POLANCZYK

Uma Ferramenta baseada em Hipertexto para o Desenvolvimento de Software

Este exemplar corresponde a redação final da tese devidamente corrigida e defendida pelo Sr. Carlos Alexandre Polanczyk e aprovada pela comissão julgadora.

Campinas, 22 de janeiro de 1991.

Prof. Dr. Rogério D. B. P. de Mello Filho
Orientador

Dissertação apresentada no Instituto de Matemática, Estatística e Ciências da Computação, UNICAMP, como requisito parcial para a obtenção do Título de Mestre em Ciências da Computação.

Agradecimentos

A todas as pessoas que, direta ou indiretamente, colaboraram para a concretização deste trabalho apresento os meus mais sinceros agradecimentos. Em especial, ao professor Dr. Rogério Drummond que me proporcionou uma orientação segura e completa.

Aos professores da banca examinadora Dr. Marcos Roberto da Silva Borges e Dr. Ricardo de Oliveira Anido e aos colegas Hernan Piñón, Carlos Alberto Furuti e Cassius Di Cianni pelas valorosas colaborações, sugestões e críticas. Ao CNPq[1], Unicamp[2] e Fapesp[3], que através de bolsa de estudo viabilizaram a realização desta dissertação.

[1] Conselho Nacional de Desenvolvimento Científico e Tecnológico
[2] Universidade Estadual de Campinas
[3] Fundação de Amparo a Pesquisa

Para minha namorada
Betina.

Sumário

Programas grandes e complexos devem ser organizados como uma hierarquia de módulos. Desta forma a manipulação destes módulos fica facilitada, pois cada módulo tem a função de abstrair os detalhes de sua implementação, e dos outros módulos dos quais depende. Macroscopicamente, um programa é um grafo orientado com raiz onde cada vértice representa um módulo e cada aresta uma dependência entre dois módulos.

A documentação terá uma estrutura análoga e isomorfa ao do grafo do programa. O mesmo ocorrendo com a sua especificação. Portanto, torna-se necessária ferramentas que manipulem estes grafos. É fácil verificar que estes grafos podem ser vistos como hipertextos, ou seja, um texto hierarquicamente estruturado.

Este trabalho apresenta uma ferramenta para manipular programas como objetos hierárquicos. Ela é baseada em hipertexto e sua funcionabilidade é geral o suficiente a outras aplicações que não seja programação. Para alcançar tal generalidade, todo e qualquer arquivo ASCII é um hipertexto.

O sistema desenvolvido é independente de terminal e facilmente portável para outros sistemas compatíveis com Unix[4]. Já está adaptado para Digix, Clix e SunOS. Estes sistemas são derivados de versões bastante diferentes do Unix: System III, System V 3.2 e BSD 4.2, respectivamente.

[4] marca registrada da AT&T

Abstract

Large and complex programs should be organized as a module hierarchy. In this way, the handling of modules becomes easy, because each module has the function of hiding its detail of implementation and from others modules it depends upon. Macroscopicly, a program is a rooted oriented graph where each vertice represents a module and each edge represents a dependence between two modules.

The documentation will have as analogous and isomorphic structure. The same happens to the specification. So, it becomes necessary a tool to handle this graph. It is easy to check that this graph can be viewed as hypertexts, it est, a structured hierarchy text.

This work presents a tool for handling programs as a hierarchy objects. It is basead on hypertext and its funcionality is enough general to others aplications than programation. To achieve this generality, even a plain ASCII file is a hypertext.

The system is device independent and easy to port to any other Unix-like systems. At present, it is already working on Digix, Clix and SunOS. These systems were derivated from diferentes versions of Unix: System III, System V3.2 and BSD 4.2.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Esta tese tem por primeiro objetivo apresentar um levantamento sobre hipertexto: seus conceitos, sistemas atuais, vantagens e desvantagens do seu uso. Descreve sua adequação para manipular e organizar grandes volumes de documentos. Apresenta os ambientes e as técnicas de desenvolvimento de programas enfatizando seus pontos fracos no que concerne a organização e integração de programas.

Finalmente, propõe uma ferramenta de trabalho que manipula hipertexto destinada principalmente a otimizar o processo de desenvolvimento de *software*.

1.1 O que é hipertexto?

Hipertexto é a técnica de organizar textos lineares de forma estruturada não linear. A estrutura do hipertexto depende da inter-relação existente entre os textos. Essas inter-relações podem ser, por exemplo, citações ou referências que um texto faz a outros. Cada referência é denominada *ligação*[1] e cada texto propriamente dito é um *nó*[2]. Em outras palavras, um hipertexto pode ser definido como um grafo cujos vértices são textos ligados por arestas que são as ligações.

A estrutura dos textos pode ser criada manualmente com anotações no próprio texto ou por outros mecanismos, como livros de índice e cartões de

[1] do inglês: *link*
[2] do inglês *node*

referência e, atualmente, por computadores. Os computadores em especial têm apresentado um espantoso crescimento nesta área na última década, principalmente pela sua velocidade, capacidade gráfica e capacidade de armazenamento.

1.2 Ambiente de Desenvolvimento de Software

Um ambiente de desenvolvimento de *software* é um conjunto de terminologias, técnicas e ferramentas que otimizam o processo de desenvolvimento de programas. Este processo cobre desde a especificação até re-utilização de *software* passando por todas as fases de produção que podem ser sintetizadas em:

especificação conjunto de termos e regras que são obtidas a partir da discussão do projeto entre o grupo de trabalho e o usuário;

modularização divisão do trabalho em módulos de desenvolvimento que são alocados aos diversos componentes do projeto com o objetivo de abstrair detalhes de especificação e implementação;

estruturação organizar os módulos em níveis hierárquicos, com *interfaces* definidas, de acordo com a relação de dependência entre eles, de modo a criar níveis de abstração.

codificação escrever os módulos, propriamente ditos, em alguma linguagem de programação;

testes testar cada módulo e o programa durante o seu desenvolvimento;

documentação documentar cada módulo do programa - abordagem funcional;

manual de utilização destinado ao usuário final, com instruções sobre como utilizar o sistema;

manual de manutenção como proceder em caso de alterações e atualizações futuras do sistema.

Os ambientes em uso na atualidade não são capazes de manter integradas todas as fases acima descritas. Isto provoca a perda de informações

na especificação, módulos de programas esparsos no sistema de arquivos, dificuldade na integração dos módulos, documentação desatualizada e até inexistente e manuais sem coerência com o programa, sem contar o tempo disperdiçado no desenvolvimento.

1.3 Sistema HiperTexto: ht

O Sistema **ht** é uma ferramenta de trabalho que manipula hipertextos e tem por objetivo principal auxiliar o processo de desenvolvimento de *software*. Sua utilidade está na capacidade de integrar e gerenciar grandes programas em desenvolvimento. Assim, tem-se o projeto organizado e armazenado como um única entidade. Esta unicidade permite solucionar os problemas apresentados na seção 1.2. O sistema **ht** não pretende ser um ambiente mas um utilitário de grande potencialidade. Ele é uma das ferramentas do ambiente A_HAND [Dru 87] descrito no capítulo 3.

O **ht**, tem uma linguagem simples e um reduzido número de comandos, facilitando muito a recuperação, navegação e edição de nós. Possui uma *interface* configurável que permite ao usuário redefinir os comandos de entrada. Ele foi escrito para Unix e é independente de terminal e máquina, que o torna extremamente portável. Atualmente, uma primeira versão já está instalada e em uso no DCC da Unicamp em três arquiteturas distintas.

Para um melhor entendimento de como está estruturado este trabalho, os parágrafos que seguem apresentam uma breve descrição de cada capítulo.

O capítulo 2 apresenta o conceito de hipertexto, abordando todos os tópicos referentes às ligações e aos nós. Segue um pequeno histórico onde surgiram os primeiros hipertextos e como está se desenvolvendo com o apoio da computação. Descreve uma taxionamia dos hipertextos e alguns sistemas que estão sendo desenvolvidos. Ao final apresenta uma comparação, vantagens e desvantagens do uso de hipertexto.

O capítulo 3 apresenta alguns itens de grande importância para o desenvolvimento de *software*, que podem ser gerenciados facilmente na forma de hipertexto. Descreve suscintamente o sistema A_HAND, um ambiente de desenvolvimento de *software*, e mostra como um sistema de hipertexto é

útil para gerenciar algumas fases do desenvolvimento de programas.

O capítulo 4 apresenta o sistema **ht**: conceitos, utilização e exemplos de aplicações para o desenvolvimento de *software*. Mostra uma comparação do **ht** com alguns sistemas apresentados no capítulo 2, propondo no final algumas extensões.

O capítulo 5 descreve as técnicas adotadas na implementação do **ht**: como foi estruturado e desenvolvido. Apresenta suas fases de desenvolvimento, alguns problemas e uma abordagem de como desenvolver rotinas com funções de alto nível.

O capítulo 6 encerra este trabalho com uma visão geral do sistema **ht** e sua utilidade para o desenvolvimento de *software*.

O apêndice A descreve detalhadamente o exemplo apresentado no capítulo 4 que é um programa desenvolvido na forma de hipertexto utilizando o **ht**.

O apêndice B apresenta a tabela dos comandos do **ht**.

Capítulo 2

Hipertexto

O progresso da humanidade sempre esteve inteiramente ligado com a capacidade de representação do conhecimento. A descoberta e uso de símbolos para comunicação foi um marco importante no desenvolvimento do homem. A formalização da linguagem escrita foi possivelmente o maior responsável pela disseminação e desenvolvimento da filosofia e religiosidade na antigüidade.

A descoberta da tipografia possibilitou a produção em larga escala de documentos impressos. Como evento isolado, esta talvez tenha sido a descoberta mais importante para o desenvolvimento intelectual da humanidade. Mas, felizmente, grande parte destes documentos se relacionam com outros por meio de referências ou citações, criando assim uma grande rede que compõe o conhecimento humano escrito.

Com o aumento do volume desta literatura, o processo de pesquisa é dificultado principalmente na localização de documentos específicos, mesmo que existam referências explícitas a eles. Verifica-se então que a estrutura linear a que estamos acostumados carece de um mecanismo que agilize o processo de busca e também da escrita de documentos com assuntos afins. Para exemplificar, imagine então a seguinte situação:

> "Você está lendo um artigo sobre os *escritores brasileiros*, e um deles é *Jorge Amado*. Você se interessa por este autor e procura artigos referentes a ele, localiza-os e começa a lê-los. Suponha que algum destes textos cite uma obra do autor, como o livro *Capitães da Areia*, e você queira saber mais sobre esta obra, necessitando, portanto, de nova consulta para encontrar o

livro. Finalmente, você satisfaz a necessidade de conhecimento sobre *Capitães da Areia* e *Jorge Amado*, e retorna ao texto original: *escritores brasileiros*. Em uma segunda passagem, o texto refere-se a *Érico Veríssimo*, que é seu o maior ídolo. Novamente outra pesquisa, outro artigo e mais tempo perdido. Observe que este processo de ir e voltar nos textos é cansativo e dispendioso, terminando somente quando a necessidade de conhecimento for satisfeita ou quando acabar a paciência. Mesmo que você estivesse em uma grande biblioteca com um sistema de pesquisa *on-line* por computador o trabalho seria mais rápido porém não deixaria de ser um tanto laborioso."

O ideal é criar uma técnica para organizar estes documentos de forma que assuntos afins sejam colocados lado a lado, e que a recuperação de um deles implique na rápida localização dos demais. Hipertexto é exatamente esta técnica que procura suprir esta necessidade de manipular eficientemente um grande volume de documentos.

Com o advento de computadores, maiores possibilidades surgiram para estender a noção de textos lineares a organizações mais complexas e estruturadas.

Este capítulo descreve uma técnica de estruturação de texto chamado *hipertexto*: sua definição, histórico, sistemas atuais, comparações, vantagens e desvantagens do seu uso.

2.1 Introdução

Como já foi dito, hipertexto é a técnica de organizar textos lineares de forma estruturada e não linear. A estrutura é definida pela proximidade de informação dos textos, dada por citações ou referências entre eles. Assim, um texto deixa de ter a forma linear seqüêncial e passa a ser visto como uma organização complexa que permite rápido acesso a informações relacionadas. Assim, chama-se cada texto individualmente de *nó* e as citações ou referências, *ligações*.

Do ponto de vista computacional, segundo [Con 87] hipertexto pode ser definido como:

- um método de banco de dados que permite o acesso direto aos dados, bastante diferente das consultas tradicionais;

- um esquema de representação, um tipo de rede semântica que funde informação textual, operações e processos;
- um tipo de interface orientada a comandos curtos que possibilita rápida navegação pelo hipertexto.

A visualização de um hipertexto em computador é feita através de janelas de informação, ou simplesmente *janelas*. Cada nó está associado a uma janela e as referências estão inclusas no conteúdo do nó. Assim, ao ser criada uma janela, junto com o texto do respectivo nó, também são apresentadas as referências a outros nós de forma destacada. As referências na tela são sensíveis a algum dispositivo apontador[1], e em resposta ao seu acionamento, o sistema localiza o nó referenciado e o apresenta em uma segunda janela conforme a figura 2.1. É possível continuar a pesquisa em novas ligações ou retornar a nós anteriores. Este processo de ir e voltar em nós de um hipertexto é chamado de *navegação*. Estas características definem os chamados Sistemas de Hipertexto, e os textos armazenados e editados nestes ambientes são *hiperdocumentos* ou *hipertextos*.

2.2 Ligação

Segundo [Con 87], a capacidade de ligação entre os nós é que permite organizar os textos de forma não linear, sendo a principal característica dos hipertextos. Os itens que seguem apresentam alguns pontos importantes sobre ligações.

2.2.1 Capacidade de Navegação

Quando se está lendo o texto de um nó, é desejável que o usuário faça o mínimo de esforço para recuperar o novo texto referenciado. Esse mínimo consiste em apontar a referência e ativá-la por um dispositivo qualquer[1], deixando a cargo do sistema a localização e a visualização do novo nó.

Outra característica importante dos sistemas de hipertexto é o tempo de resposta quando uma referência é acionada. Isto porque, muitas vezes, o próprio usuário não sabe qual é o conteúdo do nó referenciado. Se o sistema apresentar respostas lentas, o usuário é desencorajado a visitar o nó, por

[1] teclado, *mouse*, *light pen*.

este ter um conteúdo ou utilidade duvidosos. O problema se agrava se o hipertexto estiver distribuído em uma rede; neste caso o sistema poderia estimar e fornecer a informação de quanto tempo levará a operação antes de realmente executá-la.

2.2.2 Propriedades das Ligações

Ligações podem ser usadas para diversas funções, que podem ser:

- conectar textos e documentos relacionados;
- incluir comentários e anotações a documentos;
- associar informações, como uma tabela de índice a seu material referênciado;
- ligar fragmentos de texto (nós) de forma a tornar um documento linear;

As ligações podem ainda ter nome e tipo. Os nomes permitem, por exemplo, criar *visões*: somente ligações com o mesmo nome podem ser referenciadas, desta forma selecionando assuntos de interesse. O tipo de uma ligação cria relações entre os nós, como por exemplo: especialização, generalização e oposição. Estes dois aspectos orientam o usuário na navegação do hipertexto.

2.2.3 Classes de Ligações

Em hipertexto exitem três classes principais de ligações que o usuário pode seguir.

Referenciais são as ligações mais simples e essencialmente caracterizam hipertextos. Cada ligação referencial une apenas dois nós, e é, geralmente, orientada: um nó representa a origem e outro o destino da ligação. A origem da ligação é representada no texto por uma referência de forma destacada a fim de permitir melhor visualização, conforme a figura 2.2. O nó destino, pode ser um texto ou parte dele. As ligações referenciais e todas aquelas que definem um nó de origem são ditas *explícitas*.

Organizacionais como as referenciais, são ligações explícitas, mas propiciam a criação de uma estrutura ao hipertexto, como por exemplo, a criação de um documento que possui seções e sub-seções, formando uma árvore. A navegação neste tipo de hipertexto dá-se pela sua estrutura.

Palavras-chaves são uma classe implícita de ligações porque não existe nó de origem da referência. Consiste em procurar por nós contendo uma determinada cadeia de caracteres ou de determinadas palavras-chaves em todo hipertexto. A pesquisa por cadeia de caracteres consiste em vasculhar todo o conteúdo dos nós do hipertexto, e por palavras-chaves somente em um conjunto de palavras que foram previamente definidas pelo usuário nos nós. Naturalmente, esta classe de ligação implica em um tempo de resposta maior, principalmente no caso de seqüências de palavras, no entanto constitui um recurso extremamente poderoso.

Na implementação de busca por palavras-chaves surge o problema de abrangência e precisão. Que palavras do texto escolher como palavras-chaves? O artigo [Fri 88] discute largamente o problema.

2.3 Nó

Embora a essência de hipertexto seja a sua capacidade de ligação, os nós contribuem de forma significativa na definição das operações de um sistema de hipertexto. Comumente nós armazenam idéias, como fragmentos de textos, menores que os arquivos tradicionais. Quando um nó é utilizado desta forma, hipertexto introduz um nível intermediário de armazenamento entre caracteres e arquivos. Porém, o tamanho de cada nó é indefinido, estando a cargo do usuário.

2.3.1 Modularização de Idéias

O uso de hipertexto induz o escritor a dividir suas idéias em módulos (nós) permitindo que estes possam ser referenciados posteriormente, fornecendo ao leitor opções de seguir no texto, deixando-o livre da estrutura linear dos documentos tradicionais. Por exemplo, na leitura de um nó sobre um país, é possível que o texto faça referências à sua história, geografia, política ou esporte. Um documento bem estruturado com hipertexto facilita o leitor na escolha entre as diferentes formas de ver o texto. Contudo, o escritor

deve ter em mente que um nó contém uma idéia completa sendo, de certa forma, isolado dos demais nós, mas deve existir uma certa simetria com seus vizinhos[2]. Alguns sistemas de hipertexto permitem a visualização de um conjunto de nós em um único texto como se este fosse linear.

O processo de identificar cada conceito ou idéia em um nó não força o leitor a seguir uma seqüência previamente definida. Esta possibilidade de desvio induz o leitor a sair do caminho em que ele estava inicialmente interessado. Alguns sistemas de hipertexto introduziram o conceito de *path*, que é uma seqüência ordenada de nós onde cada um possui uma única referência com o atributo *path* (referência *default*). Assim, o leitor seguindo as referências com o atributo *path* tem a sensação de estar lendo um texto linear. A escolha da referência que fará parte do *path* é feita pelo escritor, para facilitar a leitura de seu documento.

2.3.2 Visão a Nível de Objetos

Um nó que contém parte de um texto, como um parágrafo, pode ser visto como um objeto, passível de ser movido, apagado ou alterado sem afetar outros objetos do texto. Hierarquicamente, num primeiro nível, pode estar o próprio documento composto por seções, que faz apenas referências. Num nível abaixo, temos as seções que fazem referências aos parágrafos. Esta visão hierárquica de um documento permite modificar qualquer parágrafo ou seção sem afetar os outros objetos (parágrafos e seções) do objeto maior que é o documento. Hipertexto facilita e simplifica em muito aplicações deste modo de pensar e trabalhar.

2.3.3 Nó com Tipo associado

Pode ser extremamente útil a associação de tipos aos nós, principalmente se estes tiverem alguma estrutura interna especial ou exigirem algum processamento particular durante a sua apresentação. Essas características são especificadas explicitamente pelo tipo. Por exemplo, um nó do tipo documento, que durante a sua visualização o texto é formatado de acordo com o tamanho da janela.

Os sistemas de hipertexto utilizam geralmente uma cor, tamanho ou ícone diferente para cada tipo de nó. Esta distinção ajuda o usuário a

[2] nós referenciados por ele e que o referencia

navegar no hipertexto.

2.3.4 Nó semi-estruturado

Nó semi-estruturado é um tipo de nó com estrutura interna definida, como por exemplo, na implementação de um dicionário, onde cada nó é formado por um campo de rótulo, a palavra, e outro é sua definição. Desta forma o sistema de hipertexto orienta o escritor na entrada dos dados e fornece ao leitor mecanismos de busca pelo campo de rótulo de modo mais eficiente.

2.3.5 Nó Composto

Outro mecanismo para agrupar informações relacionadas em hipertexto são os nós compostos. Vários nós de um hipertexto podem ser agrupados e o conjunto é tratado como se fosse único, com nome e tipo próprio. Por exemplo, diferentes entidades de uma tabela podem ser distinguidos como itens isolados, mas num nível acima, fazem parte do mesmo objeto: tabela. A decisão do tipo de visualização deste nó pode, por exemplo, depender do tipo da ligação.

2.4 Formalismo

A maioria dos sistemas define hipertexto como um grafo orientado rotulado. Assim, um hipertexto é uma tripla do tipo:

- <N,L,E>

onde N é um conjunto de nós que armazenam determinada informação; L é o conjunto de rótulos; e E:NxL—N é o conjunto de arestas rotuladas orientadas de um nó para outro.

A navegação neste modelo requer a noção de nó corrente, atual estado do usuário, e o conjunto de possíveis nós destinos. Define-se então *estado do usuário* como uma tripla <m,l,e> e um nó $m \in N$, que serve como uma marca para representar o nó corrente do usuário.

A figura 2.3 apresenta um exemplo deste modelo, onde elementos de N são representados por círculos, elementos de E são setas e os de L são

mostrados como palavras. O símbolo $\oplus$ indica o valor corrente de M, que representa o estado do usuário. Quando a marca está em um nó,

- o conteúdo do nó é apresentado em uma janela;
- rótulos associados a aresta saindo do nó podem ser apresentados, embutidas no texto ou em menus;
- quando um rótulo é selecionado, a marca M é movida para o nó destino do vetor correspondente.

Como E é uma função, este modelo é equivalente a um autômato finito determinista. Entretanto, este modelo não permite representar um nó com a noção de página e também não possui uma forte associação entre os rótulos e as arestas. Por exemplo, pode-se definir acidentalmente um rótulo chamado *abortar* a uma aresta que deveria ser *cont.* como na figura 2.3. Por outro lado, este formalismo não abrange um histórico dos valores de M nem características semânticas necessárias a um sistema realmente utilizável. Portanto, propriedades adicionais devem ser especificadas e seguidas pelo implementador. O artigo [Tom 89] apresenta uma solução para estes problemas.

2.5 Histórico

A história de hipertexto é rica e variada porque não é uma idéia recente e que exige o uso de computadores. Muitas pessoas contribuíram, e cada uma delas parece ter uma visão e uma aplicação diferente para hipertexto.

Os primeiros textos que apresentavam indícios de referências foram encontrados no *Talmud*[3] e nas anotações de Aristóteles que datam de V a.C.. Estas referências são simplesmente passagens de onde encontrar mais informações relacionadas.

Uma aplicação usual de hipertextos manuais são os dicionários e enciclopédias. Quando se lê uma definição ou uma teoria em um artigo (enciclopédia), normalmente se segue a indicação de onde encontrar mais informações afins. Já um dicionário é um hipertexto auto-contido que qualquer palavra é uma referência que pode ser encontrada facilmente na ordem alfabética.

[3] textos religiosos da literatura judaica

Evidentemente, que hipertextos manuais não permitem uma navegação intensa no textos, provocando a introdução de computadores nesta área. Por conseguinte, a história já é bem mais recente e diversificada.

2.6 Hipertextos Automatizados

A concepção original de hipertexto foi a de desenvolver uma ferramenta capaz de armazenar e recuperar grande volume de dados relacionados eficientemente do ponto de vista de facilidade de pesquisa para o usuário. Muitos sistemas foram e estão sendo desenvolvidos nesta área, mas não há uma classificação universalmente aceita das aplicações que envolvem os diversos tipos de hipertextos.

Em [Con 87] Conklin propôs uma divisão, apresentada a seguir, segundo a área de aplicação dos hipertextos.

2.6.1 Sistemas Macro Literários

São sistemas de hipertexto voltados ao suporte de grandes massas de dados com manipulação dos textos *on-line*. As referências entre os documentos são criadas pelo usuário durante a inclusão de novos documentos ao hipertexto. Alguns exemplos deste sistema são:

Memex o objetivo de Vannevar Bush [Fid 88] quando propôs, em 1945, o seu artigo *"As We May Think"* foi criar um mecanismo para otimizar o armazemamento e recuperação de documentos da literatura científica. O *Memex* como foi chamado, continha uma extensa biblioteca de artigos, anotações, fotografias e diagramas. Permitia múltiplas janelas e a facilidade de estabelecer ligações entre os documentos.

Como na época não existiam computadores digitais, o *Memex* foi implementado por células fotoelétricas e microfilmes. Com isto, Bush recebeu o mérito de prever a explosão da informação.

NLS/Augment na década de 60, Douglas Engelbart, do Instituto de Pesquisa de Stanford, influenciado pelas idéias de Bush, propôs um sistema chamado H-LAM/T (*Human using Language, Artifacts and Methodology, in which he is Trained*) que tem como principal elemento a interação. Segundo Elgelbart, entre o usuário e o computador deve

existir uma troca dinâmica de informações que tem como conseqüência o aumento do conhecimento do usuário.

Cinco anos depois, em 1968, as idéias de Engelbart sobre "aumento de conhecimento" se tornaram mais específicas e foi criado o sistema NLS (*oN Line System*). O NLS permite multiplas janelas, visões e ligações dinâmicas entre os nós e também teleconferência[4].

Com a evolução, hoje o NLS é conhecido como *Augment* e é comercializado pela McDonnell Douglas. Além das características originais, o sistema permite varias formas de comunicação, facilidade de controle, documentação, gerência e organização de produtos e engenharia de *software*.

Xanadu durante o desenvolvimento do projeto *Augment*, outro "visionário de hipertexto", Ted Nelson, desenvolveu vários conceitos sobre como criar um ambiente literário unificado de grande escala. O projeto Xanadu possuía abordagens extravagantes para época e permitia um expressivo número de ligações. Apesar de permitir o armazenamento de diferentes versões de um mesmo documento, o Xanadu armazena apenas a primeira versão e as modificações que produzem as versões seguintes, reduzindo o espaço de armazenamento necessário. O grande objetivo do projeto Xanadu era de prover uma facilidade revolucionária de armazenar toda literatura então existente. Suas principais características são a separação da *interface* com o usuário do banco de dados, proteção, contabilização de consultas e distribuição de *royalties*. Nelson previu o surgimento de bibliotecas *on-line* e criou todo um mecanismo para organizar e indexar este grande número de informações. Foi ele quem realmente introduziu o termo *hipertexto*.

Texnet Randall Triggs escreveu a primeira tese de PhD sobre hipertexto, onde descreve o sistema Texnet que suporta textos não-lineares. Estes textos, que são partes de documentos, são chamados de *fragmentos primitivos* e estão inter-conectados por ligações formando uma rede. O principal enfoque da tese é a criação de ligações com tipos associados para suportar críticas a documentos.

O Texnet implementa dois tipos de nós: os *chuncks* que permitem uma organização hierárquica e os *tocs* que viabilizam uma estrutura não hierárquica. Triggs propôs uma taxionomia específica para o tipo

[4] técnica de comunicação via computador entre um grupo de pessoas simultaneamente

de ligações para anexação de críticas e sugestões ao hipertexto. Ele argumenta que para um certo conjunto de comentários existe um tipo de ligação, por exemplo, comentários a favor ou contra um documento. Por fim, o Texnet introduziu o conceito de *path* a hipertexto, que é uma lista ordenada de nós por referências para visualização linear do documento.

2.6.2 Sistemas de Investigação

São sistemas de hipertexto altamente interativos com baixo tempo de resposta para um pequeno conjunto de comandos. Uma característica importante é a facilidade de suprimir detalhes em vários níveis de especificação para o usuário, como por exemplo navegar em um documento somente pelas seções, omitindo as sub-seções e parágrafos. Outra característica importante é a facilidade de estruturar um grande número de informações disjuntas em um único e homogêneo hipertexto. Estas características são adequadas para especificação e programação de sistemas.

IBIS o método IBIS (*Issue-Based Information Systems*) [Beg 88] [Con 88], desenvolvido por Horst Rittel, baseia-se no princípio de que o processo de solução de problemas complexos[5] está fundamentado na interação das pessoas envolvidas nele. Segundo Rittel, problemas complexos são aqueles que não podem ser resolvidos pela abordagem tradicional: definir o problema, coletar dados, analisar os dados e construir a solução. No método IBIS, quando surge um problema, este é apresentado na forma de premissa – *issue*. Com base neste método, foi desenvolvido o sistema gIBIS (*graphical* IBIS) que implementa o processo de discussão em hipertexto. O gIBIS define três tipos de nós (*Issue, Positions* e *Arguments*) e nove tipos de ligações (*responds-to, questions, supports, objects-to, specializes, generalizes, refers-to, replaces* e *other*) (vide figura 2.4).

SYNVIEW um sistema conceitualmente similar ao IBIS, mas Davis Lowe incluiu uma característica importante. Ele propôs que os participantes da discussão, além de anexarem sua posição em relação ao assunto, apresentassem uma nota de validação e importância para os demais nós. Desta forma, se permite uma maior visualização e orientação de como está evoluindo o problema abordado.

[5] wicked problems

WE um ambiente de edição de textos desenvolvido por um grupo de pesquisa da *University of North Caroline*. Suas pesquisas baseam-se no modelo cognitivo do processo de comunicação, que explica a leitura como tomar um texto linear, compreendê-lo estruturando seus conceitos hierarquicamente e absorvendo-o como uma rede de informações conexas. A escrita é o processo inverso: o autor cria a partir de idéais inter-relacionadas uma estrutura hierárquica que então é linearizada para produzir o texto final.

O WE utiliza um banco de dados relacional para armazenar os nós e as ligações. Basicamente utiliza cinco janelas: três para edição de texto, uma para consulta ao banco de dados e a última de controle. O WE é um sistema experimental e a aplicação do modelo cognitivo do processo de comunicação ainda não está totalmente validado.

Processadores de Esboço[6] semelhantes a processadores de texto porém tem a característica de movimentar, criar e alterar esboços, que são fragmentos de textos. Normalmente têm um editor de texto e algum tipo de formatador, assim o usuário pode, a partir de um primeiro esboço, produzir o texto final. Uma de suas principais características, como no NLS, é a abstração de níveis de detalhe do esboço.

A maioria deste processadores são projetados para microcomputadores: o primeiro surgiu em 1984, o ThinkTank, e a partir daí, apareceram muitos outros: MaxThink, Executive Writer/Executive Filer, Thor, FrameWork, Kamas, Fact Cruncher, Freestyle, More, Ideia e PC-Outline. Recentemente dois novos produtos surgiram, o Houdini, uma extensão do MaxThink, que permite referencias não-hierárquicas entre nós e o ForComment que permite até 15 usuários manipularem uma estrutura semelhante a um hipertexto numa rede local (LAN - *Local Area Network*). Importante caracterizar que estes produtos não são sistemas reais de hipertexto.

2.6.3 Sistemas de Informação Estruturada – "Folheadores"

São hipertextos de fácil uso e rápido acesso a informação, destinados a aplicações que envolvem grandes quantidades de dados como por exemplo ao ensino, informações públicas e referências. A inclusão de dados é feito

[6] Outline Processors

uma única vez no momento da criação do hipertexto. Portanto, a edição de novos textos (nós) normalmente não é permitida ou suportada.

ZOG/KMS ZOG é uma sistema baseado em menu desenvolvido em 1972 na *Carnegie-Mellow University* [Aks 88], que consiste de um banco de dados de pequenos segmentos (janelas) que são visualizados individualmente. O ZOG foi desenvolvido com o objetivo de servir uma grande comunidade de usuários simultaneamente, e, conseqüentemente, poder operar em qualquer terminal de um sistema *time sharing*.

Em 1981, dois dos principais projetistas, Donald McCracken e Robert Akscyn, fundaram a companhia *Knowledge Systems* e desenvolveram a versão comercial do ZOG, chamado *Knowledge Management System* (KMS).

Emacs INFO System é o sistema de *help* do editor EMACS, bastante semelhante ao ZOG. Possui estrutura hierárquica e permite ao usuário navegar a qualquer parte do hipertexto sem seguir sua estrutura. O INFO apresenta apenas uma janela e não possui *browser*[7] estrutural.

Hyperties este projeto desenvolvido na *University of Maryland* e seguiu duas direções: uma ferramenta prática e de fácil aprendizado para vasculhar grandes bancos de dados e uma plataforma experimental para o estudo de *interfaces* para hipertexto.

A unidade básica do Hyperties são artigos curtos (tipicamente de 50 a 1000 palavras), que estão inter-conectados por um número arbitrário de ligações. As ligações são palavras ou frases apresentadas de forma destacada da tela. O usuário ativa uma ligação tocando-a com o dedo (em uma tela sensível) ou pelo teclado. Ativando uma ligação, o texto referenciado é localizado e apresentado em uma segunda janela. O sistema mantém o *path* permitindo ao usuário escolher e percorrer caminhos alternativos.

Hyperties foi projetado para microcomputadores IBM-PC e atualmente permite gráficos, havendo um esforço para inclusão de um *browser* estrutural gráfico. Atualmente já existe uma versão para Unix.

SuperBook é um sistema de hipertexto destinado a aumentar a usabilidade dos documentos tradicionais. O *SuperBook* [Ega 89] tem como entrada

[7] ferramenta que permite visualizar o hipertexto na forma de um grafo

textos comuns e os converte automaticamente em uma estrutura de hipertexto, provendo assim, facilidades de busca, navegação, inclusão de comentários e visualização. Basicamente existem dois grandes objetivos que o diferencia de um sistema e hipertexto: utiliza como entrada documentos comuns (lineares) e provê o processamento para hipertexto com um mínimo de interferência humana.

O *SuperBook* foi implementado numa estação Sun-3 com um gerenciador de janelas próprio chamado MGR *window manager*.

Verificador de Documentos Simbólicos são mais avançados que os sistemas de *help on-line*, permitindo apresentar páginas de um manual de vários volumes em tempo reduzido. Alguns campos no texto são sensíveis e, quando tocados, as seções selecionadas são apresentas na tela. O sistema também permite pesquisa por palavra-chave ou frases ao banco de dados; entretanto não permite a inclusão de novos dados.

2.6.4 Sistemas de Propósito Geral

As seções anteriores apresentaram sistemas de hipertexto com aplicações específicas. Os sistemas que aqui seguem tem normalmente várias aplicações, com o propósito principal de experimentar hipertexto como uma nova tecnologia.

NoteCard é a versão mais conhecida e completa de um sistema de hipertexto, que foi desenvolvida pela Xerox PARC. A motivação original em construir o NoteCard foi desenvolver uma ferramenta de suporte à análise de informações. Os projetistas verificaram que a análise de informação normalmente segue um procedimento padrão que consiste de uma série de passos: ler documentos (artigos, relatórios, livros, etc.), agrupá-los, arquivá-los e produzir um relatório analítico. Também foi observado que durante este processo, os analistas mentalmente formavam uma análise e um modelo conceitual. Procurou-se desenvolver uma tecnologia que ajudasse o analista a formar um melhor modelo conceitual e ajudá-lo a encontrar a melhor maneira de expressar este modelo e a análise.

Uma *interface* programável torna o NoteCard uma arquitetura aberta permitindo aos usuários criarem novas aplicações sobre ele. O NoteCard permite a criação de novos tipos de nós, atualmente, existem mais de 50 tipos, incluindo texto, video, animação, gráficos e ações.

Parte do sucesso do NoteCard é devido ao fato de que foi desenvolvido em uma máquina Lisp[8], uma estação poderosa com grandes recursos gráficos. Atualmente existe entre 50 a 100 usuários do NoteCard, sendo a maioria da própria Xerox. Muitos deles já construiram grandes bancos de dados (da ordem de 1600 nós com 3500 ligações).

Intermedia um dos mais antigos e numerosos grupos de pesquisa sobre hipertexto é o da *Brown University*. O Intermedia [Yan 88] é um projeto idealizado na década de 60 e já está em sua terceira geração. Sua linha de pesquisa está voltada ao ensino superior. O professor armazena, além do curso normal, todas as aulas e tarefas na forma de hipertexto. O aluno ao entrar no sistema obtém um guia que o orienta nas lições, selecionando itens e aplicando testes. Desta forma, o professor pode verificar a evolução do aluno no curso. O Intermedia é composto de um editor de texto, um editor gráfico, um digitalizador, um manipulador de figuras tri-dimensionais e um editor temporal que organiza os nós no tempo.

Neptune é um sistema de hipertexto que como particularidade de projeto tem uma arquitetura aberta. Neptune separa claramente a *interface* baseada em *Smalltalk* do manipulador do banco de dados chamado *Hypertext Abstract Machine* (HAM). O HAM é um modelo de hipertexto genérico que fornece facilidades de criação, alteração e acesso aos nós e ligações. Ele mantém um histórico completo da versão de cada nó do hipertexto permitindo um rápido acesso a qualquer versão de um documento. Provê ainda o acesso distribuído em uma rede, sincronização de múltiplos acessos, controle de versões sobre a rede e recuperação em caso de falha.

A nível de interface, possui vários *browsers*: um *graph browser* que fornece uma visão pictórica dos nós e ligações de um subgrafo do hipertexto, um *document browser* que suporta a visualização da estrutura hierárquica de um documento, um *node browser* que acessa um nó induvidual do hipertexto. Outros são *attribute browser*, *version browser*, *node differences browser* e *daemon browser*.

Boxer é uma linguagem de programação interativa destinada a usuários leigos em computação. A unidade de representação de informação é o *box*. Um *box* pode conter outros *boxes* ou dados, como por exemplo

[8]Xerox D-series Lisp machines

texto e gráfico. Como o Boxer é uma linguagem de programação, ele trata as ligações entre *boxes* como uma porta, formando naturalmente um representação hierárquica do documento.

CREF o *Cross-Referenced Editing Facility* é um protótipo com um editor especializado de texto e gráfico que foi originalmente desenvolvido como uma ferramenta de análise de transcrições de experimentos psicológicos[9], mas seu uso foi extendido a hipertextos. Os nós são chamados de segmentos que estão ordenados linearmente, contendo palavras-chaves e ligações a outros segmentos. A noção de um conjunto linear de segmentos é natural para o problema de análise do protocolo. O CREF suporta quatro tipos de ligação: referencial, sumária, versão[10] e procedência.

Guide é o sistema de hipertexto para o *Macintosh*[11], baseado numa versão Unix, desenvolvido na Inglaterra. O Guide fornece capacidade gráfica e textual para nós e tem três tipo de ligações que podem ser usadas pelo usuário para criar e navegar no hipertexto:

substituição o texto da janela corrente é substituído pelo da referência;

anotação o texto referenciado é apresentado em uma segunda janela, para anotações, que posteriormente desaparece;

referencial o texto referenciado é apresentado em uma nova janela.

PlaneText desenvolvida no MCC *Software Technology Program*, é baseado no sistema de arquivo Unix e utiliza o gerenciador de janelas *SunView* da *Sun MicroSystems*. As ligações são apresentadas na tela entre chaves[12], que podem ser desligadas, e são basicamente ponteiros para os arquivos que referenciam. Isto permite uma fácil integração com o sistema Unix e suas ferramentas. O PlaneText suporta nós gráficos que podem ser livremente ligados ao hipertexto.

H é um sistema de hipertexto que está sendo desenvolvido pelo grupo de Linguagens de Engenharia de *Software* da Universidade Federal de Pernambuco (UFPE) que tem por objetivo manipular hiperdocumentos,

[9] protocolos
[10] supersede
[11] marca registrada da Apple
[12] entre os caracteres { e }

que são exclusivamente textos e figuras [Mei 89]. O H está sendo implementado em uma linguagem de programação orientada a objetos, Smalltalk/V, e trabalha em um ambiente multi-usuário com proteção semelhante ao Unix, embora tenha sido inicialmente desenvolvido num IBM PC-AT.

Basicamente, ao entrar no sistema, o usuário deve se identificar na janela `H-Usuário`. A partir daí, é ativada a janela `H-Documentos` de onde podem ser realizadas operações sobre documentos que estão estruturados hierarquicamente. O sistema permite ainda a inclusão de rodapés, comentários, controle de versões, visões, correio eletrônico e pesquisa por seqüência de palavras. Também está previsto a inclusão de um *browser* estrutural gráfico de modo que o usuário tenha uma visão global do hiperdocumento.

SPRINT o nome é uma acrossemia para *Strategic Plan and Resource INTegration*. O SPRINT é um sistema baseado em hipertexto, para suportar a representação de um modelo mental como uma rede de associações de elementos. Segundo [Car 90] qualquer sistema de hipertexto pode suportar esta rede de associações, mas o sistema desenvolvido incorpora duas significantes extensões: regras heurísticas e comunicação entre os modelos individuais.

HyperCard é atualmente o sistema mais conhecido de hipertexto para os equipamentos *Macintosh*. Suporta texto e figuras, porém, uma vez definido o tamanho da janela de visualização, não pode ser mais alterado. As referências são *botões*, possivelmente visíveis, que se encontram esparsos no nó. Isto permite que um botão esteja sobre uma palavra, que quando pressionada ativa a referência do botão invisível. Para se criar um hipertexto existe uma linguagem de especificação, chamada *HyperTalk*, que possui uma precária documentação.

αTrellis é um sistema de hipertexto baseado no modelo *Trellis* para criação de documentos estruturados. O modelo *Trellis* não só representa as relações que agregam pedaços de um documento, mas define uma semântica que pode ser associada ao hipertexto. O modelo é baseado em *redes de Petri*, sendo uma generalização de sistemas baseados em grafos orientados.

O sistema *α Trellis* [Sto 89] está sendo implementado em uma estação Sun-3 utilizando o gerenciador de janelas SunView, e tem por objetivo

principal a prova de teoremas. Assim sendo, a implementação está voltada para representação e implementação de *redes de Petri*, sem um sofisticado mecanismo de visualização. O *Trellis* divide a tela em duas metades: uma para apresentar o hipertexto como uma *rede de Petri* e a segunda, dividida em outras quatro, que apresenta os nós do hipertexto. Assim, a navegação pode ser feita tanto nas referências como na *rede de Petri*.

2.7 Comparação dos Sistemas de Hipertextos

As tabelas 2.1 e 2.2 [Con 87] apresentam uma comparação dos principais sistemas anteriormente descritos. Cada coluna apresenta uma capacidade ou facilidade que o sistema de hipertexto dispõe. A afirmação ou negação indica se o respectivo sistema de hipertexto possui tal facilidade. Os critérios de avaliação são os seguintes:

- hierárquico: suporta estrutura hierárquica;
- grafo: sistema suporta estrutura em rede;
- ligações com tipo: permite ligações com tipo associado;
- atributos: associa atributos a nós e ligações;
- *path*: existência de ligações pré-definidas para seguir no hipertexto;
- versões: nós e ligações com mais de uma versão;
- ações: são comandos (eventos) associados a nós que quando ativados disparam um processo;
- pesquisa: pesquisar o hipertexto por palavra-chave ou seqüência de palavras;
- editor de texto: editor usado para criação e alteração dos nós e ligações;
- multiusuário: permite vários usuários editarem e percorrerem o hipertexto simultaneamente;
- figuras ou gráficos: existe algum suporte gráfico além de textos;

Sistema	Hierárquico	Grafo	Ligações com tipo	Atributos	*Path*	Versões	Ações
Boxer	Sim	Sim	Fixo[13]	Não[13]	Não	Não	Sim
CREF	Sim	Sim	Sim	Não	Não	Sim	Não
INFO	Sim	Não	Não	Não	Não	Não	Não
gIBIS	Sim	Sim	Sim	Não	Não	Sim	Não
Intermedia	Sim	Sim	Sim	Sim	Não[14]	Não	Não[14]
KMS	Múltiplas	Sim	Fixo	Não	Não[13]	Sim	Sim
Neptune	Sim	Sim	Sim	Sim	Não	Sim	Sim
NLS	Sim	Sim	Sim	Sim	Sim	Sim	Sim
NoteCard	Múltiplas	Sim	Sim	Nós	Não	Não	Sim
Processador de esboços	Sim	Sim	Sim	Sim	Sim	Sim	Sim
PlaneText	Unix file sys.	Sim	Não	Não	Não	Não	Não
Verificador de docum. simbólico	Sim	Sim	Não	Não	Sim	Não	Não
SYNVIEW	Sim	Não	Não	Não	Não	Não	Não
Textnet	Múltiplas	Sim	Sim	Sim	Sim	Não	Não
Hyperties	Não	Sim	Não	Não	Não	Não	Não
WE	Sim	Sim	Não	Fixo	Não[14]	Não[14]	Não[14]
Xanadu	Não	Sim	Sim	Sim	Sim	Sim	Não
ZOG	Sim	Não	Não	Não	Não	Não	Não
H	Sim	Não	Não	Não	Não	Sim	Não

Tabela 2.1: Comparação entre os Sistemas de Hipertextos (primeira parte).

- *browser* gráfico: existe algum dispositivo de representação gráfica dos nós e das ligações no hipertexto;
- independência de terminal: permite trabalhar em vários tipos de terminais.

[13] programável pelo usuário.
[14] disponível na próxima versão.

Sistema	Pesquisa	Editor de texto	Multi-usuário	Figuras gráficos	*Browser* gráfico	Indep. de terminal
Boxer	Sim	Emacs	Não	Sim	Sim	Sim
CREF	Sim	Zmacs	Não	Sim	Não	Sim
INFO	Sim	Emacs	Não	Não	Não	Sim
gIBIS	Não	Editor simples	Sim	Não	Não	Não
Intermedia	Sim	Custom	Sim	Sim	Não	Não
KMS	Sim	texto/ gráfico WYSIWYG	Sim	Sim	Sim	Sim
Neptune	Sim	Smalltall-80	Sim	Sim	Sim	Não Não
NLS	Sim	Custom	Sim	Sim	Não	Sim
NoteCard	Sim	Interlisp	Sim	Sim	Sim	Não
Processador de esboços	Sim	Vários	Não	Não	Não	Não
PlaneText	Unix/ grep	SunView text ed.	Sim	Sim	Sim	Sim
Verificador de doc. simbolico	Sim	Não	Não	Não	Não	Não
SYNVIEW	Não	Unix/ ed. linha	Não	Não	Não	Sim
Textnet	pal-chave	qualquer	Não	Não	Não	Sim
Hyperties	Não[14]	Editor	Não	Sim	Não	Não
WE	Não	Smalltalk-80	Não[14]	Sim	Sim	Não
Xanadu	Não	qualquer	Não	Sim	Não	Sim
ZOG	Sim	próprio	Sim	Não	Não	Sim
H	Sim	Smalltalk	Sim	Sim	Sim[14]	Não

Tabela 2.2: Comparação entre os Sistemas de Hipertextos (segunda parte).

2.8 Vantagens do Uso de Hipertexto

A idéia de manter referências inter-textuais não é nova, mas sua importância transparece com a utilização de computadores. Como hipertexto, a literatura tradicional é rica em referências e organizada hierarquicamente. Os meios visuais de impressão normalmente restringem o leitor a uma seqüência linear, mas muitas vezes é conveniente seguir caminhos alternativos para buscar mais informações. Por exemplo, qualquer um que deseje fazer uma pesquisa sabe que necessitará de um considerável tempo para obter referências a documentos relacionados, dicionários, glossários, figuras e tabelas. Mesmo um simples livro está constantemente referenciando outras seções, rodapés e bibliografias, induzindo o leitor a sair da seqüência linear para assuntos mais interessantes.

Contudo, há problemas com o método tradicional:

- não se pode encontrar as referências de um documento: o leitor não pode facilmente encontrar aonde um livro ou um artigo é referenciado em um documento, nem o autor saber em quais documentos seu artigo é referenciado;
- quando se vasculha vários documentos por referências, o leitor deve anotar quais documentos já visitou;
- o leitor escreve anotações sobre os documentos lidos nas margens ou em folhas separadas que são normalmente difíceis de serem posteriormente encontradas;
- seguir referências em documentos requer um esforço substancial e uma grande perda de tempo, mesmo numa grande biblioteca. Evidentemente, em pesquisa *on-line*, o trabalho é mais fácil e rápido, porém não deixa de ser tedioso.

2.8.1 Edição de Documentos e Projetos

Hipertexto oferece novas possibilidades para escrever documentos e desenvolver projetos. Edição de documentos atualmente é uma atividade orientada a palavras e frases. Logicamente, os editores de texto são boas ferramentas a este nível. Entretanto, o ato de escrever é constituído por muito mais tarefas, como estruturar idéias e organizar a apresentação. Num nível mais amplo,

criar textos pode ser entendido como *projetar documentos*. A unidade a este nível são idéias e conceitos que são facilmente suportadas por hipertexto. Quando novas idéias aparecem, elas são alojadas em nós distintos e desenvolvidas em separado até o momento adequado para ligação. Processos especializados de refinamento de ambientes de hipertexto assistem o autor para, a partir de informações não-estruturadas, produzir o documento final.

2.8.2 Pesquisa

Hipertexto oferece novas possibilidades para obter acesso a grandes e complexas fontes de dados. Um documento pode ser facilmente lido na ordem em que foi escrito, sem perder a vantagem de textos não lineares e a possibilidade de organizá-los de diferentes maneiras com diferentes pontos de vista.

Outra vantagem que é bastante natural em um sistema de hipertexto é a suspensão temporária da leitura de um texto enquanto algum outro detalhe está sendo analisado. A facilidade do *path* definido formalmente por Triggs estabelece um caminho linear para outro documento através das ligações que permitem o leitor seguir o documento como se este fosse linear.

2.8.3 Outras Vantagens

- facilidade de localizar as referências: como as referências são manipuladas por computador, elas podem ser facilmente localizadas e seguidas;
- facilidade de criar novas referências: o usuário pode acrescentar novas ligações ao hipertexto, como anexar comentários;
- estruturar informações: qualquer organização, hierárquica ou não, pode ser imposta a qualquer tipo não-estruturado de informação;
- costumização de documentos: segmentos de texto podem ser tratados e utilizados de várias maneiras, permitindo que o mesmo sirva para várias funções;
- módulos de informação: como um mesmo texto pode ser referenciado de diferentes lugares, as idéias podem ser expressas mais rapidamente e sem duplicação;

- consistência de informação: como as referências estão embutidas no texto, a movimentação de um nó para outro documento não interfere nas ligações existentes;
- caminho: possibilidade de questionar ou vasculhar caminhos alternativos no hipertexto;
- colaboração: muitos usuários podem colaborar, com documentos ou referências, na produção de um mesmo texto;

2.9 Desvantagens do uso de Hipertexto

Basicamente, existem duas classes de problemas que afetam hipertexto: problemas de implementação e problemas que parecem ser endêmicos a hipertextos. Os da primeira classe incluem a demora na localização e visualização de um nó referenciado, restrições de nome, falta de *browser* e outros detalhes técnicos. Os da segunda classe são os principais e mais preocupantes que de fato limitam o uso do conceito: *desorientação* e *excesso de concentração*.

2.9.1 Desorientação

Junto com a capacidade de organizar informações muito mais complexas, o hipertexto introduz ao usuário, tanto o criador como o consultor, o problema de ter que saber aonde ele está no grafo e como poder chegar a algum outro lugar que ele conheça ou pensa que existe. Este problema é chamado de *desorientação*. Naturalmente, pode-se ter problemas de orientação com textos lineares, mas neste caso, o leitor tem somente duas opções: ir para frente ou retornar. Já o hipertexto oferece um maior grau de liberdade, conseqüentemente mais dimensões e maior potencialidade do usuário se perder. Em uma rede da ordem de 1000 nós, as informações podem facilmente se tornarem difíceis de serem encontradas ou recordadas.

Há duas soluções técnicas para o problema: um *browser* gráfico e um mecanismo de consulta por palavra-chave. Os *browsers* normalmente necessitam de grande capacidade gráfica e permitem uma boa visualização da rede. Também existem facilidades de abstração de informações na visualização para não saturar a tela (vide figura 2.5). A outra solução é aplicar ao banco de dados uma pesquisa para localizar determinados nós pela aplicação de operações lógicas sobre uma combinação de palavras-chaves,

frases, autor, data de criação e outros. Ao final da pesquisa o sistema pode filtrar informações irrelevantes ao usuário ou abstrair níveis de detalhe na visualização do resultado obtido na consulta.

O artigo [Tri 88] descreve uma ferramenta com guias e tabelas para orientar o usuário na leitura de um hipertexto utilizando o sistema NoteCard. Ele viabiliza a inclusão de comentários, *layouts* gráficos e ordena a apresentação dos nós.

Outro artigo [Nie 90] apresenta um hipertexto implementado no sistema HyperCard, utilizando o HyperTalk, com uma elaborada *interface* gráfica e um sofisticado mecanismo de *browser* para auxiliar o usuário na navegação.

2.9.2 Excesso de concentração

Operar sistemas de hipertexto requer um esforço adicional para criar, nomear e manter o controle dos nós e ligações do hipertexto ao mesmo tempo. Este problema ocorre tanto na leitura como na escrita, mas até presentemente não há uma solução eficiente para o problema, embora alguns sistemas como WE e Processadores de Esboço tentem abrandá-lo.

Um outro sistema que tenta abrandar os problemas de desorientação e excesso de concentração é o *Web View* [Utt 89], desenvolvido na *Brown Univesity* para o sistema Intermedia. É uma ferramenta de visualização e navegação num hipertexto com um mínimo de confusão e desorientação. Seu sucesso foi de combinar o caminho percorrido pelo usuário com um mapa gráfico do hipertexto, provendo uma visão global do sistema. Assim, a navegação pode tanto ser feita no mapa como nas referências.

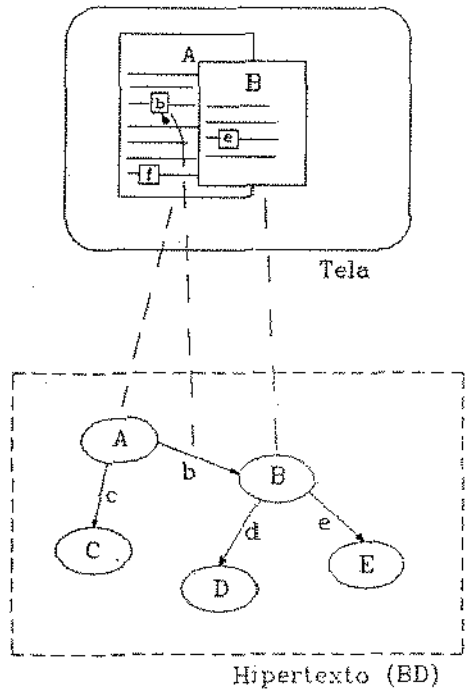


Figura 2.1: Modelo de hipertexto, com alguns nós sendo visualizados em uma tela. A ligação b no banco de dados é representado na janela A de modo destacado (uma referência). Como esta ligação foi ativada, o sistema apresentou, em uma segunda janela, o texto B.

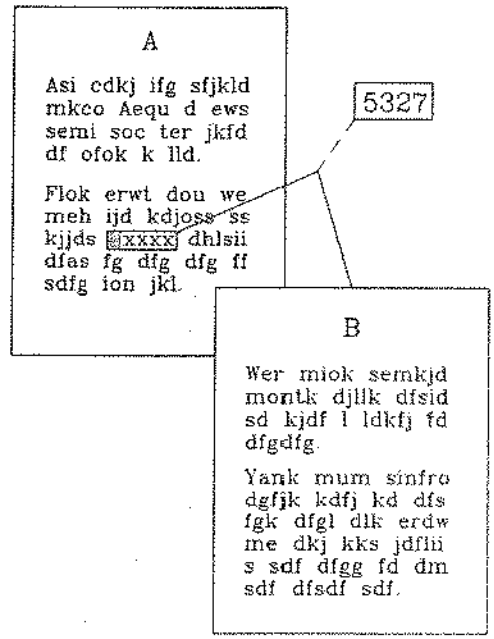


Figura 2.2: Exemplo de uma ligação referencial com ponto de origem e um de destino. A origem da ligação é representada no texto A pelo *token* xxxx que não necessariamente representa o texto destino B. Ao *token* pode estar associado um identificador que representa o nome do nó destino, o nome da ligação ou uma seqüência de caracteres qualquer. Neste caso, o nó destino é o B e a ligação possui um identificador (5327) que eventualmente pode ser visível ao usuário.

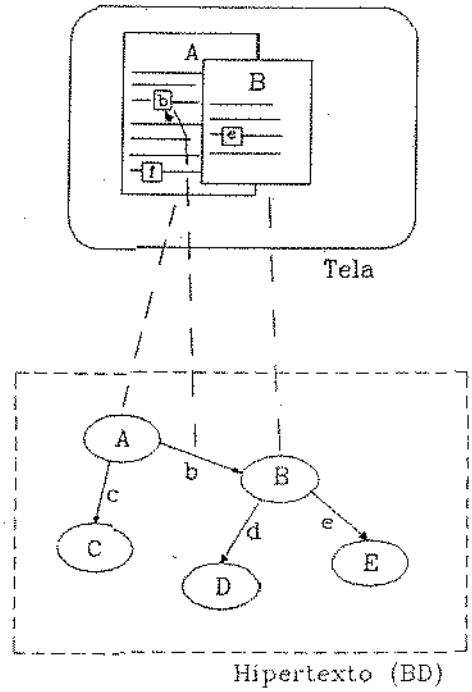


Figura 2.1: Modelo de hipertexto, com alguns nós sendo visualizados em uma tela. A ligação b no banco de dados é representado na janela **A** de modo destacado (uma referência). Como esta ligação foi ativada, o sistema apresentou, em uma segunda janela, o texto **B**.

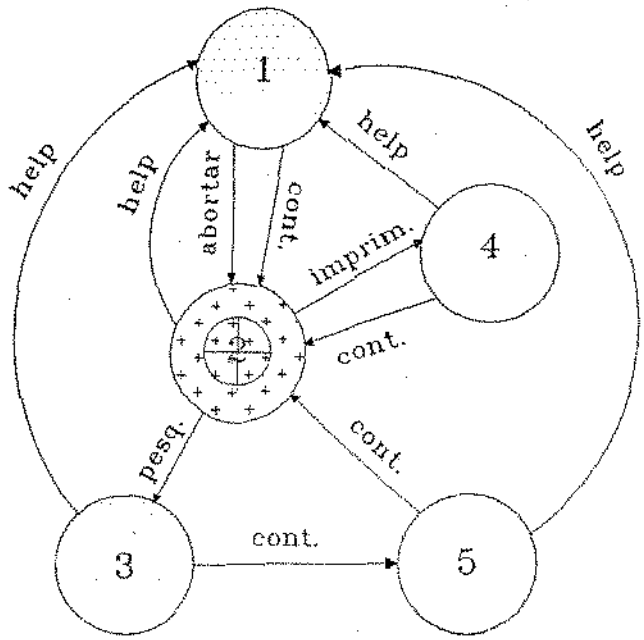


Figura 2.3: Representação de um hipertexto <N,L,E>. N={1, 2, 3, 4, 5}, L={help, pesq., cont., abortar, imprim.}, E={(2, help), (4, help), (1, abortar), (4, cont.), (3, help), (5, help), (2, imprim.), (5, cont.), (1, cont.), (3, cont.), (2, pesq.)}

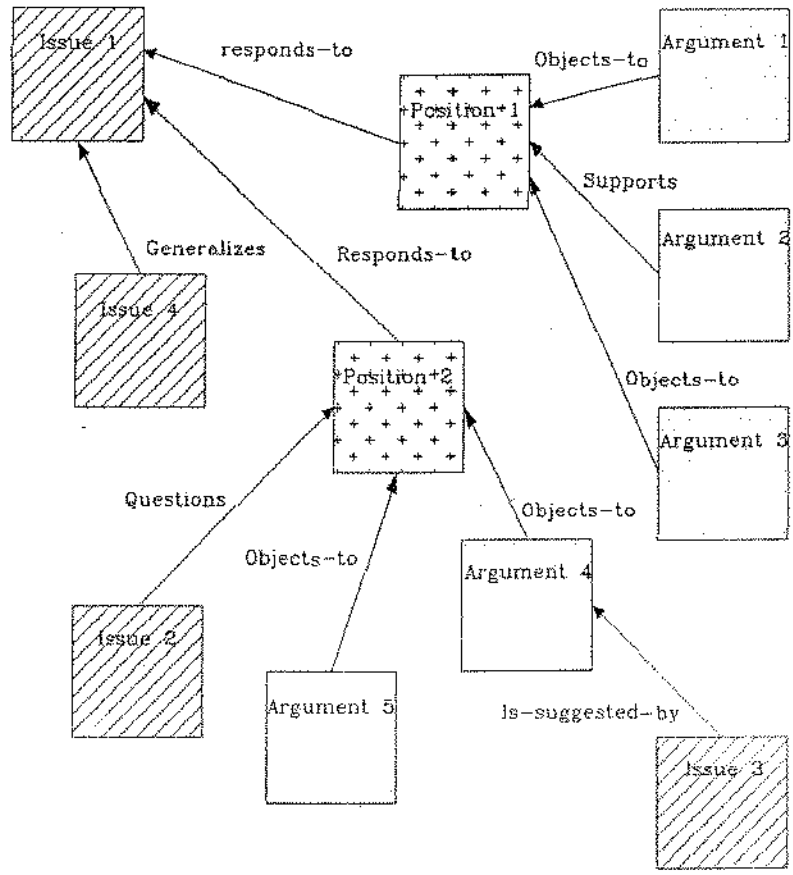


Figura 2.4: Mostra o segmento de uma possível discussão usando o método IBIS. Cada nó possui informações do tipo, hora e data de criação, autor, assunto, texto, cometários, lista de palavras-chaves e uma lista de ligações. O sentido dos vetores representa como um nó se relaciona com outro, mas no hipertexto as ligações são invertidas.

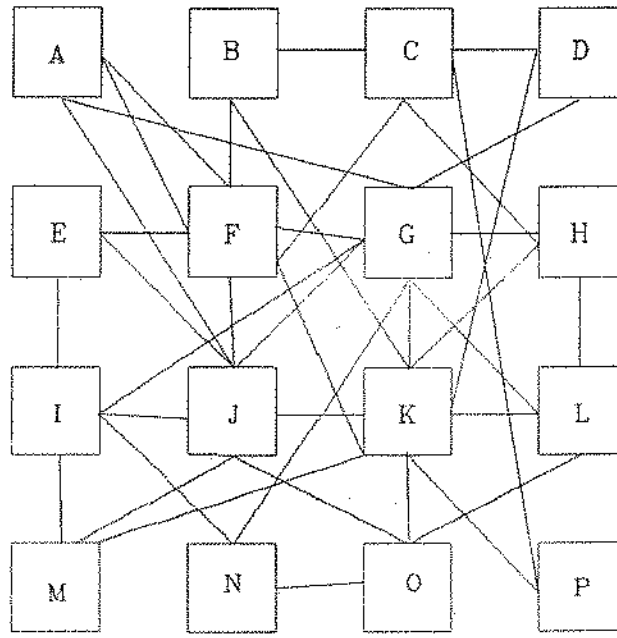


Figura 2.5: *Boruser* estrutual gráfico que apresenta um hipertexto com um número excessivo de nós e ligações, o que dificulta a orientação do usuário.

Capítulo 3

Ambientes de Desenvolvimento de Software

Este capítulo descreve alguns pontos que envolvem Ambientes de Desenvolvimento de *Software* no que tange sua utilização a hipertexto. Este capítulo não pretende fazer um levantamento completo sobre os Ambientes de Desenvolvimento de *Software* limitando-se a tópicos relevantes.

3.1 Introdução

Com recentes avanços da tecnologia, computadores de grande potencialidade foram lançados no mercado, com velocidades, memórias, periféricos e dimensões jamais visto anteriormente. Assim, faz-se necessário desenvolver novas técnicas e ferramentas que tirem o melhor proveito destas novas máquinas para produzir *software* mais rápido, correto e eficiente.

Ambientes de desenvolvimento de *software* vêm exatamente de encontro a esta lacuna e têm como principais funções a edição, interpretação, compilação, ligação, depuração e gerenciamento de programas. Entretanto, pode-se ver ambientes de desenvolvimento num nível de abstração diferente, com funções mais abrangentes e complexas como especificação, estruturação, codificação, integração, testes, documentação, edição de manuais, manutenção e re-utilização de *software*. Esta última abordagem assiste todo o

desenvolvimento de um projeto, sendo extremamente desejável para sistemas grandes e complexos.

3.2 Auxílio à Modularização

Normalmente, um programa é dividido em partes ou módulos durante o seu processo de desenvolvimento, ficando cada programador responsável por apenas um ou um grupo de módulos. Cada módulo pode ser usado por outros módulos e vários programas podem compartilhar o mesmo módulo. A estrutura das dependências entre os módulos pode ser representada como um grafo. Note que o compartilhamento de módulos impede que a estrutura seja uma árvore simples. Não sendo uma árvore, não se pode criar uma estrutura de diretórios que seja isomorfa à estrutura de dependências. Normalmente os módulos são localizados em função do projeto a que faz parte e do usuário responsável pelo seu desenvolvimento. Como resultado, estes módulos ficam espalhados no sistema de arquivos sem qualquer conexão física que indique o seu inter-relacionamento. Em programas grandes, é muito difícil o gerenciamento de forma isolada, sendo necessário uma ferramenta que integre o programa como um todo.

Se o programa for desenvolvido na forma de um hipertexto, isto é, cada módulo é um nó, a integração dos módulos é feita naturalmente pelas dependências existentes entre eles, representados por referências.

3.3 Re-utilização de Software

A re-utilização de *software* no processo de desenvolvimento de programas grandes é indispensável. Com auxílio de um hipertexto, a re-usabilidade de um módulo é indicada simplesmente por uma referência, ou seja, uma ligação.

3.4 Documentação

Documentar programas sem uma ferramenta adequada é muito difícil, principalmente pela independência física entre programa e documentação. Isto propicia uma documentação desatualizada ou até mesmo inexistente para certos módulos.

Assim, a documentação pode ter a mesma estrutura que o programa. É conveniente existir uma ligação entre cada módulo de programa e sua documentação.

3.5 Outras Aplicações

A criação de uma biblioteca de *software* e de sua documentação na forma de hipertexto permite o re-aproveitamento de *software* na própria concepção da palavra. Pode existir não só uma biblioteca de *software*, mas de documentos onde todos usuários acrescentam textos e comentários.

Um mecanismo de teleconferência é possível utilizando hipertexto onde um nó é uma janela de comunicação (compartilhada ou individual). Assim, para realizar uma conferência é necessário marcar a data numa agenda eletrônica, que pode também ser um hipertexto. A conferência é realizada pelas janelas de comunicação. Evidentemente, deve haver um controle estrito de acesso e permissões na comunicação.

O correio eletrônico tem por objetivo principal a troca de informações, como por exemplo, documentos, programas, comentários e gráficos. Normalmente, deseja-se enviar um conjunto de informações, que podem ser transmitido individualmente ou agrupadas numa única mensagem. Neste último caso, o destinatário tem que fazer a separação das mensagens. Usualmente, esta mensagem é composta de um cabeçalho que descreve o pacote, indicando o conteúdo, o começo e o fim de cada parte. Este conceito é bastante similar a um hipertexto. Assim, o modo mais adequado de enviar mensagens é via hipertexto, onde o usuário referencia as diversas informações a serem enviadas. O hipertexto, por um comando especial, agrupa as mensagens num único arquivo, que é então enviado. O processo de recepção separaria as mensagens na forma de um hipertexto.

Durante a leitura de um documento no hipertexto, é possível acrescentar comentários, observações e referências a outros textos ao documento. Esta é uma atividade extremamente comum e útil, podendo ser implementada de maneira muito simples em hipertexto.

3.6 Integração usando hipertexto

As seções anteriores descrevem algumas fases do processo de desenvolvimento de *software* e mostra que todas elas falham no sentido de não integrar o sistema como um todo. Não existe um intra e inter-relacionamento das fases de desenvolvimento. Portanto, o problema reside na integração do sistema como um objeto único de trabalho.

3.7 A_HAND

O A_HAND [Dru 87] é um Ambiente de desenvolvimento baseado em Hierarquias de Abstração em Níveis Diferenciados que está sendo desenvolvido no Departamento de Ciências da Computação da Unicamp. Ele corresponde a parte do AIDSH[1] epecífica para o desenvolvimento de *software* e está sendo desenvolvido no sistema Unix (System V, release 3). Para o A_HAND, módulos ou programas são tratados como objetos que podem ser compostos de objetos mais simples (outros módulos ou programas). Assim, os objetos do A_HAND que são os programas formam uma hierarquia, promovendo níveis de abstração que dão origem ao nome do projeto. Os programas que serão desenvolvidos sob o A_HAND são sistemas grandes e complexos.

O sistema deve prover as mesmas facilidades de desenvolvimento em todas as fases de programa, mesmo que este se encontre em um estágio parcialmente concluído. Portanto, o ambiente de desenvolvimento deve ser capaz de manipular sistemas incompletos, gerenciando sua documentação, promovendo testes e permitindo execução de parte do sistema.

O desenvolvimento de programas grandes exige um grupo de pessoas que estão trabalhando, não necessariamente em um mesmo local de trabalho. O A_HAND deve suportar o desenvolvimento distribuído promovendo ferramentas específicas para integração das partes projetadas, e ainda permitir testes e execução das partes de forma isolada ou já integradas.

Outra característica importante é a redução do tempo de desenvolvimento e a capacidade de re-utilização de código, o que implica no aumento de produtividade dos projetistas. A linguagem de programação de mais baixo nível do A_HAND é o Cm[2], que oferece além das características da

[1] Ambiente Integrado de Desenvolvimento de Software e Hardware

[2] "C" modular e polimórfico

liguagem "C", facilidades de programação para grandes projetos (herança e abstração via classes).

O sistema ht servirá como uma ferramenta de trabalho do A_HAND. Sua principal função será a integração, estruturação e documentação de programas. Isto proporciona aos componentes de um projeto ter uma visão geral do sistema de como está sendo desenvolvido.

3.8 Hipertextos para Desenvolvimento de Software

Esforços já estão sendo realizados no sentido de desenvolver sistemas de hipertextos para desenvolvimento de software. Como foi dito, o gIBIS (seção 2.6.2) é uma ferramenta para auxiliar na especificação de projeto em grupo. Existem outros sistemas mais orientados a esta área que visam abranger todas as fases da produção de software.

3.8.1 DIF

O DIF (*Documents Integration Facility*) [Gar 88] [Lim 89] [Mir 90] é um sistema de hipertexto para gerenciar os documentos criados durante o ciclo de vida do *software* de vários projetos. Estes documentos representam as diversas fases que passa o projeto e é conhecido como *System Factory*. Este método divide o processo de desenvolvimento de *software* em oito fases distintas [Mir 90]:

Especificação de Requisitos apresenta os requisitos operacionais do projeto, implementação do sistema e recursos organizacionais;

Especificação Funcional define as funções computacionias do sistema;

Especificação da Arquitetura do Projeto define a estrutura e a *interface* dos módulos do sistema;

Especificação de Detalhes do Projeto refinamento da fase anterior com relação a recursos e comportamento dos módulos;

Código Fonte implementação;

Documento de Teste e Qualificação resultados dos testes do sistema;

Manual do Usuário apresenta ao usuário como operar o sistema;

Manual de Manutenção apresenta como dar suporte ao sistema.

O DIF suporta dois modos de operação: *super-usuário* e *usuário-geral*. O primeiro é responsável pela estrutura do sistema: define, modifica e cria os domínios do sistema (*Forms Domains* e *Basic Templates* – BT). Os *Forms* contém o resultado de cada fase do ciclo de vida do software enquanto os *Basic Templates* contém o resultado obtido em cada ação que compõe a fase. O *usuário-geral* é responsável pelo desenvolvimento do sistema propriamente dito. Em outras palavras, os *Forms* definem a estrutura organizacional do projeto, estabelecendo assim, um padrão para todos os individuos envolvidos. Os BTs armazenam informações do tipo: narrativa, gráfico, NuMil[3], Gist[4], texto, código fonte e código objeto.

A estrutura do hipertexto é mantida num banco de dados relacional *Ingres* e a informação do hipertexto é mantida no sistema de arquivos Unix. Ele foi implementado numa estação *Sun BSD 4.2* utilizando *X-windows V11* e *Sun Ingres V5*.

3.8.2 Dynamic Design

O *Dinamic Design* [Big 88] [Mir 90] é uma ferramenta CASE[5] para o desenvolvimento de sistemas em "C" baseado num ambiente de hipertexto. Este sistema de hipertexto, derivado do *Neptune* (vide seção 2.6.4), utiliza o *Hypertext Abstract Machine*, um tipo de banco de dados, que oferece operações genéricas para criar, modificar e acessar os componentes de um hipertexto. Além disto provê acesso distribuído, sincronização e recuperação de transações incompletas.

O *GraphBuild* é um utilitário para converter código "C" num grafo fonte de hipertexto, onde cada função é alocada em um nó. Desta forma, o usuário por meio do *source browser* pode navegar e editar o código como um hipertexto.

Para o desenvolvimento de *software* o *Dinamic Design* define o que seriam os componentes de um projeto:

[3] Linguagem de configuração
[4] Analisador e simulador de especificação
[5] Computer Aided Software Engeneering

- Especificação e requisitos;
- Modelagem de notas e documentos;
- Implementação das notas;
- Código fonte e objeto;
- Documentação.

Os nós do hipertexto contém os componentes do projeto e as ligações associam os nós descrevendo a relação existente entre eles.

Capítulo 4

Sistema HiperTexto: ht

Este capítulo tem por objetivo apresentar o sistema HiperTexto, chamado de **ht**, que foi desenvolvido no Departamento de Ciência da Computação da Unicamp. Descreve seus conceitos, compara com outros sistemas, apresenta algumas aplicações do **ht** e sua utilização no desenvolvimento de *software*.

4.1 Introdução

Os sistemas de hipertexto apresentados na seção 2.6 são destinados basicamente à edição, recuperação e manipulação de documentos sem uma aplicação específica em ambientes de desenvolvimento de *software*. Os sistemas apresentados na seção 3.8 são arquiteturas fechadas e orientadas a linguagem "C", não satisfazendo alguns requisitos do ambiente A_HAND, como por exemplo a liberdade de utilizar outras linguagens de programação. Além disto, são sistemas completos para o desenvolvimento de *software* compreendendo todas as fases do ciclo de vida de um projeto, não sendo o objetivo do *ht* no A_HAND. O sistema **ht**, além de manipular hipertextos, é uma ferramenta de grande potencialidade para o desenvolvimento de programas, sanando as questões de integração e gerência que afetam em geral os ambientes de desenvolvimento de *software* (vide capítulo 3), atendendo os propósitos do A_HAND.

Um nó de um hipertexto no sistema **ht** é armazenado em um único arquivo do sistema de arquivos. Uma ligação é o nome do arquivo a que o nó se refere convenientemente inserido nele. Assim, define-se um nó como

um *arquivo* e uma ligação como o *nome de um arquivo*. A decisão de se alojar um nó em um arquivo deve-se à facilidade oferecida pelo sistema Unix, no tocante a organização hierárquica de arquivos, abrangendo restrições de acesso e alterações. Desta forma, representa-se cada módulo de programa em um nó de hipertexto, sendo sua estrutura livre, dependendo apenas do programa, ou seja, da maneira com que os módulos se relacionam.

Como em outros sistemas de hipertexto, o **ht** armazenados os nós com seqüências de caracteres ASCII com comandos embutidos. Quando um nó é referenciado, o sistema interpreta tais comandos e apresenta o seu conteúdo na tela. O tempo de leitura, formatação e visualização são desprezíveis com a utilização dos atuais computadores. Além disto viabiliza a independência de terminal, que é um fator importantíssimo no desenvolvimento de programas que envolve um grande número de pessoas.

Os comandos do **ht** são inseridos nos nós e definem suas características, como por exemplo, tamanho da janela de visualização e ligações a outros nós. Estes comandos tem sintaxe semelhante aos comandos do formatador de texto LaTeX. Este formatador, além de armazenar seus textos como uma seqüência de caracteres, é o mais utilizados nos ambientes acadêmicos, de pesquisa e de desenvolvimento. Portanto, todos os comandos do **ht** são precedidos do caracteres "\", seguido do seu nome. Quando necessário, seguem os argumentos entre chaves separados por vírgula:

\comando {*arg_1* [,*arg_2* ...]}

No ht pode-se especificar o *path* (vide seção 2.3.1), que é uma seqüência de nós pré-definidos a serem visitados. Para isto, cada nó que faz parte desta seqüência possui uma marca em uma única referência indicando que esta faz parte do *path* que leva ao nó seguinte. O *path* é utilizado principalmente para gerar a documentação do programa.

O processo de desenvolvimento de *software* é feito normalmente por um grupo de pessoas que freqüentemente podem estar trabalhando ao mesmo tempo. Esta simultaneidade é suportada pelo **ht**, permitindo que várias pessoas acessem um mesmo nó, porém somente uma delas pode estar alterando o seu conteúdo.

O **ht** oferece a possibilidade de busca por expressão regular em outros nós do hipertexto. Esta facilidade permite uma melhor orientação do usuário na sua navegação, caracterizando ligação implícita.

Como o **ht** é destinado principalmente ao desenvolvimento de *software* criou-se o conceito de tipo de nó. Assim, conforme o tipo do nó a ser visualizado, o **ht** comporta-se diferentemente, como por exemplo, um nó de programa (código) e outro de documentação.

O **ht** não é de uso exclusivo ao desenvolvimento de programas, mas explicitamente possui tal capacidade graças ao conceito de modos de operação. Um modo define como o **ht** vai manipular, navegar e visualizar um hipertexto. Assim pode-se definir novos modos, por exemplo, um modo chamado LaTeX*MODE* que entende comandos de LaTeX.

Uma vez visualizado um nó, o usuário pode utilizar uma série de funções utilizando o *browser*, como por exemplo, navegar no texto, ativar uma referência e procurar por expessão regular. A associação dos comandos do *browser* ao teclado pode ser reconfigurada pelo usuário a qualquer momento durante a visualização de um nó. Importante a distinção dos comandos do *browser* com comandos inseridos em um nó.

A localização de nós pelo **ht** é feita de forma semelhante à que o processador de comandos *shell* usa para localizar programas. Define-se uma variável de ambiente chamada **HT** que contém os diretórios onde o **ht** deve procurar por nós quando estes forem referenciados.

Existe o conceito de referência corrente, ou seja, qual de todas as referências de um nó que está selecionada para ser ativada. Uma referência corrente está em video reverso ou piscante. A seleção é feita via teclado utilizando apenas uma tecla. Esta seleção, em outros sistemas de hipertexto, é feita normalmente por um *mouse*, mas como o **ht** se propõe a ser um programa independente de terminal, optou-se pelo teclado.

O **ht** também oferece facilidades para estruturação e documentação de programas, além de viabilizar a criação de bibliotecas de *software*. Oferece opções de linearização do hipertexto em um único arquivo para depuração e impressão. O **ht** foi implementado no sistema Unix na linguagem "C" [Ker 86], sendo totalmente portável para outras plataformas. Hoje, ele executa no Digirede 8000 (sistema *Digix*), Interpro 220 (sistema *CLIX*) e SparcStation 1+ (sistema *SunOS 4.1*).

4.2 Conceitos

A seguir são apresentados os conceitos que envolvem a utilização do **ht** como ferramenta para o desenvolvimento de *software*:

4.2.1 Tipos de Nó

O sistema **ht** permite associar um tipo a um nó no hipertexto. Cada tipo define de que modo um nó será visualizado e que operações serão permitidas. No total exitem cinco tipos de nós:

Document: antes de visualizado, o texto do nó é pré-processado e formatado de acordo com as dimensões da janela de apresentação. Os comandos de formatação de texto constituem um subconjunto dos comandos LaTeX[Lam 86];

Text: o texto é apresentado como editado, sem qualquer alteração;

Program: é um nó que contém código de programa, e é apresentado como editado, se bem que, em algumas linguagens de programação, o código pode passar por um *pretty printing* antes da visualização;

Card: é um tipo de nó destinado a apresentação de mensagens; é muito rápido;

Command: quando ativada uma referência a este tipo de nó, os comandos nele definidos são executados com a entrada e saída padrão redirecionados para um janela, possivelmente definida pelo usuário.

4.2.2 Tipos de Letra

Além do tipo de nó, pode-se definir o tipo de letra de apresentação do texto. Como a independência de dispositivo é fundamental para o projeto, o uso de terminais alfanuméricos limita a fidelidade de apresentação dos caracteres. Portanto, convencionou-se um mapeamento entre tipos de letra e atributos de vídeo (definidos em parênteses):

- \bf **bold face** (bold)
- \rm roman (normal)

- \em \it *emphatic* e *italic* (reverse)
- \tt \sf `typewriter` e sans (blink)
- \sl *slanted* (dim)
- \sc CAPS (UNDERLINE)

Os comandos em LaTeX acima apresentados, precedidos de "\", que especificam o tipo de letra, também são válidos.

4.2.3 Variáveis Internas

São variáveis locais e globais a cada nó do hipertexto que definem como o sistema **ht** vai manipular e apresentar os nós e as ligações. As são as seguintes:

GREP: indica em quais diretórios será feira a busca por nós que possuem determinada expressão regular, fornecida pelo usuário.

LINE: quando visualizado o texto de um nó numa janela, ele é apresentado a partir da linha definida na variável LINE (*default* LINE=1);

REFERENCE: o número da referência (contada a partir da primeira) é definida como corrente, assim pode-se definir *path* no hipertexto (*default* REFERENCE=1). Portanto, nem sempre a primeira referência será a corrente na ativação de um nó;

DIR: define um conjunto de diretórios nos quais, além dos já mencionados na variável de ambiente HT, o sistema irá procurar por nós do hipertexto (*default* DIR indefinido).

EDITOR: define novo editor de texto em substituição ao da variável de ambiente EDITOR. Assim pode-se definir editores diferentes para nós com conteúdos diferentes.

As variáveis internas também são reponsáveis pela associação dos comandos (vide tabela 4.1 e 4.2) ao teclado. Assim, o usuário pode facilmente reconfigurar o teclado conforme seção 4.2.4.

4.2.4 Comandos - A Linguagem de "Markup"

Por definição, qualquer arquivo é um nó do hipertexto mesmo que ele não referencie ou seja referenciado por outros. Entretanto, pode-se utilizar os comandos de especificação, que são inseridos no texto, que definem características dos nós e das ligações. Os comandos têm sintaxe semelhante aos comandos LaTeX, que são de simples edição facilitando a geração de documentos para impressão. Os comandos são de dois tipos, de acordo com a sua disposição no nó:

fixos: numa mesma linha de texto que está definido algum comando fixo, somente comandos (fixos ou flutuantes) são reconhecidos. Se existir qualquer outro tipo de texto, provoca erro de visualização;

flutuantes: são comandos que podem estar em qualquer lugar do nó, normalmente junto com o texto.

Os itens que seguem apresentam a sintaxe e a semântica destes comandos.

Header

O comando **header** especifica o nome e o tipo do nó e, opcionalmente, o tipo de letra do texto.

\header {"*cabeçalho*", *tipo-nó*, [*tipo-letra*]}

O *cabeçalho* é uma seqüência de caracteres que são apresentados na primeira linha da janela de apresentação se o comando **window** for especificado. Na ausência do comando **header** é assumido o tipo *text* e o tipo de letra *normal* sem cabeçalho. **Header** é um comando fixo.

Exemplos:

\header {"Módulo Pilha", program, normal}

\header {"Introdução", document, bold}

Window

Este comando define a posição e as dimensões da janela de apresentação do nó. Na sua ausência, o nó é apresentado em toda a tela do terminal sem moldura.

\window {*x*, *y*, *lin*, *col*}

As coordenadas *x* e *y* especificam o canto superior esquerdo da janela e *lin* e *col* são suas dimensões acrescidas de duas unidades para comportar a borda. Os argumentos *x*, *y*, *lin* e *col* são números inteiros positivos. Se *lin* e/ou *col* excederem a capacidade do terminal, um aviso será dado e os limites serão ajustados.

Se o tipo do nó for *document*, é necessário que o comando **window** apareça antes de qualquer texto, para indicar ao formatador o número de colunas da janela. Esta restrição permite ao carregador de nós realizar somente uma passagem pelo texto. O **window** é um comando fixo.

Exemplo:

\window {10, 0, 9, 70}

Mark

Além das referências entre-nós, é possível referenciar segmentos de texto de um mesmo nó. O segmento a ser referenciado é rotulado pelo comando **mark** com um identificador de nome. Opcionalmente, pode-se redefinir o tipo de letra do novo segmento.

\mark {*rótulo*, [*tipo-letra*]}

Não há limite para o número de rótulos em um mesmo nó, porém não pode existir mais de um rótulo com o mesmo nome. Este comando é do tipo flutuante.

Exemplos:

\mark {parte_2}

\mark {tabela_1, reverse}

Link

O comando **link** estabelece uma ligação de um nó para outro nó. Vários atributos do nó referenciado podem ser redefinidos.

\link {[*novo-tipo-nó*,] *arquivo*, [*argumento*, ...] "*mensagem*", *tipo-letra* [,*x*, *y*, *lin*, *col*, *marca*]}

Uma referência é criada para o nó destino *arquivo* e, dentro do *arquivo*, para o rótulo *argumento* se este estiver definido por um comando **mark**. Caso a ligação especifique um rótulo e este não esteja definido, um erro de visualização ocorre. A *mensagem* é apresentada na janela de visualização do nó que possui o **link** com o *tipo-letra* especificado. A *marca* indica que esta é a referência corrente na visualização do nó: isto viabiliza a criação de *path*. Opcionalmente, pode-se modificar a janela de visualização do nó referenciado definindo seus novos limites em *x*, *y*, *lin* e *col*.

O comando **link** permite ainda redefinir o tipo do nó referenciado com o argumento *novo-tipo-nó*. Caso redefinido, o nó referenciado quando ativado, é tratado e processado de acordo com o novo tipo. Se o novo nó é do tipo *command*, então, o *arquivo* é o nome de um programa executável e *argumento* são seus argumentos na linha de comando. É importante destacar que não serão processados os *wildcards*[1] dos argumentos.

O sistema ht permite estabelecer ligações a nós inexistentes, embora sua visualização não seja permitida. Este tipo de facilidade é conveniente para edição de documentos *top-down*.

O comando **link** é do tipo flutuante.

Exemplos:

\link {pilha.c,"Módulo Pilha",bold}

\link {capítulo_2,seção_2.1,"Seção 2.1", bold, 10, 0, 14, 70, mark}

\link {command, ps, -efl, "ps", reverse}

Exec

O comando **exec** define um comando em um nó, que é executado quando a referência é ativada.

[1] são os caracteres de generalização: ? e *.

\exec {*comando* [, *arg1*, *arg2*, ...]}

O *comando* é o nome de um programa executável e *argn* são seus argumentos. Não serão analisados os *wildcards* sendo simplesmente repassados ao processo criado. A opção de não se processar os *wildcards* foi tomada em favor de um ganho na velocidade de criação do processo *comando*. Pode existir num mesmo nó do tipo *command* mais de um comando **exec**, assim, os processos serão executados seqüencialmente.

Se o comando **exec** estiver definido em um outro tipo de nó, diferente de *command*, o processo é executado em paralelo[2], com a entrada padrão fechada[3] e com a saída padrão redirecionada para um arquivo temporário que posteriormente pode ser visualizado pelo **ht**. Comando do tipo fixo.

Exemplos:

\exec {make}

\exec {cc, main.c, pilha.c, -o, pilha}

Setvar

O comando **setvar** atribui valores às variáveis internas de um nó.

\setvar {*variável*}[{*valor*}]

Se *valor* for especificado, então *valor* é associado a *variável* e pode ser um dos especificados na seção 4.2.3.

O comando **setvar** permite também associar comandos a serem executados quando uma tecla é pressionada. Para isto a *variável* é a tecla ou um código ASCII e *valor* é um comando entre aspas. O *comando* pode ser qualquer cadeia válida, inclusive *wildcards*. Isto é possível porque, neste caso, o processo é executado através de uma *shell* do sistema operacional, que em geral é capaz de expandir os nomes de arquivos ambigüos. O **ht** tem vários comandos pré-definidos associados às teclas, que podem ser redefinidos (vide tabelas 4.1 e 4.2).

\setvar {'*letra*'}[{"*comando*"}]

[2] *background*

[3] redirecionada para */dev/null*

\setvar {*código*} [{"*comando*"}]

As variáveis alteradas em um nó, valem somente para esse nó. Portanto, se um novo nó é ativado as variáveis passam a ter os valoras *defaults* a não ser que sejam modificados. O retorno a um nó provoca a restauração de suas variáveis.

Exemplos:

\setvar {REFERENCE}{3}

\setvar {DIR}{ cap/ht:/usr/doc}

\setvar {'m'}{"make"}

\setvar {'p'}{PUSH_HT}

Setmode/Resetmode

Idêntico ao comando **setvar**, porém uma definição utilizando este comando permanece até o fim da sessão ou até a sua remoção por um **resetmode**. Uma declaração local prevalece sobre uma global.

Exemplos:

\setmode {DIR}{/lib/man: pedro/doc:/lib/ht}

\setmode {'m'}{"make"}

Multilink

Multilink estabelece uma ligação de um nó para vários nós simultaneamente. Quando ativada, abre uma janela com as opções das ligações para o usuário.

\multilink {*mensagem, tipo-letra, ref1* [, *ref2* ...]}

Este comando é útil para edição de documentos com várias versões, como por exemplo, a edição de manuais em que uma função tem características diferentes em máquinas diferentes. Pode ser implementado visões com ele.

Exemplo:

```
\multilink {"Acesso a Arquivos", bold,
    \link {dos.doc,"DOS",bold},
    \link {unix.doc,'Unix",bold,mark},
    \link {vms.doc,"VMS",bold},
    \link {command,phone,root,"Chamar Operador",bold}}
```

4.2.5 Variáveis de Ambiente

As variáveis de ambiente especificam algumas variáveis internas do **ht**. Atualmente, o **ht** utiliza duas variáveis que identificam o editor de texto e os diretórios onde se encontram os nós do hipertexto.

A variável EDITOR define o nome do editor preferido pelo usuário. Atualmente, o **ht** reconhece dois editores de texto: vi e emacs (Micro-emacs), sendo o primeiro o *default*. Os comandos do ht são configuráveis de acordo com o editor utilizado.

A segunda variável, HT, define os diretórios onde estão os nós do hipertexto, além do diretório corrente. O formato é semelhante ao da variável de ambiente PATH.

A possibilidade de utilizar variáveis de ambiente, além das variáveis internas, capacita o usuário a definir seu próprio ambiente de trabalho sem ter que especificar no nó adicional, evitando alterar o hipertexto.

4.2.6 Localização de Nós

O sistema ht se utiliza de três recursos que indicam onde pesquisar por nós de um hipertexto. O primeiro é a variável de ambiente HT, que especifica os diretórios onde devem ser procurados os nós. A segunda é a variável interna DIR, definida localmente em nó, e tem o mesmo formato da variável de ambiente HT. A última é o diretório de trabalho, que é atualizado de acordo com o diretório do nó que está sendo visualizado. Observe que o diretório de trabalho é dinâmico e permite localizar nós de acordo com a pesquisa realizada. Portanto, o diretório de trabalho não é necessariamente aquele de onde o usuário ativou o ht.

4.2.7 Proteção

O sistema ht utiliza a proteção de arquivos disponível no sistema Unix para restringir o acesso a nós. Assim, cada nó possui três níveis de proteção: dono do arquivo (*user*), membro do grupo (*group*) e outros. Para cada tipo de usuário define-se as permissões de leitura, escrita e execução do nó.

4.3 Comandos do Browser

Os comandos *browser* estão divididos em duas categorias funcionais. A primeira é responsável pela movimentação do texto na janela de visualização e a segunda é encarregada da navegação no hipertexto. Cada comando está associado a uma tecla que eventualmente pode ser redefinida pelo nó através do comando **setvar** e **setmode**. Deve-se ter apenas o cuidado de não associar dois comandos a uma mesma tecla.

4.3.1 Comandos de Movimentação

São comandos funcionalmente comuns a qualquer editor de texto, embora associados a teclas diferentes. A tabela 4.1 apresenta estes comandos seguidos de uma breve descrição.

4.3.2 Comandos de Navegação

São comandos encarregados de operações que envolvem nós e ligações do hipertexto. A tabela 4.2 apresenta os comandos e uma breve descrição.

Segue uma melhor explicação destes comandos:

PUSH_HT a referência corrente é ativada, e, em uma segunda janela, o texto é apresentado de acordo com o tipo do nó.

POP_HT retorna ao nó anterior, ou seja, àquele em que se estava antes da última referência ser ativada. Se o nó corrente for um processo, é necessário primeiro que ele termine para depois ser invocado o comando POP_HT. Se existir algum processo executando no *background* que foi ativado por este nó por um comando *exec*, ele será abortado.

Comando	função
UP	cursor para cima
DN	cursor para baixo
LF	cursor para esquerda
RT	cursor para direita
TOP	início do texto
BOTTOM	fim do texto
NEXT_PAGE	página seguinte
PREV_PAGE	página anterior
SCROLL_DOWN	meia página seguinte
SCROLL_UP	meia página anterior
SEARCH	pesquisa por expressão regular (nó corrente)
NEXT_SEARCH	próxima ocorrência do padrão no texto

Tabela 4.1: Comandos de movimentação.

NEXT_REF posiciona a próxima referência como corrente. Se esta for a última, então posicionará a primeira como corrente. Ao ser posicionada uma referência como corrente, o tipo de letra da referência é alterado para reverso (*reverse*) ou piscante (*blink*) conforme o tipo corrente.

PREV_REF idem a NEXT_REF, porém seleciona referência anterior como corrente.

EDITOR edita o nó corrente usando o editor especificado na variável EDITOR. Ao retornar, o nó é processado e visualizado na mesma janela que estava quando foi chamado o editor. Portanto, uma redefinição do comando window não modificará a janela de visualização. Esta alteração só terá efeito quando o nó for desativado (fechado) e ativado novamente. Caso ocorra algum erro de processamento, o nó que está sendo visualizado não é alterado.

O usuário para visualizar um nó deve ter o direito de leitura do arquivo que aloja este nó, caso deseje modificá-lo é necessário o direito de escrita. Como o ht é um sistema multiusuário, ao se editar um nó, é criado um *lock* para o arquivo[4], assim nenhum outro usuário pode

[4]em inglês: *lockfile*

editar o mesmo arquivo. Ao sair do editor o *lock* é removido. Observe que este tipo de *lock* não impede que algum usuário altere o conteúdo do nó por outros meios.

EDIT_REF semelhante ao EDITOR, mas edita referência corrente. Se o arquivo da referência não existir, será criado um arquivo no diretório corrente, caso a referência (link) não possua o *pathname* completo.

VIEW permite visitar outros nós do hipertexto que não estão nas referências do nó corrente. Ao ser ativado, pede para entrar o nome de um nó – nome do arquivo. Se for encontrado ele é visualizado, caso contrário provoca erro. A execução deste comando é semelhante a inclusão e ativação de um comando link no nó corrente para o nó que se deseja visualizar.

GREP procura pela palavra que está sob o cursor nos arquivos do diretório corrente. A palavra selecionada é colocada em video reverso ou piscante. Caso a palavra exista em outros arquivos, é criado um nó em memória somente com referências aos arquivos que foram selecionados. Ao ser ativada alguma destas referências, o ht tenta posicionar o texto de tal forma que a palavra fique dentro na janela de visualização.

O nó que foi criado temporariamente com as referências a outros nós não pode ser editado. A pesquisa pela palavra é feita pelo utilitário *grep* do Unix em paralelo.

Os diretórios onde a pesquisa será feita depende da variável GREP (vide 4.2.3). Portanto os possíveis valores que a variável GREP pode assumir são:

LOCAL procura nos nós do diretório de trabalho;

GLOBAL procura nos nós de todos os diretórios especificados na variável de ambiente HT e no de trabalho;

LINK procura nos nós em que o nó corrente faz referência.

PATTERN_GREP semelhante ao GREP, mas permite ao usuário especificar uma expressão regular.

TOGGLE alterna o tamanho da janela de visualização de um nó entre o especificado no window e a tela inteira. No caso do tipo do nó ser *document*, o texto é reformatado; permitindo uma maior flexibilidade no tamanho da janela de visualização.

HELP ativa o hipertexto de auxílio *help* do **ht**. O diretório corrente não é alterado para o diretório de auxílio.

ABORT aborta execução do **ht**.

PUSH_HT	ativa referência corrente
POP_HT	retorna à referência anterior
NEXT_REF	posiciona referência seguinte como corrente
PREV_REF	posiciona referência anterior como corrente
EDITOR	edita o nó corrente no editor que foi especificado na variável EDITOR
EDIT_REF	edita referência corrente no editor que foi especificado na variável EDITOR
VIEW	visita outro nó do hipertexto que o usuário especificar
GREP	pesquisa pela palavra que está sob o cursor nos nós do diretório corrente. Se a palavra for encontrada, então é mostrado um nó com todas as referências a estes nós.
PATTERN_GREP	semelhante a GREP, porém o usuário especifica o padrão a ser pesquisado.
TOGGLE	alterna o tamanho da janela de visualização.
HELP	ativa o hipertexto de *help*
ABORT	aborta execução do **ht**

Tabela 4.2: Comandos de navegação do sistema **ht**.

4.3.3 Comandos de LaTeX

O tipo de nó *document* do ht não pretende ser um processador e visualizador completo de documentos escritos em LaTeX [Lam 86]. Simplesmente definiu-se um subconjunto reduzido de comandos do LaTeX que o **ht** é capaz de reconhecer e apresentar em um terminal alfanumérico.

As tabelas 4.3 e 4.4 apresentam, respectivamente os comandos e os ambientes que o **ht** reconhece.

\section	\subsection	\subsubsection
\paragraph	\subparagraph	\bf
\rm	\em	\item
\tt	\sf	\sl
\sc	\it	

Tabela 4.3: Comandos de LaTeX reconhecidos pelo **ht**.

document	itemize	description	enumerate	center

Tabela 4.4: Ambientes LaTeX reconhecidos pelo **ht**.

4.4 Utilização

O sistema **ht** é chamado pelo comando **ht**, com o seguinte formato:

ht [-L # -R # -h | -t (-p | -b # (-w|-d) #)] *arquivo1* [*arquivo2* ...]

Os *arquivos* são visualizados como nós de hipertexto seqüencialmente de acordo com seus comandos e as opções da linha de comando:

- -L #: atribui o valor numérico inteiro # à variável LINE do primeiro nó (*arquivo1*). Este argumento é semelhante à inclusão de um comando \setvar {LINE}{#} no *arquivo1*.
- -R #: atribui o valor numérico inteiro # à variável REFERENCE do primeiro nó (*arquivo1*). Este argumento é semelhante à inclusão de um comando \setvar {REFERENCE}{#} no *arquivo1*.
- -h: antes da visualização dos nós, é apresentado o hipertexto de *help* dos comandos de movimentação e navegação do **ht**. O nó de *help* apresentado depende do editor de texto definido na variável EDITOR.
- -t : modo de depuração. Gera na saída padrão a linearização dos nós e suas referências recursivamente, como as opções -d e -w. Contudo, em vez de gerar o conteúdo do nó, somente inclui o nome do nó (arquivo) seguido de todos os seus comandos na saída padrão.

- -b #: especifica a partir de qual nível no hipertexto será feita a linearização a contar de *arquivos* (*default* b=1).
- -d #: gera na saída padrão a linearização dos nós *arquivos* e suas referências seqüencialmente e recursivamente. A profundidade nas referências é dado por #.
- -w #: idem a opção -d, porém a linearização é feita em largura e não em profundidade nas referências. O # representa o número de níveis.
- -p : gera na saída padrão a linearização dos nós *arquivos* seguindo o *path* especificado no comando link de cada nó. O processo termina quando um nó não possuir a especificação do *path*.

Observe que as opções -L e -R são válidas somente para o primeiro nó e não para os demais. São opções úteis para outros programas ativarem o ht sem ter que editar e manipular o conteúdo dos nós. O processo de linearização gera somente uma vez um mesmo nó na saída padrão.

4.5 Exemplos de Aplicações

A seguir são apresentados dois exemplos de utilização do sistema ht. O primeiro é a edição de artigos e a segunda é o processo de especificação de um projeto em grupo.

4.5.1 Edição de Artigos

Uma aplicação clássica do sistema ht é a edição e geração automática de artigos para impressão. Primeiramente, define-se a estrutura básica do artigo, como por exemplo seções e sub-seções. O passo seguinte é especificar as ligações entre os nós para realmente criar o hipertexto do artigo. O tipo dos nós pode ser, por exemplo, *document* a fim de permitir posterior linearização da estrutura do artigo para impressão. Os textos dos nós devem conter comandos em LaTeX e comandos do ht, embora estes últimos sejam filtrados no processo de linearização. É conveniente definir o *path* para auxiliar na leitura do hiperdocumento.

Para facilidade de edição do documento pode-se criar um nó raiz, um *meta-nó*, que contém referências para todos os nós que compõe o artigo. A

estrutura interna deste meta-nó, a fim de melhorar sua visualização e edição, deve conter na primeira coluna referências ao primeiro nível de divisão do artigo, por exemplo para seções. Nas colunas seguintes, abaixo de cada seção, referências ao próximo nível, por exemplo as sub-seções, até chegar a nós indivisíveis. É importante destacar que cada nó, não necessariamente, deve conter uma seção, podendo esta estar dividida em vários nós ligados, ou também várias seções em um único nó.

A figura 4.1 apresenta uma artigo com 3 seções, 4 sub-seções e o meta-nó. A aparência deste meta-nó, tanto em um editor de texto e como visualizado pelo ht, é apresentada na figura 4.2.

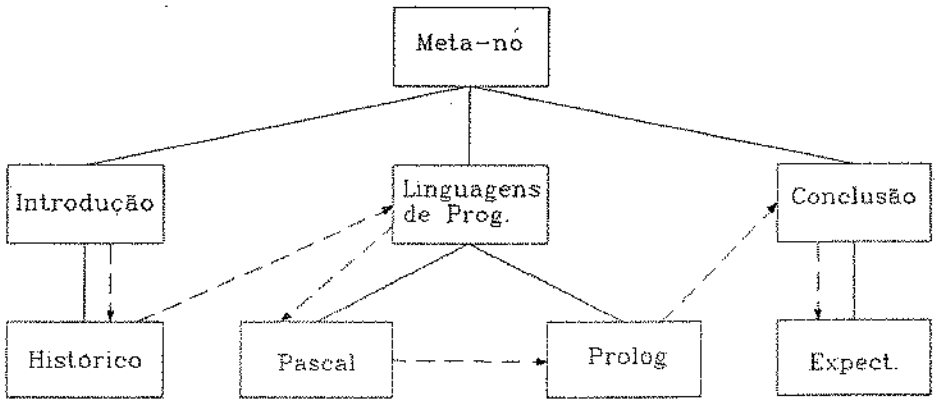


Figura 4.1: Hipertexto que representa a estrutura de um artigo. Todas as linhas representam as ligações e os vetores tracejados indicam o *path*.

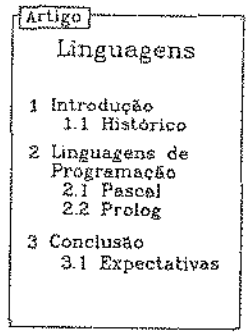
A linearização deste artigo no ht pode ser feita de duas maneiras:

- *ht -p introdução* : lineariza a partir do nó introdução seguindo o *path* de vetores tracejados.
- *ht -b 2 -d 2 meta-nó* : lineariza a partir do segundo nível a contar do meta-nó dois níveis de referências em profundidade.

A seguir são apresentados algumas vantagens do uso do ht na edição de artigos:

1. abstração de níveis de detalhe (*outline*): o usuário pode visualizar o documento como quiser, como por exemplo ver somente as seções e subseções;

Formatado



Não Formatado

```
\header {"Artigo", document, normal}
\window {0,0,14,40}
        Linguagens

\link {intro.doc, "1 Introdução", bold}
\link {histo.doc, "     1.1 Histórico", bold}

\link {lingu.doc, "2 Linguagens de Programação", bold}
\link {pascal.doc,"     2.1 Pascal", bold}
\link {prolog.doc,"     2.2 Prolog", bold}

\link {concl.doc, "3 Conclusões",bold}
\link {expec.doc, "     3.1 Expectativas}
```

Figura 4.2: O texto à direita é o conteúdo de um nó visto num editor de texto, e a esquerda como é visualizado no ht.

2. múltiplos autores: durante a autoria de um mesmo artigo, várias pessoas podem estar editando partes diferentes dele;

3. facilidade de manipulação: tando o autor como o leitor podem percorrer o documento da forma achar mais adequada.

4.5.2 Especificação de Projetos em Grupo

A especificação de um projeto consiste de um conjunto de regras que são obtidas a partir da interação das diversas pessoas envolvidas. Esta interação dá-se normalmente via reuniões e correspondência eletrônica (*e-mail*), que ficam esparsas em folhas, anotações e arquivos. A conseqüência é a perda de informações que podem ser relevantes ao projeto, por não estarem reunidas em um único documento. A dispersão também dificulta a manutenção da coerência e sincronização das comunicações.

Toda especificação pode ser feita em hipertexto com a aplicação do método IBIS [Beg 88] usando o sistema **ht**. O chefe do projeto apresenta uma proposta inicial para discussão em um nó do hipertexto. Os demais componentes acrescentam sugestões em novos nós que estão ligados ao inicial ou a outras sugestões já apresentadas. Ao final da discussão é possível analisar a evolução do problema temporalmente e verificar seu estado atual. Este processo é basicamente um refinamento a partir da proposta inicial.

Observe que ao final da discussão podem existir vários nós folhas no

hipertexto que não são a solução ideal mas podem ter soluções particulares que podem ser incorporadas à proposta final, conforme figura 4.3.

O método apresentado deixa a cargo do chefe de projeto definir a terminologia a ser adotada, o conteúdo de cada nó e o tipo de ligação. Assim, o **ht** se adapta à técnica e não a técnica ao **ht**. É conveniente que o tipo do nó seja *document* para permitir posterior impressão no formato LaTeX. Importante destacar que atualmente o **ht** não possui todos os requisitos[5] para suportar o método IBIS.

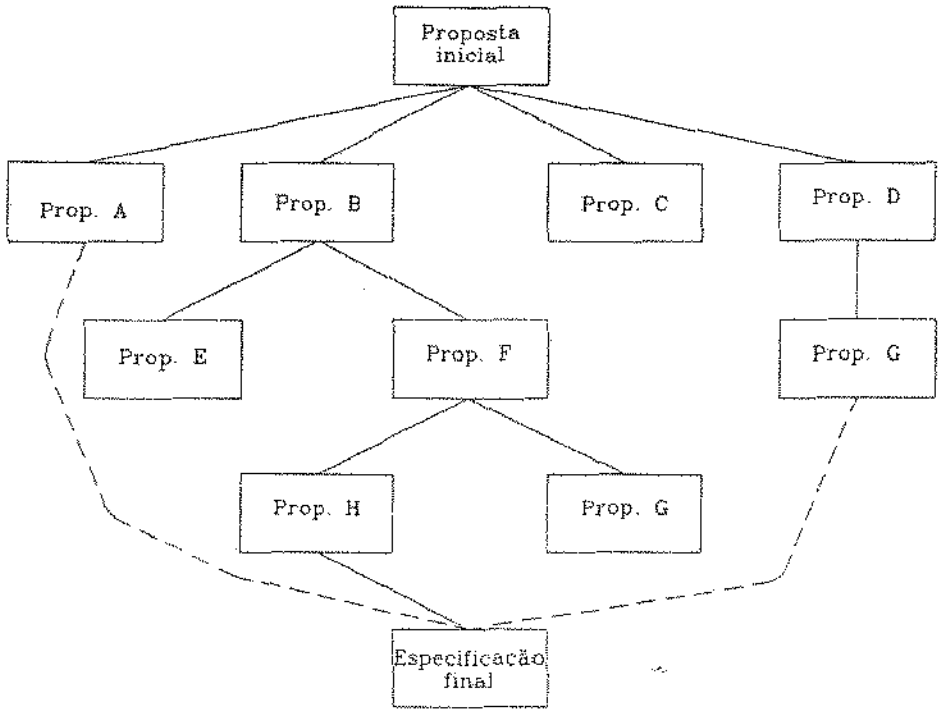


Figura 4.3: Processo de especificação de um projeto em hipertexto a partir de um nó inicial. A especificação final é um conjunto das diversas soluções.

4.6 ht como ferramenta de Desenvolvimento

Hipertexto, por definição, é a técnica de relacionar informações em uma rede de tal forma que ao recuperar algum dado, dados afins também possam ser

[5] tipo de nó e tipo de ligação

recuperados facilmente. O modo como as informações estão estruturadas no hipertexto depende de seu conteúdo e do usuário. Pode-se observar então que hipertexto tem grande capacidade de integração, estruturação e recuperação de informações relacionadas.

Os ambientes de desenvolvimento de *software*, conforme capítulo 3, carecem de uma ferramenta para especificação, integração de módulos e geração automática da documentação. O conceito de hipertexto vem exatamente de encontro a esta lacuna para prover principalmente a integração do programa e ainda, o sistema **ht** fornece facilidades para geração automática da documentação a partir de um hipertexto, no formato LaTeX para impressão. Além disto, o sistema **ht** é capaz de estruturar e manter uma biblioteca de *software* com, documentação associada, a todos usuários.

4.6.1 Especificação do Projeto

A especificação do projeto pode ser feita em hipertexto, usando por exemplo o método apresentado da seção 4.5.2. Entretando, é conveniente incorporar a especificação final no hipertexto do projeto, mesmo que seja um nó, para que todos os componentes do projeto tenham acesso a ela. Isto facilita a compreensão e documentação do projeto e futuras alterações que venham a ocorrer no programa.

4.6.2 Modularização

Uma vez de posse da especificação, é necessário dividir o trabalho em unidades funcionais, chamados módulos, que apresentem um mínimo de intercomunicação. Para cada módulo, executa-se um processo de refinamento até chegar a módulos que não serão mais divididos. Neste ponto, pode-se verificar se existem módulos que apresentam comportamentos semelhantes e que possam ser aglutinados de forma a eliminar componentes redundantes. Deve-se também procurar, na biblioteca de *software* disponível por módulos que apresentem características funcionais semelhantes aos definidos e que possam ser utilizados satisfatoriamente. É possível, dependendo da diversificação e porte da biblioteca, evitar o desenvolvimento de grande parte dos módulos, restando apenas os de gerenciamento.

Uma vez estando o programa dividido em módulos que se interagem, pode-se associar cada módulo do programa a um nó de hipertexto. Esta associação é natural e ocorre sem problemas no **ht**. A modularização tem

por objetivo abstrair partes do programa para seu melhor desenvolvimento, re-aproveitamento e manutenção.

4.6.3 Estruturação

Com o programa dividido em módulos e definida a melhor maneira como eles se relacionam, cria-se uma estrutura hierárquica de vários níveis. Este conceito de estrutura de programa é isomorfo a hipertexto. Um módulo de programa é visto no hipertexto como um nó. O uso de um módulo por outros é indicado por uma ligação, caracterizando as idéias de dependência e importação.

Estruturalmente, a hierarquia de um programa é representada no hipertexto por *ligações*. O módulo principal faz referências a sub-módulos e assim por diante até chegar a módulos básicos que podem ser compartilhados por vários outros módulos ou programas. É importante destacar que qualquer alteração em um nó é transparente aos demais que o referenciam. O A_HAND define estrutura de um programa como modular e hierárquico, o que reforça a usabilidade do ht para este tipo de aplicação.

4.6.4 Documentação

A documentação é feita em paralelo com o desenvolvimento do programa, criando-se uma estrutura hierárquica isomorfa conforme figura 4.4. Cada módulo, que está em um nó do ht, está ligado a um nó de documentação. Assim, para se obter a documentação de qualquer módulo basta seguir a referência de documentação. É conveniente que o tipo do nó de documentação seja *document* e que ele possua ligações *default* para definição do *path*. Pode-se também associar regras aos módulos, como por exemplo, um módulo somente será liberado para uso, se sua documentação existir. É importante destacar que a documentação final de um programa é obtida seguindo o hipertexto de documentação já criado.

As ligações entre as estruturas de especificação, programa e documentação podem facilmente ser automatizadas sem a necessidade dos *links* explícitos. Esta e outras extensões são analisadas na seção 4.8 com a utilização de modos de operação.

Note que sistemas como o WEB [Knu 86] se restringem a uma ferramenta de desenvolvimento e documentação de módulos sem entrar no mérito da

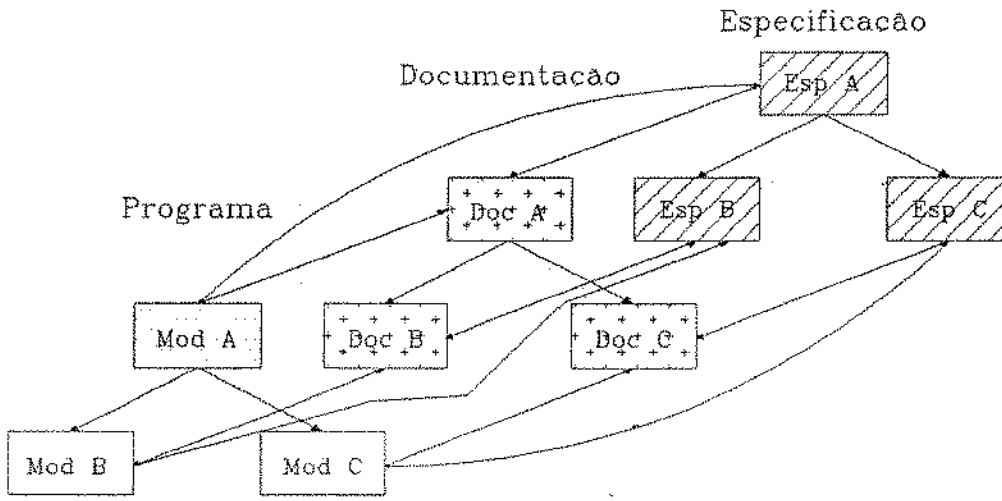


Figura 4.4: Grafo que representa a estrutura de um hipertexto, com nove nós, três de especificação, três de programa e três de documentação. A utilização dos módulos B e C pelo módulo A é representada por arestas incidentes a eles, que são as ligações. A documentação dos módulos (A, B, e C) é feita por uma estrutura paralela idêntica a do programa, com uma aresta (ligação) unindo cada módulo com sua respectiva documentação. O mesmo ocorre com a especificação

especificaõ e estrutura do programa. Algumas linguagens de programação, como Módula-2 e ADA permitem estruturar programas hierarquicamente porém não provêm uma ferramenta adequada para documentação. Outras, deixam a estrutura e a documentação do programa totalmente a cargo do programador como PASCAL e C, dificultando em muito o desenvolvimento de programas grandes. O ht além de viabilizar a estruturação e documentação de programas, é totalmente independente da linguagem de programação.

4.6.5 Exemplo de uma Aplicação

Suponha que se deseje desenvolver um programa, chamado de FF, que calcule, a partir de uma opção de entrada, o fatorial ou o número de Fibonacci de um dado número inteiro positivo. Como se pode esperar o programa é composto de um módulo gerenciador e dois outros módulos que realizam os cálculos. Para facilidade de programação define-se um meta-nó, chamado simplesmente de menu. O menu conterá referências aos arquivos fontes, ao executável e configura uma tecla, conforme a figura 4.5.

```
\header {"Fatorial & Fibonacci", text, normal}
\window {0, 0, 24, 40}
                Menu
\link {ff.c, "Main", bold}
    \link {fatorial.c, "Fatorial", bold}
    \link {fibonacci.c, "Fibonacci", bold}
\link {command, ff, "ff", bold, 40, 0, 24, 40}
\setvar {'m'} {"make"}
```

Figura 4.5: Meta-nó do programa FF (Fatorial-Fibonacci).

Pode-se verificar na figura 4.6 a estrutura do programa: o módulo principal (FF) que utiliza dois sub-módulos (**fatorial** e **fibonacci**). A última referência é acrescentada para permitir a execução do programa sem ter que sair do ht. Observe que é uma ligação que define o tipo do nó referenciado como *command* que será executado em uma janela. O último comando, **setvar**, associa a palavra "make" a tecla 'm', que quando acionada cria uma

shell para execução do comando.

O comando **link** do comando **ff** poderia perfeitamente ser substituído por um **setvar** {'f'}{"ff"}, sendo assim necessário pressionar apenas uma tecla (f). Entretanto, a entrada e saída é feita em toda a tela e não em uma janela que ocupa a metade direita da tela. Outra desvantagem é o atraso no tempo de execução do programa, uma vez que há uma indireção para *shell*. Analogamente, o comando **setvar** também poderia ser substituído por um **link**.

A documentação tem uma estrutura isomorfa à do programa. Da mesma forma, na documentação teria um meta-nó próprio que faz referências a seus nós de documentação conforme figura 4.7.

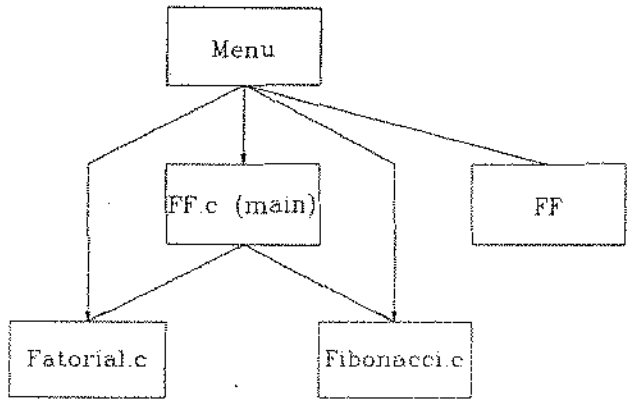


Figura 4.6: Estrutura do programa Fatorial-Fibonacci.

Suponha agora, que outro programa deseja utilizar a função fatorial. Para isto, é necessário apenas que o programa referencie o módulo fatorial; assim, já tem acesso à sua documentação. Observe que os módulos são independentes dos projetos que fazem parte e não tem conhecimento de quem os referencia.

Finalmente, pode-se gerar a documentação dos módulos para impressão ou catalogá-los em uma biblioteca de *software* a fim de torná-los públicos. Seria conveniente existir também um hipertexto de projetos que referenciam os meta-nós de todos os projetos assistidos pelo **ht**.

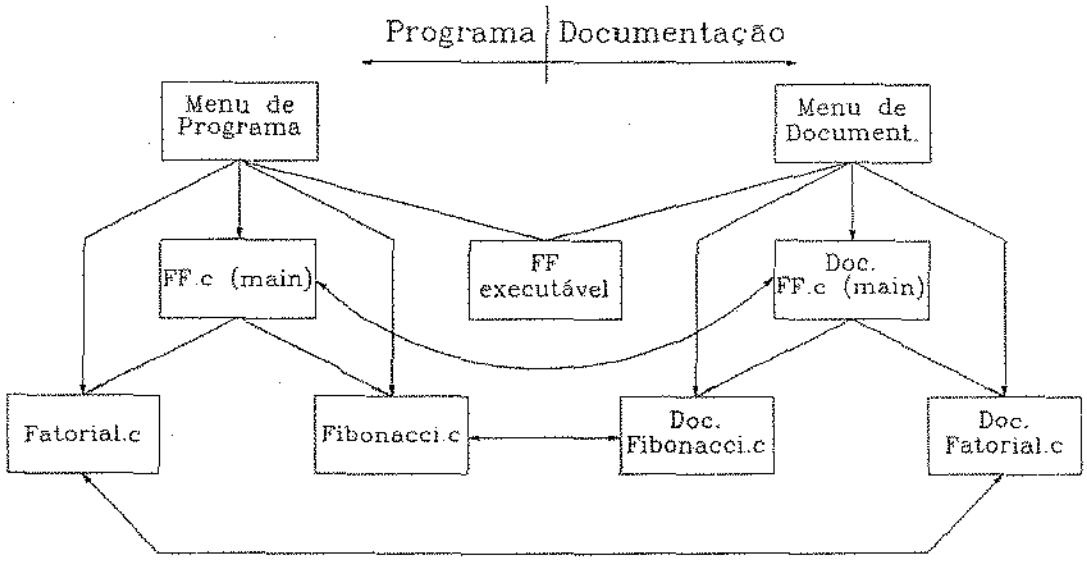


Figura 4.7: Estrutura geral do programa Fatorial-Fibonacci. A porção da esquerda representa o programa e a da direita a sua documentação.

4.7 Comparação com Outros Sistemas de Hipertexto

A melhor maneira de apresentar uma comparação com os sistemas apresentados na seção 2 com o ht é analisar as tabelas 2.1 e 2.2 com as tabelas 4.5 e 4.6.

	Hierárquico	Grafo	Ligações com tipo	Atributos	*Path*	Versões	Ações	Visões
ht	Unix file sys.	Sim	Não	Sim	Sim	Não	Sim	Multilink

Tabela 4.5: Recursos do Sistema ht (primeira parte).

A versão atual do ht não implementa tipos de ligação. É conveniente primeiro especificar os tipos necessários para uma boa utilização no desenvolvimento de *software*. Entretanto deve-se ter o cuidado de não restringir o uso do ht a estas aplicações, porque acima de tudo ele é um manipulador

	Pesquisa	Editor de texto	Multi-usuário	Figuras gráficos	*Browser* gráfico	Indep. de terminal
ht	Unix/grep	vi/emacs	Sim	Não	Não	Sim

Tabela 4.6: Recursos do Sistema ht (segunda parte).

de hipertextos.

Um controle de versões pode ser incorporado ao ht, como por exemplo o SCCS[6] disponível no Unix System V. Quando um nó é acessado, o sistema é encarregado de localizar e devolver a versão adequada de acordo com o usuário que o referenciou. Outro controle de versões que pode ser incorporado é definido no [Vic 89] para o ambiente A_HAND. Cada nó pode estar em um dos três estados: particular, experimental ou liberado. Novamente, de acordo com o usuário que acessar o nó, o sistema devolverá a versão adequada. Observe que ambos os sistemas de controle de versão estão associados a nós e não as ligações.

4.8 Extensões

A operacionalidade do ht para o desenvolvimento de *software* e outras tarefas pode ser estendida de muitas maneiras. Os itens que seguem apresentam algumas novas sugestões para ampliar o uso do ht.

4.8.1 Mail - "Correio Eletrônico"

O correio eletrônico, comumente chamado de *mail*, disponível atualmente no sistema Unix é dito linear, envia e recebe mensagens seqüenciais. Suponha então que se deseja enviar um programa modular e hierárquico. Usualmente, o usuário enviaria cada módulo (arquivo) isoladamente e um texto descrevendo sua estrutura. Uma segunda maneira é agrupá-los convenientemente em um único arquivo e enviá-lo. Na recepção o usuário separaria a mensagem em diversos arquivos. Observe que ambos os métodos são ineficientes, o primeiro por enviar um grande número de mensagens, e o segundo pela interferência do usuário em agrupar e separar os módulos.

[6] Source Code Control System

A maneira ideal é enviar uma mensagem na forma de um hipertexto. Inicialmente, o programa está estruturado em um hipertexto. Para enviá-lo por um *mail*, o sistema automaticamente linearizaria o programa, que na forma de um hipertexto e o transmitiria. Na recepção, quando o *mail* reconhece uma mensagem na forma de um hipertexto, ele chamaria o ht que recuperaria o estrutura original do programa. Observe que o usuário não precisa se preocupar com a transmissão da estrutura do programa, porque ela já está embutida no hipertexto. Para otimizar a transmissão, o ht também poderia compactar a mensagem.

4.8.2 News

O *news* é um utilitário desenvolvido na Universidade de Berkeley para divulgação e controle de notícias (novas informações) para os usuários do sistema. As notícias são informações novas de interesse, como por exemplo livros e periódicos recebidos e novos utilitários do sistema.

O ht pode exercer a função de *news* com pouquíssimas alterações. É necessário que se crie um nó cujo conteúdo é uma lista de referências as notícias, chamado por exemplo de *news.ht*. Se a referência (notícia) ainda não foi lida, ela aparece de forma destacada (*bold*), caso contrário aparece com pouca intensidade (*normal* ou *dim*). Quando se ativa uma referência, a notícia associada ao nó é apresentada e o tipo de letra da referência passa de *bold* para *normal*. Esta troca na letra teria que ser feita pelo ht. Por fim, é necessário desenvolver uma ferramenta para gerenciar o *news.ht*, que inclua e remova referências (notícias) nele.

4.8.3 Browser Estrutural Gráfico

O *browser* gráfico é uma ferramenta que constrói graficamente um mapa de um hipertexto na forma de um grafo. Os vértices representam os nós, e as arestas são as ligações. Como o ht é uma ferramenta que se propõe a ser independente de terminal, a manipulação de figuras fica restrita aos recursos oferecidos pelos terminais, normalmente alfanuméricos. Mesmo que eles ofereçam recursos gráficos, o tempo gasto para transmitir uma tela por um canal serial a uma velociadade de 9600 bps é muito grande.

Entretanto, pode-se desenvolver um *browser* alfanumérico com restrições. Evidentemente que a visualização de todo o hipertexto, dependendo do seu tamanho, seja inviável. Porém, para orientação do usuário, pode-se criar

um mapa com os nós mais próximos do nó que está sendo visualizado.

4.8.4 Controle de Versões

Como o ht se propõe a ser uma ferramenta de desenvolvimento de *software* é desejável que ele tenha um mecanismo para o controle de versões. O tipo de controle é irrelevante (vide seção 4.7), o importante é que esteja associada ao nó e não a ligação. Assim, qualquer usuário que desejar utilizar um módulo de programa fará apenas uma referência ao nó. Desta forma, ao acessar o módulo (nó), o sistema se encarrega de devolver a versão adequada ao usuário.

4.8.5 Modos de Operação

Modos de Operação são operações pré-definidas que o ht oferece conforme o que está sendo manipulado. Um modo essencial para o desenvolvimento de programa em 'C' seria o CMODE[7]. Com este modo ativado, o sistema reconheceria programas em 'C' definindo algumas ligações implícitas:

- todo o comando *#include* é uma referência;
- cada definição de função é um rótulo (\mark) e cada chamada de função é uma referência à sua definição, semelhante a *tags*[8].

O modo CMODE também poderia gerar, a partir do módulo principal, o *makefile* do programa, uma vez que a estrutura do hipertexto define as dependências de cada módulo.

Outro modo útil seria o LatexMODE. Sua função seria na linearização de um hipertexto. Ele incluiria automaticamente o cabeçalho, como por exemplo \documentstyle e \begin{document}, e o final do documento, como por exemplo o \end{document}. Eventualmente poderia gerar uma capa.

4.8.6 Editor de Hipertexto

Atualmente, toda criação e modificação de um nó e da estrutura de um hipertexto é feita por um editor de texto comum (*vi* ou *emacs*). Por conse-

[7] semelhante ao emacs

[8] Um arquivo gerado pelo comando *ctags* que contem a informação em que módulos as funções estão declaradas

guinte, o usuário deve saber a sintaxe dos comandos e a maneira correta de manipulá-los. O ideal é ter um editor de hipertexto que permita ao usuário ver o conteúdo de um nó numa janela, e realizar alterações sem ter que chamar o editor de texto. Por exemplo, tornar uma palavra qualquer do texto uma referência, simplesmente apontando-a e indicando a qual nó se liga.

Capítulo 5

Implementação do ht

Este capítulo aborda alguns tópicos do desenvolvimento e implementação da primeira versão do ht. Descreve os dois primeiros protótipos, técnicas utilizadas, problemas e soluções encontradas.

5.1 Introdução

Há muito tempo vêm-se desenvolvendo sistemas que manipulam hipertextos. A seção 2.6 apresenta a descrição funcional suscinta de diversos sistemas desenvolvidos, mas para informações mais detalhadas, os artigos de origem são ótimas referências. Os artigos [Ges 90] e [Kin 90] descrevem a implementação de dois sistemas simples de hipertextos. Junto com os artigos estão também as listagens dos sistemas, respectivamente em Pascal e 'C', para IBM-PC. O artigo [Tom 89] descreve um modelo de dados e uma implementação para armazenar hipertextos.

O ht propriamente dito, no seu desenvolvimento, passou por duas fases bem caracterizadas. A primeira foi a especificação e a implementação de um protótipo, chamado protótipo I, e segunda foi uma fase de testes com o protótipo, a re-especificação e a implementação do protótipo II e sua documentação. O ht foi implementado em 'C' no sistema operacional Unix e, atualmente, já está instalado e disponível o protótipo II do ht no DCC[1] da Unicamp no *Digirede*, na *InterPro 220* e na *SparcStation 1+*.

Um ponto importante na segunda fase de desenvolvimento do ht foi a uti-

[1]Departamento de Ciências da Computação

lização do primeiro protótipo como ferramenta de trabalho. Isto foi possível porque os comandos não sofreram qualquer tipo de alteração funcional na re-especificação. Assim, a nova versão foi desenvolvida de forma totalmente modular e hierárquica como um hipertexto no **ht**.

5.2 Módulos de Programa

Estruturalmente, pode-se dividir o **ht** conforme a figure 5.1. O módulo do nível superior é responsável pela seleção da função a ser executada pelo **ht**. O segundo nível apresenta os módulos que desempenham as funções de linearizador e manipulador de hipertexto. O último nível é composto de módulos que oferecem funções básicas como, por exemplo, a visualização de nós, recuperação de erros, localização de arquivos e gerência de memória. Observe que dois módulos do terceiro nível são compartilhados pelo segundo nível.

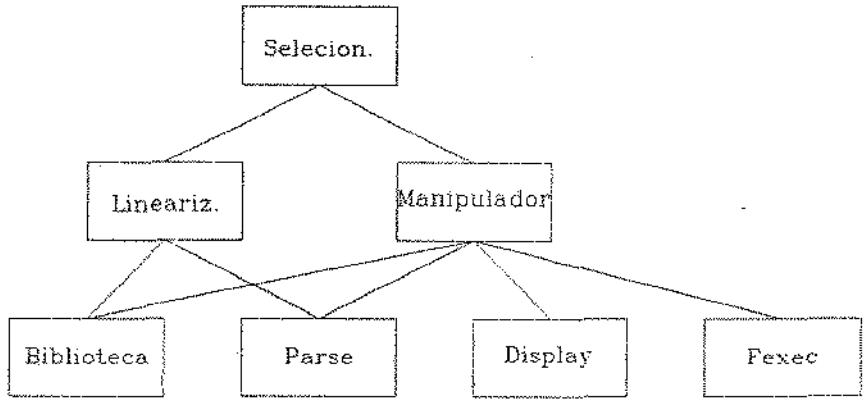


Figura 5.1: Estrutura do **ht**.

5.2.1 Selecionador

É um módulo simples, encarregado de analisar a linha de comando. Conforme os argumentos, ele chama o módulo *linearizador* ou o *manipulador* de hipertexto.

As opções -L e -R da linha de comando são úteis para que outros programas ativem o **ht** posicionando uma linha e uma referência de um nó sem

ter que alterar o seu conteúdo. Um exemplo de aplicação que se utilizaria destas opções seria o *news* apresentado na seção 4.8.2.

5.2.2 Linearizador

A linearização de um hipertexto é o processo de juntar em um único arquivo vários nós de um hipertexto por algum método de busca pré-definido. O linearizador dispõe de três métodos de busca para linearizar um hipertexto: em profundidade, em largura e pelo *path*. Estes três métodos cobrem todos os modos de linearização usualmente desejáveis pelo usuário.

O principal objetivo da linarização é gerar a documentação de programas que foram desenvolvidos com o auxílio do **ht**. Nenhum dos sistemas apresentados na seção 2.6 dispõe desta capacidade.

A facilidade de linearização também é útil para depuração de um hipertexto. Uma das opções é exatamente gerar somente os comandos, ao invés do texto, na saída padrão. Desta forma, o usuário pode ter uma visão global de como estão definidos e estruturados os nós.

5.2.3 Manipulador

É o módulo mais aprimorado e complexo do **ht** e pode ser visto em detalhe na figura 5.2. A decisão de separar a implementação do módulo *htsearch* (comandos GREP e PATTERN_GREP) do módulo *hypertext* foi pela sua complexidade de criar o processo filho *grep* e gerenciar a entrada e saída.

O *manipulador* basicamente faz uma alternância de chamadas aos módulos *parse* e *display*, mas se o tipo do nó carregado for *command*, ao invés do *display* é chamado o *fexec*.

5.2.4 Parse

O *parse* é encarregado da leitura de um nó e sua devida conversão para uma estrutura interna. As ferramentas utilizadas foram o *lex*[2] e *yacc*[3], porque são ferramentas fáceis de usar, confiáveis, portáveis e de grande adaptabilidade. Assim, para qualquer alteração na sintaxe e semântica da linguagem de um nó, basta modificar os módulos fontes do *lex* e *yacc*.

[2] LEXical analizer

[3] Yet Another Compiler to Compiler

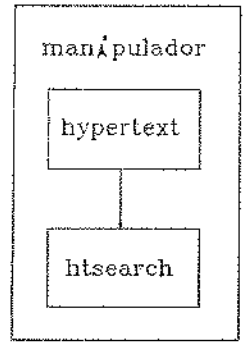


Figura 5.2: Estrutura do módulo *manipulador*.

Durante a leitura de um nó, um *parse*, se for encontrado algum erro, como por exemplo sintático, o ht se recupa e termina o *parse* apresentando todos os erros encontrados até o final do arquivo. Alguns deles são simplesmente avisos, ignorados pelo ht. Entretanto, é conveniente que estes avisos sejam removidos pelo usuário.

5.2.5 Display

É o módulo mais independente e extremamente útil ao ht e a qualquer outro programa. O *display*, chamado de WIZard window, provê uma única rotina para visualização e manipulação de textos em janelas. Ele possui comandos pré-definidos, como por exemplo subir uma linha, ir para o início do texto e busca por expressão regular. Cada um destes comandos está associado a uma tecla que depende do editor de texto definido na variável de ambiente EDITOR. Quando o *display* lê uma tecla que não possui comando associado, retorna o código da tecla a rotina que o chamou sem alterar a tela do terminal. Assim, a rotina ativadora pode tratar da função a ser executada pela tecla pressionada e retorna o controle ao *display* sem alterar o conteúdo da tela. A figura 5.3 apresenta uma rotina que utiliza o *display* para mostrar o conteúdo de uma arquivo em uma janela.

O módulo *display* é responsável pela independência de terminal oferecida pelo ht. A independência se deve a utilização da biblioteca *libcurses* disponível no Unix. A *libcurses* utiliza, por sua vez, o *terminfo* que contém a descrição dos recursos de cada tipo de terminal suportado pelo sistema.

```
1  mostra(char *nomeArquivo)
2  {
3    int jan[4]={0,0,10,70};
4    char *texto;
5   /* le o nomeArquivo na variavel texto */
6    wiz(CRIAR, texto, NULL, jan, nomeArquivo);
7    while (wiz(CONTINUAR, texto, NULL) != 'f');
8    wiz(TERMINAR);
9  }
```

Figura 5.3: Rotina lê o conteúdo do arquivo <nomeArquivo> para variável texto e o apresenta em uma janela de dimensões jan. As linhas 6 e 8, respectivamente, alocam e desalocam a estrutura associada ao texto e a janela. A linha 7 é um laço que termina quando o for teclado a letra f.

Uma nova versão do *display* poderia ser desenvolvida utilizando o *X-window* e incorporada ao ht para oferecer facilidades gráficas nos nós. Evidentemente, seu uso se restringe a terminais gráficos.

5.2.6 Fexec

O módulo *fexec* provê rotinas de criação e controle de processos. O ht quando reconhece um descritor exec ou um link com redefinição do tipo do nó para *command*, ativa este módulo. O processo criado pode ser executado livremente ou sob controle do *fexec*. A diferença básica é que no segundo, a entrada e saída do processo é feita em uma janela na tela. Embora pareça mais atraente, ela apresenta uma limitação que é o não reconhecimento, por parte do processo filho, da entrada padrão como um terminal e sim com um *pipe*. Isto limita a utilização de programas que acessam diretamente o terminal, como por exemplo o *stty* e *vi*.

Os comandos EDITOR e EDIT_REF implementados no módulo *manipulador* utilizam este módulo para criar um processo para o editor de texto. Logicamente, a interação do editor com o usuário não sofre interferência do *fexec*.

Quando um processo está executando, todos os sinais capturados pelos ht são repassados ao processo filho. Por exemplo, se o ht receber um SIGINT

(ctrl-C) [Bou 87], ele é repassado para o filho. Assim, o usuário tem todo o controle do processo filho de forma interativa.

5.2.7 Biblioteca

A *biblioteca* é um módulo com um grande número de rotinas que possuem as mais diversas funções, como por exemplo localizar um nó, alocar memória, implementar listas e pilhas e outras. Sua estrutura é apresentada na figura 5.4. Dispõe de rotinas para começar e encerrar o **ht**, gerenciamento de memória, recuperação de erros, localização de nós, configuração do teclado e funções mais básicas como manipulação de listas e pilhas. A seguir é apresentado uma descrição de todos os módulos que compõe a *biblioteca*:

htLib ativa e desativa o **ht**, localiza nós e configura o teclado;

ExpFn gerência os diretórios de acesso aos nós;

Lista mantém uma lista dos nós que foram localizados pelo módulo *htsearch* e os que já foram linearizados;

htError controle de erros;

htAlloc gerência de memória.

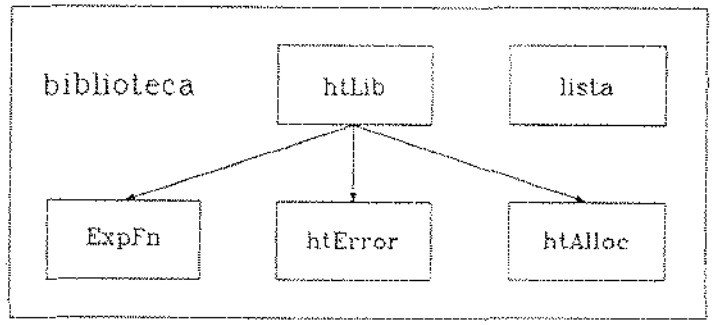


Figura 5.4: Estrutura do módulo *biblioteca*.

5.3 Fases do Desenvolvimento

Como foi dito, o ht passou por duas fases distintas no seu desenvolvimento. A primeira caracterizada pelo protótipo I e segunda pela re-especificação e o protótipo II.

5.3.1 Primeira Fase: Protótipo I

A idéia inicial foi de desenvolver uma ferramenta que manipulasse hipertextos para auxiliar o processo de desenvolvimento de *software*. Outro ponto importante era a facilidade que o sistema deveria ter para linearizar um hipertexto num único arquivo para impressão. Como a melhor ferramenta para geração de documentos é o LaTeX, o ht deveria ter comandos semelhantes ao LaTeX para facilitar esta geração e a adaptação do usuário ao ht. Além disso, porque ele é o padrão de formatador *off-line* mais utilizado em ambientes de pesquisa e desenvolvimento. Por fim, o ht deveria ser uma ferramenta de poucos comandos para que o usuário não perdesse tempo em ter que aprendê-los.

A proposta inicial era de representar um nó de um hipertexto de três formas diferentes conforme o contexto e o meio de armazenamento:

HT forma de armazenamento de um nó em disco;

HP estrutura de um nó em memória, independente da janela de visualização;

HD estrutura de um nó dependente de sua apresentação em uma janela;

De posse da especifação, partiu-se para a primeira implementação na *InterPro 220*. Ao final de implementação fez-se alguns testes e constatou-se o seguinte:

- o tempo de conversão de um formato para outro era grande;
- tempo extra gasto na compilação da forma HT para HP e para HD;
- a não utilização do *lex* e *yacc* no módulo *parse* provocou um atraso no desenvolvimento, dificuldade na alteração da sintaxe e semântica da linguagem (comandos).

- comandos fixos a teclas dificultavam a navegação do usuário que estava acostumado a trabalhar em outros editores de textos que possuiam associações diferentes. Assim, havia um tempo gasto no aprendizado do usuário para comandos do teclado do ht que exerciam funções idênticas a do editor que estava acostumado.

- a incompatibilidade de algumas funções do Digix[4] com o Unix System V[5].

O último item foi decisivo para uma nova especificação e implementação do ht.

5.3.2 Segunda Fase: Versão Final

Com base nas observações feitas no protótipo I decidiu-se re-especificar grande parte do ht. O único item que não foi alterado foram os comandos.

Primeiramente, separou-se o processo da visualização (*interface*) do ht, definindo-se assim um módulo independente chamado *display*. Foi eliminada a redundância de se ter duas estruturas internas: HP e HD. Optou-se pela utilização do *lex* e *yacc*[6] e pela adaptabilidade dos comandos do *browser* do ht conforme o editor de texto preferido pelo usuário.

Portanto, pode-se melhorar sensivelmente a operacionaliade e desempenho do ht. Embora ainda existam alguns pontos a serem aperfeiçoados, ele apresenta um ótimo desempenho na localização, visualização e navegação num hipertexto, mesmo no Digirede.

5.4 Problemas de Implementação

Durante a fase de desenvolvimento vários problemas surgiram e alguns deles interessantes que serão apresentados. Embora a solução para alguns seja trivial, estas ainda não foram implementadas.

[4] Versão do Unix System III da Digirede para seus equipamentos
[5] Unix da InterPro 220
[6] Atualmente o Bison

5.4.1 Incompatibilidade: Digix & Unix System V

Quando a implementação do protótipo I na *InterPro 220* foi concluída, decidiu-se portá-lo para o Digirede. Entretanto a incompatibilidade dos dois sistemas foi tão grande que inúmeras alterações se fizeram necessárias. Decorrido algum tempo, com o protótipo ainda não funcionando, seu código virou uma "colcha de retalhos". Esta, pode-se dizer, foi uma das principais razões para re-escrever o código do ht, mas agora não na *InterPro 220*, mas no Digirede. Assim, ao final da implementação a adaptação do ht na *InterPro 220* foi simplíssima. A razão desta incompatibilidade foi a versão diferente do sistema Unix utilizado nas duas máquinas, e a péssima implementação do próprio Unix feito pela Digirede, chamado *Digix*.

5.4.2 Gerência de Memória

Todo controle da memória dinâmica no ht é feita no módulo *htAlloc*, mas existe um problema. Se o usuário começar a navegar em um hipertexto em profundidade, sem retornar, chegará um momento que não haverá mais memória disponível. Atualmente, o sistema simplesmente aborta a execução com uma mensagem de erro.

A solução para este problema "trágico" é manter uma lista ligada dos nós visitados e que estão em memória. Cada vez que um nó é visitado, se já estiver em memória, ele simplesmente é reposicionado como primeiro da lista sem carregar do disco. Caso contrario, ele é carregado do disco e colocado como primeiro da lista. Assim, temos no final da lista, os nós mais antigos, que foram menos visitados.

Para o controle da memória, o carregador verificaria primeiro se existe espaço disponível. Se não existir, libera o espaço ocupado pelo último da lista, removendo-o. Na pilha de nós visitados, deve-se deixar uma indicação que o nó não está mais em memória. Esta indicação pode ser um descritor virtual com o mínimo de informações capaz de recuperar o estado do nó. Desta forma, quando o usuário retornar nas referências e atingir um nó virtual, este então, é recarregado e recolocado na lista. Esta solução é boa, porém pode acontecer de existirem muitos nós virtuais em memória provocando nova lotação. Para isto, a única solução é liberar um conjunto de nós virtuais em disco. Entretanto, esta possibilidade é remota.

5.4.3 Acesso Concorrente

Imagine a seguinte situação:

> Um usuário visita um nó e ativa uma referência para um segundo nó. Outro usuário, neste momento, altera o conteúdo do primeiro nó. Assim, quando o primeiro usuário retornar ao primeiro nó, ele terá uma versão diferente em memória da que está em disco.

A solução deste problema é guardar na carga de um nó a data da última alteração. Esta data é fornecida e mantida pelo sistema de arquivos do Unix. Assim, a todo retorno de uma referência é necessário fazer a comparação de datas do nó em memória da que está em disco. Caso seja diferente, o sistema deverá fazer nova carga. Observe que atualmente o ht não suporta este tipo de consistência, porém a edição concorrente de um nó é controlada.

5.5 Função-Utilitário

No decorrer do desenvolvimento do ht verificou-se a necessidade de rotinas que desempenhassem funções específicas que seriam úteis também a outros programas. Por exemplo, o módulo *display* é encarregado de simplesmente apresentar uma seqüência de caracteres em uma janela e controlar a movimentação do texto. Esta função é útil a qualquer programa que deseje uma *interface* com o usuário, orientada a janelas com visualização e navegação de textos.

Outro exemplo, é um sistema de controle de versões que tem basicamente duas rotinas, uma para abrir e outra para fechar arquivos. Ao abrir um arquivo, o sistema de controle de versões devolveria a versão adequada ao usuário que está acessando o arquivo. Da mesma forma, ao fechar, se o arquivo foi alterado, criar uma nova versão. O interessante é que se pode conectar e desconectar o sistema de controle de versões sem afetar o resto do programa. A figura 5.5 apresenta dois segmentos de código em 'C' para abrir e fechar arquivos. A chamada da função `fopen` e `fclose` com o prefixo `cv` são rotinas que devolvem a versão adequada do arquivo `nomeArquivo` ao usuário que chamou. Observe que pode-se retirar o controle versão facilmente, bastando remover o sufixo.

```
     .                                   .
FILE *fd;                           FILE *fd;
     .                                   .
fd = cvfopen(nomeArquivo, "r"); fd = fopen(nomeArquivo, "r");
     .                                   .
cvfclose(fd);                       fclose(fd);
     .                                   .
```

Figura 5.5: Função-Utilitário: controle de versões.

Capítulo 6

Conclusão

Hipertexto provou ser uma ferramenta bastante poderosa e eficiente para organizar informações complexas e disjuntas. Esta capacidade vem exatamente de encontro a itens que afetam os ambientes e as técnicas de desenvolvimento de *software*.

O sistema ht, baseado no conceito de hipertexto, pretende ser acima de tudo um sistema de hipertexto eficiente e de fácil uso para qualquer tipo de tarefa, em especial para o desenvolvimento de *software*. Por possuir uma *interface* configurável, poucos comandos e facilidade de linearização torna-se um sistema versátil, e, com a inclusão de modos de operação, principalmente o CMODE, o ht torna-se uma excelente ferramenta para o desenvolvimento de *software*.

A implementação do ht foi bastante proveitosa, principalmente pela decisão de abandonar totalmente a primeira implementação e começar uma nova. Assim, o protótipo II foi projetado e implementado baseado na experiência anterior, conseqüentemente um sistema melhor estruturado, modular e eficiente. Um ponto que auxiliou em muito o seu desenvolvimento foi por estar totalmente estruturado como num hipertexto, desta forma, o protótipo I serviu como ferramenta de trabalho para a sua implementação, provando ser bastante útil, não só para a programação mas para documentação.

Atualmente, o ht já está instalado e em uso no DCC da Unicamp por componentes do projeto A_HAND para estruturar, modularizar e documentar programas. Sua disponibilidade não se restringe aos integrantes do projeto, mas a qualquer usuário interessado em trabalhar com hipertexto. Seu

código está organizado em 20 módulos, com aproximadamente 5000 linhas.

Desenvolvendo a LegoShell

A LegoShell ([Dru 88]), é parte do projeto A_HAND, onde programas são acoplados para especificar uma computação, uma extensão do conceito de *pipe* do Unix. Seu desenvolvimento está sendo feito por três pessoas do grupo A_HAND com o auxílio do ht para a programação e documentação. Até o momento mostrou ser bastante adequando a estruturação e gerência no desenvolvimento da LegoShell.

Expectativas e Extensões

O sistema ht já desenvolvido é um núcleo bastante poderoso capaz de manipular hipertextos genéricos totalmente expandível. A possibilidade de incluir novos modos de operação permite ao usuário definir estruturas de hipertextos que o sistema reconheça, podendo executar operações pré-definidas sobre os mesmos. Os modos CMODE e LaTeXMODE são apenas alguns exemplos da versatilidade oferecida.

A edição de nós no sistema ht, atualmente, é feita utilizando um editor de texto comum (*vi* e *emacs*). Contudo, é desejável que o sistema ofereça a possibilidade de alterar o conteúdo e/ou os comandos de um nó sem ter que chamar o editor. Portanto, é necessário que o editor opere sobre janelas e seje mais um módulo de todo o sistema. Mais uma vez, o editor pode entender do modo de operação corrente e facilitar a edição do mesmo. Atualmente, um editor de texto orientado a janela já está em desenvolvimento com a *interface* e estrutura de função utilitário.

Com a disponibilidade das estações de trabalho *SUN Sparc 1+* no DCC/-IMECC/Unicamp, maiores recursos gráficos serão possíveis. Assim, é conveniente desenvolver um novo módulo *display* para permitir a criação e visualização de nós gráficos no ht, que já está sendo desenvolvido.

Enfim, são módulos a serem acrescentados no sistema ht para oferecer vantagens de extrema utilidade.

Apêndice A

Exemplo de um Programa no ht

Este apêndice apresenta o conjunto completo de módulos que compõe o exemplo Fatorial_Fibonacci descrito na seção 4.6.5.

A.1 Módulos de Programação

- Meta-nó de programação. Contém basicamente referências aos módulos de programação. O nome do arquivo é <fat_fibo.m.p>.

```
\header{"Menu Programa",text,normal}
\window{0,0,9,19}
    Menu

\link{fat_fibo.c,"Modulo Principal",bold}
   \link{fatorial.c,"Fatorial",bold}
   \link{fibonacci.c,"Fibonacci",bold}
\link{command,fat_fibo,"Execucao",bold,0,10,14,31}
\link{fat_fibo.m.d,"Menu documentacao",bold}
```

- Módulo principal do FF. Faz referências a sub-módulos e a sua documentação. O nome do arquivo é <fat_fibo.c>.

```
/*******************************************************
                    FAT_FIBO.C

Modulo: Fatorial_Fibonacci
Funcao: Calcula, segundo uma opcao do usuario,
        o fatorial ou o numero de Fibonacci.
Versao: 1.0 (maio 1990)
Autor: C. A. Polanczyk

\header{"Calcula Fatorial / Fibonacci",program,normal}
\window{20,0,9,61}
\link{fat_fibo.m.c,"Menu Programa",bold}
\link{fat_fibo.m.d,"Menu Doccumentacao",bold}
\link{fat_fibo.doc,"Documentacao",bold}
*******************************************************/
#include <stdio.h>

main()
{
 char s[10];
 do {
printf ("\nFatorial / Fibonacci\n");
    printf ("1 - Fatorial\n2 - Fibonacci\n3 - sair\nOpcao:");
scanf("%s\n",s);
    switch (*s) {
    case '1':fatorial(); /* \link{fatorial.c,"Fatorial",bold} */
             break;
    case '2':fibonacci(); /* \link{Fibonacci.c,"Fibonacci",bold} */
    case '3':break;
    } /* switch */
 } while (*s != '3');
}
```

- Módulo que implementa o cálculo do fatorial. O nome do módulo é <fatorial.c>.

```
/*************************************************
                FATORIAL.C

Modulo: fatorial
Funcao: Calcula o fatorial de um numero que
        lido da entrada padrao na saida padrao.
Bug: Se o fatorial calculado for maior que um
     inteiro, o retorno e imprevisivel.

\header{Fatorial,program,normal}
\window{0,10,14,80}
\link{fat_fibo.m.c,"Menu Programa",bold}
\link{fat_fibo.m.d,"Menu Documentacao",bold}
\link{fat_fibo.c,"Modulo Principal",bold}
\link{fatorial.doc,"Documentacao fatorial",bold}
*************************************************/
#include <stdio.h>
fatorial()
{
 char s[10];
 int  n;
 printf("Entre numero: ");
 scanf("%s\n",s);
 printf ("fatorial de %s = %u\n",s,fat(atoi(s)));
}

/*
 * \mark{fat}
 * fat: calcula fatorial de um numero.
 * Parametro: n  numero inteiro.
 * Retorno:   -1 erro,
 *            fatorial calculado de n
 */
fat(n)
int n;
{
 if (n < 0)
    return(-1); /* ERRO */
 if (n <= 1)
    return(1);
 return (n * fat(n-1));
}
```

- Módulo de programa que implementa o cálculo do número de Fibonacci. O nome do arquivo é <fibonacci.c>.

```
/*************************************************************
                        FIBONACCI.C
Modulo: Fibonacci
Funcao: Calcula o numero de fibonacci de um
        numero que e lido da entrada padrao.
        O resultado e jogado na saida padrao.
Bug: Se o numero de Fibonacci calculado exceder a
     capacidade de um inteiro, o retorno e indefinido.

\header{Fibonacci,program,normal}
\window{0,10,14,80}
\link{fat_fibo.m.c,"Menu Programa",bold}
\link{fat_fibo.m.d,"Menu Documentacao",bold}
\link{fat_fibo.c,"Modulo Principal",bold}
\link{fibonacci.doc,"Documentacao fibonacci",bold}
*************************************************************/
#include <stdio.h>
fibonacci()
{
 char s[10];
 int  n;
 printf("Entre numero: ");
 scanf("%s\n",s);
 printf ("fibonacci de %s = %u\n",s,fibo(atoi(s)));
}

/*
 * \label{fibo}
 * fibo : calcula o numero de Fibonacci
 * Parametro: n  numero inteiro
 * Retorno:   -1 erro
 *            fibonacci calculado
 */
fibo(n)
int n;
{
 if (n < 0)
    return (-1);  /* ERRO */
 if (n <= 1)
    return (n);
 return (fibo(n-1) + fibo(n-2));
}
```

A.2 Módulos de Documentação

- Meta-nó de documentação. Contém basicamente referências aos módulos de documentação. O nome do arquivo é <fat_fibo.m.d>.

```
\header{"Menu Documentacao",text,normal}
\window{60,0,8,20}
    Menu

\link{fat_fibo.doc,"Modulo Principal",bold}
   \link{fatorial.doc,"Fatorial",bold}
   \link{fibonacci.doc,"Fibonacci",bold}
\link{fat_fibo.m.c,"Menu programa",bold}
```

- Módulo de documentação do programa principal. O nome do arquivo é <fat_fibo.doc>.

```
\header{"Documentacao",text,normal}
\window{40,16,9,40}
\section{Fatorial & Fibonacci}
\link{fat_fibo.m.c,"Menu Programa",bold}
\link{fat_fibo.m.d,"Menu Documentacao",bold}
\link{fat_fibo.c,"Modulo Principal <fat_fibo.c>",bold}

  Este programa calcula, segundo opcao
de entrada o fatorial  ou o numero  de
Fibonacci. As opcoes sao:
        1 - \link{fatorial.doc,"Fatorial",bold,path};
        2 - \link{fibonacci.doc,"Fibonacci",bold};
        3 - sair;
```

A.2 Módulos de Documentação

- Meta-nó de documentação. Contém basicamente referências aos módulos de documentação. O nome do arquivo é <fat_fibo.m.d>.

```
\header{"Menu Documentacao",text,normal}
\window{60,0,8,20}
    Menu

\link{fat_fibo.doc,"Modulo Principal",bold}
   \link{fatorial.doc,"Fatorial",bold}
   \link{fibonacci.doc,"Fibonacci",bold}
\link{fat_fibo.m.c,"Menu programa",bold}
```

- Módulo de documentação do programa principal. O nome do arquivo é <fat_fibo.doc>.

```
\header{"Documentacao",text,normal}
\window{40,16,9,40}
\section{Fatorial & Fibonacci}
\link{fat_fibo.m.c,"Menu Programa",bold}
\link{fat_fibo.m.d,"Menu Documentacao",bold}
\link{fat_fibo.c,"Modulo Principal <fat_fibo.c>",bold}

  Este programa calcula, segundo opcao
de entrada o fatorial  ou o numero  de
Fibonacci. As opcoes sao:
        1 - \link{fatorial.doc,"Fatorial",bold,path};
        2 - \link{fibonacci.doc,"Fibonacci",bold};
        3 - sair;
```

- Módulo de documentação do módulo Fibonacci. O nome do arquivo é <fibonacci.doc>.

```
\header{"Documentacao do modulo Fibonacci",text,normal}
\window{20,0,16,40}
\subsection{Fibonacci}

\link{fat_fibo.m.c,"Menu Programa",bold}
\link{fat_fibo.m.d,"Menu Documentacao",bold}
\link{fat_fibo.doc,"Documento principal",bold}
\link{fibonacci.c,"fonte <Fibonacci.c>",bold}

   O modulo Fibonacci consiste de duas
funcoes:

Fibonacci: le da entrada  entrada  um numero
   e imprime o resultado obti-
   do pela ativacao da segunda
   funcao na saida padrao;

\link{fibonacci.c,fibo,"Fibo",bold}: funcao  recursiva  que calcula a
  serie de  fibonacci de um numero
  <n>. Em caso de erro retorna -1.

Obs.: Se o numero calculado exceder a
      capacidade de um inteiro, o retorno
      e indefinido.
```

- Módulo de documentação do módulo Fibonacci. O nome do arquivo é <fibonacci.doc>.

```
\header{"Documentacao do modulo Fibonacci",text,normal}
\window{20,0,16,40}
\subsection{Fibonacci}

\link{fat_fibo.m.c,"Menu Programa",bold}
\link{fat_fibo.m.d,"Menu Documentacao",bold}
\link{fat_fibo.doc,"Documento principal",bold}
\link{fibonacci.c,"fonte <Fibonacci.c>",bold}

   O modulo Fibonacci consiste de duas
funcoes:

Fibonacci: le da entrada  entrada  um numero
   e imprime o resultado obti-
   do pela ativacao da segunda
   funcao na saida padrao;

\link{fibonacci.c,fibo,"Fibo",bold}: funcao  recursiva  que calcula a
  serie de  fibonacci de um numero
  <n>. Em caso de erro retorna -1.

Obs.: Se o numero calculado exceder a
      capacidade de um inteiro, o retorno
      e indefinido.
```

uma segunda janela conforme figura A.2 Observe que não há, a princípio, nesta segunda janela, referências a outros nós.

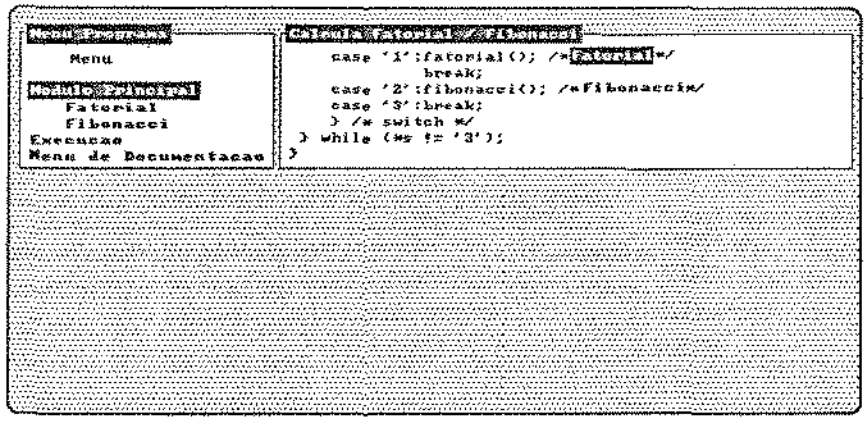


Figura A.3: Seleciona referência ao nó de implementação `Fatorial`.

Se movimentar o texto dessa janela verifica-se que existe duas referências, uma para o nó `Fatorial` (a corrente) e outra para o nó `Fibonacci` (vide figura A.3.

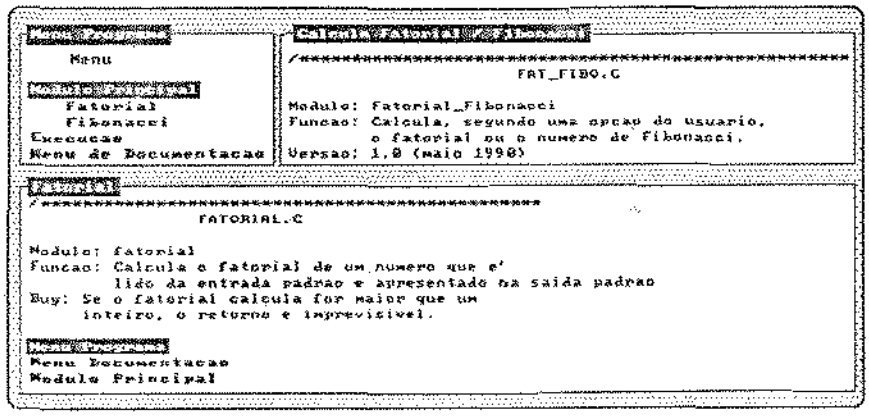


Figura A.4: Ativada referência ao nó `fatorial.c`.

Ativando a referência, o nó `fatorial` é apresentado em uma terceira janela, conforme figura A.4.

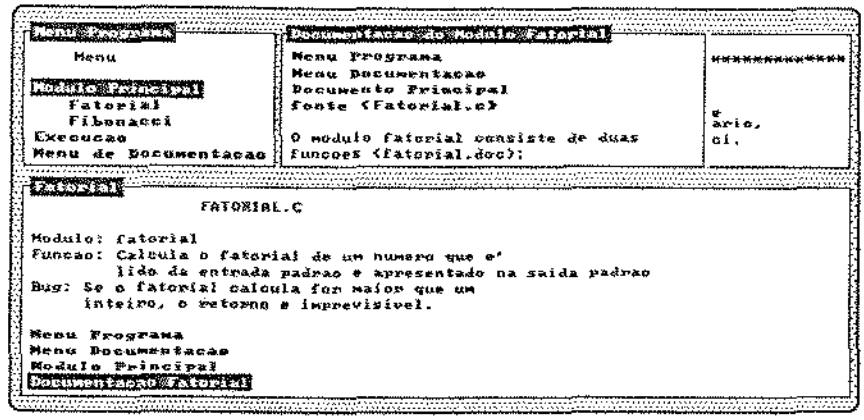


Figura A.5: Ativada referência ao seu nó de documentação.

Para visualizar a documentação deste módulo, seleciona-se a referência que leva ao nó de documentação (vide figura A.5). Observe que existe 5 referências a outros nós, e nenhuma delas é a corrente. Isto, porque a referência **continua...**, que está no final do texto, possui o atributo *path* no comando **link**.

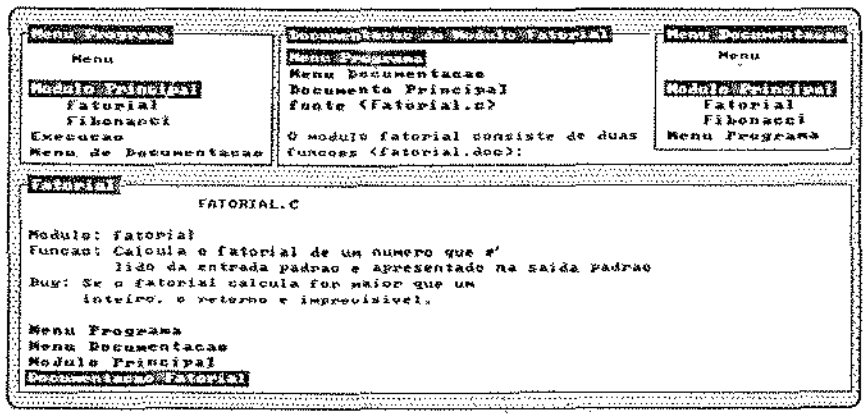


Figura A.6: Ativada referência ao meta-nó de documentação.

Assim, se desejar visualizar o meta-nó de documentação seleciona-se a referência **Menu Programa** conforme a figura A.6. Finalmente, o retorno nas referências é o processo inverso, da figura A.6 à figura A.1.

Apêndice B

Comandos do ht

A associação dos comandos do **ht** às teclas depende do editor definido na variável de ambiente EDITOR. As tabelas B.1 e B.2 apresentam as associações pré-definidas dos comandos conforme o editor.

tecla		comando	função
↑		UP	cursor para cima
↓		DN	cursor para baixo
←		LF	cursor para esquerda
→		RT	cursor para direita

vi	emacs	comando	função
k	↑p	UP	cursor para cima
j	↑n	DN	cursor para baixo
h	↑b	LF	cursor para esquerda
l	↑f	RT	cursor para direita
↑h	M-<	TOP	início do texto
G	M->	BOTTOM	fim do texto
↑f	↑v	NEXT_PAGE	página seguinte
↑b	M-v	PREV_PAGE	página anterior
↑d		SCROLL_DOWN	meia página seguinte
↑u		SCROLL_UP	meia página anterior
/	↑s	SEARCH	pesquisa por padrão no nó corrente
n	↑w	NEXT_SEARCH	próxima ocorrência do padão

Tabela B.1: Comandos de movimentação do texto.

tecla	comando	função
<enter>	PUSH_HT	ativa referência corrente
<esc>	POP_HT	retorna à ref. anterior
L	NEXT_REF	posiciona ref. seguinte como corrente
H	PREV_REF	posiciona ref. anterior como corrente
e	EDITOR	edita o nó corrente no editor que foi especificado na variável de ambiente EDITOR
r	EDIT_REF	edita ref. corrente no editor que foi especificado na var. de ambiente EDITOR
v	VIEW	visita outro nó do hipertexto que o usuário especifica
g	GREP	pesquisa pela palavra que está sob o cursor nos nós do dir. corrente. Se a palavra for encontrada, então é mostrado um nó com todas as ref. a estes nós.
↑g	PATTERN_GREP	semelhante a GREP, porém o usuário especifica o padrão a ser pesquisado.
↑t		alterna o tamanho da jan. de visualização.
?	HELP	ativa o hipertexto de *help*
q	ABORT	aborta execução do ht

Tabela B.2: Comandos de navegação no hipertexto.

Bibliografia

[Aks 88] Akscyn, R. M., McCracken, D. L. & Yoder, E. A. *"KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations"*. Communications of ACM, **31**(7):820-835, julho 1988.

[Beg 88] Begeman, M. L. & Conklin, J. *"The Right Tool for the Job"*. BYTE, **13**(10):255-266, outubro 1988.

[Big 88] Bigelow, James. *"Hypertext and CASE"*. IEEE Software, 23-27, março 1988.

[Bou 87] Bourne, Stephen R. *"The Unix System V Environment"*. Addison-Wesley, Reading, Massachussetts, 1987.

[Car 90] Carlson, David & Ram, Sudha. *"HyperIntelligence: The Next Frontier"*. Communications of ACM, **33**(3):311-321, março 1990.

[Con 87] Conklin, J. *"Hypertext: An Introduction and Survey"*. IEEE Computer, **20**(9):17-41, setembro 1987.

[Con 88] Conklin, J. & Begeman, L. *"gIBIS: A Hypertext Tool for Exploratory Policy Discussion"*. ACM Transactions on Office Information Systems, **6**(4):303-331, outubro 1988.

[Dru 87] Drummond, R. & Liesenberg, H. *"A_HAND: Ambiente de Desenvolvimento de Software Baseado em Hierarquias de Abstração em Níveis Diferenciados"*. IV Encontro de Trabalho do Projeto ETHOS, Petrópolis, RJ, abril 1987.

[Dru 88] Drummond, R. *"LegoShell: Linguagem de Computações"*. Relatório Interno do Projeto, Campinas, 1988.

[Ega 89] Egan, E. et. all. *"Formative Design Evaluation of SuperBook"*. ACM Transaction System, 7(1):30-57, janeiro 1989.

[Fid 88] Fiderio, J. *"A Grand Vision"*. BYTE, 13(10):237-244, outubro 1988.

[Fri 88] Frisse, M. *"From Text to Hypertext"*. BYTE, 13(10):247-253, outubro 1988.

[Gar 88] Garg, Pankaj K. *"Abstract Mechanisms in Hypertext"*. Communications of the ACM, *bf 31*(7):862-869, julho 1988.

[Ges 90] Gerssner, Rick. *"Building a Hypertext System"*. Dr. Dobb's Journal, (165):22-33, junho 1990.

[Lam 86] Lamport, Leslie. "LaTeX: *A Document Preparation System*". Addison-Wesley Publishing Company, Reading, Massachussetts, 1986.

[Lim 89] Lima, Maria J. D. *"Hipertexto e Suas Aplicações"*. Projeto de Fim de Curso de Matemática (modalidade Informática), UFRJ, maio 1989.

[Ker 86] Kernighan, Brian & Ritchie, Dennis. *" C - A Linguagem de Programação"*. Campus, Rio de Janeiro, 1986.

[Kin 90] King, Todd. *"A Self-Referential Hypertext Engine"*. Dr. Dobb's Journal, (165):34-39, junho 1990.

[Knu 86] Knuth, D. "TeX: *The Program*". Addison-Wesley Publishing Company, Reading, Massachussetts, 1986.

[Mar 88] Marchionini, G & Shneiderman, B. *"Finding Facts vs. Browsing Knowledge in Hypertext Systems"*. IEEE Computer, 21(1):70-79, janeiro 1988.

[Mei 89] Meira, S. et. all. *"Hipertexto: O projeto do Sistema H"*. III Simpósio Brasileiro de Engenharia de Software, Recife, 1989 - *Anais*.

[Mir 90] Miranda, Maria & Menezes, Mônica. em "Hipertextos Aplicados à Engenharia de Software". Projeto de Fim de Curso de Matemática (modalidade Informática), UFRJ, agosto 1990.

[Nie 90] Nielsen, J. *"The Art of Navegating through Hypertext"*. Communications of ACM, **33**(3):296-310, março 1990.

[Sto 89] Stotts, P. & Furuta, R. *"Petri-Net-Based Hypertext: Document Structure with Browsing Semantics"*, ACM Transactions on Information Systems, **7**(1):3-29, janeiro 1989.

[Tom 89] Tompa F. *"A Data Model for Flexible Hypertext Database Systems"*, ACM Transactions on Information Systems, **7**(1):85-100, janeiro 1989.

[Tri 88] Trigg, R. *"Guided Tours and Tabletops: Tools for Communicating in a Hypertext Environment"*, ACM Transactions on Office Information Systems, **6**(4):398-414, outubro 1988.

[Utt 89] Utting K. & Yankelovich N. *"Context and Orientation in Hipermedia Networks"*, ACM Transaction on information Systems, **7**(1):58-84, janeiro 1989.

[Vic 89] Victorelli, E. et all. *"Mecanismo de Gerenciamento de Versões e Configurações do A_HAND"*. DCC-UNICAMP, agosto 1989.

[Yan 88] Yankelovich, N. et all. *"Intermedia: The Concept and Construction of a Seamless Information Environment"*. IEEE Computer, **21**(1):81-96, janeiro 1988.