeção baseda no elemento escolhido.

HP_Util_RemoveAttributes ⇒ Remove todos os atributos de um determinado tipo do documento.

2. É acessada a referência ao cpd da relação correspondente a entrada selecionada do índice, e, então obtido o elemento Predicate_def. O elemento Relation_def ancestral de Predicate_def é acessado.

TtaGetLastChild ⇒ Retorna o último filho de um elemento.
TtaGetPreviousSibling ⇒ Retorna o elemento irmão anterior de um elemento.
TtaGiveReferredElement ⇒ Retorna o elemento referenciado por um elemento do tipo referência.
TtaGetTypedAncestor ⇒ Retorna o elemento ancestral de um dado elemento que tenha o mesmo tipo que o fornecido como parâmetro.

3. Se não houver o atributo de visibilidade da vista associado à Predicate_def então é feita a associação.

   TtaGetAttribute ⇒ Retorna o atributo de um elemento dada as informações de tipo deste atributo.
   TtaNewAttribute ⇒ Cria novo atributo.
   TtaAttachAttribute ⇒ Associa um atributo a um elemento.

4. É acessada a primeira chamada da lista de chamadas da entrada selecionada do índice, elemento Use.

   TtaGetFirstChild ⇒ Acessa o primeiro filho de um elemento.

5. É obtido o elemento Predicate_def referenciado por Use. O elemento Relation_def ancestral de Predicate_def é acessado.

TtaGiveReferredElement ⇒ Retorna o elemento referenciado por um elemento do tipo referência.

TtaGetTypedAncestor ⇒ Retorna o elemento ancestral de um dado elemento que tenha o mesmo tipo que o fornecido como parâmetro.

6. Se não houver o atributo de visibilidade da vista associado à Predicate_def então é feita a associação.

7. Caso haja outro elemento Use na lista de chamadas este é acessado e voltamos ao passo 4.

   TtaNextSibling ⇒ Acessa o primeiro irmão a direita de um elemento.

8. Abre a Projeção Baseada no Índice de referência cruzada.

   HP_Util_OpenProj ⇒ Abre a vista de uma projeção.

6. Referências

[1] V. Quint, The Languages of Thot, Abril de 1997, Relatório Interno, INRIA, França, maio de 1992.

[2] V. Quint and I. Vatton, The Thot Application Generation Language, INRIA, França, maio de 1997.

[3] V. Quint and I. Vatton, The Thot Tool Kit API, INRIA, França, julho de 1997.

[4] J. de Siqueira e F. M. Schmidt, Manual do Usuário: Índices e Projeções Baseadas em Índices para o HyperPro Básico, Relatótio Técnico do DCC 013/99, Departamento de Ciência da Computação, UFMG, outubro de 1999.

[5] P. Deransart, R. Bigonha, A. Ed-Dbali, J. de Siqueira, M.A.S. Bigonha, Basic HyperPro Functionalities and Utilities, Relatório Técnico DCC 023/97, Departamento de Ciência da Computação, UFMG, 1997.

[6] P. Deransart, R. Bigonha, P. Parot, M.A.S. Bigonha, J. de Siqueira, A Hypertext Based Environment to Write Literate Logic Programming, Anais do JICSLP'96, Bonn Germany, setembro de 1996, pp. 247-252.

7. Anexos

7.1 Anexo A

Na página seguinte, mostramos o diagrama do esquema de estrutura do HyperPro.

7.2 Anexo B

Este anexo mostra o esquema de estrutura para o documento HyperPro e o esquema de apresentação para os índices.

```
{------------------------------------------------------------------------ }
{ Document model for Logic programming environment HyperPro                 }
{ Pierre Deransart / Ali Ed-Dbali / Khalid El Qorchi/Flavia Peligrinelli }
{ Fabricio Schmidt                                                        }
{ May, 04th, 1999.                                                  }
{------------------------------------------------------------------------ }

STRUCTURE HyperPro;

DEFPRES HyperProP;

ATTR
Manual_proj_visible    = Yes;
Recursive_proj_visible         = Yes;
Reg_expression_proj_visible    = Yes;
Index_IV_proj_visible  = Yes;
Index_CRI_proj_visible = Yes;
Version_proj_visible         = Yes;

STRUCT

HyperPro (ATTR First_page_number = Integer;
First_section_number = Integer) =
BEGIN
Document_title  = Lines;
CRIndex         = CRI;   {*CRI*}
IVIndex         = IV;    {*IV*}
? Document_date   = Lines;
? Authors         = LIST OF (Author);
? Affiliations    = LIST OF (Affiliation = Lines);
? Key_words    = Lines;
Sections_seq;
? Bibliography   = LIST OF (Citation_biblio = RefBib);
? Annexes         = LIST OF (Annexe);
END;

Author (ATTR Author_type = Principal_author,Secondary_author) =
Content + (Affiliation_ref)
- (Rel_def_ref, Current_pred, Figure_ref, Formula_ref,
Section_ref, Annexe_ref, Titled_group_ref);

Affiliation_ref = REFERENCE (Affiliation);

Annexe =
BEGIN
Annexe_title = Lines;
Sections_seq;
END;

Sections_seq = LIST OF (Section);

Section =
BEGIN
Section_title = TEXT;
Section_body =
LIST OF ( Paragraphs_or_Relations =
CASE OF
Paragraphs_seq;
Relation_def;
END);
? Sections_seq;
END;

Paragraphs_seq = LIST [0..*] OF (Paragr);

Relation_def =
BEGIN
Relation_title =
```

```
BEGIN
Relation_name = TEXT;
Relation_arity = TEXT;
Current_predicate_ref = REFERENCE (Predicate_def);
END;
Versions_bar = LIST OF (Version);
Definitions = LIST OF (Predicate_def);
END;

Version (ATTR Version_ref = REFERENCE (Predicate_def)) = TEXT;


{Version =
BEGIN
Version_name = TEXT;
Version_ref = REFERENCE (Predicate_def);
END;}

Predicate_def =
BEGIN
Informal_comments = LIST [1..*] OF (Paragr);
? Predicate_types = LIST OF (Predicate_type);
? Assertions = LIST OF (Assertion);
Packet = LIST OF (Clause);
END;

Paragr =
CASE OF
Paragraphe;
Titled_group =
BEGIN
Group_title = Lines;
Group = LIST OF (Paragr);
END;
Image = PICTURE;
Numbred_formula = Math;
Figure;
Other_nature = NATURE;
END;

Figure =
BEGIN
Illustration =
CASE OF
Picturial = PICTURE;
Other_figure = NATURE;
END;
Figure_Legend = Content;
END;

Content = LIST OF (Unity = UNIT);

Predicate_type = Lines;

Assertion = Lines;

Clause =
CASE OF
Comments;
Fact;
Rule;
Goal;
Other = Lines;
END;

Fact =
BEGIN
Predication_head;
END;

Rule =
BEGIN
Head;
Body;
END;

Head =
```

```
BEGIN
Predication_head;
END;

Predication_head = Lines;

Goal =
BEGIN
Body ;
END;

Body = LIST OF (BIP_construction);

BIP_construction =
CASE OF
Predication;
Disjunction;
Conjunction;
If_Then;
If_Then_Else;
Comments;
Other_const;
END;

Comments = Lines;

Predication = Lines + (Relation_def_ref);

Disjunction =
BEGIN
First_op_disjunction = BIP_construction;
Second_op_disjunction = BIP_construction;
END;

Conjunction =
BEGIN
First_op_conjunction = BIP_construction;
Second_op_conjunction = BIP_construction;
END;

If_Then =
BEGIN
Condition = BIP_construction;
Alternative = BIP_construction;
END;

If_Then_Else =
BEGIN
Condition ;
Alternative1 = BIP_construction;
Alternative2 = BIP_construction;
END;

Other_const = Lines + (Relation_def_ref);

Relation_def_ref = REFERENCE (Relation_title);

Lines = LIST OF (Text_line = TEXT);


{*Element CRI*}

CRI =
BEGIN
CRI_View_intro  = Lines;
CRI_Entries = LIST OF (Relations_lists);
END;

Relations_lists =
BEGIN
CRI_Relation_Name       = TEXT;
Page_Relation_title_ref = REFERENCE ( Relation_title );
CRI_Page_CPD_ref        = REFERENCE ( Predicate_def );
Uses_list               = LIST OF (Use);
END;

Use = REFERENCE ( ANY );
```

```
{*End of element CRI*}


{*Element IV*}

IV =
BEGIN
IV_View_intro = Lines;
IV_Entries    = LIST OF (Versions_lists);
END;

Versions_lists =
BEGIN
Version_Name = TEXT;
RelPag_list  = LIST OF (RelPag);
END;

RelPag =
BEGIN
IV_Relation_Name = TEXT;
IV_Page_CPD_ref  = REFERENCE ( Predicate_def );
END;

{*End of element IV*}


ASSOC
Note = Paragraphs_seq;

UNITS
Subscript = TEXT;
Superscript = TEXT;
Ref_rel_def = Relation_def_ref;
Current_pred_def = Current_predicate_ref;
Note_ref = REFERENCE (Note);
Biblio_ref = REFERENCE (Citation_biblio);
Figure_ref = REFERENCE (Figure);
Formula_ref = REFERENCE (Numbred_formula);
Section_ref = REFERENCE (Section);
Annexe_ref = REFERENCE (Annexe);
Titled_group_ref = REFERENCE (Titled_group);
Other_thing = NATURE;

EXCEPT

Authors:                        Hidden;
Affiliations:          Hidden;
Sections_seq:          Hidden;
Version_ref:                    ActiveRef;

END
```

Esquema de apresentação para o Índice de Versões

```
{*IV Rules*}

IVIndex:
BEGIN
VertPos : Top = Enclosing . Top + 60 pt;
Width : Enclosing . Width;
HorizPos: VMiddle = Enclosing . VMiddle;
Visibility: 0;
IN IV_view
Visibility: 8;
END;

IV:
BEGIN
Visibility: 0;
IN IV_view
Visibility: 8;
END;

IV_View_intro:
```

```
BEGIN
Size : 17 pt;
Style : italics;
Width : Enclosing . Width - 40 pt;
HorizPos : Left = Enclosing . Left + 20 pt;
Adjust : Left;
Line;
Visibility: 0;
IN IV_view
Visibility: 8;
END;

IV_Entries:
BEGIN
VertPos : Top = Previous . Bottom + 30 pt;
Size : 15 pt;
Visibility: 0;
IN IV_view
Visibility: 8;
END;


Versions_lists:
BEGIN
VertPos : Top = Previous . Bottom + 20 pt;
Line;
Visibility: 0;
IN IV_view
Visibility: 8;
END;

Version_Name:
BEGIN
Size : 20 pt;
Width : Enclosed . Width;
Style : Bold;
CreateAfter(Box_Colon);
Visibility: 0;
IN IV_view
Visibility: 8;
END;

RelPag_list:
BEGIN
width : Enclosed.width;
Visibility: 0;
IN IV_view
Visibility: 8;
END;

RelPag:
BEGIN
IF NOT ONE (CptRelPag)
CreateBefore (Index_Box_Comma);
width : Enclosed . width;
Visibility: 0;
IN IV_view
Visibility: 8;
END;

IV_Relation_Name:
BEGIN
Width : Enclosed . Width;
Visibility: 0;
IN IV_view
Visibility: 8;
END;

IV_Page_CPD_ref:
BEGIN
CreateBefore (Box_Space);
Style : Bold;
Copy (Box_Index_Page_number);
Visibility: 0;
In IV_view
Visibility: 8;
END;
```

```
{*End of IV Rules*}
```

Esquema de apresentação para o Índice de referência Cruzada.

```
{CRI Rules}

CRIndex:
BEGIN
VertPos : Top = Enclosing . Top + 60 pt;
Width : Enclosing . Width;
HorizPos: VMiddle = Enclosing . VMiddle;
Visibility: 0;
IN CRI_view
Visibility: 8;
END;

CRI:
BEGIN
Visibility: 0;
IN CRI_view
Visibility: 8;
END;


CRI_View_intro:
BEGIN
Size : 17 pt;
Style : italics;
Width : Enclosing . Width - 40 pt;
HorizPos : Left = Enclosing . Left + 20 pt;
Adjust : Left;
Line;
Visibility: 0;
IN CRI_view
Visibility: 8;
END;

CRI_Entries:
BEGIN
VertPos : Top = Previous . Bottom + 30 pt;
Size : 15 pt;
Visibility: 0;
IN CRI_view
Visibility: 8;
END;

Relations_lists:
BEGIN
VertPos : Top = Previous . Bottom + 20 pt;
Line;
Visibility: 0;
IN CRI_view
Visibility: 8;
END;

CRI_Relation_Name:
BEGIN
CreateAfter(Box_Colon);
Size : 20 pt;
Width : Enclosed . Width;
Style : Bold;
Visibility: 0;
IN CRI_view
Visibility: 8;
END;

Page_Relation_title_ref:
BEGIN
CreateAfter(Index_Box_Comma);
Style: Italics;
Copy (Box_Index_Page_number);
Visibility: 0;
IN CRI_view
```

```
Visibility: 8;
END;

CRI_Page_CPD_ref:
BEGIN
CreateAfter(Index_Box_Comma);
Style: Bold;
Copy (Box_Index_Page_number);
Visibility: 0;
IN CRI_view
Visibility: 8;
END;

Uses_list:
BEGIN
width : Enclosed.width;
Visibility: 0;
IN CRI_view
Visibility: 8;
END;

Use:
BEGIN
IF NOT ONE (CptUse)
CreateBefore (Index_Box_Comma);
Copy (Box_Index_Page_number);
Visibility: 0;
IN CRI_view
Visibility: 8;
END;

{End of CRI Rules}
```

7.3 Anexo C

Este anexo mostra o conteúdo do arquivo de inclusão da função de construção do Índice de Versões.

```
/* Arquivo include para o Índice de versões, contém a estrutura de dados e métodos para
armazenamento das informações que serão usadas na construção do Índice de Versões */

//Corresponde a uma célula da lista de relações e cpds de uma entrada
typedef struct _RelCPD{
char RelationName[40];
Element CPD;
struct _RelCPD *Next;
} SRelCPD;

//Corresponde a uma entrada do índice
typedef struct _Entry{
        char VersionName[40];
        SRelCPD *FirstRelCPD;
        SRelCPD *LastRelCPD;
        struct _Entry *Next;
}SEntry;

//Índice
typedef struct _Index{
        SEntry *FirstEntry;
        SEntry *LastEntry;
}SIndex;

//Inicializa a estrutura do índice
void InitIndex(SIndex *Index){
   if((Index->FirstEntry = (SEntry*) malloc(sizeof(SEntry))) == NULL){
      printf("not enough memory for InitIndex(). \n");
exit (1);
}
Index->LastEntry = Index->FirstEntry;
Index->FirstEntry->Next = NULL;
}
```

```
//Verifica se o índice está vazio
int EmptyIndex(SIndex Index)
if(Index.FirstEntry == Index.LastEntry) return(1);
return(0);
}

//Insere no índice uma nova entrada
void InsertEntry(char *VersionName, SIndex *Index){
 if((Index->LastEntry->Next    =    (SEntry*)    malloc(sizeof(SEntry)))    ==    NULL)    {
       printf("not enough memory for InsertEntry(1). \n");
exit (1);
}
Index->LastEntry = Index->LastEntry->Next;
strcpy(Index->LastEntry->VersionName, VersionName);
Index->LastEntry->Next = NULL;
if((Index->LastEntry->FirstRelCPD = (SRelCPD*) malloc(sizeof(SRelCPD))) == NULL){
printf("not enough memory for InserEntry(2).\n");
exit (1);
}
Index->LastEntry->LastRelCPD = Index->LastEntry->FirstRelCPD;
Index->LastEntry->FirstRelCPD->Next = NULL;
}

//Procura a entrada de acordo com o nome da versão, e, se ela não existir a cria
void SearchEntry(char *VersionName, SIndex *Index, SEntry **Entry){
SEntry *aux;

aux = Index->FirstEntry->Next;
while((aux != NULL) && (strcmp(aux->VersionName, VersionName))) aux = aux->Next;
if(aux == NULL){
InsertEntry(VersionName, Index);
*Entry = Index->LastEntry;
}else *Entry = aux;
}

//Insere na entrada do Índice o par nome relação e cpd
void   InsertRelCPD(SRelCPD   RelCPD,   SEntry   *Entry){ if((Entry->LastRelCPD->Next   =
(SRelCPD*) malloc(sizeof(SRelCPD))) == NULL){
if((Entry->LastRelCPD->Next = (SRelCPD*) malloc(sizeof(SRelCPD))) == NULL){
printf("not enough memory for InsertRelCPD(),\n");
exit (1);
}
Entry->LastRelCPD = Entry->LastRelCPD->Next;
strcpy(Entry->LastRelCPD->RelationName, RelCPD.RelationName);
Entry->LastRelCPD->CPD = RelCPD.CPD;
Entry->LastRelCPD->Next = NULL;
}
//Imprime o índice para efeitos de debug
void PrintIndex(SIndex Index){
SEntry *aux1;
SRelCPD *aux2;

aux1 = Index.FirstEntry->Next;
while(aux1 != NULL){
printf("%s : ", aux1->VersionName);
aux2 = aux1->FirstRelCPD->Next;
while(aux2 != NULL){
printf("%s %p", aux2->RelationName, aux2->CPD);
aux2 = aux2->Next;
}
printf("\n");
aux1 = aux1->Next;
}
}
```

7.4 Anexo D

Este anexo mostra o conteúdo do arquivo de inclusão da função de construção do Índice de Referência Cruzada.

```
/* Arquivo include para o Índice de Referência Cruzada, contém a estrutura de dados e
métodos para armazenamento das informações que serão usadas na construção do Índice de
Versões */

//Corresponde a uma célula da lista de chamadas de uma entrada do índice
typedef struct _Call{
Element Call;
struct _Call *Next;
} SCall;

/* Estrutura que armazena os ponteiros para o cpd e título da relação além do seu nome
*/
typedef struct _Relation{
Element Title;
char Name[50];
Element CPD;
}SRelation;

//Correponde a uma entrada do índice
typedef struct _CRIEntry{
SRelation Relation;
SCall *FirstCall;
SCall *LastCall;
struct _CRIEntry *Next;
}SCRIEntry;

//índice
typedef struct _CRIIndex{
SCRIEntry *FirstEntry;
SCRIEntry *LastEntry;
}SCRIIndex;

//Função que inicializa o índice
void CRIInitIndex(SCRIIndex *Index){
if((Index->FirstEntry = (SCRIEntry*) malloc(sizeof(SCRIEntry))) == NULL){
printf("not enough memory for InitIndex(). \n");
exit (1);
}
Index->LastEntry = Index->FirstEntry;
Index->FirstEntry->Next = NULL;
}

//Função para verificar se o índice está vazio
int CRIEmptyIndex(SCRIIndex Index){
if(Index.FirstEntry == Index.LastEntry) return(1);
return(0);
}

void CRIInsertEntry(SRelation Relation, SCRIIndex *Index){
if((Index->LastEntry->Next = (SCRIEntry*) malloc(sizeof(SCRIEntry))) == NULL) {
printf("not enough memory for CRIInsertEntry(1). \n");
exit (1);
}
Index->LastEntry = Index->LastEntry->Next;
Index->LastEntry->Relation = Relation;
Index->LastEntry->Next = NULL;
if((Index->LastEntry->FirstCall = (SCall*) malloc(sizeof(SCall))) == NULL){
printf("not enough memory for CRIInserEntry(2).\n");
exit (1);
}
Index->LastEntry->LastCall = Index->LastEntry->FirstCall;
Index->LastEntry->FirstCall->Next = NULL;
}

//Procura no índice uma entrada de acordo com Title
void CRISearchEntry(Element Title, SCRIIndex *Index, SCRIEntry **Entry){
SCRIEntry *aux;
```

```
aux = Index->FirstEntry->Next;
while((aux != NULL) && (aux->Relation.Title != Title)) aux = aux->Next;
if(aux == NULL) printf("Relation_def_ref points to hide Relation_title \n");
else *Entry = aux;
}

//Insere na entrada Entry uma nova chamada
void CRIInsertCall(Element Call, SCRIEntry *Entry){
if((Entry->LastCall->Next = (SCall*) malloc(sizeof(SCall))) == NULL){
printf("not enough memory for InsertCall(),\n");
exit (1);
}
Entry->LastCall = Entry->LastCall->Next;
Entry->LastCall->Call = Call;
Entry->LastCall->Next = NULL;
}

//Imprime o índice para debug
void CRIPrintIndex(SCRIIndex Index){
SCRIEntry *aux1;
SCall *aux2;

aux1 = Index.FirstEntry->Next;
while(aux1 != NULL){
printf("%p, %s, %p : ", aux1->Relation.Title, aux1->Relation.Name,
aux1->Relation.CPD);
aux2 = aux1->FirstCall->Next;
while(aux2 != NULL){
printf("%p ", aux2->Call);
aux2 = aux2->Next;
}
printf("\n");
aux1 = aux1->Next;
}
}
```

7.5 Anexo E

Este anexo mostra o código comentado da função que cria o índice de versões.

```
/*-------------------------------------------------------------------------
-------------------------------------------------------------------------*/
#ifdef __STDC__
void HP_Menu_VersionIndex (Document document, View view)
#else /* __STDC__*/
void HP_Menu_VersionIndex (document, view)
Document document;
View view;
#endif /* __STDC__*/
{
Element rootHP, teste;
Element elem, BasicUnit, elemr;
Element target;
ElementType ElemType, ElemType1;
SRelCPD Rel;
SEntry *Entry;
SIndex Index;
Element CPred_def, CPred_ref;
char *RelationName, *TextContent, *Doc_IV_Name, *Doc_Target_Name[100], label[100];
char slash[1], extension[3];
int length;
Language lang;
Document Doc_IV, Doc_Target;
SEntry *aux1;
SRelCPD *aux2;
Attribute VersionRef;
AttributeType AttrType;
int attrKind;
View IVView;

strcpy(slash, "/");
// Incializa a estrutura que aramazenará os elementos que figurarão no índice
InitIndex(&Index);
```

```
// rootHP referencia o elemento correspondente à raíz do documento, neste caso o
elemento HyperPro
rootHP = TtaGetMainRoot(document);
// Armazena em ElemType informação do tipo Relation_title
ElemType.ElSSchema = TtaGetSSchema("HyperPro", document);
TtaGiveTypeFromName(&ElemType, "Relation_title");
// rootHP referencia o primeiro elemento do tipo Relation_title do documento
rootHP = TtaSearchTypedElement(ElemType, SearchInTree, rootHP); //rootHP =
Relation_title
// Enquanto existir elemento do tipo relation_title no documento executa o loop
ElemType = TtaGetElementType(rootHP);
while(rootHP != NULL){
// Cpred_def referencia a referência de predicado corrente da relação em processo
CPred_def = TtaGetFirstChild(rootHP);
TtaNextSibling(&CPred_def);
TtaNextSibling(&CPred_def);
// Relation_name recebe o nome da relação
rootHP = TtaGetFirstChild(rootHP); //rootHP = Relation_name
ElemType1 = TtaGetElementType(rootHP);
elem = TtaGetFirstChild(rootHP);
length = TtaGetTextLength(elem) + 1;
RelationName = TtaGetMemory(length);
TtaGiveTextContent(elem, RelationName, &length, &lang);
strcpy(Rel.RelationName, RelationName);
TtaNextSibling(&rootHP); //rootHP = Relation_arity
elem = TtaGetFirstChild(rootHP);
length = TtaGetTextLength(elem) + 1;
RelationName = TtaGetMemory(length);
TtaGiveTextContent(elem, RelationName, &length, &lang);
strcat(Rel.RelationName, slash);
strcat(Rel.RelationName, RelationName);
rootHP = TtaGetParent(rootHP); //rootHP = Relation_title
TtaNextSibling(&rootHP); //rootHP = Versions_bar
rootHP = TtaGetFirstChild(rootHP); //rootHP = Version
// Enquanto houver elemento Version na Versions_bar em processo executa o loop
while (rootHP != NULL){
TtaGiveAttributeTypeFromName("Version_ref", rootHP, &AttrType, &attrKind);
VersionRef = TtaGetAttribute(rootHP, AttrType);
BasicUnit = TtaGetFirstChild(rootHP);
length = TtaGetTextLength(BasicUnit) + 1;
TextContent = TtaGetMemory(length);
TtaGiveTextContent(BasicUnit, TextContent, &length, &lang);
TtaGiveReferenceAttributeValue(VersionRef, &CPred_ref, Doc_Target_Name, &Doc_Target);
// Rel.CPD recebe a referência da definição de predicado corrente
Rel.CPD = CPred_ref;
// Procura a entrada correta do índice
SearchEntry(TextContent, &Index, &Entry);
// Insere na entrada correta
InsertRelCPD(Rel, Entry);
// rootHP referencia a próxima versão da barra de versões
TtaNextSibling(&rootHP);
}
rootHP = elem;
// rootHP referencia o próximo elemento Relation_title do documento
rootHP = TtaSearchTypedElement(ElemType, SearchForward, rootHP); //rootHP =
Relation_title
}

// Fecha a vista do índice de versões se esta se encontra aberta
IVView = TtaGetViewFromName(document, "IV_view");
TtaCloseView(document, IVView);
// rootHP referencia a raíz do documento
rootHP = TtaGetMainRoot(document);
ElemType.ElSSchema = TtaGetSSchema("HyperPro", document);
TtaGiveTypeFromName(&ElemType, "IVIndex");
// rootHP referencia o elemento IVIndex do documento
rootHP = TtaSearchTypedElement(ElemType, SearchInTree, rootHP); //rootHP = IVIndex
// Apaga a estrutura anteriormente construída para o elemento CRIndex
TtaDeleteTree(rootHP, document);
// Cria um novo elemento IVIndex
elem = TtaNewTree(document, ElemType, "IVIndex");
// rootHP referencia aa raíz do documento
rootHP = TtaGetMainRoot(document);
TtaGiveTypeFromName(&ElemType, "CRIndex");
// rootHP referencia o elemento CRIndex
rootHP = TtaSearchTypedElement(ElemType, SearchInTree, rootHP); //rootHP = CRIndex
// O elemento IVIndex é inserido a esquerda do elemento CRIndex na estrutura do
documento
TtaInsertSibling(elem, rootHP, False, document);
```

```
TtaGiveTypeFromName(&ElemType, "IV_Entries");
rootHP = TtaSearchTypedElement(ElemType, SearchForward, rootHP); //rootHP = IV_Entries
rootHP = TtaGetFirstChild(rootHP); //rootHP = Versions_lists
elem = rootHP;
TtaGiveTypeFromName(&ElemType, "RelPag");
ElemType1.ElSSchema = TtaGetSSchema("HyperPro", document);
TtaGiveTypeFromName(&ElemType1, "Versions_lists");
aux1 = Index.FirstEntry->Next;
// Enquanto houverem entradas no elemento Index executa o loop
while(aux1 != NULL){
elem = TtaGetFirstChild(elem); //elem = Version_Name
BasicUnit = TtaGetFirstChild(elem);
// Armazena em Version_Name o nome da versão
TtaSetTextContent(BasicUnit, aux1->VersionName, lang, document);
TtaNextSibling(&elem); //elem = RelPag_list
elemr = TtaGetFirstChild(elem); //elemr = RelPag
elem = elemr;
// Acessa a primeira estrutura (relação e CPD) da entrada em processo de Index
aux2 = aux1->FirstRelCPD->Next;
// Enquanto houver relações que participam da versão em processo executa o loop
while(aux2 != NULL){
elem = TtaGetFirstChild(elem); //elem = Relation_name
BasicUnit = TtaGetFirstChild(elem);
TtaSetTextContent(BasicUnit, aux2->RelationName, lang, document);
TtaNextSibling(&elem); //elem = IV_Page_CPD_ref
TtaSetReference(elem, document, aux2->CPD, document);
aux2 = aux2->Next;
// Se ainda existe relação que participe da versão cria um novo elemento RelPag para
recebê-la
if(aux2 != NULL){
teste = TtaNewTree(document, ElemType, label);
elem = teste;
TtaInsertSibling(elem, elemr, False, document);
elemr = elem;
}
}
// aux1 referencia nova entrada de Index
aux1 = aux1->Next;
// Se ainda existe versão cria novo elemento Versions_list para armazená-la
if(aux1 != NULL){
elem = TtaNewTree(document, ElemType1, label);
TtaInsertSibling(elem, rootHP, False, document);
rootHP = elem;
}
}
// Abre a vista de Ïndice de Versões
IVView = TtaOpenView(document, "IV_view", 197, 2, 170, 230);
// Define o título da vista como “Versions index”
TtaChangeViewTitle(document, IVView, "Versions index");

}
```

7.6 Anexo F

Este anexo mostra o código comentado da função que cria o índice de Referência Cruzada.

```
/*----------------------------------------------------------------------
----------------------------------------------------------------------*/
#ifdef __STDC__
void HP_Menu_PredCrossRefIndex (Document document, View view)
#else /* __STDC__*/
void HP_Menu_PredCrossRefIndex (document, view)
Document document;
View view;
#endif /* __STDC__*/
{
char slash[1], extension[3];
SCRIIndex Index;
Element rootHP, Elaux1, Elaux2, Elaux3, CPred_def, RelationTitle, elem, BasicUnit,
elemr;
ElementType ElemTypeaux1, ElemTypeaux2;
```

```
char *RelationName, *arity, Doc_Target_Name[100], RelationNameArity[40], *Doc_CRI_Name,
*label;
SRelation Relation;
Element Call;
SCRIEntry *Entry;
int length;
Language lang;
Document Doc_Target, Doc_CRI;
SCRIEntry *aux1;
SCall *aux2;
View CRIView;

strcpy(slash, "/");
// Inicializa a estrutura que armazenará os elementos que participarão do índice
CRIInitIndex(&Index);
// rootHP referencia o elemento correspondente a raíz do documento, neste caso o elemento
HyperPro
rootHP = TtaGetMainRoot(document);
// Armazena em ElemTypeaux1 informação do tipo Relation_title
ElemTypeaux1.ElSSchema = TtaGetSSchema("HyperPro", document);
TtaGiveTypeFromName(&ElemTypeaux1, "Relation_title");
// Elaux1 referencia o primeiro elemento do tipo Relation_title do documento
Elaux1    =    TtaSearchTypedElement(ElemTypeaux1,    SearchInTree,    rootHP);    //Elaux1
=Relation_title
ElemTypeaux1 = TtaGetElementType(Elaux1);

    /* Enquanto existir elemento Relation_title no documento armazena as informações de

                        cada relação em      Index */
while(Elaux1 != NULL){
RelationTitle = Elaux1;
// Acessa a referência de predicado corrente do elemento Relation_title e armazena em
Cpred_def
CPred_def = TtaGetFirstChild(Elaux1);
TtaNextSibling(&CPred_def);
TtaNextSibling(&CPred_def);
// Armazena em elem o elemento referenciado pela referência de predicado corrente
TtaGiveReferredElement(CPred_def, &elem, Doc_Target_Name, &Doc_Target);
// Armazena em RelationName o string correspondente ao nome da relação
Elaux1 = TtaGetFirstChild(Elaux1); //Elaux1 = Relation_name
Elaux2 = TtaGetFirstChild(Elaux1);
length = TtaGetTextLength(Elaux2) + 1;
RelationName = TtaGetMemory(length);
TtaGiveTextContent(Elaux2, RelationName, &length, &lang);
// Armazena em RelationNameArity o nome da relação concatenado com / e sua aridade
TtaNextSibling(&Elaux1); //Elaux1 = Relation_arity
Elaux2 = TtaGetFirstChild(Elaux1);
length = TtaGetTextLength(Elaux2) + 1;
arity = TtaGetMemory(length);
TtaGiveTextContent(Elaux2, arity, &length, &lang);
strcpy(RelationNameArity, RelationName);
strcat(RelationNameArity, slash);
strcat(RelationNameArity, arity);
strcpy(Relation.Name, RelationNameArity);
// Armazena na estrutura correspondente à uma entrada do índice o título e o cpd da
relação
Relation.Title = RelationTitle;
Relation.CPD = elem;
// Insere a estrutura correspondente à uma entrada do índice na estrutura do índice
CRIInsertEntry(Relation, &Index);
// Armazena em Elaux1 o próximo elemento Relation_title do documento
Elaux1   =   TtaSearchTypedElement(ElemTypeaux1,   SearchForward,   Elaux1);   //Elaux1   =
Relation_title
}
// Armazena em ElemTypeaux1 as informações correspondentes ao tipo Relation_def_ref
ElemTypeaux1.ElSSchema = TtaGetSSchema("HyperPro", document);
TtaGiveTypeFromName(&ElemTypeaux1, "Relation_def_ref");
// Armazena em Elaux1 a referência para o primeiro elemento Relation_def_ref do
documento
Elaux1   =   TtaSearchTypedElement(ElemTypeaux1,   SearchInTree,   rootHP);   //Elaux1   =
Relation_def_ref
ElemTypeaux1 = TtaGetElementType(Elaux1);
// Enquanto existem elementos Relation_def_ref no documento executa o loop
while(Elaux1 != NULL){
// Armazena em elem o elemento referenciado por Elaux1
TtaGiveReferredElement(Elaux1, &elem, Doc_Target_Name, &Doc_Target);
// Calll referencia o elemento correspondente à chamada da relação
Call = TtaGetParent(Elaux1);
Call = TtaGetParent(Call);
```

```
// Procura no índice a entrada de inserção
CRISearchEntry(elem, &Index, &Entry);
// Insere na entrada correta do índice
CRIInsertCall(Call, Entry);
// Elaux1 referencia o próximo elemento do tipo Relation_def_ref do documento
Elaux1   =   TtaSearchTypedElement(ElemTypeaux1,   SearchForward,   Elaux1);   //Elaux1   =
Relation_def_ref
}
// Fecha a vista do índice de referência cruzada se esta se encontra aberta
CRIView = TtaGetViewFromName(document, "CRI_view");
TtaCloseView(document, CRIView);
// rootHp referencia a raíz do documento
rootHP = TtaGetMainRoot(document);
// Armazena em ElemTypeaux1 informações sobre o tipo CRIndex
ElemTypeaux1.ElSSchema = TtaGetSSchema("HyperPro", document);
TtaGiveTypeFromName(&ElemTypeaux1, "CRIndex");
// rootHp referencia o elemento CRIndex do documento
rootHP = TtaSearchTypedElement(ElemTypeaux1, SearchInTree, rootHP); //rootHP = CRIndex
// Apaga a estrutura anteriormente construída para o elemento CRIndex
TtaDeleteTree(rootHP, document);
// Cria um novo elemento CRIndex
elem = TtaNewTree(document, ElemTypeaux1, "LCRIndex");
// rootHP referencia a raíz do documento
rootHP = TtaGetMainRoot(document);
// Armazena as informações sobre o tipo Document_title em ElemTypeaux1
TtaGiveTypeFromName(&ElemTypeaux1, "Document_title");
// rootHP referencia o primeiro elemento  do tipo Document_title do documento
rootHP   =   TtaSearchTypedElement(ElemTypeaux1,   SearchInTree,   rootHP);   //rootHP   =
Document_title
// Posiciona o elemento CRIndex a esquerda do elemento Document_title
TtaInsertSibling(elem, rootHP, False, document);
// rootHP referencia o elemento Relations_list do elemento CRIndex
TtaGiveTypeFromName(&ElemTypeaux1, "CRI_Entries");
rootHP   =   TtaSearchTypedElement(ElemTypeaux1,   SearchForward,   rootHP);   //rootHP   =
CRI_Entries
rootHP = TtaGetFirstChild(rootHP); //rootHP = Relations_lists
// Deste ponto até o final são inseridas em CRIndex as informações armazenadas na
estrutura Index
elem = rootHP;
TtaGiveTypeFromName(&ElemTypeaux1, "Use");
// ElemTypeaux2 armazena as informações do tipo Relations_lists
ElemTypeaux2.ElSSchema = TtaGetSSchema("HyperPro", document);
TtaGiveTypeFromName(&ElemTypeaux2, "Relations_lists");
// aux1 referencia a primeira entrada da estrutura Index
aux1 = Index.FirstEntry->Next;
while(aux1 != NULL){
elem = TtaGetFirstChild(elem); //rootHP = CRI_Relation_Name
BasicUnit = TtaGetFirstChild(elem);
// Armazena em CRI_Relation_name o nome da relação
TtaSetTextContent(BasicUnit, aux1->Relation.Name, lang, document);
// Page_Relation_title_ref tem a referência definida
TtaNextSibling(&elem); //elem = Page_Relation_title_ref
TtaSetReference(elem, document, aux1->Relation.Title, document);
// CRI_Page_CPD_ref tem areferência definida
TtaNextSibling(&elem); //elem = CRI_Page_CPD_ref
TtaSetReference(elem, document, aux1->Relation.CPD, document);
// elem referencia a lista de chamadas da relação em processo
TtaNextSibling(&elem); //elem = Uses_list
// aux2 referencia a lista de chamadas da estrutura correspondente à entrada em Index
aux2 = aux1->FirstCall->Next;
elemr = TtaGetFirstChild(elem); //elemr = Use
elem = elemr;
// Enquanto houverem chamadas da relação executa o loop
while(aux2 != NULL){
// Use tem a referência definida
TtaSetReference(elem, document, aux2->Call, document);
aux2 = aux2->Next;

    /* Se tem mais chamadas cria outro elemento Use para armazenar a referência e insere

                                        ao lado do  Use anterior */
if(aux2 != NULL){
elem = TtaNewTree(document, ElemTypeaux1, label);
TtaInsertSibling(elem, elemr, False, document);
elemr = elem;
}
}
// aux1 referencia a nova entrada de Index
aux1 = aux1->Next;
```

```
    /* Se ainda existem entradas em Index cria novo elemento Relations_lists e insere na
                                        árvore do documento */
if(aux1 != NULL){
elem = TtaNewTree(document, ElemTypeaux2, label);
TtaInsertSibling(elem, rootHP, False, document);
rootHP = elem;
}
}
// Abre a vista do Índice de Referência Cruzada
CRIView = TtaOpenView(document, "CRI_view", 177, 2, 170, 230);
// Define o título da vista como “Predicate cross-reference index”
TtaChangeViewTitle(document, CRIView, "Predicate cross-reference index");
}
```

7.7 Anexo G

Este anexo mostra o código comentado da função que cria as Projeções Baseadas em Índices.

```
#ifdef __STDC__
void HP_Menu_IndexBasedProjection (Document document, View view)
#else /* __STDC__*/
void HP_Menu_IndexBasedProjection (document, view)
Document document;
View view;
#endif /* __STDC__*/
{
Document doc;
Element element, relpaglist, relpag, pageCPDref, CPD, predicatedef, relationdef,
relationslists,
use, versionslists;
int firstCharacter, lastCharacter;
Attribute attribute;
char documentname[100];

// Seta as variáveis globais da aplicação
HP_Util_Setup(document);

//                  TtaDisplayMessage(INFO,                  TtaGetMessage(MessageTable,
MESSAGES_CREATEMANUALPROJECTION));

TtaClickElement(&doc, &element);

// Se nenhum elemento foi selecionado sai da função
if (!doc || !element)
return;

/* Set global types,... for this document
*/

HP_Util_Setup(doc);

/* Se houver algum ancestral que seja Paragraph ou entao realtion def, ele pega os
respectivos. */

/* ************** tit!
PRINT("=====");
TYPE(element);
*/
/* Se o elemento selecionado possui um ancestral do tipo Versions_lists executa a
         projeção baseada no Índice de Versões */
versionslists = TtaGetTypedAncestor(element, type_Versions_lists);



if (versionslists){
// Remove os atributos de visibilidade anteriormente adicionados
HP_Util_RemoveAttributes(document, attr_Index_IV_projection_visible);
/* ************** tit!
PRINT("-----");
TYPE(relpaglist);
```

```
*/
// relpaglist referencia o elemento RelPag_list
relpaglist = TtaGetLastChild(versionslists);
// relpag referencia o primeiro elemento RelPag de RelPag_list
relpag = TtaSearchTypedElement(type_RelPag, SearchForward, relpaglist);
// Enquanto houver relação participando da versão eme processo executa o loop
while(relpag != NULL){
// pageCPDref referencia IV_Page_CPD_ref
pageCPDref = TtaGetLastChild(relpag);
/* predicatedef recebe a referência para a definição de predicado referenciada por
              pageCPDref */
TtaGiveReferredElement(pageCPDref, &predicatedef, documentname, &doc);
// relationdef referencia o elemento Relation_def ancestral de predicatedef
relationdef = TtaGetTypedAncestor(predicatedef, type_Relation_def);
// Se nÃo houver atributo de visibilidade na definição de relação o adiciona
attribute = TtaGetAttribute(relationdef, attr_Index_IV_projection_visible);
if (!attribute){
// Cria novo atributo de visibilidade
attribute = TtaNewAttribute(attr_Index_IV_projection_visible);
// Associa o atributo de visibilidade à definição de relação
TtaAttachAttribute(relationdef, attribute, document);
}
// relpag referencia a próxima relação participanteda versão em processo
TtaNextSibling(&relpag);
}
// Abre avista da projeção baseada no Índice de Versões
HP_Util_OpenProj(document, 'I');
}
// Se o elemento selecionado não for do tipo Relations_lists
else{
// Inicializa as variáveis globais da aplicação
HP_Util_Setup(document);
/* Se o elemento selecionado possui um ancestral do tipo Versions_lists executa projeção
              baseada no Índice Versões */
relationslists = TtaGetTypedAncestor(element, type_Relations_lists);
if (relationslists){
// Remove os atributos de visibilidade anteriormente adicionados
HP_Util_RemoveAttributes(document, attr_Index_CRI_projection_visible);
/* ************** tit!
PRINT("-----");
TYPE(relationslists);
*/
// use referencia a lista de chamadas da entrada escolhida do índice (Uses_list)
use = TtaGetLastChild(relationslists);
// pageCPDref referencia o elemento CRI_Page_CPD_ref
pageCPDref = use;
TtaPreviousSibling(&pageCPDref);
// predicatedef referencia o elemento referenciado por pageCPDref
TtaGiveReferredElement(pageCPDref, &predicatedef, documentname, &doc);
// relationdef referencia o elemento Relation_def ancestral de predicatedef
relationdef = TtaGetTypedAncestor(predicatedef, type_Relation_def);
// Se não houver o atributo de visibilidade na definição de relação o adiciona
attribute = TtaGetAttribute(relationdef, attr_Index_CRI_projection_visible);
if (!attribute){
// Cria novo atributo de visibilidade
attribute = TtaNewAttribute(attr_Index_CRI_projection_visible);
// Associa o atributo de visibilidade à definição de relação
TtaAttachAttribute(relationdef, attribute, document);
}
// use referencia a primeira chamada da relação em processo
use = TtaGetFirstChild(use);
// Enquanto houver chamada executa o loop
while(use != NULL){
// predicatedef referencia a definição de predicado referenciada por use
TtaGiveReferredElement(use, &predicatedef, documentname, &doc);
// predicatedef referencia o elemento Predicate_def ancestral de predicatedef
predicatedef = TtaGetTypedAncestor(predicatedef, type_Predicate_def);
// relationdef referencia o elemento Relation_def ancestral de predicatedef
relationdef = TtaGetTypedAncestor(predicatedef, type_Relation_def);
// Se não houver o atributo de visibilidade na definição de relação o adiciona
attribute = TtaGetAttribute(relationdef, attr_Index_CRI_projection_visible);
if (!attribute){
// Cria novo atributo de visibilidade
attribute = TtaNewAttribute(attr_Index_CRI_projection_visible);
// Associa o atributo de visibilidade à definição de relação
TtaAttachAttribute(relationdef, attribute, document);
}
// use referencia a próxima chamada de relação
TtaNextSibling(&use);
```

```
}
// Abre a projeção baseada no índice de referência cruzada
HP_Util_OpenProj(document, 'C');
} else return;
        }
}
```