UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

GERADOR DE DOCUMENTAÇÃO E APOIO A PADRONIZAÇÃO DE SOFTWARES IMPLEMENTADOS NA LINGUAGEM PROGRESS 4GL

JULIO ANDERSON MAAS

BLUMENAU
2004

2004/1-20

JULIO ANDERSON MAAS

GERADOR DE DOCUMENTAÇÃO E APOIO A PADRONIZAÇÃO DE SOFTWARES IMPLEMENTADOS NA LINGUAGEM PROGRESS 4GL

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Marcel Hugo – Orientador

BLUMENAU
2004

2004/1-20

GERADOR DE DOCUMENTAÇÃO E APOIO A PADRONIZAÇÃO DE SOFTWARES IMPLEMENTADOS NA LINGUAGEM PROGRESS 4GL

Por

JULIO ANDERSON MAAS

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Marcel Hugo – Orientador, FURB
Membro:	Prof. Everaldo Artur Grahl, FURB
Membro:	Prof. Roberto Heinzle, FURB

Blumenau, 12 de julho de 2004

Dedico este trabalho aos meus pais que sempre me incentivaram aos estudos e a todos os meus amigos que de alguma forma contribuíram para a realização deste.

RESUMO

Este trabalho apresenta a importância da existência de um padrão para o desenvolvimento de um sistema, dando ênfase para a padronização de nomenclaturas. Trata também da importância da documentação de sistemas apresentando algumas formas de documentação de códigos-fonte. E por fim apresenta uma ferramenta, voltada em especial para os desenvolvedores de sistemas que utilizam a linguagem *PROGRESS* para implementação, pois seu objetivo é auxiliar na documentação, através de relatórios de referências cruzadas e na padronização dos códigos-fonte escritos em *PROGRESS*.

Palavras chaves: *PROGRESS*, padronização, documentação.

ABSTRACT

This work presents the importance of the existence of a standard for the development of a system, giving emphasis for the standardization of nomenclatures. It also deals with the importance of the documentation of systems presenting some forms of code-source documentation. Finally presents a tool specially developed toward the developers of systems that use language PROGRESS for implementation, therefore its objective is assistant in the documentation, through reports of cross references and in the standardization of the codification written in PROGRESS.

Words keys: PROGRESS, standardization, documentation.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Este capítulo trata das considerações iniciais sobre o trabalho, sua importância, objetivos a serem alcançados e organização do trabalho.

1.1 ORIGEM

Todo e qualquer software é desenvolvido seguindo uma sequência básica de atividades, denominada de ciclo de vida (FERNANDES, 1995). Segundo (Schach, apud Feltrim, 1999), a fase reconhecidamente problemática refere-se à manutenção de software, que é responsável por custos de proporções superiores aos das demais fases. Porém, alguns fatores importantes podem ajudar a facilitar o processo de manutenção de software, como uma codificação padronizada e uma documentação consistente e atualizada.

No entanto, educar os desenvolvedores de sistemas, a fim de que estes obedeçam a certas regras para padronização e documentação de códigos-fonte, é normalmente uma tarefa árdua a ser realizada. Algumas vezes por culpa da pressa, outras por culpa do esquecimento, falta de costume, e até por relaxamento, acaba-se deixando de lado o cumprimento destas regras, incrementando assim a ausência de mais um requisito fundamental para a qualidade do processo do software.

Por este motivo este trabalho tem como foco principal a engenharia de software e será aplicado a uma das áreas da reengenharia da informação denominada de reestruturação de código-fonte. Segundo Furlan (1994), reestruturação de código-fonte é o processo de padronização de nomes e estruturação de programas sem alterar sua funcionalidade.

O processo de documentação e padronização pode se tornar muito mais fácil e agradável quando se dispõe de uma ferramenta de apoio. Conforme Staa (2000), ferramentas são programas que auxiliam no desenvolvimento de módulos, programas, documentação, projeto e especificações.

Contudo este trabalho propõe auxiliar o processo de documentação e padronização de códigos-fonte de softwares escritos em *PROGRESS 4GL*, através do desenvolvimento de uma ferramenta que:

a) auxilie a padronização de nomes de variáveis, *procedures*, parâmetros, arquivos, e até mesmo palavras reservadas da linguagem *PROGRESS 4GL,* sendo que o modelo de padrão a ser aplicado para a nomenclatura dos itens citados, ficará a critério do usuário da ferramenta, pois o mesmo vai dispor da possibilidade de configurar seu próprio padrão;
b) automatize a criação de dois modelos de documentos: um que apresentará uma relação de tabelas do banco de dados por arquivo de código-fonte; e outro que irá apresentar a relação de arquivos de código-fonte por tabelas do banco de dados. Para ambos os relatórios será verificado e descriminado o tipo de acesso ou transação feita na tabela referida.

Para implementação foi utilizada a ferramenta de desenvolvimento *PROGRESS 4GL "Progress Fourth Generation Language"* (PROGRESS, 2002), versão 9.1D. A escolha por esta ferramenta aconteceu principalmente por ela ser independente de plataforma, pois conforme Costa (2000), ela "funciona em praticamente todos os sistemas operacionais existentes como DOS, Windows 3x,95,NT, UNIX, OS/2, Novell, VMS, Motif, Xenix, CTOS entre diversos outros, isso utilizando o mesmo código fonte."

Novaes (2003) define *PROGRESS* como uma "linguagem de programação de 4ª geração que inclui um completo ambiente de desenvolvimento visual construído para aumentar a produtividade de desenvolvimento, melhorar as interfaces de aplicações para usuários e criar eficientes transações lógicas de negócios."

A ferramenta criada tem a utilidade de auxiliar o controle de padronização dos códigos-fonte, fazendo com que qualquer desenvolvedor de sistema tenha a possibilidade de identificar facilmente as diferenças entre variáveis, *procedures*, nomes de campos das tabelas e palavras reservadas da linguagem *PROGRESS 4GL*, através dos tipos de nomenclatura, definidas pelo usuário na própria ferramenta.

Além da padronização, a ferramenta auxilia no processo de documentação dos códigos-fonte, através de uma varredura em busca de comandos que façam qualquer tipo de acesso ao banco de dados. Este item é de grande utilidade para que os desenvolvedores de software possam manter a documentação de seus sistemas sempre atualizada, tornando mais rápida e precisa a manutenção do software. Esta ferramenta serve tanto para empresas que já

possuem seus produtos acabados, como para empresas que estão com seus sistemas em pleno estado de desenvolvimento.

1.2 OBJETIVO DO TRABALHO

O objetivo deste trabalho é possibilitar aos profissionais da área de desenvolvimento de sistemas um processo automatizado de documentação e auxílio à padronização dos códigos-fonte de sistemas escritos na linguagem *PROGRESS 4GL*.

Os objetivos específicos do trabalho são:

a) relacionar detalhadamente para cada rotina, ou apenas pelas rotinas selecionadas pelo usuário do sistema, todos os acessos que ela faz ao banco de dados, apresentando, para cada campo de cada tabela acessada, o tipo de acesso efetuado, ex: leitura, inserção, alteração, exclusão;
b) relacionar, a partir das tabelas selecionadas pelo usuário do sistema, todas as rotinas que fazem algum processo de leitura, inserção, alteração ou exclusão de dados nestas tabelas;
c) relacionar, a partir de um padrão definido pelo usuário desta ferramenta em uma tela para configuração de padrões, todas as rotinas onde encontram-se códigos-fonte escritos fora dos padrões especificados pela sua empresa, e já propor a correção desta codificação.

1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho é composto por cinco capítulos. No primeiro capítulo, foram apresentados a origem do trabalho, objetivo e organização.

No segundo capítulo são apresentadas algumas características mais relevantes da linguagem *PROGRESS* 4GL e também é abordado o conceito de linguagens de quarta geração.

No terceiro capítulo é apresentada a importância da existência da documentação e padronização na codificação de programas.

O quarto capítulo apresenta a ferramenta desenvolvida juntamente com sua especificação e resultados obtidos.

Finalmente no quinto capítulo são apresentadas as conclusões e sugestões a respeito do trabalho desenvolvido.

2 PROGRESS

Neste capítulo será dado foco principal à ferramenta de desenvolvimento *PROGRESS* versão caracter, já que um dos objetivos principais do trabalho é o desenvolvimento de uma ferramenta que possa ser executada tanto no sistema operacional *WINDOWS* quanto no *UNIX*.

2.1 4GL

A quarta geração das linguagens de programação (4GL - *Fourth Generation Language*) começou a ser desenvolvida a partir de 1986 e tem como características principais a geração de sistemas especialistas, o desenvolvimento de inteligência artificial e a possibilidade de execução dos programas em paralelo(GONÇALVES, 2004)**.**

A maioria das linguagens de programação comumente utilizadas, como C, PASCAL, JAVA, são chamadas de linguagens de terceira geração (3GLs - *Third Generation Languages*). Estas linguagens permitem que o programador controle o computador num nível mais baixo, contudo a utilização de uma sintaxe em Inglês resulta em um código-fonte compreensível (GONÇALVES, 2004)(PROGRESS, 2002).

As linguagens de quarta geração, conhecidas também como linguagens artificiais, contém uma sintaxe distinta para representação de estruturas de controle e dos dados. Essas linguagens, por combinarem características procedurais e não procedurais, representam estas estruturas com um alto nível de abstração, eliminando a necessidade de especificar algoritmicamente esses detalhes (GONÇALVES, 2004).

Uma linguagem de quarta geração proporciona uma programação mais facilitada ao implementador e ainda a flexibilidade das linguagens de programação mais utilizadas, pois segundo Ribeiro (2004) e Gonçalves (2004), alguns dos principais benefícios proporcionados pelas 4GL são:

a) Melhora na determinação dos requisitos: menor chance de erros de interpretação por especialistas técnicos;
b) Redução nos atrasos da aplicação: com o elevado nível de abstração, o programador pode concentrar-se mais na solução do problema ao invés de preocupar-se como o hardware vai tratar do problema;

Pode-se tomar como exemplo o Quadro 1, que mostra um programa em *PROGRESS* que percorre uma tabela do banco de dados e apresenta todas os seus dados na tela.

```
FOR EACH tbEmpresa:
    DISPLAY tbEmpresa.
END.
```

Quadro 1 – Exemplo de código-fonte *PROGRESS*

Este compacto *loop*, sendo que "tbEmpresa" é o nome de uma tabela do banco de dados, executa as seguintes funções:

a) Percorre todos os registros da tabela "tbEmpresa", partindo do primeiro registro;
b) Verifica o modo de apresentação definido para cada campo da tabela;
c) Cria para cada campo seus respectivos modos de apresentação de dados;
d) Mostra todos os registros da tabela adaptando cada valor ao seu modo de apresentação definido;
e) Aguarda um sinal do usuário para a apresentação dos próximos registros.

Para cada tarefa executada acima, seriam necessárias várias linhas de código para uma 3GL. Uma 4GL executa todas estas tarefas implicitamente.

As linguagens de quarta geração fazem suposições úteis do que o programador deseja, dispensando grande parte das entediantes tarefas de codificação de mínimos detalhes. Por exemplo, observe o comando *DISPLAY* no Quadro 1, ele não exige nenhum detalhe sobre como formatar a saída. O *PROGRESS* constrói um formato de saída padrão combinando a informação do formato para cada campo da base de dados com os algoritmos de saída. Esta manipulação intuitiva de comandos é chamada de procedimento padrão. Esta vantagem não chega a atrapalhar a customização do programa, pois cada procedimento padrão do *PROGRESS* inclui as opções que permitem sobrescrever os mesmos (PROGRESS, 2002).

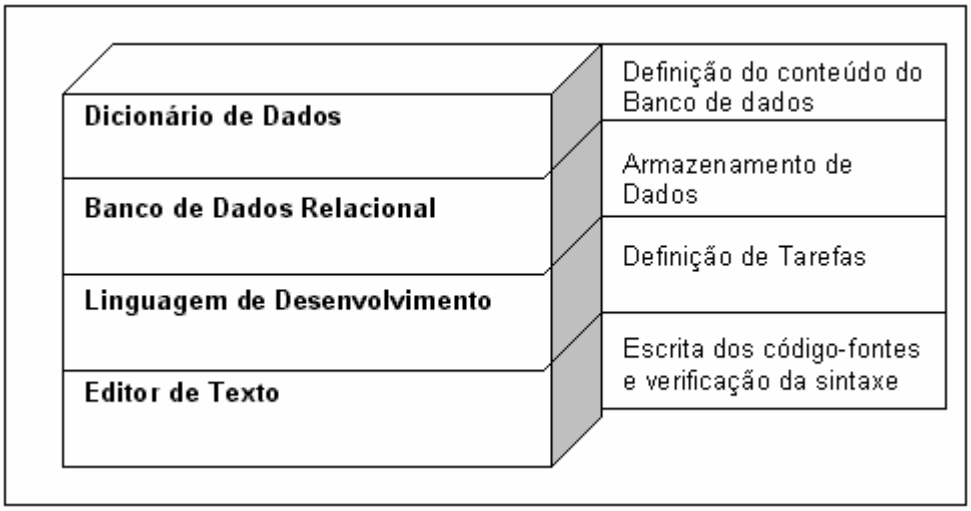
2.2 ORIGEM DO *PROGRESS*

A linguagem *PROGRESS* foi desenvolvida pela empresa *Progress Software Corporation* em 1984, na cidade de Bedford, Massachusetts USA. Inicialmente para sistema operacional UNIX com uso em *mainframes* para processamento de grande volume de dados, como alternativa para outras linguagens da época como Cobol, Adabas, Natural, etc, que

exigiam do programador escrever um código-fonte muito extenso para qualquer aplicação. Também uma alternativa como Banco de Dados Relacional de alta performance e segurança, embutido em um único produto (COSTA, 2000).

2.3 CARACTERÍSTICAS *PROGRESS*

Basicamente *PROGRESS* é uma linguagem de desenvolvimento de aplicações e um sistema gerenciador de banco de dados relacional. Portanto, a linguagem e o banco de dados são chamados de *PROGRESS Fourth Generation Language(4GL)* e *Relational Database Management System(RDBMS)*. A Figura 1 mostra os componentes do produto *PROGRESS 4GL/RDBMS*(PROGRESS, 2002).



Fonte: Progress (2002, p. 1-5)

Figura 1 – Componentes do PROGRESS 4GL/RDBMS

PROGRESS 4GL/RDBMS é uma ferramenta pertencente ao pacote *PROGRESS PROVISION*® que é um ambiente de desenvolvimento baseado em componentes e oferece um conjunto de ferramentas voltadas para o desenvolvimento de aplicações *e-busines* (PROGRESS, 2001).

Segundo Costa (2000), *PROGRESS* suporta tanto uma instalação e armazenamento centralizados, ou seja, banco de dados e aplicações em um servidor disponível para acesso de diversos clientes em rede, como também pode ser configurado para trabalhar com múltiplos servidores, banco de dados distribuídos entre servidores e/ou plataformas distintas.

2.4 SINTAXE PROGRESS

Serão apresentadas a seguir algumas particularidades mais relevantes pertencentes à linguagem *PROGRESS*.

2.4.1 COMENTÁRIOS

Os comentários são trechos de código-fonte que são ignorados pelo compilador. Deve-se utilizar deste artifício para esclarecimento de rotinas, motivo de execução ou qualquer outro tipo de informação que possa ser passado para o desenvolvedor e que venha a auxiliar na legibilidade do código-fonte.

A sintaxe *PROGRESS* reconhece uma instrução em comentário quando a mesma estiver delimitada pelos caracteres "/*", início do comentário e "*/", fim de comentário.

No Quadro 2 pode-se visualizar um código-fonte em que o trecho que está em negrito é um comentário no programa.

```
/* Listagem dos Funcionários da Empresa 2 */

FOR EACH tbFunc WHERE tbFunc.bdEmpresa = 2 NO-LOCK:
      DISP tbFunc.bdCodFunc tbFunc.bdNomeFunc. /*tbFunc.bdDataNasc*/
END.
```

Quadro 2 – Exemplo de Comentário em *PROGRESS*

2.4.2 ABREVIATURA DE EXPRESSÕES

Embora em muitos casos não seja recomendável por atrapalhar a legibilidade do código-fonte, a sintaxe *PROGRESS* permite a abreviatura de certos comandos que podem tanto ser palavras reservadas como campos de tabelas.

```
DEFINE VARIABLE lch_testeA AS CHARACTER NO-UNDO.

DEF VAR lch_testeB AS CHAR NO-UNDO.

DEFIN VARIA lch_testeC AS CHARAC NO-UNDO.
```

Quadro 3 – Exemplo de abreviações de códigos-fonte em *PROGRESS*

Repare nas três declarações de variáveis constantes no Quadro 3, o código-fonte apresentado é perfeitamente compilável e sem que apresente qualquer mensagem de erro, pois todas as três declarações estão corretas e tem a mesma funcionalidade, mudando apenas os nomes das variáveis. Agora repare no exemplo do Quadro 4, o quanto pode se tornar prejudicial para legibilidade se mal aplicado este recurso.

```
FOR EACH tbEmpresa WHERE tbEmpresa.bdC < 10 NO-LOCK:
    DISPLAY tbEmpresa.bdN tbEmpresa.bdR tbEmpresa.bdI
            WITH FRAME frEmpresa.
END.
```

Quadro 4 – Exemplo de abreviatura dos campos de uma tabela em *PROGRESS*.

Para um programador que não possui um profundo conhecimento do sistema e de sua base de dados, fica muito difícil compreender o que o código-fonte no exemplo do Quadro 4, deve executar. Os campos da tabela de empresas do exemplo anterior estão declarados no dicionário de dados como: bdCodEmpresa; bdNomeResumido; bdRazaoSocial; bdInscrEstadual. Mesmo assim a sintaxe *PROGRESS* permite que no lugar de "tbEmpresa.bdNomeResumido" seja utilizado no código-fonte apenas "tbEmpresa.tbN", porém isto só será possível, se não existir outro campo do cadastro de Empresas que contenha as iniciais "tbN".

Este pode se tornar um recurso muito bom desde que se estabeleça um padrão para que seja usado sempre os mesmos tipos de abreviações.

2.4.3 VARIÁVEIS PROGRESS

De acordo com Costa (2000), para uma variável em *PROGRESS* não é apenas definido o tipo de dado, pois o *PROGRESS* trata uma variável como sendo uma entidade com atributos e eventos, ou seja, um objeto. Tanto que em uma variável pode-se configurar *label*, cor, formato, visualização, eventos, etc.

```
DEFINE [ [ NEW [ GLOBAL ] ] SHARED ] VARIABLE variable
  { AS datatype | LIKE field }
  [ NO-UNDO ]
  [ BGCOLOR expression ]
  [ COLUMN-LABEL label ]
  [ DCOLOR expression ]
  [ DECIMALS n ]
  [ DROP-TARGET ]
  [ EXTENT n ]
  [ FONT expression ]
  [ FGCOLOR expression ]
  [ FORMAT string ]
  [ INITIAL
      { constant | { [ constant [ , constant ] ... ] } }
  ]
  [ LABEL string [ , string ] ... ]
  [ MOUSE-POINTER expression ]
  [ [ NOT ] CASE-SENSITIVE ]
  [ PFCOLOR expression ]
  { [ view-as-phrase ] }
  { [ trigger-phrase ] }
```

Fonte: *Help PROGRESS*

Quadro 5 – Definição de uma variável em *PROGRESS*

Conforme testes realizados, a nomenclatura de uma variável *PROGRESS*, pode conter no máximo 32 caracteres. Obrigatoriamente toda variável deve ser explicitamente declarada.

Os tipos de variáveis disponíveis no *PROGRESS* são :

a) *Character* : pode conter qualquer dado tipo texto e suporta até 3.000 caracteres;
b) *Date* : pode conter datas de 01/01/32.768 A.C até 31/12/32.767 D.C;
c) *Decimal* : pode conter qualquer número sendo de no máximo 50 dígitos para inteiros e decimais;
d) *Integer* : pode conter números inteiros de –2.147.483.648 até +2.147.483.647;
e) *Logical* : contém valores lógicos, TRUE/FALSE ou YES/NO;
f) *Handle* : armazena o endereço de memória de *procedures*, componentes ou parâmetros ativos no *PROGRESS;*
g) *Memptr* : especifica um ponteiro utilizado para referenciar DLL's;
h) *Raw* : armazena dados em nível de *byte*, utilizado para manipular dados atribuídos no banco desconsiderando as características quanto ao tipo definido;
i) *Recid* : pode guardar o endereço fixo de um registro da tabela em formato inteiro;
j) *Rowid* : pode guardar o endereço fixo de um registro da tabela em formato hexadecimal.
k) *Widget-Handle* : pode conter o endereço de memória de objetos.

2.4.4 INCLUDE FILES

São arquivos que podem ser utilizados como extensão de qualquer código-fonte, para tornar um programa mais organizado. Na maioria das vezes utilizados para declarações de variáveis globais, definições de *FRAMES* ou rotinas mais comuns ou específicas a um determinado assunto. Por convenção os arquivos *INCLUDES* são gravados com extenssão ".i"(PROGRESS, 2002)(PROGRESS, 1997).

Os arquivos *INCLUDES* são referenciados pelo programa principal através do nome do arquivo ".i" entre chaves. Exemplo "{ c:\padoc\IncludePadoc.i }".

2.4.5 ARGUMENTOS

São valores que podem ser passados de forma literal como parâmetro para arquivos de procedimento ".p" ou *includes* ".i". A desvantagem está na legibilidade que fica altamente prejudicada no programa que recebe o argumento. Isto por que no programa receptor o argumento é reconhecido por um número inteiro delimitado por chaves. Exemplo "{1}"(PROGRESS, 1997).

Para se passar um argumento como parâmetro não é necessário nenhum tipo de declaração ou tratamento especial. Basta se chamar um programa normalmente através do comando "RUN <nome do programa>", e logo após o nome do programa colocar os argumentos que se deseja referenciar, conforme Quadro 6.

```
/* A.p */

RUN c:\padoc\padoc.p "tbSistema" "bdCodSistema".

/* ou */

{ c:\padoc\padoc.i "tbSistema" "bdCodSistema" }
```

Quadro 6 – Exemplo de chamada de um programa com argumento.

No exemplo do Quadro 6, são referenciados dois argumentos sendo que o primeiro é a tabela "tbSistema" e o segundo é um campo desta tabela "bdCodSistema". Agora veja no Quadro 7, como estes argumentos podem ser aproveitados no programa que foi solicitado.

```
/* B.p */

FIND {1} WHERE {1}.{2} = 3 NO-LOCK NO-ERROR.

IF AVAILABLE {1} THEN
   DISP {1}.bdNomeSistema.
```

Quadro 7 – Exemplo de utilização de dois argumentos.

Repare que o tratamento dado a cada argumento do programa B depende da posição em que eles foram referenciados no programa A, por exemplo, o argumento "{1}" é tratado como uma tabela porque esta foi a primeira a ser referenciada no programa A.

É muito importante que quando da utilização deste recurso, esteja bem especificado no programa receptor a utilidade de cada argumento para que futuros programadores ao analisarem o código-fonte, tenham conhecimento da função de cada argumento.

2.5 ESTRUTURAS DE CONTROLE

Dentre as estruturas de controle existentes no *PROGRESS* serão citadas apenas aquelas consideradas mais importantes conforme (PROGRESS, 2002), dando ênfase à estrutura "*FOR EACH* ".

Além das estruturas já citadas anteriormente como, *INCLUDE FILES* e argumentos, o *PROGRESS* suporta a criação de "*PROCEDURES*" internas, funções, blocos de laços como as representadas nos quadros 8, 9 e 10:

```
/* Exemplo de utilização do comando DO */

DEFINE VARIABLE lin_conta AS INTEGER NO-UNDO.

DO lin_conta = 1 TO 10:
   DISPLAY lin_conta.
END.
```

Quadro 8 – DO TO.

```
/* Exemplo de utilização do comando DO WHILE */

DEFINE VARIABLE lin_conta AS INTEGER NO-UNDO.

DO WHILE lin_conta <= 10:
   DISPLAY lin_conta.
   lin_conta = lin_conta + 1.
END.
```

Quadro 9 – DO WHILE

```
/* Exemplo de utilização do comando REPEAT */

REPEAT:
   FIND NEXT tbSistema.
   DISPLAY tbSistema.bdCodSistema
           tbSistema.bdNomeSistema.
END.
```

Quadro 10 – REPEAT

No exemplo representado no Quadro 10, o conteúdo controlado pelo comando *REPEAT* é executado até que se atinja o último elemento da tabela. Quando isto acontecer, automaticamente a execução irá sair deste laço e prosseguir no programa.

```
/* Exemplo de utilização do comando FOR EACH */

FOR EACH tbFuncionario NO-LOCK BREAK BY tbFuncionario.bdSetor
                                     BY tbFuncionario.bdNomeFunc:

    IF FIRST-OF(tbFuncionario.bdSetor) THEN
       DO:
            DISPLAY "Funcionários do Setor " AT 12
                    tbFuncionario.bdSetor ":" WITH FRAME frSetor.

            CASE tbFuncionario.bdSetor:
                 WHEN 1 THEN
                      MESSAGE "Diretoria" VIEW-AS ALERT-BOX.
                 WHEN 2 THEN
                      MESSAGE "Gerência" VIEW-AS ALERT-BOX.
                 OTHERWISE
                      MESSAGE "Demais Funcionários" VIEW-AS ALERT-BOX.
            END CASE.

       END. /* FIM DO IF */

    DISPLAY tbFuncionario.bdCodFunc tbFuncionario.bdNomeFunc
            WITH FRAME frDescrFunc CENTERED.

END. /* Fim do FOR EACH */
```

Quadro 11 – Exemplo comando estrutura FOR EACH, IF e CASE.

A estrutura FOR EACH é uma das mais importantes da sintaxe *PROGRESS.* Trata-se de um laço implícito, com alocação e leitura de registros, permitindo que se trabalhe com múltiplas tabelas, classificando a listagem dos registros por qualquer campo constante na tabela em evidência(PROGRESS, 2002).

No exemplo representado no Quadro 11, o comando *FOR EACH* percorre todos os registros da tabela "tbFuncionario". No entanto o comando *NO-LOCK* garante a integridade dos registros atribuindo-lhes a proteção de somente leitura. Existe ainda o comando *BREAK BY* que neste caso está referenciando os campos "bdSetor" e "bdNomeFunc". Este comando

associado ao *FIRST-OF* ou *LAST-OF*, possibilita fazer um controle do primeiro e último funcionário de cada setor listado. O Comando *BY* tem outra utilidade ainda que é a ordenação dos registros, que no caso acima, a listagem de funcionários irá ser apresentada por ordem de setor e por nome do funcionário.

No Quadro11, pode-se reconhecer ainda outras duas estruturas de controle, que são “IF THEN” e a estrutura “CASE”, que não possuem diferença funcional em relação as demais linguagens encontradas no mercado.

3 DOCUMENTAÇÃO E PADRONIZAÇÃO

Conforme Paula (2001), a qualidade de um produto pode ser avaliada conforme o seu grau de conformidade com os respectivos requisitos, e esta qualidade é decorrente diretamente da qualidade do processo utilizado na produção deste produto.

Segundo Pfleeger (2004), na grande maioria das empresas os softwares são desenvolvidos por equipes e não por um só programador, por esta razão, é muito importante que cada programador entenda não somente o que seu companheiro escreveu, mas porque escreveu e como este código se encaixa no seu trabalho. É necessário que estes conheçam os padrões e procedimentos de sua organização antes de começar a escrever o código-fonte. "Muitas empresas exigem o código em conformidade com o estilo, formato e os padrões de conteúdo, de modo que o código e a documentação associada sejam claros para qualquer um que os leia."(PFLEEGER, 2004).

Por estes aspectos, este trabalho enfoca a importância da documentação e da padronização para a produção de um software de qualidade.

3.1 DOCUMENTAÇÃO DE SOFTWARES

Barreto (1999) define documentação como o registro de informações produzidas por um processo ou atividade. Inclui planejamento, projeto, desenvolvimento, produção, edição, distribuição e manutenção dos documentos necessários a gerentes, engenheiros e usuários do software. Conforme Rocha (2001), "cabe à documentação registrar a evolução do software para que sejam criadas as bases necessárias para uma melhor utilização e manutenção do software."

De acordo com Duarte (2000), os documentos estão presentes em todas as fases da construção de um software, mostrando as negociações entre a empresa desenvolvedora do software e o cliente, o que o sistema faz, como ele será desenvolvido, as especificações e ainda como o usuário final deverá utilizá-lo.

Quando um software mostra-se bem documentado e com as suas informações atualizadas, demonstrando organização por parte dos desenvolvedores deste sistema, pode-se dizer que há um grande indício de que se trata de um produto de boa qualidade. Pois,

conforme alguns autores como Barreto (1999), Paula (2001), a qualidade de um produto final é o reflexo da qualidade dos processos que compuseram este produto.

A quantidade de documentos que envolvem um projeto de software é muito grande, pois trata-se desde um levantamento inicial de pesquisa de campo, ou levantamento das necessidades dos usuários, até os manuais para auxílio aos usuários finais para operacionalidade do sistema.

Em pesquisa feita a monografias e outros trabalhos correlatos a este assunto, foi encontrado em Feltrim (1999), a construção de um hiperdocumento que possibilite a fidelidade do conteúdo da documentação com relação ao produto de software sendo documentado. A meta deste trabalho foi obter a consistência entre as partes do hiperdocumento e os componentes do software com mais facilidade por meio de *links* definidos.

Outro trabalho é o desenvolvido pela empresa "Diversified Software" da Califórnia, que criou a ferramenta chamada "Docu/Text", um produto que gera documentação da operação e dos sistemas a partir de fontes existentes em seu Centro de Dados, possibilitando também adicionar informações textuais tal como instruções de reinício (DIVERSIFIED, 2003).

Segundo Pfleeger (2004), a documentação de um programa é um conjunto de descrições que explica a um leitor o que os programas fazem e como fazem. Pode ser dividida em duas partes, documentação interna que é o material descritivo escrito diretamente no código-fonte e documentação externa que pode ser considerada todo o restante.

A documentação interna também conhecida como comentário tem como finalidade, além de fornecer uma explicação linha por linha do que o programa está fazendo, dividir também o código-fonte em fases que representam as principais atividades e ainda apresentar informações que identifiquem o código-fonte. Geralmente as informações de identificação são colocadas no começo de cada arquivo de código-fonte, em um conjunto de comentários chamado de bloco de comentário do cabeçalho (PFLEEGER, 2004).

De acordo com Pfleeger (2004), Paula (2001), o bloco de comentário de cabeçalho é como uma introdução ao programa e deve conter as seguintes informações: nome do programa; autor juntamente com numero do telefone e e-mail; data em que o código-fonte foi

escrito e revisado; onde este código-fonte se ajusta no projeto geral do sistema; objetivo geral da existência deste código.

Conforme Paula (2001), todos os comentários constantes no código-fonte devem estar sempre atualizados, o programador que altera o código-fonte é também responsável por alterar os respectivos comentários e outras formas de documentação. “É essencial que os comentários reflitam o comportamento real do código.” (PFLEEGER, 2004). Os comentários devem acrescentar informações novas e não o óbvio.

Segundo Pfleeger (2004), uma das coisas mais difíceis para os leitores de programa é entender o modo como os dados estão estruturados e utilizados. É muito útil a utilização de um mapa de dados correspondente ao dicionário de dados para que se tenha um acompanhamento mais detalhado do fluxo e manipulação de dados efetuado a cada arquivo de código-fonte constante no sistema. Especialmente quando um sistema lida com muitos arquivos de tipos e propósitos variáveis, associados a *flags* e parâmetros fornecidos.

Este mapa de dados se enquadra no conceito de documentação externa, onde de certa maneira, pode-se dizer que o projeto é o esqueleto da documentação e a parte complementar é fornecida pela narrativa que discute as particularidades do código-fonte. Estas particularidades vão desde diagramas especificados ainda durante a análise do projeto até os controles de alteração e revisão de códigos-fonte (PFLEEGER, 2004).

3.2 PADRONIZAÇÃO DE CÓDIGOS FONTE

Outro aspecto fundamental para o desenvolvimento de software de qualidade é o fato de sua codificação estar bem desenhada e padronizada. De acordo com Dias (2003), a padronização da codificação de um programa serve para facilitar o entendimento e a revisão do código-fonte, escrito pelo mesmo programador ou por outra pessoa, mesmo depois de meses sem trabalhar com ele. Não importa qual é o padrão, contanto que haja um, e não precisa ser arbitrariamente definido, pois ele emerge naturalmente(WUESTEFELD, 2001).

Conforme Dias (2003), mais do que tornar o código “bonito”, a padronização e a formatação devem ressaltar a estrutura lógica do programa, dando pistas ao desenvolvedor a partir da visualização do código-fonte.

Em Dolla (2001), são apresentados alguns itens para padronização que podem ser aplicados na maioria das linguagens de programação bloco-estruturada: endentação; identificadores; comentários/documentação; tamanhos de programas e sub-rotinas. O trabalho realizado por Dolla (2001) consiste no desenvolvimento de uma ferramenta que analisa e faz a reestruturação interna de códigos-fonte em PL/SQL, utilizando padrões de legibilidade. Ele desenvolveu um protótipo que efetua a documentação dos programas e a formatação de alguns atributos do código-fonte como por exemplo a endentação, palavras-chave, colunas de tabelas e variáveis internas.

Dalmolin (2000) desenvolveu uma ferramenta que analisa e faz a reestruturação de código-fonte em C++, utilizando padrões de legibilidade que por consequência aumenta a qualidade dos códigos-fonte escritos em C++, através do padrão geral e o padrão de estilo.

A seguir serão feitas algumas considerações a respeito de padrão de nomenclaturas e declaração de variáveis.

3.2.1 PADRÃO DE NOMENCLATURAS

De acordo com Pfleeger (2004), nomenclaturas padronizadas podem ajudar a organizar os pensamentos e evitar equívocos. A escolha dos nomes para variáveis, componentes, tabelas e arquivos deve refletir sua utilização ou seu significado. Segundo Paula (2001), estudos indicam que a qualidade dos identificadores é mais importante que a qualidade dos comentários.

No entanto é essencial que se diferencie facilmente um identificador dentro do código-fonte e se saiba que este identificador é um componente de interface, uma variável, um parâmetro de uma sub-rotina interna ou ainda uma tabela definida no dicionário de dados.

Para isto é necessário que cada empresa utilize um padrão para nomenclatura de identificadores. A maneira mais fácil e comum para diferenciar variáveis de parâmetros, funções ou tabelas temporárias é o uso prefixos ou sufixos na nomenclatura dos mesmos.

Observe no exemplo a seguir a utilização de prefixos: “btCancelar” onde o prefixo “bt” indica que trata-se de um componente visual do tipo “*button*”; “tbEmpresa” onde o

prefixo “tb” indica que se trata de uma tabela do banco de dados; “prNomeEmpresa” onde o prefixo “pr” indica que se trata de um parâmetro pertencente a uma sub-rotina.

Paula (2001) sugere que para a diferenciação dos tipos de identificadores, sejam utilizados sufixos precedidos de um sublinhado “_”. Ex: “TipoEmpresa_i”, “NomeEmpresa_c”. Porém deve-se cuidar para que o sufixo não se torne a única diferença entre dois identificadores.

3.2.2 DECLARAÇÃO DE VARIÁVEIS

Segundo Paula (2001), cada variável deve estar declarada em uma linha diferente e toda declaração de variável deve ser acompanhada de um comentário que deve descrever as restrições e hipóteses relativas a essa variável. Este comentário deverá conter ainda o nome completo caso o nome da variável seja uma abreviação. Cada variável deve ser utilizada para uma única finalidade, deve-se evitar variáveis com propósitos diferentes.

Quanto ao escopo é muito importante que as referências a uma mesma variável estejam localizadas o mais próximo possível, “variáveis locais a uma rotina são preferíveis a variáveis locais a um módulo, que são preferíveis a variáveis globais”(PAULA, 2001)**.** É válido o uso de prefixos ou sufixos para diferenciar as variáreis quanto ao escopo.

De acordo com Paula (2001), variáveis globais tendem a introduzir sérios problemas como impedimentos a reutilização de código-fonte, impedimentos à reentrância ou quebra de moduralidade, em geral. Mas caso seja estritamente necessário a utilização, então deve-se tomar os seguintes cuidados: adotar um padrão para nomenclatura de variáveis globais, para que estas sejam facilmente identificadas; quando as variáveis forem globais para todo o sistema, procurar criar um módulo específico para declaração das mesmas.

“Todos os nomes devem descrever de forma completa e acurada a entidade que representam”(PAULA, 2001). Os nomes voltados ao problema, na maioria dos casos, são de melhor compreenssão do que os nomes voltados à implementação. Para nomes compostos, é muito útil o uso de caracteres como os sublinhados “_”, ou caracteres maiúsculos e minúsculos para separação dos nomes. Se a linguagem permitir, as abreviações devem ser evitadas para tornar os nomes o mais pronunciável possível.

Quanto ao tipo de dados assim como no caso do escopo, também pode-se usar prefixos e sufixos para a identificação de variáveis. Outra vez o sublinhado "_" aparece como uma boa sugestão para formatação do nome da variável, dividindo o nome propriamente dito com o padrão utilizado para identificação do tipo de variável.

4 DESENVOLVIMENTO DA FERRAMENTA

Com o objetivo de auxiliar as empresas, usuárias da linguagem de desenvolvimento *PROGRESS* 4GL, a automatizar a verificação dos padrões de nomenclatura e a gerar documentação técnica, foi desenvolvida uma ferramenta cujos requisitos, especificação e características operacionais serão apresentados neste capítulo.

4.1 REQUISITOS DO SOFTWARE

A ferramenta desenvolvida possui duas características funcionais básicas. Uma refere-se ao processo de auxílio à documentação com a criação de matrizes que identifiquem referências de tabelas em determinados códigos-fonte (mapa de dados conforme tópico 3.1). Outra refere-se ao processo de padronização de nomenclaturas ligadas ao código-fonte requerido (conforme tópico 3.2).

O software deve apresentar ao usuário uma alternativa já pré-definida para padronização de códigos-fonte para a linguagem de desenvolvimento de sistemas *PROGRESS 4GL*, porém deixando ao usuário a oportunidade de criar seu próprio modelo de padronização de codificação.

O processo de padronização resume-se na formatação da nomenclatura de variáveis, *procedures*, funções, parâmetros, arquivos de código-fonte, componentes e palavras reservadas da linguagem *PROGRESS 4GL*. Para as variáveis a formatação da nomenclatura pode ser diferenciada ainda quanto ao escopo e tipo de dado agregado aquela variável.

A varredura dos arquivos de códigos-fonte a serem analisados dar-se-á através da localização e extensão dos arquivos informados pelo usuário, que poderá optar em selecionar a quantidade de arquivos desejada, sendo um ou vários arquivos.

A formatação automática do código-fonte analisado fica a critério do usuário, que pode optar pela geração de um relatório apresentando todas as falhas encontradas a partir do padrão configurado e pode ainda optar pela correção automática destas falhas.

Quando para fins de documentação, a pesquisa será feita a cada um dos arquivos selecionados e deverá resultar em uma relação de todas as tabelas mencionadas, em cada arquivo pesquisado, e os tipos de transação ou acesso feitos para cada campo destas tabelas.

O usuário também poderá optar pelo inverso do citado no item anterior, escolhendo uma tabela do banco de dados, e a partir da tabela informada, o sistema irá percorrer todos os arquivos, conforme extensão e diretórios pré-definidos pelo usuário, em busca das rotinas que façam qualquer tipo de acesso aos dados desta tabela, relacionando assim as informações encontradas em um relatório claro e objetivo para o usuário.

4.2 ESPECIFICAÇÃO

A especificação da ferramenta foi realizada com a utilização da UML, "*Unified Modelling Language*", (MATOS, 2002). Sendo que o diagrama de classes e o diagrama de casos de uso foram desenvolvidos com auxílio da ferramenta CASE Rational Rose versão7.6.

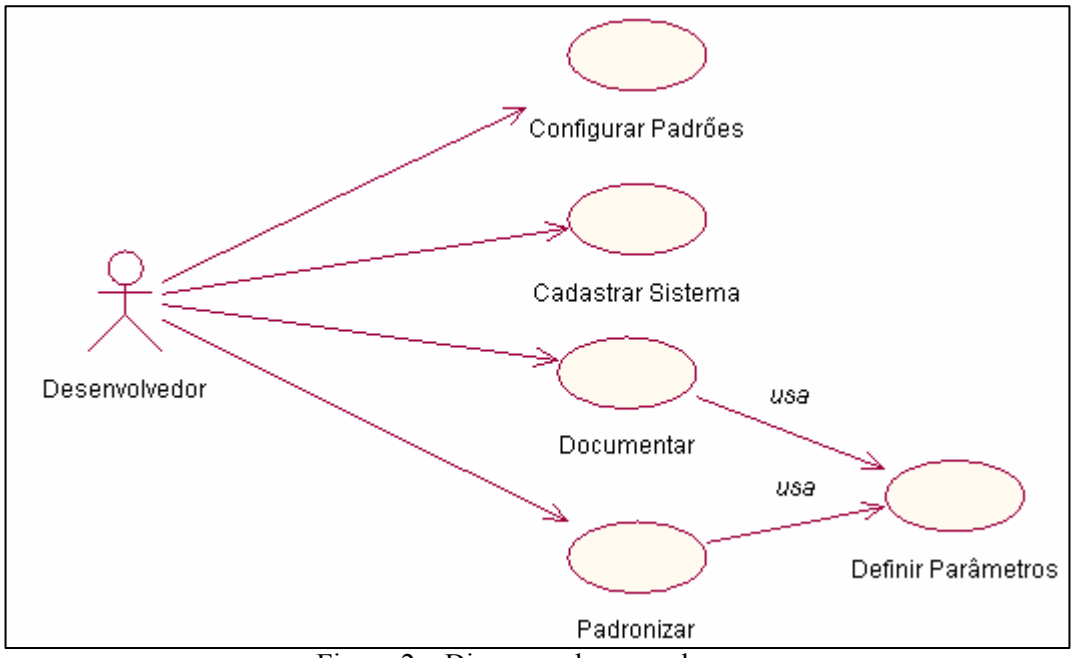


Figura 2 – Diagrama de casos de uso.

Cada caso de uso é descrito a seguir:

a) CONFIGURAR PADRÕES: consiste em configurar a formatação dos nomes de variáveis, funções, *procedures*, parâmetros, tabelas, campos das tabelas, palavras reservadas e arquivos de código-fonte. Quanto à nomenclatura das variáveis, ainda pode ser dividido quanto ao escopo e tipo de dados;

b) CADASTRAR SISTEMA: neste caso o usuário da ferramenta deverá cadastrar o sistema, código e nome, e cadastrar todos os banco de dados que são conectados pelo sistema;

c) DEFINIR PARÂMETROS: definir os diretórios onde se encontram os arquivos de código-fonte a serem analisados, definir os tipos de arquivos a serem analisados, escolher se o sistema deve varrer *includes*, que se encontram dentro dos arquivos selecionados e selecionar arquivos de código-fonte;

d) DOCUMENTAR: selecionar tabelas e definir tipo de documentação;

e) PADRONIZAR: selecionar configuração de padrões, definir se o sistema deve efetuar a correção automática de expressões fora do padrão e definir se deve ser gerado o relatório de inconsistências ou não.

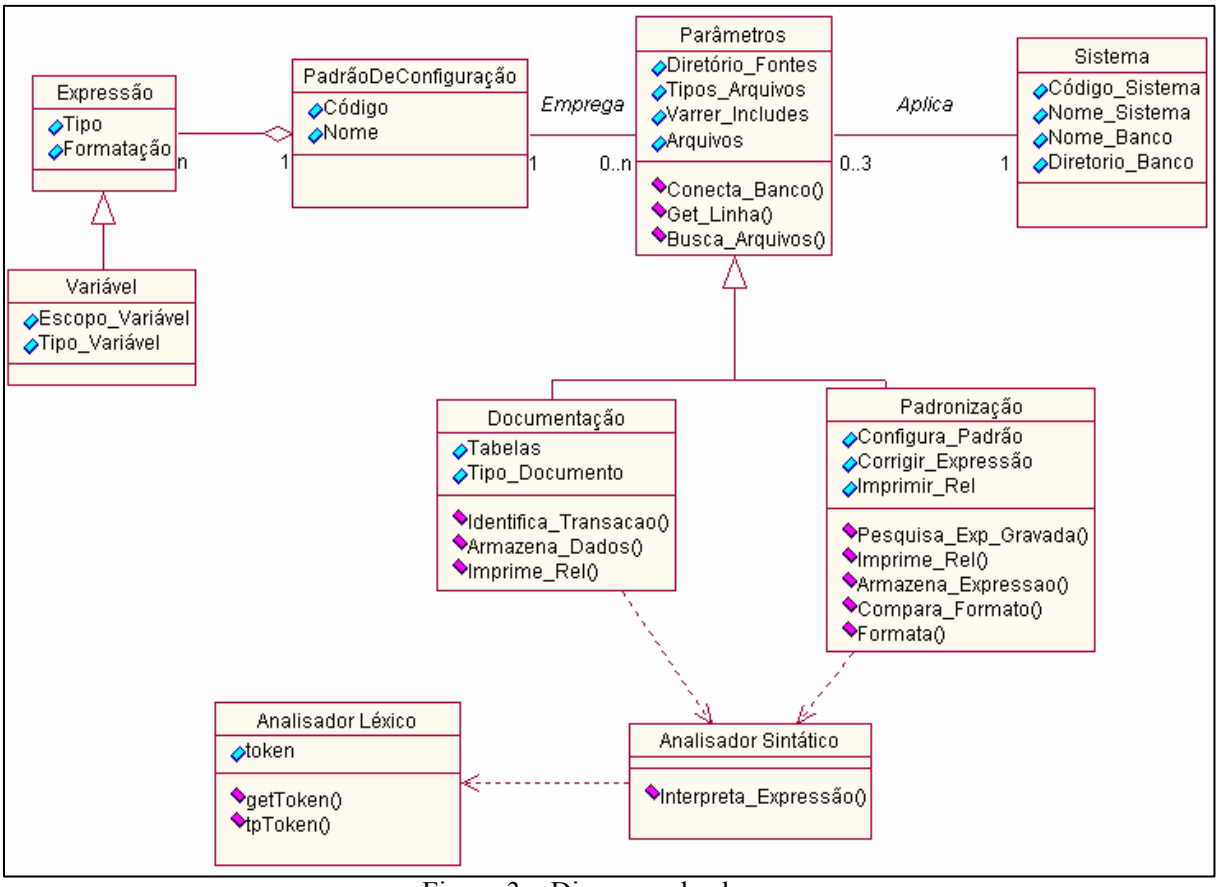


Figura 3 – Diagrama de classes.

A Figura 3 mostra o diagrama de classes especificado. A seguir será apresentada uma breve descrição sobre cada classe e suas funções:

a) Classe PadrãoDeConfiguração: utilizada para identificação dos padrões de configuração de nomenclatura de diversas expressões;

b) Classe Expressão: está agregada a classe PadrãoDeConfiguração e é utilizada para configuração da formatação da nomenclatura de diversos tipos de expressões;

c) Classe Variaveis: é uma subclasse da classe Expressão específica para variáveis, onde pode-se configurar o padrão para as variáveis quanto ao tipo de dado e escopo;

d) Classe Sistema: contém os atributos pertencentes a cada sistema cadastrado;

e) Classe Parâmetros: está associada às classes Sistema e PadrãoDeConfiguração. Contém os parâmetros definidos necessários ao processo de padronização e documentação. Possui as seguintes funções:

- Conecta_Banco: conectar aos banco de dados de acordo com o sistema definido;
- Busca_Arquivo: abre o próximo arquivo que estiver na lista de arquivos para análise;
- Get_Linha: faz a leitura de cada linha do arquivo em evidência;

f) Classe Documentação: contém atributos e funções específicos para o processo de documentação:

- Identifica_Transacao: ao encontrar um campo de tabela do banco de dados no código-fonte, faz a identificação do tipo de transação que está ocorrendo naquele trecho de código;
- Armazena_Dados: grava em uma tabela temporária os dados referentes a transação encontrada;
- Imprime_Rel: executa a impressão dos relatórios a partir das informações gravadas anteriormente;

g) Classe Padronização: contém atributos e funções específicos para o processo de padronização:

- Pesquisa_Exp_Gravada: de acordo com o tipo de expressão, *token* encontrado, verifica no banco de dados a formatação a ele configurada conforme PadrãoDeConfiguração;
- Compara_Formato: compara o formato da expressão encontrada com a formatação desejada conforme PadrãoDeConfiguração;
- Formata: faz a formatação da expressão encontrada conforme PadrãoDeConfiguração caso assim estiver definido no atributo Corrigir_Expressão;

- Armazena_Expressão: armazena o *token* no formato encontrado e no formato atualizado em uma tabela temporária;
- Imprime_Rel: imprime relatório de inconsistências se assim estiver definido através do atributo Imprimir_Rel;

h) Classe Analisador Sintático: faz a identificação de cada *token* encontrado;
   - Interpreta_Expressão: faz a identificação de cada *token* encontrado através das funções geradas a partir de uma BNF especificada (Quadro 12);

i) Classe Analisador Léxico: responsável pela extração do *token* do arquivo;
   - GetToken: faz a leitura de cada caractere da linha extraída do arquivo até obter a formação de um *token*, desprezando comentários e *strings*.

A geração do analisador sintático foi realizada seguindo a regra gramatical especificada em BNF (*Backus Naur Form*)(PRINCE, 2001). O Quadro 12 apresenta a BNF especificada.

Observando esta BNF, pode-se notar que não há um detalhamento completo da sintaxe da linguagem *PROGRESS*, já que esta possui uma sintaxe muito mais complexa. Mas na BNF citada, consta apenas a especificação necessária para que se possa colher os identificadores do código-fonte analisado e que se identifique as transações realizadas nas tabelas do banco de dados no código-fonte referenciados.

```
<COMANDO>        ::= <DEFINE>|<IF>|<EXP-LACO>|<PROCEDURE>|<FUNCTION>|
                     <EXP-ALTERA>|<EXP-INCLUI>|<EXP-EXCLUI>

<DEFINE>         ::= DEFINE|DEF <EXP-DEFINE>|<EXP-NEWSHAR>

<EXP-DEFINE>     ::=  <EXP-FUNCPARAM>|<EXP-BUTTON>|<EXP-IMAGE>|
                     <EXP-RECT>|<EXP-SUBMENU>.

<EXP-NEWSHAR>    ::= NEW|SHARED|GLOBAL|Λ  <EXP-VAR>|<EXP-WORKFILE>|
                     <EXP-TEMPTABLE>|<EXP-BROWSE>|<EXP-BUFFER>|<EXP-FRAME>|
                     <EXP-MENU>|<EXP-QUERY>|<EXP-STREAM>

<EXP-VAR>        ::= VAR|VARIABLE <identif> <ASLIKE> <resto>.
<ASLIKE>         ::= AS <TIPO>|LIKE <identif>

<TIPO>           ::= CHAR|CHARA|CHARAC|CHARACT|CHARACTE|CHARACTER|
                     INT|INTE|INTEG|INTEGE|INTEGER|DECI|DECIM|DECIMA|DECIMAL|
                     LOGI|LOGIC|LOGICA|LOGICAL|DATE|COM-HANDLE|HANDLE|MEMPTR|
                     RAW|RECID|ROWID|WIDGET-HANDLE

<EXP-FUNCPARAM>  ::= INPUT|OUTPUT|INPUT-OUTPUT|RETURN <PARAM> <EXP-TIPOPARAM>

<PARAM>           ::= PARAMETER|PARAM
<EXP-TIPOPARAM>  ::= <EXP-PARAM>|<EXP-BUFFER>|<EXP-TABLE>
<EXP-PARAM>       ::= <identif> <ASLIKE> <resto>.
<EXP-WORKFILE>   ::= WORK-TABLE|WORKFILE <identif> <resto>.
<EXP-TEMPTABLE>  ::= TEMP-TABLE <identif> <resto>.
<EXP-BUTTON>     ::= BUTTON <identif> <resto>.
<EXP-BROWSE>     ::= BROWSE <identif> <resto>.
<EXP-BUFFER>     ::= BUFFER <identif> <resto>.
<EXP-FRAME>      ::= FRAME <identif> <resto>.
<EXP-IMAGE>      ::= IMAGE <identif> <resto>.
<EXP-MENU>       ::= MENU <identif> <resto>.
<EXP-QUERY>      ::= QUERY <identif> <resto>.
<EXP-RECT>       ::= RECTANGLE <identif> <resto>.
<EXP-STREAM>     ::= STREAM <identif> <resto>.
<EXP-SUBMENU>    ::= SUB-MENU <identif> <resto>.

<PROCEDURE>      ::= PROCEDURE <identif> <resto> .|:
<FUNCTION>       ::= FUNCTION <identif> <resto> .|:

<IF>             ::= IF <CONDICAO> THEN
<CONDICAO>       ::= qualquer caracter <CONDICAO> <CAMPO>|Λ


<EXP-LACO>       ::= FOR|DO|REPEAT|CASE <CONDICAO> .|:

<EXP-ALTERA>     ::= UPDATE <identif> <id_1> <resto>.
<EXP-INCLUI>     ::= INSERT|CREATE <identif> <resto> .
<EXP-EXCLUI>     ::= DELETE <identif> <resto> .

<resto>          ::= qualquer caracter <resto> |Λ

<id_1>           ::= <identif> <id_1>|Λ
<identif>        ::= identificador necessário para análise
```

Quadro 12 – BNF

4.3 IMPLEMENTAÇÃO

Para o desenvolvimento desta ferramenta, além dos conceitos de análise orientada a objetos e de um estudo mais profundo da linguagem *PROGRESS* e suas particularidades, também foi necessário um estudo detalhado sobre técnicas utilizadas para a construção de compiladores, como utilização de tabelas de símbolos, gramáticas para geração de sentenças, analisadores léxico e sintático.

4.3.1 TÉCNICAS E FERRAMENTAS UTILIZADAS

Para a implementação foi utilizada a ferramenta de desenvolvimento *PROGRESS* 4GL versão 9.1D. A opção pela utilização desta aconteceu por ser uma ferramenta de alto nível de abstração, apresentar facilidade de se trabalhar com arquivos texto e pela capacidade de se trabalhar em diferentes sistemas operacionais utilizando-se do mesmo código-fonte.

A Figura 4 apresenta o diagrama de entidade e relacionamento (DER) modelo físico. Ele foi criado a partir da derivação do diagrama de classes (Figura 3). O DER Físico foi desenvolvido com o auxílio da ferramenta Power Designer versão 10.

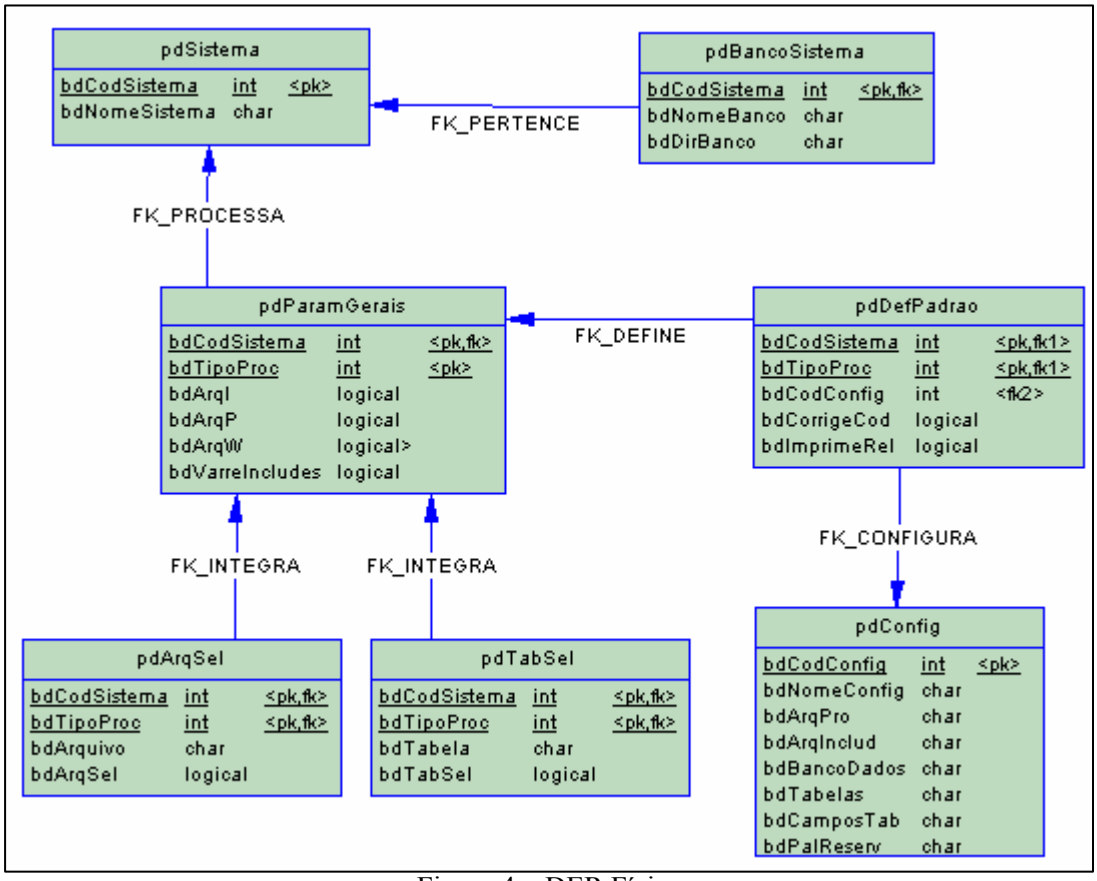


Figura 4 – DER Físico

A seguir será apresentada uma breve descrição a respeito de cada entidade constante no DER físico e explicação da derivação a partir do diagrama de classes:

a) pdSistema: entidade correspondente ao cadastro de sistemas (classe “Sistema”);
b) pdBancoSistema: entidade correspondente ao cadastro de banco de dados por sistema (criado a partir da normalização dos atributos “Nome-Banco” e “Diretório-Banco” da classe “Sistema”;

c) pdConfig: entidade correspondente ao cadastro de padrões de nomenclatura. De acordo com o diagrama de classes (Figura 3), existe uma classe chamada "PadrãoDeConfiguração". A esta classe está agregada a sub-classe "Expressão", que por sua vez está generalizada a uma terceira classe denominada "Variável". Como trata-se de um simples cadastro de um conjunto fixo de expressões e variáveis, foi possível agrupá-las em uma só tabela;
d) pdParamGerais: Contém todos os atributos em comum para os processos de padronização e documentação, cujo objetivo é guardar as definições do último processo executado para cada sistema. (Classe "Parâmetros");
e) pdDefPadrão: entidade que pode ser vista como uma extensão da entidade "pdParamGerais", porém com atributos pertencentes apenas ao processo de padronização. (Classe "Padronização");
f) pdArqSel: entidade que contém todos os atributos selecionados para pesquisa. Criada a partir da normalização do atributo "Arquivos" da classe "Parâmetros";
g) pdTabSel: entidade que contém todas as tabelas selecionadas para pesquisa, porém é utilizada somente para o processo de padronização. Criada a partir da normalização dos atributos "Tabelas" e "Tipo_Documento" da classe "Documentação".

O respectivo dicionário de dados é apresentado no ANEXO A, onde é possível visualizar todas as definições e estruturação do banco de dados da ferramenta.

A implementação do analisador léxico e do analisador sintático foi necessária para identificação das variáveis, parâmetros, funções, *procedures* e componentes, para formatação conforme padrão especificado e também para identificação das transações com o banco de dados quando a finalidade for a geração do relatório que se refere a função de documentação do sistema. Por não conterem atributos persistentes, mas apenas métodos, não se tornaram tabelas no modelo físico.

O analisador sintático, gerado a partir da BNF apresentada no Quadro 12, foi desenvolvido com base no algoritmo apresentado no Quadro 13.

PARA cada regra com a forma

```
<NT> ::=  X1,1   X1,2 . . . X1,n1  |
          X2,1   X2,2 . . . X2,n2  |
          ...
          Xm,1   Xm,2 . . . Xm,nm
```

Criar a função: "`Function` **`NT`**`: boolean; Begin`"
onde **NT** é o nome do não terminal do lado esquerdo da regra.

SE **NT** é o símbolo inicial da gramática, acrescentar à função: "`tpToken := getToken;`"

PARA cada alternativa (separadas por "|") do lado direito da regra FAÇA (i de 1 a m)

PARA cada elemento X_{ij} da alternativa FAÇA (j de 1 até ni)

SE X_{ij} é não terminal, acrescentar à função
"`if` **`Xij`** `then begin`"
SE X_{ij} é palavra reservada, acrescentar à função
"`if ((tpToken = id) and (`**`Xij`**` = token)) then begin`
`tpToken := getToken;`"
SE X_{ij} é símbolo especial, acrescentar à função
"`if ((tpToken = se) and (`**`Xij`**` = token)) then begin`
`tpToken := getToken;`"
SE X_{ij} é identificador, acrescentar à função
"`if (tpToken = id) then begin`
`tpToken := getToken;`"
FIM PARA

PARA cada elemento X_{ij} da alternativa FAÇA (j de ni até 1)

SE $X_{ij} <> \varepsilon$
SE j = ni, acrescentar à função "**`NT`**` := true; exit;`"
SE j = 1, acrescentar "`end;`"
SENÃO, acrescentar "`end else begin` **`NT`**` := false; exit; end;`"
FIM PARA
FIM PARA{ todas as alternativas }
SE $X_{m,1} = \varepsilon$, acrescentar "**`NT`**` := true;`"
SENÃO, acrescentar"**`NT`**` := false;`"
Acrescentar "`end;`"
FIM PARA { todas as regras }

Quadro 13 – Algoritmo para gerar um analisador sintático a partir de uma gramática especificada em BNF.

O Quadro 14 mostra um trecho de código-fonte gerado com base no algoritmo apresentado no Quadro 13. A função "fAsLike" apresentada no Quadro 14 é a codificação do não terminal "<ASLIKE>" especificado na BNF.

```
/* fTpToken = 0 -> Palavra Reservada

FUNCTION fAsLike RETURN LOGICAL:

  IF (fTpToken(lch_token) = 0) AND (lch_token = "AS") THEN
     DO:
        lch_token = fGetToken().
        RETURN fTipoVar().
     END.
  ELSE
  IF (fTpToken(lch_token) = 0) AND (lch_token = "LIKE") THEN
     DO:
        lch_token = fGetToken().
        RETURN fTipoCampo().
     END.
  ELSE
     RETURN FALSE.

END.
```

Quadro 14 – Exemplo de código-fonte gerado a partir da BNF especificada.

Para leitura dos arquivos de código-fonte, foi necessária a utilização de uma função do *PROGRESS* denominada "*QUOTER*". Esta função adiciona o caracter " ‟ " (aspas duplas), no início e no final de cada linha do arquivo-fonte no formato texto puro, tornando assim possível a leitura do arquivo inteiro onde cada linha será tratada como uma só *string*. Para o caso do processo de padronização, após a leitura cada linha será gravada em um novo arquivo que ao final do processo será o novo arquivo de código-fonte, agora reestruturado. Ao arquivo antigo, será atribuído a extensão ".bak", e o novo arquivo herdará o nome do anteriormente analisado.

Para a leitura do dicionário de dados, foi utilizada uma estrutura do *PROGRESS* denominada de "*METASCHEMA*". Esta estrutura possui todas as informações correspondentes ao banco de dados conectado, tornando possível assim a relação das tabelas e seus atributos necessários para o processo de padronização e documentação.

4.3.2 OPERACIONALIDADE DO SISTEMA

Conforme a Figura 5, a tela inicial do sistema apresenta as seguintes opções: Configurar Padrões; Cadastrar Sistema; Documentar; Padronizar; Sobre e Sair do sistema.

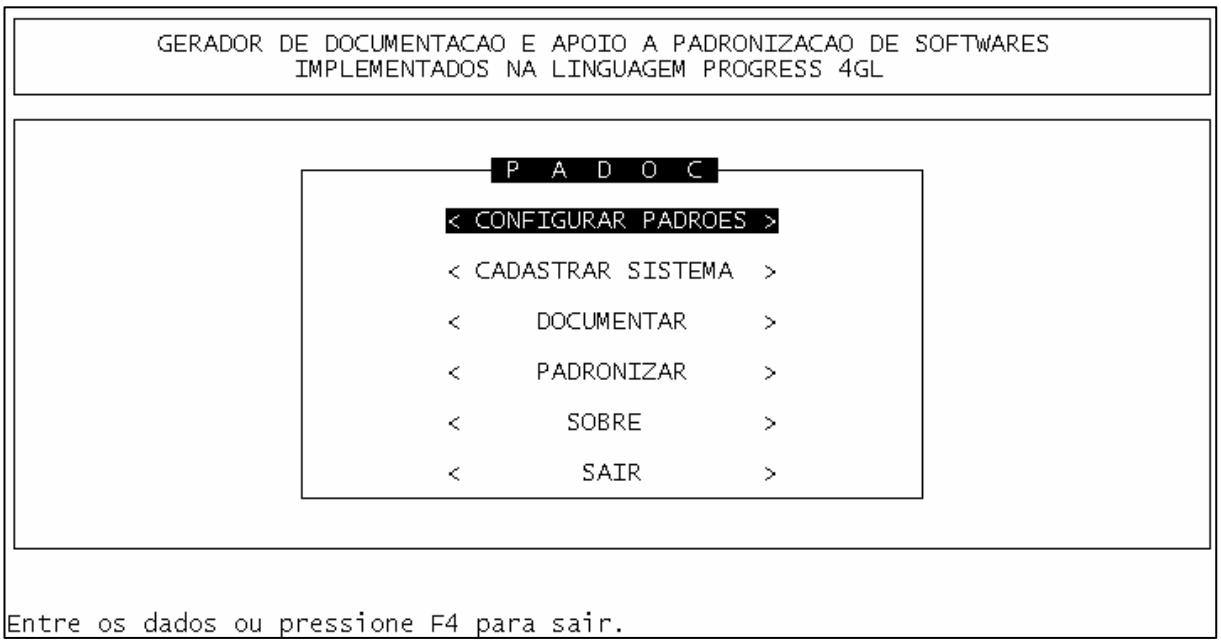


Figura 5 – Tela inicial da ferramenta.

Inicialmente o usuário deverá entrar na opção “Configurar Padrões”(Figura 6). Nesta tela são oferecidas as opções de consulta, inclusão, alteração e exclusão de padrões configurados, além da opção voltar para a tela inicial.

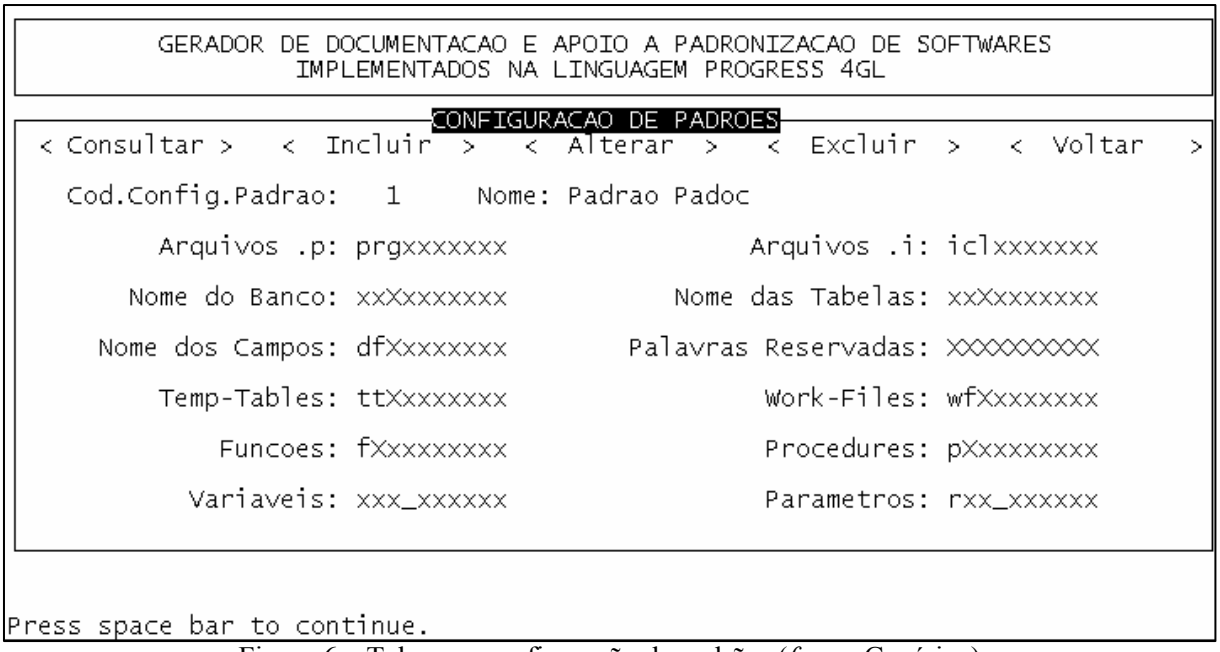


Figura 6 – Tela para configuração de padrões (*frame* Genérico)

A tela de configuração de padrões está dividida em quatro *frames*. O primeiro apresentado na Figura 6 disponibiliza os campos para configuração num âmbito mais genérico, arquivos, campos, tabelas, variáveis e etc.

Será dado como exemplo a formatação de uma variável, conforme configuração apresentada na Figura 6.

O campo referente as variáveis está com a seguinte configuração: “xxx_xxxxxx”. Isto significa que para este padrão especificado, toda variável encontrada no código-fonte a ser analisado deverá conter três caracteres minúsculos como prefixo, seguido de um sublinhado “_” e o restante do nome da variável sendo que todos os caracteres devem ser apresentados em letra minúscula.

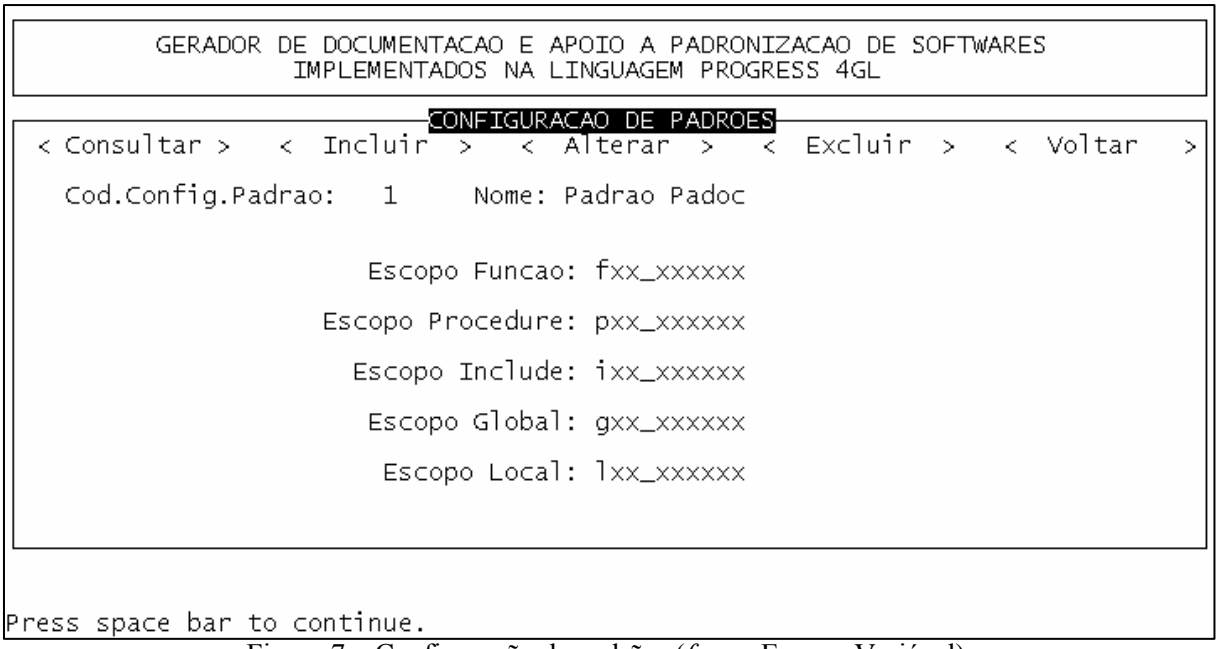


Figura 7 – Configuração de padrões (*frame* Escopo Variável).

O *frame* apresentado na Figura 7 serve exclusivamente para configurar a formatação do nome de variáveis quanto ao escopo. Seguindo esta linha e dando continuidade ao exemplo anterior, uma variável que estiver declarada dentro de uma função, conforme o padrão configurado na Figura 7, deverá estar formatada, com o primeiro caracter sendo a letra “f” em minúsculo e o restante conforme formatação já especificada na Figura 6.

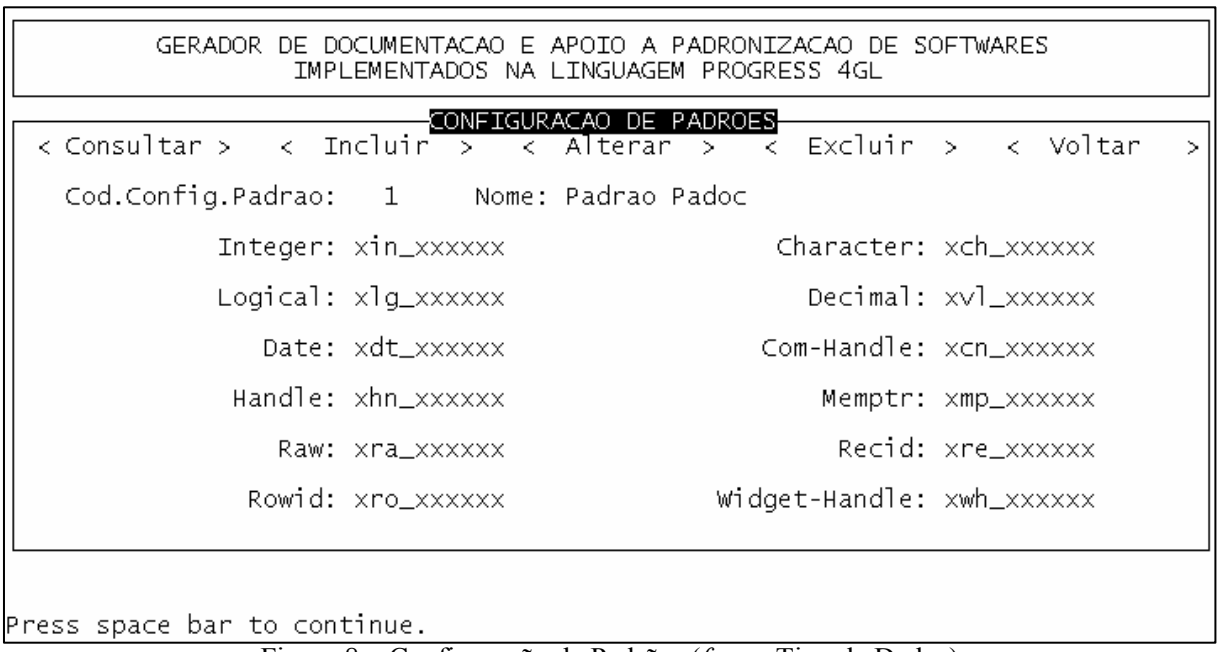


Figura 8 – Configuração de Padrões (*frame* Tipo de Dados)

Este terceiro *frame* demonstrado na Figura 8 já serve tanto para parâmetros quanto para variáveis e nele é possível se configurar a formatação destes identificadores quanto ao tipo de dados. No exemplo da variável, supondo que a ela foi definido o tipo de dado "*Character*", ou seja, é uma variável que armazena dados do tipo texto, esta variável vai ter a seguinte forma: "fch_xxxxx". Para complementar o exemplo, será atribuído a variável o nome de "cidade", portanto na formatação vista anteriormente ela ficaria como "fch_cidade". Portanto sem a necessidade de que se veja a declaração desta variável, apenas através do nome "fch_cidade", já se sabe que é uma variável do tipo *character* e que está declarada dentro de uma função.

A Figura 9 mostra o quarto e último *frame* da tela de configuração de parâmetros. Este *frame* traz os componentes que podem ser padronizados quanto à nomenclatura, *button*, *browse*, *buffer*, *frame,* etc.

Quando o usuário optar por deixar em branco o campo referente a qualquer item desta tela de configuração, o mesmo será ignorado no momento em que o sistema estiver efetuando a varredura para fins de padronização.

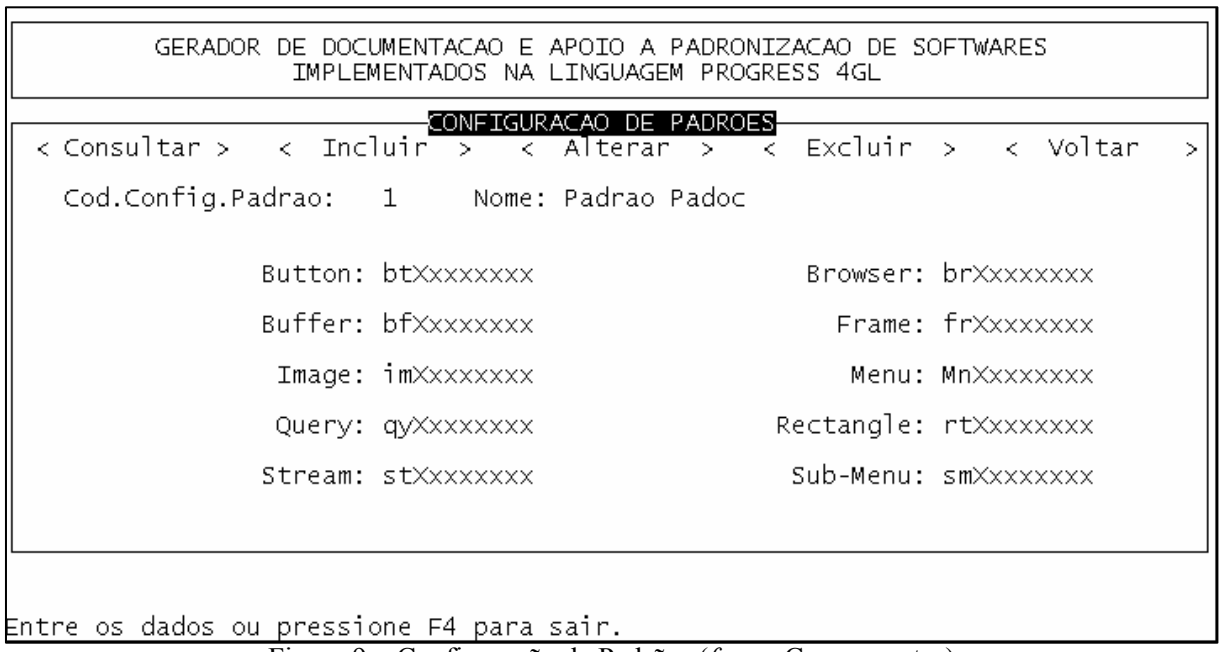


Figura 9 – Configuração de Padrões (*frame* Componentes)

Tendo configurado os padrões de nomenclatura o usuário deverá cadastrar o sistema referente aos arquivos de código-fonte que deseja analisar e indicar o nome e o diretório em qual se encontram os banco de dados que são conectados pelo seu sistema (Figura 10). Repare que esta tela também possui as opções se consulta, inclusão, exclusão e alteração.

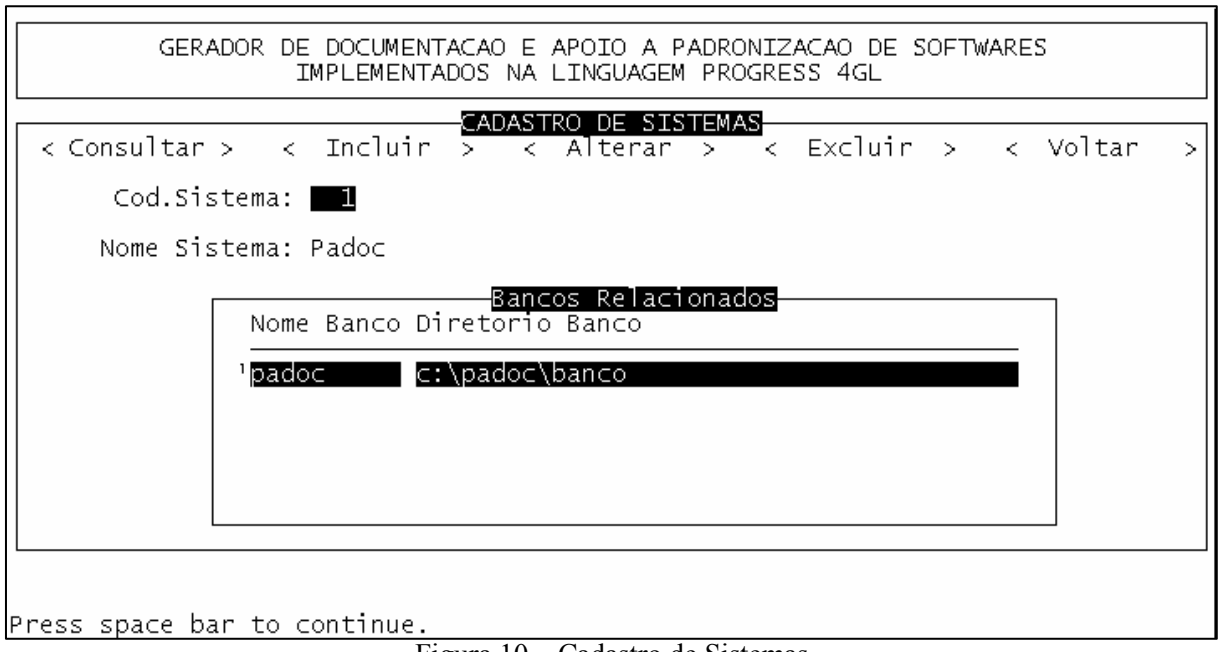


Figura 10 – Cadastro de Sistemas

Efetuados os cadastros a ferramenta está pronta para executar tanto a rotina de padronização como a rotina de documentação.

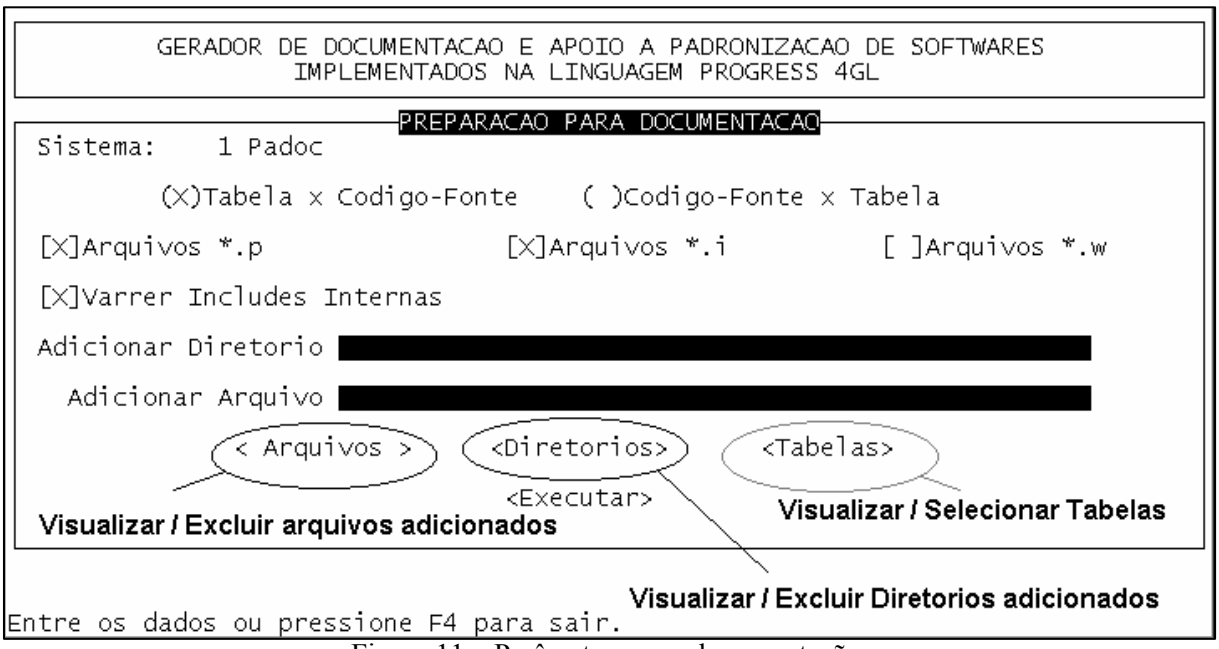


Figura 11 – Parâmetros para documentação.

A Figura 11 apresenta a tela para preparação da documentação. Observe o campo "Adicionar Diretórios", ao escrever um caminho de diretório neste campo e pressionar a tecla "ENTER", automaticamente todos os arquivos com as extensões definidas acima, que constam neste diretório, passarão a fazer parte da lista de arquivos. Do mesmo modo se o usuário acionar a opção para visualização de diretórios e excluir um diretório qualquer, todos os arquivos constantes na lista de arquivos que pertencem ao diretório excluído, também serão automaticamente eliminados.

A opção de seleção e visualização de tabelas, só estará disponível para a opção de documentação "Código-Fonte x Tabela", onde o usuário pode selecionar uma tabela e a ferramenta fará a busca por todos os arquivos códigos-fonte constante na lista de arquivos, apresentando ao final da busca um relatório com o resultado obtido.

Para dar início ao processo de documentação, basta entrar na opção executar, "<Executar>"(Figura 11). Neste momento todos os parâmetros configurados antes da execução serão gravados no bando de dados, e na próxima vez em que o usuário entrar nesta tela ao escolher o sistema e o tipo de documentação, automaticamente as últimas configurações serão restauradas.

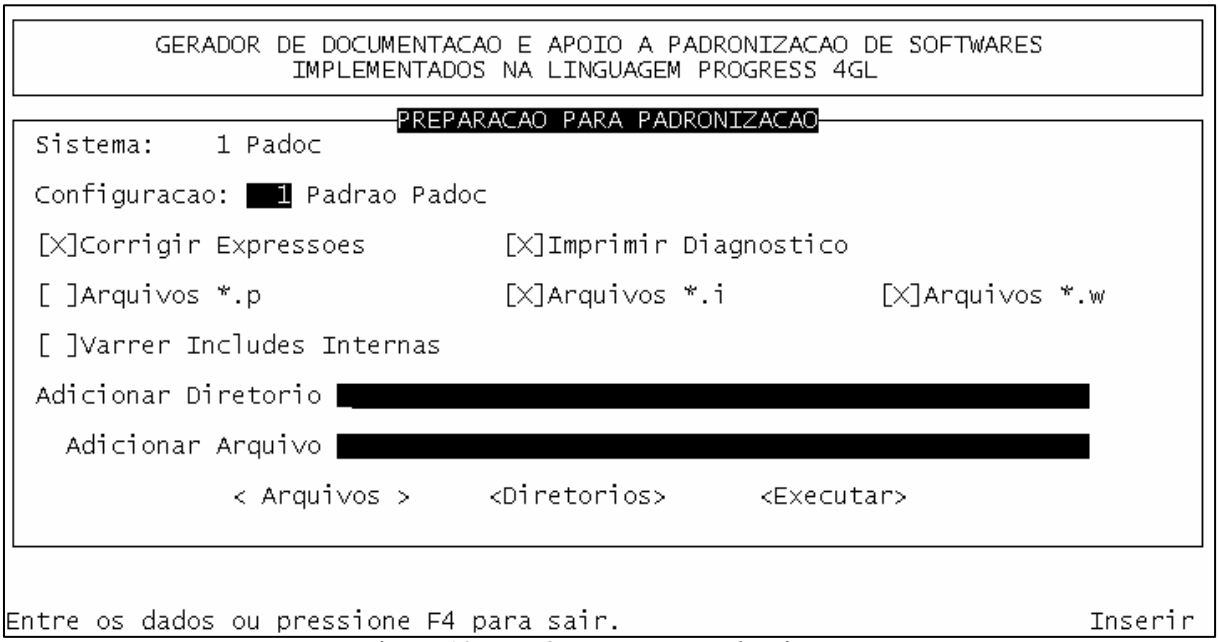


Figura 12 – Parâmetros para padronização

A Figura 12 apresenta a tela de parâmetros para padronização. Todos os atributos desta tela possuem as mesmas características que os atributos em comum com a tela de parâmetros para documentação, mencionada na Figura 11.

Outra característica comum, desta vez para todas as telas, é a consulta através de um *browser* de determinados cadastros. Utilizando-se como exemplo a Figura 12, caso o usuário esteja com o foco no campo “Configuração”, ele tem a possibilidade de pressionar a tecla “F9” e visualizar todos os padrões de configuração cadastrados, conforme Figura 13.

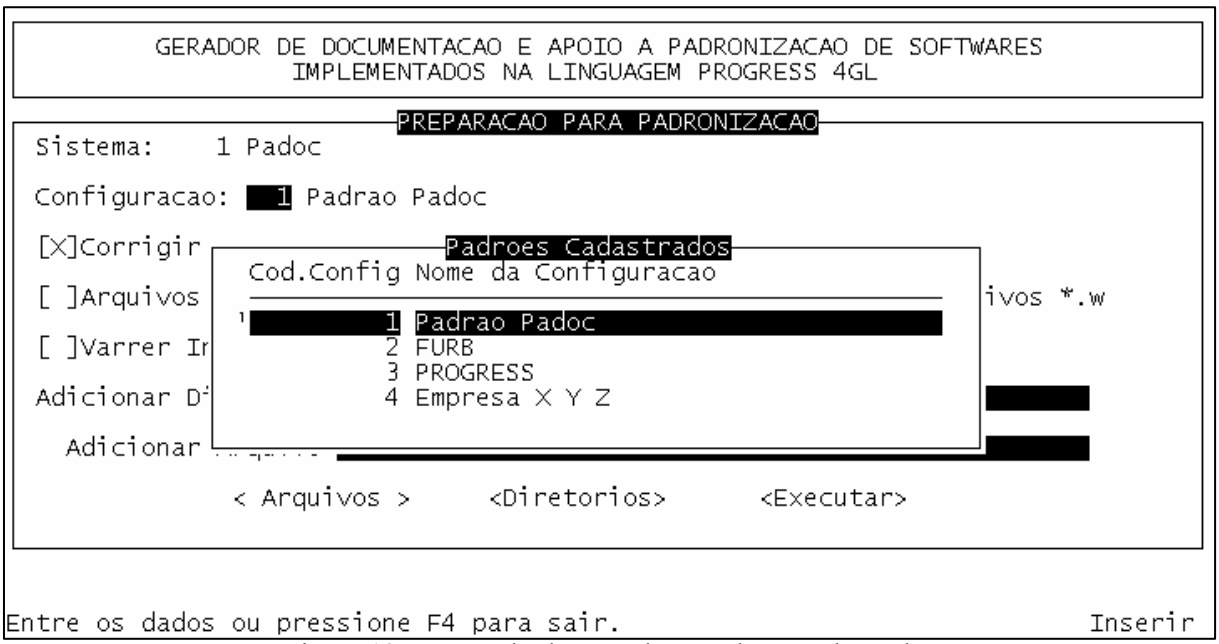


Figura 13 – Exemplo de consulta a padrões cadastrados

4.4 RESULTADOS E DISCUSSÃO

A seguir será apresentado, através de um estudo de caso, o resultado da solicitação de um processo de padronização e outro de documentação a partir de um mesmo arquivo de código-fonte.

4.4.1 PADRONIZAÇÃO

Para o processo de padronização, será utilizado como modelo de configuração de padrões o apresentado nas Figura 6, Figura 7, Figura 8 e Figura 9.

A Figura 14 apresenta o código-fonte a ser analisado “prgListaSistema.p”. Neste código-fonte, existe uma referência a um arquivo *include* “iclFormataNomeSis.i”. Este arquivo pode ser observado na Figura 15.

```
/*****************************************************************************
Programa: prgListaSistema
Autor: Julio 999-9999 julio@julio.com.br
Data: 13/06/2004
Objetivo: Listar os sitemas cadastrados no banco de dados
*****************************************************************************/

DEFINE VARIABLE conta_sistema   AS INTEGER  INITIAL 0              NO-UNDO.

DEFINE VARIABLE mensagem        AS CHARACTER                       NO-UNDO.

PROCEDURE mensagem:
    DEF INPUT PARAM total_sistema AS INTEGER                       NO-UNDO.

    DEF VAR         proximo_codigo LIKE pdsistema.bdcodsistema     NO-UNDO.

    mensagem = "O total de sistemas cadastrados é " + STRING(total_sistema).

    FIND LAST pdsistema NO-LOCK NO-ERROR.
    IF AVAILABLE pdsistema THEN
       proximo_codigo = pdsistema.bdcodsistema + 1.
    ELSE
       proximo_codigo = 1.

    MESSAGE mensagem VIEW-AS ALERT-BOX.

    MESSAGE "O próximo código a ser utilizado deve ser '"
            proximo_codigo "'.".

END.

FOR EACH pdsistema NO-LOCK:
    conta_sistema = conta_sistema + 1.
    DISPLAY pdsistema.bdcodsistema pdsistema.bdnomesistema.
END.

RUN mensagem(conta_sistema).

{ iclFormataNomeSis.i }

FOR EACH pdBancoSistema EXCLUSIVE-LOCK:
    FIND pdSistema OF pdBancoSistema NO-LOCK NO-ERROR.

    IF NOT AVAILABLE pdSistema THEN
       DELETE pdBancoSistema.

END.
/*****************************************************************************/
```

Figura 14 – Código-fonte “prgListaSistema.p” a ser reestruturado e documentado

```
/*****************************************************************************
Programa: iclFormataSis
Autor: Julio 999-9999 julio@julio.com.br
Data: 13/06/2004
Objetivo: Incluir o código do sistema juntamente com um hifem antes do nome do
          sistema.
*****************************************************************************/

DEF VAR nome_sistema    LIKE pdsistema.bdnomesistema             NO-UNDO.

FOR EACH pdsistema EXCLUSIVE-LOCK:
    nome_sistema = STRING(pdsistema.bdcodsistema) + "-" +
                   pdsistema.bdnomesistema.

    ASSIGN pdsistema.bdnomesistema = nome_sistema.

END.

/*****************************************************************************/
```

Figura 15 – “iFormataNomeSis.i” arquivo *include* referenciado no código-fonte da Figura 13.

Após configurar as opções para o processo de padronização (Figura 12), sendo que as opções “Corrigir Padrões”, “Imprimir Diagnóstico” e “Varrer Includes Internas” estão ativas, a Figura 16 mostra o resultado do processo de reestruturação do código-fonte apresentado na Figura 14.

```
/*****************************************************************************
Programa: prgListaSistema.
Autor: Julio 999-9999 julio@julio.com.br
Data: 13/06/2004
Objetivo: Listar os sitemas cadastrados no banco de dados
*****************************************************************************/

DEFINE VARIABLE lin_conta_sistema   AS INTEGER  INITIAL 0               NO-UNDO.

DEFINE VARIABLE lch_mensagem        AS CHARACTER                        NO-UNDO.

PROCEDURE pMensagem:
    DEF INPUT PARAM rin_total_sistema AS INTEGER                        NO-UNDO.

    DEF VAR         pin_proximo_codigo LIKE pdSistema.bdCodsistema      NO-UNDO.

    lch_mensagem = "O total de sistemas cadastrados é " + STRING(rin_total_sistema).

    FIND LAST pdSistema NO-LOCK NO-ERROR.
    IF AVAILABLE pdSistema THEN
       pin_proximo_codigo = pdSistema.bdCodsistema + 1.
    ELSE
       pin_proximo_codigo = 1.

    MESSAGE lch_Mensagem VIEW-AS ALERT-BOX.

    MESSAGE "O próximo código a ser utilizado deve ser '"
            pin_proximo_codigo "'.".

END.

FOR EACH pdSistema NO-LOCK:
    lin_conta_sistema = lin_conta_sistema + 1.
    DISPLAY pdSistema.bdCodSistema pdSistema.bdNomesistema.
END.

RUN pMensagem(lin_conta_sistema).

{ iclFormataNomeSis.i }

FOR EACH pdBancoSistema EXCLUSIVE-LOCK:
    FIND pdSistema OF pdBancoSistema NO-LOCK NO-ERROR.

    IF NOT AVAILABLE pdSistema THEN
       DELETE pdBancoSistema.

END.

/*****************************************************************************/
```

Figura 16 - Código-fonte “prgListaSistema.p” após processo de reestruturação

A Figura 16 torna possível a comparação entre o código-fonte antigo (Figura 14) e o código-fonte reestruturado. Por exemplo, a variável antes denominada “conta_sistema”, passou a se chamar “lin_conta_sistema”. Por ser uma variável do tipo “*integer*” e com escopo local para esta rotina, foi acrescentado em sua nomenclatura o prefixo “lin_”, conforme padrão configurado na própria ferramenta.

```
/*****************************************************************************
Programa para teste de Padronização e Ducumentação.
Autor: Julio 999-9999 julio@julio.com.br
Data: 13/06/2004
Objetivo: Incluir o código do sistema juntamente com um hifem antes do nome do
          sistema.
*****************************************************************************/

DEF VAR ich_nome_sistema    LIKE pdSistema.bdNomesistema              NO-UNDO.

FOR EACH pdsistema EXCLUSIVE-LOCK:
    ich_nome_sistema = STRING(pdSistema.bdCodsistema) + "-" +
                   pdSistema.bdNomesistema.

    ASSIGN pdSistema.bdNomeSistema = ich_nome_sistema.

END.

/*****************************************************************************/
```

Figura 17 – Arquivo Include “iclFormataNomeSis.i” após processo de reestruturação

Além da reestruturação efetuada e apresentada na Figura 16 e na Figura 17, conforme a opção “Imprimir Diagnóstico”, o processo efetuado resultou no relatório apresentado na Figura 18.

```
RELATORIO DE INCONSISTENCIAS ENCONTRADAS NO CODIGO-FONTE prgListaSistemas.p.

Sistema : 1 - Padoc
Data da Verificacao : 13/06/2004.

-> VARIAVEIS:

Nome                 Escopo         Tipo           Formatacao Correta
-------------------- ------------   ------------- ------------------------
conta_sistema        local          integer        lin_nome_sistema
mensagem             local          character      lch_mensagem
proximo_codigo       procedure      integer        pin_proximo_codigo
nome_sistema         include        character      ich_nome_sistema


-> PARAMETROS:

Nome                 Tipo           Formatacao Correta
-------------------- ------------   --------------------------
total_sistema        integer        rin_nome_sistema


-> PROCEDURE:

Nome                 Formatacao Correta
-------------------- --------------------------
mensagem             pMensagem


-> TABELAS:

Nome                 Formatacao Correta
-------------------- --------------------------
pdsistema            pdSistema


-> CAMPOS:

Nome                 Formatacao Correta
-------------------- --------------------------
bdcodsistema         bdCodsistema
bdnomesistema        bdNomesistema
```

Figura 18 – Relatório de Inconsistências

O relatório apresentado na Figura 18, mostra todas as expressões fora do padrão encontradas nos códigos-fonte das Figuras 14 e 15. Para o caso das variáveis é apresentado o nome encontrado, o escopo da variável, o tipo de dado a ela atribuído e a formatação correta. Para o caso dos parâmetros além do nome encontrado e a formatação correta, é apresentado o tipo de dado a ele atribuído. E para os demais tipos de expressões é apresentado apenas o nome encontrado e como seria a formatação correta. Sempre tendo como base o padrão de nomenclatura definido no cadastro de configuração padrão.

4.4.2 DOCUMENTAÇÃO

Aproveitando os mesmos códigos-fonte utilizados para o processo de padronização, será exemplificado o processo de documentação.

Dado que os parâmetros mais relevantes para o exemplo, pertencentes à tela de preparação para a documentação, estão configurados da seguinte forma: opção “Tabela x Codigo-Fonte” selecionada, “Varrer Includes Internas” ativada e adicionado o arquivo “prgListaSistemas.p”. Ao acionar o botão “<Executar>”, a ferramenta gerou o relatório apresentado na Figura 19.

```
|------------------------------------------------------------------------------ |
| TABELAS REFERENCIADAS NO PROGRAMA prgListaSistemas.p                          |
|------------------------------------------------------------------------------ |
|TABELA         | CONSULTA           |INCLUSAO|  EXCLUSAO  |ALTERACAO             |
|------------------------------------------------------------------------------ |
|pdSistema      |bdCodSistema        |        |            |bdNomeSistema         |
|               |bdNomeSistema       |        |            |                      |
|============================================================================== |
|pdBancoSistema |                    |        |     X      |                      |
|------------------------------------------------------------------------------ |
```

Figura 19 – Relatório gerado a partir do processo de documentação solicitado

A Figura 19, mostra que ao código-fonte “prgListaSistemas.p” estão sendo referenciadas duas tabelas do banco de dados, sendo elas “pdSistema” e “pdBancoSistema”. Para a tabela “pdSistema” estão sendo efetuadas as seguintes transações: consulta através dos campos “bdCodSistema” e “bdNomeSistema”; alteração através do campo “bdNomeSistema”. Quanto a tabela “pdBancoSistema”, o caracter “X” na coluna “EXCLUSÃO” indica que está sendo feita a exclusão de determinado registro da tabela.

5 CONCLUSÕES

É muito importante que cada empresa tenha um padrão para a codificação de sistemas. Pois com um código-fonte padronizado fica muito mais compreensível, para uma equipe de desenvolvimento que já esteja habituada a este padrão, efetuar atividades de manutenção ou até mesmo consulta.

A ferramenta desenvolvida atende bem a esta questão por oferecer aos desenvolvedores a opção de configurar seu próprio padrão. Ainda é possível escolher fazer a reestruturação do código-fonte ou apenas alertar através da geração de um relatório, sobre as inconsistências encontradas no código-fonte especificado.

A documentação atualizada dos códigos-fonte é outra questão de muita valia para as empresas, pois além de auxiliar a evitar futuros erros decorridos de uma manutenção mal analisada, ainda pode ajudar futuros desenvolvedores a entender melhor o funcionamento do sistema em que irão trabalhar.

Neste caso a ferramenta que foi desenvolvida para este trabalho, auxilia parte deste processo por oferecer a geração de um tipo de documento muito útil, principalmente no processo de manutenção. Pois com a opção de se escolher uma tabela do banco de dados ou um código-fonte a que se deseja efetuar algum tipo de alteração, a ferramenta irá gerar um mapa de dados apresentando todas as transações efetuadas por determinados códigos-fonte em determinadas tabelas.

Desta forma os objetivos do trabalho foram alcançados, visto que ambos os processos estão satisfazendo os requisitos especificados, que seria a reestruturação do código-fonte conforme padrão configurado na própria ferramenta e a geração de documentos que são a relação de códigos-fonte por tabela ou tabelas por código-fonte. Além destes itens a ferramenta ainda dispõe da capacidade de operar em diferentes sistemas operacionais como *UNIX* e *WINDOWS*.

Em uma visão geral, pode-se dizer que é uma ferramenta flexível. Visto que além de poder cadastrar diversas configurações de padronização, é possível também cadastrar vários sistemas e relacionar para estes todos os bancos de dados que são conectados e ainda guardar as definições dos últimos processos executados para cada sistema.

A especificação da BNF e a utilização do algoritmo para a geração do código-fonte a partir da BNF especificada, facilitaram muito a implementação, visto que poucos erros foram encontrados no analisador sintático durante a fase de testes.

5.1 LIMITAÇÕES E SUGESTÕES

A ferramenta desenvolvida limita-se apenas a analisar códigos-fonte escritos em *PROGRESS* para versão caracter. Uma sugestão é desenvolver uma ferramenta com o mesmo propósito utilizando *WEB SPEED* (PROGRESS, 2001), que é uma ferramenta para desenvolvimento *WEB* da *PROGRESS*.

Além de configurar padrões de nomenclatura, é possível ainda implementar a padronização com relação a endentação do código-fonte, verificação de variáveis declaradas e não utilizadas, comentários internos para sub-rotinas e arquivos de código-fonte e a partir deste último, elaborar a geração de um relatório que apresente para cada código-fonte, todas as funções, *procedures* e arquivos acessados, utilizando-se inclusive dos comentários encontrados em cada sub-rotina.

Outra opção para extensão deste trabalho, é generalizar para qualquer linguagem através da entrada das primitivas de cada linguagem.

REFERÊNCIAS BIBLIOGRÁFICAS

BARRETO Júnior, José. **Qualidade de software.** Rio de Janeiro, [1999]. Disponível em: <http://www.utp.br/informacao>. Acesso em: 13 jun. 2004.

COSTA, Márcio Brener. **Dominando o progress.** [s.l.], [2000]. Disponível em: <http://www.geocities.com/marcio-brener>. Acesso em: 10 maio 2004.

DALMOLIN, Denis Alberto. **Ferramenta de apoio a reestruturação de código fonte em linguagem C++ baseado em padrões de legibilidade**. 2000 69f. Monografia (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Fundação Universidade Regional de Blumenau, Blumenau.

DIAS, André Felipe. **Estilo de codificação – Delphi.** [s.l.]. Disponível em: <http://attach1.groups.yahoo.com>. Acesso em: 12 out. 2003.

DIVERSIFIED SOFTWARE SYSTEMS. **DOCU/TEXT**: para sistemas do legado MVS. [s.l.], 2003. Disponível em: <http://www.diversifiedsoftware.com.br>. Acesso em: 13 jun. 2004.

DOLLA, Dyckson Dyorgio. **Ferramenta de apoio a reestruturação de código fonte em linguagem PL/SQL baseado em padrões de legibilidade**. 2001 61f. Monografia (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Fundação Universidade Regional de Blumenau, Blumenau.

DUARTE, Alexandre da Silva. **Software de apoio ao processo de documentação baseado em normas de qualidade.** 2000. Monografia (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Fundação Universidade Regional de Blumenau, Blumenau.

FELTRIM, Valéria Delisandra. **Apoio à documentação de engenharia reversa de software por meio de hipertextos.** 1999. 120f. Dissertação (Mestrado em Ciências – Área de Ciências da Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos.

FERNANDES, Aguinaldo Aragon. **Gerência de software através de métricas.** São Paulo: Atlas, 1995.

FURLAN, José Davi. **Reengenharia da informação**. São Paulo: Makron Books, 1994.

GONÇALVES, Luiz Marcos Garcia. **Algoritmo e lógica de programação**: conceitos de linguagens de progamação. Natal, [2004]. Disponível em: <http://www.dca.ufrn.br/~lmarcos/courses/DCA800/notas_de_aulas.html>. Acesso em: 20 maio 2004.

MATOS, Alexandre Veloso de. **Unified modeling language**: prático e descomplicado. São Paulo: Érica, 2002.

NOVAES, Paulo Roberto da Rocha. [s.l.]. Disponível em: <http://www.as400.hpg.ig.com.br/_progress.html>. Acesso em: 02 maio 2004.

PAULA, Wilson de Pádua Filho. **Engenharia de software**: fundamentos, métodos e padrões. Rio de Janeiro: LTC, 2001.

PFLEEGER, Shari Lawrence. **Engenharia de software**: teoria e prática. São Paulo: Prentice Hall, 2004.

PRINCE, Ana Maria de Alencar; TOSCANI, Simão Sirineu. **Implementação de linguagens de programação**: compiladores. Porto Alegre: Sagra Luzzatto, 2001.

PROGRESS SOFTWARE CORPORATION, São Paulo, [2001]. Disponível em: <http://www.progress-software.com.br>. Acesso em: 13 jun. 2004.

PROGRESS SOFTWARE CORPORATION. **Programming hand book**. USA, 1997.

PROGRESS SOFTWARE CORPORATION. **Progress language tutorial**. USA, 2002.

RIBEIRO, Vinícios G. **Técnicas alternativas para o desenvolvimento de sistemas de informação**.Porto Alegre, [2004]. Disponível em: <http://www.sinpro-rs.org.br/paginasPessoais/layout1/index.asp?id=163>. Acessado em: 13 jun. 2004.

ROCHA, Ana Regina Cavalcanti da. **Qualidade de software:** teoria e prática. São Paulo: Prentice Hall, 2001.

STAA, Arndt Von. **Programação modular**: desenvolvendo programas complexos de forma organizada e segura. Rio de Janeiro: Campus, 2000.

WUESTEFELD, Klaus.[s.l], [2001]. Dispónível em: <http://www.xispe.com.br>. Acesso em: 12 out. 2003.

ANEXO A – Dicionário de Dados da Ferramenta Desenvolvida

```
14/06/04 01:42:48       PROGRESS Report
Database: c:\padoc\banco\padoc (PROGRESS)


Table Name     Description
-------------------------------------------------------------------------------
pdArqSel       Tabela de Arquivos Selecionados
pdBancoSistema Tabela de Banco de Dados por Sistema
pdConfig       Tabela para configuracao de padroes
pdDefPadrao    Tabela que contem as definicoes para padronizacao
pdParamGerais  Tabela de parametros para execucao de processos
pdSistema      Tabela de Sistemas
pdTabSel       Tabelas Selecionadas

================================================================================
============================== Table: pdArqSel ===============================

              Table Flags: "f" = frozen, "s" = a SQL table


Table                          Dump     Table Field Index Table
Name                           Name     Flags Count Count Label
------------------------------ -------- ----- ----- ----- --------------------
pdArqSel                       pdarqsel           5     1 Arquivos Selecionad

  Description: Tabela de Arquivos Selecionados
 Storage Area: N/A
============================== FIELD SUMMARY ==============================
============================== Table: pdArqSel ===============================

Flags: <c>ase sensitive, <i>ndex component, <m>andatory, <v>iew component

Order Field Name                 Data Type   Flags Format          Initial
----- -------------------------- ----------- ----- --------------- ---------
   10 bdCodSistema               inte        im    zz9             0
   30 bdArquivo                  char        c     x(80)
   40 bdSelArq                   logi              yes/no          no
   60 bdTipoProc                 inte        im    9               0

Field Name                     Label                  Column Label
------------------------------ ---------------------- --------------------
bdCodSistema                   Codigo do Sistema      Cod.Sistema
bdArquivo                      Arquivo                Arquivo
bdSelArq                       Arquivo Selecionado    Arq.Sel.
bdTipoProc                     Tipo de Processo       Tipo de Processo


============================== INDEX SUMMARY ==============================
============================== Table: pdArqSel ===============================

Flags: <p>rimary, <u>nique, <w>ord, <a>bbreviated, <i>nactive, + asc, - desc

Flags Index Name                       Cnt Field Name
----- -------------------------------- --- --------------------------------
p     ixArqSel                           2 + bdCodSistema
                                           + bdTipoProc

** Index Name: ixArqSel
 Storage Area: N/A

============================== FIELD DETAILS ==============================
============================== Table: pdArqSel ===============================

** Field Name: bdCodSistema
  Description: Contem o codigo do sistema
         Help: Informe o Codigo do Sistema

** Field Name: bdArquivo
  Description: Contem o endereco e nome do arquivo
         Help: Informe o endereco e nome do arquivo

** Field Name: bdSelArq
  Description: Indica se o arquivo esta selecionado ou nao
```

```
        Help: Indica se o arquivo esta selecionado ou nao

** Field Name: bdTipoProc
  Description: Contem o Tipo de Processo (1)Tab x Cod (2)Cod x Tab
               (3)Padronizar
        Help: Informe o Tipo de Processo (1)Tab x Cod (2)Cod x Tab
              (3)Padrao


================================================================================
============================== Table: pdBancoSistema ===========================

              Table Flags: "f" = frozen, "s" = a SQL table


Table                          Dump     Table Field Index Table
Name                           Name     Flags Count Count Label
------------------------------ -------- ----- ----- ----- --------------------
pdBancoSistema                 pdbancos           3     1 Banco de Dados por

  Description: Tabela de Banco de Dados por Sistema
 Storage Area: N/A


============================== FIELD SUMMARY ===============================
============================== Table: pdBancoSistema ===========================

Flags: <c>ase sensitive, <i>ndex component, <m>andatory, <v>iew component

Order Field Name                 Data Type   Flags Format          Initial
----- -------------------------- ----------- ----- --------------- ----------
   10 bdCodSistema               inte        im    zz9             0
   20 bdNomeBanco                char        cim   x(20)
   30 bdDirBanco                 char        cm    x(80)

Field Name                     Label                  Column Label
------------------------------ ---------------------- ----------------------
bdCodSistema                   Codigo do Sistema      Cod.Sistema
bdNomeBanco                    Nome do Banco de Dados Nome Banco
bdDirBanco                     Diretorio do Banco     Diretorio Banco


============================== INDEX SUMMARY ===============================
============================== Table: pdBancoSistema ===========================

Flags: <p>rimary, <u>nique, <w>ord, <a>bbreviated, <i>nactive, + asc, - desc

Flags Index Name                       Cnt Field Name
----- -------------------------------- --- --------------------------------
pu    ixBancoSistema                     2 + bdCodSistema
                                           + bdNomeBanco

** Index Name: ixBancoSistema
 Storage Area: N/A


============================== FIELD DETAILS ===============================
============================== Table: pdBancoSistema ===========================

** Field Name: bdCodSistema
  Description: Contem o codigo do sistema
        Help: Informe o Codigo do Sistema

** Field Name: bdNomeBanco
  Description: Contem o nome do banco de dados
        Help: Informe o nome do banco de dados

** Field Name: bdDirBanco
  Description: Contem o diretorio do banco de dados
        Help: Informe o diretorio onde se encontra o banco de dados
```

```
================================================================================
============================== Table: pdConfig ===============================

              Table Flags: "f" = frozen, "s" = a SQL table


Table                          Dump     Table Field Index Table
Name                           Name     Flags Count Count Label
------------------------------ -------- ----- ----- ----- --------------------
pdConfig                       pdconfig          41     1 Configuracao de Pad

  Description: Tabela para configuracao de padroes
 Storage Area: N/A


============================== FIELD SUMMARY ==============================
============================== Table: pdConfig ==============================

Flags: <c>ase sensitive, <i>ndex component, <m>andatory, <v>iew component

Order Field Name                Data Type   Flags Format          Initial
----- ------------------------- ----------- ----- --------------- ----------
   10 bdCodConfig               inte        im    >>9             0
   20 bdNomeConfig              char        cmv   x(35)
   30 bdArqProg                 char        c     x(10)
   40 bdArqIncl ud              char        c     x(10)
   50 bdBancoDados              char        c     x(10)
   60 bdTabelas                 char        c     x(10)
   70 bdCamposTab               char        c     x(10)
   80 bdPalReserv               char        c     x(10)
   90 bdTabelaTemp              char        c     x(10)
  100 bdWorkFile                char        c     x(10)
  110 bdFuncoes                 char        c     x(10)
  120 bdProcedures              char        c     x(10)
  130 bdVariaveis               char        c     x(10)
  140 bdParametros              char        c     x(10)
  150 bdEscopFunc               char        c     x(10)
  160 bdEscopProc               char        c     x(10)
  170 bdEscopInclud             char        c     x(10)
  180 bdEscopGlobal             char        c     x(10)
  190 bdEscopLocal              char        c     x(10)
  200 bdInteger                 char        c     x(10)
  210 bdCharacter               char        c     x(10)
  220 bdLogical                 char        c     x(10)
  230 bdDecimal                 char        c     x(10)
  240 bdDate                    char        c     x(10)
  250 bdComHandle               char        c     x(10)
  260 bdHandle                  char        c     x(10)
  270 bdMemptr                  char        c     x(10)
  280 bdRaw                     char        c     x(10)
  290 bdRecid                   char        c     x(10)
  300 bdRowid                   char        c     x(10)
  310 bdWidgetHnd               char        c     x(10)
  320 bdButton                  char        c     x(10)
  330 bdBrowse                  char        c     x(10)
  340 bdBuffer                  char        c     x(10)
  350 bdFrame                   char        c     x(10)
  360 bdImage                   char        c     x(10)
  370 bdMenu                    char        c     x(10)
  380 bdQuery                   char        c     x(10)
  390 bdRectangle               char        c     x(10)
  400 bdStream                  char        c     x(10)
  410 bdSubMenu                 char        c     x(10)

Field Name                     Label                  Column Label
------------------------------ ---------------------- ----------------------
bdCodConfig                    Codigo da Configuracao Cod.Config
bdNomeConfig                   Nome da Configuracao   Nome da Configuracao
bdArqProg                      Arquivo de Programa    Arq.Prog.
bdArqIncl ud                   Arquivos de Includes   Arq.Includ
bdBancoDados                   Banco de Dados         Banco de Dados
bdTabelas                      Tabelas                Tabelas
bdCamposTab                    Campo da Tabela        Campo da Tabela
bdPalReserv                    Palavras Reservadas    Pal.Reserv.
bdTabelaTemp                   Tabelas Temporarias    Tab.Temp.
```

```
bdWorkFile                      Work Files             Work Files
bdFuncoes                       Funcoes                Funcoes
bdProcedures                    Procedures             Procedures
bdVariaveis                     Variaveis              Variaveis
bdParametros                    Parametros             Parametros
bdEscopFunc                     Decl.Intern.Funcao     Decl.Int.Func.
bdEscopProc                     Var.Interna Procedure  Var.Int.Proc.
bdEscopInclud                   Decl.Int.Includes      Decl.Int.Includ.
bdEscopGlobal                   Declaracoes Globais    Decl.Globais
bdEscopLocal                    Declaracoes Locais     Declar.Locais
bdInteger                       Tipos Integer          Tip.Integer
bdCharacter                     Tipos Character        Tip.Char.
bdLogical                       Tipos Logical          Tip.Logic
bdDecimal                       Tipos Decimal          Tip.Decimal
bdDate                          Tipos Date             Tipo Date
bdComHandle                     Tipos ComHandle        Tip.ComHandle
bdHandle                        Tipos Handle           Tipo Handle
bdMemptr                        Tipos Memptr           Tip.emptr
bdRaw                           Tipos Raw              Tipos Raw
bdRecid                         Tipos Recid            Tipo Recid
bdRowid                         Tipos Rowid            Tipo Rowid
bdWidgetHnd                     Tipos Widget-Handle    Tip.WidgtHnd
bdButton                        Button                 Button
bdBrowse                        Browse                 Browse
bdBuffer                        Buffer                 Buffer
bdFrame                         Frame                  Frame
bdImage                         Image                  Image
bdMenu                          Menu                   Menu
bdQuery                         Query                  Query
bdRectangle                     Rectangle              Rectangle
bdStream                        Stream                 Stream
bdSubMenu                       Sub-Menu               Sub-Menu


============================== INDEX SUMMARY ==============================
============================== Table: pdConfig ============================

Flags: <p>rimary, <u>nique, <w>ord, <a>bbreviated, <i>nactive, + asc, - desc

Flags Index Name                        Cnt Field Name
----- --------------------------------- --- ---------------------------------
pu    ixConfig                            1 + bdCodConfig

** Index Name: ixConfig
 Storage Area: N/A


============================== FIELD DETAILS ==============================
============================== Table: pdConfig ============================

** Field Name: bdCodConfig
  Description: Codigo da configuracao do padrao
         Help: Digite o codigo da configuracao do padrao

** Field Name: bdNomeConfig
  Description: Nome da configuracao do padrao
         Help: Digite o nome da configuracao do padrao

** Field Name: bdArqProg
  Description: Formatacao dos nomes dos arquivos de programas
         Help: Faca a formatacao para os nomes dos arquivos de programa

** Field Name: bdArqInclud
  Description: Formatacao dos nomes dos arquivos de includes
         Help: Faca a formatacao para os nomes dos arquivos de includes

** Field Name: bdBancoDados
  Description: Formatacao dos nomes dos banco de dados
         Help: Faca a formatacao para os nomes dos banco de dados

** Field Name: bdTabelas
  Description: Formatacao dos nomes das tabelas
         Help: Faca a formatacao para os nomes das tabelas

** Field Name: bdCamposTab
```

```
  Description: Formatacao dos nomes dos campos das tabelas
         Help: Faca a formatacao para os nomes dos campos das tabelas

** Field Name: bdPalReserv
  Description: Formatacao das palavras reservadas
         Help: Faca a formatacao para as palavras reservadas da linguagem

** Field Name: bdTabelaTemp
  Description: Formatacao dos nomes das tabelas temporarias
         Help: Faca a formatacao para os nomes das tabelas temporarias

** Field Name: bdWorkFile
  Description: Formatacao dos nomes dos work files
         Help: Faca a formatacao para os nomes dos work files

** Field Name: bdFuncoes
  Description: Formatacao dos nomes das funcoes
         Help: Faca a formatacao para os nomes das funcoes

** Field Name: bdProcedures
  Description: Formatacao dos nomes das procedures
         Help: Faca a formatacao para os nomes das procedures

** Field Name: bdVariaveis
  Description: Formatacao dos nomes das variaveis
         Help: Faca a formatacao para os nomes das variaveis

** Field Name: bdParametros
  Description: Formatacao dos nomes dos parametros
         Help: Faca a formatacao para os nomes dos parametros

** Field Name: bdEscopFunc
  Description: Formatacao dos nomes das declaracoes internas das funcoes
         Help: Faca a formatacao para o nome das declaracoes internas
               funcoes

** Field Name: bdEscopProc
  Description: Formatacao para o nome das declaracoes internas das
               procedures
         Help: Faca a formatacao para os nomes das declar. internas
               procedure

** Field Name: bdEscopInclud
  Description: Formatacao dos nomes das declaracoes internas das includes
         Help: Faca a formatacao para os nomes das declar. internas
               includes

** Field Name: bdEscopGlobal
  Description: Formatacao dos nomes das declaracoes globais
         Help: Faca a formatacao para os nomes das declaracoes globais

** Field Name: bdEscopLocal
  Description: Formatacao dos nomes das declaracoes locais
         Help: Faca a formatacao para os nomes das declaracoes locais

** Field Name: bdInteger
  Description: Formatacao dos nomes dos tipos integer
         Help: Faca a formatacao para os nomes dos tipos integer

** Field Name: bdCharacter
  Description: Formatacao dos nomes dos tipos character
         Help: Faca a formatacao para os nomes dos tipos character

** Field Name: bdLogical
  Description: Formatacao dos nomes dos tipos logical
         Help: Faca a formatacao para os nomes dos tipos logical

** Field Name: bdDecimal
  Description: Formatacao dos nomes dos tipos decimal]
         Help: Faca a formatacao para os nomes dos tipos decimal

** Field Name: bdDate
  Description: Formatacao dos nomes dos tipos date
         Help: Faca a formatacao para os nomes dos tipos date
```

```
** Field Name: bdComHandle
  Description: Formatacao dos nomes dos tipos ComHandle
         Help: Faca a formatacao para os nomes dos tipos ComHandle

** Field Name: bdHandle
  Description: Formatacao dos nomes dos tipos Handle
         Help: Faca a formatacao para os nomes dos tipos Handle

** Field Name: bdMemptr
  Description: Formatacao dos nomes dos tipos Memptr
         Help: Faca a formatacao para os nomes dos tipos Memptr

** Field Name: bdRaw
  Description: Formatacao dos nomes dos tipos Raw
         Help: Faca a formatacao para os nomes dos tipos Raw

** Field Name: bdRecid
  Description: Formatacao dos nomes dos Tipos Recid
         Help: Faca a formatacao para os nomes dos tipos Recid

** Field Name: bdRowid
  Description: Formatacao dos nomes dos tipos Rowid
         Help: Faca a formatacao para os nomes dos tipos Rowid

** Field Name: bdWidgetHnd
  Description: Formatacao dos nomes dos tipos Widget-Handle
         Help: Faca a formatacao para os nomes dos tipos Widget-Handle

** Field Name: bdButton
  Description: Formatacao dos nomes dos Button's
         Help: Faca a formatacao para os nomes dos Button's

** Field Name: bdBrowse
  Description: Formatacao dos nomes dos Browser's
         Help: Faca a formatacao para os nomes dos Browser's

** Field Name: bdBuffer
  Description: Formatacao dos nomes dos Buffer's
         Help: Faca a formatacao para os nomes dos Buffer's

** Field Name: bdFrame
  Description: Formatacao dos nomes dos Frames
         Help: Faca a formatacao para os nomes dos Frames

** Field Name: bdImage
  Description: Formatacao dos nomes dos Images
         Help: Faca a formatacao para os nomes dos Images

** Field Name: bdMenu
  Description: Formatacao dos nomes dos Menus
         Help: Faca a formatacao para os nomes dos Menus

** Field Name: bdQuery
  Description: Formatacao dos nomes das Queries
         Help: Faca a formatacao para os nomes das Queries

** Field Name: bdRectangle
  Description: Formatacao dos nomes dos Rectangle's
         Help: Faca a formatacao para os nomes dos Rectangle's

** Field Name: bdStream
  Description: Formatacao dos nomes das Stream's
         Help: Faca a formatacao para os nomes das Stream's

** Field Name: bdSubMenu
  Description: Formatacao dos nomes dos Sub-Menu's
         Help: Faca a formatacao para os nomes dos Sub-Menu's
```

```
================================================================================
============================== Table: pdDefPadrao ==============================

              Table Flags: "f" = frozen, "s" = a SQL table


Table                          Dump     Table Field Index Table
Name                           Name     Flags Count Count Label
------------------------------ -------- ----- ----- ----- --------------------
pdDefPadrao                    pddefpad           4     1 Definicoes de Padro

  Description: Tabela que contem as definicoes para padronizacao
 Storage Area: N/A


============================== FIELD SUMMARY ===============================
============================== Table: pdDefPadrao ==========================

Flags: <c>ase sensitive, <i>ndex component, <m>andatory, <v>iew component

Order Field Name                 Data Type   Flags Format          Initial
----- ------------------------- ----------- ----- --------------- ----------
   10 bdCodSistema              inte        m     zz9             0
   20 bdCodConfig               inte        m     >>9             0
   30 bdCorrigeCod              logi              yes/no          no
   40 bdImprimeRel              logi              yes/no          no

Field Name                     Label                  Column Label
------------------------------ ---------------------- ----------------------
bdCodSistema                   Codigo do Sistema      Cod.Sistema
bdCodConfig                    Codigo da Configuracao Cod.Config
bdCorrigeCod                   Corrigir Codigo        Corrigir
bdImprimeRel                   Imprimir Relatorio     Imprimir


============================== INDEX SUMMARY ===============================
============================== Table: pdDefPadrao ==========================

Flags: <p>rimary, <u>nique, <w>ord, <a>bbreviated, <i>nactive, + asc, - desc

Flags Index Name                       Cnt Field Name
----- -------------------------------- --- --------------------------------
p     ixDefPadrao                        1 + bdCodSistema


** Index Name: ixDefPadrao
 Storage Area: N/A


============================== FIELD DETAILS ===============================
============================== Table: pdDefPadrao ==========================

** Field Name: bdCodSistema
  Description: Contem o codigo do sistema
         Help: Informe o Codigo do Sistema

** Field Name: bdCodConfig
  Description: Codigo da configuracao do padrao
         Help: Digite o codigo da configuracao do padrao

** Field Name: bdCorrigeCod
         Help: Opcao para corrigir ou nao o codigo fonte fora do padrao.

** Field Name: bdImprimeRel
         Help: Optar por imprimir ou nao relacao das expressoes fora do
               padrao
```

```
================================================================================
============================== Table: pdParamGerais ======================

              Table Flags: "f" = frozen, "s" = a SQL table


Table                          Dump     Table Field Index Table
Name                           Name     Flags Count Count Label
------------------------------ -------- ----- ----- ----- --------------------
pdParamGerais                  pdexepar           6     1 Parametros para Exe

  Description: Tabela de parametros para execucao de processos
 Storage Area: N/A


============================== FIELD SUMMARY ==============================
============================== Table: pdParamGerais ======================

Flags: <c>ase sensitive, <i>ndex component, <m>andatory, <v>iew component

Order Field Name                Data Type   Flags Format          Initial
----- ------------------------- ----------- ----- --------------- ----------
   10 bdCodSistema              inte        im    zz9             0
   20 bdArqI                    logi              yes/no          no
   30 bdArqP                    logi              yes/no          no
   40 bdArqW                    logi              yes/no          no
   50 bdVarreIncludes           logi              yes/no          no
   60 bdTipoProc                inte        im    9               0

Field Name                     Label                  Column Label
------------------------------ ---------------------- ----------------------
bdCodSistema                   Codigo do Sistema      Cod.Sistema
bdArqI                         Pesquisar arquivos *.i *.i
bdArqP                         Pesquisar arquivos *.p *.p
bdArqW                         Pesquisar arquivos *.w *.w
bdVarreIncludes                Varrer Includes Intern Varre Includes
bdTipoProc                     Tipo de Processo       Tipo de Processo


============================== INDEX SUMMARY ==============================
============================== Table: pdParamGerais ======================

Flags: <p>rimary, <u>nique, <w>ord, <a>bbreviated, <i>nactive, + asc, - desc

Flags Index Name                       Cnt Field Name
----- -------------------------------- --- --------------------------------
pu    ixExeParam                         2 + bdCodSistema
                                           + bdTipoProc

** Index Name: ixExeParam
 Storage Area: N/A


============================== FIELD DETAILS ==============================
============================== Table: pdParamGerais ======================

** Field Name: bdCodSistema
  Description: Contem o codigo do sistema
         Help: Informe o Codigo do Sistema

** Field Name: bdArqI
  Description: Selecionar arquivos ".i"
         Help: Informe se deseja selecionar arquivos ".i"

** Field Name: bdArqP
  Description: Selecionar arquivos ".p"
         Help: Informe se deseja selecionar arquivos ".p"

** Field Name: bdArqW
  Description: Selecionar arquivos ".w"
         Help: Informe se deseja selecionar arquivos ".w"

** Field Name: bdVarreIncludes
  Description: Varrer includes internas ou nao
         Help: Informe se deve varrer includes internas ou nao
```

```
** Field Name: bdTipoProc
  Description: Contem o Tipo de Processo (1)Tab x Cod (2)Cod x Tab
               (3)Padronizar
         Help: Informe o Tipo de Processo (1)Tab x Cod (2)Cod x Tab
               (3)Padrao


================================================================================
============================== Table: pdSistema ==============================

              Table Flags: "f" = frozen, "s" = a SQL table


Table                          Dump     Table Field Index Table
Name                           Name     Flags Count Count Label
------------------------------ -------- ----- ----- ----- --------------------
pdSistema                      pdsistem           2     1 Sistema

  Description: Tabela de Sistemas
 Storage Area: N/A


============================== FIELD SUMMARY ==============================
============================== Table: pdSistema ==============================

Flags: <c>ase sensitive, <i>ndex component, <m>andatory, <v>iew component

Order Field Name                Data Type   Flags Format          Initial
----- ------------------------- ----------- ----- --------------- ----------
   10 bdCodSistema              inte        im    zz9             0
   20 bdNomeSistema             char        m     x(20)

Field Name                     Label                  Column Label
------------------------------ ---------------------- ----------------------
bdCodSistema                   Codigo do Sistema      Cod.Sistema
bdNomeSistema                  Nome do Sistema        Nome Sistema


============================== INDEX SUMMARY ==============================
============================== Table: pdSistema ==============================

Flags: <p>rimary, <u>nique, <w>ord, <a>bbreviated, <i>nactive, + asc, - desc

Flags Index Name                       Cnt Field Name
----- -------------------------------- --- --------------------------------
pu    ixSistema                          1 + bdCodSistema

** Index Name: ixSistema
 Storage Area: N/A


============================== FIELD DETAILS ==============================
============================== Table: pdSistema ==============================

** Field Name: bdCodSistema
  Description: Contem o codigo do sistema
         Help: Informe o Codigo do Sistema

** Field Name: bdNomeSistema
  Description: Contem o nome do sistema
         Help: Informe o nome do sistema



================================================================================
============================== Table: pdTabSel ==============================

              Table Flags: "f" = frozen, "s" = a SQL table


Table                          Dump     Table Field Index Table
Name                           Name     Flags Count Count Label
------------------------------ -------- ----- ----- ----- --------------------
pdTabSel                       pdtabsel           6     1 Tabelas Selecionada
```

```
  Description: Tabelas Selecionadas
 Storage Area: N/A


============================== FIELD SUMMARY ==============================
============================== Table: pdTabSel ============================

Flags: <c>ase sensitive, <i>ndex component, <m>andatory, <v>iew component

Order Field Name                Data Type   Flags Format          Initial
----- ------------------------- ----------- ----- --------------- ----------
   10 bdCodSistema              inte        m     zz9             0
   40 bdSelTab                  logi              yes/no          no
   50 bdTabela                  char              x(20)
   60 bdNomeBanco               char        cm    x(20)
   70 bdTipoProc                inte        m     9               0

Field Name                    Label                  Column Label
----------------------------- ---------------------- ----------------------
bdCodSistema                  Codigo do Sistema      Cod.Sistema
bdSelTab                      Tabela Selecionada     Tab.Sel.
bdTabela                      Tabela                 Tabela
bdNomeBanco                   Nome do Banco de Dados Nome Banco
bdTipoProc                    Tipo de Processo       Tipo de Processo


============================== INDEX SUMMARY ==============================
============================== Table: pdTabSel ============================

Flags: <p>rimary, <u>nique, <w>ord, <a>bbreviated, <i>nactive, + asc, - desc

Flags Index Name                      Cnt Field Name
----- ------------------------------- --- --------------------------------
p     ixTabSel                          2 + bdCodSistema
                                          + bdTipoProc

** Index Name: ixTabSel
 Storage Area: N/A


============================== FIELD DETAILS ==============================
============================== Table: pdTabSel ============================

** Field Name: bdCodSistema
  Description: Contem o codigo do sistema
         Help: Informe o Codigo do Sistema

** Field Name: bdSelTab
  Description: Indica se a tabela esta selecionada ou nao
         Help: Indica se a tabela esta selecionada ou nao

** Field Name: bdTabela
  Description: Nome da Tabela
         Help: Nome da Tabela

** Field Name: bdNomeBanco
  Description: Contem o nome do banco de dados
         Help: Informe o nome do banco de dados

** Field Name: bdTipoProc
  Description: Contem o Tipo de Processo (1)Tab x Cod (2)Cod x Tab
               (3)Padronizar
         Help: Informe o Tipo de Processo (1)Tab x Cod (2)Cod x Tab
               (3)Padrao
```