BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
UFMG-ICEX-DCC

RELATÓRIO FINAL DE INICIAÇÃO CIENTÍFICA

UM AMBIENTE PARA DESENVOLVER PROGRAMAÇÃO EM LÓGICA BASEADO NO PARADIGMA DE ESTILO LITERÁRIO

Orientadora : Mariza Andrade da Silva Bigonha
Aluna : Flávia Peligrinelli Ribeiro

Data : 31 de agosto de 1998

Sumário

1 Introdução

O projeto de pesquisa para o qual apresentamos este relatório é um subprojeto de um projeto maior que engloba a construção da segunda versão da ferramenta `HyperPro` [1, 2, 3, 4, 5], denominada `HyperPro Básico`. Este projeto faz parte do programa de cooperação internacional entre o *Institut National de Recherche en Informatique e Automatique* (INRIA) e o Departamento de Ciência da Computação da UFMG.

Documentação de *software* é um eterno problema, e ainda não existe uma solução satisfatória. É sabido que documentação de programas tendem a se constituir em uma grande coleção de arquivos, o que torna difícil manter sua consistência em relação ao programa correspondente. Um bom sistema de documentação deve oferecer recursos para ajudar a manutenção da consistência devendo ser compilável e processável por um computador.

Um dos objetivos do projeto `HyperPro` é auxiliar o projetista de grandes programas Prolog a melhorar a qualidade de seus programas por meio de comentários coerentes baseados em uma metodologia consistente. Para atingir esse objetivo era necessário criar mecanismos que facilitassem a escrita e depuração de programas e textos no ambiente de programação em lógica `HyperPro`. Na primeira fase desse projeto, 1996, foi desenvolvido um sistema de documentação, o sistema `HyperPro`, de programas em Prolog [13]. Durante essa fase, meu envolvimento com o projeto consistiu no estudo de ferramentas relacionadas com a programação literária [14, 15, 16, 17, 18, 19], o sistema `Thot` [7, 8, 9] e o próprio sistema `HyperPro`, conforme pode ser visto em [6].

No presente projeto, HyperPro Básico, a idéia é enriquecer o sistema `HyperPro` estendendo-o para prover outras facilidades necessárias a produção de documentação de boa qualidade.

2 Trabalho Proposto

O `HyperPro` é um ambiente experimental que foi projetado para desenvolver programação em lógica baseado em programação literária, especificadamente na ferramenta Grif-Thot. A programação literária enfatiza a escrita de programas para serem lidos por indivíduos ao invés de computadores. Ela consiste em um único arquivo de entrada contendo documentação e programas. O ponto principal da programação literária advém do fato de que o código fonte e sua respectiva documentação são escritos simultaneamente.

O objetivo do HyperPro é documentar programas CLP (*Constraint Logic Programming*[20]) oferecendo ao usuário a possibilidade de editar, em um ambiente homogêneo e integrado, diferentes programas e versões de programas, comentários, informações de verificação formal, assim como a possibilidade de executar, depurar e testar os programas. Nesse sistema, toda a história de desevolvimento de um programa CLP pode ficar registrado em sua documentação, este fato permite a presença de várias versões. Para prover todas estas facilidades, foi desenvolvida uma estrutura genérica de documentos para programas lógicos de acordo com a metodologia de programação em lógica. Os programas são vistos como documentos executáveis. As funções básicas do HyperPro atualmente disponíveis permitem:

1. diferentes visões do documento;
2. testar programas e versões documentadas;
3. índices manuais;
4. exportações de documentos para outros formatos como por exemplo: Latex [22] e ASCII;
5. verificação sintática em diferentes linguagens CLP [20].

As diferentes visões ou níveis de abstrações já implementadas no `HyperPro` são:

- Vista de comentário,
- vista de asserção,
- vista de tipos,
- vista de programas,
- tabela de conteúdo.

No projeto atual, nosso trabalho consiste na implementação de um novo conjunto de vistas, a saber:

- vista de projeção manual,
- vista de projeção recursiva,
- vista de projeção automática,
- vista de projeção baseada em índices.

Nos próximos capítulos são mostrados o sistema `HyperPro`, o desenvolvimento de nosso trabalho, nossa contribuição e as tarefas a serem desenvolvidas na próxima fase do projeto de iniciação científica, de agosto/98 até dezembro/98.

3 HyperPro

Um documento HyperPro [1, 2, 3, 4, 5] é basicamente um documento `Thot`. Ele possui um título, pelo menos uma seção, uma tabela de conteúdo e um índice de referência cruzada.

Em um documento HyperPro, um parágrafo também pode ser uma definição de relação. Uma definição de relação é definida por um título e uma lista de pelo menos uma definição de predicado. O título é um indicador de predicado, nome do predicado e sua aridade, ou um nome.

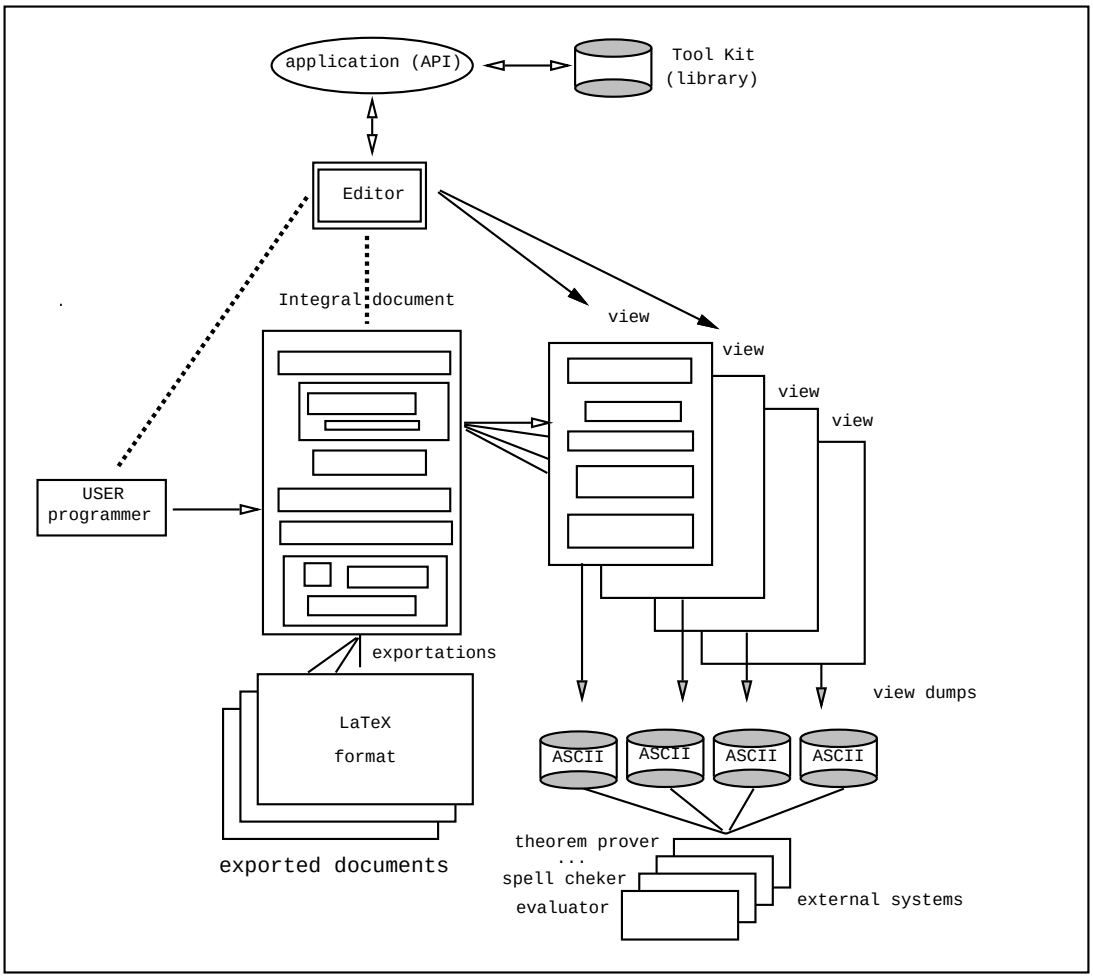

Figura 1: Arquitetura do HyperPro

A Figura 1 ilustra a arquitetura do HyperPro. Ele utiliza-se de um editor de texto, o `Thot`, e sua API [11]. As API's (*Aplication Program Interface*) do `Thot` permitem desenvolver aplicações específicas que potencialmente podem atuar na edição de um documento. O kit de ferramentas do `Thot` é um conjunto de funções de edição, escritas em C, que pode ser usado na construção das API's; tais funções executam operações em ambientes estruturados Unix X-Window.

O usuário entra com programas escritos na linguagem S [10] e na linguagem P [10] que definem a estrutura genérica do documento. O documento todo é visualizado em uma só vista integral do documento. Além desta vista, mais quatro vistas podem ser definidas como mostrado em 2. HyperPro permite definir diferentes esquemas de exportação de documentos na forma canônica para outros tipos de formalismos, por exemplo, LaTex, ascii, etc.

3.1 Funcionalidades do HyperPro

As principais funções do HyperPro são os índices e as vistas de diferentes partes do documento associados a diferentes utilidades tais como, teste de programas e verificação sintática de linguagens lógicas.

3.1.1 Vistas

O documento pode ser visto por meio de várias perspectivas chamadas vistas, que são especificadas na apresentação genérica. Essas vistas são formas de visualização de partes específicas do programa que são úteis ao programador durante o estágio do desenvolvimento da aplicação, por exemplo, a parte dos comentários.

O documento todo é visualizado em uma única vista, denominada vista integral do documento. As vistas podem ser abertas simultaneamente e são automaticamente sincronizadas. Ao invés de escrever na vista integral, o usuário pode editar em uma vista específica, e a informação aparece nas demais vistas abertas. Quatro tipos de vistas foram especificadas: comment_view, assertion_view, typing_view e program_view. As Figuras 2, 3, 4, 5, 6 mostram as vistas dos comentários, asserções, tipos, programa e tabela de conteúdos respectivamente.

Comment_View : Vista dos comentários que permite ao usuário ver apenas os comentários relativos às definições de predicado.

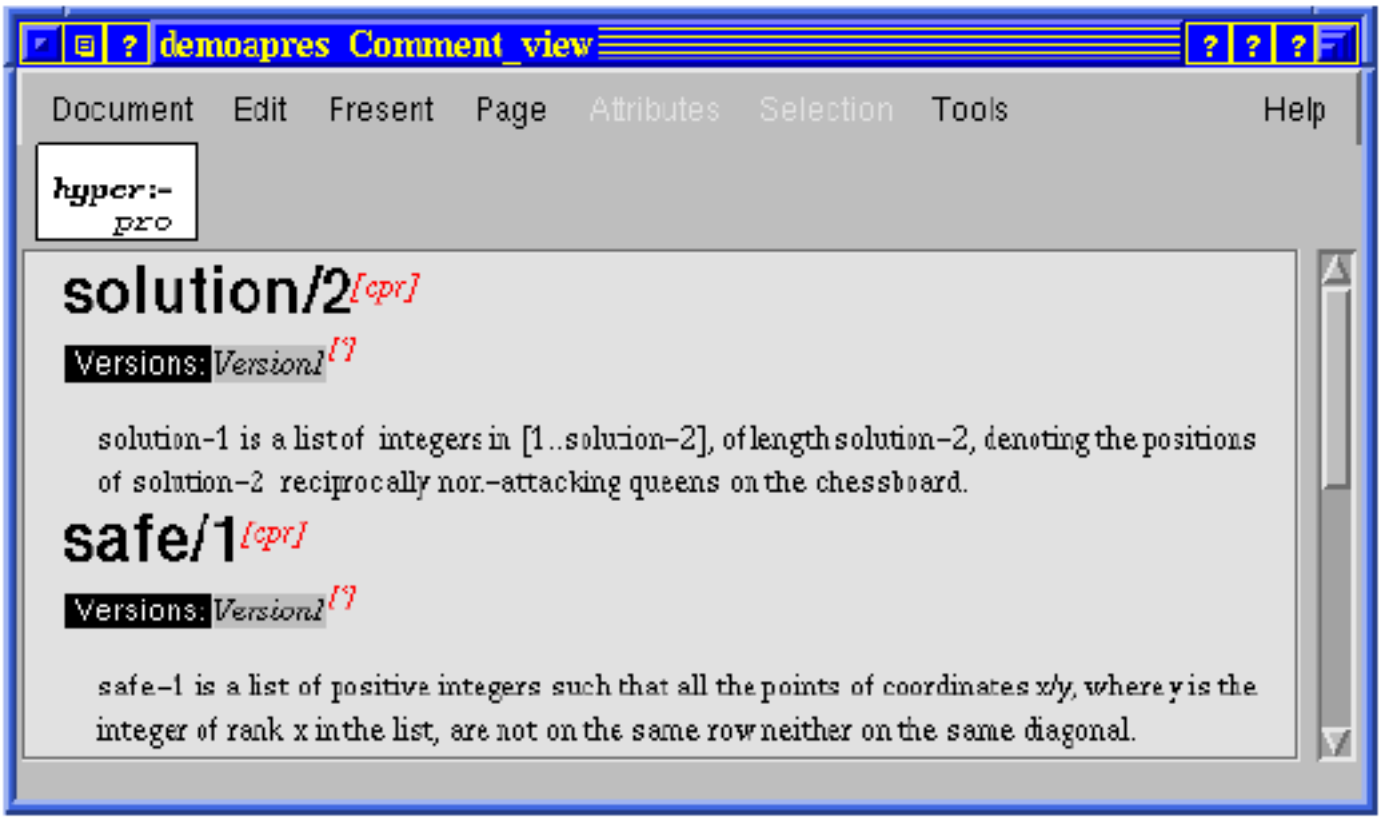

Figura 2: Visão de Comentários

Assertion_View : Vista das asserções que permite o usuário visualizar exclusivamente as partes de asserções relativas às definições de predicado.

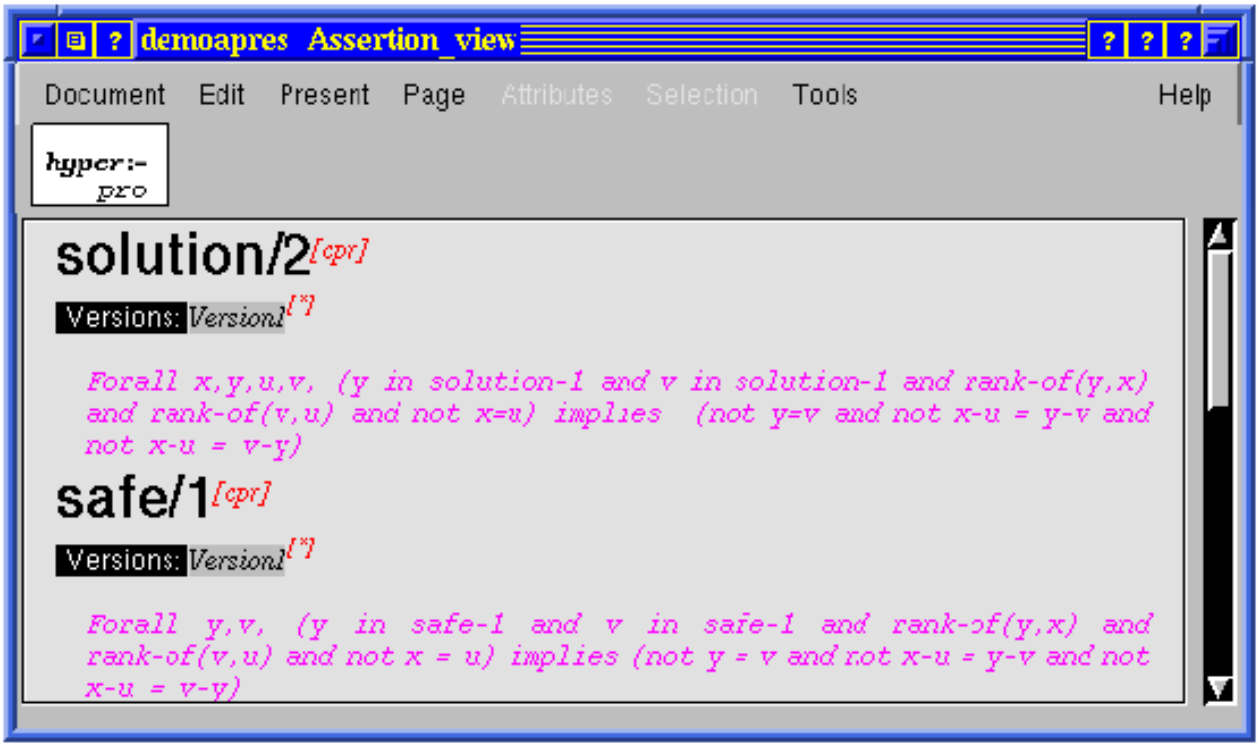

Figura 3: Visão de Asserção

Typing_View : Vista de tipos que permite o usuário visualizar apenas os tipos relacionados às definições de predicado.

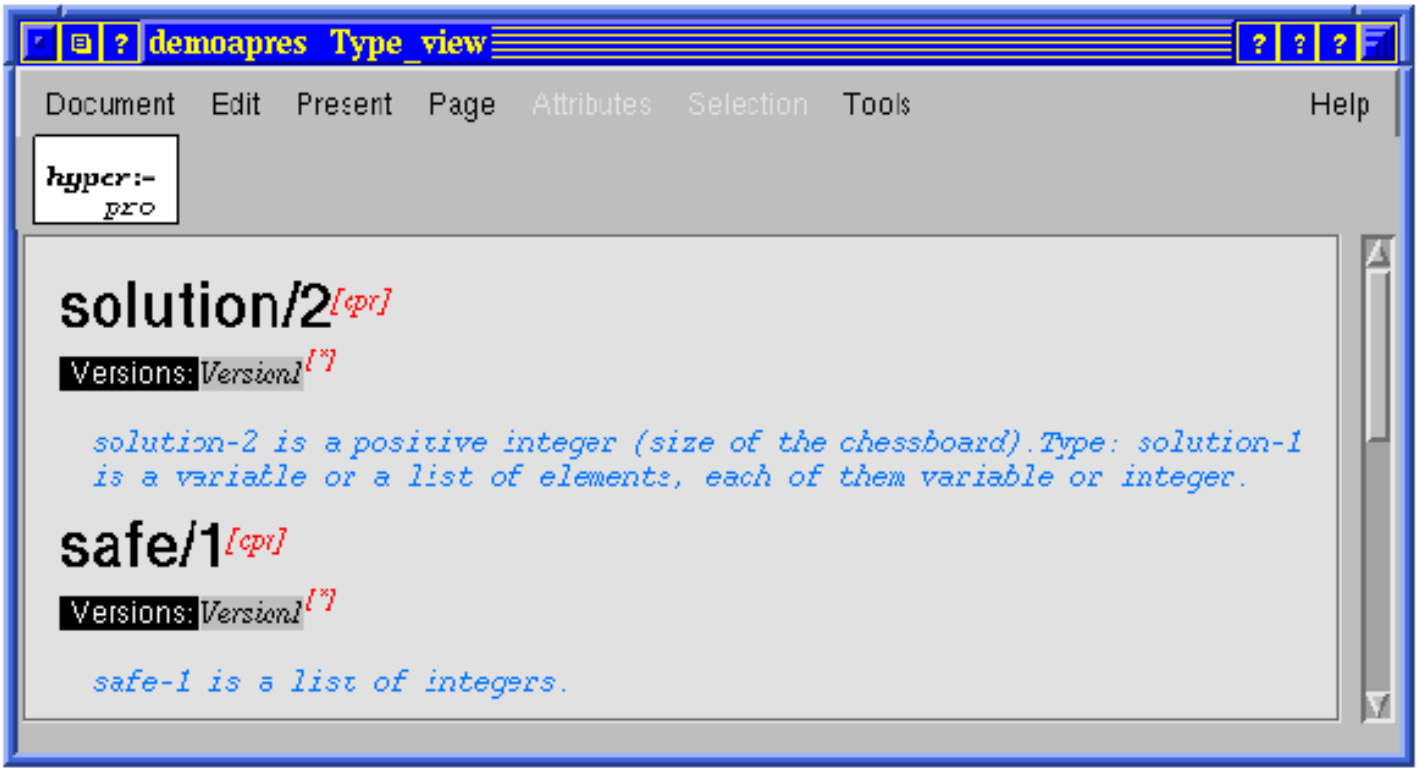

Figura 4: Visão de Tipos

Program_View : Vista do programa que permite o usuário ver somente as partes de *clauses* e definição de predicados.

Figura 5: Visão de Programas

Table_of_contents : Vista do programa que permite que o usuário veja a tabela de conteúdos.

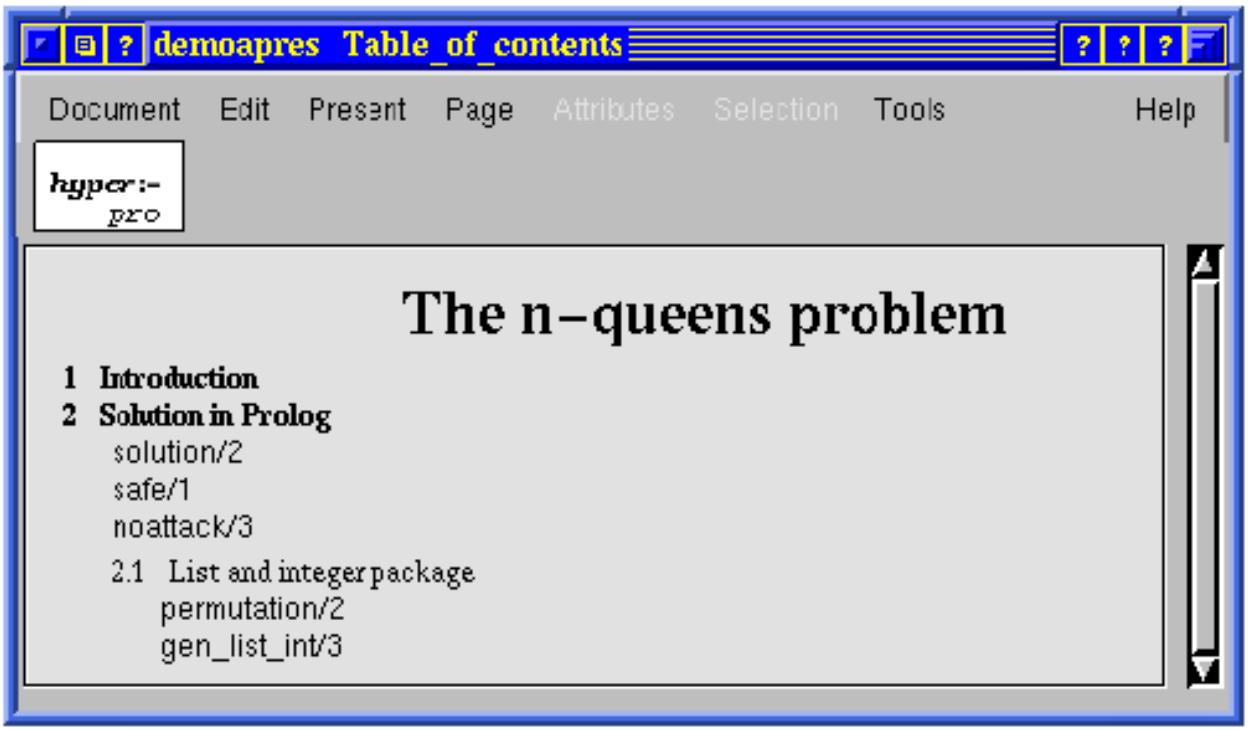

Figura 6: Visão da Tabela de Conteúdos

O usuário pode especificar a visibilidade de um elemento e também a apresentação de certos elementos que aparecerão em cada vista.

3.1.2 Programas e Versões

Um programa é um conjunto de pacotes de cláusulas. Um documento pode conter diferentes programas. O usuário decide como o documento estará organizado, definindo seus programas por meio das seções. O usuário pode definir seus programas selecionando convenientemente no documento suas definições de predicado e colocando referências.

O HyperPro permite que o usuário teste manualmente e automaticamente seu programa.

O usuário pode definir para qualquer relação de definição, diferentes versões de uma definição de predicado que é documentada e gerenciada com as mesmas utilidades usadas para definir programas. De fato, versões de programas são programas que diferem em pelo menos uma cláusula.

3.1.3 Índices

O HyperPro possui o seguinte tipo de índice: índice de referência cruzada, que indica onde a relação aparece no documento, onde a sua definição de predicado é encontrada e onde a relação é usada em outros programas ou versões no documento. O índice de programas e versões que mostra onde o programa e suas versões foram primeiramente definidos está sendo projetado pela equipe francesa.

3.1.4 Conversões e Exportações

O HyperPro é um sistema aberto, o que significa que ele pode trocar documentos com outros sistemas por meio de um mecanismo de exportação.

O HyperPro produz documentos em um nível abstrato chamado de forma canônica. Pode ser definida uma série de regras de tradução de documentos. Na versão atual, foram definidos dois esquemas diferentes de tradução: exportação de documentos para LaTex e exportação de documentos para ASCII. Também foram feitos esquemas de exportação de certas partes do documento: exportação do programa, exportação dos comentários informais, exportação das asserções e exportação dos tipos.

Os arquivos de tradução (.T) contém as regras de tradução que convertem o documento do formato PIVOT (.PIV) para os diferentes formalismos.

4 Desenvolvimento do Trabalho Proposto

4.1 O `Thot` e suas linguagens

Para programar as novas vistas apresentadas em 4.2.1 tivemos que estudar as linguagens A [12] e T [6, 10] do `Thot` [7, 8, 9] além de sua API [11]. O `Thot` é um sistema desenvolvido para produzir documentos estruturados. É um poderoso editor WYSIWYG (*What You See Is What You Get*) que possui entre outras, facilidades de hipertexto [8]. Ele permite que o usuário crie, modifique e consulte interativamente os documentos. Os modelos permitem a produção de documentos homogêneos. O `Thot` possui outras operações tais como numeração, referências cruzadas e índices. Todos os serviços que o `Thot` provê são baseados no sistema interno de representação de documentos.

O modelo de documento do `Thot` permite que o usuário trabalhe com certas entidades que ele tem em mente quando faz um documento. E são essencialmente essas entidades lógicas, por exemplo, parágrafos, seções, capítulos, notas, títulos e referências cruzadas que fornecem ao documento sua estrutura lógica. O modelo `Thot` é baseado no aspecto lógico dos documentos. As diferenças entre documentos não são apenas as entidades que neles aparecem, mas também as relações entre essas entidades e as formas como elas estão ligadas.

Para programar no Thot existe disponível no sistema quatro linguagens: S [6, 10], P [6, 10], T [6, 10] e A [12], responsáveis respectivamente pelo Esquema de Estrutura, Esquema de Apresentação, pelo Esquema de Tradução e pelo Esquema de Aplicação do documento.

A LINGUAGEM T

O Thot pode produzir documentos de forma abstrata em alto nível. Essa forma, chamada de forma canônica é específica do Thot. Ela se adapta bem às manipulações do editor, mas não se adapta necessariamente a outras operações que possam ser aplicadas aos documentos. Por esse motivo, o editor Thot oferece a escolha de salvar documentos na forma canônica ou em um formato definido pelo usuário. Neste caso, o documento Thot é transformado pelo programa de tradução. Essa facilidade pode também ser usada para exportar documentos do Thot para outros sistemas usando outros formalismos. Por exemplo, a tradução pode ser usada para exportar documentos para formatadores básicos como Tex, Latex, Scribe e troff. Também pode ser usado para traduzir documentos para SGML e HTML [21]. Essa tradução é feita via esquemas de tradução [6, 10].

```
{-------------------------------------------------------------}
{ Translation schema to generate the HTML version from an     }
{ HyperPro document.                                          }
{ Mariza       06/04/98 - Manha   -    DCC/ICEx/UFMG          }
{-------------------------------------------------------------}

TRANSLATION HyperPro ;
RULES
   HyperPro :
        BEGIN
        Use ParagraphH for Paragraphe;
        ...
        END;
   Document_title:
        BEGIN
        Create '\12<h1 ALIGN=CENTER>';
        Create '</h1>\12' After;
        Get Document_date After;
        Get Authors After;
        END;
   Document_date:
        BEGIN
          Create '<h4 ALIGN=RIGHT> ' ;
          Create '</h4>\12' After;
        END;
   ...
END
```

Exemplo 1

A LINGUAGEM A

No sistema `Thot`, a geração da aplicação baseia-se em `esquemas de aplicação` que são escritos em uma linguagem chamada A [12] (*Generation Application Language*). Esta linguagem é usada para a definição da interface gráfica e para a criação de menus os quais podem ser associados às funções padrões do sistema ou a novas funções específicas.

Uma aplicação é formada por comandos e ações. Os comandos são executados ao se escolher um item de menu e as ações são executadas quando os eventos aos quais elas estão associadas ocorrem. Ambos estão especificados nas seções DEFAULT, ELEMENTS e ATTRIBUTES do `esquema de aplicação`. Os comandos de edição do `Thot` geram dois tipos de eventos cujos nomes diferem pela extensão:

- `.Pre`: quando o comando padrão é chamado pelo usuário antes de ser processado pelo editor e retorna um booleano;
- `.Post`: quando a ação é executada depois que o editor executou o comando padrão e não possuem valor de retorno.

Um `esquema de aplicação` se relaciona a um `esquema de estrutura` e possui o mesmo nome deste com a extensão `.a`, ou então, é o esquema principal com o nome `EDITOR.a`, onde estão definidos os menus e comandos específicos associados.

Um `esquema de aplicação` começa com a palavra-chave APPLICATION seguida da palavra EDITOR ou do nome do `esquema de estrutura` relacionado e termina com a palavra-chave END.

Um `esquema de aplicação` pode ser composto de uma ou mais seções:

- DEFAULT: possui associação de eventos e ações e em geral se aplica a todos os tipos de elementos e a todos os atributos definidos no `esquema de estrutura` correspondente.
- ELEMENTS: contem ações que são chamadas por um elemento.
- ATTRIBUTES: define ações que são chamadas por um dado atributo.

Pelo menos uma dessas seções deve estar presente nos `esquemas de aplicação` associados.

O `esquema de aplicação` principal, `EDITOR.a`, possui ainda duas outras seções que não aparecem nos outros esquemas:

- USES: essa seção possui os nomes dos outros `esquemas de aplicação` bem como a lista com todos os módulos necessários à aplicação. Essa seção é opcional.
- MENUS: última seção do `EDITOR.a` que define os menus da barra de menus. Para cada item do menu, o `esquema de aplicação` associa um comando específico exceto para os menus padrão.

O conjunto de ferramentas do `Thot` permite a declaração de menus em cascata de um nível e ainda suporta a opção de vários idiomas, *multilingual dialogue*. Além dessas facilidades, o kit de ferramentas permite que o usuário execute comandos via teclado.

O exemplo abaixo mostra um trecho do `esquema de aplicação EDITOR.a` do protótipo do `HyperPro Básico`.

```
APPLICATION EDITOR;

USES
{ HyperPro schema }
HyperPro,
        ...
DEFAULT
  BEGIN
   Init.Post -> HP_Event_InitHyperPro;
   ViewOpen.Post -> HP_Event_AfterOpenView;
   ...
  END;
MENUS
   Main Window:
BEGIN
File button:BNew ->  TtcCreateDocument;
        ...
END;
   Document Windows:
BEGIN
...
Present.Views toggle:TSynchro -> HP_Menu_LocChangesynchronize;
...
   HyperPro Windows:
BEGIN
...
Tools.Indexes button:BTextIndex -> TtcIndex;
Tools.Indexes button:BPredCrossRefIndex -> HP_Menu_PredCrossRefIndex;
Tools.Indexes button:BVersionIndex -> HP_Menu_VersionIndex;
Tools separator;
Tools.Indexes button:BAllIndexes -> HP_Menu_AllIndexes;
Tools.Version button:BNameVersion -> HP_Menu_NameVersion;
Tools.Version button:BClearVersion -> HP_Menu_ClearVersion;
Tools.Projection button:BManual -> HP_Menu_ManualProjection;
Tools.Projection button:BRegExpression -> HP_Menu_RegExpressionProjection;
Tools.Projection button:BVersion -> HP_Menu_VersionProjection;
Tools.Projection button:BRecursive -> HP_Menu_RecursiveProjection;
Tools.Projection button:BIndexBased -> HP_Menu_IndexBasedProjection;
Tools.Test button:BVersion -> HP_Menu_VersionTest;
Tools.Test button:BClauses -> HP_Menu_ClausesTest;
Tools.Test button:BRelation -> HP_Menu_RelationTest;
Tools.SyntaxVerif button:BVersion -> HP_Menu_VersionSyntaxVerif;
Tools.SyntaxVerif button:BClauses -> HP_Menu_ClausesSyntaxVerif;
Tools.SyntaxVerif button:BRelation -> HP_Menu_RelationSyntaxVerif;
Help_ button:BHyperProInfo ->  HP_Menu_HyperProHelp;
Help_ button:BThotInfo ->  HP_Menu_ThotHelp;
END;
END
```

Exemplo 2

A API

O conjunto de feramentas do Thot é um conjunto de funções C que lidam com documentos estruturados no ambiente UNIX/ X Windows. Geralmente uma aplicação usa essas funções para, entre outras tarefas, criar documentos novos, modificar documentos existentes, extrair informações de documentos e mostrar partes de documentos.

O conjunto de ferramentas do Thot possui cerca de 200 funções colocadas em grupos onde cada um enfatiza um aspecto diferente do documento, todas baseadas no modelo Thot de documentos. Os grupos são:

- Grupo de Aplicação: possui funções de gerência da API.
- Grupo de Interface: permite que a aplicação estenda ou modifique o Thot
- Grupo de Mensagens: permite a aplicação gerenciar mensagens e caixas de diálogo.
- Grupo de Diálogo: com estas funções a aplicação é capaz de gerenciar menus e formulários.
- Grupo de Documentos: essas funções gerenciam os esquemas e todas as ações relacionadas ao documento em si.
- Grupo Árvore: essas funções lidam com operações relacionadas a árvore de estrutura que representa a organização lógica do documento.
- Grupo de Conteúdo: possui funções que manipulam as folhas da árvore de estrutura.
- Grupo de Atributos: contém funções que lidam com os atributos dos elementos.
- Grupo de Referência: funções que manipulam as referências do documento, *links* de hypertexto.
- Grupo de Linguagem: contém funções que gerenciam o idioma usado em um texto.
- Grupo de Apresentação: lida com a apresentação específica do elemento.
- Grupo de Vistas: possui funções que manipulam as várias vistas do documento.
- Grupo de Seleção: funções que manipulam a seleção de elementos.

Essas funções são acessadas por meio da API(*Aplication Programming Interface*) [11]. O conjunto de ferramentas do Thot é composto de duas bibliotecas: a biblioteca do *kernel* do Thot e a biblioteca do editor Thot. A primeira permite que a aplicação lide, de forma automática, com a estrutura lógica e o conteúdo do documento. A segunda contém todas as facilidades incluídas na primeira e ainda acrescenta funções que mostram o aspecto gráfico do documento.

4.2 HyperPro Básico

4.2.1 Especificação das Projeções

Uma das funcionalidades determinadas para o HyperPro Básico são as projeções. Em um ambiente integrado de programação e documentação, é importante que o usuário tenha uma forma de testar seu programa separado do resto do documento. Daí surge a idéia da projeção. Uma vista de projeção exibe

na tela porções selecionadas do documento em uma janela separada. O processo de seleção depende do tipo de projeção desejada. As projeções diferem das vistas no processo de seleção, as vistas, já estão incorporadas no `Thot` e as projeções têm que ser implementadas. Foram definidas quatro tipos de vistas de projeção:

Projeção Manual : o usuário pode selecionar livremente qualquer parte do documento que será visto.

Projeção Automática : o sistema pede para que o usuário entre com uma expressão regular e o `HyperPro Básico` mostra em uma vista separada todas as porções do documento em que esta expressão aparece.

Projeção Recursiva : o usuário seleciona uma ou mais relações e o `HyperPro` mostra em uma vista separada todos os pacotes de cláusulas CPR/PR (*Current Predicate Reference/ Predicate Reference*) as quais a relação está inserida. Esse tipo de vista é muito útil para se detectar predicados ainda não definidos ou a localização de suas definições.

Projeção Baseada em Índices : essa projeção permite ao usuário selecionar qualquer entrada de índice e mostra em uma vista separada todas as partes do documento relacionada ao índice.

4.2.2 Estrutura de Dados e Funções Usadas

Para implementar a projeção recursiva foi preciso implementar uma estrutura de dados que é uma lista encadeada e contem o elemento e um apontador para o próximo elemento.

Nas implmentações das projeções recursiva e manual, foram usadas funções da API do `Thot`.As funções utilizadas foram:

- `void TtaGiveFirstSelectedElement(Document document, Element *selectedElement, int *firstCharacter, int *lastCharacter)`
  retorna o primeiro elemento da seleção corrente em um dado documento.
- `View TtaOpenSubView(Document document, char *viewName, int x, int y, int w, int h, Element subtree)`
  mostra uma visão ccom apenas uma subárvore. A subávore cuja raiz é o elemento passado para a função.
- `void TtaChangeViewTitle(Document document, View view, char *title)`
  muda o título de uma visão.
- `void TtaGiveReferredElement(Element element, Element *target, char *targetDocumentName, Document *targetDocument)`
  retorna o elemento referenciado por uma dada referência.
- `ElementType TtaGetElementType(Element element)`
  retorna o tipo de um dado elemento.

- `Element TtaGetLastChild(Element parent)`
  retorna o valor do útimo filho de um dado elemento na árvore de estrutura.
- `Element TtaSearchReferenceElement(SearchDomain scope, Element element)`
  procura o próximo elemento que é uma referência.
- `Element TtaGetSuccessor(Element element)`
  retorna o elemento seguinte de um dado elemento no mesmo nível ou no nível acima.

4.2.3 Implementação da Projeção Manual

A implementação da projeção manual foi feita no arquivo EDITORactions.c. Nessa projeção o usuário pode selecionar qualquer parte do documento `HyperPro` e a projeção mostrará o elemento que contem o trecho marcado.

Para implementar a projeção manual não foi preciso adaptar as funções do `Thot`, foram utilizadas apenas as funções contidas na API do `Thot`.

A Figura 7 mostra um exemplo da projeção manual. Neste exemplo, selecionamos um tipo, `solution-2`, e o sistema automaticamente exibiu o texto correspondente.

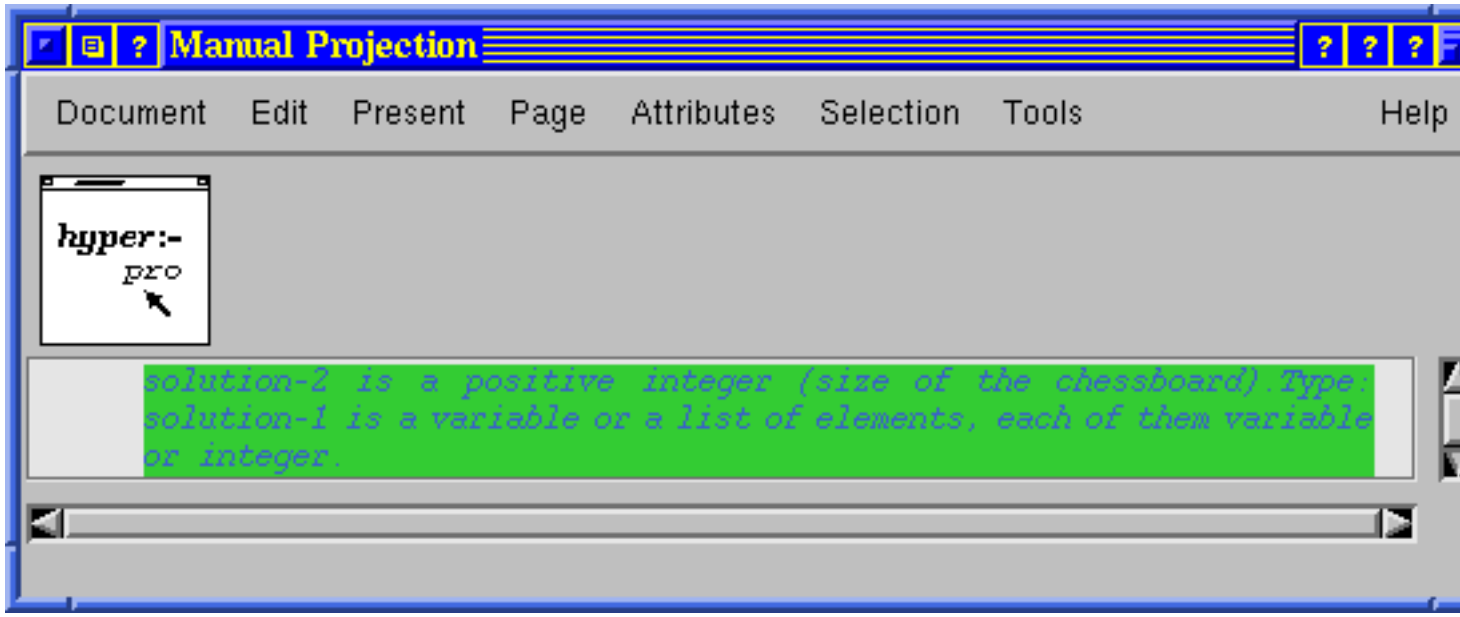

Figura 7: Visão da Projeção Manual

A Figura 8 mostra como a projeção manual foi implementada. Primeiramente, o elemento selecionado é reconhecido e armazenado com a função `TtaGiveFirstselectedElement`. Depois que esta função é

executada, são feitas algumas verificações para se saber se tem ou não elemento selecionado. Então com elemento selecionado, abre-se uma visão com este elemento somente através da função `TtaOpenSubView`.

```
TtaGiveFirstSelectedElement;
TtaOpenSubView(selectedElement);
VerificaVista;
```

Figura 8: Projeção Manual

4.2.4 Implementação da Projeção Recursiva

A implementação da projeção recursiva também foi feita no arquivo EDITORactions.c gerado ao compilar o arquivo EDITOR.a no compilador app . Ela foi escrita na liguagem C utilizando funções da API do tipo: pegar o elemento selecionado, pegar o elemento referenciado, etc.

A Figura 9 mostra em alto nível o algoritmo utilizado.

```
get selected element
if selected element != relation_def
   error;
   exit;
get cpr;
get cpr referenced element;
search reference element in subtree (element);
while referece element
{
 while reference in subtree
 {
   get referenced relation;
   get cpr;
   get cpr referenced element;
   search (reference element ++) in subtree (element);
 }
 element = (referenced element --);
}
```

Figura 9: Projeção Recursiva

Inicialmente o usuário seleciona um elemento. Se o elemento não for uma relação de definição, então o programa termina e fornece uma mensagem de erro. Caso contrário, ele caminha na árvore de estrutura do documento até encontrar a referência do predicado atual (CPR). Ao encontrar a referência, ele procura o elemento referenciado que é uma definição de predicado. A partir dai, ele caminha na árvore de estrutura e ao encontrar o pacote de cláusulas ele as imprime.

Nesse ponto, o programa tem apenas o pacote de cláusulas do predicado corrente. Na definição de projeção recursiva apresentada na Seção 4.2.1 note que além do CPR, é preciso que as referências de predicado também apareçam. Então, é necessário procurar na subárvore do predicado atual se existe

alguma referência. Caso exista, esta referência é guardada em uma lista. Depois percorre-se esta lista e procura-se novos pacotes de cláusulas até que essa corrente termine. No final desse *loop*, a lista conterá elementos que são pacotes de cláusulas.

Aposó coletados todos os pacotes de cláusulas relacionados àquela relação de definição, eles são mostrados em uma janela. Mas como a API no grupo `Vista` (veja Seção 4.1) possui uma função que abre uma janela com apenas uma subárvore, ou seja, um pacote de cláusulas, foi preciso, para exibir todos os pacotes, criar uma nova biblioteca com as funções da API modificadas para que elas se adequassem à projeção. Essa biblioteca está sendo construída.

A Figura 10 mostra um exemplo da projeção recursiva.

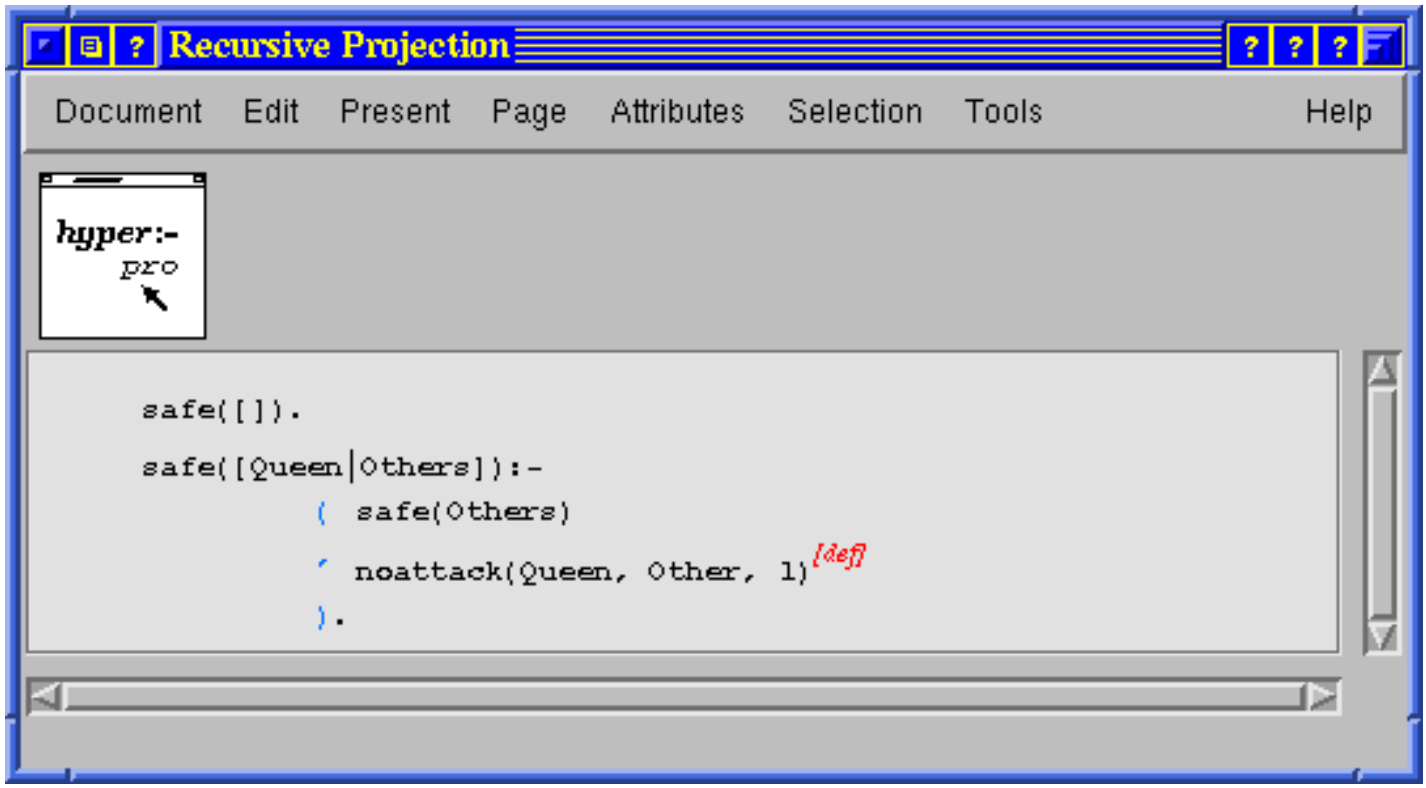

Figura 10: Visão da Projeção Recursiva

5 Conclusão

5.1 Trabalho Realizado

Os resultados obtidos durante esta fase do projeto foram:

- estudo das linguagens T [6, 10], A [12] e API [11] do `Thot` [7];
- implementação das projeções manual e recursiva;
- instalação do `HyperPro Básico`;
- re-instalação do `HyperPro` [1, 2, 3, 4, 5].

5.2 Problemas Detectados na Execução do Projeto

Durante algumas fases deste projeto, surgiram algumas dificuldades. Uma delas foi a falta de uma documentação mais elaborada. A maior parte da literatura sobre programação literária , e os sistemas disponíveis foram encontradas em manuais de usuário e artigos. Estudar o manual do sistema `Thot` para que pudéssemos entender o sistema e conseguir superar essa deficiência não foi uma tarefa fácil.

Outro problema encontrado foi durante a re-instalação do protótipo. Devido a um problema operacional, perdemos algumas bibliotecas do sistema. Recuperá-las, sem as informações adequadas foi muito trabalhoso. Relacionado também com a reinstalação do HyperPro, nos deparamos com outro problema: encontrar a arquitetura certa para instalar o protótipo. Mas apesar das dificuldades, o protótipo foi instalado com sucesso, depois de certo tempo de pesquisa.

Com o `HyperPro` instalado, tivemos que compilar os novos esquemas de estrutura e apresentação. Depois de estarem compilados, foi preciso construir novamente a aplicação usando o compilador app. Ele compila os esquemas de aplicação gerando os arquivos: EDITOR.SCH, EDITOR.h, EDITORAPP.c, EDITORactions.proto, EDITORactions.h e EDITORdialogue.

O arquivo EDITORactions.proto contem os protótipos das funcionalidades que foram definidas para o `HyperPro Básico` e é nesse arquivo que está implementada a projeção recursiva.

Além dos problemas mencionados acima, tivemos problemas com a interface gráfica do `Thot` porque as funções existentes na API do `Thot` não eram adequadas para mostrar a projeção recursiva. Para solucionar este problema foi feito então um estudo do código que gera a interface do `HyperPro Básico` e atualmente está em construção uma biblioteca com essas funções alteradas de forma a se adaptarem às projeções.

5.3 Trabalhos Futuros

Faz parte da próxima etapa do projeto de iniciação científica, a implementação das projeções automática e baseada em índices além da implementação da tradução do documento `HyperPro` para HTML. Também serão feitos dois manuais: de sistema e do usuário.

Referências

[1] Deransart, P., Bigonha, Roberto S, Ed-Dbali, AbdelAli, Siqueira, Jose, Bigonha, Mariza A S, "Basic HyperPro Functionalities and Utilities", Relatório Técnico n^o 23, Departamento de Ciência da Computação, ICEx, UFMG, 1997, 17 páginas.

[2] Deransart, P., Bigonha R.S., Parot, P., Bigonha, M.A.S., Siqueira, J., *A Hypertext Based Environment to Write Literate Logic Programms*, Relatório Técnico n^o 15, Departamento de Ciência da Computação, ICEx, UFMG, 1996.

[3] Parot, Patrick, *The HyperPro Experimental System*, Relatório Técnico n^o 16/96, Departamento de Ciência da Computação, ICEx, UFMG, 1996.

[4] Deransart, P., Bigonha Roberto S., Parot, P., Bigonha, Mariza A.S., Siqueira, J., *A Literate Logic Programming System*, *Anais do I Simpósio Brasileiro de Linguagens de Programação da* SBC, Belo Horizonte, 1996, páginas:1-16.

[5] Deransart, P., Bigonha Roberto S., Parot, P., Bigonha, Mariza A.S., Siqueira, J., *A Hypertext Based Environment to Write Literate Logic Programms*, *Anais do* JICSLP'96, Bonn, Germany, setembro/1996, páginas:247-252.

[6] Ribeiro, Fávia P., *Um Ambiente para Desenvolver Programação em Lógica Baseada no Paradigma de Estilo Literário*, relatório final de Iniciação científica, 1996-1997.

[7] Quint, V., *The Thot user manual*, 1995

[8] Quint, V. and Vatton, I., *Hypertext aspects of the Grif structured editor: design and applications*, INRIA Rocquencourt, 1992, July.

[9] Quint, V., *Édition de documents structurés*, Le traitement électronique du document, 1994, 11–17, ADBS éditions, October.

[10] Quint, V., *The Languages of Thot*, Internal report, INRIA-CNRS, 1992, May.

[11] Quint, V. and Vatton, I., *The Thot Tool Kit API*, July 10, 1997.

[12] Quint, V. and Vatton, I., *The Thot Application Generation Language*, May 7, 1997.

[13] Deransart, Pierre and Ed-Dbali, Abdelali and Cervoni, Laurent, Springer Verlag, *Prolog, The Standard; Reference Manual*, 1996.

[14] Knuth, Donald D., *Literate Programming*, The Computer Journal, Vol. 27, No. 2, 1984, pp. 97-111.

[15] Knuth, Donald, *The* `web` *System of Structured Documentation*, Technical Report 980, Stanford Computer Science, Stanford, California, September 1983.

[16] Ramsey Norman, *Literate-Programming Tools Can be Simple and Extensible*, Departament of Computer Science, Princeton University, November 1993.

[17] Ramsey Norman, *Literate Programming Simplified*, IEEE Software, V.11(5), 97-105, September 1994.

[18] Ramsey Norman, *The noweb Hacker's Guide*, Departament of Computer Science, Princeton University, September 1992 (Revised August 1994).

[19] Ramsey Norman, *Literate Programming: Weaving a language-independent Web*, Communications of the ACM, 32(9): 1051-1055, September 1989.

[20] Leler, William, *Constraint Progamming Languages: their specification and generation*

[21] Venetianer, Tomas, *HTML - Desmistificando a Linguagem da Internet*, Editora mcGraw-Hill Ltda, São Paulo.

[22] Lamport, Leslie, *LATEX: A Document Preparation System*, Editora Addison-Wesley Pub Co

A Apêndice

A seguir, será mostrada a figura da árvore de estrutura de elementos do `HyperPro`.

s

Belo Horizonte, 31 de agosto de 1998.

Flávia Peligrinelli Ribeiro

Mariza Andrade da Silva Bigonha