Aplicando reuso de software na construção de ferramentas de engenharia reversa

Felipe Gouveia de Freitas
Julio Cesar Sampaio do Prado Leite
{freitas, julio}@inf.puc-rio.br
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro
R. Marquês de São Vicente, 255.
Rio de Janeiro - RJ - CEP 22453-900

Resumo

Engenharia reversa é uma área de interesse emergente principalmente pelo volume de sistemas legados que precisam evoluir. Diz-se engenharia reversa, porque ao contrário da engenharia tradicional, partimos do produto para a sua definição. Portanto, antes de evoluir um determinado sistema de software, faz-se necessário reverte-lo à descrições mais abstratas para então modificá-lo.
A engenharia reversa de sistemas representa um problema de grande complexidade computacional, no qual é constante a necessidade de interagir com humanos para complementar lacunas do conhecimento embutido em software legados. Por outro lado, é imprescindível o auxílio de ferramentas computacionais. Este trabalho propõe a aplicação do paradigma Draco na construção de ferramentas de engenharia reversa. A proposta foi parcialmente validada por um estudo de caso, no qual foram recuperadas informações a nível de desenho utilizando-se diferentes notações.
Palavras-chave: engenharia reversa, reuso de software, Draco-PUC, ferramentas de engenharia reversa.

Abstract

Reverse engineering deals with the challenges of deriving specifications from a detailed implementation. Our work proposes a singular approach to reverse engineering. We have built two domain modeling languages that make it possible the construction of reverse engineering tools by the reuse of such languages. This approach is based on the Draco paradigm and implemented in the Draco-PUC machine. We have used the domains to build a specific tool and applied it to recover the design of a public domain browser.

Keywords: reverse engineering, software reuse, Draco-PUC, reverse engineering tools.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

1. Introdução

O principal objetivo do estudo de engenharia reversa é o de se poder compreender artefatos de software já desenvolvidos. A motivação pode ser a manutenção do software, a migração, a reengenharia [Lei96] ou qualquer outra tarefa que envolva conhecimento a respeito do comportamento do software.

Os pesquisadores de engenharia reversa utilizam várias metáforas para descrever as atividades que são exercidas neste campo de pesquisa. Uma dessas metáforas, compara a atividade de engenharia reversa a de um médico radiologista que procura enxergar estruturas internas encobertas por outras partes do organismo. O médico, nesta especialidade, precisa de exames com uma grande precisão da estrutura interna para que consiga ter um diagnóstico para uma possível medicação ou cirurgia [WC96]. No caso de software, muitas vezes a engenharia reversa depara-se, justamente, com o problema de visibilidade da estrutura interna, o que torna a tarefa de compreensão ainda mais difícil.

A área de pesquisa de engenharia reversa está em sua infância, os trabalhos publicados, em sua maioria, narram experiências de explorações empíricas do problema sendo estes em geral independente dos trabalhos anteriormente publicados por outros autores, o que torna extremamente complicado coletar conceitos comuns aos trabalhos ou mesmo uma taxonomia dos termos utilizados. Um outro problema que enfrenta esta linha de pesquisa é o de se ter uma predominância de trabalhos realizados em cima de dados de testes inventados, em geral programas pequenos e conhecidos apenas pelo autor [PRE93].

O objetivo principal da engenharia reversa, a compreensão do software, só se dá completamente quando o ser humano é capaz de explicar o comportamento do programa, sua relação com o domínio de aplicação no qual este está inserido, e o seu efeito no ambiente da aplicação[TBD94]. Para atingir tal compreensão em grandes sistemas de software é imprescindível que se utilize ferramentas que auxiliem a aquisição deste conhecimento.

Existem muitos tipos de informações a serem extraídas, em diferentes tipos de situações, dependendo principalmente da disponibilidade e da precisão de informações sobre o software. Para realizar tal tarefa são necessárias uma variedade de abordagens e habilidades[WC96]. Portanto uma boa ferramenta de engenharia reversa não pode se limitar a um determinado tipo de informação.

Os principais requisitos de uma boa ferramenta de engenharia reversa seriam portanto:

- Ferramentas de engenharia reversa devem permitir seus usuários analisarem cada dimensão do software em separado mas também cruzando suas informações [PRE93].
- Deve ser possível coletar e armazenar informações sobre qualquer aspecto do software que se deseje.
- Deve-se possibilitar filtro das informações a serem analisadas a qualquer hora e em qualquer ponto.
- Deve-se ainda possibilitar a subdivisão do sistema de várias maneiras distintas.
- Por último deve-se facilitar a visualização dos resultados pelo usuário.

Considerando esses fatores, propomos uma arquitetura de software que possibilite a construção de ferramentas de engenharia reversa eficazes e adaptáveis. A adaptabilidade é conseguida através de reuso a nível de definição do problema.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

Neste trabalho, o foco principal é o que Biggerstaff chama de *programming concepts*, ou seja ferramentas de engenharia reversa que tem como base de informações apenas o código fonte dos sistemas, ao contrário de outros trabalhos[Pra92][Lei95] em que a fase de engenharia reversa é auxiliada por outros tipos de documentos e por experimentos práticos, *human concepts*, segundo Biggerstaff [TBD94].

A necessidade de reuso de código e de desenho em ferramentas de engenharia reversa apareceu em experiências anteriores [Bra96], aonde ficou evidente a necessidade de um mecanismo de reuso mais eficaz e automatizado. Em geral as ferramentas de engenharia reversa precisam atender a situações onde podem ocorrer as seguintes variações:

- Necessidade de se trabalhar com uma nova informação que não estava prevista anteriormente à ferramenta.
- Necessidade de se trabalhar com uma nova linguagem de programação.
- Necessidade de se trabalhar com novos tipos de filtro sobre os dados do sistema.
- Necessidade de se implementar novas heurísticas de extração de informações para sistemas com outras características de implementação.
- Necessidade de se criar uma nova forma de visualização dos dados.

A arquitetura que será apresentada está baseada no paradigma Draco de desenvolvimento de software[Nei84] [LSF94] e procura encapsular o conhecimento sobre a tarefa de engenharia reversa baseada em código fonte. Este conhecimento está detalhado na Seção 2. A Seção 3 faz um sumário das idéias do paradigma Draco e descreve a máquina Draco-PUC. A arquitetura de um gerador de ferramentas de engenharia reversa está descrita na Seção 4. Na Seção 5 demostramos o uso de uma ferramenta, gerada pela arquitetura, num navegador HTML de razoável complexidade. Concluímos, enfatizando os resultados obtidos e mapeando trabalhos futuros.

2. O Desenho de uma Ferramenta Típica de Engenharia Reversa

Da literatura de engenharia reversa, quatro trabalhos [Frei97] tiveram influência predominante na nossa proposta :

- o projeto GRASP[CH95] que tem como objetivo produzir representações gráficas de estruturas estáticas de programas através do código fonte,
- o projeto RECAST[EM93] que tem por objetivo criar um método, com auxílio de ferramentas, para realizar engenharia reversa de programas Cobol,
- o trabalho na MITRE[DAH96] que tem por objetivo construir uma biblioteca de transformações que reconhecem elementos de arquitetura em código fonte e
- o trabalho de Jarzabeck e Keam[JK95] que propõe uma ferramenta em que o resultado da extração da informação é armazenada em uma base de conhecimento, posteriormente utilizada para montar diagramas de desenho.

Com base na literatura e, principalmente, nesses trabalhos referidos acima, acredita-se que uma ferramenta de engenharia reversa (a partir do código fonte) deve atender a quatro pontos fundamentais, que seriam: como extrair as informações do código, como armazenar estas informações, como visualiza-las, e como visualizar apenas uma parte das informações.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

A primeira decisão dessa proposta foi a de utilizar um sistema transformacional na extração das informações [DAH96] e [JK95]. Em função disso escolhemos o sistema transformacional hoje disponível na máquina Draco-PUC, já utilizado anteriormente em sua própria re-engenharia [LPS92].

A segunda decisão foi a utilização de um banco de dados para armazenar as informações extraídas do código fonte [JK95]. Esta decisão teve os seguintes argumentos:

- Necessidade de se trabalhar com um grande volume de dados, pois diferentes visões do código fonte devem estar disponíveis.
- Necessidade de não haver nenhum limite quanto ao tamanho dos sistemas analisados, uma vez que o problema de engenharia reversa ocorre geralmente em grandes sistemas.
- Necessidade de acessar várias vezes as informações do sistema sem que seja necessário re-análise do código fonte.

A terceira decisão tomada foi ao que diz respeito a visualização das informações. Este problema foi apontado na Introdução, que deixa claro a necessidade de se poder trabalhar com tipos de informações completamente distintas, e ao mesmo tempo poder cruzar estas informações. Para tal, foi construído[Frei96] um visualizador de modelos[1] genéricos, utilizando a biblioteca gráfica multi-plataforma wxWindows[WxWin][Sma95], que permite a incorporação de novos modelos e que contém o conceito de link entre os modelos visualizados.

Estas decisões levaram a montagem de um desenho de uma típica ferramenta de engenharia reversa, que descrevemos a seguir.

O extrator seria um conjunto de transformações que trabalhariam na DAST, *Draco Abstract Syntax Tree*, dos programas a serem recuperados. As informações extraídas seriam armazenadas num banco de dados. Para banco de dados escolhemos MetalBase [Mbase], que está disponível na internet e implementa, a menos de junções, um banco de dados relacional.

O visualizador foi implementado e denominado VG (visualizador gráfico). Ele trabalha lendo um arquivo numa linguagem genérica de atributos, LA (permite que a descrição dos objetos do modelo, sejam nós ou arcos, seja feita através de seus atributos), e carregando classes dinamicamente permite visualização e navegação entre modelos recuperados [Frei96].

Para interligar o banco de dados com o visualizador é necessário exportar os dados recuperados na extração para arquivos na linguagem de atributos, LA. Para isto precisamos de procedimentos que, acessando a base de dados, criassem os arquivos.

O toque final nesse desenho era possibilitar que essa ferramenta típica pudesse filtrar as informações, visto que um dos problemas da engenharia reversa é justamente o de se trabalhar com um conjunto bem delineado de informações. A conclusão a que chegamos, a respeito do uso de filtros na arquitetura das ferramentas, é a de que o melhor lugar para que os filtros sejam utilizados é quando da exportação dos dados da base para o visualizador. Isto porque, neste ponto, como existe toda uma base de dados preenchida, têm-se uma visão global do software. Portanto, têm-se a possibilidade de implementar filtros de melhor qualidade.

---

[1] Um modelo, em geral, é um grafo que tem uma determinada semântica. São exemplos de modelos: o diagrama de módulos de um sistema [Boo94], um Modelo de Associação de Classes [RBPE91] e o digrama de Entidades Estruturadas [Mas90].

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

A seguinte figura sintetiza a proposta:

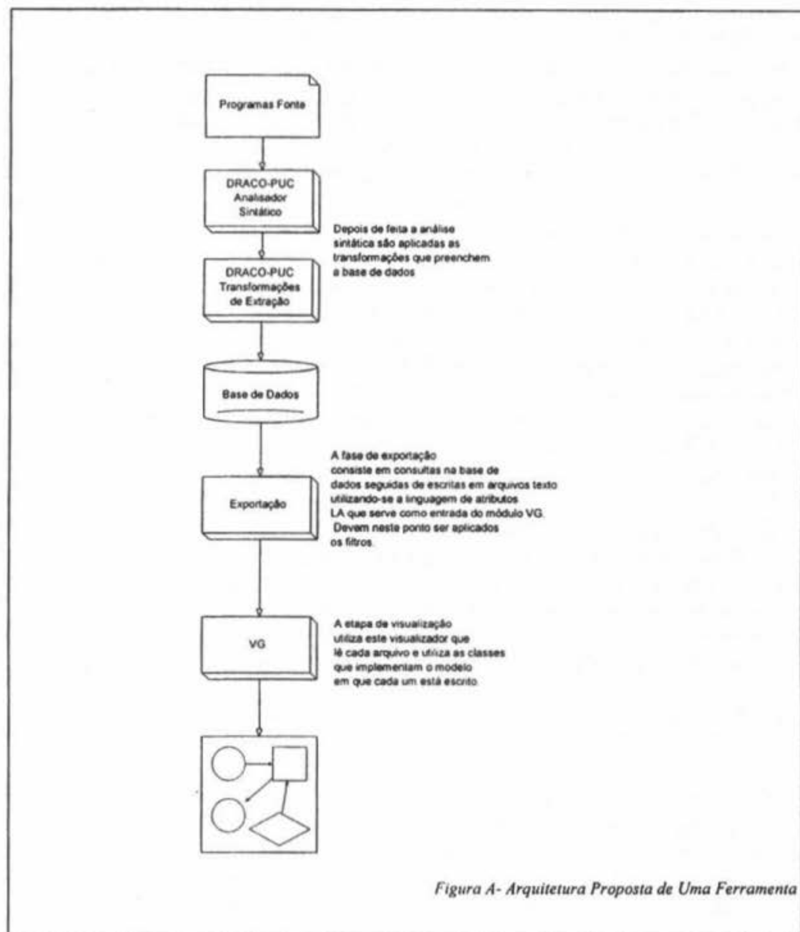


Figura A- Arquitetura Proposta de Uma Ferramenta

3. O Paradigma Draco

O paradigma Draco pode ser visto, e desta forma caracterizado, por vários pontos de vista[Free87]. A motivação original para o seu desenvolvimento foi o de prover uma forma de construção de sistemas de software a partir de componentes preexistentes, ao contrário de construir o sistema através de métodos tradicionais. Um ponto de vista próximo seria o de

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

construir sistemas através de um gerador de geradores, conseguindo, desta forma, gerar e manter facilmente uma classe de sistemas similares. Um terceiro ponto de vista é o de construir linguagens de especificação de alto nível para serem utilizadas na construção de sistemas, e por mecanismo de transformações produzirem os programas em linguagens executáveis.

O que está por traz destas definições é uma forma de organizar componentes de software para reuso diferente das disponíveis em linguagens de programação: bibliotecas de funções, ou bibliotecas de classes. No paradigma Draco, o elemento de reuso são linguagens formais que são denominadas domínios. Estas linguagens são construídas com intuito de encapsular os objetos e operadores de um determinado domínio, ou área de conhecimento. Os programas escritos nas linguagens dos domínios por sua vez devem sofrer um processo de refinamento até atingirem uma linguagem executável.

Para que se possa produzir software no ponto de vista do paradigma Draco, é necessário portanto que o processo seja dividido em duas etapas distintas: o encapsulamento do conhecimento do domínio, e a utilização do conhecimento encapsulado.

A máquina Draco-PUC, por sua vez, é um sistema de software que tem como objetivo principal implementar o paradigma Draco. O papel da máquina é o de com seu uso construir uma implementação real por tradução de todas as linguagens de domínio. Como a tradução ocorre entre os domínios (linguagens) surge a necessidade de domínios executáveis por exemplo: C, Pascal, Basic, Lisp entre outras. Estes são então os domínios alvos de nossas transformações.

O processo em que uma especificação escrita em um domínio fonte atinge um domínio alvo pode passar por inúmeros domínios intermediários. Esta propriedade diferencia o Draco-PUC de um Gerador de Geradores simples, pois além de construir o Gerador a máquina Draco-PUC permite que isto seja feito reutilizando outros domínios, o que acreditamos facilitar bastante o trabalho de construção das transformações de traduções.

Descrições mais detalhadas do paradigma Draco podem ser encontradas em versões anteriores do SBES[LPS92][LSF95], além de publicações internacionais[Nei84][LSF94].

4. A Rede de Domínios Draco Construída para a Geração das Ferramentas

Como uma ferramenta de engenharia reversa é por sua vez um sistema de software, e que por sua vez é especificado por uma série de linguagens distintas (linguagem de transformações, linguagem de definição de dados, C na exportação, e C++ nos modelos), podemos observar a dificuldade de se propor um mecanismo de reuso baseado em uma linguagem de programação. Portanto, conforme observamos na Seção 2, há a necessidade de um paradigma mais robusto que permita a geração de famílias de ferramentas. O paradigma Draco preenche exatamente essas necessidades, porque prevê a geração automática de múltiplos documentos, ou arquivos, em linguagens distintas[LSF94].

Ao contrário do que propõe o trabalho no MITRE [DAH96] onde o projetista necessita escrever as transformações específicas para recuperação de programas reutilizando eventualmente bibliotecas de transformações, a arquitetura proposta poderá recuperar estas informações utilizando uma linguagem de especificação de ferramentas de engenharia reversa apropriada à recuperação dos modelos que resolvem o seu problema, conseguindo com isso maior grau de reuso de software[Nei84] [LSF94].

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

O conceito utilizado neste trabalho foi o de se dividir os domínios (ou linguagens) da rede de domínios em três tipos[NBP89]: Domínios Executáveis são aqueles em que existe um compilador ou interpretador disponível fora do ambiente Draco; os Domínios de Aplicação que são aqueles que expressam a semântica existente em domínios do mundo real, ou de classes de problema; e os Domínios de Modelagem que são os domínios criados para facilitar a implementação dos domínios de aplicação diminuindo a distância existente entre estes e os domínios de executáveis.

Uma rede de domínios tradicional tem como ponto de partida os domínios de aplicação, passam por domínios de modelagem, e chegam a domínios executáveis.

Para resolver o problema de ferramentas de engenharia reversa foi proposta a seguinte rede de domínios:

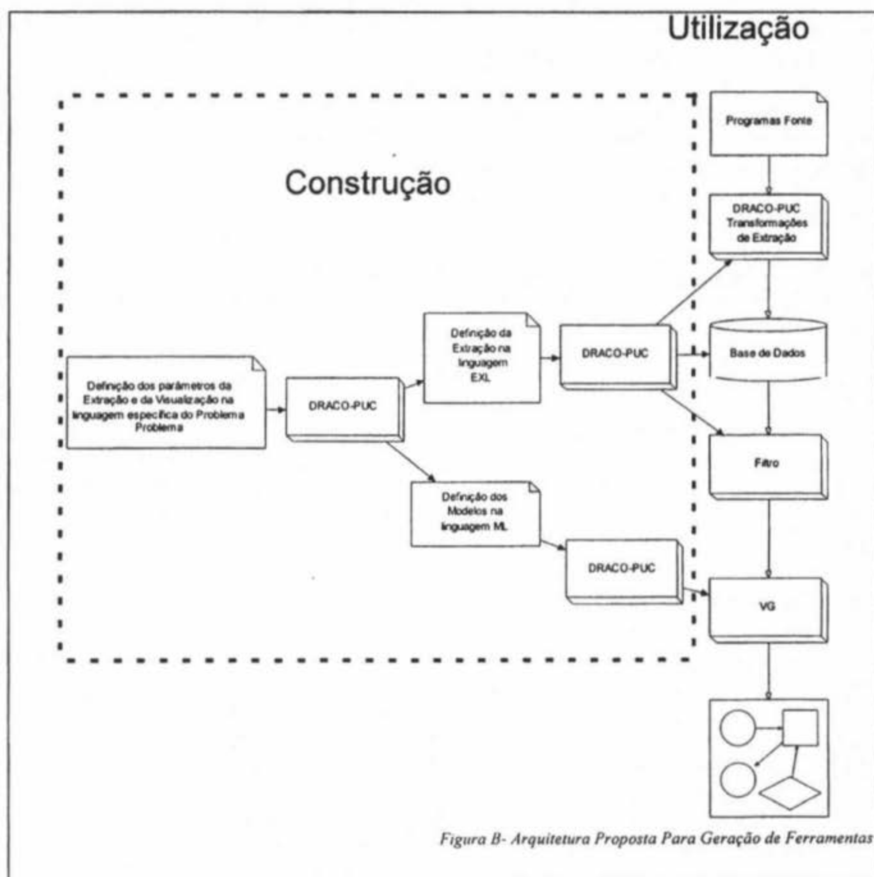


Figura B- Arquitetura Proposta Para Geração de Ferramentas

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

Nesta rede de domínios, observamos claramente a presença de dois domínios de modelagem: o domínio de extração e o domínio de modelos. O domínio de extração, denominado EXL (*Extraction Language*), é uma linguagem de alto nível que contempla a definição de dados, a extração de informações preenchendo a base de dados, e a exportação de dados para arquivos texto. O domínio de modelos, denominado ML (*Model Language*), é a linguagem de descrição dos modelos.

Estes domínios tornam a implementação de domínios de aplicação simplificada, podendo os projetistas destes domínios se preocuparem mais com as propriedades do domínio sem que tenham de se preocupar com banco de dados, linguagens de transformação e visualizadores entre outros detalhes.

Um exemplo de um domínio de aplicação seria o de ferramentas de recuperação que mostrem possíveis problemas enfrentados pelos sistemas no ano 2000 (conhecido como problema do ano 2000). No ano 2000 estima-se que uma grande quantidade de sistemas de software deixarão de funcionar pois previram apenas dois dígitos para se armazenar a data. Por este motivo existe um trabalho extremamente importante de engenharia reversa para indicar que pontos do sistema devem ser mexidos e que reflexos isto teria no resto do sistema.

Esta classe de problemas, seguindo a arquitetura proposta, teria sua linguagem própria com os respectivos parâmetros, como a linguagem de programação utilizada ou critérios utilizados na implementação. Os programas escritos nesta linguagem produziriam as ferramentas e os modelos a serem utilizados, seguindo o desenho expresso na Seção 2.

Com base nesta arquitetura o processo de construção e uso das ferramentas teria as seguintes etapas para um domínio de aplicação hipotético D:

1. O encapsulamento do conhecimento do domínio de engenharia reversa D, que significa construção da linguagem D e nas transformações que produzam os modelos utilizados, a especificação da extração no formato EXL, e dos modelos na linguagem ML.
2. A instanciação da ferramenta para um problema da classe D.
3. A aplicação das transformações nos programas fontes e a visualização dos modelos.

Na implementação desta arquitetura [Frei97], houve uma preocupacão na disponibilização de um protótipo operacional. Portanto, deixamos para uma segunda etapa a exploração de domínios específicos de aplicação e implementamos os domínios de modelagem. Deste modo a implementação atual consiste das linguagens EXL e ML, descritas a seguir.

A linguagem EXL

A linguagem EXL possui uma divisão em três seções: a seção de definição de dados, a seção de definição dos métodos de extração, e a definição da exportação. Apesar de estarem separados em seções, estas três partes da linguagem estão integradas. Por exemplo, se acrescentarmos um campo a uma tabela, automaticamente a semântica da inclusão nesta tabela será modificada na seção de extração, e a semântica da leitura desta tabela também mudará na seção de exportação.

O projeto desta linguagem sofreu forte influência de linguagens de quarta geração presentes em gerenciadores de banco de dados. Por este motivo contém comandos de alto nível freqüentemente encontrados nos gerenciadores como por exemplo: FOR_EACH e EXIST. Por outro lado, na seção de extração, EXL procura simplificar o trabalho de extração

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

de informações do código fonte através de transformações, contendo comandos mais enxutos para tais tarefas.

A seguir será mostrado um exemplo da parte de extração de dados:

```
EXTRACTION SECTION

PUBLIC BOTTON_UP_METHOD Analisys

IF_MATCH ((dast cpp.class_specifier
  class [[CLASS_NAME CN]] (
  [[member_declaration *mdef]]
)
))
 DO
  GET_VAR CN AT class_name;
  GET_VAR CN AT classe.nome;
  INSERT_REG classe;
  strcpy(modper.nomeclasse,class_name);
  strcpy(modper.nomemodulo,module_name);
  INSERT_REG modper;
  CALL_METHOD MetInline AT mdef;
END_IF
```

Este exemplo mostra um método de extração e uma regra de reconhecimento de padrão. A regra reconhece o padrão especificado no comando IF_MATCH que contém a variável CN do tipo CLASS_NAME. Ao reconhecer este padrão a regra copia o valor da variável CN para uma variável global "class_name" e para o campo "nome" da tabela "classe". Em seguida a regra insere o registro corrente na tabela "classe" registrando a existência da classe. O próximo passo é preencher o registro da tabela "modper" com os valores da classe atual e do módulo atual (variável global previamente preenchida) efetuando sua inserção. Por último a regra invoca o método "MetInline" sobre a variável "mdef" que buscará apenas dentro desta variável declarações de métodos inline.

Nesta regra observamos o uso dos comandos "GET_VAR", "INSERT_REG" para capturar e armazenar as informações sobre a definições de uma classe, e ainda o comando "CALL_METHOD" que significa uma chamada a outro método de extração.

O próximo exemplo é da parte de exportação de dados:

```
EXPORTATION SECTION

FUNCTION GeraModulos(file f1) AS void;
integer curr;

PRINTF f1,"[\nTitle='Modulos'\nobjfile='mod.of'\nBC=0\nArranged=0\n]\n";
 FOR_EACH modulo:indlmodulo DO
  sprintf(strtmp,"%s.vg",modulo.nome);
  IF modulo.nome[strlen(modulo.nome)-1]='h' THEN
     PRINTF f1,"Node %s
     [\nTitle='%s'\nX=100\nY=100\nW=80\nH=80\nLink(1)='Todas as
     Classes@classes.vg'\nLink(2)='Classes@%s'\nNodeType='NHeader'\n]\n",
     modulo.nome,modulo.nome,strtmp;
  ELSE
     PRINTF f1,"Node %s
     [\nTitle='%s'\nX=100\nY=100\nW=80\nH=80\nLink(1)='Todas as
     Classes@classes.vg'\nLink(2)='Classes@%s'\nNodeType='NBody'\n]\n",
     modulo.nome,modulo.nome,strtmp;
  END_IF
END_FOR
DO curr=0;
```

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

```
 FOR_EACH moddep DO
  IF EXIST modulo:indlmodulo(moddep.nomedependente) THEN
    IF EXIST modulo:indlmodulo(moddep.nome) THEN
      PRINTF fl,"Edge e%d [\nSource='%s'\nDest='%s'\nEdgeType='MDep'\n]\n",
          curr,moddep.nome,moddep.nomedependente;
      DO curr=curr+1;
    END_IF
  END_IF
 END_FOR

END_FUNCTION
```

Este exemplo mostra uma função de exportação que exporta para um arquivo "fl" todos os módulos e suas dependências.

Esta função não possui valor de retorno e utiliza uma variável local "curr" do tipo inteira. Ela realiza escritas no arquivo utilizando a sintaxe da linguagem LA, necessária ao módulo VG, através de comandos PRINTF. Primeiramente ela percorre a tabela de módulos gerando os nós do grafo; em seguida, a tabela de dependências ("moddep"), gerando os arcos do grafo. A variável local "curr" serve para produzir uma numeração seqüencial aos nós e arcos.

Nele podemos observar comandos de acesso ao banco de dados como "FOR_EACH", e "EXIST", que são bastante poderosos, e ainda a sintaxe de acesso às tabelas utilizadas, por exemplo: "modulo:ind1modulo(moddep.nome) ", que significa que estão sendo acessados os registros da tabela "modulo" pelo índice "ind1modulo" e que atendem ao critério "moddep.nome".

A linguagem ML

Para que pudéssemos completar a nossa rede de domínios, além da linguagem EXL responsável pela extração e exportação das informações, necessitávamos de uma linguagem que pudesse nos ajudar na construção dos modelos recuperados. Estes modelos são compostos por seus objetos que podem ser: Nós, Arcos ou uma Página. A página contém os nós e arcos, estes últimos por sua vez possuem um nó fonte e um destino.

Os modelos, como pudemos ver, são modelados de maneira mais fácil com o uso de orientação a objetos; portanto este foi o paradigma escolhido para construirmos o domínio ML de definição dos modelos.

O domínio ML foi implementado utilizando uma característica extremamente útil do Draco-PUC, a de permitir a escrita de programas utilizando-se múltiplas linguagens. Este domínio funciona integrado com o domínio C++, no qual devem ser escritos os métodos das classes escritas em ML. Neste domínio são descritos o comportamento da página, dos nós e dos arcos do modelo. Ele foi projetado como uma linguagem de definição de classes.

O exemplo a seguir mostra como foi definido o nó "Sequencial" do modelo de estruturas de JSD [Mas90].

```
NODE Sequential {
 METHOD: {{method_body CPP.function_definition.RC("vg.ctx")
virtual void Draw(VGDC *dc)
{
  float strw,strh;
  int strx,stry;
  dc->DrawRectangle(x, y,w,h);
  // Impede que a regiao externa ao no seja invadida
  dc->SetClippingRegion(x,y,w,h);
```

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

```
    // Calcula melhor posicao do titulo
    dc->GetTextExtent(title,&strw,&strh);
    strx=x+(w-strw)/2;
    stry=y+(h-strh)/2;
    DrawText(dc,title, strx,stry);
    // Desenha caracter diferenciador do nó
    DrawText(dc,"*", x+w-13,y+1);
    dc->DestroyClippingRegion();
}
}}
};
```

Nele está descrita a classe "Sequencial", que possui o método "Draw". A sintaxe da mudança de domínio aparece após a palavra "METHOD:", onde está indicado que houve uma mudança do domínio ML pela regra gramatical "method_body" para o domínio CPP pela regra gramatical "function_definition", utilizando um arquivo de contexto "vg.ctx"[Frei94].

O método de desenho traça o retângulo, em seguida o título, e, ao final, o carácter diferenciador do nó. Ele toma cuidado, através dos métodos "SetClippingRegion" e "DestroyClippingRegion", de não desenhar fora de sua região.

A maior vantegem da linguagem ML é que as classes definidas nesta linguagem encapsulam completamente os aspectos de carregamento dinâmico dos modelos, característica que não está disponível em C++ e que é bastante importante na arquitetura proposta.

5. Resultados Obtidos

Como estudo de caso, houve uma preocupação de se trabalhar com um sistema real, ou seja um sistema grande e desconhecido; ao contrário de exemplos onde são utilizados dados de teste pequenos e com resultados esperados conhecidos [PRE93].

A aplicação escolhida é de grande complexidade e de tecnologia de ponta. Ela chama-se wxWeb[WxWeb] e se constituí num browser HTML. Este browser possui a maioria das funcionalidades dos browsers mais utilizados Netscape e Internet Explorer, possuindo recursos como Mail e Autentificação.

Nossa meta foi validar a capacidade da arquitetura proposta. Para isto escolhemos três modelos como alvo da recuperação são eles: o diagrama de módulos de Booch [Boo94], o diagrama de classes de Rumbaugh [RBPE91] e o modelo de estruturas de JSD [Mas90]. O motivo da escolha foi a de utilizar modelos aceitos pela comunidade de Engenharia de Software que descrevem diferentes perspectivas de representação (estrutura do sistema, relacionamentos e estrutura interna).

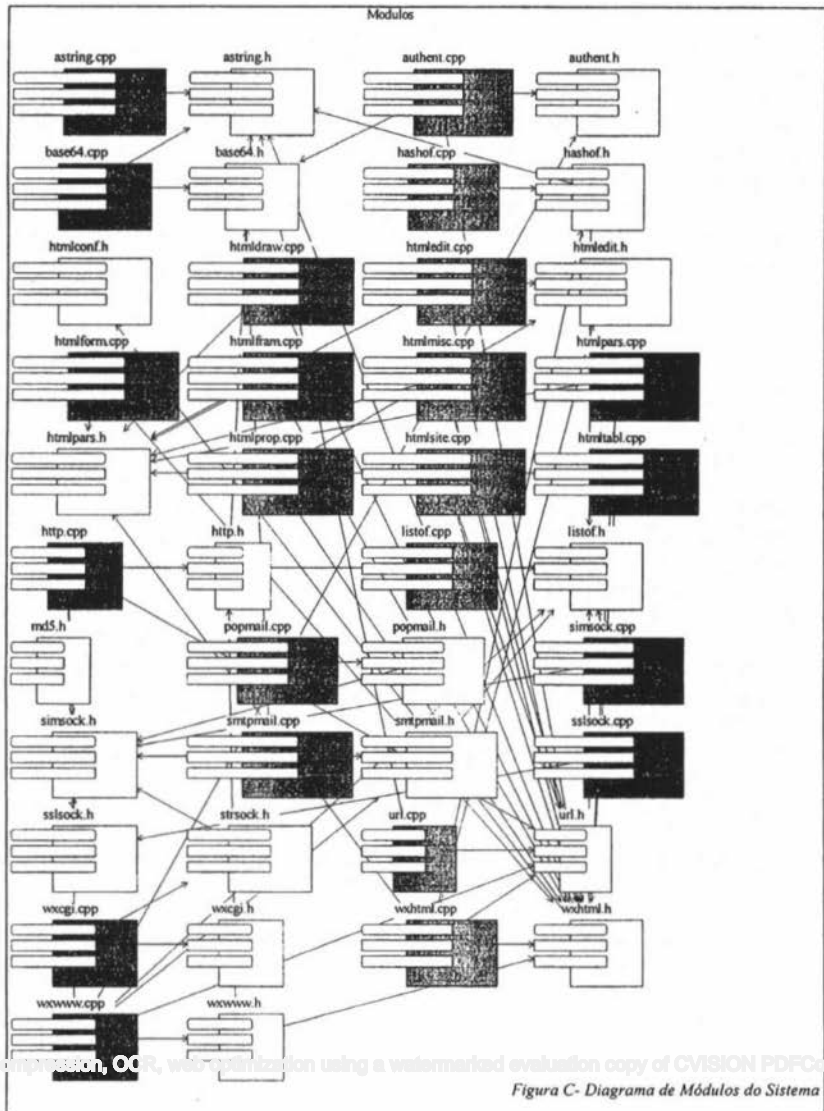
No caso do wxWeb fizemos dois programas (especificações) em **EXL** para retirar e exportar as informações e três programas em **ML**, referente aos três modelos citados acima. Os programas em EXL ficaram na ordem de 850 linhas cada (estes ficariam bem menores se não fosse a utilização de uma linguagem de gramática tão extensa como C++, na qual o número de regras de extração se torna elevado) e os programas em ML eram da ordem de 120 linhas cada. Dada a funcionalidade desses programas fica fácil verificar o ganho de produtividade em re-utilizar essas duas linguagens, ao invés de programar esta complexa funcionalidade (extração e exportação de informações) com uma linguagem de programação tradicional. Abaixo, mostraremos os resultados obtidos tanto para o caso global, como para um caso em que utilizamos .o conceito de filtro e centramos nossa atenção em uma determinada classe. Para cada um casos ressaltados abaixo, uma vez a base de dados

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

preenchida, se utiliza os três programas em ML para tornar possível a visualização através do programa VG [Frei96].

Recuperação Global

O primeiro programa EXL gera os diagramas de módulos e classes para todo o sistema, permitindo a navegação para os métodos e entre os métodos, podendo ainda voltar para as classes ou módulos. A Figuras 3 mostra o modelo de módulos para todo o sistema.



PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

Figura C- Diagrama de Módulos do Sistema

Do diagrama mostrado na Figura 3, modelos de módulos, pode-se ainda navegar para as classes definidas em um determinado módulo.

Fazendo isto para o módulo "Htmledit.h" temos o que mostra a Figura 4

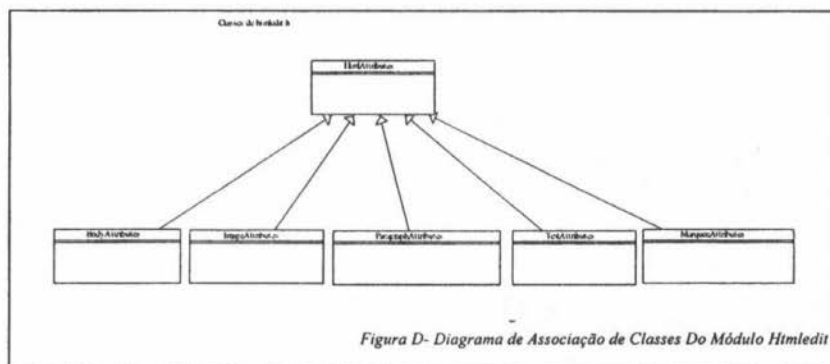


Figura D- Diagrama de Associação de Classes Do Módulo Htmledit

No modelo de classes, Figura 4, existe, por exemplo, a possibilidade de navegação para os métodos das classes ou ainda voltar para os métodos. Visualizando um determinado método temos por exemplo (Figura 6):

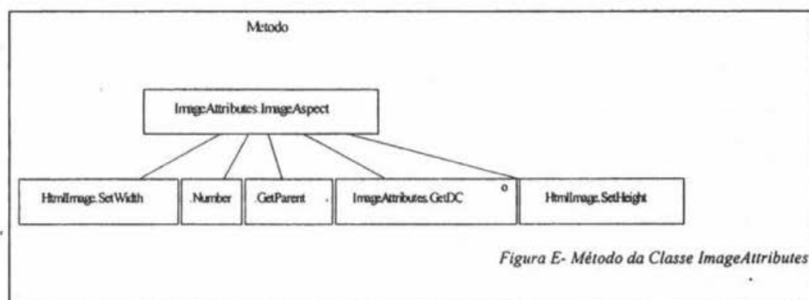


Figura E- Método da Classe ImageAttributes

Recuperação de Classe

Como pudemos observar, trabalhar com modelos contendo todas as classes e todos os módulos, apesar de darem uma visão global, não permitem um entendimento mais pormenorizado. Por este motivo foi desenvolvido um segundo programa EXL, que realiza um filtro maior e tem como objetivo mostrar apenas os dados de uma determinada classe. Ele mostra o módulo em que a classe está definida, os módulos que ele tem dependência, além da classe, das classes relacionadas com os respectivos métodos.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

Este segundo programa é um bom exemplo do caso em que procura-se entender uma determinada classe para termos uma clara visão de sua composição e de de seu relacionamento, com isso ajudando a evitar efeitos colaterais numa possível manutenção ou re-engenharia. A aplicação do exemplo a classe "SimSock" produziu as Figuras 6 e 7 .

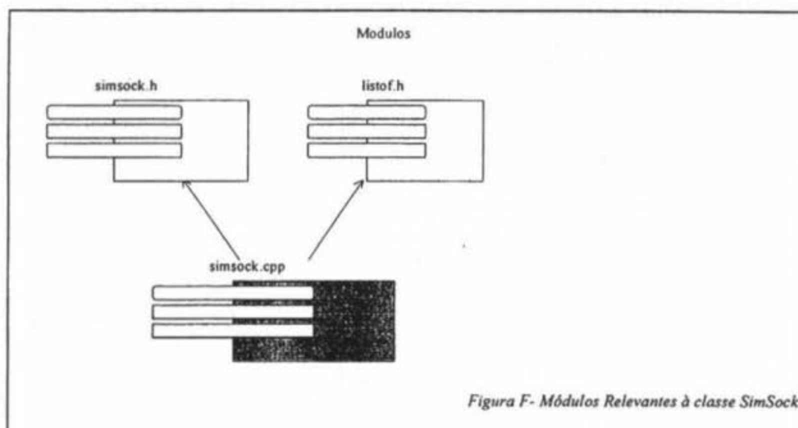


Figura F- Módulos Relevantes à classe SimSock

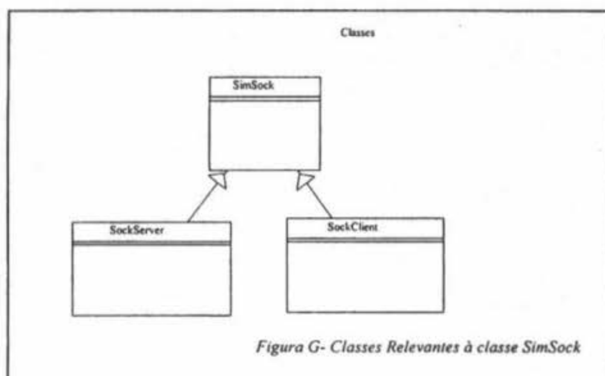


Figura G- Classes Relevantes à classe SimSock

6. Conclusão

A proposta de uma arquitetura de geradores de ferramentas de engenharia reversa [Frei97], que resumimos neste artigo, traz importantes contribuições tanto no campo da engenharia reversa como no campo de reuso. Mostramos que um desenho de uma ferramenta típica de engenharia reversa, a nível de *programming concepts*, pode ser generalizado através da aplicação do conceito de domínios [Nei84].

No estudo do domínio, conclui-se que dois domínios de modelagem eram necessários: a linguagem de extração e a linguagem de modelos. Esses domínios foram implementados

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

na máquina Draco-PUC e aplicados a um software de razoável complexidade e por nós desconhecido. O uso das linguagens demonstrou a capacidade de adaptação a outros casos e a aplicação demonstrou a praticabilidade da proposta.

Portanto, não só produzimos um protótipo de ferramenta de engenharia reversa que contribui para o estado da arte, como montamos a base de um domínio orientado para a geração de ferramentas. Consideramos nossa proposta de ferramenta de engenharia reversa uma contribuição na medida em que ao contrário de manter uma biblioteca de transformações [DAH96], organiza essas informações por domínios. Consideramos também que pelo fato de usarmos um banco de dados relacional [Mbase] temos mais flexibilidade de recuperação do que a proposta por Jarzabeck e Keam [JK95], mantendo como esse, uma independência em relação a linguagem sendo utilizada. Em termos de reuso, avançamos nosso conhecimento e prática do paradigma Draco, na medida em que dentro do projeto Draco-PUC construímos uma rede de domínio complexa e já no nível de domínios de modelagem, diferente dos trabalhos anteriores [LPS92] [LSF95][BPL96] que se basearam fortemente em domínios executáveis.

Vale lembrar que nossa proposta parte de uma filosofia, preponderante em engenharia reversa, que é a análise estática dos artefatos. Recentemente, estudos da capacidade reflexiva de linguagens [Cam96], mostram que visualizações da dinâmica de um programa podem aumentar a compreensão desses programas. Dentro do nosso projeto de pesquisa em engenharia reversa acreditamos, que conjuntamente com a evolução das propostas aqui apresentadas, como a construção de domínios de aplicação, devemos também investigar o uso da capacidade de reflexão para registro de comportamento como forma de ajuda à compreensão do software. Além disso temos também que integrar os resultados a nível de *programming concepts*, aqui representados, com outros trabalhos que consideram também o nível de *human concepts* [PRA92] [Lei95] [PGM95].

7. Bibliografia

[Boo94] G. Booch. *Object Oriented Analysis and Design with Applications*, second edition. Benjamin/Cummings 1994

[BPL96] Bergmann, U, Prado, A. F., Leite, J.C.S.P., Desenvolvimento de Sistemas Orientados a Objetos Utilizando o Sistema Transformacional Draco-PUC, *Em Anais do X Simpósio Brasileiro de Engenharia de Software*, Sociedade Brasileira de Engenharia de Software, José Carlos Maldonado, ed., São Carlos, 1996, pp. 173-188.

[Bra96] C.O. Braga. *Documentu* Um sistema para a geração de documentação hipertextual para o desenvolvedor de software. Dissertação de Mestrado; Departamento de Informática; Puc-Rio; 1996

[Cam96] Campo, M.R. , Price, R.T. Um Framework Reflexivo para Ferramentas de Visualização de Software, *Em Anais do X Simpósio Brasileiro de Engenharia de Software*, Sociedade Brasileira de Engenharia de Software, José Carlos Maldonado, ed., São Carlos, 1996, pp. 153—172.

[CH95] James H. Cross, T. Dean Hendrix. *Using Generalized Markup and SGML for Reverse Engineering Graphical Representation os Software.* Second Working Conference on Reverse Engineering, 1995.

[DAH96] D.R.Harris, A.S.Yeh, H.B.Reubenstein. *Extracting Architectural Features from Source Code.* Automated Software Engineering, 3(1/2), June 1996.

[EM93] H.E. Edwards; M. Munro. *RECAST - Reverse Engineering from COBOL to SSADM.* Working Conference on Reverse Engineering; May 1993, Baltimore, Maryland

[Free87] Peter Freeman. *A Conceptual Analysis of the Draco Approach to Constructing Software Systems.* IEEE Transactions on Software Enginnering, SE-13(7):830-844, July 1987.

[Frei94] F.G.Freitas . A Geração de Parsers da Máquina DRACO-PUC. Trabalho Final de Curso -

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

Engenharia de Computação; Departamento de Informática; Puc-Rio; 1994

[Frei96] F.G.Freitas . VG Um visualizador gráfico genérico. Projeto Final de Programação; Departamento de Informática; Puc-Rio; 1996

[Frei97] F.G.Freitas . Aplicando técnicas de reuso de software na construção de ferramentas de engenharia reversa. Dissertação de Mestrado; Departamento de Informática; Puc-Rio; 1997

[JK95] Stan Jarzabek, Tan Poh Keam. *Design of a Generic Reverse Engineering Assistant Tool.* Second Working Conference on Reverse Engineering, 1995.

[Lei95] Leite, J.C.S.P. Recovering business rules from structured analysis specifications, *In Proceedings of the WCRE'95*, IEEE Computer Society Press, Los Alamitos, CA, pp. 13-21

[Lei96] J.C.Leite. *Working Results on Software Re-Engineering.* Software Engineering Notes 21(2) March 1996.

[LPS92] Leite, J.C.S.P, Prado, A.F., Sant"Anna, M. Draco-PUC, Experiências e Resultados de Re-Engenharia de Software, *Em Anais do VI Simpósio Brasileiro de Engenharia de Software*, Sociedade Brasileira de Engenharia de Software, Daltro Nunes, ed., Gramado, 1992, pp. 115--128.

[LSF94] J.C.Leite, M.Sant'Anna, F.G.Freitas. *Draco-PUC: a Technology Assembly for Domain Oriented Software Development.* Third International Conference on Software Reuse (Proceedings) 1994.

[LSF95] Leite, J.C.S.P., Sant'Anna, M., Freitas, F.G., O Uso do Paradigma Transformacional no Porte de Programas Cobol, *Em Anais do IX Simpósio Brasileiro de Engenharia de Software*, Sociedade Brasileira de Engenharia de Software, Jäelson Castro, ed., Recife 1995, pp. 397--414.

[Mas90] Paulo C. Masiero. Análise de Sistemas pelo Método de Jackson. ICMSC-USP, 1990

[MBase] **http://cui.unige.ch/~scg/FreeDB/FreeDB.6.html**

[NBP89] James M. Neighbors, T. Biggerstaff, A. Perlis. *Draco: A Method for Engineering Reusable Software Systems.* Software Reusability, Addison-Wesley 1989.

[Nei84] James M. Neighbors. *The Draco Approach to Constructing Software from Reusable Components.* IEEE Transactions on Software Enginnering, SE-10(5):564-574, Sep. 1984.

[PGM95] Penteado, R.A.D., Germano, F.S.R., Masiero, P.C. Engenharia Reversa Orientada a Objetos do Ambiente StatSim, *Em Anais do IX Simpósio Brasileiro de Engenharia de Software*, Sociedade Brasileira de Engenharia de Software, Jaelson Castro, ed., Recife 1995, pp. 345—362.

[Pra92] Antônio F. do Prado. *Estratégia de Re-Engenharia de Software Orientada a Domínios.* PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro, 1992.

[PRE93] P.G.Selfridge, R.C.Walters, E.J.Chikofsky. *Challenges to the Field of Reverse Engineering.* Working Conference on Reverse Engineering, 1993.

[RBPE91] James Rumbaugh, Michael Blaha, William Premerlani, Fredrick Eddy and William Lorensen. *Object Oriented Modeling and Design.* Prentice Hall ISBN 0-13-629841-9, 1991

[Sma95] Julian Smart. *wxWindows User Manual.* University of Edinburgh. Artificial Intelligence Applications. Institute. 80 South Bridge, Edinburgh. 1995

[TBD94] T.J.Biggerstaff, B.G.Mitbander, D.E.Webster. *Program Understanding and the Concept Assignment Problem.* Communications of The ACM, 37(5), May 1994.

[WC96] Linda M. Wills, James H. Cross II. *Recent Trends and Open Issues in Reverse Engineering.* Automated Software Engineering, 3(1/2), June 1996.

[WxWeb] **http://www.ozemail.com.au/~adavison/wxweb.html**

[WxWin] **http://www.aiai.ed.ac.uk/~jacs/wxwin.html**

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor