UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Visualização de Informações Aplicada à Gerência de *Software*

por

CRISTINA CEMIN

Dissertação submetida à avaliação,
como requisito parcial para obtenção do grau de Mestre
em Ciência da Computação

Prof[a]. Dr[a]. Carla Maria Dal Sasso Freitas
Orientadora

Porto Alegre, setembro de 2001.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Cemin, Cristina

Visualização de Informações Aplicada à Gerência de Software/ por Cristina Cemin. – Porto Alegre:PPGC da UFRGS, 2001.
70f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR – RS, 2001. Orientadora: Freitas, Carla Maria Dal Sasso.

1. Visualização de Informações 2. Visualização de Software 3. Computação Gráfica 4. Compreensão de Programas 5. Visualização de grafos. I. Freitas, Carla Maria Dal Sasso. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitoria: Profa. Wrana Maria Panizzi
Pró-Reitor de Ensino: Prof. José Carlos Ferraz Hennemann
Pró-Reitor Adjunto de Pós-Graduação: Prof. Philippe Olivier Navaux
Diretor do Instituto de Informática: Prof. Philippe Olivier Navaux
Coordenador do PPGC: Prof. Carlos Alberto Heuser
Bibliotecária – Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

Sumário

Lista de Figuras **5**
Lista de Tabelas **6**
Resumo **7**
Abstract **8**
1 Introdução **9**
2 Visualização de Informações **12**
2.1 Introdução **12**
2.2 Processo de Visualização **13**
2.3 Técnicas de Visualização **13**
2.3.1 Visualização de textos 14
2.3.2 Visualização de valores diversos 15
2.3.3 Visualização de estrutura e relacionamentos 16
3 Visualização de *Software* **19**
3.1.Introdução **19**
3.1.1 Captura da Informação 22
3.1.2 Representação da Informação 23
3.1.3 Apresentação da Informação 24
3.2 Exemplos de Ferramentas de Visualização de S*oftware* **26**
3.2.1 BALSA [BRO84] 26
3.2.2 Zeus [PRI94] 26
3.2.3 TANGO [STA90] 27
3.2.4 Visual Insights [FYO97] 27
3.3 Recuperação e Extração de Informações **28**
3.4 Requisitos de uma Ferramenta de Visualização de S*oftware* **29**
3.5 Comentários Finais **31**
4 Estudo de Caso **33**
4.1 Introdução **33**
4.2 Características gerais da linguagem *Progress* **34**
4.3 Gestões **35**
4.4 Metodologia de desenvolvimento e manutenção utilizada pela equipe **38**
4.5 Necessidades Identificadas **39**
5 O Visualizador *VisProgress* **41**
5.1 Arquitetura **41**
5.2 Arquivo Descritor **43**
5.3 Interface **46**
5.4 Arquivos com descrição dos grafos **50**
6 Visualização de Grafos **52**
6.1 Ferramenta DOTTY **52**
6.2 DOT **54**
6.3 NEATO **55**
6.4. LEFTY **55**

6.5 Comentários Finais 56
7 Avaliação do Protótipo *VisProgress* 57
7.1 Método e Instrumento 57
7.2 Resultados 58
8 Conclusões e Possibilidades Futuras 62
Anexo1 Questionário de Avaliação do Visualizador V*isProgress* 64
Anexo 2 Exemplo de um Arquivo Descritor 66
Bibliografia 67

Lista de Figuras

FIGURA 2.1 - Técnica de Visualização Document Lens [ROB93] 15
FIGURA 2.2 - Utilização de glifos para representação de informações para manutenção de software. (a) Timewheel glyph; (b) 3D wheel glyph e (c) infobug glyph. ... 16
FIGURA 2.3 - Técnica de Visualização *Cone Tree* [ROB91] 17
FIGURA 2.4 - Técnica de Visualização *Cam Tree* [ROB93] 17
FIGURA 2.5 - Técnica de Visualização Hyperbolic Tree (implementação atual da empresa Inxight) 18
FIGURA 3.1 - Visualizador de código *Visual Insights* [FYO97] 28
FIGURA 4.1 - Exemplo de um programa em *Progress* 35
FIGURA 4.2 - Metodologia de desenvolvimento e manutenção de *software* da Benfare Informática 39
FIGURA 5.1 - Arquitetura do visualizador. 42
FIGURA 5.2 - Trechos de programas em *Progress* ilustrando os comandos *for each* e *find* 44
FIGURA 5.3 - Formato do arquivo descritor. 46
FIGURA 5.4 - Interface inicial do *VisProgress* 46
FIGURA 5.5 - Grafo de chamada de programas exibido através da ferramenta *Dotty*. Os retângulos representam os programas, não importando seu tipo (cadastro, relatório, cálculo). 47
FIGURA 5.6 - Grafo de compartilhamento de variáveis exibido pela ferramenta *Dotty*. Os retângulos representam os programas e as elipses as variáveis compartilhadas pelos programas. 48
FIGURA 5.7 - Janela para escolher o programa que deseja visualizar as informações. 48
FIGURA 5.8 - Janela com as informações textuais: exemplo tem várias opções selecionadas. 49
FIGURA 5.9 - Janela com as informações textuais: exemplo com poucas opções selecionadas. 50
FIGURA 5.10 - (a) Exemplo de arquivo para a visualização de um grafo dirigido e (b) grafo criado pela ferramenta DOTTY correspondente ao arquivo (a). 51
FIGURA 5.11 - (a) Exemplo de arquivo para a visualização de um grafo não dirigido e (b) o grafo criado pela ferramenta DOTTY correspondente ao arquivo (a). 51
FIGURA 6.1 - Componentes da ferramenta *Dotty* 54

Lista de Tabelas

TABELA 4.1 - Componentes do sistema Benfare 37
TABELA 7.1 - Itens avaliados do Visualizador *VisProgress*..... 59

Resumo

O desenvolvimento e manutenção de *software* fazem parte de um processo intrinsecamente difícil e que consome tempo e custos, principalmente quando o sistema consiste de milhares de linhas de código. Por isso, sistemas de visualização de *software* objetivam prover mecanismos para construir representações visuais de informações sobre programas e sistemas, através das quais o programador pode analisar e compreender características de sua estrutura e funcionamento em um maior nível de abstração do que o código fonte. Assim, ferramentas visuais de *software* que suportam as tarefas de desenvolvimento, depuração, manutenção e reutilização tornam-se mais necessárias pelo fato de ajudarem a reduzir a complexidade inerente do processo de compreensão.

Esse trabalho tem como objetivo principal o desenvolvimento de um visualizador que exiba as informações existentes nos programas de forma mais rápida e legível, evitando que o programador/analista tenha que percorrer as linhas de código.

O texto inicialmente situa a área de visualização de informações, abordando a área de visualização de software, uma vez que a visualização de *software* é assim chamada por tratar da visualização de informações que são extraídas de programas. Em seguida, é apresentado um estudo de caso baseado no desenvolvimento dos sistemas da empresa Benfare Informática, no qual caracteriza-se a necessidade de ferramentas que auxiliem a compreensão de programas, com o objetivo de otimizar as operações de manutenção e desenvolvimento de programas. O restante do trabalho trata do sistema *VisProgress* que foi um protótipo desenvolvido como ferramenta de apoio para a equipe de desenvolvimento da empresa e como forma de avaliar o uso de técnicas de visualização em tal situação.

A ferramenta desenvolvida é um visualizador de informações que percorre programas escritos em *Progress,* ferramenta de desenvolvimento utilizada pela empresa, e extrai as informações encontradas nos programas. A visualização é dividida em três partes. A primeira permite a visualização de informações textuais extraídas diretamente do código fonte dos programas que compõem os sistemas, a segunda faz a visualização de um grafo que representa a chamada de programas, e a terceira e última faz a visualização também de um grafo, porém representando o compartilhamento de variáveis entre os programas. A obtenção dos grafos foi construída em Delphi, porém a visualização gráfica é feita através da ferramenta Dotty, ferramenta específica para visualização de grafos.

Após a descrição do protótipo implementado, são apresentados os resultados obtidos com a avaliação da ferramenta feita nas empresas Benfare Informática e Dzset Soluções e Sistemas para Computação.

PALAVRAS-CHAVE: Visualização de Informações, Visualização de *Software,* Visualização de Grafos, Compreensão de Programas.

TITLE: “INFORMATION VISUALIZATION APPLIED TO SOFTWARE MANAGEMENT”

Abstract

Software development is a hard process and much time and money are spent to maintain a large system. Software visualization systems attempt to provide mechanisms to create visual representations of information about programs and systems, in such a way that the user can analize and understand chacarcteristics of its structure and function on a higher abstraction level than source code.

Then main goal of this work is to develop a visualization tool for to displaying information about the programs more quick and easily without the need of analyzing the source code. A study in the Benfare Informática company was carried out, and it was demonstrated the need of visualization tools for helping software development.

The text introduces the reader to information visualization and, more specifically, to software visualization. Then, a case study based on the systems developed by Benfare Informática is presented, allowing estabilishing the tools that are needed in the process of software comprehension. The rest of the text discusses the VisProgress system developed as a prototype for testing ideas about software visualization.

VisProgress is an information viewer that analyzes the programs implemented in Progress, derives information from the source code and produces visualizations. In the visualization module, three visual representations are available: graph visualization showing the relation "calls" (i.e., which programs are called by others), graph visualization with shared variables, and numerical information visualization. The visualization module was developed in Delphi and used Dotty as a tool for displaying the graphs. Experimental results obtained from tests with potential users are also discussed.

KEYWORDS: Information Visualization, Software Visualization, Graph Visualization.

1 Introdução

Ao longo das duas últimas décadas a produção e coleta de dados através dos mais diversos equipamentos e processos, a disseminação de tecnologias de informação e comunicação fez aumentar de forma vertiginosa a quantidade de informações disponíveis. Mais do que a disponibilidade de informações, entretanto, é o entendimento das mesmas e sua conseqüente utilização para a geração de novos conhecimentos e tecnologias que permite a evolução das diferentes áreas da atividade humana.

Do tratamento de dados estruturados, convencionais, armazenados em sistemas de bancos de dados na forma de registros numéricos e caracteres, as organizações passaram a ter necessidade de armazenar e manipular dados semi-estruturados como imagens, vídeos, som, e-mails, palestras e documentos [SIL96]. Além disso, a possibilidade de estabelecimento dos mais diversos relacionamentos entre conjuntos de dados introduziu um grau de abstração adicional no que tange à representação de informações.

Já nas áreas científicas e de engenharia, a quantidade de dados gerados por processos de simulação, ou coletados por instrumentos de medição cada vez mais precisos e com maior resolução, trouxe a necessidade de armazenamento, manipulação e análise de volumes cada vez maiores de dados associados a objetos, fenômenos ou entidades físicas. Para manipulação, entendimento e comunicação de dados dessas áreas consolidaram-se, na década de 90, técnicas computacionais de representação visual e de interação. Tais técnicas fazem uso de computação gráfica, processamento de imagens e sinais, análise numérica e interação homem-computador para propiciarem um maior entendimento dos dados e constituem a área que passou a ser denominada de Visualização de Dados Científicos ou Visualização Científica [MCC87].

Em paralelo ao desenvolvimento da Visualização Científica, o tratamento de grandes volumes de dados estruturados ou não por parte da comunidade de Sistemas de Informações e Banco de Dados deu-se pelo desenvolvimento e aplicação de técnicas de:

a) recuperação de informações (*information retrieval*), para coleta e classificação de objetos como documentos, por exemplo, contendo informações relevantes com textos, palavras e padrões pré-estabelecidos [SPA97];
b) extração de informações (*information extraction*), para a inferência de informações em textos ou dados semi-estruturados, transformando-as em informações estruturadas [RIL94];
c) descoberta de conhecimento (*knowlegde discovery*) e mineração de dados (*data mining*), para a identificação e extração de informações implícitas previamente descobertas e potencialmente úteis em grandes conjuntos de dados [SHA94].

Mais recentemente [GER97], técnicas de visualização passaram a ser empregadas tanto para a comunicação de grandes volumes de informações como para apoiar os processos de exploração e entendimento dessas informações. Exemplos típicos que originaram as demandas por essas técnicas são análise de dados financeiros de grandes corporações, dados sobre a organização e estrutura de rede de computadores e de sistemas de arquivos em grandes sistemas

computacionais e dados sobre a organização semi-hierárquica em hiperdocumentos e em *sites* na WWW.

Assim, a visualização de informações que abrange aspectos de visualização científica, interfaces homem-máquina, processamento de imagens e computação gráfica, é uma área que emergiu dentro da comunidade de visualização para tratar alguns problemas fundamentais associados à representação visual de novas classes de dados e informações, geralmente abstratas, e que não possuem uma representação física óbvia e natural, fazendo com que a informação, na maioria das vezes, não seja automaticamente mapeada para o mundo físico [ROB93]. O problema principal é, então, descobrir novas metáforas visuais para representar a informação, ou seja, que meios utilizar para representar dados abstratos e como mapear esses dados em um espaço físico.

A visualização de informações tem por objetivo facilitar o entendimento de dados, processos e fenômenos por elas representados, uma vez que a representação visual de dados e informações aliada a processos interativos proporciona uma forma mais simples e intuitiva de entender melhor e mais rapidamente o significado desses dados. Qualquer área que gere dados de volume ou complexidade razoável é, potencialmente, área de aplicação de técnicas de visualização. Uma dessas áreas de aplicação é a Engenharia de Software.

O desenvolvimento de *software* é um processo intrinsecamente difícil e que consome tempo e custos, principalmente quando o sistema consiste de milhares de linhas de código. Por isso, os sistemas de visualização de *software* objetivam prover mecanismos para construir representações visuais de informações sobre programas e sistemas, através das quais o usuário pode analisar e compreender características de sua estrutura e funcionamento em um maior nível de abstração do que o código fonte. As ferramentas visuais de *software* suportam as tarefas de desenvolvimento, depuração, manutenção e reutilização e tornam-se mais necessárias pelo fato de ajudarem a reduzir a complexidade inerente do processo de compreensão de *software* [FYO97].

Pode-se dizer que a utilização de técnicas de visualização de software durante o processo de desenvolvimento cria vantagens competitivas tais como: melhoria na produtividade dos desenvolvedores de *software*, alta qualidade do *software*, redução do tempo gasto e, conseqüentemente, redução do custo de desenvolvimento [FYO97].

O PPGC/UFRGS atua há vários anos na área de visualização científica, com trabalhos, até certo ponto isolados, na visualização de dados astrofísicos, meteorológicos, geológicos e médicos [OLA93, MAN94, MAN96, BAS98, SIL00]. Atualmente, tendo sido concluída a primeira dissertação em Visualização de Informações [EST00], foi iniciado um projeto visando abordar com maior ênfase a área de visualização de informações. Este projeto permitirá investigar técnicas de visualização de informações com aplicação inicialmente a dados estruturados, organizados hierarquicamente ou na forma de grafos. Vinculados a esse projeto encontram-se dois trabalhos em40 andamento [LUZ01, CAV01]. Uma das metas previstas no projeto é a avaliação de técnicas quanto à efetividade de representação visual e possibilidades de interação com o espaço de informação [LUZ01].

Como forma de complementar o escopo de atuação do PPGC/UFRGS na área de visualização de informações, o presente trabalho tratará a questão da visualização de *software*,

em particular, da estrutura de grandes sistemas de *software*, visto ser esta uma das possibilidades de aplicação de técnicas de visualização.

Esse trabalho tem como objetivo principal o desenvolvimento de um visualizador que exiba informações a respeito de programas de forma mais rápida e legível, sem que o programador precise percorrer as linhas de código. Um estudo realizado na empresa Benfare Informática demonstrou a necessidade de ferramentas que auxiliem a equipe de desenvolvimento na manutenção e desenvolvimento dos seus sistemas. Desta forma, a avaliação dos benefícios de tais técnicas em um ambiente de desenvolvimento pode ser realizado.

Nesse trabalho, foi desenvolvido um programa visualizador, que faz a visualização das informações contidas nos programas desenvolvidos em *Progress* pela empresa Benfare Informática. Esse visualizador inicialmente percorre as linhas de código e cria um arquivo descritor com as informações consideradas relevantes. A partir desse, mostra algumas dessas informações na forma textual e outras na forma de grafos através da ferramenta de visualização de grafos *Dotty*.

A seguir é descrito, no capítulo 2, um estudo realizado sobre Visualização de Informações, abordando também recuperação e extração de informações. O capítulo 3 trata especificamente de Visualização de *Software*, bem como de requisitos de uma ferramenta de visualização de *software*, já que este é o tema central da dissertação.

O quarto capítulo é destinado a relatar o estudo de caso, a empresa Benfare Informática, visto ser ela a empresa que serviu de base para o desenvolvimento e teste do visualizador. A Benfare desenvolve sistemas baseados em *Progress*, uma linguagem de quarta geração integrada a um banco de dados relacional.

O capítulo 5 descreve o protótipo implementado *VisProgress*, enquanto o sexto capítulo apresenta um estudo da ferramenta Dotty e seus componentes, a qual foi utilizada para a visualização dos grafos gerados pelo protótipo desenvolvido.

O capítulo 7 apresenta a avaliação do visualizador, a qual foi aplicada na empresa que serviu de estudo de caso assim como em outra empresa que também utiliza o *Progress* como base de seus sistemas.

Finalmente, no oitavo e último capítulo, são apresentadas as conclusões e as possíveis extensões ao trabalho.

2 Visualização de Informações

2.1 Introdução

A visualização de informações é uma área recente de pesquisa da Computação Gráfica que procura criar novos algoritmos e desenvolver novas técnicas de visualização para grandes quantidades de informações, tais como documentos da Web, estruturas de diretórios e arquivos, e outros tipos de dados abstratos, que incluem informações temporais, lineares, bidimensionais, tridimensionais, multidimensionais e hierárquicas. A necessidade de novos recursos para facilitar a representação decorre das necessidades de como tratar, atualmente, dados oriundos de diversas aplicações [GER97].

Como já mencionado, o principal objetivo da visualização de informações é facilitar o entendimento das informações, uma vez que representações visuais tendem a proporcionar uma forma mais simples e intuitiva de entender melhor e mais rapidamente o significado dos dados representados [GER97]. Esta necessidade por ferramentas e técnicas para tratamento de dados estruturados ou não-estruturados se torna cada vez mais urgente na medida em que aumenta o volume de informações armazenadas, seja na Internet ou em ambientes corporativos privados.

Chen [CHE94] cita a frustração dos usuários com o problema da "sobrecarga de informações". Ela ocorre quando o usuário tem muita informação ao seu alcance, mas não tem condições de tratá-la ou de encontrar o que realmente deseja ou lhe interessa. A visualização de informações utiliza computação gráfica e animação para estimular reconhecimento de padrões e estruturas em dados das mais variadas fontes. Isto se dá, pela exploração do sistema perceptivo humano de forma similar à da Visualização Científica, que permite cientistas perceberem padrões em grandes coleções de dados.

Para atingir isso, sentiu-se a necessidade de saber como apresentar a informação, de tal maneira, que seja possível sua assimilação pelo sistema perceptivo humano e também como apresentar detalhes da informação sobre um tópico específico, mantendo uma visão completa de todo o conjunto disponível.

Por isso, é necessário utilizar meios para reduzir o volume de informações a representar na tela. Tais meios incluem: a) filtros, b) identificação de diferentes níveis de interesse, c) redução do número de variáveis a representar, eliminando e/ou agrupando variáveis e d) utilização de diferentes simbologias com nível de detalhe variando com o fator de escala e/ou importância do objeto.

A utilização da visualização de informações para apoiar a resolução de problemas ou a análise de dados requer o conhecimento do sistema onde será inserida, assim como das necessidades de visualização e a agregação de funcionalidades para tornar as visualizações efetivas, ou seja, é necessário primeiro estudar o problema e depois utilizar técnicas de visualização, se adequado. É necessário, sobretudo entender melhor como o ser humano interage com a informação, ou seja, como ele a percebe visualmente, e não visualmente e como a mente trabalha quando está procurando por informações conhecidas e não conhecidas. Assim, não

somente é necessária uma representação visual adequada das informações, mas é imprescindível qualidade na interação, com interfaces flexíveis, ferramentas de navegação e métodos de pesquisa apropriados para cada um dos tipos de usuários e de aplicações.

Considerando que as técnicas de visualização visam, sobretudo, apresentar visualmente uma dada informação e permitir interação, ela é essencialmente um processo de mapeamento. Este mapeamento se dá entre representações computacionais e representações perceptíveis, através da seleção de técnicas de codificação que maximizem a comunicação e o entendimento humano.

2.2 Processo de Visualização

No processo de visualização pode-se identificar três fases, que são: **preparação dos dados** (pré-processamento), **mapping (**mapeamento) e **rendering** (renderização) [CAM97]. A fase de pré-processamento inclui operações de formatação de dados e/ou normalização. Na fase de mapeamento é realizada a associação entre os dados e as representações gráficas e no *rendering* é gerada a imagem. Mais três componentes no processo de visualização estão referenciados em [CAM97], são eles: **abstração** (*abstraction*): onde é realizada uma filtragem semântica dos dados, sendo um processo anterior ao *mapping,* **interação**: que trata os aspectos de interação com o usuário e **estado de visualização** (*visualization state*) que armazena a informação sobre as operações do usuário e o estado corrente do processo de visualização.

Assim, um sistema de visualização pode ser caracterizado pela provisão de suporte para os seguintes aspectos [CAM97]:

- *Apresentação e Codificação visual da informação:* é a característica que define os sistemas de visualização. Uma representação visual permite que o usuário possa observar a informação que tenta compreender através de representações visuais compactas que ressaltam aspectos relevantes da informação analisada.
- *Navegação:* é a técnica que permite ao usuário percorrer o espaço de informação visualizado, permitindo que o usuário consiga "ver" com mais detalhes as informações, bem como "ver" a informação de diferentes perspectivas.
- *Interação:* essa característica permite ao usuário interagir com a representação da informação que está sendo visualizada, podendo realizar mudanças na forma como os dados estão sendo apresentados, selecionar um subconjunto de interesses, etc.
- *Relacionamento com os dados:* o sistema de visualização deve estar vinculado, de alguma forma, com os dados visualizados.

2.3 Técnicas de Visualização

Técnicas e ferramentas de visualização possibilitam cientistas explorarem seus dados de pesquisa, permitindo assim, a comunicação de suas descobertas às outras pessoas. Estas técnicas também permitem que profissionais em geral consigam visualizar informações de maneira mais clara e legível. A visualização permite em última análise, a conversão de informação que não pode ser diretamente percebida pelo olho humano em formas satisfatórias para as pessoas.

Uma técnica de visualização envolve vários passos, que são:

- A construção de um modelo dos dados;
- A seleção de algum mecanismo de mapeamento do modelo em um objeto de visualização abstrato, como uma imagem ou mapa de contornos;
- A geração da imagem no dispositivo gráfico de exibição.

Sabe-se que é possível criar vários modelos, porém deve-se escolher o mais adequado, uma vez que este afeta a interface dos sistemas de visualização [SCH96]. Vale salientar que, de acordo com os tipos de dados e com a tarefa a ser realizada pelos usuários, pode-se identificar qual é a representação visual mais adequada [FRE93].

Esta seção dedica-se a apresentar brevemente as técnicas de visualização de informações que teriam aplicabilidade na visualização de dados oriundos de processos da Engenharia de Software, mais especificamente, aqueles de construção e manutenção de sistemas. Normalmente, estão envolvidos nesses processos dados na forma de textos (os programas), de valores numéricos (estatísticas obtidas, como tamanho dos programas em linhas), de informações abstratas como a estrutura de relacionamentos entre os programas.

2.3.1 Visualização de textos

Uma das principais técnicas de visualização para dados como textos é a ***Document Lens*** [ROB93] que é usada no sistema *Information Visualizer*, desenvolvido pela Xerox, Palo Alto, para integrar detalhes e contexto. É projetada para permitir uma visão detalhada de uma porção de uma superfície planar contendo um conjunto de informações textuais, permitindo que uma grande parte do contexto planar seja visível simultaneamente, superando assim, alguns dos problemas de visualização com o uso de lentes. A técnica ***Document Lens*** mapeia essa superfície dentro de um modelo sólido de uma pirâmide truncada, e na área retangular truncada, ela mantém a área em detalhe e nos lados inclinados da pirâmide, ela mantém o contexto (fig. 2.1). A área truncada pode ser arrastada em direção ao visualizador para aumentar os detalhes e diminuir o tamanho do contexto, a inclinação da pirâmide pode ser aumentada para aumentar a quantidade da superfície visualizada na área truncada. O nome ***Document Lens*** aproxima a aplicação da técnica para um documento de texto multipáginas inserido em uma grade retangular de páginas.

FIGURA 2.1 - Técnica de Visualização Document Lens [ROB93]

Além da técnica citada acima, existe a ***Filter Zooming***, que foi desenvolvida para grandes documentos, como páginas da Web. É uma técnica, segundo [HOL97], para visualização de grandes conjuntos de dados, oferecendo uma visão global dos dados, permitindo ao usuário acesso instantâneo para qualquer parte do conjunto de dados. O conjunto de dados que é visualizado é, primeiramente dividido em um número de partes de tamanho iguais, que são chamadas páginas. Em um grande documento, cada página é uma seção do texto. As páginas são então dispostas como uma descrição rápida sobre a exibição da esquerda para a direita e de cima para baixo. Quando os usuários querem visualizar uma certa página, podem clicar sobre ela e trazê-la para o foco. A página é então expandida para um tamanho apropriado para leitura. Todas as páginas vizinhas são diminuídas e reagrupadas para criar lugar, mantendo a página intacta da esquerda para direita, e de cima
para baixo.

2.3.2 Visualização de valores diversos

Duas grandes classes de técnicas de visualização podem ser aqui utilizadas: gráficos convencionais e ícones ou glifos[1].

Diversos gráficos convencionais podem ser utilizados para demonstrar tendências de valores, muitas vezes num domínio bastante extenso. A técnica ***Information Mural*** (GVU, Georgia Tech) é uma representação bidimensional reduzida de um espaço de informação que preenche completamente a tela ou janelas de exibição. O ***Mural*** é criado usando técnicas de *anti-aliasing* e vários atributos visuais, tal como intensidade de sombreamento, cor e tamanho do pixel. É uma representação proveitosa para a visualização de grandes fontes de informação. A técnica ***Information Mural*** cria uma versão miniatura do espaço de informação usando atributos visuais e pode ser usada como visualizações *stand-alone* ou em visões navegacionais globais. Ela é útil para visualização de tendências e padrões sobre a distribuição total da informação, permitindo ainda capacidade de *panning* e *zooming*.

[1] Ícones são símbolos bidimensionais utilizados em geral para representar entidades ou pontos no espaço de dados. Já glifos são objetos cujos atributos geométricos e visuais podem ser utilizados para mapear atributos de dados.

A Visualização baseada em glifos foi utilizada por Chuah e Eick [CHU 98] para representar informações de manutenção de software. Os autores propuseram diferentes glifos como *timewheel glyph*, o *3D wheel glyph* e o *infobug glyph*, para representar diferentes informações obtidas ao longo do tempo em processo de manutenção de software (fig. 2.2). O *timewheel glyph* é, na realidade, um conjunto de gráficos de linha convencionais, dispostos ao redor de um círculo que representa o domínio tempo. Assim, múltiplas séries temporais são mostradas na mesma representação. Os mesmos atributos são mapeados para um glifo no *3D wheel glyph*. Já o *infobug glyph* utiliza a estrutura básica de um inseto para mapear esses atributos. A variação no tempo é representada pelo eixo Y, enquanto o eixo X das diversas partes reflete uma variável numérica. Por exemplo, a largura da asa esquerda codifica o número de linhas de código enquanto a da asa direita, o número de erros.

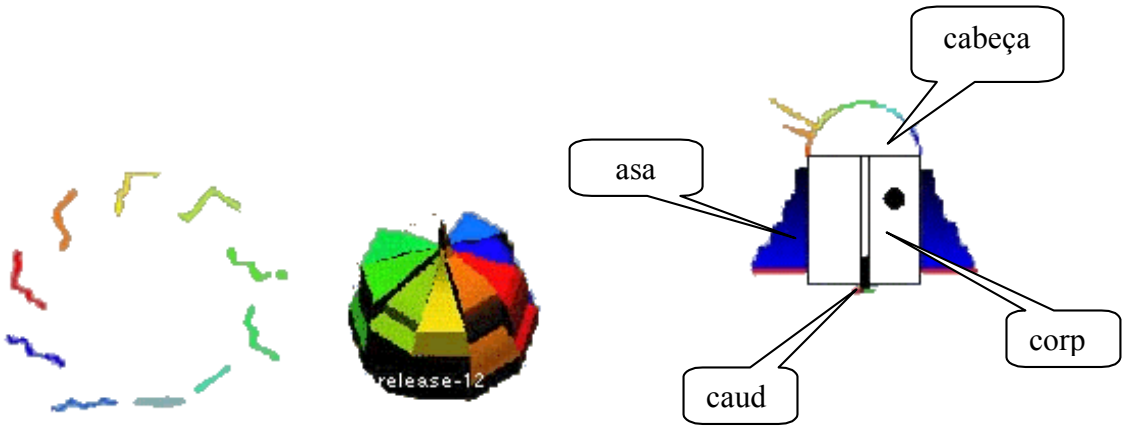


FIGURA 2.2 - Utilização de glifos para representação de informações para manutenção de software. (a) Timewheel glyph; (b) 3D wheel glyph e (c) infobug glyph.

2.3.3 Visualização de estrutura e relacionamentos

Além das técnicas mencionadas acima, existem outras técnicas que são importantes na visualização de documentos e textos, as técnicas de visualização de árvores e grafos, que permitem representar a estrutura desses documentos e textos. Um modo usual de representar dados hierárquicos é estruturá-los como uma árvore de nodos de um diagrama de links. Mas para grandes árvores, estes diagramas rapidamente tornam-se desordenados e sem uso. Assim, vários autores dedicaram-se a encontrar formas alternativas de apresentar hierarquias e grafos. Algumas das técnicas mais importantes são:

- ***Cone Tree (fig. 2.3):*** Esta técnica foi desenvolvida pelo Centro de Pesquisa da Xerox Palo Alto (PARC). A técnica ***Cone Tree*** é usada para representar hierarquias tais como Web sites e diretórios de arquivos. É uma representação tridimensional de informação hierárquica: um nodo da árvore é localizado no vértice do cone e todos os filhos são colocados ao redor da base circular de outro cone. Diferentemente de uma árvore de links e nodos, a técnica ***Cone Tree*** torna uma grande quantidade de informação visível quase todo o tempo. A técnica ***Cone Tree*** projeta uma árvore dentro de uma visão tridimensional semitransparente. As árvores de

tamanho modesto podem ser apresentadas usando esta técnica; apesar disso, muitas árvores grandes ainda são difíceis de gerenciar [ROB91].

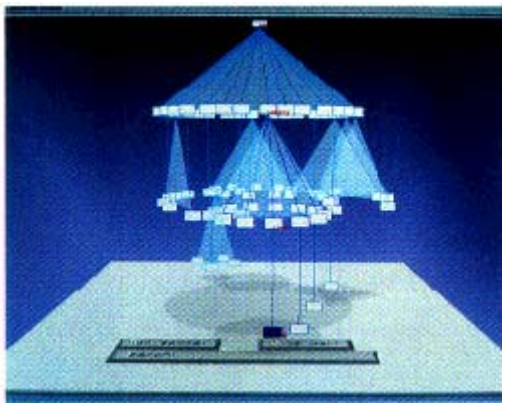


FIGURA 2.3 - Técnica de Visualização *Cone Tree* [ROB91]

- ***Cam Tree (fig. 2.4)***: A técnica ***Cam Tree*** é idêntica a *Cone Tree*, exceto que na ***Cam Tree***, as árvores crescem horizontalmente ao invés de verticalmente. O objetivo dessas técnicas é permitir que uma grande quantidade de informação seja navegada e exibida de uma forma intuitiva. Essas técnicas são construídas colocando o nodo raiz no ápice de um cone translúcido próximo ao topo da exibição. Todos os nodos filhos são então distribuídos em distâncias iguais ao longo da base do cone. Este processo é repetido para todos os nodos da hierarquia, como o diâmetro base dos cones reduzindo a cada nível, para garantir espaço suficiente para acomodar todos os nodos folhas. Visualizações ***Cam Tree*** e ***Cone Tree*** originais produzidas pela Xerox PARC possibilitam que a árvore seja rotacionada suavemente trazendo algum nodo em particular para um ponto de foco [ROB93].

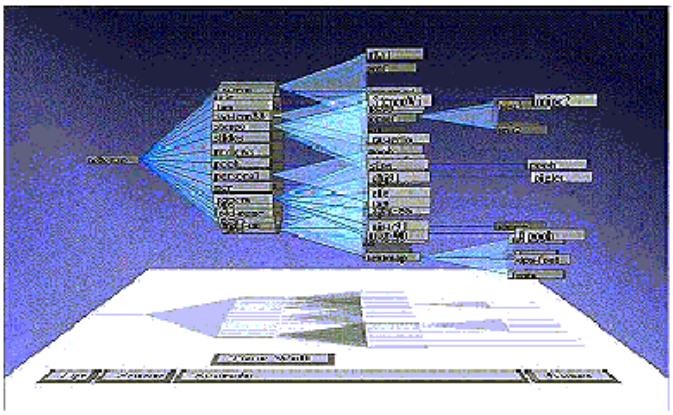


FIGURA 2.4 - Técnica de Visualização *Cam Tree* [ROB93]

- ***Hyperbolic Tree*** (Xerox)/ ***Hyperbolic Browser***: Nesta técnica, o diagrama de nodos e links é planejado de uma maneira uniforme num plano hiperbólico que é mapeado para uma região de exibição circular [LAM 95]. A técnica ***Hyperbolic Tree*** possui as seguintes características: os componentes diminuem de tamanho em direção aos extremos da área circular; crescimento exponencial do número de componentes significa que pode-se exibir um número infinito de componentes; o foco está no centro da exibição, mas um nodo diferente pode ser movido para o ponto focal, ocasionando a alteração de todo o *layout*. Esta técnica foi originalmente implementada na forma 2D (figura 2.5) [LAM 95] tendo sido patenteada e hoje é produto comercial da Inxight[2]. A forma tridimensional foi criada por Munzner [MUN 97] sendo o mapeamento das posições dos nodos realizado para uma circunferência (fig. 2.5).

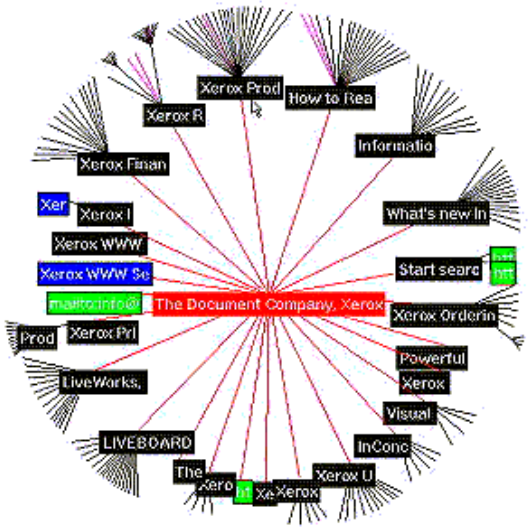


FIGURA 2.5 - Técnica de Visualização Hyperbolic Tree (implementação atual da empresa Inxight)

[2] http://www.inxight.com

3 Visualização de *Software*

3.1.Introdução

Geralmente, entende-se por compreensão a capacidade ou conhecimento que possui uma pessoa para explicar um dado conceito, tanto do ponto de vista do seu significado como do seu funcionamento. Um programa expressa uma solução para um problema dentro de um domínio de aplicação em termos de linguagem formal, a qual constitui o domínio de programação. Uma pessoa compreende um dado programa, quando é capaz de explicar como funciona, sua estrutura, seu comportamento, seu relacionamento com o domínio de aplicação, e modificá-lo para adaptá-lo a novos requisitos, ou atualizá-lo adequadamente para o objetivo que foi projetado.

A dificuldade de compreensão de programas pode ser atribuída a uma combinação de fatores que abrangem aspectos tais como complexidade do domínio de aplicação, limitações cognitivas dos programadores, práticas de desenvolvimento e características da linguagem de programação. Uma pessoa é capaz de assimilar, simultaneamente, aproximadamente sete itens de informação. Estes itens não são lembrados após menos de um minuto, se não são novamente exercitados. Um item é uma peça simples de informação, a qual pode representar um dado elementar ou uma estrutura complexa tratada como unidade. Essa limitação é um dos principais problemas que enfrenta um programador para compreender e manipular sistemas complexos, pois inibe a capacidade de processar simultaneamente as diferentes partes do sistema que interagem entre si [CAM97].

A compreensão de programas pode ser caracterizada como um processo através do qual um programador constrói um modelo mental do programa. Este modelo é uma representação conceitual que um programador tem do programa em consideração, obtida a partir de diferentes aspectos do processo de desenvolvimento e manutenção do *software*. Estes conhecimentos são utilizados para reconhecer estruturas abstratas a partir do conhecimento adquirido durante o processo de análise, tal como estrutura de controle do programa, funcionalidade e arquitetura global. Através do modelo mental obtido no processo, o programador é capaz de decidir como o programa deveria ser modificado para satisfazer os novos requisitos e, conseqüentemente, alterá-lo num ciclo de tempo menor, se utilizadas ferramentas de visualização de *software.*

Ferramentas de *software* para apoio à compreensão de programas são de grande importância para ajudar a reduzir a complexidade inerente do processo de compreensão. Estas ferramentas auxiliam o programador na construção de um modelo mental do programa, através de mecanismos de análise, exploração e visualização da informação em diferentes níveis de abstração. Freqüentemente, essas ferramentas fornecem diferentes representações visuais que sintetizam propriedades relevantes do programa analisado. As características destes mecanismos, referidos genericamente como técnicas de suporte, determinam a eficácia de uma ferramenta para adequar-se a diferentes atividades e domínios de aplicação [TIL96].

Segundo [FYO97], o efeito da conversão de *software* do ano 2000 ilustra dramaticamente como se consome tempo e custos para manter grandes sistemas de computadores, especialmente quando cada sistema consiste de milhões de linhas de código fonte. Entender como os componentes do sistema interagem é o fator chave em mudanças na implementação, adição de novas características e fornecimento de manutenção no sistema. Qualquer ajuda que os programadores conseguirem para entenderem, ou seja, tornar o entendimento do sistema mais fácil, reflete na melhora da produtividade do programador, qualidade do *software*, reduzindo o tempo gasto.

Grandes sistemas podem conter milhares de linhas de código. Mesmo que pareçam simples, com projetos pequenos, são complicados. Entender, alterar e reparar código em grandes sistemas consome bastante tempo e custo. O conhecimento do código diminui com a idade do *software* e os programadores que desenvolveram e o projetaram migram para novas tecnologias. Os documentos do projeto também geralmente expiram, tornando o código apenas como um guia para o comportamento do sistema. É tedioso reconstruir sistemas complexos analisando o código.

Talvez a maior dificuldade nos projetos de engenharia de *software* envolva "programação em massa". Esses projetos grandes requerem, freqüentemente na manutenção, valorização envolvendo mudanças no código complexo escrito há muitos anos.

A visualização de *software* pode ajudar engenheiros de *software* a enfrentarem essa complexidade enquanto aumenta a produtividade dos programadores. O *software* é intangível, não tem estrutura física ou tamanho. Depois de escrito, o código "desaparece" em arquivos mantidos em discos. Ferramentas de visualização de *software* utilizam técnicas gráficas para tornar o *software* visível, através da apresentação dos programas, trechos de programas e comportamento dos mesmos. Figuras do *software* podem ajudar os membros do projeto a lembrarem - e novos membros a descobrirem - como o código funciona.

Para demonstrar a efetividade dos mecanismos providos, é necessário fazer experimentos com usuários não experientes, alguns utilizando uma ferramenta de visualização de *software* e outros não. Pode-se verificar que aqueles que utilizam ferramentas de *software* produzem aplicações de maior qualidade [CAM97]. Entretanto, é evidente que existe limitações nas ferramentas, pois usuários requerem níveis de abstração diferentes. Por isso, mostra-se necessário elaborar mecanismos que permitam adaptar ferramentas aos diferentes níveis de abstração nos quais o *software* pode ser representado, bem como as diferentes formas de representação preferidas dos usuários.

Determina-se com isso, a necessidade de construção de ferramentas de visualização que permitam alto grau de flexibilidade para a adaptação do comportamento dessas em tempo de execução, tanto nos aspectos de captura de informação quanto nos aspectos de abstrações e técnicas de visualização.

Como falado anteriormente, grandes sistemas de *software* são difíceis e complexos de entender, e para tais convém aplicar técnicas de visualização visando alguns benefícios como: encontro de padrões repetidos no código fonte, redução do tempo necessário para investigar problemas, entendimento do código e relacionamentos, identificação de oportunidades para reconstrução do *software*, diminuição da dificuldade de aprendizagem e integração de novos desenvolvedores em um projeto. Esses benefícios criam vantagens competitivas para o ambiente

de desenvolvimento, as quais incluem, melhoria na produtividade dos desenvolvedores de *software*, alta qualidade de *software*, redução no tempo gasto e, conseqüentemente redução de custos de desenvolvimento [FYO97].

Nesse contexto, uma das grandes necessidades atualmente para a comunidade de sistemas de informação é a utilização de aplicações de visualização de *software* no ambiente de desenvolvimento. Desenvolvedores e seus gerentes têm adotado aplicações de visualizações integradas e interativas, por trazerem benefícios estratégicos que afetam positivamente o produto.

O projeto de uma ferramenta requer a identificação adequada das dimensões essenciais através das quais o domínio de aplicação pode ser caracterizado. No domínio dos sistemas de visualização de *software*, é definido um espaço de quatro dimensões que os caracteriza: escopo, abstração, técnica e método de especificação [CAM97].

- **Escopo:** representa aspectos do programa a serem visualizados como código, dados, estados ou comportamento, identificando assim, a natureza da informação que deverá ser obtida para ser visualizada.
- **Abstração:** identifica a natureza de como mapear conceitos do programa em representações gráficas, como *representação direta* (ex. visualização de listas encadeadas), *representação estrutural* (ex. visualização de grafos) ou *representação sintetizada* (ex. visualização de subsistemas).
- **Técnica:** representa os aspectos sintáticos e semânticos da representação gráfica e as facilidades de organização e manipulação dessa representação (ex. linguagem visual ou projeto da interface).
- **Método de Especificação:** define o mecanismo utilizado para capturar a informação a ser visualizada (ex. anotação de código, predefinição, associação).

[FYO97] define ainda alguns padrões de visualização, a saber:

- Foco da tarefa: as telas são constituídas com um entendimento das tarefas e análise dos objetivos dos programadores, e incorporam seus domínios pela tecnologia.
- Representação de domínio específico: a representação mostrada na tela deve ter atributos como cor, posição, estrutura e detalhes.
- Alta densidade de informação: representações que utilizam ícones compactos, coloridos mostram muitas informações em uma tela e mostram também um conjunto grande de dados.
- Filtragem interativa: filtros interativos fornecem uma interface natural com bastante detalhes que permitem os usuários reduzirem a quantidade de informação mostrada na tela.
- Visões de múltiplas ligações: a força de representação da visualização de sistemas é através da interação e filtragem.

Os sistemas de visualização de software podem ser caracterizados por apresentarem uma seqüência básica de quatro atividades que abrangem:

- captura de informação do programa alvo;
- representação dessa informação em termos adequados para sua análise automática ou manual;

- análise da informação e recolhimento de abstrações;
- apresentação da informação ao usuário;

Cada uma delas representa uma complexa área de problemas. Diferentes ferramentas requerem soluções particulares dependendo dos objetivos ou tarefas para as quais são projetadas. Por esta razão, diferentes técnicas têm sido propostas na literatura e adotadas em combinações particulares ou em ferramentas distintas [CAM97]. As diferentes técnicas são apresentadas a seguir.

3.1.1 Captura da Informação

A captura da informação do programa alvo é o primeiro passo para gerar a informação necessária para a análise ou visualização desse programa. As técnicas de análise utilizadas habitualmente para implementar este processo podem ser classificadas como estáticas e dinâmicas.

A análise estática consiste no reconhecimento sintático (*parsing*) do código fonte, a partir do qual uma representação abstrata do programa é gerada. A representação mais comum é realizada a partir de árvores de sintaxe abstrata, as quais contêm uma grande quantidade de unidades de informação sintática de granularidade fina em conjunto com as diferentes dependências entre essas unidades. Sobre esta representação, diferentes funções de análise são aplicadas para gerar ou recuperar abstrações de maior nível que o código, descobrir relacionamentos entre essas abstrações e produzir diferentes representações visuais do programa analisado.

As técnicas de análise dinâmica consistem na captura de informação acerca do comportamento do programa em tempo de execução. A informação de tempo de execução pode ser muito valiosa para compreender aspectos do comportamento de um programa que não podem ser deduzidos estaticamente. Ferramentas de medição de desempenho (*profilers*), por exemplo, são de utilidade para identificar aquelas porções de um programa nas quais concentra-se a maior carga de computação e, portanto, alvos de otimização.

O mecanismo utilizado pela maior parte das ferramentas atuais para capturar a informação dinâmica é a instrumentação de código fonte para o anúncio de eventos de interesse ou transições de estados do programa. Um evento de interesse denota uma ação realizada pelo programa, como por exemplo ativação ou retorno de um procedimento ou modificação do estado de uma variável. A execução de um programa pode ser representada por uma seqüência de eventos, a qual é utilizada para produzir uma representação, geralmente gráfica, da execução. A noção de evento é especialmente adequada para construir ferramentas de análise dinâmica por suportar a captura da informação em diferentes níveis de abstração. Um evento pode representar ações elementares de um programa, como envio de uma mensagem, até a execução de uma função completa do sistema.

O mecanismo de instrumentação consiste na aplicação, manual ou automática, do código da aplicação com anotações que invocam procedimentos de animação ou geram a informação relativa ao evento que representam.

Dentre as técnicas para capturar informação dinâmica, a reflexão computacional é uma técnica que permite coletar informações de um programa. É a capacidade que um sistema computacional tem para realizar computação sobre sua própria computação, ou seja, é capaz de analisar e modificar as entidades que definem seu próprio comportamento.

Esta capacidade implica na existência de uma representação das entidades que definem o comportamento do programa (funções, sentenças, dados, etc..) acessível em termos de dados. Esta representação é denominada auto-representação, pois é uma representação que o programa possui de si mesmo. Lisp é um exemplo deste tipo de representação. Linguagens reflexivas baseadas em Lisp, caracterizam-se por serem implementadas por um interpretador meta-circular, isto é, o próprio interpretador é implementado na mesma linguagem e acessível como um dado. A modificação da representação do interpretador produzirá a modificação do seu comportamento, modificando, em conseqüência, o comportamento do próprio programa.

A capacidade de modificar a auto-representação é denominada efetuação. A capacidade de inspecionar a auto-representação é denominada introspecção. Através da análise da auto-representação, tanto aspectos estáticos da estrutura do programa quanto aspectos dinâmicos de seu comportamento podem ser extraídos sem necessidade de ferramentas ad-hoc para tal fim [CAM97].

3.1.2 Representação da Informação

A representação da informação coletada define tanto as capacidades da ferramenta quanto a sua escalabilidade. Para sistemas de mediana complexidade, a quantidade de informação coletada pode ser enorme. Isto impõe a necessidade de organizá-la eficientemente em termos de espaço, facilidade de acesso e geração de abstrações.

O conjunto adequado de eventos primitivos que serão representados em uma ferramenta, depende fortemente do nível de abstração necessário para cada atividade. Por exemplo, eventos que denotam modificação de estado de variáveis são úteis para tarefas de depuração mas podem ser desnecessários para atividades de compreensão. Por esta razão, mecanismos para definir e adaptar eventos tornam-se um componente essencial para que uma ferramenta seja efetiva em uma ampla gama de aplicações que requerem a compreensão de um programa.

Um outro problema é a necessidade de organizar, de maneira eficiente, os milhares eventos gerados durante a execução do programa. Se estes eventos são processados de forma direta pelo sistema de visualização, podem produzir-se atualizações rápidas demais na visualização, inibindo a capacidade do programador para processar a informação mostrada pela ferramenta. Uma possível solução, é a utilização de *buffer* de eventos que é controlado pelo programador suspendendo a geração de novos eventos, enquanto o programador analisa detalhadamente a visualização.

3.1.3 Apresentação da Informação

A utilização de técnicas visuais é um componente essencial para facilitar o processo de compreensão de sistemas complexos. Basicamente, a visualização é um processo de conversão de dados em dimensões perceptuais, com o objetivo de facilitar a compreensão desses dados [ROB93].

Independente do mecanismo de captura, a grande quantidade de dados coletados para compreender sistemas complexos requer mecanismos para visualizar e ressaltar aspectos relevantes dessa informação. Grandes quantidades de dados podem facilmente inibir a capacidade humana para assimilá-los e construir um modelo mental adequado [SIL96]. Por esta razão, as técnicas de visualização providas são um fator chave para uma ferramenta apoiar eficazmente a compreensão de programas. Deste modo, o objetivo principal de uma visualização é condensar a maior quantidade possível de informação em uma mesma apresentação. Uma visualização compacta facilita o processamento imediato da informação transmitida [SIL96].

Técnicas de visualização, gráficas e textuais, permitem concentrar maior densidade de informação em uma apresentação e explorar a capacidade humana para análise de informação espacial e interpretação de cores. Também favorecem a construção de animações que facilitam o processo de compreensão de comportamento dinâmico, o qual não pode ser sinteticamente representado por notações textuais.

A utilização de técnicas de visualização suportadas por computador introduz a capacidade de interação do usuário com a informação apresentada. O usuário pode, além de visualizar os dados, interagir e realizar mudanças na apresentação. Entretanto, é necessário levar em consideração que o propósito de uma visualização é oferecer um veículo para a obtenção de conhecimento acerca da realidade sendo visualizada. Assim, um usuário deve ser capaz de observar os dados utilizando a técnica adequada para facilitar a compreensão dos dados. Segundo Andrienko e Andrienko [AND97], a seleção da técnica de visualização apropriada depende de vários fatores, como por exemplo, das características dos dados e seus relacionamentos, os objetivos perseguidos pelo usuário e fatores de percepção humana.

Os sistemas de visualização de informação, e particularmente os sistemas de visualização de *software* surgiram como um meio de fornecer suporte para o processo de compreensão de informação, através de facilidades para a construção de diferentes técnicas de visualização e de interação do usuário com essas visualizações. As ferramentas de manipulação direta para exploração de relacionamentos entre os dados visualizados são um componente essencial de qualquer sistema de visualização.

A navegação é uma técnica que faz parte integral das atividades de compreensão. O processo de compreensão de um programa consiste na exploração do espaço da informação que representa o programa analisado, através de apresentações visuais que facilitem a análise dessa informação, com seus relacionamentos, de diferentes perspectivas.

A navegação pode ser facilitada através de interfaces de manipulação direta que habilitem a seleção e o *zooming* das apresentações visuais, fornecendo mecanismos alternativos de exploração.

A facilidade para derivar e representar abstrações da informação coletada é um aspecto central nos sistemas de visualização de *software*. O termo abstração pode admitir um significado dual, derivado do processo através do qual a abstração é obtida:

- abstração como resultado da análise da informação do programa através da qual, relacionamentos funcionais implícitos no código são deduzidos, como por exemplo, subsistemas.
- abstração como resultado de um processo de filtragem de informação não relevante, para gerar representações sintéticas da informação.

No sentido de recuperação de abstrações, uma abstração denota aquelas construções que não são habitualmente suportadas de forma explícita pela linguagem de programação, mas que formam parte integral da forma na qual o *software* é conceitualizado pelo projetista.

Esses tipos de abstrações podem ser recuperados, ao menos parcialmente, através da análise da informação coletada. Para esta atividade, o mapeamento padrão é uma das técnicas mais importantes. Grande parte do processo de compreensão consiste no reconhecimento de padrões estruturais conhecidos, seja mentalmente pelo programador ou automaticamente pela ferramenta. Técnicas de mapeamento padrão são limitadas a problemas de pequena escala.

Neste contexto, o termo abstração é utilizado para denotar um nível de detalhe determinado de descrições ou visualizações, que só inclui aspectos relevantes da informação coletada, de acordo com alguma semântica fixada pela ferramenta ou selecionada pelo usuário.

Representações desta natureza são, habitualmente, o resultado de filtrar a informação do programa de acordo com propriedades das entidades envolvidas ou de consultas por relacionamentos e propriedades selecionadas pelo usuário. A filtragem de informação, segundo critérios estabelecidos, pode ser um complemento para auxiliar o usuário na procura de abstrações que não podem ser automaticamente derivadas.

Dentre as diversas alternativas de filtragem de informação existentes, a técnica de *slicing* [WIE94] consiste na divisão do código de um programa em fragmentos que podem afetar o valor de uma ou várias variáveis. Isolando essas sentenças, se reduz a carga cognitiva necessária para compreender uma porção de código potencialmente muito grande. Cada fatia pode ser apresentada de forma textual ou gráfica.

A construção de visualizações, e, particularmente, a disponibilização de recursos de manipulação direta, é uma das tarefas que implica mais custo de implementação na construção de ferramentas de visualização. A maior parte das ferramentas fornece pouca ou nenhuma facilidade para a construção de visualizações alternativas, limitando, em conseqüência, sua adaptabilidade. Na maioria dos casos a construção de visualizações é suportada por algum *framework* de interfaces.

A capacidade de adaptação das ferramentas é um dos aspectos principais a serem contemplados para adequarem-se a diferentes capacidades cognitivas, nível de conhecimento e referências dos usuários. É necessário fornecer mecanismos que permitam adaptar uma ferramenta para cada uma das atividades envolvidas no processo de compreensão.

Três elementos básicos de *software* podem ser visualizados: estrutura do *software* (através de um grafo direcionado), tempo de execução (através de algoritmos de animação) e o próprio código.

Algoritmos de visualização são, em geral, escritos manualmente e requerem um projetista para entender o código antes de visualizá-lo, tornando essa técnica inviável para grandes sistemas ou tarefas envolvendo descoberta de programadores. A estratégia geral para grandes projetos é decompor o projeto em módulos, em geral hierarquicamente, e mostrar cada módulo individualmente. Na prática, essa decomposição é freqüentemente o aspecto mais difícil da visualização. Quando o *software* é decomposto, a "figura grande", o todo é perdido, freqüentemente anulando o objetivo da visualização.

Para direcionar essas falhas, alguns autores desenvolveram técnicas padrões para visualizações de programas textuais, propriedades textuais e relacionamentos envolvendo programas textuais, visto que texto é o elemento dominante para implementar grandes sistemas. Com isso, autores tem que aplicar suas ferramentas para visualizar a versão do código, diferenças entre versões, propriedades estáticas do código, esboço do código e execução e partes de programas dinâmicos.

3.2 Exemplos de Ferramentas de Visualização de S*oftware*

Dentro da ampla gama de sistemas de visualização de *software* existentes, os sistemas de animação de algoritmos, como por exemplo, BALSA [BRO84], BALSAII [BRO88], TANGO [STA90], etc, representam uma classe particularmente relevante. Estes sistemas fornecem o suporte para a construção de animações do funcionamento de um programa, as quais são especialmente adequadas para ajudar na compreensão da forma em que algoritmos e programas complexos se comportam em tempo de execução.

Já, Visual Insights [FYO97], é um sistema de visualização de *software* que exibe o código fonte, bem como os arquivos e módulos que compõem o sistema em análise.

3.2.1 BALSA [BRO84]

Brown University Algorithm Simulador and Animator, mais conhecido como BALSA, é um animador e simulador de execuções de algoritmos. Foi desenvolvido quase que completamente em C, apenas a parte de animação foi desenvolvida em Pascal. Ele permite mostrar múltiplas visões de algoritmos em execução simultaneamente, bem como múltiplas visões da mesma estrutura de dados. Além disso, foi o primeiro sistema interativo que implementou a disputa entre algoritmos em execução na mesma tela.

BALSA foi mais do que um protótipo: ele foi utilizado por diversos anos como uma ferramenta de auxílio para os estudantes de graduação e, também, para o desenvolvimento de projetos e análises de algoritmos.

3.2.2 Zeus [PRI94]

Zeus foi a última evolução de BALSA, um sistema escrito em Modula-3 para estações DEC. Como BALSA, Zeus suporta múltiplas visões sincronizadas e permite ao usuário editar essas visões. Zeus também é muito útil para projetos orientados a objetos e especificações gráficas de visões. Pelo fato de ter sido implementado em um ambiente multi-tarefa e multi-processável, Zeus pode facilmente animar programas paralelos. Como Zeus é um protótipo, ele tem sido limitado para ser utilizado fora do laboratório, porém está sendo utilizado como uma ferramenta para desenvolvimento de algoritmos por pesquisadores.

3.2.3 TANGO [STA90]

Também desenvolvido na Brown University, é um *framework* de animação de algoritmos que incorpora facilmente transações homogêneas em qualquer animação de algoritmo. TANGO é composto por três partes: definição de operações e eventos no programa que comanda a animação, projeção da animação para simular as abstrações e operações, e o mapeamento das operações dos programas para as cenas de animação correspondente. A primeira parte envolve a inserção de chamadas de operações de algoritmos no código fonte, a qual pode ser feita manualmente ou com um editor especial que não modifica o código fonte.

A parte de projetar a animação do *framework* introduz quatro tipos abstratos de dados para definir transações. Esses tipos podem definir a trajetória ou mudança no tamanho, cores, ou visibilidade.

O terceiro componente do *framework* é o mapeamento dos códigos dos programas e suas execuções para a animação. BALSA permite apenas mapeamento um-para-um entre eventos no programa e as cenas de animação, enquanto que, TANGO permite utilizar um *framework* formal que suporta mapeamentos um-para-muitos, muitos-para-um e também muitos-para-muitos. Assim os mesmos componentes de animação podem ser referenciados repetidamente por diferentes operações de algoritmos em um programa.

Para animar um programa em TANGO, o visualizador deveria primeiro anotar o código fonte em C com chamadas de operações de algoritmos. Depois, as cenas de animação são projetadas.

3.2.4 Visual Insights [FYO97]

O visualizador de código *Visual Insights* permite visualizar grande quantidade de código fonte em uma única tela. A figura 3.1 mostra como o código de um sistema é visualizado pelo *Visual Insights*. A visualização consiste de dezenas de arquivos e milhares de linhas de código em uma representação reduzida. Os retângulos verticais representam os arquivos , e as linhas horizontais dentro de cada retângulo representam as linhas de código. As linhas que são modificadas mais recentemente são vermelhas, e as mais antigas são azuis e a escala de cores indica o estado intermediário. O visualizador de código *Visual Insigths* ajuda ver arquivos, módulos, e até mesmo entrar em subsistemas, e entender como uma troca em um arquivo fonte pode afetar outros arquivos em um subsistema.

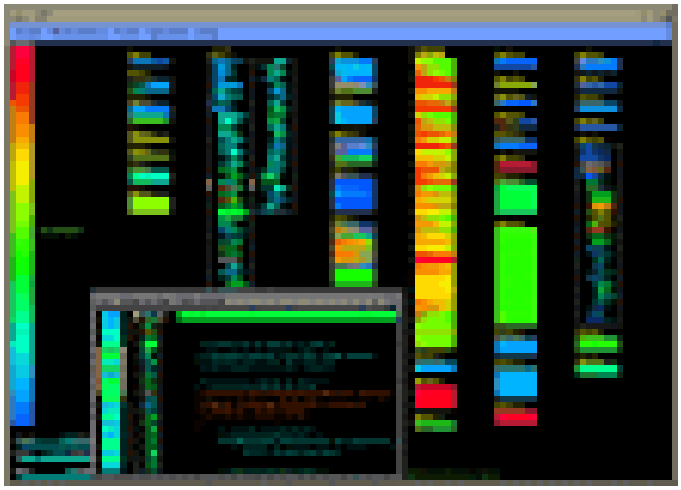


FIGURA 3.1 - Visualizador de código *Visual Insights* [FYO97]

3.3 Recuperação e Extração de Informações

As áreas de recuperação e extração de informações são duas áreas essenciais para a visualização de informações, visto que, para poder exibir informações, as mesmas devem ser recuperadas ou extraídas da estrutura em que estão representadas, sejam elas estruturadas ou não.

No caso de visualização de *software*, para possibilitar a visualização das informações, é necessário, primeiramente extrair as informações relevantes contidas nos programas fonte, para depois fazer a visualização das mesmas.

Tanto a área de recuperação como a de extração de informações não são específicas para a visualização de *software,* por isso, as mesmas serão descritas a seguir mostrando como são recuperadas e extraídas as informações em estruturas variadas e não especificamente de programas.

A área de recuperação de informações procura coletar documentos e artigos com base em textos ou palavras, a partir de determinados padrões pré-estabelecidos [COW96]. As ferramentas de recuperação de informações, geralmente, trabalham com técnicas de indexação, capazes de identificar e acessar mais rapidamente documentos de um banco de dados textual [YAT96].

Existem três tipos de indexação:

•indexação tradicional: é aquela onde uma pessoa determina os termos descritivos ou caracterizadores dos documentos, os quais farão parte do índice de busca;
•indexação *full-text* (ou indexação do texto todo): onde todos os termos que compõem o documento fazem parte do índice;
•indexação por *tags* - marcas que possam ser identificadas no texto; geralmente sendo representadas por palavras-chave: onde apenas algumas partes do texto são escolhidas automaticamente para gerar as entradas no índice, somente aquelas consideradas mais importantes ou mais caracterizadoras.

O terceiro tipo de indexação, por *tags*, procura indexar somente as partes relevantes do documento. Para tanto, as ferramentas automatizadas deverão analisar cada documento, procurando por marcas - *tags* - que identificam estas partes mais importantes.

Nesta mesma linha, as ferramentas descritas em [HAR93] extraem palavras-chave com base em marcas da macroestrutura dos documentos. Há uma tabela de *tags* específica para cada tipo de arquivo, por exemplo: programas contêm procedures, funções, comentários, etc; mensagens de correio eletrônico contêm destinatário, remetente, assunto.

O maior problema das técnicas de indexação por *tags* é que os tipos de documentos devem ter sido analisados previamente, ou seja, já se deve conhecer que tipo de conteúdo compõe os documentos e quais as marcas que identificam as partes relevantes de um documento. Então, as ferramentas acabam sendo específicas para determinados tipos de informação e documentos.

Enquanto a área de recuperação de informações trata de achar textos e documentos relevantes, a área de extração de informações trata de encontrar informações relevantes em textos ou em dados não estruturados, transformando-as em informações estruturadas para que sejam melhor analisadas.

As técnicas de extração de informação transformam dados não estruturados em informações explícitas. Elas isolam partes relevantes do texto, extraem informação destas partes, transformando-as em informações mais digeridas e melhor analisadas [COW96].

As principais técnicas de extração de informação reconhecem as estruturas de um texto, os dados ou informações, através da análise de *tags*. Cowie e Lehnert [COW96] citam como exemplos o reconhecimento de nomes próprios (como nomes de pessoas, companhias, etc), por começarem com uma letra maiúscula e por, geralmente, virem próximos de termos como "Senhor", "limitada", etc., ou então comandos de uma linguagem.

Gramáticas regulares, *parsers* parciais e autômatos finitos podem ser utilizados para o reconhecimento das informações através de análise léxico-sintática. Entretanto, algumas técnicas podem fazer uso de técnicas mais complexas como análise semântica.

3.4 Requisitos de uma Ferramenta de Visualização de S*oftware*

Técnicas e ferramentas de visualização permitem aos cientistas explorarem seus dados de pesquisa, permitindo assim, a comunicação de suas descobertas a outras pessoas. A visualização permite ainda, a conversão da informação que não pode ser percebida pelo olho humano em formas satisfatórias para as pessoas.

Para escolher uma ferramenta de visualização deve-se levar em consideração um conjunto de requisitos. Considerando visualização em geral, Mukerjea e Stasko [MUK95] comentam que a ferramenta deve ser a mais completa possível, contemplando:

1. *Manipulação de Dados* - as tarefas de manipulação de dados envolvem a seleção de porções de dados ou a transformação de dados em novas formas mais simples. Utiliza-se essas

tarefas para reduzir a complexidade do espaço da informação. Uma vez que a visualização e a compreensão de um espaço de informação grande e complexo é difícil, os módulos de manipulação de dados necessitam ser integrados com a visualização de dados.

2. *Análise de dados* - as tarefas de análise de dados envolvem a obtenção de estatísticas sobre porções dos dados, proporcionando uma boa percepção do espaço da informação. Essa análise é útil não somente para dados numéricos, mas para informações mais abstratas também.

3. *Visualização dos dados* - as tarefas de visualização dos dados envolvem obter representações apropriadas dos dados através de representações apropriadas. Além dos diagramas de barras e gráficos tradicionais, métodos de visualização alternativos devem ser desenvolvidos.

Portanto, uma ferramenta de visualização de *software* deve permitir ao usuário executar várias manipulações de dados e tarefas de análise, bem como permitir a eles desenvolvem tipos de visualização para suas informações. E também, prover módulos para essas tarefas, permitindo ainda, a interação entre elas. Por exemplo, depois de fazer uma análise sobre os dados, pode-se reduzir o número de informações para depois serem visualizadas. Assim, filtragem de dados seria uma tarefa e a outra tarefa seria a visualização propriamente dita. Essas tarefas devem interagir uma com a outra.

Tem-se ainda outros requisitos importantes para ressaltar, conforme levantamento realizado por Estivalet [EST00].

1. Permitir o controle de usuários (experientes) sobre as propriedades visuais:

As visualizações para serem efetivas devem utilizar propriedades visuais para representar as informações. Essa estratégia tem sido utilizada efetivamente nos sistemas de Visualização Científica e de *Software* como o Glyphmaker [RIB94] e Document Lens [ROB 93]. A quantidade de informação é, geralmente, muito maior do que a quantidade de propriedades visuais que podem ser usadas, por isso, o projetista não deve, a priori, fixar ligações entre as informações e as propriedades visuais, já que ele pode não saber quais informações serão úteis para os usuários. Já os usuários devem estar aptos a controlar as ligações dinamicamente e adequar as visualizações de acordo com suas necessidades.

2. Visualizações simultâneas:

É bastante difícil produzir uma visualização efetiva de espaços da informação grandes e complicados que seja compreensível para o usuário. Por esse motivo, produzir diferentes visualizações do mesmo espaço de informação, com diferentes perspectivas pode ser mais útil, pois torna as visualizações mais fáceis e mais compreensíveis, pelo fato dessas vistas destacarem somente certos aspectos do espaço. [NEU987] também observou que a habilidade para ver o conhecimento de diferentes perspectivas é importante.

3. Prover detalhe com o contexto:

Geralmente, o espaço da informação que é visualizado, é muito grande para ser mostrado numa única tela e, se o tamanho é reduzido, os detalhes se tornam muito pequenos para serem

vistos. Assim, uma alternativa seria utilizar barras de rolagem para visualizar, mas isso tende a ocultar a estrutura global. O objetivo seria mostrar os detalhes e o contexto, ambos integrados num único espaço, pois esse é um dos maiores requerimentos para uma visualização de *software* efetiva. As diversas técnicas de visualização do tipo foco+contexto [FUR 86] [SAR 92] [SAR 94] [LAM 95] podem utilizar formas de distorção como uma lente "olho de peixe" para proverem informação detalhada sobre um aspecto dos dados, sem perder a visão geral.

6. Facilidade de uso:

Diferentes combinações de operações de exploração de dados serão necessárias dependendo dos objetivos do usuário. O usuário deve ser apto a experimentar várias opções para decidir qual é a melhor visualização que se adequa aos seus interesses. Conseqüentemente, se é muito difícil ou se consome muito tempo para experimentar diferentes operações de exploração de dados, o uso da ferramenta será limitado. Por esse motivo, a facilidade de uso é um dos maiores requisitos para ferramentas de visualização.

7. Extensibilidade:

É impossível conseguir predizer todas as explorações de dados necessárias para o usuário. Assim, mesmo que vários módulos de exploração de dados sejam oferecidos, para alguma tarefa imprevista alguns outros módulos podem vir a ser necessários. Por exemplo, uma tarefa de exploração de dados pode requerer um novo modo de visualização de dados ou uma nova operação de manipulação de dados. Se a ferramenta não puder ser utilizada para essas tarefas, ela vai ter seu uso limitado. Por isso, a ferramenta deve ser extensível, ou seja, ela deve permitir a adição de código apropriado para realizar novas funções requeridas.

3.5 Comentários Finais

O objetivo da Visualização de Informações é realmente a percepção da informação. Trata-se da tarefa de assimilar a informação através do sistema perceptivo para o sistema cognitivo. Isso significa que a informação deve ser representada de maneira que o usuário possa rapidamente distinguir características relevantes. Se a informação pode ser fornecida tal qual o sistema perceptivo pode processá-la, pode-se então, acelerar a tarefa de assimilação humana da informação em diversas ordens de magnitude.

Para isso, deve-se apresentar a informação utilizando características que possam ser distingüidas rapidamente. Essas características incluem cor, tamanho, forma, localização/posição, entre outras.

As pessoas podem reconhecer a configuração espacial de elementos em uma figura e perceber rapidamente relacionamentos entre os mesmos. Esse sistema visual altamente

desenvolvido significa que as pessoas podem compreender o conteúdo de uma figura mais rapidamente do que elas podem reproduzir e entender um texto.

Além disso, verificou-se que a visualização de informações oferece alguns meios que outras interfaces não oferecem. São eles: um meio de ver tendências nos dados, um meio de identificar agrupamentos nos dados, um meio de encontrar estrutura em informações heterogêneas, um meio de ver uma quantidade enorme de dados mostrados em uma única tela e um meio de ver um item particular de interesse dentro do contexto de uma quantidade enorme de informações.

Pode-se concluir ainda, que a representação deve satisfazer tanto a sofisticação visual como as preferências dos usuários e que os resultados devem ser exibidos garantindo a atenção (do usuário) adequada ao processo de percepção. Um grande objetivo é permitir a criação de visualizações mais eficientes que tornem os usuários hábeis para perceberem a informação rapidamente, procurando sempre representar os dados e as informações fielmente, ou seja, minimizar ao máximo a perda de informações.

Considerando que a definição das técnicas de visualização importantes para visualização de software deve ser feita em função das tarefas a serem realizadas pelos usuários, optou-se por realizar um estudo de caso que é relatado no próximo capítulo. Dessa forma, embasou-se a determinação de quais requisitos específicos deveriam ser contemplados na ferramenta a ser desenvolvida a partir desse estudo de caso.

4 Estudo de Caso

4.1 Introdução

O sistema que foi utilizado como estudo de caso para o desenvolvimento do visualizador é o sistema "Benfare", totalmente desenvolvido em *Progress* pela empresa Benfare Informática[1], a qual possui clientes em diversos estados brasileiros.

O produto "Benfare" é um sistema integrado reunindo recursos capazes de otimizar processos existentes dentro de uma organização. Ele foi desenvolvido para atender as necessidades gerenciais de empresas em geral e o seu conteúdo contempla técnicas de gerenciamento de negócios. A proposta de trabalho da empresa é fazer uma parceria com o cliente onde a empresa possa adequar os sistemas Benfare à empresa parceira.

Cada projeto passa pelas seguintes fases:

1. PDI (Plano Diretor de Informática - levantamento das reais necessidades do cliente);
2. Personalização;
3. Instalação dos programas;
4. Conversão de todos os dados cadastrais, históricos e movimentações que a empresa dispõe em seu sistema atual;
5. Treinamento;
6. Implantação.

O sistema "Benfare" está preparado para suportar diversos segmentos da empresa-cliente, nas seguintes gestões:

- Recursos Humanos
- Transportes
- Administrativa
- Comercial
- Construção Civil

As gestões são compostas por módulos que podem ser implantados gradativamente, com integração absoluta, e cada módulo é composto de diversos programas.

[1]A empresa Benfare, nome fantasia, Benfare Informática LTDA, razão social, é uma empresa brasileira, com sede em Caxias do Sul – RS, que iniciou suas atividades em 1984.

Os programas que compõem o sistema "Benfare" consistem basicamente de manutenções (cadastros), cálculos, consultas e relatórios, e podem ser independentes, ou seja, não dependerem de nenhum outro programa para serem executados, ou podem ser dependentes entre si, ou seja,

um programa não é executado pelo menu diretamente e sua execução é disparada por outro programa.

4.2 Características gerais da linguagem *Progress*

A linguagem *Progress* foi desenvolvida pela *Progress Software Corporation*[1], em 1984 inicialmente para sistemas UNIX para uso em *mainframes* e processamento em grandes volumes de dados. A *Progress* tem sua sede em Bedford, Massachusetts, USA, com filiais em diversos países. No Brasil, sua representante é a *Progress* do Brasil em São Paulo[2].

Progress é uma linguagem de 4ª geração com diversos recursos que a tornam bastante poderosa para qualquer tipo de desenvolvimento de sistema para o setor empresarial. Uma das maiores vantagens da linguagem é a sua portabilidade, pois ela funciona em praticamente todos os sistemas operacionais existentes, DOS, Windows 3x, 95, NT, Unix, OS/2, Netware, VAX/VMS, Xenix, Edix-5, Ultrix, Aix, CTOS/BTOS. Com isso, em ampliação de máquina ou mesmo na sua substituição, a conversão dos dados e dos programas praticamente não existe – utiliza-se o mesmo código fonte.

Progress está disponível através de um ambiente para criar aplicações, onde se caracteriza aplicação como uma solução computadorizada para qualquer tipo de problema. Normalmente, essas aplicações podem representar problemas simples de solucionar, mas que podem ser difíceis de desenvolver. Progress facilita o desenvolvimento das aplicações e o acesso aos dados pela utilização de um banco de dados relacional e uma linguagem de 4ª geração para manipulação dos dados, cálculos e relatórios.

Cinco componentes podem ser usados para manipular a informação numa aplicação Progress:

- Banco de Dados Relacional;
- Dicionário de Dados;
- Linguagem de Aplicação;
- Editor de Procedimentos;
- Ferramenta de geração de relatórios e telas.

No caso da Benfare, a versão utilizada para escrever os programas depende da versão que os clientes possuem de *Progress,* pois, as versões mais atuais possuem recursos que não estão disponíveis nas versões anteriores. Assim, a empresa se detém nos recursos oferecidos pela versão do cliente. As versões que se tem disponíveis e em uso atualmente são: Progress 6 e 7 versão caracter, Progress 7, 8 e 9, versão gráfica.

[1]http://www.progress.com [2]http://www.progress.com/psc_companies/index.htm

Os programas do sistema "Benfare", como dito anteriormente, são todos escritos em *Progress*, uma linguagem estruturada que possui uma sintaxe bem específica. Além de comandos de controle de fluxo e expressões aritméticas, relacionais e lógicas comuns às linguagens de programação, *Progress* oferece um conjunto de comandos para acesso a banco de dados. A figura 4.1 apresenta um exemplo de um programa escrito em *Progress*.

FIGURA 4.1 - Exemplo de um programa em *Progress*

4.3 Gestões

O *software* "Benfare" apresenta algumas características:

- Flexível: adaptável às necessidades de cada cliente;
- Inteiramente seguro: controle de acesso por senhas;
- Corporativo: multi-empresas e multi-estabelecimentos, com indexação de várias moedas;
- Definido com o máximo de parametrização possível;

- Amigável: *Help on line* onde o usuário pode interagir alterando ou completando o manual;
- Ágil, preciso e fácil de operar;
- Pode ser totalmente integrado;
- Possui uma facilidade na criação de novas aplicações;
- Acessa outros bancos de dados com total transparência;
- Otimiza a utilização do espaço de armazenamento;
- Alta performance.

I – Gestão de Recursos Humanos

A **Gestão de Recursos Humanos** é um *software* modular, totalmente personalizado aos critérios do usuário, que pretende atender todas as necessidades legais e gerenciais da área de RH e colaborar na certificação da qualidade das empresas.

O tratamento do apontamento de horas, bem como, banco de horas, cálculo automático de pensão judicial e fórmulas de cálculo totalmente parametrizadas para a descentralização do usuário são alguns dos diversos recursos do aplicativo.

Os módulos ***Folha de Pagamento***, ***Medicina e Segurança***, ***Treinamento***, ***Cargos e Salários*** e ***Recrutamento e Seleção***, que compõem a Gestão de RH, podem ser utilizados separadamente ou totalmente integrados. Além disso, é possível a integração com módulos da área Administrativa, Comercial, da Construção Civil e de Transporte.

II – Gestão de Transporte

A **Gestão de Transporte** é um sistema modular, integrado que atende transportadoras de qualquer porte, nos mais diferentes tipos de cargas e formas de transporte (rodoviário, aéreo ou fluvial).

O aplicativo faz o controle efetivo de todo o negócio de Transporte, desde a parte documentacional, cálculo de frete com suas peculiaridades, controle da coleta e entrega, gerenciamento da frota, até a integração com outros módulos da área Administrativa, Comercial e de Recursos Humanos.

Os módulos da Gestão de Transporte são ***Controle de Frete***, ***Coleta e Entrega*** e ***Controle de Frota***.

III – Gestão Comercial

A Gestão Comercial tem o ***Crediário*** como módulo básico e completa-se com a integração desse sistema aos módulos ***Compras***, ***Estoque***, ***Faturamento***, ***Contas a Pagar***, ***Contas a Receber***, ***Livros Fiscais***, ***Caixa e Bancos***, ***Darf e Capital de Giro***, ***Contabilidade*** e ***Patrimônio*** que fazem parte da Gestão Administrativa. Ainda é possível a integração com módulos da área de Recursos Humanos, de Transporte e da Construção.

IV – Gestão Administrativa

Descrição resumida:

A Gestão Administrativa é composta pelos módulos ***Patrimônio***, ***Contabilidade***, ***Compras***, ***Estoque***, ***Faturamento***, ***Contas a Pagar***, ***Contas a Receber***, ***Livros Fiscais***, ***Caixa e Bancos*** e ***Darf e Capital de Giro*** que integrados podem fornecer informações dos mais diversos setores de uma organização. Ainda é possível a integração com módulos da área de Recursos Humanos, de Transporte e da Construção Civil.

V – Gestão da Construção Civil

A **Gestão da Construção Civil** possui recursos capazes de gerenciar todo o negócio da Construção Civil desde o projeto, o controle de todas as etapas de cada obra, o acompanhamento da situação dos imóveis com suas particularidades e as devidas relações com os clientes/proprietários, até a finalização.

Os módulos da Construção Civil são ***Edificação, Contas a Pagar***, ***Contas a Receber*** e ***Caixa e Bancos*** que permitem total integração com módulos de outras áreas da empresa.

A tabela 4.1 sumariza a composição do sistema Benfare, indicando o número de programas e tabelas de cada módulo.

TABELA 4.1 - Componentes do sistema Benfare – dados levantados em abril de 2001

Módulos	**Número de Programas**	**Número de Tabelas**
Folha de Pagamento	1987	130
Medicina e Segurança	293	87
Treinamento	280	64
Cargos e Salários	27	8
Recrutamento e Seleção	38	9
Controle de Frota	240	47
Controle de Frete	580	65
Coleta e Entrega	52	7
Crediário/Compras	50	8
Estoque	223	5
Faturamento/Contas a Receber	290	47
Contas a Pagar	154	17
Livros Fiscais	5	2
Caixas e Bancos/Darf/Capital de Giro	44	6
Contabilidade	90	15
Patrimônio	35	15
Edificação	20	5

Todos os módulos citados acima são integrados ao módulo ***Estabelecimento.*** Nesse módulo ficam armazenados todos os dados das empresas e filiais. Este módulo não é muito complexo, mas é grande, possuindo 450 programas e 25 tabelas.

O tamanho dos programas que compõem o sistema Benfare é muito diversificado dependendo da complexidade do mesmo, variando de 17 linhas e um máximo de 2893.

Além disso, existem outros componentes que são:

- *Macros*: trechos de programas que são chamados no meio de um programa e que não precisam ser compiladas, pois são apenas trechos que, em tempo de execução, são copiados para dentro do programa, como se fizessem parte do código. As macros existentes no sistema "Benfare" são padrões para todos os módulos existentes, não existindo dessa forma, nenhuma macro específica para um determinado módulo. O sistema "Benfare" possui atualmente 900 macros.

- *Zooms*: são programas inteiros e não somente trechos, por isso são compilados como qualquer outro programa. Os *Zooms* tem a função semelhante dos componentes tipo combo-box, onde um determinado campo possui esse recurso para facilitar a consulta, alteração, exclusão e até mesmo inclusão de registros. Quando acionados – no caso dos programas "Benfare" é a tecla F5 – mostram os campos dos registros já cadastrados, permitindo selecionar o desejado. Assim o usuário não precisa memorizar e nem possuir uma tabela com campos de registros que já foram cadastrados. Existem 550 programas desse tipo, todos disponíveis para qualquer módulo.

- *Variáveis Compartilhadas:* são variáveis globais, criadas por um determinado programa, porém podem ser utilizadas por qualquer outro, precisando apenas declará-las como tal. Essas variáveis são muito utilizadas e muitas vezes substituem a utilização de parâmetros. Atualmente o sistema tem 230 variáveis compartilhadas.

4.4 Metodologia de desenvolvimento e manutenção utilizada pela equipe

O sistema "Benfare" possui como característica forte o fato de ser personalizável. A empresa não vende sistemas fechados, ela possui um sistema padrão, o qual é utilizado apenas nas demonstrações. Vendido o produto, o sistema é personalizado de acordo com as necessidades do cliente.

A empresa possui atualmente em torno de 200 clientes em diversos estados brasileiros, por isso precisa de um controle bastante rígido na organização das versões de cada cliente. Alguns programas, não sendo específicos de cada cliente, são mantidos em versão única. A solicitação de uma modificação em algum programa pode acarretar a criação de uma nova versão para o mesmo, se ele for de uso comum e a modificação for específica para o cliente.

A maioria dos clientes está conectada com a empresa através de modem usando um sistema de transmissão de dados *PCAnywhere*, e em alguns casos de versões caracteres, pelo

QuickLink. Após a empresa ter alterado e/ou desenvolvido o programa conforme solicitação, ela o envia ao cliente. A partir daí o cliente está liberado para utilizar o que solicitou.

Essas conexões servem também, muitas vezes, para testar os programas, pois a empresa não possui o banco de dados totalmente completo e atualizado de todas as empresas, precisando assim muitas vezes conecta-lo para poder fazer uma determinada solicitação.

Os clientes que não possuem conexão direta com a empresa, correspondentes à minoria, são atendidos quando possível pela Internet, caso contrário por visita pessoal.

Segue abaixo uma figura que ilustra a metodologia da empresa.

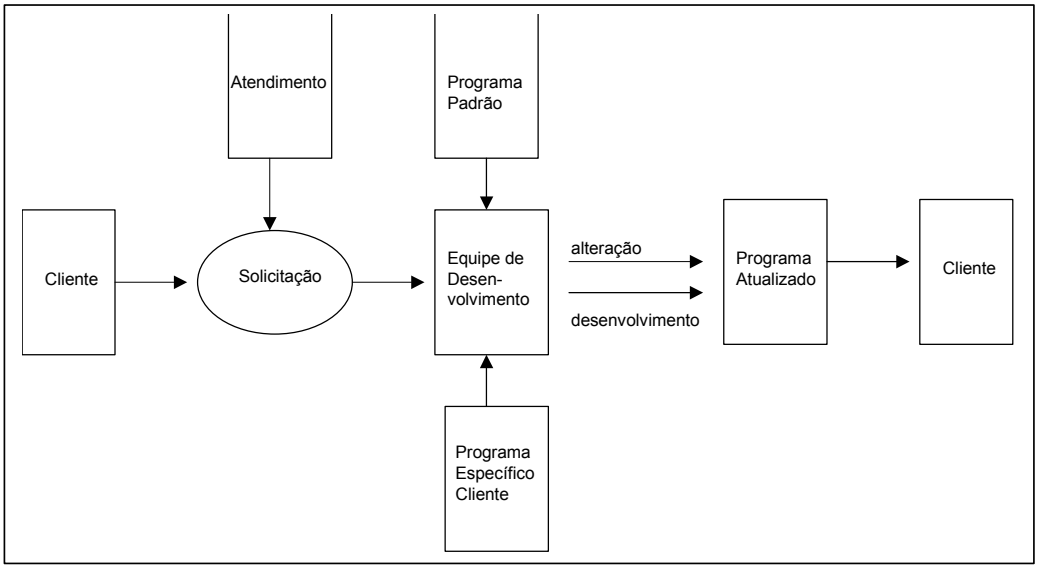


FIGURA 4.2 - Metodologia de desenvolvimento e manutenção de *software* da Benfare Informática

4.5 Necessidades Identificadas

Com o estudo de caso, pode-se verificar que o sistema Benfare é um sistema extremamente grande e complexo, do ponto de vista da equipe de desenvolvimento. Por isso, surgiu a necessidade de se otimizar tanto o desenvolvimento, quanto a manutenção desse conjunto imenso de programas. Foi a partir disso, que surgiu a idéia de desenvolver um visualizador de informações, o qual auxiliasse a equipe de desenvolvimento a obter informações relevantes do sistema de forma mais rápida e precisa, sem precisar percorrer milhares de linhas de código.

Os programas são atualizados freqüentemente e novos programas são desenvolvidos diariamente, por isso, a necessidade ainda maior de se ter uma ferramenta que auxilie na compreensão dos programas, bem como do sistema como um todo, visto que para realizar essas tarefas, a equipe de desenvolvimento necessita ter conhecimento de uma série de informações que não estão explícitas nos programas.

Os programas apresentam diversas informações que poderiam ser visualizadas, porém seria inviável conseguir discriminar todas e fazer a visualização das mesmas, pois além de ocuparem bastante espaço de armazenamento, isso tornaria o visualizador bastante lento e complexo, devido a quantidade de informações existentes nos programas. Por isso, verificou-se as informações mais necessárias para a equipe de desenvolvimento, principalmente aquelas que são buscadas com maior freqüência. São elas: data da última alteração de um programa, variáveis locais, varáveis globais, entidades, macros, chamadas de programas, procedimentos internos, módulos, número de linhas, tipo de programa (se manutenção ou relatório), *buffer's* e tabelas temporárias criadas no próprio programa.

Conclui-se com isso, que tem-se necessidades de visualização de diferentes tipos:

- Informações textuais e numéricas de gerência do sistema;
- Relacionamentos entre módulos e programas;
- Uso de variáveis globais por diferentes programas.

No primeiro caso, as informações devem ser mostradas basicamente em sua forma numérica e textual, enquanto nos últimos dois casos, a solução é a visualização em forma de diagrama, recaindo em visualização de grafos.

Estas informações devem ser coletadas diretamente através da análise dos programas fonte, justificando com isso, o estudo realizado das técnicas de extração de informações (seção 3.3).

O próximo capítulo descreve o protótipo implementado, com as soluções adotadas para busca e visualização das informações mencionadas acima.

5 O Visualizador *VisProgress*

O ***VisProgress*** é um protótipo de um visualizador de *software* que faz a visualização de informações contidas em programas desenvolvidos em *Progress*. Esse visualizador mostra as seguintes informações: nome do programa, data da última alteração do programa, variáveis locais e globais, entidades, macros, chamadas de programas, procedimentos internos, módulos, número de linhas, tipo de programa – se manutenção ou relatório, *buffers* e tabelas temporárias criadas no próprio programa. Além dessas informações, ele exibe ainda dois diagramas, um representando as chamadas de programas e outro o compartilhamento de variáveis.

Apesar do visualizador ter sido desenvolvido para uma empresa específica, um de seus módulos é genérico e pode ser usado para quaisquer outras linguagens, uma vez que para adaptá-lo a outro sistema, basta alterar o gerador do arquivo descritor.

5.1 Arquitetura

Como outros visualizadores de informações [FYO97][CHU98] há necessidade de utilizar um arquivo descritor que contém as informações relevantes dos programas e que devem ser visualizadas. Assim, o visualizador foi desenvolvido em duas etapas:

- Criação do arquivo descritor do sistema;
- Visualização das informações contidas no arquivo descritor.

A visualização, por sua vez, resulta na exibição de informações numéricas e textuais a respeito de programa e na exibição de diagramas representando relações de chamada entre programas e de uso de variáveis. Para a visualização dos diagramas optou-se por utilizar a ferramenta *Dotty* [KOU96] que exibe grafos descritos numa linguagem específica. Assim, (fig. 5.1) num primeiro momento, *VisProgress* percorre o código fonte dos programas cujas informações serão visualizadas, e monta estruturas de dados, listas duplamente encadeadas, com as informações relevantes (seção 4.5, seção 5.2) contidas nos mesmos. Num segundo momento, *VisProgress* cria um arquivo descritor com as informações contidas nas estruturas de dados. Num terceiro e último momento, é criada a representação gráfica das informações contidas no arquivo descritor, tanto das informações textuais como os descritores para *Dotty*.

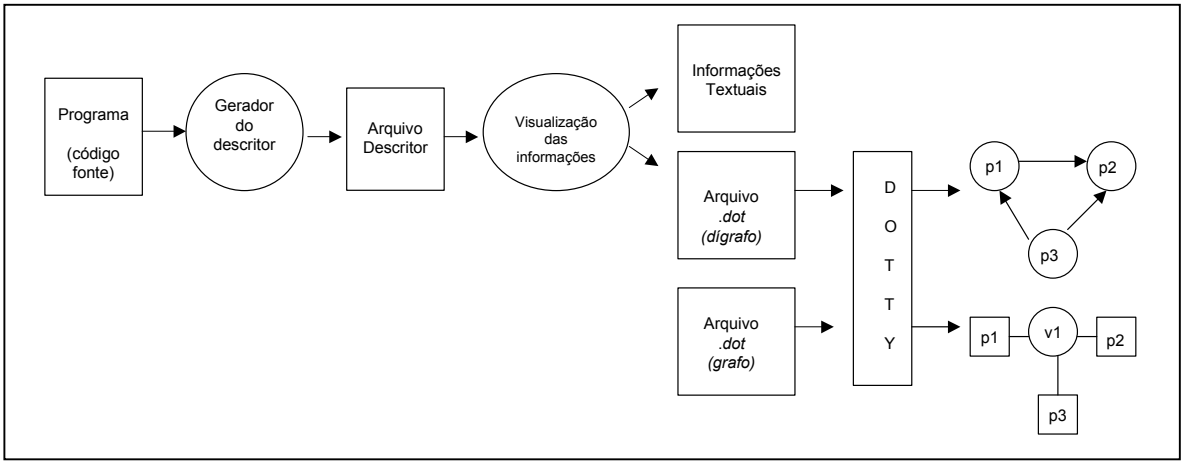


FIGURA 5.1 - Arquitetura do visualizador.

*O **Gerador do Arquivo Descritor*** é o módulo que analisa o código fonte dos programas, cria as estruturas de dados, e a partir delas cria o arquivo descritor baseado em "tags". Foi desenvolvido em Pascal 7.0, tendo como base, as palavras reservadas da linguagem *Progress*, sendo assim específico para esta linguagem, uma vez que cada linguagem possui uma sintaxe própria.

O ***Visualizador*** tem dupla função: lê o arquivo descritor e cria os arquivos com extensão *.dot* para a visualização dos grafos, além de exibir outras informações específicas de cada programa, mostradas ao usuário de forma textual. Foi desenvolvido em Delphi 3 e, com a utilização da ferramenta *Dotty* abrange a visualização de um grafo que representa as chamadas dos programas, ou seja, um grafo no qual cada nodo corresponde a um programa e as ligações entre eles dependem de um programa chamar ou não um outro programa. Um segundo grafo que representa as variáveis que são compartilhadas, ou seja, que programas compartilham variáveis e quais. Esse grafo é formado por nodos que representam os programas que compartilham variáveis e nodos que representam as variáveis compartilhadas, onde as ligações entre eles existirão conforme o compartilhamento das variáveis.

Dotty [KOU96] é uma ferramenta para visualização, criação e edição de grafos que será detalhada no capítulo 6.

Observando a arquitetura do *VisProgress,* vemos que para adaptar esse visualizador a outro sistema de informação, será necessário alterar o programa que cria o arquivo descritor, uma vez que o programa que mostra as informações para o usuário e cria os arquivos *.dot* não se baseia na sintaxe da linguagem, mas no arquivo descritor, o qual possuirá os rótulos dos *tags* diferenciados para cada linguagem, bem como o número de *tags*, pois os mesmos dependem do número de informações que o programa apresenta.

5.2 Arquivo Descritor

Como é criado apenas um arquivo descritor para todos os programas, os mesmos são identificados dentro do arquivo descritor pelo *tag* <descritor> nome-do-programa</descritor> ou seja, essa linha permite identificar que as informações que se encontram abaixo dela, são as informações do programa discriminado entre os *tags*.

O arquivo descritor está estruturado com base em "*tags*", onde cada tipo de informação é armazenado numa linha e é precedida pelos marcadores "<nome da informação armazenada>" e finalizada por "</nome da informação armazenada >". As informações armazenadas no arquivo descritor são: nome do programa, data da última alteração do programa, variáveis locais, variáveis compartilhadas, entidades, macros, chamadas de programas, procedimentos internos, módulos, número de linhas, tipo do programa, se manutenção ou relatório, buffer's e tabelas temporárias criados no próprio programa.

Essas informações são representadas no arquivo descritor da seguinte forma:

a) **nome do programa:** <descritor> </descritor>
Nessa informação é armazenado o nome do arquivo (programa em *Progress)*, que o criador do arquivo descritor percorre para discriminar as informações.

b) **data** : <date> </date>
Essa informação é encontrada através do comentário **#DATA** presente no programa. Todos os programas em *Progress* desenvolvidos pela Benfare possuem esse comentário, e o mesmo é atualizado toda vez que algum componente da equipe de desenvolvimento fizer alguma alteração no mesmo.

c) **variáveis locais**: <variables> </variables>
Essas são encontradas no programa através do comando **def var**, onde os nomes após esse comando representam o nome das variáveis locais.

d) **variáveis compartilhadas**: <shared> </shared> e <new shared> </new shared>
Estão divididas em shared e new shared. O programa que apresenta o comando **def shared var** é o programa que apenas utiliza uma variável compartilhada criada por outro programa. Já o programa que apresenta o comando **def new shared var** é o programa que cria a variável compartilhada e a utiliza; então, os nomes que vierem após **def shared var** são colocados entre os tags <shared> </shared>, enquanto que os que vierem seguidos do comando **def new shared var** são colocados entre os tags <newshared> </newshared>.

e) **entidades:** <entity> </entity>
As entidades são encontradas através dos comandos **find** e **for each**, esses comandos são utilizados para encontrar registros de uma determinada entidade. O **for each**, percorre registro por registro daquela entidade e o comando **find** é utilizado para encontrar um determinado registro (único).

FIGURA 5.2 - Trechos de programas em *Progress* ilustrando os comandos *for each* e *find*

A figura 5.2 mostra dois exemplos onde o gerador do descritor identifica entidades. O primeiro exemplo, irá buscar todos os registros da tabela FPCAD, um por um. Já o segundo, irá buscar o registro da entidade FPCAD da empresa 001 com código 100, o qual, encontrará somente um registro. Esses dois comandos são os comandos para acessar as entidades, por isso, todas as entidades que estiverem sendo utilizadas nesses comandos são entidades do dicionário de dados que estão sendo acessadas por este programa.

f) **macros:** <macro> </macro>

As macros são encontradas no programa através dos marcadores { } (abre e fecha chaves) e são finalizadas com **.i** (isso determina uma macro).

g) **chamada de programas:** <program> </program> e
procedimentos internos: <procedure> </procedure>

Essas informações são identificadas através do comando **run**. Por isso, para se certificar que o nome que aparece depois do comando **run** é um programa ou um procedimento, *VisProgress* pega esse nome e verifica se não existe no programa uma linha que inicia com a palavra **procedure** seguida desse nome. Caso isto seja encontrado, o mesmo identifica que é um procedimento, caso contrário, é um programa.

h) **módulos:** <module> </module>

Essa informação é encontrada pelas duas iniciais dos nomes das entidades.
Por exemplo, tem-se a informação <entity> FPCAD, BGEMPR </entity>, a informação dos módulos será <module> FP, BG</module> pois são as iniciais das entidades acessadas. Caso as iniciais sejam iguais em mais de uma entidade, as mesmas são ignoradas, ficando armazenadas uma única vez.

i) **número de linhas:** <lines> </lines>

Essa informação é encontrada através da contagem de linha por linha. Quando o visualizador chega ao final do arquivo armazena essa informação.

j) **tipo de programa:** <type> </type>

O tipo do programa é uma informação encontrada no programa através da presença ou não da palavra **output**, caso essa palavra seja encontrada o visualizador *VisProgress* coloca a palavra "relatório" entre os *tags* <type> </type>, caso contrário, coloca a palavra "manutenção" entre os *tags*.

k) **buffers:** <buffer> </buffer>

Essa informação é encontrada através do comando **def buffer,** onde, se esse comando aparecer no programa, significa que o programa criou buffers, logo, os nomes que seguem este comando serão armazenados no descritor.

l) **tabelas temporárias:** <temp-table> </temp-table>

Essa informação representa as tabelas temporárias criadas pelo programa, sendo as mesmas encontradas através do comando **def temp-table.** Quando esse comando for encontrado no programa significa que foram criadas tabelas temporárias, e o nome que aparece seguido desse comando é a informação que será armazenada entre os *tags* citados acima.

Como foi dito anteriormente, o arquivo descritor é único para o sistema, isso porque seria inviável criar um arquivo para cada programa, visto que o sistema "Benfare" possui milhares de programas. Por esse motivo, os conjuntos de informações dos programas são separados pelo nome dos programas e dentro de cada programa por data, caracterizando um histórico de alterações de cada programa.

Sempre que a criação do arquivo descritor for solicitada, o visualizador *VisProgress* verifica a existência da linha com os *tags* <descritor> nome-do-programa </descritor> no arquivo descritor. Caso não exista, cria-se no final do arquivo um identificador para esse programa e abaixo, discrimina-se as informações obtidas nele. Caso exista, o visualizador concatena com arquivo já existente, separando as informações do programa pela data. Nesse caso, serão armazenadas as informações do mesmo programa novamente, só que agora com outra data. Não será gravado somente o que foi alterado, os *tags* serão todos repetidos, porém agora com as informações que o programa apresenta nessa data, mantendo-se, assim, registro das alterações realizadas no período entre as duas datas. Isso permite a visualização das alterações referentes às informações armazenadas que foram feitas em cada programa.

A figura a seguir ilustra o formato do arquivo descritor.

```
<descritor>fp100r.p</descritor>
<date>.............</date>
<shared>.............</shared>
...
...
...
<date>.............</date>
<shared>.............</shared>
...
...
<descritor>fp200r.p</descritor>
<date>...
...
...
```

FIGURA 5.3 - Formato do arquivo descritor.

A criação do arquivo descritor e sua atualização foram desenvolvidas nesse trabalho, porém, são independentes do sistema desenvolvido, o visualizador. Essa parte será controlada pela empresa, é ela definirá de quanto em quanto tempo o arquivo descritor deverá ser atualizado e quem ficará responsável por isso. Em conseqüência disso, a integridade das informações mostradas pelo visualizador depende da atualização do arquivo descritor.

5.3 Interface

Ao ser visualizado, o *VisProgress* primeiramente apresenta um menu (fig. 5.4) com as três opções, além da opção de sair do visualizador. O primeiro botão com rótulo "Informações", faz a visualização das informações de forma textual em outras janelas. O segundo botão, "Programas", faz a visualização do grafo de chamada de programas e o terceiro botão, "Variáveis", faz a visualização do grafo de compartilhamento de variáveis. A partir daí fica a critério do usuário escolher a opção desejada, visualizando assim graficamente as informações de seu interesse.

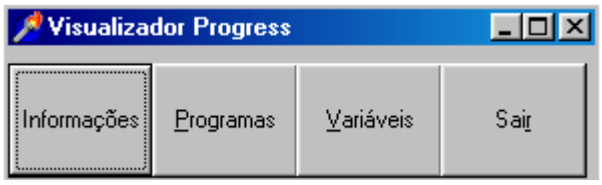


FIGURA 5.4 - Interface inicial do *VisProgress*

Conforme já mencionado, os dois grafos são visualizados pela ferramenta *Dotty*, um editor de grafos que permite a visualização, bem como a edição dos mesmos. Com o estudo realizado sobre essa ferramenta, verificou-se que ela é uma ferramenta específica para a visualização de grafos e que é adequada para fazer a visualização dos grafos criados pelo visualizador. Então, quando selecionada a opção de visualização de grafos, o *VisProgress* chama a ferramenta *Dotty* passando como parâmetro os arquivos .dot criados e a ferramenta apresenta

os grafos na tela. A visualização desses grafos através da ferramenta *Dotty* pode ser vista nas figuras 5.5 e 5.6. A figura 5.5 mostra um grafo de chamadas de programas, e a figura 5.6 mostra um grafo que representa o compartilhamento das variáveis.

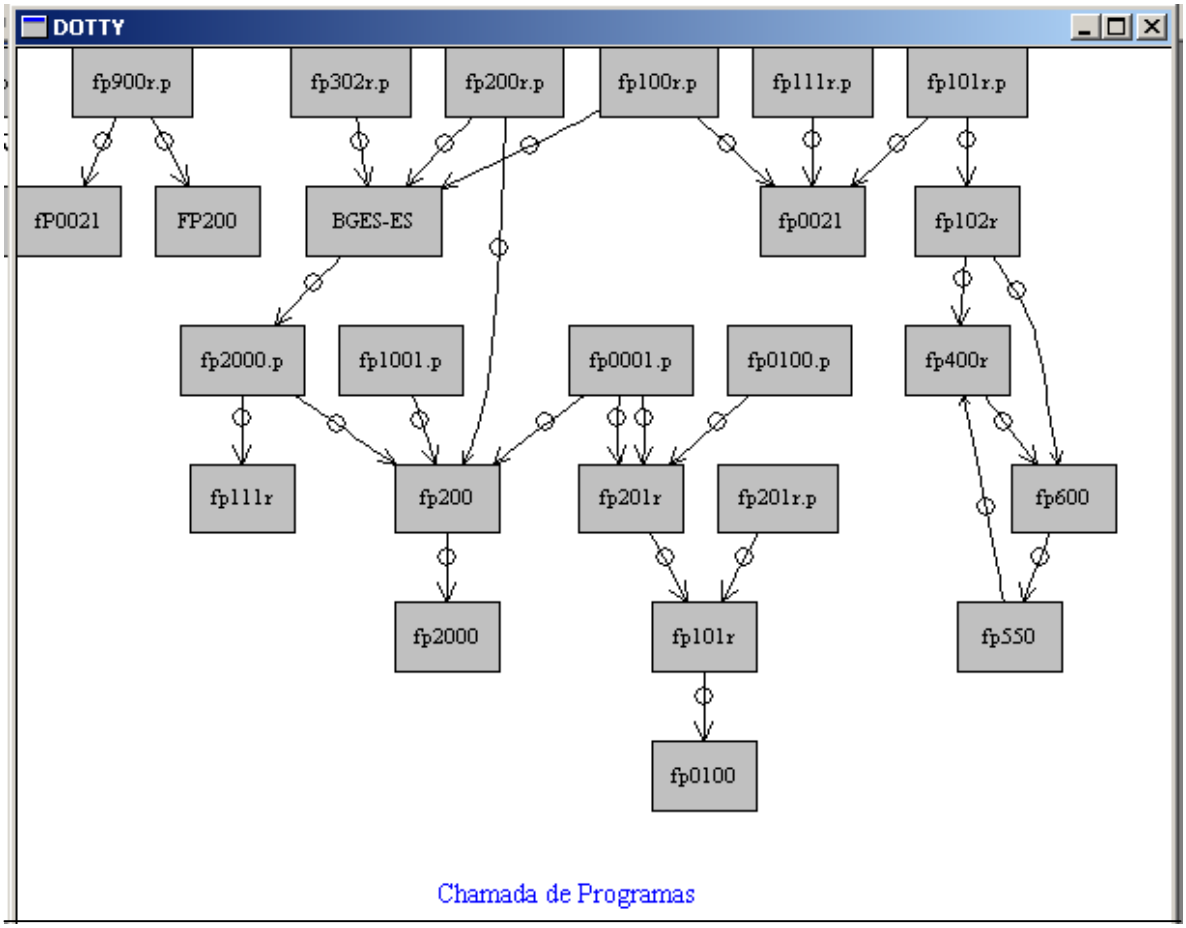


FIGURA 5.5 - Grafo de chamada de programas exibido através da ferramenta *Dotty*. Os retângulos representam os programas, não importando seu tipo (cadastro, relatório, cálculo).

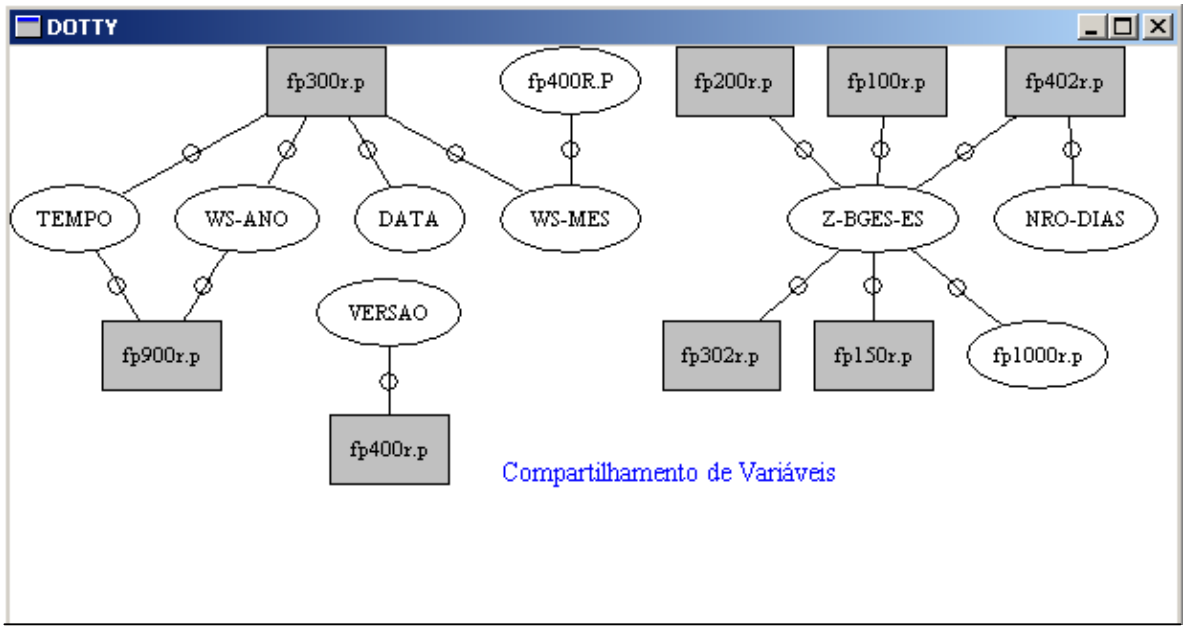


FIGURA 5.6 - Grafo de compartilhamento de variáveis exibido pela ferramenta *Dotty*. Os retângulos representam os programas e as elipses as variáveis compartilhadas pelos programas.

Com relação às informações textuais e numéricas o módulo foi totalmente desenvolvido em Delphi e permite que o usuário interaja com o sistema, apresentando um menu para escolher o programa cujas informações deseja visualizar, conforme figura 5.7.

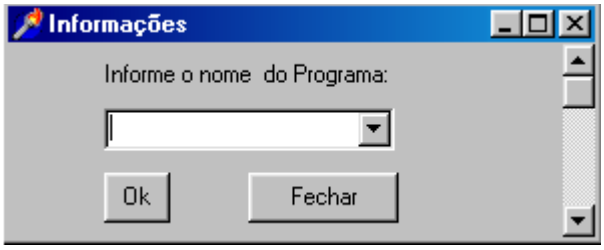


FIGURA 5.7 - Janela para escolher o programa que deseja visualizar as informações.

A partir daí, é apresentado um menu apenas com as informações que o programa que foi escolhido possui. Essa opção foi desenvolvida de maneira dinâmica, ou seja, o visualizador percorre o arquivo descritor e disponibiliza na tela para o usuário somente as informações que estão no arquivo descritor para o programa selecionado. O usuário deverá selecionar com *clicks* do mouse quais informações apresentadas na tela ele deseja visualizar. Escolhidas as informações, o visualizador exibe em tela ao lado, as informações conforme foram selecionadas, fazendo com que o usuário visualize apenas as informações que lhe são de interesse. Na figura 5.8, pode-se visualizar como são apresentadas essas informações textuais para o usuário.

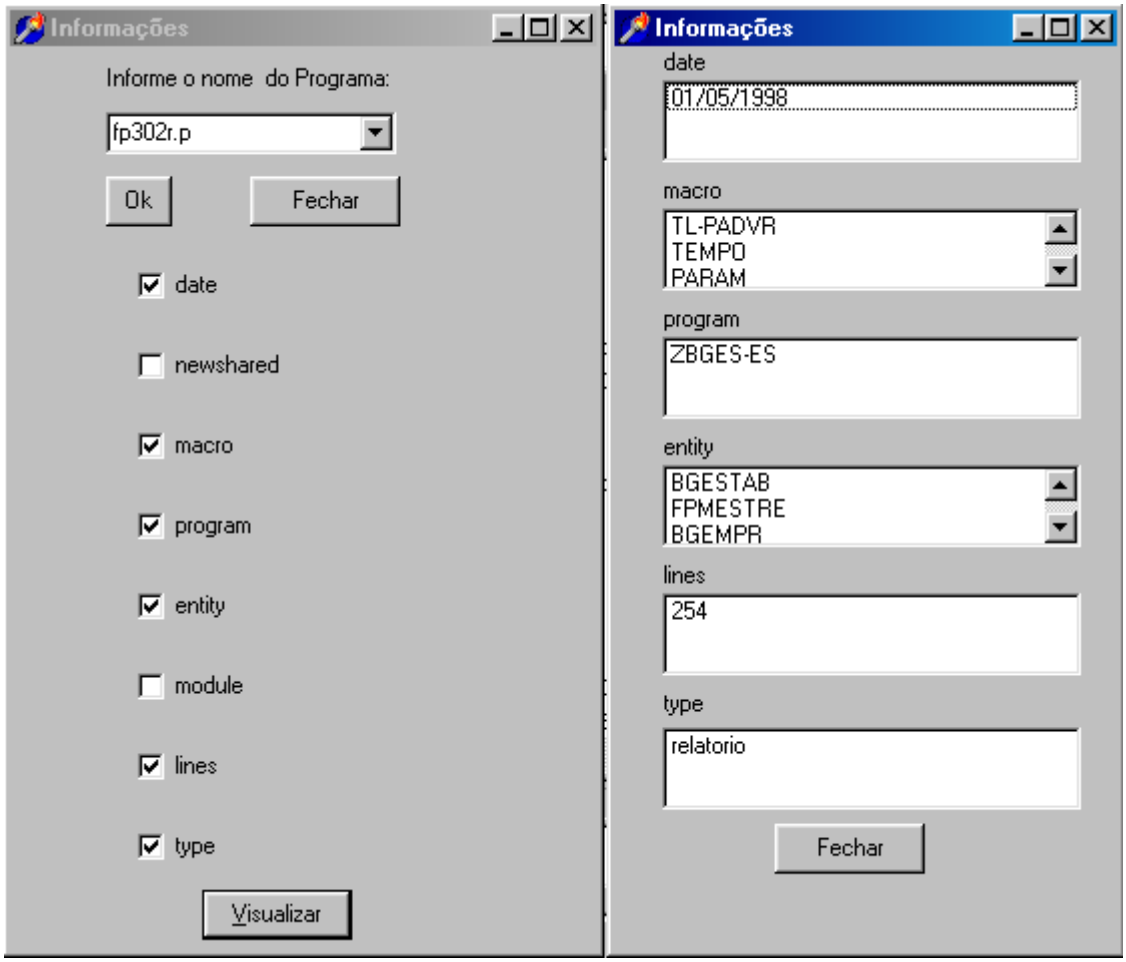


FIGURA 5.8 - Janela com as informações textuais: exemplo tem várias opções selecionadas.

No desenvolvimento dessa parte, é controlado o posicionamento e tamanho das janelas mostradas ao usuário, pois, como o número de informações varia para cada programa e o usuário seleciona o número de opções que desejar, cada janela é redimensionada conforme o número de informações que são mostradas. A figura 5.8 possui um exemplo de uma visualização, na qual o usuário selecionou quase todas informações disponíveis para a visualização. Já a figura 5.9 mostra como ficam as janelas no caso do usuário selecionar poucas opções.

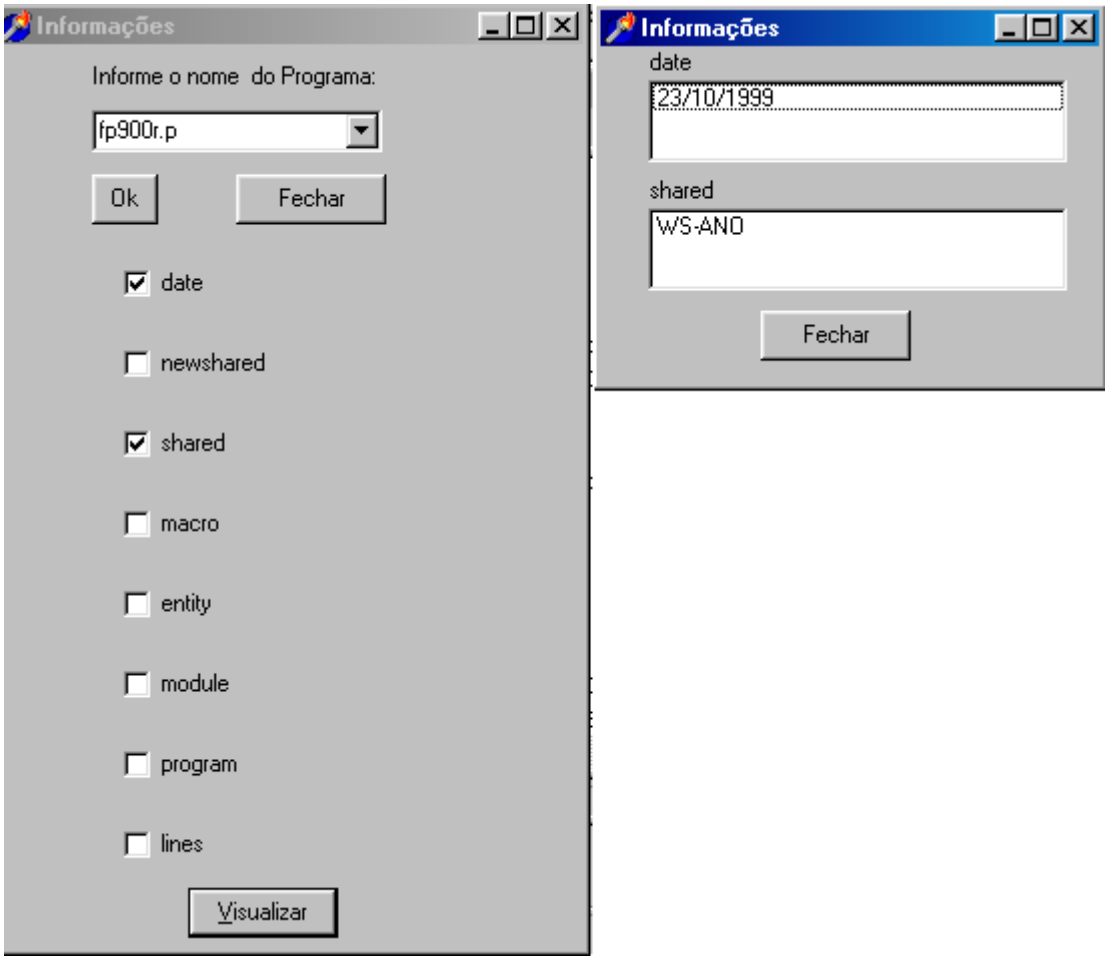


FIGURA 5.9 - Janela com as informações textuais: exemplo com poucas opções selecionadas.

5.4 Arquivos com descrição dos grafos

Como mencionado anteriormente, o visualizador cria dois arquivos com as informações existentes no arquivo descritor original específicas de chamadas de programas e compartilhamento de variáveis. Esses arquivos possuem extensão .dot, os quais serão lidos pela ferramenta *Dotty* para a exibição dos grafos.

O grafo de chamada de programas é um grafo dirigido, pois é uma informação relevante saber que nodo aponta para que nodo, pois está representando que programa chama que programa.

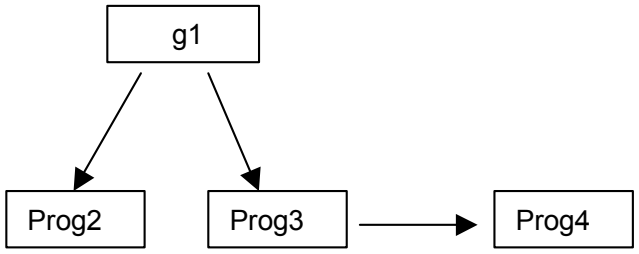
Já o grafo de compartilhamento de variáveis é um grafo não dirigido, pois os nodos representam dados diferentes, uns representam os programas e os outros as variáveis, o que não torna necessário ser um grafo dirigido, pois independente da direção do arco, se um nodo que representa um programa aponta para um nodo que representa uma variável, entende-se que esse programa compartilha essa variável. Para facilitar essa visualização, os nodos do grafo de

compartilhamento de variáveis possuem formatos distintos, ou seja, os nodos que estão representando os programas são retângulos e os que representam as variáveis são elipses.

Os arquivos *.dot* são descritos numa linguagem com uma sintaxe específica, a qual *lefty*, interpretador da ferramenta DOTTY, consegue interpretar. A seguir seguem dois pequenos exemplos de arquivos criados com essa sintaxe específica (figuras 5.10 e 5.11).

```
digraph progs{
   “prog1’ -> “prog2”;
   “prog1” -> “prog3”;
   “prog2” -> “prog4”;
}
```

(a)

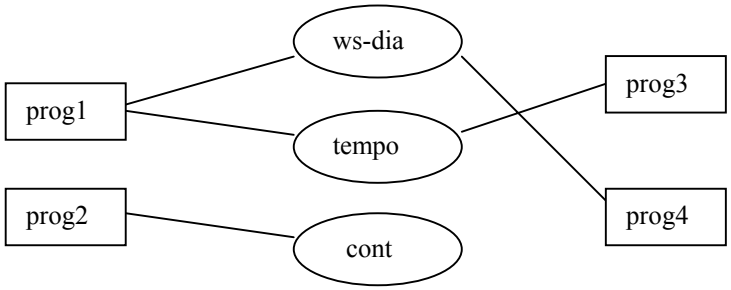


(b)

FIGURA 5.10 - (a) Exemplo de arquivo para a visualização de um grafo dirigido e (b) grafo criado pela ferramenta DOTTY correspondente ao arquivo (a).

```
graph varshared{
   “prog1” -- “ws-dia”;
   “prog1” -- “tempo”;
   “prog2“ -- “cont”;
   “prog3” -- “tempo”;
   “prog4” -- “ws-dia”;
}
```

(a)



(b)

FIGURA 5.11 - (a) Exemplo de arquivo para a visualização de um grafo não dirigido e (b) o grafo criado pela ferramenta DOTTY correspondente ao arquivo (a).

6 Visualização de Grafos

O desenho de grafos é uma das melhores maneiras de apresentar as informações técnicas na forma de diagramas, pois estes são particularmente apropriados para mostrar relações entre objetos e por serem mais fáceis de ler e entender. Por isso, diversos problemas de apresentação de dados envolvem a visualização de grafos [NOR92]. Máquinas de estado finito, redes, grafos de chamadas de programas, e vários outros tipos de dependência entre objetos são exemplos de informações que podem ser compreendidas mais facilmente quando apresentadas na forma de diagramas. Conseqüentemente, é importante possuir um bom conjunto de ferramentas para mostrar e manipular grafos. Tais ferramentas poderiam incluir recursos para construir grafos, visualizar e interagir com os grafos.

No decorrer do desenvolvimento do visualizador, para a exibição dos diagramas de chamadas de programas e de compartilhamento de variáveis, realizou-se uma pesquisa sobre visualização de grafos e optou-se por utilizar a ferramenta *Dotty,* por atender as necessidades da visualização e também por ter sido desenvolvida em C e seus fontes estarem disponíveis na Web. Mais detalhes da ferramenta a seguir.

6.1 Ferramenta DOTTY

Dotty é, na realidade, um editor de grafos. Seus principais componentes são um visualizador programável, chamado *lefty*, e dois geradores de *layout* de grafos, *dot* para grafos hierárquicos e *neato* para grafos simétricos. *Dotty* compreende, então, três processos que operam conjuntamente: *dot*, n*eato* [NOR92] e l*efty* [KOU91]. *Lefty* é o módulo que tem a função de mostrar os grafos na tela e permitir que o usuário interaja sobre eles, podendo manipular os grafos e seus componentes. O programa inclui funções para inserir e remover nodos e arcos, bem como desenhar esses objetos de acordo com atributos específicos tais como: cores, formatos, estilos, entre outros. *Dotty* apresenta ainda uma função que processa o *layout*. Essa função envia o grafo para um processo *dot* ou *neato,* o qual processa o *layout* e envia o grafo com as informações do *layout* inseridas como atributos do grafo para o processo *lefty. Lefty,* então, atualiza as coordenadas dos nodos e dos arcos e redesenha o grafo na tela.

A figura 6.1 mostra a arquitetura básica de *Dotty,* cujos processos serão basicamente descritos nas seções a seguir.

A ferramenta D*otty* pode ser executada sozinha, independente, mas, pode também ser programada para se comunicar com outros processos, servindo de *front end* para outras ferramentas. Nesse caso, uma ferramenta pode passar seu estado para *dotty*, como um grafo, e sempre que o usuário quiser mudar o grafo, *dotty* enviará uma mensagem para a ferramenta, a qual mudará o estado do grafo. Alguns exemplos interessantes com outras aplicações são *ciao,* código fonte de um navegador de banco de dados, *vdbx,* uma extensão visual para *dbx* para mostrar estruturas de dados na forma de grafos, um depurador gráfico, e *vpm,* um gerenciador visual de processos distribuídos [KOU96].

Quando inicializado, *Dotty* abre uma janela com rótulo *DOTTY*. A janela é aberta vazia, caso nenhum nome de arquivo de grafo tenha sido especificado na linha de comando; caso contrário, o grafo é mostrado. O usuário pode então adicionar e remover nodos e arcos ou mudar os atributos dos mesmos, tais como cor e formato dos nodos e arcos. *Dotty* não gera novas exibições automaticamente, o usuário tem que solicitar por um novo grafo explicitamente, utilizando a opção ***load graph*** presente no menu.

Os *layouts* de grandes grafos são freqüentemente complexos e difíceis de ler. Por isso, bons algoritmos de *layout* ajudam quando um grafo é muito grande. Técnicas apropriadas para filtrar e particionar grafos e aplicar cores podem ajudar realçar propriedades de interesse [NOR99].

Dotty suporta múltiplos tipos de visão gráfica. Cada tipo de visão pode ter sua própria maneira de mostrar um grafo e seu próprio conjunto de ações do usuário em cima do grafo e/ou operações da figura. Por *default*, *dotty* fornece dois tipos de visão: uma visão normal e uma visão *bird's eye* ("olho de pássaro").

A visão normal mostra o grafo numa escala 1:1 (1 ponto *dot* por 1 pixel). Caso o grafo seja muito grande, o usuário pode usar as barras de rolagem para navegar através do grafo. O usuário pode também modificar escalas para aumentar ou diminuir o grau de visualização (*zoom in - zoom out*).

A visão *bird's eye* mantem ajustada a escala, ou seja, quando o tamanho do grafo aumenta a escala muda também. Quando o usuário clica em algum local nessa visão, *dotty* ajusta as posições da visualização de todas as outras visões do mesmo grafo.

Essas visões possuem uma série de funções que podem ser executadas pelo usuário: desfazer a última operação, copiar, recortar e colar, gerar um grafo novo, carregar um grafo, redesenhar o grafo, salvar o grafo, *zoom in, zoom out*, encontrar nodo, imprimir o grafo, agrupar subgrafos, inserir e remover nodos e arcos e alterar atributos do grafo, para citar algumas.

Dotty não é apenas um bom editor de grafos de propósito geral, mas uma ferramenta flexível para construir *front end*s gráficos para outras ferramentas. A boa performance de *dotty* e a alta qualidade dos *layout*s resultam em respostas rápidas e figuras com boa apresentação visual [KOU96b]

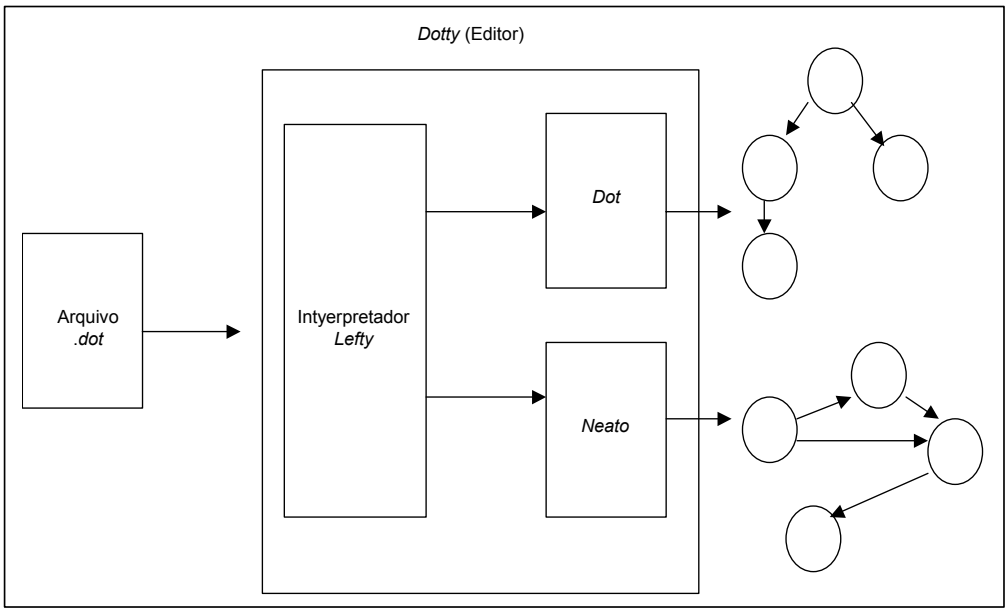


FIGURA 6.1 - Componentes da ferramenta *Dotty*

6.2 DOT

Dot exibe *layouts* hierárquicos de grafos dirigidos, além de possuir uma variedade de estruturas, formatos, estilos, cores apropriadas para diagramas relacionados a *software*. Por exemplo, *Dot* pode criar grafos de estruturas de dados, mostrando registros como uma lista lateral. *Dot* também incorpora algoritmos para desenho de grafos com agrupamento de grafos ou como um conjunto de nodos recursivos. Quando ocorrem agrupamentos, são desenhados retângulos ao redor dos grafos a serem agrupados.

Dot lê arquivos textos com extensão *.dot* ou arquivos PostScript com extensão .ps e desenha o grafo definido no arquivo.

Dot tem três tipos de itens: grafos, nodos e arcos. O grafo principal pode ser um grafo dirigido ou um não dirigido. Dentro do grafo principal, um subgrafo define um subconjunto de nodos e arcos. Um nodo é criado num primeiro momento com os nomes que aparecem no arquivo, e os mesmos são ligados pelo operador ->, se o grafo for dirigido e --, se não dirigido.

Dot é útil para ajustar representações ou localizações de nodos e arcos no *layout*. Isso é feito setando os atributos dos nodos, dos arcos, ou dos subgrafos no arquivo de entrada. Por *default*, os nodos são desenhados com formato de elipse e rotulados com o nome do nodo. Outros formatos existentes além de elipses são: quadrados, círculos, etc. Já os arcos, os grafos e os agrupamentos de subgrafos por *default* não são rotulados, mas podem ter rótulos desde que se defina os atributos. Os atributos do fonte são pré-definidos, tamanho 14, Times New Roman, em preto, podendo ser setados para outras fontes. Nodos e arcos têm também atributos de cor, tamanho, estilo, entre outros.

6.3 NEATO

Para grafos simétricos, tal como uma rede de computadores, *layouts* hierárquicos podem não ser tão informativos quanto outros tipos que representam a conectividade entre os nodos. *Neato* é o componente que desenha grafos simétricos, é compatível com *dot* pelo fato de aceitar os mesmos arquivos de entrada e opções na linha de comando [KOU91b]. De fato, *dotty* pode escolher qual dos componentes utilizar mudando apenas o nome do caminho. [NOR99]

Já que o formato do arquivo inclui tanto grafos dirigidos como não dirigidos, *Neato* desenha grafos preparados por *dot*, e vice-versa. Ambos os programas tem as mesmas opções de setar cores, formatos, rótulos, fontes de texto e paginação, e de gerar código em linguagens como, PostScript, FramMaker, MIF e HPGL/2.

Um arquivo de um grafo é formado de um cabeçalho pequeno e um corpo consistindo de nodos, arcos, e atributos. Por *default,* o formato dos nodos são elipses rotuladas com o nome do nodo. Da mesma forma que *Dot,* os grafos não dirigidos são criados através do operador - - e grafos dirigidos pelo operador ->. Arcos são desenhados como linhas retas e do mesmo tamanho. As opções cores, formatos, fontes, tamanhos, rótulos são convenientes para muitos tipos de *layout*. As opções como tamanho do desenho, paginação são as mesmas em *dot.* Além disso, *Neato* permite ajustar o *layout*.

O *Neato* permite também ajustar o tamanho dos arcos para melhorar a visualização. O tamanho de um arco é a distância existente entre os nodos que serão ligados. Arcos invisíveis podem também ser inseridos para ajustar a localização dos nodos. Existe ainda, uma maneira de fornecer as coordenadas inicial e final de nodos individuais. A posição inicial, formada de dois números, é o valor de entrada do atributo posição do nodo.

Para melhorar a visualização dos grafos, existe a possibilidade de eliminar sobreposições de nodos e arcos não intencionalmente. Esses erros poderão ser tratados a partir de regras, as quais poderiam oferecer também ao usuário uma maneira natural de ajustar os *layout*s.

6.4. LEFTY

Cada figura é descrita em um arquivo .dot que contém funções para desenhar a figura e editá-la. Esse arquivo é desenvolvido em uma linguagem de programação procedural chamada *Lefty,* a qual implementa as operações da estrutura de dados necessárias, tais como: inserção e remoção de nodos, arcos e subgrafos, incluindo funções de entrada e saída (leitura e escrita) de grafos através de arquivos.

Além disso, *Lefty* implementa as seguintes operações[NOR99]:

- Criação e remoção de grafos
- Criação e remoção de visões de grafos (um grafo pode ter diversas visões)
- Carregar e salvar grafos para arquivos
- Aumentar e diminuir o zoom de uma visão
- Procurar por um nodo pelo nome

- Mover geometricamente um nodo e permanecer com todos os seus arcos ligados
- Editar atributos de um objeto

Assim, as funções acima são interpretadas e *Lefty* envia o resultado para *Dot ou Neato* mostrarem o grafo.

Umas das propriedades principais de *Lefty* é a exatidão, uma vez que a razão de desenhar figuras técnicas é mostrar alguns objetos abstratos de maneira que fique fácil de entender. Por isso, é muito importante que as posições, tamanhos, e outros atributos gráficos das primitivas gráficas em uma figura sigam regras estritas.

Muitos editores de grafos fornecem maneiras para desenhar figuras com exatidão, mas fornecem poucas maneiras para manter a consistência das figuras técnicas, onde se um determinado nodo for deslocado para outro lugar, automaticamente todos os nodos interligados a ele serão deslocados juntamente [KOU96c].

6.5 Comentários Finais

Dotty foi, então, a ferramenta de visualização de grafos escolhida para fazer a visualização dos grafos de chamadas de programas e compartilhamento de variáveis por diversos motivos.

Em termos de tratamento de desenhos de grafos como objetos estruturados, *Dotty* fornece um menu com comandos para carregar ou criar grafos, editar os grafos criados ou carregados, gravar e, até mesmo modificá-los. Os atributos dos grafos são descritos na linguagem de dados *Lefty* bem flexível em termos de atributos, podendo facilmente adicionar atributos aos grafos.

Com *Dotty*, é possível escrever diversas aplicações portáveis, uma vez que *Lefty* esconde o servidor dos grafos e o sistema operacional. Assim os programadores de aplicações não precisam se preocupar com os detalhes de fontes, menus, cores e outras características de baixo nível que impedem a portabilidade. Uma prova disso é que se pode rodar os mesmos scripts no Microsoft Windows bem como em UNIX versão X11.

Quanto à qualidade do *layout* e robustez, *Dotty* se beneficia por utilizar *Dot* e *Neato* para fazer os *layout*s, tornando-os legíveis automaticamente, sem requerer a intervenção do usuário. Robustez também é importante porque grafos da vida real podem ter múltiplos arcos, exclusão de componentes, etc [NOR99].

Uma caraterística importante de *Dotty* é que ela pode ser facilmente customizada. Para customizar *Dotty* é necessário modificar os programas *Lefty*. Por exemplo, as funções que operam na estrutura de dados do grafo, como inserir um nodo, poderiam ser modificadas para apenas permitir operações que são apropriadas para um tipo de grafo específico. A parte mais interessante de customização é que *Dotty* é programada para agir como *front end* para outros processos, ou seja, uma ferramenta que gera e mantêm informações que podem ser expressadas como grafos pode utilizar *Dotty* para mostrar as informações graficamente.

7 Avaliação do Protótipo *VisProgress*

7.1 Método e Instrumento

Após o desenvolvimento do visualizador, o mesmo foi avaliado pelas equipes de desenvolvimento das empresas Benfare Informática e Dzset Soluções e Sistemas para Computação, as quais utilizam o ambiente *Progress* para o desenvolvimento dos sistemas. A avaliação foi feita nessas duas empresas por dois motivos:

1) pelo fato da Benfare Informática ter sido a empresa do estudo de caso, onde procurou-se abordar no visualizador, as lacunas que a empresa tinha quanto à facilidade de compreender programas e de se ter informações sobre eles mais rapidamente do que analisar o código;
2) porque seria inadequado validar o visualizador apenas com os resultados dos questionários submetidos à equipe de desenvolvimento da empresa Benfare Informática, uma vez que, como dito anteriormente, foi a empresa escolhida para o estudo de caso.

As equipes das duas empresas receberam o visualizador, uma breve explicação sobre seu uso e trabalharam com o visualizador conforme sua disponibilidade.

Depois da utilização do visualizador pelos membros das equipes de desenvolvimento das empresas citadas acima, foi solicitado a eles que fizessem uma avaliação da ferramenta para que fosse possível verificar suas vantagens e desvantagens em relação às necessidades de uma equipe de desenvolvimento. Essa avaliação teve como instrumento um questionário (anexo 1) entregue a cada um dos membros da equipe, com alguns pontos a serem avaliados. São eles:

- facilidade de utilizar o visualizador;
- interação com a ferramenta;
- apresentação de menus com funções claras;
- qualidade na forma pela qual as informações são mostradas;
- distribuição das janelas na tela;
- distribuição dos dados nas janelas;
- clareza na visualização dos grafos, facilidade em compreender os grafos;
- atendimento das necessidades da equipe de desenvolvimento;
- rapidez no acesso às informações;
- número de informações dos programas que podem ser visualizadas;
- flexibilidade na visualização das informações, ou seja, a possibilidade do usuário visualizar somente as informações que lhe são de interesse;
- visualizar a relação existente entre os programas através de grafos;
- visualizar o compartilhamento das variáveis também através de grafos;
- mostrar mensagens de alerta para uma dada ação incorreta ou indisponível;
- utilização da ferramenta Dotty para a visualização dos grafos;

- recursos que a ferramenta Dotty oferece trazem vantagens para a equipe de desenvolvimento;
- melhoria no desenvolvimento e manutenção dos programas com a utilização do visualizador;
- redução do tempo gasto com manutenção e desenvolvimento de programas utilizando o visualizador.

7.2 Resultados

A tabela abaixo ilustra os resultados obtidos na avaliação do visualizador feita com as equipes de desenvolvimento da empresa Benfare Informática e Dzet Soluções – Datasul. Dos treze programadores que utilizaram a ferramenta, oito são integrantes da equipe de desenvolvimento da Benfare e cinco da Datasul.

TABELA 7.1 - Resultados dos itens avaliados do visualizador *VisProgress*

Ítens Avaliados	Excelente	Muito Bom	Bom	Regular	Ruim
Facilidade em utilizar o visualizador	5	5	3		
Possibilidade do usuário interagir com o visualizador	4	5	4		
Rapidez no acesso as informações	8	4	1		
Qualidade na forma como são visualizadas as informações	5	5	3		
Distribuição das janelas na tela (Ocupação do espaço na tela)	1	2	4	6	
Distribuição dos dados nas janelas	3	5	5		
Flexibilidade na visualização das informações (Permitir que o usuário visualize somente as informações que lhe são de interesse)	4	7	2		
Clareza na visualização dos grafos	4	7	2		
Facilidade na compreensão dos grafos	4	7	2		
Facilidade de observar a relação entre programas através de grafos	6	6	1		
Facilidade de observar o compartilhamento das variáveis entre os programas através de grafos	10	3			
Mostrar mensagens de alerta ao usuário	7	5	1		
Utilização da ferramenta Dotty para visualização dos grafos	5	3	5		
Melhoria no desenvolvimento e manutenção dos programas com a utilização do visualizador	2	2	6	1	
Redução do tempo gasto com manutenção e desenvolvimento de programas com a utilização do visualizador	2	6	3	1	

Além dos itens abordados na tabela acima, foram avaliados outros no questionário de avaliação submetido às equipes de desenvolvimento. Se o visualizador oferece um menu com opções que deixam suas funções claras foi um dos ítens e o outro, se os recursos que a ferramenta Dotty oferece são vantajosos para a equipe de desenvolvimento. As duas perguntas tiverem como resposta por unanimidade sim.

Já, o questionamento se o visualizador supria as necessidades de uma equipe de desenvolvimento, tendo como alternativas todas, quase todas, algumas, poucas e nenhuma, obteve como resposta 7 quase todas, 5 algumas e apenas um poucas. As opções todas e nenhuma, não foram escolhidas por nenhum membro das equipes de desenvolvimento.

Outra pergunta presente no questionário avaliado foi se a quantidade de informações que podem ser visualizadas por programa era excelente, muito boa, boa ou poucas. Como resultado, um programador considerou excelente, 3 muito bom, 5 bom e 4 poucas.

Com base nas respostas obtidas no questionário de avaliação e nos comentários feitos pelos programadores durante a avaliação do questionário, pode-se dizer que o visualizador foi considerado uma ferramenta bem projetada, bem interessante e principalmente útil para uma equipe de desenvolvimento, uma vez que a ferramenta abrange tanto a visualização gráfica, através de grafos, como uma visualização textual, conseguindo assim relacionar perfeitamente os tipos de informações a serem mostradas com o tipo de visualização mais adequado. Além disso, consideraram a ferramenta fácil de utilizar, mas como a maioria das ferramentas, depois de um pequeno treinamento. Verificou-se que apesar dos botões serem nomeados, não fica totalmente clara qual a função deles somente pelo nome.

A interação do usuário com a ferramenta foi considerada muito boa, uma vez que ela permite que o usuário selecione somente o que ele realmente deseja visualizar, sem precisar perder muito tempo e espaço de visualização com as informações que são consideradas irrelevantes.

As equipes consideraram excelente a rapidez pela qual o visualizador busca as informações nos programas e mostra-as na tela. Isso se justifica, pelo fato do visualizador não ser uma ferramenta muito complexa e por ter sido implementada com auxílio de estruturas de dados dinâmicas.

Quanto à qualidade e quantidade de informações que podem ser visualizadas, pode-se concluir que o visualizador apresenta uma qualidade muito boa, onde as informações mostradas são extraídas diretamente do arquivo descritor, o qual é montado a partir do código fonte dos programas, tornando assim as informações totalmente confiáveis. Vale ressaltar que o volume de informações poderia ser maior, uma vez que, o número de informações que um programa possui, é muito maior do que as mostradas pelo visualizador. Com isso, percebeu-se que o visualizador supre quase todas as necessidades das equipes, porém deixa de mostrar algumas que também seriam úteis para um melhoria na manutenção e no desenvolvimento dos programas. Algumas delas foram sugeridas pelas equipes de desenvolvimento, mostrar os parâmetros de entrada e saída dos programas e os relacionamentos entre as entidades através das suas chaves primárias.

Um outro item avaliado foi a distribuição das janelas do visualizador na tela. Esse item foi um dos mais criticados pelas equipes de desenvolvimento, onde consideraram as janelas muito pequenas, fazendo com que as janelas de visualização perdessem um pouco a priorização das janelas do visualizador em relação aos outros aplicativos do Windows.

Já, a maneira como foram distribuídos os dados nas janelas, consideraram muito boa, inclusive a maneira fácil de selecionar os programas e as informações desejadas, pois foram utilizadas caixas de seleção (*checkbox*) para selecionar as informações de interesse e listas (*listbox*) para a escolha do programa, o que facilita de fato a seleção dos mesmos, visto que, o usuário não precisa digitar nada, apenas clicar com o mouse, pois as opções já estão montadas.

Um dos itens que teve resultados mais positivos na avaliação do visualizador foi a flexibilidade, já que são disponibilizadas somente informações que o programa selecionado apresenta, impedindo assim, que o usuário selecione uma informação que não possa ser

visualizada. Ainda, somente as informações selecionadas são exibidas minimizando a carga de dados na tela. Essa característica é muito importante, pois as equipes fazem alterações das mais diversas em um programa, e também desenvolvem diferentes programas, quase que diariamente, precisando assim, visualizar informações diferentes a cada momento, dependendo dos tipos de informação que serão alteradas.

Quanto às mensagens de alerta em caso de erros ou ações indisponíveis na ferramenta, as equipes consideraram importantes, porque o usuário está apenas utilizando-a, não tendo a obrigação de saber dos possíveis erros que podem acontecer, bem como ações que o programa não disponibiliza, evitando com isso que ocorram inconsistências.

As equipes de desenvolvimento acharam excelente o uso de grafos para mostrar as chamadas de programas e o compartilhamento de variáveis. Isso se dá, pelo fato dos grafos apresentarem uma estrutura bastante simples, o que torna o ambiente visualmente claro, uma vez que os grafos possuem poucos componentes, basicamente nodos e arcos.

Como visto anteriormente no texto, esses grafos são apresentados no visualizador através da ferramenta *Dotty*. Uma ferramenta específica para visualização de grafos e que atende as necessidades desse visualizador. Consideraram *Dotty*, uma ferramenta muito boa, com bastante recursos para visualizar grafos dos mais variados, dirigidos e não dirigidos, coloridos ou não, com nomes, atributos de tamanho, estilo, entre outros formatos. Essa ferramenta possibilita ao usuário personalizar o grafo conforme ache conveniente e até mesmo criar seu próprio grafo, tudo isso com o objetivo de facilitar a compreensão dos programas para mais facilmente e eficazmente manter e desenvolver novos programas. Assim, quando questionada sobre a necessidade de eliminar os recursos de edição de *Dotty*, a equipe de desenvolvimento considerou útil que ficassem disponíveis esses recursos.

Devido ao pouco tempo de utilização do visualizador, as equipes não conseguiram ter uma noção exata à respeito da melhoria na manutenção e desenvolvimento dos programas, bem como na redução do tempo gasto com essas operações. Mas, pelo que puderam perceber acreditam que o visualizador, quando utilizado com mais freqüência e por mais tempo, irá influenciar diretamente nessas operações, visto que o maior tempo dispendido atualmente pelos programadores durante a manutenção dos programas é o tempo gasto tentando entender o código dos programas e fazendo relações com o que já foi definido e o que se quer alterar e/ou definir.

8 Conclusões e Possibilidades Futuras

A visualização de informações, mais especificamente a visualização de *software* é uma área que emergiu dentro das comunidades de visualização e engenharia de *software* para auxiliar os programadores na compreensão dos programas, visto ser esta uma tarefa complexa e demorada se não acompanhada de ferramentas de visualização.

O interesse em desenvolver e avaliar um visualizador de *software* veio da necessidade de se ter uma visão mais ampla de grandes sistemas como o da Benfare, bem como da interligação existente entre os módulos de uma maneira mais legível e rápida. Um aspecto importante é o acesso as informações necessárias durante o ciclo de vida do *software* sem precisar analisar o código fonte de cada programa. Atualmente, observa-se uma tendência das empresas em adotar aplicações de visualização integradas e interativas, porque estas trazem benefícios estratégicos que afetam positivamente o produto.

A motivação para construção de sistemas "*visuais"* tem sido bem citada na literatura. Enquanto novos ambientes de programação tem começado a fazer uso das mais diversas técnicas visuais para mostrar a informação, a maioria dos programadores tem observado a permanência do uso de tecnologia de visões antigas: eles poderiam utilizar janelas de múltiplas rolagens em telas grandes, mas muitos permanecem editando seu programa em textos simples de cores pré-definidas e observando seus programas utilizando depuradores convencionais.

Por que a tecnologia de visualização de *software* não está sendo utilizada e por que os novos sistemas visuais não estão sendo totalmente adotados? Uma resposta para isso seria que os engenheiros de software não têm visto demonstração dessas tecnologias em uso. E é claro que se sistemas de visualização de *software* estão contribuindo para a engenharia de *software*, então resultados sólidos demonstrando seus benefícios estão se tornando necessários**.**

Com o presente trabalho pode-se perceber que as equipes de desenvolvimento de sistemas perdem, muitas vezes, bastante tempo para entenderem o funcionamento de um programa e buscarem informações relevantes através do código fonte, o que é extremamente necessário para realizarem as operações de manutenção e desenvolvimento de programas novos ou já existentes. Acredita-se que o uso de uma ferramenta que auxilie a equipe a obter informações dos programas de maneira mais rápida e legível, um visualizador de informações encontradas no código fonte dos programas que compõem o sistema, podem contribuir sensivelmente para o aumento da produtividade da equipe.

O visualizador desenvolvido teve como base a situação da empresa Benfare Informática e foi avaliado por essa empresa e também pela empresa Dzset Soluções e Sistemas de Computação. O resultado obtido com essas avaliações foi bastante satisfatório, tendo sido possível certificar-se que há uma necessidade grande em se utilizar ferramentas de visualização que auxiliem na compreensão dos programas, uma vez que essas tem o objetivo de otimizar a manutenção e o desenvolvimento de programas.

Existem, ainda, diversos itens que podem ser melhorados no visualizador e outros até mesmo acrescentados, como por exemplo o número de informações mostradas, para tornar essa

ferramenta ainda mais útil, mas a essência principal, conseguiu-se atingir. O que surpreendeu é que esperava-se que esse protótipo iria satisfazer somente a equipe de desenvolvimento da empresa Benfare, pelo fato do visualizador ter sido desenvolvido baseado nas pendências que ela apresentava, mas verificou-se que outra empresa também considerou a ferramenta útil e importante para auxiliar na compreensão dos programas.

Como trabalho futuro, é proposto um estudo em XML para uma possível transformação do arquivo descritor, o qual contém as informações relevantes dos programas, para uma estrutura XML, com o intuito de torná-lo mais legível e possível de ser usado por outros interpretadores. Outros trabalhos que podem ser feitos são na linha de utilização de ícones e gráficos para a visualização das informações numéricas e tratamento da informação temporal que está incorporada ao descritor, mas não é utilizada. O tratamento de versões no arquivo descritor e técnicas/políticas para atualização automática do arquivo descritor seriam possibilidades bem interessantes que também podem ser estudadas.

Anexo1 Questionário de Avaliação do Visualizador V*isProgress*

P/ EQUIPE DE DESENVOLVIMENTO

Pontos para a Avaliação do Visualizador:

1) Facilidade em utilizar o visualizador
( ) Excelente ( ) Muito Bom ( ) Bom ( ) Regular ( ) Ruim

2) O visualizador oferece um menu com opções que deixam suas funções claras?
( ) Sim ( ) Não

3) O visualizador supre as necessidades de uma equipe de desenvolvimento?
( ) Todas ( ) Quase todas ( ) Algumas ( ) Poucas ( ) Nenhuma

4) Possibilidade do usuário interagir com o visualizador
( ) Excelente ( ) Muito Bom ( ) Bom ( ) Regular ( ) Ruim

5) Rapidez no acesso as informações
( ) Excelente ( ) Muito Bom ( ) Bom ( ) Regular ( ) Ruim

6) Qualidade na forma como são visualizadas as informações
( ) Excelente ( ) Muito Bom ( ) Bom ( ) Regular ( ) Ruim

7) Quantidade de informações que podem ser visualizadas por programa
( ) Excelente ( ) Muito Bom ( ) Bom ( ) Poucas

8) Distribuição das janelas na tela (Ocupação do espaço na tela)
( ) Excelente ( ) Muito Bom ( ) Bom ( ) Regular ( ) Ruim

9) Distribuição dos dados nas janelas
( ) Excelente ( ) Muito Bom ( ) Bom ( ) Regular ( ) Ruim

10) Flexibilidade na visualização das informações (Permitir que o usuário visualize somente as informações que lhe são de interesse)
( ) Excelente ( ) Muito Bom ( ) Bom ( ) Regular ( ) Ruim

11) Clareza na visualização dos grafos (Maneira como os grafos estão dispostos nas janelas)
( ) Excelente ( ) Muito Bom ( ) Bom ( ) Regular ( ) Ruim

12) Facilidade na compreensão dos grafos
( ) Excelente ( ) Muito Bom ( ) Bom ( ) Regular ( ) Ruim

13) Facilidade de observar a relação entre programas através de grafos
( ) Excelente ( ) Muito Bom ( ) Bom ( ) Regular ( ) Ruim

14) Facilidade de observar o compartilhamento das variáveis entre os programas através de grafos?
( ) Excelente ( ) Muito Bom ( ) Bom ( ) Regular ( ) Ruim

15) Mostrar mensagens de alerta ao usuário
( ) Excelente ( ) Muito Bom ( ) Bom ( ) Regular ( ) Ruim

16) Utilização da ferramenta DOTTY para visualização dos grafos

( ) Excelente ( ) Muito Bom ( ) Bom ( ) Regular ( ) Ruim

17) Os recursos que a ferramenta DOTTY oferece são vantajosos para a equipe de desenvolvimento?
( ) Sim ( ) Não

18) Melhoria no desenvolvimento e manutenção dos programas com a utilização do visualizador
( ) Excelente ( ) Muito Bom ( ) Bom ( ) Regular ( ) Ruim

19) Redução do tempo gasto com manutenção e desenvolvimento de programas com a utilização do visualizador
( ) Excelente ( ) Muito Bom ( ) Bom ( ) Regular ( ) Ruim

20) Sugira um nome para o visualizador): _____

Anexo 2 Exemplo de um Arquivo Descritor

```
<descritor>fp900r.p</descritor>
<date>23/10/1999</date>
<newshared>TEMPO</newshared>
<shared>WS-ANO</shared>
<variable>ULTIMA,PRIMEIRA,VLR-PRODRURF,VLR-PRODRURJ,WS-NRODIAS,
WS-TP4,I,WS-PIS,WS-CPF,WS-NASCTO,WS-PROFNUM,WS-PROFSERIE,WS-NOME,
WS-COD,WS-DESCR,WS-ENDER,WS-CONTIND,EVEN,IGUAIS,ULTDIA</variable>
<buffer>BUF-FPRESCAL,BUF-FPCAD</buffer>
<macro>TL-PADVR,TL-PAD,PARAM,LE-ESTAB</macro>
<entity>FPEVENTO,FPMESTRE,FPCAD,FPRESCAL,WFPCAD,BGESTAB,FPFILIAL,
BGCEP,FPTOTFOL,FPCAINSS,FPAFAST,FPTABINS,FPSALINSS</entity>
<module>FP,BG,WF,PA</module>
<program>FP0021,FP200</program>
<lines>1289</lines>
<type>relatorio</type>
<descritor>fp302r.p</descritor>
<date>01/05/1998</date>
<variable>EVEN,VLR-BASE,VLR-SEGUR,VLR-TERC,VLR-DEDFPAS,VLR-EMPR,
VLR-TOTAL,TOTBGE-BASE,TOTBGE-SEGUR,TOTBGE-TERC,TOTBGE,TOTBGE-EMP,
TOTBGE-TOTAL,TOTBGE-EMPREG,TOTEMP-BASE,TOTEMP-SEGUR,TOT-TERC,
TOTEMP-DEDFPAS,TOTEMP-EMPR,TOTEMP-TOTAL,TOTEMP-EM </variable>
<newshared>Z-BGES-ES</newshared>
<macro>TL-PADVR,TEMPO,PARAM,LE-ESTAB,TL-PAD,TB-MESES</macro>
<program>ZBGES-ES</program>
<entity>BGESTAB,FPMESTRE,BGEMPR,FPEVENTO,FPCAD,FPRESCAL,FPGRUPO,F
PFILIAL</entity>
<module>BG,FP</module>
<lines>254</lines>
<type>relatorio</type>
```

Bibliografia

[AND97] ANDRIENKO, G.; ANDRIENKO, N. Iris: a Knowledge-Based System for Visual Data Exploration. In: CONFERENCE ON COMPUTER HUMAN INTERACTION, 1997. **Proceeedings...** New York: ACM Press, 1997.

[BAS98] BASSO, Karen. **Visualização de Dados Geológicos**. Porto Alegre: CPGCC da UFRGS, 1998. Dissertação de Mestrado.

[BRO84] BROWN, M. H.; SEDGEWICK, R. **T**ecniques for Algorithm Animation. **IEEE Software**, Los Alamitos, v. 2, n. 1, p. 28-29, 1984.

[BRO88] BROWN, M. H. **Algorithm Animation**. New York: MIT Press, 1988.

[CAM97] CAMPOS, Marcelo Ricardo. **Compreensão Visual de Frameworks através da Introspecção de Exemplos.** Porto Alegre: CPGCC da UFRGS, 1997. Tese de Doutorado.

[CAV2001] CAVA, R.A., FREITAS, C.M.D.S. **Visualizing hierarchies with a modified focus+context technique**. Trabalho submetido ao IEEE Visualization Information, 2001.

[CHE94] CHEN, Hsinchun. **A Textual Database/Knowledge-base coupling approach to creating computer-supported organizational memory.** Cidadde: HIS Department, University of Arizona, 1994. Disponível em: <http://ai.bpa.arizona.edu/papers/>. Acesso em: 10 fev. 2001.

[CHU98] CHUAH, M.C.; EICK, S.G. Information Rich Gliphs for Software Management Data. **IEEE Computer Graphics and Applications,** Los Alamitos, v. 18, n. 4, p. 24-29, July/Aug. 1998.

[COW96] COWIE, Jim; LEHNERT, Wendy. Information extraction. **Communications of the ACM**, New York, v.39, n.1, 1996.

[EST2000] ESTIVALET, L.F. **O uso de ícones na visualização de informações.** Porto Alegre: CPGCC da UFRGS, 2000. Dissertação de Mestrado.

[FRE93] FREITAS, C.M.S.; WAGNER, F.R. A Methodology for Selecting Visual Representations in Scientific and Simualtion Applications. In: SIMPÓSIO BRASILEIRO DE COMPUTAÇÃO GRÁFICA E PROCESSAMENTO DE IMAGENS, SIBGRAPI, 6., 1993, Recife. **Anais...** Recife: SBC/UFPE, 1993. p.89-97.

[FUR86] FURNAS, G.W. Generalized Fisheye Views. In: CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, CHI, 1986. **Proceedings...** [New York; ACM], 1986. p.16-23.

[FYO97] FYOCK, Daniel E. Using Visualization to Maintain Large Computer Systems. **IEEE Computer Graphics and Applications**, Los Alamitos, p.73-75, 1997.

[GER97] GERSHON, N; ELICK, S.G. Information Visualization. **IEEE Computer Graphics and Applications**, Los Alamitos, p.29-31, 52-59, 1997.

[HAR93] HARDY, D.R.; SCHWARTZ, M.F. Essence: a resource discovery system based on semantic file indexing. In: WINTER USENIX, 1993. **Proceedings...** [S.l.:s.n.], 1993. p. 25-29.

[HOL97] HOLMQUIST, L.E. Focus+Context Visualization with Flip Zooming and the Zoom Browser. In:CHI, 1997. **Proceedings...** Disponível em: <http://www.acm.org/sigchi/chi97/proceedings/poster.Ich.html>. Acesso em: 10 out. 2000.

[KOU91] KOUTSOFIOS, Eleftherios.; DOBKIN, David. Lefty: a Two-view editor for technical pictures. In: CONFERENCE GRAPHS INTERFACE, 1991. **Proceedings...** [S.l.: s.n.], 1991. p 68-76.

[KOU91b] KOUTSOFIOS, Eleftherios.; NORTH, Stephen.C. **Drawing graphs with *dot***. Murray Hill, NJ: AT&T Bell Laboratories, 1991. (Technical Report 910904-59113-08TM).

[KOU96] KOUTSOFIOS, Eleftherios; NORTH, Stephen C. **Editing graphs with *dotty***. Murray Hill, NJ: AT&T Bell Laboratories, 1996.

[KOU96b] KOUTSOFIOS, Eleftherios; NORTH, Stephen C. **Drawing graphs with *dot***. Murray Hill, NJ : AT&T Bell Laboratories, 1996.

[KOU96c] KOUTSOFIOS, Eleftherios. **Editing Pictures with *lefty***. Murray Hill, NJ: AT&T Bell Laboratories, 1996.

[LAM95] LAMPING, J.; RAO, R.; PIROLLI, P. A Focus+Context Technique Based on Hyperbolic Geometry, for Visualization Large Hierarchies. In: CONFERENCE HUMAN FACTORS IN COMPUTING SYSTEMS, 1995. **Proceedings...** [New York: ACM], 1995. p. 401-408.

[LUZ2001] LUZZARDI, P.R.G. **Avaliação de técnicas de visualização de informações.** Porto Alegre: PPGC-UFRGS, 2001. Proposta de tese de doutorado a ser apresentada.

[MAN94] MANSSOUR, I.H. **Algoritmo de Ray-Casting para Visualização Volumétrica de Dados Obtidos por Tomografia Computadorizada.** Porto Alegre: Instituto de Informática/UFRGS, 1994.

[MAN96] MANSSOUR, I.H. **VisualMet:** Uma ferramenta para visualização exploratória de dados meteorológicos. Porto Alegre: CPGCC da UFRGS, 1996. Dissertação de Mestrado.

[MCC87] MCCORMICK, B. H.; DEFANTI, T. A.; BROWN, M. D. Visualization in Scientific Computing. **Computer**, Los Alamitos, v. 21, n. 6, 1987.

[MUK95] MUKHERJEA, S.; FOLEY, J. D.; HUDSON, S. E. Visualization complex hypermidia networks through multiple hierachical views. In: ACM SIGCHI, 1995. **Proceedings...** [S.l.: s.n.], 1995.

[MUN97] MUNZNER, T. H3: Laying out Large Directed Graphcs in 3D Hyperbolic Space. In: IEEE INFORMATION VISUZLIZATION, SYMPOSIUM, 1997. **Proceedings...** Los Alamitos: IEEE Computer Society, 1997. p. 2-10.

[NEU87] NEUWIRTH, C. Et al. The Notes Program: A Hypertext Application for Writing from Source Texts. In: CONFERENCE OF HYPERTEXT, 1987. **Proceedings...** [S.l.: s.n.], 1987. p 121-135.

[NOR92] NORTH, Stephen C. ***Neato*** **User's Guide**. Murray Hill, NJ: AT&T Bell Laboratories, 1992.

[NOR99] NORTH, Stephen C.; KOUTSOFIOS, Eleftherios. **Applications of Graph Visualization**. Murray Hill, NJ: AT&T Bell Laboratories, 1999.

[OLA93] OLABARRIAGA, Sílvia Delgado. Visualization of Oscillantions Data via Spirals. In: SIMPÓSIO BRASILEIRO DE COMPUTAÇÃO GRÁFICA E PROCESSAMENTO DE IMAGENS, SIBGRAPI, 1993, Recife. **Anais...** Recife: SBC/UFPE, 1993. p.81-88.

[PRI94] PRICE, B.A.; BAECKER, R.M.; SMALL, I.S. A Principle Taxonomy of Sofware Visualization. **Journal of Visual Languages and Computing,** [S.l.], v. 4, n. 3, p. 211-266, 1994.

[RIB94] RIBARSKY, W. et al. Glyphmaker: Creating Customized Visualizations of Complex Data. **Computer**, Los Alamitos, v. 27, n.7, p 57-64, 1994.

[RIL94] RYLOFF, E.A; LEHNEXT, W. **Information Extraction a Basis for High-Precision Text Classification System.** Disponível em: <http://www.cop.justresearch.com/>. Acesso em: 10 mar. 2001.

[ROB91] ROBERTSON, G. C.; MACKINLEY, J. D.; CARD, S. K. Cone Tree: Animated 3D Visualizations of Hierarchical Information. In: CONFERENCE HUMAN FACTORS IN COMPUTING SYSTEMS, CHI, 1991. **Proceedings...** [New York: ACM], 1991. p. 189-194.

[ROB93] ROBERTSON, G. Information Visualization using 3D Interative Animation. **Communications of the ACM**, New York, v. 36, n. 4, p. 57-71, 1993.

[SAR94] SARKAR, M.; BROWN, M.H. Graphical Fish-Eye Views of Graphs. In: CONFERENCE HUMAN FACTORS IN COMPUTING SYSTEMS, CHI, 1992. **Proceedings...** [New York: ACM], 1992.

[SAR96] SARKAR, M.; BROWN, M.H. Graphical Fisheye Views. **Communication of the ACM,** New York, v. 37, p.73-84, 1996.

[SCH96] SCHROEDER, Will; MARTIN, Ken; LORENSEN, Bill. **The Visualization Toolkit - An Object-Oriented Approach to 3D Graphics.** Upper Saddle River, New Jersey: Prentice-Hall, 1996. 826 p.

[SHA94] SHAPIRA, G.P.; CHAN, P.K.A.; RATHEUS, C.T. ***Systems for Know-Discovery in Databases.*** Disponível em: <http://info.gte.com/~kdd>. Acesso em: 20 jul. 2000.

[SIL96] SILBERSCHATZ, Avi; STONEBRAKER Mike; ULLMAN, Jeff. Data Base Research: Achievements and Opportunities into the 21st Century. **Journal of the Brazilian Computer Society**, Rio de Janeiro, v.2, n.3, 1996.

[SIL2000] SILVA, Marcelo R.M. **Alternativas de visualização de volumes baseadas em *ray casting*.** Porto Alegre: CPGCC da UFRGS, 2000. Dissertação de Mestrado.

[SPA97] SPARCK, J.K.; WILLET, P. **Readings in Information Retrevial**. San Francisco: Morgan Kaufmann, 1997.

[STA90] STASKO, John. T. Tango: A Framework and System for Algorithm Animation. **Computer**, Los Alamitos, v. 23, n. 9, p. 27-39, 1990.

[TIL96] TILEY, S.; SANTANA, P. Towards a Framework from Program Understanding. In: IEEE CONFERENCE ON PROGRAM COMPHRESSION, 1996, Berlin. **Proceedings...** [S.l.: s.n.], 1996, v. 4.

[WIE94] WIEBE, J. M. Tracking point of view in narrative. **Computational Linguistics**, [S.l.], v. 20, n. 2, 1994.

[YAT96] YATES, Ricardo Baeza. An extended model for full text databases. **Journal of the Brazilian Computer Society**, Rio de Janeiro, v. 2, n. 3, 1996.