UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

FERRAMENTA DE APOIO À AUTOMATIZAÇÃO DE TESTES ATRAVÉS DO TESTCOMPLETE PARA PROGRAMAS DESENVOLVIDOS EM DELPHI

ADRIANA FRONZA MARCOS

BLUMENAU
2007

2007/1-01

ADRIANA FRONZA MARCOS

FERRAMENTA DE APOIO A AUTOMATIZAÇÃO DE TESTES ATRAVÉS DO TESTCOMPLOTE PARA PROGRAMAS DESENVOLVIDOS EM DELPHI

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 Ciências da Computação — Bacharelado.

Prof(a). Joyce Martins , Mestre – Orientadora

BLUMENAU
2007

2007/1-01

FERRAMENTA DE APOIO A AUTOMATIZAÇÃO DE TESTES ATRAVÉS DO TESTCOMPLETE PARA PROGRAMAS DESENVOLVIDOS EM DELPHI

Por

ADRIANA FRONZA MARCOS

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

Presidente:	Prof(a). Joyce Martins, Mestre – Orientadora, FURB
Membro:	Prof. Everaldo Artur Grahl, Mestre - FURB
Membro:	Prof. Marcel Hugo, Mestre – FURB

Blumenau, 11 de julho de 2007.

AGRADECIMENTOS

Aos meus pais, que mesmo sem entender a importância deste trabalho para minha pessoa, sempre me deram força e acreditaram no meu potencial.

A meu marido e meu filho, que me ajudaram e entenderam minha ausência para a confecção do trabalho, mas principalmente por me ajudarem a seguir em frente.

Aos meus amigos, pelo carinho, incentivo e ajuda.

A minha orientadora, Joyce Martins, por ter acreditado na conclusão deste trabalho, por ser tão dedicada e amiga. Com certeza um exemplo de pessoa a ser seguido.

Obrigada a todos!

RESUMO

O presente trabalho demonstra o desenvolvimento de uma ferramenta para apoio a automatização de testes. A ferramenta utiliza arquivos de programas desenvolvidos em Delphi para gerar arquivos de testes na linguagem DelphiScript, servindo de entrada para a ferramenta de automatização de testes TestComplete. Para isto, foram utilizados analisadores léxico, sintático e semântico para a análise de arquivos DFM, contendo a definição da interface da aplicação, e de arquivos PAS, contendo o código fonte. São utilizados *templates* para formatação dos *scripts* de testes, buscando tornar a geração mais flexível.

Palavras-chave: Teste de software. Automatização de testes. Formulários Delphi. *Templates*. Geração de *scripts*.

ABSTRACT

The present work demonstrates the development of a tool for support of automatization of tests. The tool uses archives of programs developed in Delphi to generate archives of tests in the DelphiScript language, thus working as an entrance for the automatization tool for TestComplete tests. Analytical lexicon, syntactic and semantic were used for the analysis of the programs. Templates for formatting the tests were used in order to make the generation of the test archives more flexible.

Key-words: Test of software. Automatization of tests. Delphi forms. Templates. Generation of Code.

LISTA DE ILUSTRAÇÕES

Figura 1 – *Test Log* funcionalidade TestComplete 17
Figura 2 – Interface de uma agenda telefônica 18
Quadro 1 – Código para a ferramenta TestComplete para testar a inclusão de dados 18
Quadro 2 – Exemplo de definição de *tokens* 20
Quadro 3 – Trecho da gramática de arquivos DFM 21
Quadro 4 – Trecho da gramática de arquivos DFM com ações semânticas 22
Quadro 5 – Estrutura de um arquivo DFM 23
Figura 3 – Interface 23
Quadro 6 – Exemplo de *template* para a agenda telefônica 24
Quadro 7 - Campos gerados a partir do *template* da agenda telefônica 24
Quadro 8 – Código gerado a partir do *template* do quadro 6 24
Quadro 9 – Requisitos não funcionais 26
Quadro 10 – Requisitos funcionais 27
Quadro 11 – Código do *script* para componentes extraídos do arquivo DFM 27
Figura 4 – Interface para identificação de componentes 28
Quadro 12 - Arquivo DFM com identificação de componentes 29
Quadro 13 – Componentes extraídos do arquivo DFM do quadro 12 29
Quadro 14 – Arquivo DPR com identificação do formulário principal 30
Figura 5 – Formulário principal 30
Quadro 15 – Arquivo PAS do formulário principal com comentário especial 31
Quadro 16 – Arquivo DFM do formulário principal com identificação do componente para acesso a um formulário específico 31
Quadro 17 – Comentários reconhecidos pela ferramenta nos arquivos PAS 32
Figura 6 – Componentes identificados através dos comentários arquivo PAS. 32
Quadro 18 – Elementos especificados para definição dos *templates* 34
Quadro 19 – Exemplo de *template* 35
Quadro 20 – *Script* gerado a partir do *template* definido no quadro 19 37
Quadro 21 – Gramática de arquivos DFM 38
Quadro 22 – Significado das ações semânticas 38
Figura 7 – Diagrama de casos de uso 39
Quadro 23 – Detalhamento do caso de uso Configurar *Template* 40

Quadro 24 – Detalhamento do caso de uso Gerar Testes 41
Quadro 25 – Detalhamento do caso de uso Adicionar Comentários 42
Figura 8 – Diagrama de classes 43
Figura 9 – Classes da ferramenta 44
Figura 10 – Diagrama de atividades 45
Quadro 26 – Identificação de comentários especiais 46
Quadro 27 – Identificação de comentários especiais 47
Quadro 28 – Implementação das ações semânticas 47
Quadro 29 – Geração dos testes a partir de um *template* 48
Figura 11 – Interface da ferramenta de apoio à geração automática de testes 49
Figura 12 – Selecionando um arquivo de configuração 51
Figura 13 – Adicionando arquivos DFM 52
Quadro 30 – Formato dos arquivos de dados 52
Figura 14 – Executando um *script* 53
Figura 15 – Selecionando e configurando *templates* 54
Figura 16 – Configurando *templates* 54
Figura 17 – Gerando *scripts* de testes 55
Figura 18 – Arquivos gerados no diretório de saída 55
Figura 19 – Carregando os *scripts* de testes na ferramenta TestComplete 56
Figura 20 – Executando testes de inclusão de dados 57
Figura 21 – Executando testes com erro 57
Quadro 31 – *Script* de teste para inclusão de dados 58
Figura 22 – Retornando erro na execução dos *scripts* 58
Quadro 32 – Exemplo de *template* 68

LISTA DE TABELAS

Tabela 1 – Comparativo entre os testes manuais e automatizados 16

LISTA DE SIGLAS

GLC **–** Gramática Livre de Contexto

HTML **–** *HyperText Markup Language*

PHP **–** *Hypertext Preprocessor*

RF – Requisito funcional

RNF – Requisito Não funcional

UML **-** *Unified Modeling Language*

XML **-** *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO 11
1.1 OBJETIVOS DO TRABALHO 12
1.2 ESTRUTURA DO TRABALHO 12
2 FUNDAMENTAÇÃO TEÓRICA 14
2.1 TESTES DE SOFTWARE 14
2.1.1 Testes caixa preta 15
2.1.2 Testes de regressão 15
2.2 AUTOMATIZAÇÃO DE TESTES 16
2.2.1 Ferramenta TestComplete 17
2.3 GERAÇÃO DE CÓDIGO 18
2.3.1 Modelos de geradores de código 19
2.3.2 Etapas de desenvolvimento de um gerador 19
2.4 ANALISADORES DE LINGUAGENS 20
2.4.1 Análise léxica 20
2.4.2 Análise sintática 21
2.4.3 Análise semântica 21
2.5 FORMULÁRIOS DFM 22
2.6 MOTOR DE *TEMPLATE* 23
2.7 TRABALHOS CORRELATOS 24
3 DESENVOLVIMENTO DO TRABALHO 26
3.1 REQUISITOS PRINCIPAIS 26
3.2 ESPECIFICAÇÃO DOS *SCRIPTS* 27
3.3 ANÁLISE DE ENTRADA 37
3.4 ESPECIFICAÇÃO 39
3.4.1 Diagrama de casos de uso 39
3.4.2 Diagrama de classes 42
3.4.3 Diagrama de atividades 44
3.5 IMPLEMENTAÇÃO 45
3.5.1 Técnicas e ferramentas utilizadas 45
3.5.2 Implementação da ferramenta 46
3.5.3 Operacionalidade da implementação 49

3.6 RESULTADOS E DISCUSSÃO 59
4 CONCLUSÕES 60
4.1 EXTENSÕES 60
REFERÊNCIAS BIBLIOGRÁFICAS 62
APÊNDICE A – Demonstração de testes caixa preta, *template* e testes gerados 64

1 INTRODUÇÃO

Desenvolver software com qualidade tem sido o grande desafio do mercado, sendo o teste uma das fases mais importantes dentro da engenharia de software. Desta forma, a empresa que busca qualidade em seus produtos deve investir em atividades de testes, pois é através delas que são encontradas as falhas. Segundo Inthurn (2001, p. 51), "[...] a etapa de teste, [...] é de grande importância para a identificação e eliminação de falhas [...]". No entanto, esse processo em busca da qualidade de software e, sem o apoio de uma ferramenta automatizada, pode tornar-se trabalhoso, limitando-se a programas simples. Inthurn (2001, p. 52) afirma que "sem uma infra-estrutura (facilidades automatizadas ou não) para a realização dos testes, torna-se, por vezes, impraticável a sua aplicação de forma adequada."

Diante disto, algumas empresas têm investido na automatização dos testes, utilizando ferramentas para essa finalidade, pois essas visam uma verificação mais rápida e eficiente das falhas no software. São exemplos de ferramentas desse tipo: Rational Visual Test (ARNOLD, 1999), TestComplete (AUTOMATEDQA CORPORATION, 2006) e Borland SilkPerformer Component Test (BORLAND SOFTWARE CORPORATION, 2006).

> A automatização dos testes é altamente desejada por diversos fatores, inclusive em termos de custos finais. [...] À medida que reexecutamos os testes, o ganho de tempo, controle, confiabilidade e as diversas possibilidades existentes com essa tecnologia, fica clara a vantagem inerente a esse processo. (BARTIÉ, 2002, p. 197).

Entretanto, a implantação de uma ferramenta de automatização de testes exige um investimento inicial por parte da empresa: recursos humanos, tempo e dinheiro. No início da utilização de uma ferramenta automatizada é dispensado um tempo significativo para o desenvolvimento dos *scripts* de testes. Em muitos casos, leva-se mais tempo para desenvolver os testes do que a funcionalidade em si. Além disso, sempre que o software passa por alterações é preciso atualizar esses *scripts* para que continuem funcionando corretamente.

Sendo assim, neste trabalho é apresentada uma ferramenta de apoio à geração automática de testes, que tem como objetivo diminuir o tempo de desenvolvimento e de atualização dos *scripts* de testes, para programas desenvolvidos em Delphi. A ferramenta terá como entrada formulários Delphi (arquivos com extensão DFM) contendo os componentes de interface da aplicação e o código fonte (arquivos com extensão PAS). Foram desenvolvidos analisadores léxico, sintático e semântico para extrair dos formulários os dados necessários para gerar os *scripts* de teste. Dos arquivos do código fonte são extraídas as assinaturas dos métodos para acesso aos formulários; para inclusão, alteração, exclusão e gravação dos dados,

entre outros. A saída da ferramenta é gerada utilizando *templates*[1].

Os *scripts* de testes são gerados em DelphiScript, uma das linguagens de *script* suportadas pela ferramenta de automação de testes TestComplete. TestComplete, segundo AutomatedQA Corporation (2006, p. 5), trata-se de uma ferramenta de automação de testes para aplicações desenvolvidas em Delphi, Microsoft Visual Basic, Microsoft Visual C++, Java, entre outras linguagens. Podem ser implementados vários tipos de testes, entre eles, o unitário, o de *performance*, o caixa preta e o de regressão.

Os testes gerados pela ferramenta são do tipo caixa preta. Assim, são tratadas as entradas e as saídas da aplicação. Conseqüentemente, os *scripts* são utilizados como testes de regressão, uma vez que a ferramenta permite que estes sejam executados quantas vezes for necessário.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver uma ferramenta para geração automática de testes para programas desenvolvidos em Delphi.

Os objetivos específicos do trabalho são:

a) desenvolver analisadores léxico, sintático e semântico para a extração das informações necessárias, contidas nos arquivos com extensão DFM;
b) utilizar *templates* para a formatação dos *scripts* de teste a serem gerados;
c) gerar código de teste para a ferramenta TestComplete;
d) gerar *scripts* para testes de caixa preta.

1.2 ESTRUTURA DO TRABALHO

O trabalho está organizado em quatro capítulos. O segundo capítulo contempla a fundamentação teórica, onde são apresentados conceitos e características sobre testes,

[1] Pode-se definir *template* como "um modelo que serve como guia para a construção de código. No *template* podem-se definir trechos estáticos ou dinâmicos, que serão utilizados para gerar a saída desejada." (SILVEIRA, 2006, p. 16).

automatização de testes, geração de código, analisadores de linguagens, formulários DFM, motores de *templates* e os trabalhos correlatos. No capítulo 3 são descritos os requisitos, a especificação dos *scritps*, a análise da entrada, a especificação e a implementação da ferramenta de apoio à geração automática de testes, bem como a funcionalidade da mesma. Por fim, são apresentados os resultados obtidos. No último capítulo são apresentadas a conclusão e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

No presente capítulo são apresentados alguns aspectos teóricos relacionados ao trabalho. Na seção 2.1 é dada uma visão geral sobre teste de software. Em seguida, a seção 2.2 trata da importância de se utilizar ferramentas de apoio aos testes, mais especificamente sobre a automatização dos testes. A seção 2.3 trata de tipos de geradores de códigos, modelos e etapas de desenvolvimento. A seção 2.4 aborda analisadores léxico, sintático e semântico, enquanto a seção 2.5 descreve a especificação dos formulários Delphi. Na seção 2.6 é apresentada a funcionalidade do motor de *template* FastTrac e, finalizando o capítulo, a seção 2.7 descreve os trabalhos correlatos.

2.1 TESTES DE SOFTWARE

> Durante o processo de desenvolvimento de um software existem atividades que procuram garantir a qualidade do produto final; entretanto, apesar dos métodos, técnicas e ferramentas utilizadas, falhas no produto ainda podem ocorrer. Assim a etapa de teste, a qual representa uma das atividades de garantia de qualidade, é de grande importância para a identificação e eliminação de falhas, representando assim o último passo do desenvolvimento do software. (INTHURN, 2001, p. 51).

De acordo com Moreira Filho e Rios (2003, p. 9), ao se testar um software tem-se como intenção verificar se o software está fazendo o que deveria, ou seja, o processo de teste tem a intenção de encontrar defeitos nos softwares.

Normalmente os testes são executados dentro da empresa pelos próprios desenvolvedores e pelos usuários do sistema. Porém, desta maneira o teste somente garante que os requisitos do sistema estão funcionando. Em um modelo de garantia de qualidade de software isto é insuficiente. O ideal é que seja utilizada uma metodologia para o processo de testes, tendo como objetivo minimizar os riscos causados por defeitos não descobertos. Nesta metodologia os testes devem ser executados por especialistas treinados para tal (RIOS, 2005).

Existem várias categorias ou tipos de testes de software. É fundamental entender que cada categoria possui seu ciclo de teste independente, uma vez que suas naturezas são muitas vezes conflitantes. Cada categoria de teste possui um objetivo determinado a ser alcançado (PAULA, 2005). Segundo Bartié (2002, p. 104), a categorização estabelece como serão organizados e estruturados os diversos cenários a serem executados, possibilitando à equipe

de testes conduzir seus trabalhos de forma mais objetiva e focada. A seguir são abordados dois tipos de testes: caixa preta e de regressão.

2.1.1 Testes caixa preta

O teste de caixa preta é o tipo de teste utilizado para se testar a interface do software. Neste tipo de teste são fornecidos dados de entrada, são executados os testes e o resultado obtido é comparado com um resultado previamente conhecido. Os testes de caixa preta são utilizados para demonstrar que o software é operacional e que a integridade das informações externas é mantida. (PLENTZ, 2001).

Ou ainda, conforme *Teste de software* (2007), o componente de software a ser testado é abordado como se fosse uma caixa-preta, pois não se considera o comportamento interno do mesmo. Haverá sucesso no teste se o resultado obtido (resultado do teste) for igual ao resultado esperado (resultado previamente conhecido).

2.1.2 Testes de regressão

O teste de regressão é a fase de teste aplicável a uma nova versão de software ou à necessidade de se executar um novo ciclo de teste durante o processo de desenvolvimento. Consiste em aplicar, a cada nova versão do software, todos os testes que já foram aplicados nas versões anteriores do sistema. Inclui-se nesse contexto a observação de fases e técnicas de teste de acordo com o impacto de alterações provocado pela nova versão. Para efeito de aumento de produtividade e de viabilidade dos testes, é recomendada a utilização de ferramentas de automatização de testes, de forma que, sobre a nova versão, todos os testes anteriores possam ser re-executados com maior agilidade (TESTE..., 2007).

Os testes regressivos têm como objetivo assegurar que alterações em partes do produto não afetaram as partes já testadas, ou melhor dizendo, re-executar um subconjunto (total ou parcial) de testes previamente executados. A maior utilidade dos testes de regressão aparece durante o processamento de solicitações de manutenção (PAULA, 2005).

2.2 AUTOMATIZAÇÃO DE TESTES

A fase de teste é essencial no processo de desenvolvimento de software. No entanto, apesar de representar um custo alto para as empresas, em grande parte não proporciona os resultados esperados, uma vez que as rotinas de teste se trata de um trabalho maçante, repetitivo e que exige tempo e organização por parte dos testadores (INTHURN, 2001, p. 51-54).

Este motivo fez com que algumas empresas investissem na automatização dos testes, utilizando ferramentas que automatizam essas rotinas. Com a utilização destas ferramentas, pode-se identificar erros com mais eficiência. A tabela 1 apresenta uma comparação entre o teste manual e o teste automatizado de 1.750 casos de testes com 700 defeitos existentes. Através da análise desta tabela, Bartié (2002, p. 64) afirma que os testes automatizados são mais econômicos, rápidos e eficientes do que os testes manuais.

Tabela 1 – Comparativo em horas entre os testes manuais e automatizados

etapas dos testes	**teste manual**	**teste automatizado**	**melhoria (%)**
Planejamento	32	40	-25 %
definição de casos de testes	262	117	55 %
execução dos testes	466	23	95 %
conferência dos testes	117	58	50 %
gerenciamento do erro	117	23	80 %
relatórios finais	96	16	83 %
duração total (em horas)	1.090	277	75 %

Fonte: Bartié (2002, p. 64).

Acredita-se, segundo Tomelin (2001, p. 28), que a automatização de testes significa a substituição do testador. No entanto, a automatização é apenas uma maneira de tornar os testes mais eficientes e confiáveis, representando um apoio à execução dos mesmos, pois é essencial o papel do testador que cria e atualiza os *scripts*. Com a automatização, os testes tornam-se mais precisos e encontram-se rapidamente as falhas, pois os *scripts* podem ser executados a qualquer momento, repetindo os mesmos testes, algo que seria um trabalho tedioso ao testador.

Existem algumas ferramentas para automatização de testes, entre as quais pode-se citar a TestComplete (AUTOMATEDQA CORPORATION, 2006). Porém, conforme Moreira Filho e Rios (2003, p. 152), a automatização de testes é muito mais do que simplesmente adquirir uma ferramenta, pois sua implementação não é trivial e nem barata, requer desenvolvimento e manutenção dos *scripts* e da ferramenta de automatização de testes, além de se tratar de um investimento a longo prazo .

2.2.1 Ferramenta TestComplete

A ferramenta TestComplete, segundo AutomatedQA Corporation (2006, p. 5), é um ambiente utilizado para automatização de testes de aplicações Windows e .NET. Fornece apoio para testar projetos desenvolvidos em Microsoft Visual C++, Microsoft Visual Basic, Delphi, C++Builder, C# e Java, bem como aplicações web. Paula (2005) afirma que, embora a própria ferramenta seja uma aplicação Windows, tanto o sistema operacional quanto o tipo do servidor web testado não importam, isto é, podem ser testados servidores de diversas plataformas (Windows, Linux, etc.).

TestComplete fornece várias funcionalidades, entre elas o *Test Log*, que é onde podem ser verificados os resultados da execução dos testes, conforme mostrado na figura 1.

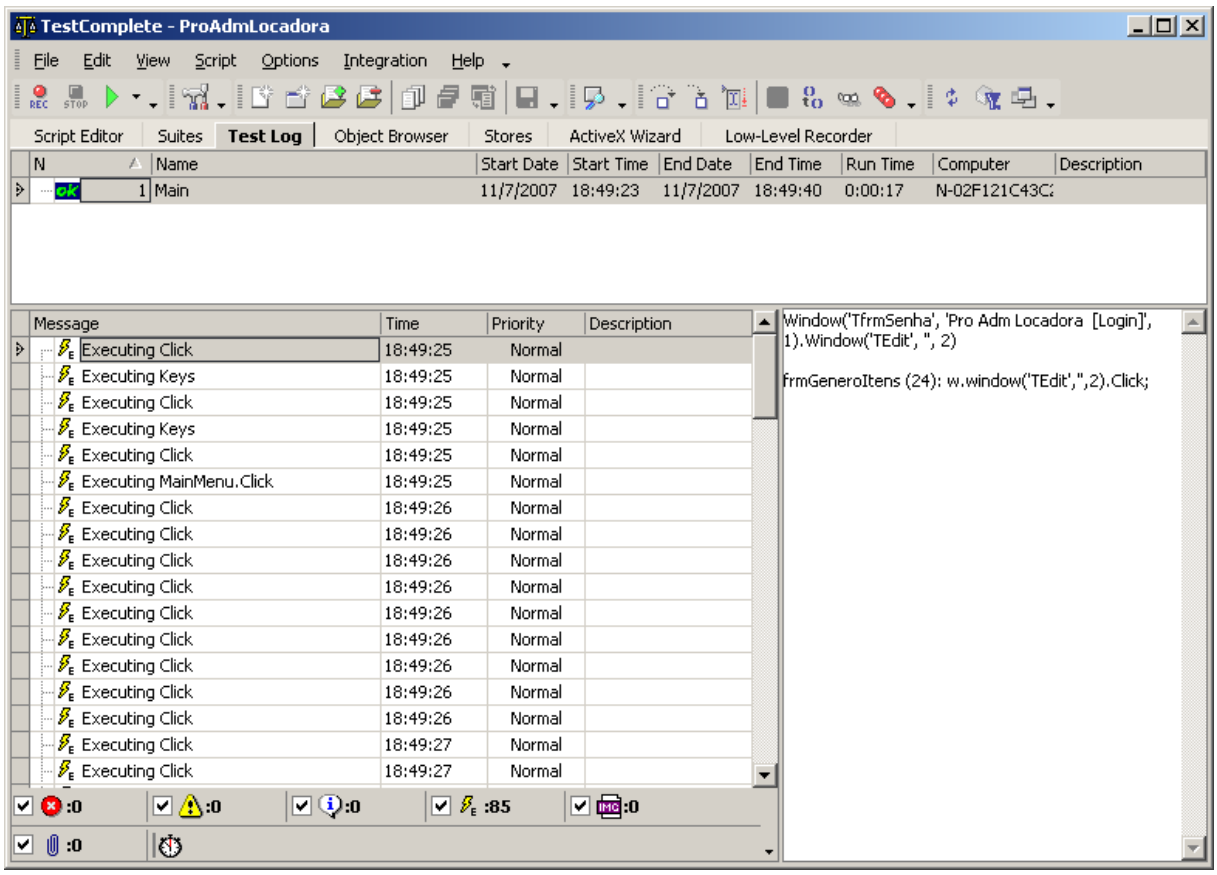

Figura 1 – *Test Log* funcionalidade TestComplete

Além disso, para elaborar os *scripts* de teste deve-se usar uma das cinco linguagens de *scripts* suportadas pela ferramenta (AUTOMATEDQA CORPORATION, 2006, p. 82). Assim, aqueles que dominam a linguagem Object Pascal podem optar por desenvolver os *scripts* em DelphiScript, um subconjunto de Object Pascal. Esta funcionalidade visa auxiliar no aprendizado da linguagem de *script* e, conseqüentemente, no uso da ferramenta. Dessa

forma, para testar, por exemplo, a inclusão de dados em uma agenda telefônica desenvolvida em Delphi (figura 2), através da ferramenta TestComplete, pode-se escrever o código de teste apresentado no quadro 1.

Figura 2 – Interface de uma agenda telefônica

```
procedure Inclui_Dados (TelaRef : OleVariant);
  var TelaPar : OleVariant;
begin
  TelaPar := TelaRef.VCLObject('AgendaForm');        // identificação do formulário
  TelaPar.VCLObject('ComboNome').Keys('Adriana Fronza Marcos');
                                                     // preenche nome a  ser incluído
  TelaPar.VCLObject('EditTelRes').Keys('33260507');  // preenche telefone residencial
  TelaPar.VCLObject('EditTelCom').Keys('32218888');  // preenche telefone comercial
  TelaPar.VCLObject('EditTelCel').Keys('91673213');  // preenche telefone celular
  TelaPar.VCLObject('BitBtnSalvar').Keys('[Enter]'); // clica no botão salvar
end;
```

Quadro 1 – Código para a ferramenta TestComplete para testar a inclusão de dados

2.3 GERAÇÃO DE CÓDIGO

Geração de código consiste em utilizar um arquivo como entrada para gerar um outro como saída. Segundo Herrington (2003, p. 3, tradução nossa), “geração de código é uma técnica que usa programas para gerar programas. Podem ser desde simples formatadores de código até ferramentas que geram aplicações complexas [...]”. Os geradores de código processam arquivos contendo texto escrito em alguma linguagem, podendo ser linguagens de programação com construções variadas como comentários, classes e métodos, ou arquivos com modelos abstratos. Para análise de comentários podem ser utilizados padrões definidos

através de expressões regulares. Para análise de classes e métodos ou outras construções sintáticas, bem como arquivos com modelos abstratos, podem ser implementados analisadores de linguagens.

2.3.1 Modelos de geradores de código

Os geradores de código podem ser classificados de acordo com a complexidade, o uso, as entradas e as saídas. Herrington (2003, p. 77-81) cita os seguintes modelos:

a) formatação de código (*code munger*): lê código fonte e, em geral usando expressões regulares ou simples analisadores de linguagens, gera um ou mais arquivos como saída, a partir de *templates* ou programação específica. Isto é, a saída pode ser parcial ou completamente dependente do projeto do gerador;
b) expansão de código (*inline-code expander*): lê código com alguma marcação especial (escrita em uma "nova" linguagem) e constrói novo código (em linguagem de alto nível) usando o código de entrada como base, mas com seções de código expandido baseadas nas marcações do código original;
c) geração mista (*mixed-code generator*): lê código com alguma marcação especial (escrita em uma "nova" linguagem) e constrói novo código usando o código de entrada como base. É similar ao modelo anterior, exceto que a saída passa a ser entrada para o gerador;
d) geração parcial de classes (*partial-class generator*): lê uma definição abstrata e, usando *templates*, constrói código para um conjunto de classes que deve ser estendido com a implementação das subclasses e dos métodos;
e) geração de camadas da aplicação (*tier or layer generator*): lê uma definição abstrata e, usando *templates*, constrói código com todas as funcionalidades para uma camada completa de uma aplicação.

2.3.2 Etapas de desenvolvimento de um gerador

Para construir um gerador de código, inicialmente deve-se verificar se o problema pode ser resolvido aplicando as técnicas de geração de código. Caso seja possível, pode-se então definir o processo de desenvolvimento. Para o desenvolvimento de um gerador de

código, Herrington (2003, p. 77) recomenda as seguintes etapas:

a) escrever o código de saída: deve-se construir manualmente o código de saída, determinando o que e como deve ser à saída do gerador;
b) projetar o gerador: deve-se especificar qual o formato do arquivo de entrada, como a entrada será analisada e como o código de saída será gerado;
c) analisar a entrada: deve-se implementar o código para ler o arquivo de entrada e extrair as informações necessárias para construir a saída;
d) especificar os *templates*: deve-se criar os *templates* que serão usados para gerar a saída conforme o código especificado na primeira etapa;
e) gerar a saída: deve-se implementar o processamento que faz análise dos dados extraídos da entrada e gera o código conforme os *templates* especificados.

2.4 ANALISADORES DE LINGUAGENS

Os analisadores (léxico, sintático e semântico) são módulos que compõem a estrutura básica de um compilador que podem ser usados no desenvolvimento de geradores de código.

2.4.1 Análise léxica

Conforme Hiebert (2003, p. 14), a análise léxica "é a primeira fase de um tradutor. Sua principal função é fazer a leitura dos caracteres do programa fonte e produzir uma seqüência de *tokens* para ser utilizada pela próxima fase do tradutor". Os *tokens* podem ser classificados como palavras reservadas, identificadores, símbolos especiais, constantes de tipos básicos (inteiro, real, literal), entre outras categorias (HIEBERT, 2003, p. 14) e são definidos através de expressões regulares, conforme mostrado no quadro 2.

```
letra: [a-zA-Z]
digito: [0-9]
identificador: {letra} ({letra}|{digito})*
constante_inteira: ("+"|"-")? {digito}+
```

Quadro 2 – Exemplo de definição de *tokens*

No quadro 2 tem-se as seguintes expressões regulares: `letra`, que representa o alfabeto, ou seja, letras maiúsculas e minúsculas; `digito`, que representa os dígitos entre 0 e

9; `identificador`, uma seqüência que inicia com uma `letra` seguida ou não (`*`) de um conjunto de `letras` ou `dígitos`; e `constante_inteira` que inicia ou não (`?`) com o sinal unário seguido de um ou mais (`+`) `dígitos`.

"Além de reconhecer os símbolos léxicos, o analisador também realiza outras funções, como armazenar alguns desses símbolos (tipicamente identificadores e constantes) em tabelas internas e indicar a ocorrência de erros léxicos." (PRICE; TOSCANI, 2001, p. 17).

2.4.2 Análise sintática

Constitui a segunda fase do tradutor e tem como função verificar se as construções utilizadas no programa fonte estão gramaticalmente corretas. O analisador sintático vê o texto do programa fonte como uma sentença que deve satisfazer as regras gramaticais de uma gramática livre de contexto (GLC) (HIEBERT, 2003, p. 15). Ao receber a sentença, seqüência de *tokens* reconhecida pelo analisador léxico, o analisador sintático irá produzir uma árvore de derivação quando esta for válida ou, do contrário, emitirá uma mensagem de erro (PRICE; TOSCANI, 2001, p. 29). No quadro 3 encontra-se uma parte da gramática utilizada no desenvolvimento deste trabalho.

```
...
<Objeto>::= object identificador : <Tipo>
                    <ListaPropriedades>
                    <ListaObjetos>
            end
<Tipo>              ::= identificador
<ListaPropriedades>::= ε | <Propriedade> <ListaPropriedades>
<ListaObjetos>      ::= ε | <Objeto> <ListaObjetos>
...
```

Quadro 3 – Trecho da gramática de arquivos DFM

No quadro 3 pode-se observar que um componente de interface deve ter a palavra reservada **`object`**, seguida de um identificador e de `:`, seguidos do tipo do componente (outro identificador), contendo uma lista de propriedades e uma lista de objetos.

2.4.3 Análise semântica

O analisador semântico utiliza a árvore sintática produzida pelo analisador sintático para determinar o significado do programa-fonte. São funções do analisador semântico: criar e manter a tabela de símbolos, contendo os identificadores encontrados no programa-fonte;

identificar operadores e operandos das expressões; fazer verificações de compatibilidade de tipo; analisar o escopo e o uso dos identificadores; e fazer verificações de correspondência entre parâmetros reais e formais.

A semântica é definida através de ações semânticas inseridas na gramática, como pode ser observado no quadro 4.

```
...
<Objeto>::= object identificador #1 : <Tipo> #2
                    <ListaPropriedades>
                    <ListaObjetos>
            end #7
<Tipo>              ::= identificador
<ListaPropriedades>::= ε | <Propriedade> <ListaPropriedades>
<ListaObjetos>     ::= ε | #5 <Objeto> <ListaObjetos>
...
```

Quadro 4 – Trecho da gramática de arquivos DFM com ações semânticas

2.5 FORMULÁRIOS DFM

Os formulários Delphi ou arquivos DFM são arquivos que contêm as informações dos componentes de interface presentes em cada formulário de uma aplicação Delphi. Nele encontram-se informações como posicionamento dos componentes, tamanho, valor das propriedades, entre outras. Assim, para cada componente, tem-se uma identificação, uma lista de propriedades com valores associados e outros possíveis objetos componentes (SASSE, 2005). Dependendo da configuração no Delphi, esses arquivos podem ser tanto arquivos binários como arquivos texto.

O quadro 5 ilustra a estrutura do arquivo DFM cuja interface está representada na figura 3. A primeira coluna do quadro 5 refere-se ao número da linha do arquivo DFM. Tem-se que no formulário `FrmPrincipal` foram especificados três objetos componentes: um *label*, um *edit* e um *button*. O componente *label*, por exemplo, foi identificado como `LbNome` (linha 23), sendo suas propriedades definidas nas linhas subseqüentes (da linha 24 a 28). A especificação dos componentes `EditTelRes` (linhas 51 a 56) e `BitBtnSalvar` (linhas 92 a 99) também encontram-se no quadro 5. Na figura 3 os três componentes estão destacados.

```
1   object FrmPrincipal:
    TFrmPrincipal
22  ...
23     object LbNome: TLabel
24        Left = 16
25        Top = 14
26        Width = 31
27        Height = 13
28        Caption = 'Nome:'
       end
50  ...
51     object EditTelRes: TEdit
52        Left = 16
53        Top = 83
54        Width = 265
55        Height = 21
56        TabOrder = 0
       end
91  ...
92     object BitBtnSalvar:
93                          TBitBtn
94        Left = 13
95        Top = 237
96        Width = 97
97        Height = 25
98        Caption = 'Salvar'
          TabOrder = 1
?      end
    ...
    end
```

Quadro 5 – Estrutura de um arquivo DFM

Figura 3 – Interface

2.6 MOTOR DE *TEMPLATE*

> Gerar código significa construir complexos arquivos textos estruturados. Para manter a integridade e a simplicidade do gerador, deve-se usar *templates*. Um *template* pode possuir, além de um conteúdo estático, um código dinâmico composto por variáveis e comandos estruturados que serão substituídos quando do seu processamento. Assim, com o uso de *templates* pode-se manter a formatação do código separada da lógica que determina o que deve ser construído. (SILVEIRA, 2006, p. 30).

Para implementação da ferramenta proposta optou-se pelo motor de *template* FastTrac Template Engine. O FastTrac trata-se de um componente desenvolvido para Delphi, que associado a *scripts* PHP gera arquivos a partir de um *template* definido. Para demonstrar o funcionamento deste motor de *template* segue exemplo no quadro 6. Este *template* pode ser utilizado para gerar comandos de teste para a ferramenta TestComplete para os campos existentes no formulário da figura 3.

```
<--section name=sequencia loop=$AgendaTelefonica-->

<--section name=SeqCampos loop=$AgendaTelefonica[sequencia].Campos -->
  <--$AgendaTelefonica[sequencia].Campos[SeqCampos].-->
<--/section-->

<--$AgendaTelefonica[sequencia].CampoSalvar-->
<--/section-->
```

Quadro 6 – Exemplo de *template* para a agenda telefônica

Os comandos da linguagem de *templates* devem estar entre os símbolos `<--section name=NomeDaSequencia loop=$Identificador-->` e `<--/section-->`, sendo que o valor atribuído ao `$Identificador` poderá ser acessado a partir de uma seqüência (`NomeDaSequencia`) pré-definida, como pode ser verificado no quadro 7.

```
<--$AgendaTelefonica[0].Campos[0].--> = TComboBox ComboNome ["Nome:"]
<--$AgendaTelefonica[0].Campos[1].--> = TEdit EditTelRes["Telefone residencial:"]
<--$AgendaTelefonica[0].Campos[2].--> = TEdit EditTelCom ["Telefone comercial:"]
<--$AgendaTelefonica[0].Campos[3].--> = TEdit EditTelCel ["Telefone celular:"]
<--$AgendaTelefonica[0].Campos[4].--> = TBitBtn BitBtnSalvar ["botão Salvar"]
```

Quadro 7 - Campos gerados a partir do *template* da agenda telefônica

Sendo assim, para o *template* definido no quadro 6, o código gerado para o formulário da figura 3 seria conforme mostrado no quadro 8.

```
objeto.Window('TComboBox','',1).Click;              // ComboNome
objeto.Window('TEdit','',3).Click;                  // EditTelRes
objeto.Window('TEdit','',2).Click;                  // EditTelCom
objeto.Window('TEdit','',1).Click;                  // EditTelCel
objeto.Window('TBitBtn','Salvar').Click;            // BitBtnSalvar
```

Quadro 8 – Código gerado a partir do *template* do quadro 6

2.7 TRABALHOS CORRELATOS

Foram analisados alguns trabalhos correlatos envolvendo os seguintes assuntos: ferramentas de automatização de testes e geração de código a partir de formulários Delphi, utilização de *templates*. Dentre os trabalhos estudados, foram selecionados: Rational Visual Test, DUnit, ConDado, DelphiToWeb e Delphi2Java-II.

Rational Visual Test (Visual Test) é, segundo Tomelin (2001, p. 29), uma ferramenta de automatização de testes semelhante à ferramenta TestComplete. Assim como o TestComplete, simula ações de usuários. Visual Test também não substitui o papel humano no ciclo de teste, pois depende da implementação dos mesmos, isto é, é preciso dizer à ferramenta o que ela deve fazer. Esta ferramenta está inserida em uma plataforma semelhante à do Developer Studio Visual C++, que não só é usada pelo Visual Test como também é o editor principal do Microsoft Visual Studio, um agrupamento das ferramentas de

desenvolvimento da Microsoft. Devido a este fato, Visual Test se adapta melhor a programas desenvolvidos em Visual C ++. Para elaboração dos *scritps* de teste deve ser usada uma linguagem de *scripts* da própria ferramenta, semelhante ao Visual Basic.

DUnit trata-se de uma ferramenta para Delphi destinada à criação de testes unitários automatizados (PAULI, 2005). Através dela criam-se testes como parte integrante do desenvolvimento de aplicações Delphi. Seu principal objetivo é proporcionar ao programador um *feedback* instantâneo, onde a cada mudança no programa, rodam-se os testes, e assim tem-se imediatamente um retorno sobre o impacto da mudança na estabilidade do projeto. DUnit possui uma interface simples que permite que sejam selecionadas as classes a serem testadas e, após serem executados os testes, tem-se um resumo do que ocorreu no teste, incluindo o número de métodos testados, o número de falhas, o número de erros e o tempo de execução.

Siqueira (2005) afirma que ConDado é uma ferramenta para auxílio à geração de testes de software especificados na forma de máquinas de estado. A ferramenta tem como entrada uma descrição textual da máquina de estados, que representa o comportamento do sistema, e produz como saída os testes a serem aplicados. O uso de restrições permite controlar os testes gerados. Sendo assim, ela é útil para auxiliar nos testes de sistemas de comunicação, permitindo a criação das entradas, e permitindo que seja verificado o funcionamento do software, constatando se o mesmo atende ao que foi especificado.

DelphiToWeb consiste em uma ferramenta para auxiliar na migração de programas desenvolvidos em Delphi para aplicações web (SOUZA, 2005). A ferramenta converte componentes de interface Delphi em componentes equivalentes em HTML, arquivos com extensão LZX e arquivos XML. Trata-se de um gerador de código que tem como entrada arquivos DFM, a partir dos quais é gerado código para a camada de interface de uma aplicação web. Para gerar a saída são feitas as análises léxica, sintática e semântica desses arquivos.

Delphi2Java-II é uma ferramenta que converte formulários Delphi em aplicações Java. Silveira (2006, p. 73) relata que são convertidos 22 componentes de visualização, 7 componentes de visualização de dados e 4 componentes de acesso à banco de dados. Para analisar os formulários são utilizados analisadores léxico, sintático e semântico da solução descrita em Souza (2005). Para formatar a saída são utilizados *templates* através do motor de *templates* Velocity.

3 DESENVOLVIMENTO DO TRABALHO

A motivação para o desenvolvimento deste trabalho está relacionada com a necessidade de agilizar a automatização de testes de software. Sabendo-se da importância de se testar um software para que este tenha mais qualidade e de como uma ferramenta de automatização de testes agiliza esse processo, este trabalho busca diminuir o tempo de criação e atualização dos *scritps* de testes. Sendo assim, a partir dos códigos fonte de um programa desenvolvido em Delphi, a ferramenta extrai os dados necessários e gera os *scripts* de testes conforme o *template* configurado pelo usuário.

Para o desenvolvimento desta ferramenta foram realizados os seguintes passos:

a) especificação dos requisitos (seção 3.1);
b) especificação dos *scripts*, determinando quais dados devem ser extraídos dos arquivos DFM, arquivos PAS e arquivo DPR de um projeto Delphi (seção 3.2);
c) especificação da análise da entrada (seção 3.3);
d) especificação da ferramenta através dos diagramas de casos de uso, de classes e de atividades da UML (seção 3.4);
e) implementação da ferramenta (seção 3.5).

3.1 REQUISITOS PRINCIPAIS

Nos quadros 9 e 10 são apresentados, respectivamente, os requisitos não funcionais e funcionais da ferramenta.

REQUISITOS NÃO FUNCIONAIS	
RNF01	Ser compatível com o sistema operacional Windows.
RNF02	Ser desenvolvida utilizando o ambiente Borland Delphi.
RNF03	Utilizar um motor de *templates* para análise dos mesmos.

Quadro 9 – Requisitos não funcionais

REQUISITOS FUNCIONAIS	
RF01	Permitir que o usuário informe qual o projeto Delphi deve ser testado.
RF02	Permitir a seleção do diretório onde serão gerados os *scripts* de teste.
RF03	Permitir que o usuário informe quais serão os formulários Delphi testados.
RF04	Permitir que sejam informados arquivos de dados para os testes de inclusão, alteração e exclusão de registros para cada formulário.
RF05	Permitir que o usuário guarde as configurações para que este possa re-gerar os testes quando o software for atualizado.
RF06	Utilizar *templates* para formatar os *scripts* a serem gerados.
RF07	Permitir a seleção dos *templates* que serão usados para gerar a saída.
RF08	Realizar as análises léxica, sintática e semântica dos arquivos DFM para extrair as informações necessárias para gerar os *scripts* de teste.
RF09	Gerar *scritps* de teste em DelphiScript através de um *template.*

Quadro 10 – Requisitos funcionais

3.2 ESPECIFICAÇÃO DOS *SCRIPTS*

O primeiro passo na especificação dos *scripts* foi definir para quais componentes serão gerados testes. Sendo assim, considerando o uso freqüente nas aplicações, foram determinados os seguintes componentes: `TBitBtn`, `TButton`, `TCheckBox`, `TComboBox`, `TDBCheckBox`, `TDBEdit`, `TDBLookupComboBox`, `TEdit`, `TMainMenu`, `TRadioButton`, `TToolButton`.

Em seguida, foi verificado como estes componentes são utilizados nos *scritps* de teste da ferramenta TestComplete. O quadro 11 mostra cada um dos componentes definidos anteriormente, bem como o código do *script* de teste correspondente.

COMPONENTE	CÓDIGO DO *SCRIPT* DE TESTE
`TBitBtn`	`objeto.Window('TbitBtn','&Salvar').Click;`
`TButton`	`objeto.Window('Tbutton','&Novo').Click;`
`TCheckBox`	`objeto.Window('TCheckBox','CheckBox1').Click(cbChecked);`
`TComboBox`	`objeto.Window('TComboBox','',1).Click;`
`TDBCheckBox`	`objeto.Window('TDBCheckBox','DBCheckBox1').Click(cbChecked);`
`TDBEdit`	`objeto.Window('TDBEdit','',1).Click;`
`TDBLookupComboBox`	`objeto.Window('TDBLookupComboBox','',1).Click;`
`TEdit`	`objeto.Window('TEdit','',16).Click;`
`TMmainMenu`	`objeto.MainMenu.Click('Cadastro|Clientes');`
`TRadioButton`	`objeto.Window('TRadioButton','RadioButton1').Click;`
`TToolButton`	`objeto.Window('TToolButton','ToolButton1').Click;`

Quadro 11 – Código do *script* para componentes extraídos do arquivo DFM

Observa-se que, além do tipo do componente (primeiro parâmetro do código do *script*), deve ser informada a identificação do mesmo (último parâmetro). Alguns dos

componentes definidos são identificados pela propriedade `Caption`. Já os componentes que não têm essa propriedade são identificados por uma seqüência numérica em ordem decrescente do componente no arquivo DFM, sendo que para cada tipo de componente tem-se uma numeração diferente. O quadro 12 exemplifica a identificação de componentes em um arquivo DFM, conforme descrito. O arquivo DFM do quadro 12 (sem as propriedades dos componentes) correspondente à interface da figura 4.

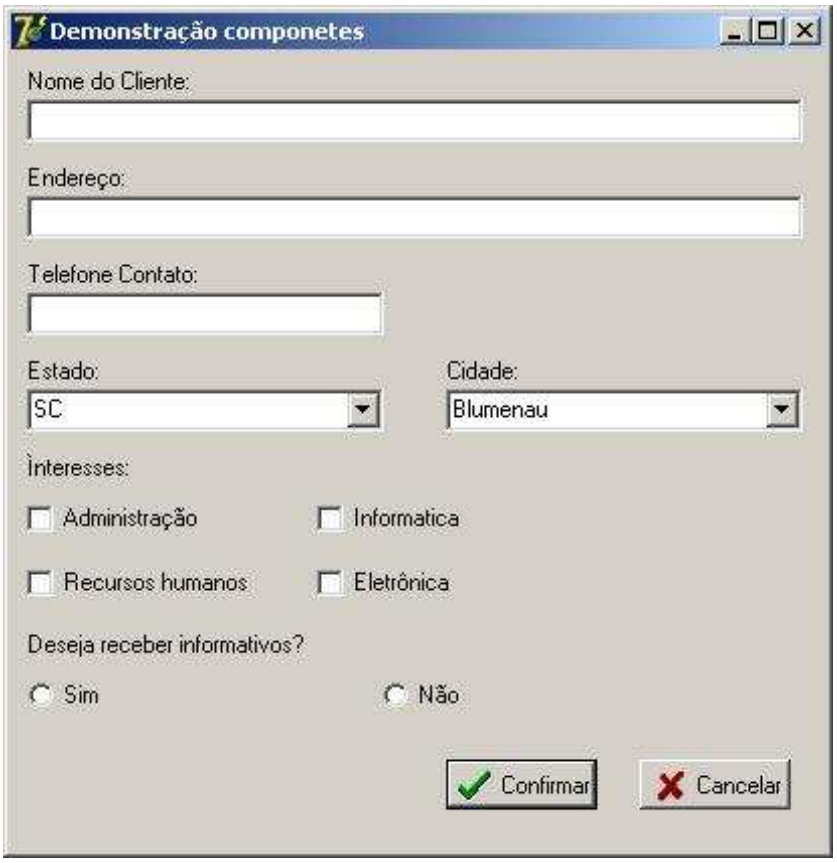

Figura 4 – Interface para identificação de componentes

COMPONENTES	IDENTIFICAÇÃO
object FrmDemonstracao: TFrmDemonstracao ...	
object EdNmCliente: TEdit ... **end**	Ordem: 3 – 1º componente TEdit
object EdEndCliente: TEdit ... **end**	Ordem: 2 – 2º componente TEdit
object EDTelefone: TEdit ... **end**	Ordem: 1 – 3º componente TEdit
object CBCidade: TComboBox ... **end**	Ordem: 2 – 1º componente TComboBox
object CBEstado: TComboBox ... **end**	Ordem: 1 – 2º componente TComboBox
object GroupBox1: TGroupBox **object** CBAdm: TCheckBox ... Caption = 'Administra'#231#227'o' **End**	Caption: 'Administração' Observa-se que os códigos ASCII são convertidos.
object CBRH: TCheckBox ... Caption = 'Recursos humanos' **End**	Caption:'Recursos humanos'
object CBInf: TCheckBox ... Caption = 'Informatica' **End**	Caption: 'Informatica'
object CBEle: TCheckBox ... Caption = 'Eletrônica' **end** **end**	Caption: 'Eletrônica'
object GroupBox2: TGroupBox **object** RBSim: TRadioButton ... Caption = 'Sim' **End**	Caption: 'Sim'
object RBNao: TRadioButton ... Caption = 'Não' **end** **end**	Caption: 'Não'
object BitBtnOK: TBitBtn ... Caption = 'Confirmar' **End**	Caption: 'Confirmar'
object BitBtnCancelar: TBitBtn ... Caption = 'Cancelar' **End**	Caption: 'Cancelar'
End	

Quadro 12 - Arquivo DFM com identificação de componentes

Para os componentes do quadro 12, devem ser gerados os códigos de *script* de teste do quadro 13.

COMPONENTE	CÓDIGO DO *SCRIPT* DE TESTE
EdNmCliente	objeto.Window('TEdit','',3).Click;
EdEndCliente	objeto.Window('TEdit','',2).Click;
EDTelefone	objeto.Window('TEdit','',1).Click;
CBCidade	objeto.Window('TComboBox','',2).Click;
CBEstado	objeto.Window('TComboBox','',1).Click;
CBAdm	objeto.Window('TCheckBox','Administração').Click(cbChecked);
CBRH	objeto.Window('TCheckBox','Recursos humanos').Click(cbChecked);
CBInf	objeto.Window('TCheckBox','Informatica').Click(cbChecked);
CBEle	objeto.Window('TCheckBox','Eletrônica').Click(cbChecked);
RBSim	objeto.Window('TRadioButton','Sim').Click;
RBNao	objeto.Window('TRadioButton','Não').Click;
BitBtnOK	objeto.Window('TBitBtn','Confirmar').Click;
BitBtnCancelar	objeto.Window('TBitBtn','Cancelar').Click;

Quadro 13 – Componentes extraídos do arquivo DFM do quadro 12

Para gerar os *scripts*, são necessários outros dados além daqueles extraídos dos componentes do arquivo DFM citados no quadro 12. Assim, tem-se duas opções: o usuário

deve informar os dados na interface da ferramenta; ou a ferramenta deve extrair os dados de arquivos do código fonte da aplicação a ser testada. Optou-se por identificar os dados utilizando outros arquivos do código fonte, aumentando assim a qualidade da ferramenta.

Para determinar qual é o formulário principal, a ferramenta analisa o arquivo DPR, buscando o primeiro *form* declarado, conforme mostrado no quadro 14.

```
program Evolution;

uses Forms,
     UfrmPrincipal in 'UfrmPrincipal.pas' {frmPrincipal},
     UfrmClientes in 'UfrmClientes.pas' {frmClientes},
...
```

Quadro 14 – Arquivo DPR com identificação do formulário principal

A maioria dos sistemas tem um formulário principal que dá acesso aos demais formulários através de botões e menus, entre outros componentes de interface (figura 5).

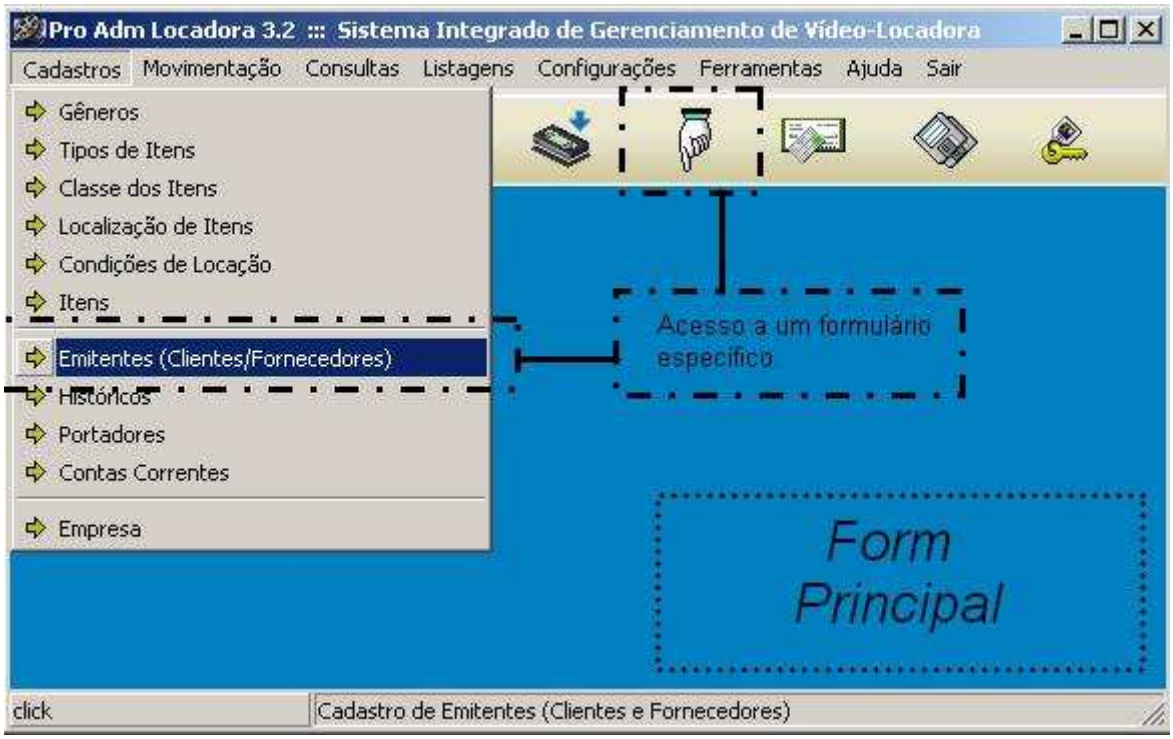

Figura 5 – Formulário principal

Não é possível identificar como acessar um determinado formulário somente através do arquivo DFM. Por este motivo, utiliza-se o arquivo PAS correspondente. Para tanto, foi especificado um comentário especial que deve ser incluído antes da assinatura do método que faz o acesso ao formulário desejado. Sendo assim, a ferramenta reconhece o comentário através de uma expressão regular, recupera o identificador do método e, através deste identificador, reconhece o componente de interface que acessa o formulário. O quadro 15 exemplifica o comentário especial utilizado para identificar os métodos que acessam determinado formulário.

```
//teste:UfrmClientes
procedure TfrmPrincipal.Clientes1Click(Sender: TObject);
begin
  try
     frmClientes:=TfrmClientes.Create(Application);
     frmclientes.ShowModal;
  finally
     frmClientes.Release;
     frmClientes:=nil;
  end;
end;
```

Quadro 15 – Arquivo PAS do formulário principal com comentário especial

Do arquivo PAS do formulário principal da aplicação (quadro 15), será extraído o nome do método (`Clientes1Click`) e, no arquivo DFM, será reconhecido o componente de interface que acessa o formulário `UfrmClientes` (quadro 16).

```
...
object MainMenu1: TMainMenu
    Left = 40
    Top = 64
    object Cadastro1: TMenuItem
      AutoCheck = True
      Caption = 'Cadastros'
      Hint = 'Cadastro de Clientes e Filmes'
      object Clientes1: TMenuItem
        Caption = 'Emitentes(Clientes/Fornecedores)'
        OnClick = Clientes1Click
      end
...
end
...
```

Quadro 16 – Arquivo DFM do formulário principal com identificação do componente para acesso a um formulário específico

Através da propriedade `OnClick` do objeto `Clientesl` (no quadro 16), cujo valor é o identificador do método recuperado (no quadro 15), recupera-se as informações para a montagem do código de teste para um formulário específico acessado através de uma opção do menu do formulário principal da aplicação, resultando em `objeto.MainMenu.Click( Cadastros|Emitentes(Clientes/Fornecedores))`. Para componentes do tipo botão basta identificar o botão através da propriedade `OnClick` e capturar a propriedade `Caption`.

Assim, por exemplo, se `UfrmClientes` for um dos formulários selecionados para que seja gerado *script* de teste, o usuário deverá incluir um comentário especial no arquivo PAS do formulário principal (quadro 15), identificando qual o método acessa `UfrmClientes`. Se o comentário não for incluído no arquivo PAS, o código não será gerado corretamente. De outra forma, o usuário pode optar por colocar o acesso fixo no *template* que será usado para gerar os *scripts*. Neste caso, o comentário não será necessário, porém o usuário estará configurando um *template* que pode ser usado somente para o formulário `UfrmClientes`, procedimento não recomendado pois a ferramenta faz o tratamento adequado.

A identificação dos componentes de interface utilizados para colocar um determinado

formulário em modo de inclusão, alteração e exclusão, assim como dos componentes que fazem a gravação dos dados e do que fecha o formulário, é outra informação que depende de arquivos além do DFM. Para resolver este problema uma solução seria definir que os campos teriam sempre um determinado nome. Porém para que a ferramenta se tornasse mais flexível, adotou-se a mesma solução de comentários especiais. Para isto, foram definidos comentários especiais para cada um destes componentes. Os comentários seguem os padrões apresentados no quadro 17.

COMENTÁRIOS	SIGNIFICADO
`//teste:Incluir`	reconhece a assinatura do método que faz a inclusão de dados no arquivo correspondente
`//teste:Alterar`	reconhece a assinatura do método que faz a alteração de dados no arquivo correspondente
`//teste:Excluir`	reconhece a assinatura do método que faz a exclusão de dados no arquivo correspondente
`//teste:Salvar`	reconhece a assinatura do método que faz salva os dados no arquivo correspondente
`//teste:Sair`	reconhece a assinatura do método que fecha o arquivo correspondente.

Quadro 17 – Comentários reconhecidos pela ferramenta nos arquivos PAS

Na figura 6 são ilustrados esses componentes em forma de botão.

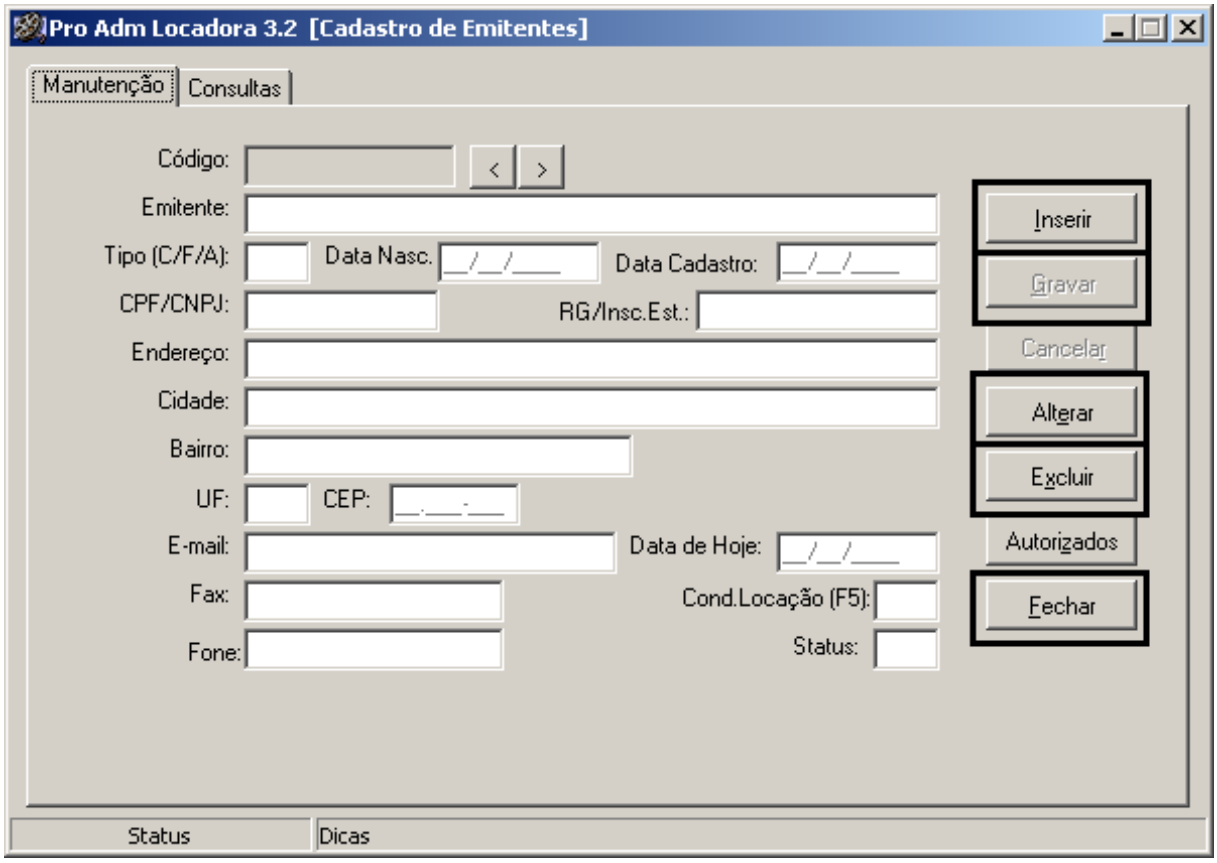

Figura 6 – Componentes identificados através dos comentários arquivo PAS.

Assim como o comentário utilizado para reconhecer o nome do método e, conseqüentemente, o componente que acessa um formulário, estes comentários são incluídos

pelo usuário antes da assinatura dos métodos correspondentes no arquivo PAS do formulário que será testado (no exemplo `UfrmClientes`). Da mesma forma, a ferramenta recupera o nome do método e, através deste nome, localiza o componente no arquivo DFM.

Ainda, no DFM do formulário principal, além dos componentes reconhecidos através dos comentários especiais, recupera-se o valor da propriedade `Caption` do formulário, uma vez que esse valor também é necessário na geração dos *scripts*. Já no arquivo DPR, além de se identificar o formulário principal, recupera-se o nome do executável, pois também trata-se de um dado utilizado no *script* de teste.

Após identificar todos os dados que a ferramenta precisava extrair para gerar os *scripts*, definiu-se para cada um deles um elemento na linguagem de *templates* do FastTrac. Sendo assim, ao montar o *template*, esses elementos devem ser utilizados, já que para gerar os *scripts* a ferramenta recupera os dados (dos arquivos DFM, DPR e PAS) e associa aos elementos definidos. No quadro 18 são listados os elementos especificados e seus significados.

ELEMENTO	SIGNIFICADO
`<--section name=i loop=$Objeto-->`	início do *script*
`<--$Objeto[i].Projeto-->`	nome do executável do projeto Delphi
`<--$Objeto[i].TipoFrmPrincipal-->`	tipo do *form* principal
`<--$Objeto[i].CaptionFrmPrincipal-->`	valor da propriedade `Caption` do *form* principal
`<--$Objeto[i].AcessoFrmPrincipal-->`	acesso aos formulários através de menu ou botão
`<--$Objeto[i].TipoFrmSelecionado-->`	tipo do *form* a ser testado
`<--$Objeto[i].CaptionFrmSelecionado-->`	valor da propriedade `Caption` do *form* a ser testado
`<--section name=j loop=$Objeto[i].FrmSelecionado-->`	início dos campos

`<--if`
`  $Objeto[i].FrmSelecionado[j].TipoCampo == TEdit`
`or $Objeto[i].FrmSelecionado[j].TipoCampo == TDBEdit`
`-->`
	verifica o tipo do campo (`TEdit` ou `TDBEdit`)

`<--if`
`  $Objeto[i].FrmSelecionado[j].TipoCampo == TCheckBox`
`or $Objeto[i].FrmSelecionado[j].TipoCampo == TDBCheckBox`
`or $Objeto[i].FrmSelecionado[j].TipoCampo == TRadioButton`
`-->`
	verifica o tipo do campo (`TCheckBox, TDBCheckBox` ou `TRadioButton`)

`<--if`
`  $Objeto[i].FrmSelecionado[j].TipoCampo == TComboBox`
`or $Objeto[i].FrmSelecionado[j].TipoCampo ==`
`TDBLookupComboBox`
`-->`
	verifica o tipo do campo (`TComboBox` ou `TDBLookupComboBox`)

`<--/if-->`	fim das verificações
`<--$Objeto[i].FrmSelecionado[j].CamposEntrada-->`	lista de campos reconhecidos através dos arquivos DFM dos *forms* a serem testados
`<--/section-->`	fim dos campos
`<--$Objeto[i].Incluir-->`	acesso ao método para incluir registros
`<--$Objeto[i].Alterar-->`	acesso ao método para alterar registros
`<--$Objeto[i].Excluir-->`	acesso ao método para excluir registros
`<--$Objeto[i].Salvar-->`	acesso ao método para salvar registros
`<--$Objeto[i].Sair-->`	forma para sair dos formulários através de menu ou botão
`<--$Objeto[i].DadosInclusao-->`	localização e nome do arquivo com os dados para inclusão de registros
`<--$Objeto[i].DadosAlteracao-->`	localização e nome do arquivo com os dados para alteração de registros
`<--$Objeto[i].DadosExclusao-->`	localização e nome do arquivo com os dados para exclusão de registros
`<--/section-->`	fim do *script*

Quadro 18 – Elementos especificados para definição dos *templates*

A partir desses elementos, montam-se os *templates*. O quadro 19 exemplifica um *template* utilizando alguns desses elementos.

```
<--section name=i loop=$Objeto-->
procedure Incluir;  forward;

procedure Main;
begin
  try
    TestedApps.RunAll();
    Incluir;
  except
    Log.Error('Exception', ExceptionMessage);
  end;
end;

procedure Incluir;
var
  processo, janela, janela2: OleVariant;
  AFileName, FileVar: OleVariant;
  PosBarra, campo, Linha: String;
begin
  PosBarra:=''; campo:=''; Linha:='';
  AFileName := '<--$Objeto[i].DadosInclusao-->';
  AssignFile(FileVar, AFileName);
  Reset(FileVar);
  processo := Sys.Process('<--$Objeto[i].Projeto-->');
  janela := processo.Window('<--$Objeto[i].TipoFrmPrincipal-->',
                                          '<--$Objeto[i].CaptionFrmPrincipal-->');
  janela.<--$Objeto[i].AcessoFrmPrincipal-->
  janela2:=processo.Window('<--$Objeto[i].TipoFrmSelecionado-->',
                                        '<--$Objeto[i].CaptionFrmSelecionado-->');
  while not Eof(FileVar) do begin
     janela2.<--$Objeto[i].Incluir-->
     Readln(FileVar, Linha);

     <--section name=j loop=$Objeto[i].FrmSelecionado -->
     PosBarra:=Pos(',',Linha);
     campo:=Copy(Linha,1,PosBarra-1);
     Delete(Linha,1,PosBarra);
     <--if $Objeto[i].FrmSelecionado[j].TipoCampo == TEdit
        or $Objeto[i].FrmSelecionado[j].TipoCampo == TDBEdit-->
        janela2.<--$Objeto[i].FrmSelecionado[j].CamposEntrada-->
        sys.keys(campo);
     <--/if-->
     <--if $Objeto[i].FrmSelecionado[j].TipoCampo == TCheckBox
        or $Objeto[i].FrmSelecionado[j].TipoCampo == TDBCheckBox
        or $Objeto[i].FrmSelecionado[j].TipoCampo == TRadioButton-->
        if (campo = 1) then
          janela2.<--$Objeto[i].FrmSelecionado[j].CamposEntrada-->
     <--/if-->
     <--if $Objeto[i].FrmSelecionado[j].TipoCampo == TComboBox
        or $Objeto[i].FrmSelecionado[j].TipoCampo == TDBLookupComboBox-->
        janela2.<--$Objeto[i].FrmSelecionado[j].CamposEntrada-->(campo);
     <--/if-->
     <--/section-->
  end;
  CloseFile(FileVar);
  janela2.<--$Objeto[i].Sair-->
end;
<--/section-->
```

Quadro 19 – Exemplo de *template*

A partir deste *template*, ao gerar um *script* de teste para a interface definida na figura 4, a ferramenta produz como resultado o *script* de teste do quadro 20.

```
procedure Incluir;  forward;

procedure Main;
begin
  try
    TestedApps.RunAll();
    Incluir;
  except
    Log.Error('Exception', ExceptionMessage);
  end;
end;

procedure Incluir;
var
  processo, janela, janela2: OleVariant;
  AFileName, FileVar: OleVariant;
  PosBarra, campo, Linha: String;
begin
  PosBarra:=''; campo:=''; Linha:='';
  AFileName := 'C:\DadosEntrada\Dados.txt';
  AssignFile(FileVar, AFileName);
  Reset(FileVar);
  processo := Sys.Process('ProjectoDemonstracao');
  janela := processo.Window('TFrmPrincipal', 'Form Principal');
  janela.Window('TButton','Acessar').Click;
  janela2:=processo.Window('TFrmDemonstracao', 'Demonstração componetes');
  while not Eof(FileVar) do  begin
     janela2.Window('TBitBtn','Confirmar').Click;
     Readln(FileVar, Linha);

     PosBarra:=Pos(',',Linha);
     campo:=Copy(Linha,1,PosBarra-1);
     Delete(Linha,1,PosBarra);
     janela2.Window('TEdit','',3).Click;
     sys.keys(campo);

     PosBarra:=Pos(',',Linha);
     campo:=Copy(Linha,1,PosBarra-1);
     Delete(Linha,1,PosBarra);
     janela2.Window('TEdit','',2).Click;
     sys.keys(campo);

     PosBarra:=Pos(',',Linha);
     campo:=Copy(Linha,1,PosBarra-1);
     Delete(Linha,1,PosBarra);
     janela2.Window('TEdit','',1).Click;
     sys.keys(campo);

     PosBarra:=Pos(',',Linha);
     campo:=Copy(Linha,1,PosBarra-1);
     Delete(Linha,1,PosBarra);
     janela2.Window('TComboBox','',2).Click(campo);

     PosBarra:=Pos(',',Linha);
     campo:=Copy(Linha,1,PosBarra-1);
     Delete(Linha,1,PosBarra);
     janela2.Window('TComboBox','',1).Click(campo);

     PosBarra:=Pos(',',Linha);
     campo:=Copy(Linha,1,PosBarra-1);
     Delete(Linha,1,PosBarra);
     if (campo = 1) then
        janela2.Window('TCheckBox','Eletrônica').Click(cbChecked);

     PosBarra:=Pos(',',Linha);
     campo:=Copy(Linha,1,PosBarra-1);
     Delete(Linha,1,PosBarra);
     if (campo = 1) then
        janela2.Window('TCheckBox','Recursos humanos').Click(cbChecked);
```

```
    PosBarra:=Pos(',',Linha);
    campo:=Copy(Linha,1,PosBarra-1);
    Delete(Linha,1,PosBarra);
    if (campo = 1) then
       janela2.Window('TCheckBox','Administração').Click(cbChecked);

    PosBarra:=Pos(',',Linha);
    campo:=Copy(Linha,1,PosBarra-1);
    Delete(Linha,1,PosBarra);
    if (campo = 1) then
       janela2.Window('TCheckBox','Informatica').Click(cbChecked);

    PosBarra:=Pos(',',Linha);
    campo:=Copy(Linha,1,PosBarra-1);
    Delete(Linha,1,PosBarra);
    if (campo = 1) then
       janela2.Window('TRadioButton','Não').Click;

    PosBarra:=Pos(',',Linha);
    campo:=Copy(Linha,1,PosBarra-1);
    Delete(Linha,1,PosBarra);
    if (campo = 1) then
      janela2.Window('TRadioButton','Sim').Click;
  end;
  CloseFile(FileVar);
  janela2.Window('TBitBtn','Cancelar').Click;
end;
```

Quadro 20 – *Script* gerado a partir do *template* definido no quadro 19

3.3 ANÁLISE DE ENTRADA

Para se extrair os dados necessários dos arquivos DFM, foram construídos analisadores léxico, sintático e semântico. Dessa forma, a gramática proposta por Souza (2005) foi utilizada como ponto inicial para análise da entrada da ferramenta. Foram necessárias algumas adaptações na gramática original para adequá-la ao problema que se pretende resolver, resultando na gramática apresentada no quadro 21.

```
<Dfm>                ::= <Objeto> #6

<Objeto>             ::=  objeto identificador #1 : <Tipo> #2
                                <ListaPropriedades>
                                <ListaObjetos>
                          fim #7
<Tipo>               ::= identificador

<ListaObjetos>       ::= ε | #5 <Objeto> <ListaObjetos>

<ListaPropriedades>::= ε | <Propriedade> <ListaPropriedades>

<Propriedade>        ::= <Nome> #3 = <Valor> #4

<Nome>               ::= identificador <Nome_>
<Nome_>              ::= ε | . identificador <Nome_>

<Valor>              ::=  <Numero> | constante_string <ListaStrings> | <Nome> |
                          <TipoBoolean> | [ <ListaValores1> ] | ( <ListaValores2> ) |
                          { <ListaValores2> } | <Colecao>

<Numero>             ::= constante_inteira | constante_real

<ListaStrings>       ::= ε | + constante_string <ListaStrings>

<TipoBoolean>        ::= falso | verdadeiro

<ListaValores1>      ::= ε | <Valor> <ListaValores1_>
<ListaValores1_>     ::= ε | , <Valor> <ListaValores1_>

<ListaValores2>      ::= <Valor> <ListaValores2_>
<ListaValores2_>     ::= ε | <ListaValores2>

<Colecao>            ::= < <ListaColecao> >
<ListaColecao>       ::= ε | <ItemColecao> <ListaColecao>
<ItemColecao>        ::= identificador <ListaPropriedades> fim
```

Quadro 21 – Gramática de arquivos DFM

As ações semânticas identificadas no quadro 21 (símbolo # seguido de um número inteiro) apresentam os significados descritos no quadro 22.

AÇÃO	SIGNIFICADO
1	armazena o nome do componente de interface, inicialmente um *form*, e cria uma lista para armazenar os sub-componentes
2	armazena o tipo do componente
3	identifica o tipo de propriedade de um componente, guardando as propriedades (`Caption`, `TabOrder`, `OnClick`), necessárias para gerar os *scripts* de testes
4	armazena o valor das propriedades (`Caption`, `TabOrder`, `OnClick`), identificadas pela ação #3
5	identifica o início da declaração de um componente de interface
6	identifica o término do arquivo DFM. A partir desta ação começa a geração dos *scripts* de testes
7	identifica o término da declaração de um objeto componente

Quadro 22 – Significado das ações semânticas

A partir da ação 1 obtém-se o identificador de um componente de interface. Para a geração dos *scripts* utiliza-se somente o identificador do formulário. Portanto, antes da ação semântica 5 ser executada pela primeira vez, sabe-se que ainda não foram definidos os componentes de interface. Conseqüentemente, a ferramenta armazena o identificador reconhecido como sendo do formulário analisado.

A ação semântica 4 guarda os valores das propriedades `Caption`, `TabOrder` e

`OnClick`, identificadas na ação semântica 3, somente para os componentes reconhecidos pela ferramenta através da ação semântica 2. A propriedade `TabOrder` é utilizada para que o *script* gere código para o preenchimento dos campos na ordem correta. Os componentes `TBitBtn`, `TButton`, `TCheckBox`, `TDBCheckBox`, `TMmainMenu`, `TRadioButton` e `TToolButton` utilizam a propriedade `Caption` para a geração dos *scripts*. Portanto a ferramenta guarda essa propriedade para estes componentes. Para os demais componentes (`TComboBox`, `TDBEdit`, `TDBLookupComboBox`, `TEdit`), guarda-se a ordem em que os mesmos estão localizados no DFM. Além disso, para os componentes `TBitBtn`, `TButton`, `TMmainMenu` e `TToolButton` armazena-se também o valor da propriedade `OnClick` e, através do valor desta propriedade, localiza-se os métodos reconhecidos pelos comentários especiais incluídos nos arquivos PAS.

3.4 ESPECIFICAÇÃO

Foi utilizada a UML como linguagem de especificação dos diagramas de casos de uso, de classes e de atividades, com a utilização da ferramenta Enterprise Architect.

3.4.1 Diagrama de casos de uso

Através do diagrama de casos de uso demonstra-se o funcionamento da ferramenta para o testador. Na figura 7 encontram-se os casos de uso da aplicação e nos quadros 23, 24 e 25 o detalhamento de cada um deles.

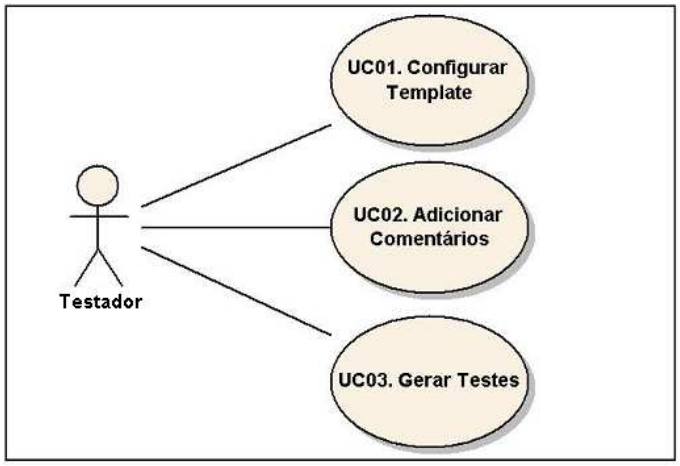

Figura 7 – Diagrama de casos de uso

UC01. Configurar *Template*
Pré-condição: O testador deve conhecer a linguagem DelphiScript e os *scripts* de testes para o TestComplete.
Cenário principal: 1. A ferramenta disponibiliza opção para configurar os *templates*. 2. O testador seleciona a opção para configurar os *templates* a partir do botão **Configurar**. 3. A ferramenta disponibiliza a interface para configuração dos *templates*. 4. O testador seleciona a opção para criar um *template* a partir do botão **Novo**. 5. O testador seleciona a opção para salvar o *template* a partir do botão **Salvar como**. 6. A ferramenta solicita o nome e a localização do arquivo. 7. O testador informa o nome e o diretório onde o arquivo deve ser salvo. 8. O testador pressiona o botão **OK** para selecionar o *template*.
Cenários alternativos: A1. Abrir *template*: no passo 4 o testador pode optar por abrir um *template* existente: 4. O testador seleciona a opção para abrir um *template* a partir do botão **Abrir** e seleciona o *template* desejado. 4.1 A ferramenta disponibiliza o *template* selecionado. 4.2 O testador altera os dados do *template.* Retorna ao passo 5 A2. Salvar: no passo 5 5. O testador seleciona a opção para salvar o *template* a partir do botão **Salvar**. 5.1 Caso o arquivo seja novo, retorna ao passo 6. 5.2 Caso o arquivo não seja novo, salva com o nome e no diretório de origem. Retorna ao passo 8.
Pós-condição: Um *template* foi criado ou alterado e selecionado.

Quadro 23 – Detalhamento do caso de uso Configurar *Template*

UC03. Gerar Testes

Pré-condição: Devem existir arquivos PAS com comentários, arquivos DFM de formulários a serem testados, arquivo DPR da aplicação e *templates* configurados.

Cenário principal:

1. A ferramenta disponibiliza a interface com os campos a serem preenchidos.
2. O testador informa o projeto Delphi a ser testado.
3. O testador informa o diretório onde serão gerados os *scripts* de teste.
4. O testadorseleciona o(s) arquivo(s) DFM.
5. O testador seleciona o arquivo com dados para inclusão de registros.
6. O testador informa o arquivo com dados para alteração de registros.
7. O testador informa o arquivo com dados para exclusão de registros.
8. O testador pressiona o botão **Adicionar à lista**.
9. A ferramenta adiciona o(s) arquivo(s) DFM à lista de arquivos a serem testados.
10. O testador seleciona o *template* a ser usado para gerar os *scripts*.
11. O testador pressiona no botão **Gerar testes**.
12. A ferramenta apresenta a mensagem "Arquivo(s) gerado(s) com sucesso!".

Cenários alternativos:

A1. Alterar: no passo 9, o testador pode optar por alterar a lista de arquivos DFM selecionados:

9. O testador seleciona o arquivo DFM já adicionado à lista.
   - 9.1 O testador pressiona o botão **Alterar**.
   - 9.2 A ferramenta habilita os campos (inclusão de dados, alteração de dados e exclusão de dados).
   - 9.3 O testador altera os dados necessários.
   - 9.4 O testador seleciona a opção para salvar as alterações a partir do botão **OK.**
   - 9.5 A ferramenta altera as informações para o arquivo DFM selecionado.

Retorna ao passo 10.

A2. Retirar: no passo 9, o testador pode optar por retirar arquivos da lista de arquivos DFM selecionados:

9. O testador seleciona o arquivo DFM já adicionado à lista.
   - 9.1 O testador pressiona o botão **Retirar da lista**.
   - 9.2 A ferramenta retira o arquivo da lista de arquivos DFM.

Retorna ao passo 10.

A3. Salvar arquivo: a partir do passo 11, o testador pode optar por salvar os dados informados em um arquivo de configuração:

11. O testador pressiona o botão **Salvar** ou **Salvar Como**:
    - 11.1 O testador informa o nome e diretório para o arquivo de configuração.
    - 11.2 A ferramenta salva o arquivo de configuração, contendo informações para geração dos *scripts* de teste.

Retorna ao passo 11.

Cenário de exceção:

E1. No passo 12, caso os arquivos informados não estejam léxica e sintaticamente corretos:

12. A ferramenta apresenta a mensagem: "Ocorreram erros na geração do arquivo: NomedoArquivo".

Pós-condição: *Scripts* de testes foram gerados.

Quadro 24 – Detalhamento do caso de uso Gerar Testes

UC02. Adicionar Comentários
Pré-condição: Devem existir arquivos PAS de uma aplicação a ser testada.
Cenário principal: 1. O testador acessa o arquivo PAS correspondente ao formulário desejado. 2. O testador identifica os métodos de acesso (inclusão, alteração, exclusão e gravação de registros), bem como o método que fecha o formulário em questão. 3. O testador inclui o comentário correspondente (`//teste:Incluir`, `//teste:Alterar`, `//teste:Excluir`, `//teste:Salvar`, `//teste:Sair`) antes da assinatura dos métodos identificados no passo anterior. 4. O testador inclui o comentário `//teste:NomeFormulario` antes da assinatura do método que faz acesso ao formulário informado. 5. O testador salva as alterações do arquivo PAS.
Pós-condição: Pelo menos um arquivo PAS possui os comentários que permitem que a ferramenta proposta reconheça os métodos necessários para gerar código.

Quadro 25 – Detalhamento do caso de uso Adicionar Comentários

3.4.2 Diagrama de classes

No diagrama de classes da figura 8 foram modeladas as classes que representam as análises léxica, sintática e semântica e as classes que armazenam os dados extraídos dos arquivos DFM necessários para conversão, assim como a classe que faz a geração dos testes através de *templates*. As classes representadas pela cor branca foram geradas pelo GALS (GESSER, 2003) e as demais implementadas na ferramenta.

A classe `Tsemantico`, destacada na figura, foi gerada pelo GALS (GESSER, 2003) porém foi implementada para atender os requisitos da ferramenta.

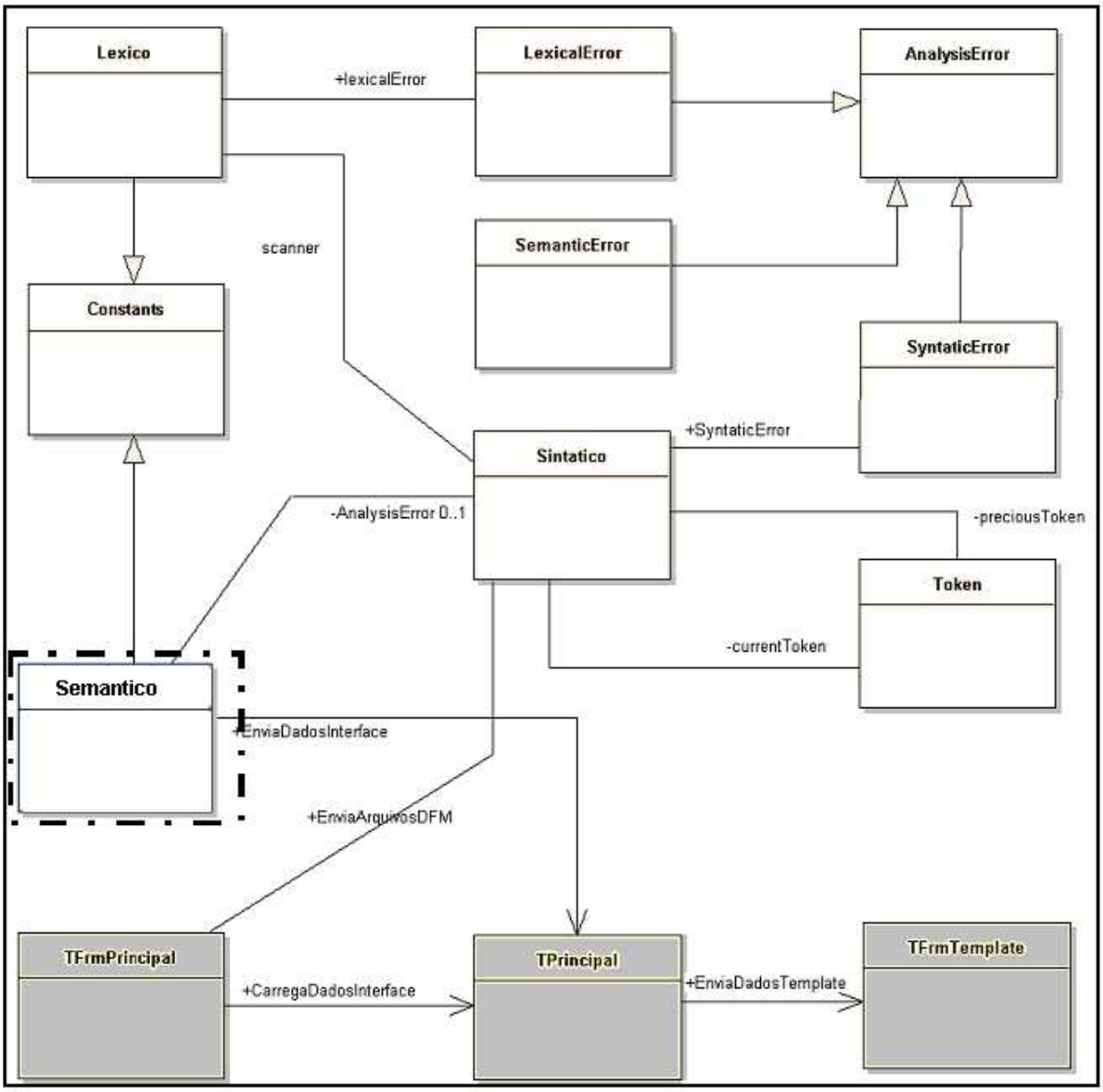

Figura 8 – Diagrama de classes

A classe `TFrmPrincipal` recebe os dados informados pelo usuário através da interface e cria um objeto da classe `TPrincipal` com esses dados, enviando o conteúdo dos arquivos DFM para a análise léxica, sintática e semântica. O objeto da classe `TSemantico` reconhece os componentes e suas propriedades e armazena os dados no objeto da classe `TPrincipal`. Finalizando, este objeto é enviado para a classe `TFrmTemplate` para que os testes sejam gerados a partir do *template* informado.

Na figura 9 são mostrados os atributos e métodos das classes implementadas na ferramenta.

Figura 9 – Classes da ferramenta

3.4.3 Diagrama de atividades

No diagrama de atividades apresentado na figura 10 demonstram-se mais detalhes dos passos que a ferramenta utiliza para o caso de uso Gerar Testes.

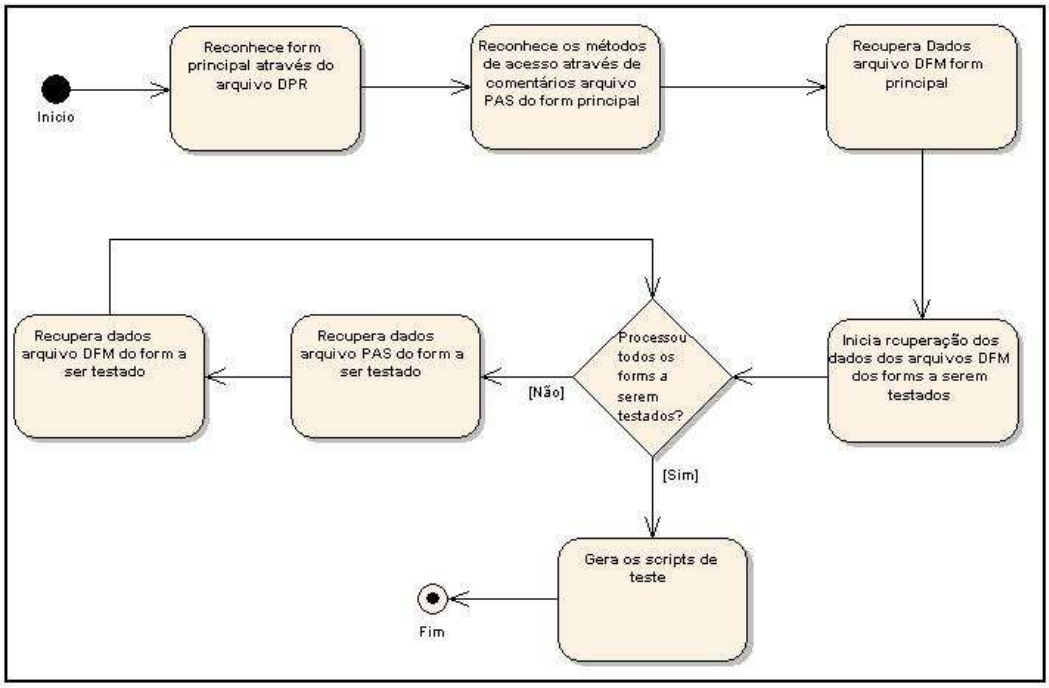

Figura 10 – Diagrama de atividades

3.5 IMPLEMENTAÇÃO

Nesta seção são apresentadas as técnicas e ferramentas utilizadas na fase de desenvolvimento do trabalho, bem como a operacionalidade da implementação.

3.5.1 Técnicas e ferramentas utilizadas

A ferramenta proposta foi desenvolvida em Delphi 6. O código que faz as análises léxica, sintática e semântica foi gerado através da ferramenta GALS (GESSER, 2003). Utilizou-se o componente FastTrac para a geração do código de testes através de *templates* e a *unit* `RegExpr` para o reconhecimento dos comentários especiais nos arquivos PAS, através de expressões regulares.

3.5.2 Implementação da ferramenta

O primeiro passo na implementação foi identificar o formulário principal da aplicação a ser testada no arquivo DPR correspondente. Para isto, a ferramenta lê o arquivo DPR e identifica qual o primeiro *form* declarado. Na seqüência são identificados os formulários a serem testados, através dos comentários especiais no formulário principal. Para identificar os comentários especiais, é utilizada a *unit* `RegExpr` que procura os comentários através de uma expressão regular. No quadro 26, a expressão regular identifica os métodos que acessam os vários formulários a serem testados.

```
Procedure TPrincipal.DadosArquivosPasPrincipal(FrmSelecionado:String);
var
        R1Print:TRegExpr;
        FileVar: TextFile;
        campo, Linha, msg: String;
        PosIni, PosFim, qtd:integer;
begin
  AssignFile(FileVar, GetArquivoPrincipal);     // arquivo principal menu de acesso
  Reset(FileVar);
  try
     R1Print := TRegExpr.Create;

     R1Print.Expression := '^\s*//\s*teste:'+FrmSelecionado+'*$';

     while not Eof(FileVar) do begin            // tratar fonte
       Readln(FileVar, Linha);
       if (R1Print.Exec(Linha)) then begin
          Readln(FileVar, Linha);
          PosIni:=Pos('.',Linha);
          PosFim:=Pos('(',Linha);
          ListaAcesso:= New(PAcesso);
          GuardaAcesso.Add(ListaAcesso);
          ListaAcesso^.form:=FrmSelecionado;
          ListaAcesso^.Procacesso:=Copy(Linha,PosIni+1,PosFim-PosIni-1);
          Break;
       end;
     end;
  finally
     FreeAndNil(R1Print);
     Close(FileVar);
  end;
end;
```

Quadro 26 – Identificação de comentários especiais

Para identificar os comentários para inclusão, alteração, exclusão e gravação de registros, assim como a opção para fechar um determinado formulário, também são utilizadas expressões regulares através do método do quadro 27.

```
Procedure TPrincipal.DadosArquivosPas();
...
     //iniciar expressões
     R2Print.Expression := '^\s*//\s*teste:Incluir*$';
     R3Print.Expression := '^\s*//\s*teste:Alterar*$';
     R4Print.Expression := '^\s*//\s*teste:Excluir*$';
     R5Print.Expression := '^\s*//\s*teste:Salvar*$';
     R6Print.Expression := '^\s*//\s*teste:Sair*$';
...
end;
```

Quadro 27 – Identificação de comentários especiais

Após este processo, são identificados os componentes dos arquivos DFM a serem testados através das análises léxica, sintática e semântica. Os componentes dos arquivos DFM foram identificados através das ações semânticas, configuradas na gramática do GALS e tratadas pela ferramenta, guardando os dados necessários, através do método `executeAction` (quadro 28).

```
Procedure TSemantico.executeAction(action : integer; const token : TToken);
var
  i:integer;
begin
  setaction(action);

  case action of
    1: begin
        if not (ChegouNosObjetos) and not (ArquivoPrincipal) then
                InicializaListaObjetos(token.getLexeme);
       end;

    2: GuardaTipoObjeto(token.getLexeme);

    3: propriedade := token.getLexeme;     //VerificaPropriedade(token.getLexeme);

    4: if ((propriedade = 'Caption') or
           (propriedade = 'TabOrder') or
           (propriedade = 'OnClick') or
           (propriedade = 'Enabled')) then
         GuardaValorPropriedade(propriedade, token.getLexeme);

    5: ChegouNosObjetos:=TRUE;

    6: IdentificaFim();

    7: ControleMenu(tempTipo);
  end;
end;
```

Quadro 28 – Implementação das ações semânticas

Quando se identifica o fim do arquivo DFM através da ação semântica 7, os dados são enviados para o método que, através do *template* definido, gera os *scripts* de teste. Neste método é feito o relacionamento entre os elementos disponíveis no *template* e todos os dados coletados. Um trecho deste método é mostrado no quadro 29.

```
procedure TFrmTemplate.gerar(ListaObjetos: TList); ...
begin
   template.Variables.clear;
   template.Variables.Schema.addDataSet('Objeto');

// associa os identificadores definidos no template com os dados extraídos dos
// arquivos DFM DPR, PAS
   template.Variables.Schema.Datasets['Objeto'].addVariable('Projeto');
   template.Variables.Schema.Datasets['Objeto'].addVariable('TipoFrmPrincipal');
   template.Variables.Schema.Datasets['Objeto'].addVariable('CaptionFrmPrincipal');
   template.Variables.Schema.Datasets['Objeto'].addVariable('AcessoFrmPrincipal');
   // outros comandos...

// carrega os dados do formulário principal
   template.Variables.Datasets['Objeto'].addRecord;
   template.Variables.Datasets['Objeto'].Records[0].Variables['Projeto'] :=
                                              // nome do arquivo que contém o projeto
   // outros comandos...

// define o formulário a ser testado
   template.Variables.Datasets['Objeto'].Records[0].Variables['DadosInclusao'] :=
                                                     FrmPrincipal.Objeto.GetDadosI;
   // outros comandos...

// define os componentes de interface
   K:=0; incluidos:=0;
   entrou:= false;
   for i := 0 to ListaObjetos.Count-1 do begin
      FrmPrincipal.Objeto.LocalLinha:= ListaObjetos.Items[i];
      for j := 0 to ListaObjetos.Count-1 do begin
         FrmPrincipal.Objeto.LocalLinha:= ListaObjetos.Items[j];
         if (FrmPrincipal.Objeto.LocalLinha^.PropriTabOrder = k) then begin
            auxiliar:='';
            if (FrmPrincipal.Objeto.LocalLinha^.tipo = 'TEdit') then begin
               auxiliar:= 'Window(*TEdit*,**,' +
                      IntToStr(FrmPrincipal.Objeto.LocalLinha^.ordem) + ').Click;';
               entrou:= true;
            end;
            if (FrmPrincipal.Objeto.LocalLinha^.tipo = 'TDBEdit') then begin
               auxiliar:= 'Window(*TDBEdit*,**,'+
                       IntToStr(FrmPrincipal.Objeto.LocalLinha^.ordem) +').Click;';
               entrou:= true;
            end;
            if (FrmPrincipal.Objeto.LocalLinha^.tipo = 'TRadioButton') then begin
               auxiliar:= 'Window(*TRadioButton*,*'+
                       FrmPrincipal.Objeto.LocalLinha^.PropriCaption + '*).Click;';
               entrou:= true;
            end;
            // demais componentes de interface
         end;
         if (auxiliar <> '') then begin
            template.Variables.Datasets['Objeto'].Records[0].
                                                  Datasets['FrmSelecionado'].addRecord;
         if ((FrmPrincipal.Objeto.LocalLinha^.tipo = 'TEdit') or
             (FrmPrincipal.Objeto.LocalLinha^.tipo = 'TDBEdit')) then begin
template.Variables.Datasets['Objeto'].Records[0].Datasets['FrmSelecionado'].Records
[incluidos].Variables['TipoCampo'] := 'TEdit';
         end;
         // demais componentes de interface
         inc(incluidos);
      end;
      entrou:=false;
      inc(K);
      FrmPrincipal.Objeto.DisposeLocalLinha(FrmPrincipal.Objeto.LocalLinha);
      Break;
   end;
...
end;
```

Quadro 29 – Geração dos testes a partir de um *template*

3.5.3 Operacionalidade da implementação

Nesta seção é apresentado um estudo de caso para demonstrar a funcionalidade da ferramenta. A figura 11 apresenta a tela principal da ferramenta de apoio à geração automática de teste.

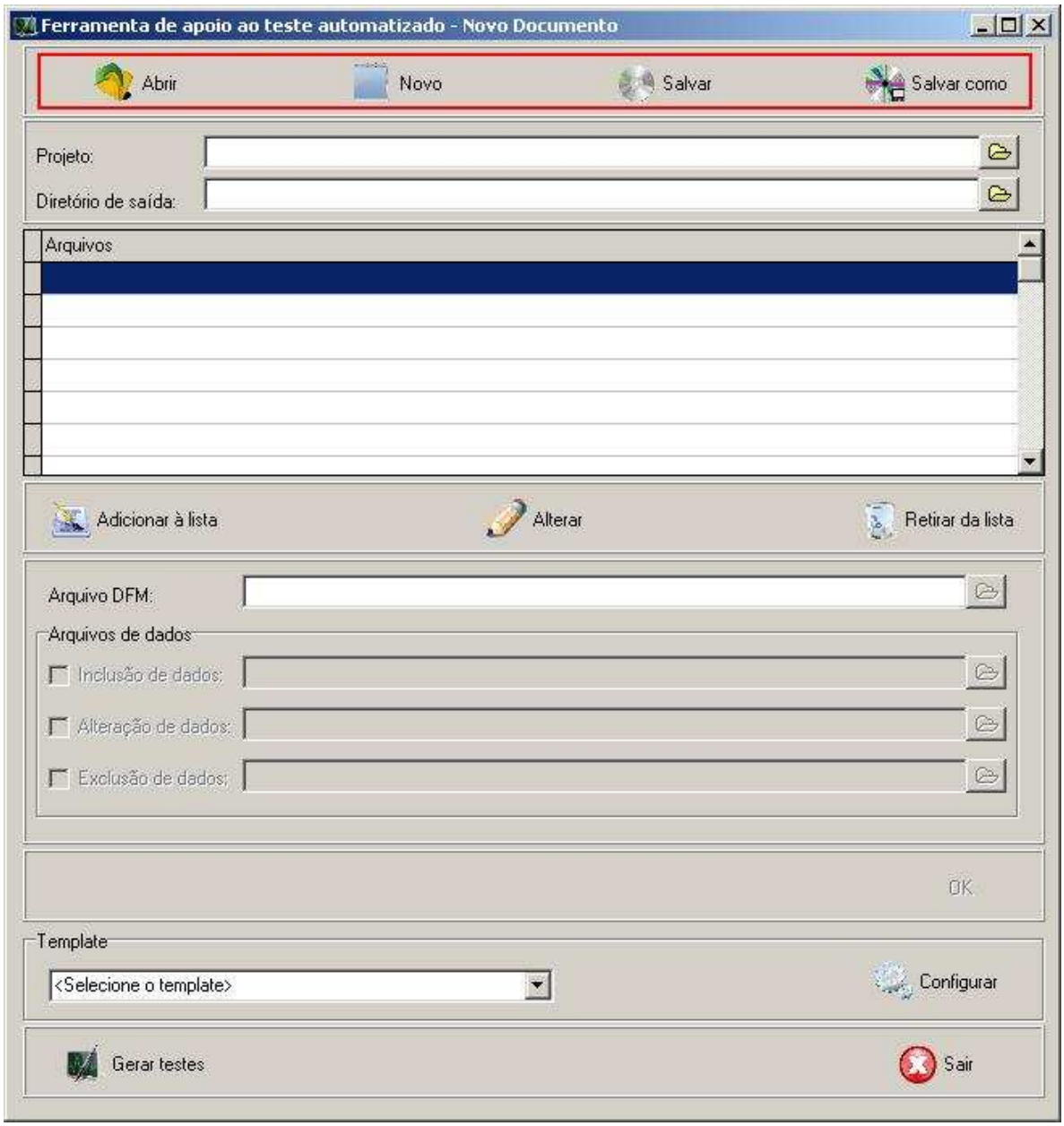

Figura 11 – Interface da ferramenta de apoio à geração automática de testes

Para gerar os *scripts* de teste devem ser informados alguns dados, sendo que alguns são obrigatórios (indicados com *) e outros opcionais:

a) campo `Projeto` (*): deve-se selecionar o arquivo do projeto Delphi (arquivo DPR) para o qual serão gerados os *scripts*;

b) campo `Diretório de saída` (*): deve-se selecionar o diretório no qual serão

gerados os testes;

c) `Arquivos`: contém a relação dos arquivos DFM para os quais serão gerados os testes;

d) botão `Adicionar à lista`: ao pressionar este botão, serão habilitados os campos para informar os arquivos DFM a serem testados e os arquivos de dados para inclusão, alteração ou exclusão. Deve-se:
   - selecionar os arquivos DFM a serem testados (campo `Arquivo DFM` (*)),
   - informar ao menos um arquivo (para inclusão, alteração ou exclusão de dados), sendo que para cada arquivo DFM são informados os arquivos com os dados para inclusão, alteração e exclusão;

e) botão `Alterar`: ao selecionar este botão, serão habilitados os campos para se alterar alguma informação sobre o arquivo DFM previamente selecionado no campo `Arquivos`;

f) botão `Retirar da lista`: ao selecionar este botão, o arquivo DFM previamente selecionado no campo `Arquivos` será retirado da lista, não sendo gerado o teste correspondente;

g) `Template` (*): deve-se selecionar o *template* que será utilizado para gerar os *scripts* de teste, sendo que o *template* pode ser configurado e/ou alterado através do botão `Configurar`;

h) botão `Gerar testes`: ao pressionar esse botão, serão gerados os *scripts* de teste conforme as configurações definidas.

Ao iniciar, a ferramenta encontra-se com os campos em branco. Pode-se criar novas configurações para geração dos testes (botão `Novo`) ou abrir um arquivo previamente configurado (botão `Abrir`). Ao pressionar o botão `Abrir`, a ferramenta disponibiliza os arquivos de configuração existentes, conforme ilustrado na figura 12. Estes arquivos têm a extensão `teste` e, por *default*, encontram-se na pasta `\Testes` do diretório raiz da ferramenta.

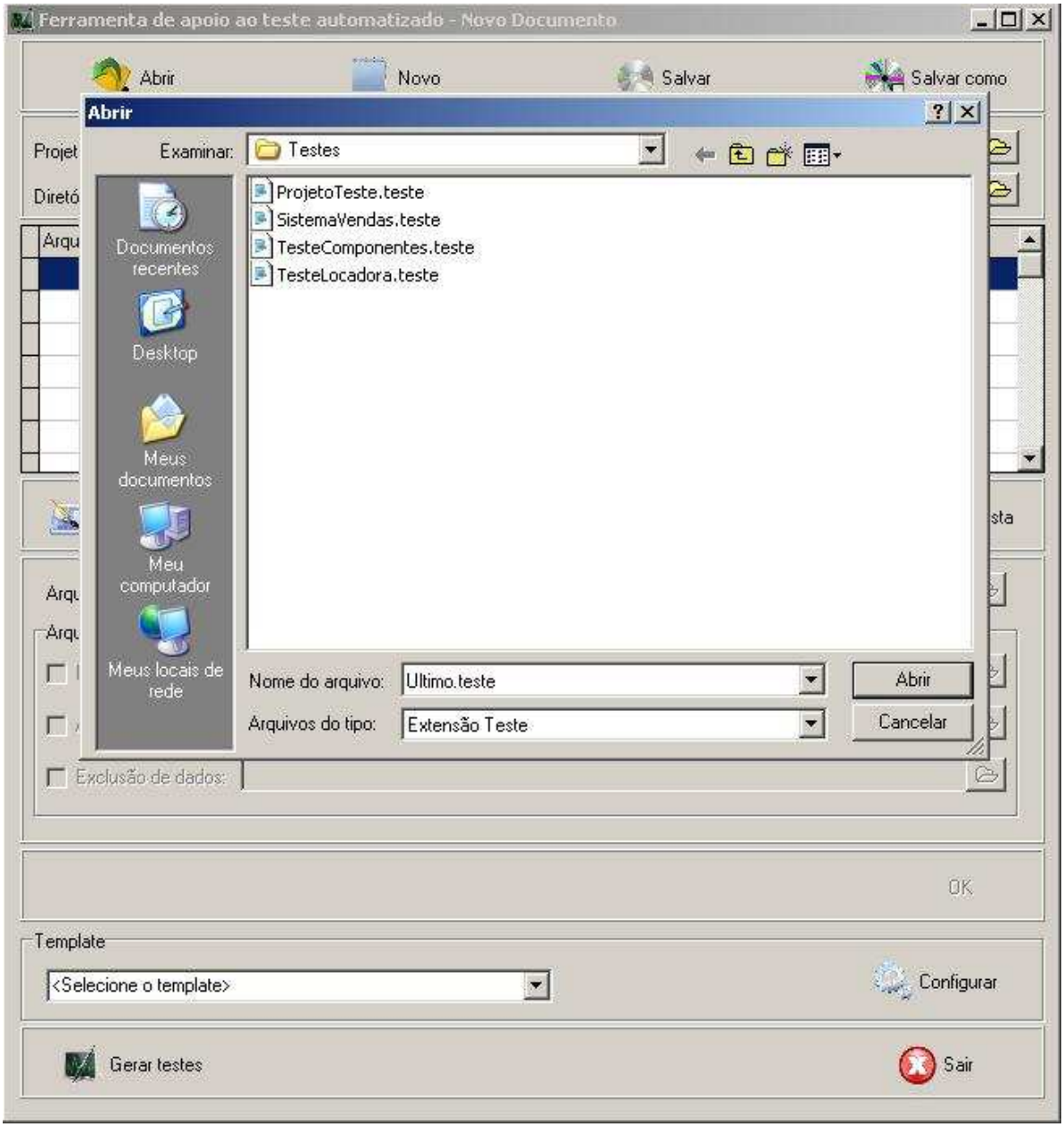

Figura 12 – Selecionando um arquivo de configuração

Para adicionar os arquivos DFM que serão testados, tanto em uma configuração nova quanto em um arquivo previamente configurado, o usuário deve clicar no botão `Adicionar à lista` (figura 13). Os campos serão habilitados e o usuário deve:

a) informar ou selecionar o arquivo DFM no campo `Arquivo DFM`;

b) selecionar um ou mais `Arquivos de dados` (para inclusão, alteração e exclusão de dados);

c) informar os arquivos com os dados (estes são arquivos com a extensão `TXT` que devem possuir o formato apresentado no quadro 30);

d) clicar no botão `OK`, quando as informações estiverem corretamente preenchidas, para gravar as configurações na lista de `Arquivos`.

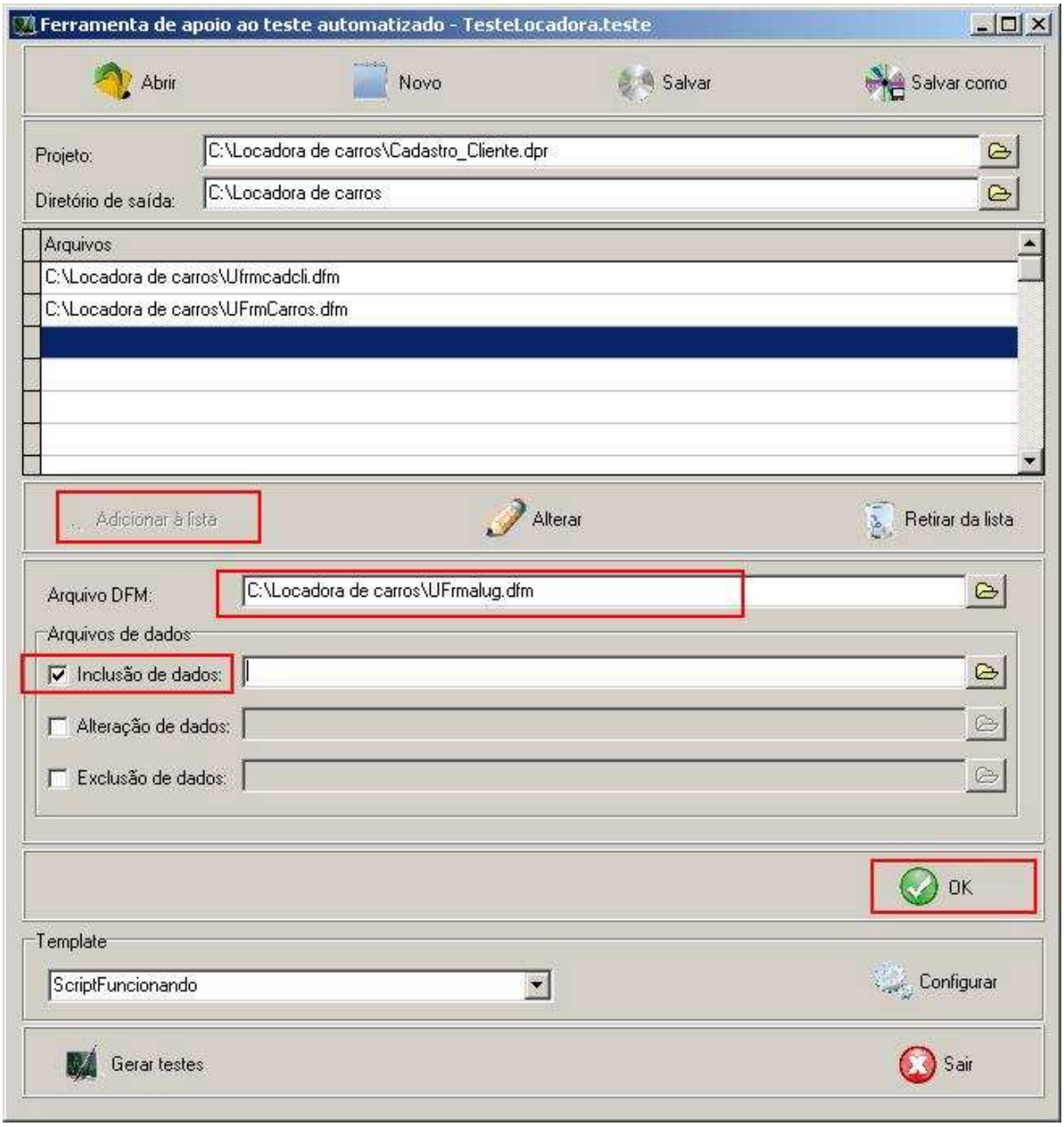

Figura 13 – Adicionando arquivos DFM

```
Adriana Fronza Marcos,Rua Alfonso Souza e Silva,91673213,SC,Blumenau,0,1,1,0,0,1
Joao Pereira,Rua Frederico Klauzer,99651236,SC,Blumenau,0,0,1,0,1,0
Juliano da Silva,Rua Frei Estanislau Schaetter,96452136,SC,Blumenau,1,0,1,0,1,0
Maria Ferreira,Rua das Missões,,SC,Blumenau,0,1,1,0,0,1
```

Quadro 30 – Formato dos arquivos de dados

Os arquivos de dados devem ter os campos separados por um delimitador conforme configurado no *template*, sendo que no quadro 30 estão separados por vírgula. Cada dado representa um campo a ser preenchido no formulário que será testado. Para os campos que recebem ou selecionam texto (`TComboBox`, `TDBLookupComboBox`, `TEdit`, `TDBEdit`), deve ser informado o texto em si. Para os campos que podem ser selecionados (`TCheckBox`, `TDBCheckBox`, `TRadioButton`), deve ser informado 0 para não selecionar e 1 para selecionar o campo.

Na figura 14 é mostrado como os dados da 1ª linha do quadro 30 seriam preenchidos pelo código de teste gerado pela ferramenta. Observa-se que os *scripts* gerados percorrem cada linha de um arquivo de dados para fazer inclusão, alteração ou exclusão de vários registros. No exemplo do quadro 30 serão cadastradas 4 pessoas.

Figura 14 – Executando um *script*

O *script* gera os campos na ordem da propriedade `TabOrder` de cada campo. Portanto, nos arquivos de dados deve-se seguir também esta ordem.

A ferramenta disponibiliza duas opções para selecionar o *template* que será utilizado para a geração dos *scripts*. A primeira opção é selecionar um *template* já existente no campo `Template` (destacado na figura 15), sendo que neste campo são mostrados todos os *templates* que foram salvos na pasta `\Templates` do diretório da ferramenta. Observa-se que é necessário que os *templates* sejam salvos neste diretório para que possam ser selecionados. Os *templates* são arquivos textos que podem ser criados livremente em qualquer editor de textos ou podem ser criados através da ferramenta a partir da opção `Configurar` (figura 15).

Figura 15 – Selecionando e configurando *templates*

Ao selecionar a opção `Configurar`, a ferramenta disponibiliza uma interface para criação, alteração e visualização dos *templates* (figura 16).

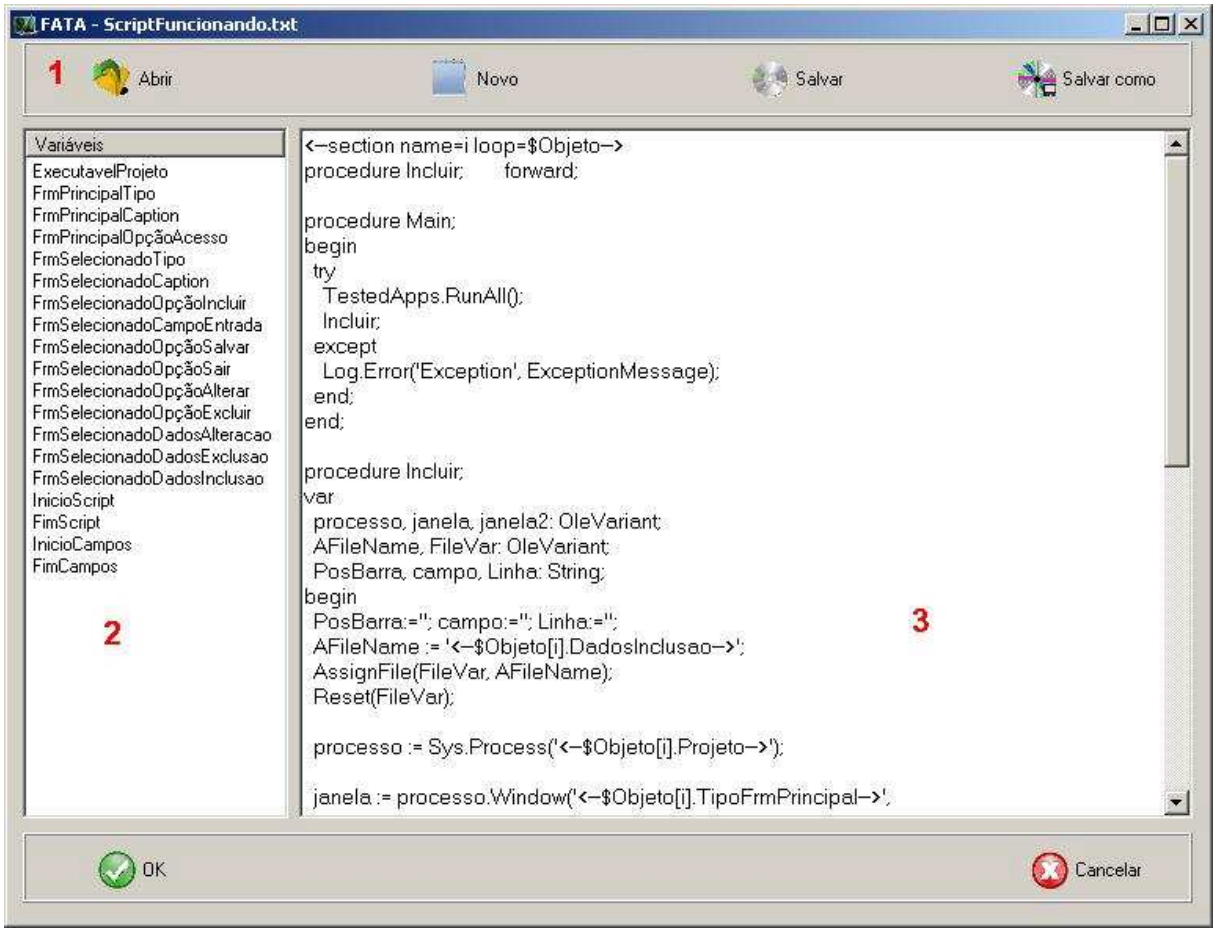

Figura 16 – Configurando *templates*

Na barra de ferramentas (1) encontram-se os botões para abrir um *template* existente, criar um novo *template*, salvar e salvar um *template* com outro nome. À esquerda (2) são mostrados os elementos disponibilizados conforme definição da linguagem de *templates*, que podem ser selecionados e inseridos no *template* que está sendo editado (3). O botão `OK` seleciona este *template* para ser usado na geração dos *scripts* e o botão `Cancelar` cancela a ação tomada.

Após informados todos os dados descritos anteriormente, estes podem ser salvos em um arquivo de configuração, através das opções `Salvar` ou `Salvar Como` (figura 11). Por fim, para gerar os testes basta clicar no botão `Gerar testes`. Se a operação for completada com

sucesso, a ferramenta irá gerar os arquivos de teste no diretório de saída, emitindo a mensagem apresentada na figura 17. Caso ocorram problemas na geração dos testes, a ferramenta informa a mensagem: `Ocorreram erros na geração do arquivo: NomeDoArquivo`.

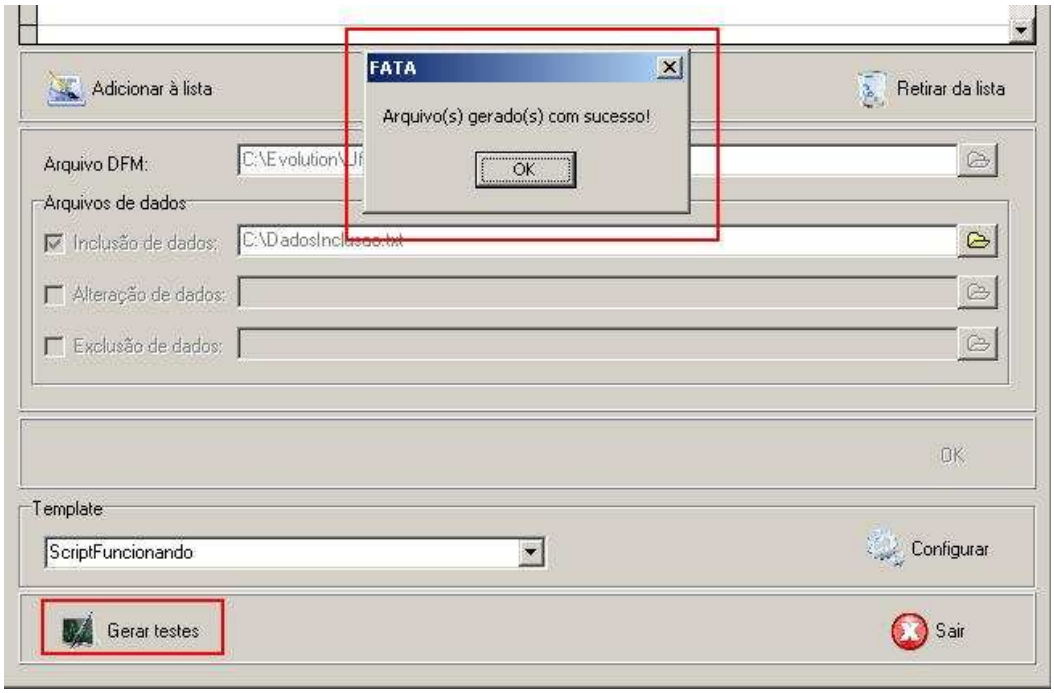

Figura 17 – Gerando *scripts* de testes

Após gerar os *scripts* de teste, basta acessar no diretório de saída o arquivo .MDS gerado (figura 18).

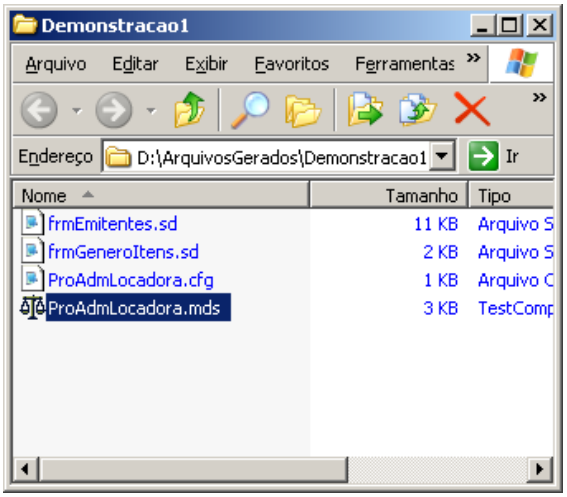

Figura 18 – Arquivos gerados no diretório de saída

Ao abrir o arquivo .MDS, que é o arquivo do projeto que foi criado para o TestComplete, os *scripts* gerados serão mostrados na tela principal do TestComplete conforme ilustrado na figura 19.

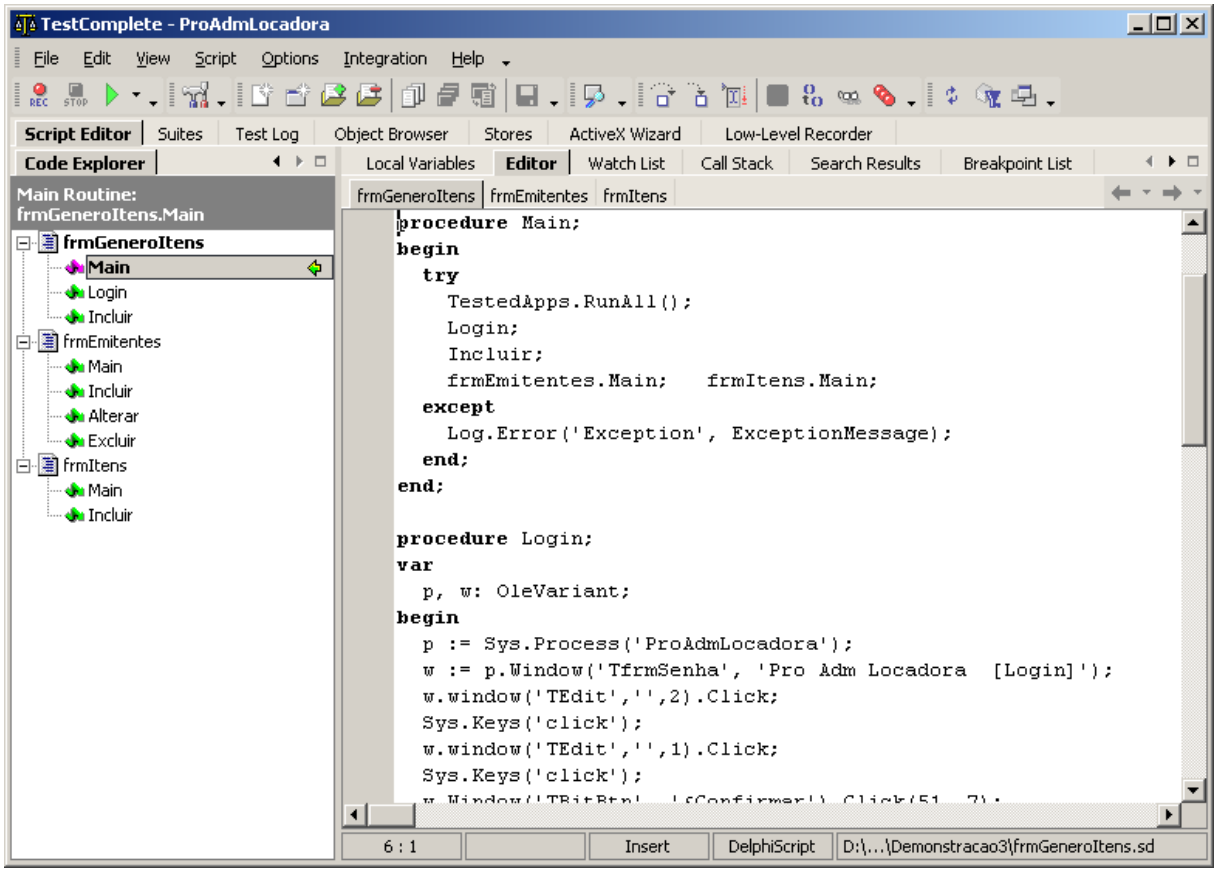

Figura 19 – Carregando os *scripts* de testes na ferramenta TestComplete

Para cada formulário a ser testado é gerada uma *unit* para o TestComplete. O nome destas *units* são os nomes dos formulários que serão testados. Na figura 19, por exmeplo, tem-se as *units* `frmGeneroItens`, `frmEmitentes` e `frmItens`, geradas para três formulários do caso de estudo.

Após abrir o arquivo no TestComplete, basta executar os testes clicando na opção `Run Main Runtime`, clicando no botão ▶ ou pressionando a tecla de atalho F9. Neste momento o TestComplete é minimizado e o usuário pode verificar os testes sendo executados, conforme figura 20 que mostra o *script* executando testes de inclusão de dados no cadastro de emitentes.

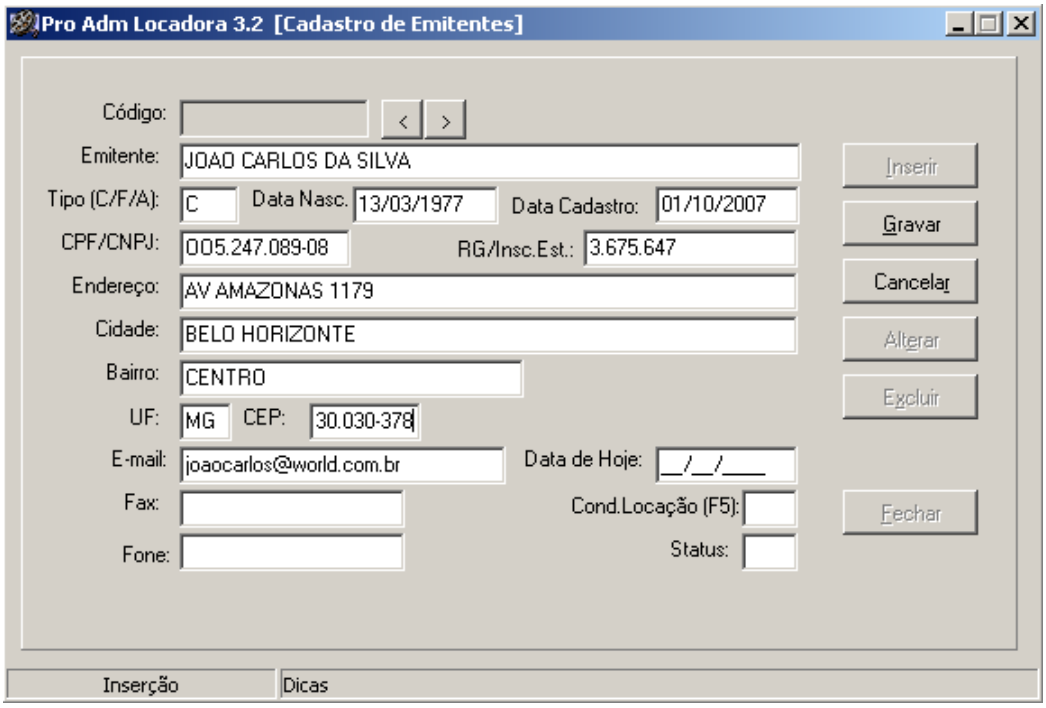

Figura 20 – Executando testes de inclusão de dados

No término da execução dos testes, o TestComplete mostra o resultado dos mesmos através do *Test Log* (figura 22), sendo que os testes podem ser executados com sucesso ou podem ocorrer erros. Neste caso o TestComplete mostra a linha do código e o erro que ocorreu.

Para demonstrar execução de testes que retornam erros, tem-se o seguinte exemplo: ao preencher os dados nos campos o *script*, é selecionado o botão `Gravar` . Neste momento, o software em vez de gravar os dados corretamente emitirá um erro, conforme figura 21.

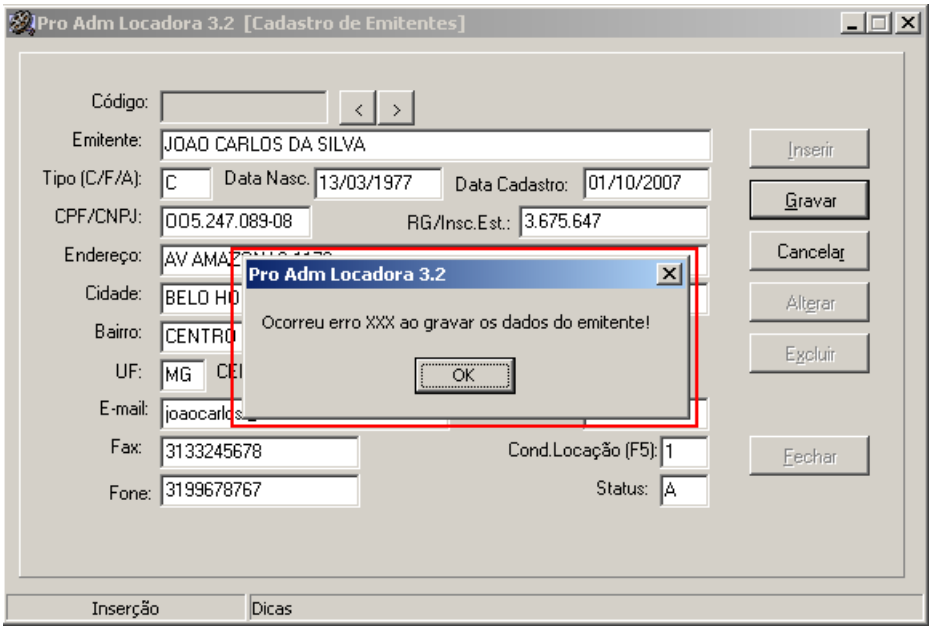

Figura 21 – Executando testes com erro

Observa-se que o *script* não foi executado corretamente até o fim, pois, conforme

demonstrado no quadro 31, após clicar no botão `&Gravar`, o *script* deveria clicar no botão `&Fechar` para sair do formulário.

```
...

begin
...
  begin
    ...
    janela2.Window('TBitBtn','&Gravar').Click;
  end;

  CloseFile(FileVar);
  janela2.Window('TBitBtn','&Fechar').Click;
end;
...
```

Quadro 31 – *Script* de teste para inclusão de dados

Como ele não conseguiu clicar no botão `&Fechar`, pois a mensagem de erro não deixou que o campo fosse acessado, o TestComplete retorna o erro no *Test Log* dizendo que não encontrou o componente `&Fechar`, conforme mostrado na figura 22.

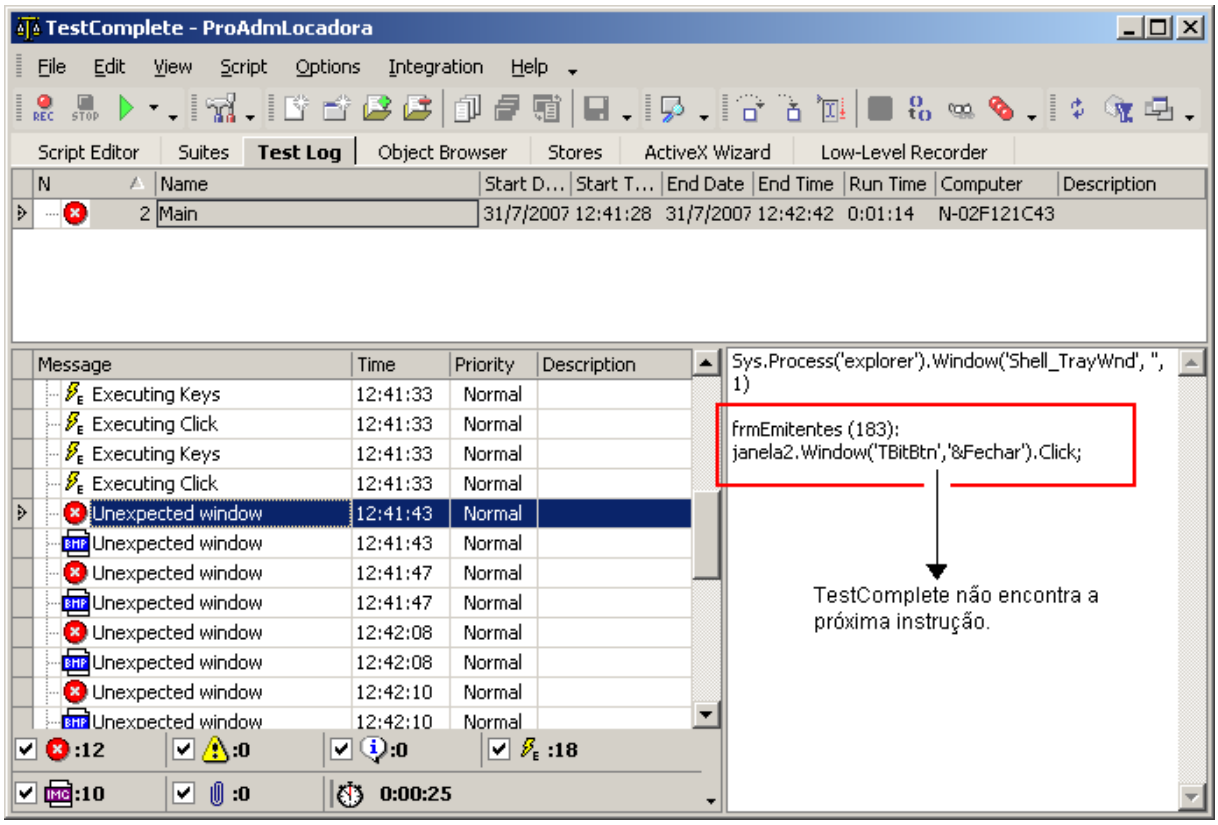

Figura 22 – Retornando erro na execução dos *scripts*

3.6 RESULTADOS E DISCUSSÃO

Com a realização deste trabalho, pode-se demonstrar que novas soluções podem contribuir e agilizar a fase de testes de software, considerando que é uma das fases mais importantes no ciclo de desenvolvimento quando se fala em qualidade. Não se tem dúvida de que a automatização de testes é uma solução que alcança a agilidade necessária, porém também demanda custos, e tempo para a criação e manutenção dos *scripts*. Por este motivo, muitas vezes esta solução não é adotada pelas empresas de software. Através da ferramenta desenvolvida buscou-se diminuir o tempo de implementação destes *scripts*, gerando automaticamente os testes, para que posteriormente sejam executados por ferramentas adequadas, no caso a ferramenta TestComplete.

Nos testes realizados, constatou-se que a ferramenta apóia de fato o profissional responsável por testar o software, sendo que uma vez definido um *template* para gerar os *scritps* de teste, as manutenções tornam-se nulas, pois basta re-gerar o *script* através da ferramenta, ou, se necessário, alterar os *templates*.

4 CONCLUSÕES

A ferramenta de apoio à geração automática de testes é uma ferramenta que gera *scritps* de teste a partir de programas desenvolvidos em Delphi para TestComplete. Tem como objetivo acelerar a criação e atualização dos *scritps*, sendo que cria *scripts* de testes através de informações dos arquivos fontes do sistema a ser testado. Na atualização dos *scripts*, ela acelera ainda mais o processo, sendo que quando o sistema é modificado, os *scripts* devem ser atualizados e com a ajuda da ferramenta basta re-gerar os testes.

Esta ferramenta tem como entrada os arquivos DFM, arquivos que guardam as informações das interfaces das aplicações Delphi. Para leitura e interpretação desses arquivos foram utilizados os analisadores léxico, sintático e semântico implementados na ferramenta DelphiToWeb (SOUZA, 2005), sendo que foram feitas algumas alterações. Além dos arquivos DFM, é utilizado o arquivo DPR (Delphi Project) para identificar o formulário principal, e os arquivos PAS para identificar alguns métodos. Foi usado o conceito de expressão regular, através da *unit* `RegExpr`, para identificação de comentários especiais incluídos nos arquivos PAS.

Para a modelagem dos *scripts* são utilizados *templates* previamente configurados pelo testador, permitindo que os testadores configurem o *template* conforme for mais adequado às rotinas de testes. Os *scripts* serão gerados em DelphiScript, linguagem própria da ferramenta TestComplete e similar à linguagem Object Pascal. Estes *scripts* serão para testes do tipo caixa preta, pois estes são eficientes e oferecem apoio aos testes de regressão.

A ferramenta atingiu todos os objetivos propostos, agregando conhecimentos em testes automatizados, geração de código e analisadores de linguagens. A ferramenta GALS, utilizada para a geração dos analisadores léxico e sintático, facilitou o desenvolvimento do trabalho, demonstrando ser uma boa opção para geração desses analisadores para qualquer linguagem.

4.1 EXTENSÕES

Como extensões para este trabalho sugere-se: implementar a conversão de mais componentes (a ferramenta está limitada a 11 componentes); gerar teste para programas

desenvolvidos em outras linguagens; gerar testes para outras ferramentas de automatização de testes.

REFERÊNCIAS BIBLIOGRÁFICAS

ARNOLD, Thomas R. **Visual Test 6**: bible. Chicago: IDG Books World Wide, 1999.

AUTOMATEDQA CORPORATION. **Getting started with TestComplete 4**. [S.l.], [2006?]. Disponível em: <http://www.automatedqa.com/downloads/Docs/Getting_Started_With_TestComplete.pdf>. Acesso em: 05 set. 2006.

BARTIÉ, Alexandre. **Garantia da qualidade de software**. Rio de Janeiro: Campus, 2002.

BORLAND SOFTWARE CORPORATION. **Borland SilkPerformer Component Test:** solução para o teste de components de software. [S.l.], [2006?]. Disponível em: <http://www.borland.com/br/products/silk/silkperformer_component_test/index.html>. Acesso em: 08 set. 2006.

MOREIRA FILHO, Trayahú R.; RIOS, Emerson. **Projeto & engenharia de software:** teste de software**.** Rio de Janeiro: Alta Books, 2003.

GESSER, Carlos E. **GALS**: gerador de analisadores léxicos e sintáticos. [S.l.], 2003. Disponível em: <http://sourceforge.net/projects/gals>. Acesso em: 17 set. 2006.

HIEBERT, Dennis. **Protótipo de um compilador para a linguagem PL/SQL**. 2003. 52 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://www.bc.furb.br/docs/MO/2002/266472_1_1.pdf> Acesso em: 19 abr. 2007.

HERRINGTON, Jack. **Code generation in action**. Greenwich: Manning, 2003.

HILL, Doug. **FastTrac template engine for Delphi.** [S.l.], 2004. Disponível em: <http://sourceforge.net/projects/ftopf/>. Acesso em: 17 set. 2006.

INTHURN, Cândida. **Qualidade & teste de software**. Florianópolis: VisualBooks, 2001.

PAULA, Devair M. **Avaliação dos testes de software e estudo das características da ferramenta TestComplete**. 2005. Não paginado.Trabalho de Diplomação (Bacharelado em Sistemas de Informação) – Pontifícia Universidade Católica de Minas Gerais, Arcos. Disponível em: <www.arcos.pucminas.br/si/documentos/monografias/SI_PUC_ARCOS_monografia_devair_de_paula.pdf >. Acesso em: 08 set. 2006.

PAULI, Guinther. **Test-driven development with Delphi**. [S.l.], 2005. Disponível em: <http://www.devmedia.com.br/visualizacomponente.aspx?comp=517&site=3>. Acesso em: 17 set. 2006.

PLENTZ, Patrícia. **Técnicas de teste de software**. [Cruz Alta], [2001?]. Disponível em: <http://dinf.unicruz.edu.br/~plentz/Teste.html>. Acesso em: 14 de jun. de 2007.

PRICE, Ana M. A.; TOSCANI, Simão. S. **Implementação de linguagens de programação**: compiladores. 2. ed. Porto Alegre: Sagra Luzzatto, 2001.

RIOS, Emerson. **Análise de riscos em projetos de teste de software**. São Paulo: Alta Books, 2005.

SASSE, Erick. **Convertendo arquivos DFM binários para texto**. [S.l.], [2005?]. Disponível em: <http://www.clubedelphi.net/Novo/Colunistas/Erick_Sasse/02.asp>. Acesso em: 8 set. 2006.

SILVEIRA, Janira. **Extensão da ferramenta Delphi2Java-II para suportar componentes de banco de dados**. 2006. 81 f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SIQUEIRA, Anderson J. **Manual MME & ConDado**. [São José dos Campos], 2005. Disponível em: <http://www.inpe.br/atifs/documentos/manuais/manual_mme_condado.pdf>. Acesso em: 17 set. 2006.

SOUZA, Ariana. **Ferramenta para conversão de formulários Delphi em páginas HTML**. 2005. 69 f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

TESTE de software. In: WIKIPÉDIA, a enciclopédia livre. [S.l.]: Wikimedia Foundation, 2007. Disponível em: <http://pt.wikipedia.org/wiki/Teste_de_software>. Acesso em: 14 jun. 2007.

TOMELIN, Marcio.**Teste de software a partir da ferramenta VisualTest**. 2001. 57 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

APÊNDICE A – Demonstração de testes caixa preta, *template* e testes gerados

Para demonstrar os testes de caixa preta no quadro 32, é mostrado um *template* completo que trata inclusão, alteração e exclusão de dados, onde a inclusão de dados é a parte do teste de caixa preta que trata a entrada de dados via interface. Em seguida no quadro 33 é demonstrado o *script* gerado referente ao *template* do quadro 32, porém somente a parte de inclusão dados tratada pelo comando `<--if $Objeto[i].SeIncluir == 0-->`, como se o testador tivesse selecionado somente a opção para inclusão de dados. No quadro 34 é apresentado o *template* de conferência de dados que no tipo de teste de caixa preta representa a comparação dos dados de saída com dados previamente conhecidos. No exemplo, os dados são os próprios dados que foram incluídos, pois estes não sofreram alteração. Finalizando no quadro 35 é demonstrado o *script* gerado a partir do *template* do quadro 34.

```
<--section name=i loop=$Objeto-->
<--if $Objeto[i].PrimeiroForm == 0-->
<--$Objeto[i].NmOutrosForms-->

procedure Login;        forward;
<--/if-->
//Se usuário selecionar a opção para incluir dados
<--if $Objeto[i].SeIncluir == 0-->procedure Incluir; forward;
<--/if-->
//Se usuário selecionar a opção para alterar dados
<--if $Objeto[i].SeAlterar == 0-->
procedure Alterar;        forward;
<--/if-->
//Se usuário selecionar a opção para excluir dados
<--if $Objeto[i].SeExcluir == 0-->
procedure Excluir;        forward;
<--/if-->

procedure Main;
begin
  try
    TestedApps.RunAll();
<--if $Objeto[i].PrimeiroForm == 0-->
    Login;
<--/if-->
<--if $Objeto[i].SeIncluir == 0-->
    Incluir;
<--/if-->
<--if $Objeto[i].SeAlterar == 0-->
    Alterar;
<--/if-->
<--if $Objeto[i].SeExcluir == 0-->
    Excluir;
<--/if-->
<--if $Objeto[i].PrimeiroForm == 0-->
    <--$Objeto[i].ChamaOutrosForms-->
<--/if-->
  except
    Log.Error('Exception', ExceptionMessage);
  end;
end;

<--if $Objeto[i].PrimeiroForm == 0-->
procedure Login;
var
  p, w: OleVariant;
begin
  p := Sys.Process('ProAdmLocadora');
  w := p.Window('TfrmSenha', 'Pro Adm Locadora  [Login]');
  w.window('TEdit','',2).Click;
  Sys.Keys('click');
  w.window('TEdit','',1).Click;
  Sys.Keys('click');
  w.Window('TBitBtn', '&Confirmar').Click(51, 7);
end;
<--/if-->

<--if $Objeto[i].SeIncluir == 0-->
procedure Incluir;
var
  processo, janela, janela2: OleVariant;
  AFileName, FileVar: OleVariant;
  PosBarra, campo, Linha: String;
begin
  PosBarra:=''; campo:=''; Linha:='';
  AFileName := '<--$Objeto[i].DadosInclusao-->';
  AssignFile(FileVar, AFileName);
  Reset(FileVar);
```

```
  processo := Sys.Process('<--$Objeto[i].Projeto-->');
  janela := processo.Window('<--$Objeto[i].TipoFrmPrincipal-->', '<--
$Objeto[i].CaptionFrmPrincipal-->');
  janela.<--$Objeto[i].AcessoFrmPrincipal-->
  janela2:=processo.Window('<--$Objeto[i].TipoFrmSelecionado-->', '<--
$Objeto[i].CaptionFrmSelecionado-->');

while not Eof(FileVar) do
begin
   janela2.<--$Objeto[i].Incluir-->

   Readln(FileVar, Linha);

<--section name=j loop=$Objeto[i].FrmSelecionado -->
<--if $Objeto[i].FrmSelecionado[j].Enabled == 0-->
<--if $Objeto[i].FrmSelecionado[j].UltimoCampo == 0-->
  //Recupera o campo do arquivo de entrada
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);
<--else-->
  //Recupera o campo do arquivo de entrada
  campo := Copy(Linha,1,Length(Linha));
  Delete(Linha,1,Length(Linha));
<--/if-->
<--if $Objeto[i].FrmSelecionado[j].TipoCampo == TEdit or
$Objeto[i].FrmSelecionado[j].TipoCampo == TDBEdit-->
  janela2.<--$Objeto[i].FrmSelecionado[j].CamposEntrada-->:=campo;
<--/if-->
<--if $Objeto[i].FrmSelecionado[j].TipoCampo == TMaskEdit-->
  janela2.<--$Objeto[i].FrmSelecionado[j].CamposEntrada-->.Click;
  sys.keys('[Home]'+campo);
<--/if-->
<--if $Objeto[i].FrmSelecionado[j].TipoCampo == TCheckBox or
$Objeto[i].FrmSelecionado[j].TipoCampo == TDBCheckBox or
$Objeto[i].FrmSelecionado[j].TipoCampo == TRadioButton-->
  if (campo = 1) then
    janela2.<--$Objeto[i].FrmSelecionado[j].CamposEntrada-->.Click;
<--/if-->
<--if $Objeto[i].FrmSelecionado[j].TipoCampo == TComboBox or
$Objeto[i].FrmSelecionado[j].TipoCampo == TDBLookupComboBox-->
  janela2.<--$Objeto[i].FrmSelecionado[j].CamposEntrada-->(campo);
<--/if-->
<--/if-->
<--/section-->
janela2.<--$Objeto[i].Salvar-->
end;

CloseFile(FileVar);
janela2.<--$Objeto[i].Sair-->
end;
<--/if-->

<--if $Objeto[i].SeAlterar == 0-->
procedure Alterar;
var
  processo, janela, janela2: OleVariant;
  AFileName, FileVar: OleVariant;
  PosBarra, campo, Linha: String;
begin
  PosBarra:=''; campo:=''; Linha:='';
  AFileName := '<--$Objeto[i].DadosAlteracao-->';
  AssignFile(FileVar, AFileName);
  Reset(FileVar);

  processo := Sys.Process('<--$Objeto[i].Projeto-->');
  janela := processo.Window('<--$Objeto[i].TipoFrmPrincipal-->', '<--
$Objeto[i].CaptionFrmPrincipal-->');
  janela.<--$Objeto[i].AcessoFrmPrincipal-->
```

```
  janela2:=processo.Window('<--$Objeto[i].TipoFrmSelecionado-->', '<--
$Objeto[i].CaptionFrmSelecionado-->');

while not Eof(FileVar) do
begin
  Readln(FileVar, Linha);
  janela2.Window('TPanel').Click(211, 40); //Primeiro registro
<--section name=j loop=$Objeto[i].FrmSelecionado -->
<--if $Objeto[i].FrmSelecionado[j].PrimeiroCampo == 0-->
  //Recupera o campo do arquivo de entrada
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);

  while (campo <> janela2.<--$Objeto[i].FrmSelecionado[j].CamposEntrada-->) do
    janela2.Window('TPanel').Click(227, 37); //próximo
  janela2.<--$Objeto[i].Alterar-->
<--/if-->
<--if $Objeto[i].FrmSelecionado[j].Enabled == 0-->
<--if $Objeto[i].FrmSelecionado[j].UltimoCampo == 0-->
  //Recupera o campo do arquivo de entrada
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);
<--else-->
  //Recupera o campo do arquivo de entrada
  campo := Copy(Linha,1,Length(Linha));
  Delete(Linha,1,Length(Linha));
<--/if-->
<--if $Objeto[i].FrmSelecionado[j].TipoCampo == TEdit or
$Objeto[i].FrmSelecionado[j].TipoCampo == TDBEdit-->
  if campo <> '' then
  begin
    janela2.<--$Objeto[i].FrmSelecionado[j].CamposEntrada-->:='';
    janela2.<--$Objeto[i].FrmSelecionado[j].CamposEntrada-->:=campo;
  end;
<--/if-->
<--if $Objeto[i].FrmSelecionado[j].TipoCampo == TMaskEdit-->
  if campo <> '' then
  begin
    janela2.<--$Objeto[i].FrmSelecionado[j].CamposEntrada-->.DblClick;
    sys.keys('[Del]');
    sys.keys(campo);
  end;
<--/if-->
<--if $Objeto[i].FrmSelecionado[j].TipoCampo == TCheckBox or
$Objeto[i].FrmSelecionado[j].TipoCampo == TDBCheckBox or
$Objeto[i].FrmSelecionado[j].TipoCampo == TRadioButton-->
  if (campo = 1) then
    janela2.<--$Objeto[i].FrmSelecionado[j].CamposEntrada-->.Click;
  <--/if-->
  <--if $Objeto[i].FrmSelecionado[j].TipoCampo == TComboBox or
$Objeto[i].FrmSelecionado[j].TipoCampo == TDBLookupComboBox-->
  janela2.<--$Objeto[i].FrmSelecionado[j].CamposEntrada-->(campo);
<--/if-->
<--/if-->
<--/section-->
janela2.<--$Objeto[i].Salvar-->
end;

CloseFile(FileVar);
janela2.<--$Objeto[i].Sair-->
end;
end;
<--/if-->

<--if $Objeto[i].SeExcluir == 0-->
procedure Excluir;
var
```

```
  processo, janela, janela2, janela3: OleVariant;
  AFileName, FileVar: OleVariant;
  campo, Linha: String;
begin
  campo:=''; Linha:='';
  AFileName := '<--$Objeto[i].DadosExclusao-->';
  AssignFile(FileVar, AFileName);
  Reset(FileVar);

  processo := Sys.Process('<--$Objeto[i].Projeto-->');
  janela := processo.Window('<--$Objeto[i].TipoFrmPrincipal-->', '<--
$Objeto[i].CaptionFrmPrincipal-->');
  janela.<--$Objeto[i].AcessoFrmPrincipal-->
  janela2:=processo.Window('<--$Objeto[i].TipoFrmSelecionado-->', '<--
$Objeto[i].CaptionFrmSelecionado-->');

while not Eof(FileVar) do
begin
  Readln(FileVar, Linha);
  janela2.Window('TPanel').Click(211, 40); //Primeiro registro

<--section name=j loop=$Objeto[i].FrmSelecionado -->
<--if $Objeto[i].FrmSelecionado[j].Enabled == 0-->
<--if $Objeto[i].FrmSelecionado[j].PrimeiroCampo == 0-->
  //Recupera o campo do arquivo de entrada
  campo := Copy(Linha,1,Length(linha));
  Delete(Linha,1,Length(linha));
  //Localiza o registro
  while campo <> janela2.<--$Objeto[i].FrmSelecionado[j].CamposEntrada--> do
    janela2.Window('TPanel').Click(227, 37); //próximo

  janela2.<--$Objeto[i].Excluir-->
  janela3 := processo.Window('#32770', 'Confirmação');
  janela3.Window('Button', '&Sim').Click;
<--/if-->
<--/if-->
<--/section-->
end;
janela2.<--$Objeto[i].Salvar-->

CloseFile(FileVar);
janela2.<--$Objeto[i].Sair-->
end;
<--/if-->
<--/section-->
```

Quadro 32 – Exemplo de *template*

```
procedure Login;        forward;
procedure Incluir;      forward;

procedure Main;
begin
  try
    TestedApps.RunAll();
    Login;
    Incluir;
  except
    Log.Error('Exception', ExceptionMessage);
  end;
end;

procedure Login;
var
  p, w: OleVariant;
begin
  p := Sys.Process('ProAdmLocadora');
  w := p.Window('TfrmSenha', 'Pro Adm Locadora  [Login]');
  w.window('TEdit','',2).Click;
  Sys.Keys('click');
  w.window('TEdit','',1).Click;
  Sys.Keys('click');
  w.Window('TBitBtn', '&Confirmar').Click(51, 7);
end;

procedure Incluir;
var
  processo, janela, janela2: OleVariant;
  AFileName, FileVar: OleVariant;
  PosBarra, campo, Linha: String;
begin
  PosBarra:=''; campo:=''; Linha:='';
  AFileName := 'D:\ArquivoDados\InclusaoEmitentes.txt';
  AssignFile(FileVar, AFileName);
  Reset(FileVar);

  processo := Sys.Process('ProAdmLocadora');
  janela := processo.Window('TfrmVirtual', 'Vídeo-Locadora');
  janela.MainMenu.Click('Cadastros|Emitentes (Clientes/Fornecedores)');
  janela2:=processo.Window('TfrmEmitentes', 'Pro Adm Locadora 3.2  [Cadastro de
Emitentes]');

while not Eof(FileVar) do
begin
janela2.Window('TBitBtn','&Inserir').Click;

  Readln(FileVar, Linha);

  //Recupera o campo do arquivo de entrada
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);
  janela2.Window('TEdit','',2).wText:=campo;

  //Recupera o campo do arquivo de entrada
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);
  janela2.Window('TEdit','',3).wText:=campo;

  //Recupera o campo do arquivo de entrada
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);
  janela2.Window('TMaskEdit','',1).Click;
  sys.keys('[Home]'+campo);
```

```
//Recupera o campo do arquivo de entrada
PosBarra:=Pos(',',Linha);
campo := Copy(Linha,1,PosBarra-1);
Delete(Linha,1,PosBarra);
janela2.Window('TMaskEdit','',2).Click;
sys.keys('[Home]'+campo);

//Recupera o campo do arquivo de entrada
PosBarra:=Pos(',',Linha);
campo := Copy(Linha,1,PosBarra-1);
Delete(Linha,1,PosBarra);
janela2.Window('TMaskEdit','',3).Click;
sys.keys('[Home]'+campo);

//Recupera o campo do arquivo de entrada
PosBarra:=Pos(',',Linha);
campo := Copy(Linha,1,PosBarra-1);
Delete(Linha,1,PosBarra);
janela2.Window('TEdit','',4).wText:=campo;

//Recupera o campo do arquivo de entrada
PosBarra:=Pos(',',Linha);
campo := Copy(Linha,1,PosBarra-1);
Delete(Linha,1,PosBarra);
janela2.Window('TEdit','',5).wText:=campo;

//Recupera o campo do arquivo de entrada
PosBarra:=Pos(',',Linha);
campo := Copy(Linha,1,PosBarra-1);
Delete(Linha,1,PosBarra);
janela2.Window('TEdit','',6).wText:=campo;

//Recupera o campo do arquivo de entrada
PosBarra:=Pos(',',Linha);
campo := Copy(Linha,1,PosBarra-1);
Delete(Linha,1,PosBarra);
janela2.Window('TEdit','',7).wText:=campo;

//Recupera o campo do arquivo de entrada
PosBarra:=Pos(',',Linha);
campo := Copy(Linha,1,PosBarra-1);
Delete(Linha,1,PosBarra);
janela2.Window('TEdit','',10).wText:=campo;

//Recupera o campo do arquivo de entrada
PosBarra:=Pos(',',Linha);
campo := Copy(Linha,1,PosBarra-1);
Delete(Linha,1,PosBarra);
janela2.Window('TMaskEdit','',5).Click;
sys.keys('[Home]'+campo);

//Recupera o campo do arquivo de entrada
PosBarra:=Pos(',',Linha);
campo := Copy(Linha,1,PosBarra-1);
Delete(Linha,1,PosBarra);
janela2.Window('TEdit','',11).wText:=campo;

//Recupera o campo do arquivo de entrada
PosBarra:=Pos(',',Linha);
campo := Copy(Linha,1,PosBarra-1);
Delete(Linha,1,PosBarra);
janela2.Window('TMaskEdit','',4).Click;
sys.keys('[Home]'+campo);

//Recupera o campo do arquivo de entrada
PosBarra:=Pos(',',Linha);
campo := Copy(Linha,1,PosBarra-1);
Delete(Linha,1,PosBarra);
janela2.Window('TEdit','',12).wText:=campo;
```

```
  //Recupera o campo do arquivo de entrada
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);
  janela2.Window('TEdit','',9).wText:=campo;

  //Recupera o campo do arquivo de entrada
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);
  janela2.Window('TEdit','',13).wText:=campo;

  //Recupera o campo do arquivo de entrada
  campo := Copy(Linha,1,Length(Linha));
  Delete(Linha,1,Length(Linha));
  janela2.Window('TEdit','',8).wText:=campo;

janela2.Window('TBitBtn','&Gravar').Click;
end;

CloseFile(FileVar);
janela2.Window('TBitBtn','&Fechar').Click;
end;
```

Quadro 33 – *Script* gerado a partir do *template* definido no quadro 32

```
<--section name=i loop=$Objeto-->

procedure Login;        forward;
procedure Conferir;     forward;

procedure Main;
begin
  try
    TestedApps.RunAll();
    Login;
    Conferir;
  except
    Log.Error('Exception', ExceptionMessage);
  end;
end;

procedure Login;
var
  p, w: OleVariant;
begin
  p := Sys.Process('ProAdmLocadora');
  w := p.Window('TfrmSenha', 'Pro Adm Locadora  [Login]');
  w.window('TEdit','',2).wText:='click';
  w.window('TEdit','',1).wText:='click';
  w.Window('TBitBtn', '&Confirmar').Click(51, 7);
end;

procedure Conferir;
var
  processo, janela, janela2, janela3: OleVariant;
  AFileName, FileVar: OleVariant;
  PosBarra, campo, Linha: String;
begin
  PosBarra:='';  campo:=''; Linha:='';
  AFileName := '<--$Objeto[i].DadosInclusao-->';
  AssignFile(FileVar, AFileName);
  Reset(FileVar);

  processo := Sys.Process('ProAdmLocadora');
  janela := processo.Window('TfrmVirtual', 'Vídeo-Locadora');
  janela.MainMenu.Click('Cadastros|Emitentes (Clientes/Fornecedores)');
  janela2:=processo.Window('TfrmEmitentes', 'Pro Adm Locadora 3.2  [Cadastro de
Emitentes]');

janela2.Window('TPanel').Click(211, 40); //Primeiro registro

while not Eof(FileVar) do
begin
  Readln(FileVar, Linha);

<--section name=j loop=$Objeto[i].FrmSelecionado -->
<--if $Objeto[i].FrmSelecionado[j].Enabled == 0-->
<--if $Objeto[i].FrmSelecionado[j].TipoCampo == TEdit-->
<--if $Objeto[i].FrmSelecionado[j].UltimoCampo == 0-->
  //Recupera o campo do arquivo de entrada
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);
<--/if-->
<--if $Objeto[i].FrmSelecionado[j].UltimoCampo == 1-->
  //Recupera o campo do arquivo de entrada
  campo := Copy(Linha,1,Length(Linha));
  Delete(Linha,1,Length(Linha));
<--/if-->
   if campo <> janela2.<--$Objeto[i].FrmSelecionado[j].CamposEntrada--> then
      Log.Message('Campo = '+ campo+' esta diferente do formulário');
<--/if-->
<--/if-->
<--/section-->
```

```
  janela2.Window('TPanel').Click(227, 37); //próximo

end;

CloseFile(FileVar);
janela2.Window('TBitBtn','&Fechar').Click;
end;
<--/section-->
```

Quadro 34 – Exemplo de *template* para conferência de dados

```
procedure Login;        forward;
procedure Conferir;     forward;

procedure Main;
begin
  try
    TestedApps.RunAll();
    Login;
    Conferir;
  except
    Log.Error('Exception', ExceptionMessage);
  end;
end;

procedure Login;
var
  p, w: OleVariant;
begin
  p := Sys.Process('ProAdmLocadora');
  w := p.Window('TfrmSenha', 'Pro Adm Locadora  [Login]');
  w.window('TEdit','',2).wText:='click';
  w.window('TEdit','',1).wText:='click';
  w.Window('TBitBtn', '&Confirmar').Click(51, 7);
end;

procedure Conferir;
var
  processo, janela, janela2, janela3: OleVariant;
  AFileName, FileVar: OleVariant;
  PosBarra, campo, Linha: String;
begin
  PosBarra:='';  campo:=''; Linha:='';
  AFileName := 'D:\ArquivoDados\InclusaoEmitentes.txt';
  AssignFile(FileVar, AFileName);
  Reset(FileVar);

  processo := Sys.Process('ProAdmLocadora');
  janela := processo.Window('TfrmVirtual', 'Vídeo-Locadora');
  janela.MainMenu.Click('Cadastros|Emitentes (Clientes/Fornecedores)');
  janela2:=processo.Window('TfrmEmitentes', 'Pro Adm Locadora 3.2  [Cadastro de
Emitentes]');

janela2.Window('TPanel').Click(211, 40); //Primeiro registro
while not Eof(FileVar) do
begin
  Readln(FileVar, Linha);

  //Recupera o campo do arquivo de entrada
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);
  if campo <> janela2.Window('TEdit','',2).wText then
      Log.Message('Campo = '+ campo+' esta diferente do formulário');

  //Recupera o campo do arquivo de entrada
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);
  if campo <> janela2.Window('TEdit','',3).wText then
      Log.Message('Campo = '+ campo+' esta diferente do formulário');

  //Recupera o campo do arquivo de entrada
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);
  if campo <> janela2.Window('TEdit','',4).wText then
      Log.Message('Campo = '+ campo+' esta diferente do formulário');

  //Recupera o campo do arquivo de entrada
```

```
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);
  if campo <> janela2.Window('TEdit','',5).wText then
      Log.Message('Campo = '+ campo+' esta diferente do formulário');

  //Recupera o campo do arquivo de entrada
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);
  if campo <> janela2.Window('TEdit','',6).wText then
      Log.Message('Campo = '+ campo+' esta diferente do formulário');

  //Recupera o campo do arquivo de entrada
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);
  if campo <> janela2.Window('TEdit','',7).wText then
      Log.Message('Campo = '+ campo+' esta diferente do formulário');

  //Recupera o campo do arquivo de entrada
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);
  if campo <> janela2.Window('TEdit','',10).wText then
      Log.Message('Campo = '+ campo+' esta diferente do formulário');

  //Recupera o campo do arquivo de entrada
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);
  if campo <> janela2.Window('TEdit','',11).wText then
      Log.Message('Campo = '+ campo+' esta diferente do formulário');

  //Recupera o campo do arquivo de entrada
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);
  if campo <> janela2.Window('TEdit','',12).wText then
      Log.Message('Campo = '+ campo+' esta diferente do formulário');

  //Recupera o campo do arquivo de entrada
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);
  if campo <> janela2.Window('TEdit','',9).wText then
      Log.Message('Campo = '+ campo+' esta diferente do formulário');

  //Recupera o campo do arquivo de entrada
  PosBarra:=Pos(',',Linha);
  campo := Copy(Linha,1,PosBarra-1);
  Delete(Linha,1,PosBarra);
  if campo <> janela2.Window('TEdit','',13).wText then
      Log.Message('Campo = '+ campo+' esta diferente do formulário');

  //Recupera o campo do arquivo de entrada
  campo := Copy(Linha,1,Length(Linha));
  Delete(Linha,1,Length(Linha));
  if campo <> janela2.Window('TEdit','',8).wText then
      Log.Message('Campo = '+ campo+' esta diferente do formulário');

  janela2.Window('TPanel').Click(227, 37); //próximo
end;

CloseFile(FileVar);
janela2.Window('TBitBtn','&Fechar').Click;
end;
```

Quadro 35 – *Script* gerado a partir do *template* definido no quadro 34